A copy of the thesis

On the Star Graph Interconnection Topology for Parallel Computing

Dilip K Saikia

May 1996

Indian Institue of Technology, Kharagpur, India

Abstract

The processor interconnection network has a strong influence on the efficiency of processing on a parallel computer. The topology of the interconnection network affects cost of building, overhead due to communication and programmability of the computer.

No single topology fully satisfy the requirements of an ideal interconnection topology. The star graph seems to provide a greater number of properties that are desirable in an ideal topology for an interconnection network. As a result it has attracted a lot of interest among researchers. In this thesis we aim to extend our knowledge of the star graph as an interconnection topology for parallel computing. We investigate the following issues related to efficient parallel processing on the star graph.

1. A scheme for ranking the nodes of the star graph that allows development of efficient algorithms.
2. Efficient communication paradigms suitable for a wide variety of algorithms.
3. Addressing problems arising from the extremely rapid rate of expansion of the star graph.
4. Efficient embedding of some important interconnection topologies.

We address these issues in an integrated fashion based on a common set of ideas. These ideas include a fundamental set of isomorphic transformations on the star graph.

We develop a hierarchical scheme for ranking the nodes and the substars of a star graph. This is used in implementing efficient order preserving communication, several embedding schemes and in efficient schemes for processor allocation on star graph based networks.

Based on the hierarchical ranking scheme and suitable decomposition of the star graph we develop four different communication paradigms suitable for a wide variety of algorithms. These communication paradigms have low communication overheads due to path length and edge-congestion. We use these paradigms to develop order preserving ascend/descend algorithms, pipelined algorithms and algorithms for global communication operations.

We introduce a class of subgraphs of the star graph called the *star cluster*. We show that a star cluster retains most of the desirable properties of the star graph. As a result most of the algorithms developed for the star graph can be easily adapted to a star cluster. The advantage of the star cluster is that it provides a way for star graph based interconnection networks to grow in a near linear fashion. Further, the concept of star cluster is also useful in the development of two efficient schemes for processor allocation.

We develop new techniques for efficient embedding of cycles, tori, full binary trees and pyramids on the star graph based on the isomorphic transformations, the hierarchical ranking scheme and the star cluster concept.

Contents

1 Introduction **2**

1.1 Interconnection network 4

1.1.1 Static interconnection networks 4

1.1.2 Interconnection topologies 7

1.2 The star graph 12

1.3 Overview of the thesis 16

1.3.1 Organization of the thesis 19

2 Survey **21**

2.1 Decomposition of the star graph 21

2.1.1 Substar labeling 22

2.1.2 Ranking of the nodes 22

2.2 Routing 23

2.2.1 Shortest path routing 24

2.2.2 Deadlock-free routing 25

2.3 Fault tolerance 26

2.3.1 Fault diameter 26

2.3.2 Parallel paths 26

2.4 Embedding schemes . . . 27
2.4.1 Cycle embedding . . . 27
2.4.2 Mesh embedding . . . 29
2.4.3 Hypercube embedding . . . 30
2.4.4 Full binary tree embedding . . . 31
2.5 Algorithms . . . 31
2.5.1 Algorithms based on order preserving communication . . . 32
2.5.2 Algorithms based on group-copy paradigm . . . 32
2.5.3 Sorting and selection . . . 33
2.5.4 Computational geometry algorithms . . . 33
2.5.5 Fast Fourier transform . . . 34
2.5.6 Global communication algorithms . . . 34
2.6 Addressing rapid expansion problem . . . 35
2.6.1 Arrangement graph . . . 36
2.6.2 Super star graph . . . 36
2.6.3 Incomplete star . . . 37
2.7 Processor allocation . . . 38

3 Foundation **41**
3.1 Notations . . . 42
3.2 Hierarchical decomposition . . . 42
3.3 Isomorphic transformations . . . 44
3.3.1 Symbol swap transformations . . . 44
3.3.2 Dimension swap transformation . . . 49
3.4 The ranking schemes . . . 52

3.4.1 The hierarchical ranking scheme 52
3.4.2 The pipeline structured ranking scheme 58
3.5 Domain decomposition 60
3.6 Summary 65

4 Communication Paradigms 67

4.1 Hierarchical order preserving communication 68
4.1.1 Confined routing or domain collection and domain distribution HOPC . 69
4.1.2 Third substar or peer 0-centre routed HOPC 71
4.2 Adjacent-domain communication 74
4.2.1 Confined routing or domain collection and domain distribution ADC . . 76
4.2.2 Third substar routed ADC 77
4.2.3 Pipelined order preserving communication (POPC) 78
4.3 Transmit-and-domain-distribute paradigm 80
4.4 Domain-distribute-and-transmit paradigm 81
4.5 The communication processor 83
4.5.1 Communication models 83
4.5.2 Routing information 83
4.6 Summary 84

5 The Star Cluster 85

5.1 Routing in a star cluster 87
5.1.1 Shortest path routing 88
5.2 Topological properties of the star cluster 89
5.2.1 Parallel paths 91

5.3 Adaptation of communication paradigms 95
5.4 Recognition of star clusters 96
5.4.1 Two dimensional arrangement of substars 96
5.4.2 Distance-k set of star clusters 98
5.4.3 Obtaining a distance-2 set of $SC_{n-2,k_{n-1}}$s 100
5.5 Summary 103

6 Embedding Schemes 105

6.1 Embedding of cycles 107
6.1.1 Hamiltonian cycle with a given edge 109
6.1.2 Construction of a cycle in an $SC_{i-1,2}$ 111
6.1.3 Construction of two cycles in an S_i sharing an S_{i-1} 113
6.1.4 Construction of multiple cycles in S_4 118
6.2 Embedding of torus 119
6.2.1 Generation of the first cycle set 119
6.2.2 Generation of additional cycles 120
6.2.3 Arrangement of cycles 121
6.2.4 Expansion 127
6.2.5 Embedding of multidimensional torus 128
6.2.6 Example constructions of 2-dimensional tori 128
6.2.7 Some communication primitives and associated costs 130
6.3 Embedding of FBTs 131
6.3.1 Construction of the root FBT 133
6.3.2 The single-CDS embedding scheme 135
6.3.3 Double-CDS embedding scheme 140

6.4 Embedding of pyramids 149

6.4.1 The structure of a pyramid 149

6.4.2 The basis embedding 150

6.4.3 Embedding of the child pyramids 150

6.4.4 Placement of the apex node 152

6.4.5 Edge-congestion 153

6.4.6 Dilation and expansion 153

6.5 Summary 154

7 Algorithms **157**

7.1 Global Communication Algorithms 159

7.1.1 One-to-all broadcast 160

7.1.2 Multi-path one-to-all broadcast 163

7.1.3 One-to-all scatter 167

7.1.4 All-to-all broadcast 169

7.1.5 Single node accumulation 172

7.1.6 Optimality of the algorithms 173

7.2 Ascend/descend algorithms 174

7.2.1 Fast Fourier Transform 174

7.2.2 Matrix multiplication 178

7.3 Pipelined algorithms 182

7.3.1 Pipelined implementation of n!-point 1-D FFT on S_{n+1} 182

7.3.2 Pipelined implementation of $n! \times n!$-point 2-D FFT on S_{n+1} 184

7.4 Parallel prefix algorithm 185

7.5 Algorithms on star clusters 188

7.5.1 Global communication algorithms 189

7.5.2 Ascend/descend algorithms 190

7.5.3 Pipelined Algorithms 192

7.6 Summary 192

8 Processor Allocation 195

8.1 Star cluster recognition 197

8.1.1 Single-CDS and double-CDS recognition of star clusters 198

8.2 Selection of star clusters 199

8.2.1 The free list 199

8.2.2 Single-CDS allocation 200

8.2.3 Double-CDS allocation 202

8.3 Simulation results 207

8.4 Summary 209

9 Conclusions 211

9.1 Future work 214

A Parallel paths in S_n 215

B Correctness of algorithm D2_Clusters 219

List of Figures

1.1 (a) A completely connected network, (b) A star-connected network. 8

1.2 (a) A linear array network, (b) A ring network. 8

1.3 (a) A two-dimensional mesh network, (b) A two-dimensional torus network. . . 10

1.4 (a) A full binary tree network, (b) A fat-tree network. 11

1.5 (a) A two-dimensional hypercube network, (b) A three-dimensional hypercube network. 12

1.6 The Star Graph. (a) S_2, (b) S_3, (c) S_4. 13

2.1 Jwo's method of cycle construction. 28

3.1 Sibling substars and their labeling. (a) S_2, (b) S_3, (c) S_4. 43

3.2 Effect of i-th symbol swap transformation on inter-substar edge. 48

3.3 Address assignment to the nodes of star graph. The nodes are labeled with two numeric sequences; the upper one is the permutation representation and the lower one is the address assigned to it. (a) S_2, (b) S_3, (c) S_4. 53

3.4 Computation of permutation of a node given the address $a_{n-1}a_{n-2}\ldots a_1$. . . . 57

3.5 Pipeline organization of star graph. (a) S_n, (b) S_4 with address assignment for pipeline structured ranking. 59

3.6 Domain of 2314 in S_3 is {2314,3214,1324}. 62

4.1 HOPC with confined routing. (a) 0-domain distribution scheme, (b) 0-domain collection scheme. 70

4.2 Peer 0-centre routed HOPC. 71

4.3 Edge congestion in peer 0-centre routed HOPC. 73

4.4 Adjacent domain communication. 75

4.5 Congestion in pair-wise adjacent domain communication. 78

4.6 Pipelined order preserving communication (POPC). (a) A single thread of the pipeline, (b) two threads of the pipeline: $(p_0, p_1, p_2, p_3, p_4)$ and $(q_0, q_1, q_2, q_3, q_4)$ 79

4.7 Transmit and domain distribute (TDD) communication paradigm. (a) Data distribution by the nodes in S_3^0 to the nodes in the parent S_4. (b) Data distribution by the nodes in S_i^k to the nodes in the parent S_{i+1}. 81

4.8 Domain distribute and transmit (DDT) communication paradigm. (a) Data distribution by a node in S_3^1 to one node in each of the S_3 in the parent S_4. (b) Data distribution by one node in S_i^k to one node in each of the S_is in the parent S_{i+1}. 82

5.1 Two star clusters $SC_{3,2}$s. 86

5.2 Construction of the parallel paths when the first symbols and the (n+1)-th symbols of the source and destination nodes u and v are identical. 92

5.3 Construction of the parallel paths when the first symbols are identical but the (n+1)-th symbols are different. 92

5.4 Construction of the parallel paths when the first symbols are different but the (n+1)-th symbols are identical. 93

5.5 Construction of the parallel paths when the first symbols and also the (n+1)-th symbols are different. 94

5.6 Sets of eight $SC_{4,3}$s in $SC_{5,5}$. (a) an arbitrary set, (b) a distance-2 set with reference substar *56. The S_4s are represented with fixed symbols. The S_4s included in an $SC_{4,3}$ are connected by arcs. 99

6.1 Generation of an approximate cycle. 108

6.2 Generation of Hamiltonian cycle H' with the edge $(1324, 4321)$ using symbol swap transformation $(t(1) = 3, t(3) = 1)$ of a Hamiltonian cycle H in an S_4. The dotted arcs form the cycle H'. 110

6.3 Construction of a cycle in an $SC_{i,2}$s. 112

6.4 Construction of two disjoint cycles in an S_i. 114

6.5 Construction of two disjoint cycles in an S_5. 117

6.6 Construction of Hamiltonian path in the adjacency graph of cycles: the basis cases for $\rho = 2$. (a) $v = sibling(u)$, (b) $v = peer(u)$, (c) $v \neq sibling(u) \neq peer(u)$. 122

6.7 Construction of Hamiltonian path in the adjacency graph of cycles in an $SC_{m,k}, i \leq m < l$: (a) u and v belongs to two different S_ms, (b) u and v belong to the same S_m. 123

6.8 Construction of Hamiltonian cycle in the adjacency graph of cycles in an $SC_{n-1,k}$: (a) $\Delta n_c > 1$, (b) $\Delta n_c = 1$: (i) $k > 2$, (ii) $k = 2$. 125

6.9 Construction of Hamiltonian cycles for $3 \leq n_c < 10$, when $\rho = 2$. (a) n_c even, (b) n_c odd > 3, (c) $n_c = 3$. 126

6.10 Special construction of three cycles in an S_i. 127

6.11 Decomposition of an FBT into smaller FBTs. 132

6.12 Embedding of T_3 in $SC_{3,3}$. 133

6.13 Construction of FBT of height $log_2\rho - 1$, $\rho = 16$, with dilation one using $\frac{\rho}{2}$ dimensions. First symbols of the leaf nodes are shown within parentheses. . . 134

6.14 Modified construction of FBT of height $log_2\rho - 1$, $\rho = 16$, with dilation one using $\frac{\rho}{2} - 1$ dimensions. (a) for height ≥ 3, (b) for height $=2$. 134

6.15 Single-CDS embedding of FBTs in $SC_{n-1,\rho+1}$, $2^l > n > 2^{l+1}$, $\rho = 2^l$; (a) $p_\rho \neq 1$; (b) $p_\rho = 1$: (i) $\rho > 4$, (ii) $\rho = 4$. 137

6.16 Single-CDS embedding of FBT in $SC_{n-1,n}$, $n = k = 2^l$. 139

6.17 The basis embedding: $P(1)$ in S_3. 150

6.18 Peer adjacency arrangement. (a) An array arrangement of 'x' elements in an S_i. (b) A 2×2 array arrangement of four S_i. 151

7.1 TDD-broadcast tree in S_4 with 1234 as the source node. 161

7.2 Graph of broadcast paths obtained through DDT paradigm in S_4 with 1234 as the source node. 162

7.3 S_4 and its substars with the orthogonal set of CDSs $\{(2,3,4),(3,4,2),(4,2,3)\}$. (a) CDS $(2,3,4)$, (b) CDS $(3,4,2)$, (c) CDS $(4,2,3)$. In each case we show the S_3s in the S_4 and the S_2s in one of the S_3s. 164

166

7.5 Data transfer in iteration-3 of gossiping in an S_4. 171

7.6 24-point FFT flow diagram with processor assignment on S_4. 176

7.7 Prime factored 24-point FFT flow diagram with processor assignment on S_4. . 179

7.8 Data distribution and communication structure for $(2 \times 2) \times (2 \times 3)$ matrix multiplication on S_3. 181

7.9 Data distribution and communication structure for $(4 \times 4) \times (4 \times 6)$ matrix multiplication on S_4. 182

7.10 Data distribution and communication structure for multiplication of matrices of various sizes on S_3. 183

7.11 Pipelined organization of S_ns in an S_{n+1}. 184

7.12 Regrouping in prefix computation; Iteration, $i = 6$. (a) Step-1: each S_6 is a leader, (b) step-2: S_6^0, S_6^2, S_6^4 and S_6^6 are leaders, (c) step-3: S_6^0 and S_6^4 are leaders. 188

7.13 24-point FFT flow diagram with processor allocation on an $SC_{3,2}$. 191

8.1 A set of observations on good star cluster allocations. x : allocated S_i, f : free S_i. 203

8.2 The miss/processor utilization plot for single-CDS and double-CDS allocation schemes for allocations in two different ranges of star clusters in an S_9. Range_1: $SC_{1,1}$ to $SC_{9,1}$, Range_2: $SC_{5,1}$ to $SC_{9,1}$. 208

8.3 The miss/processor utilization plot for single-CDS and double-CDS allocation schemes for allocations in two different ranges of star clusters in an S_9. Range_3: $SC_{5,1}$ to $SC_{8,1}$, Range_4: $SC_{7,1}$ to $SC_{8,1}$. 208

B.1 Partitioning of the array of S_{n-2}s in SC_{n-1,k_n} according to allocation ignoring the step-2(b). In the figure, $x = \rho$. 220

B.2 Allocation pattern in partition D ignoring step-2(b) in the algorithm. 221

B.3 Allocation pattern in partition D considering step-2(b) of the algorithm. . . . 222

List of Tables

1.1 Comparison of the some of the cost parameters (number of edges, degree, diameter) of star graphs and hypercubes of comparable sizes (number of nodes). . . 14

5.1 Two dimensional arrangement of S_is in an S_{i+2}. The S_is are represented with their ranks in the S_{i+2}. 96

5.2 Two dimensional arrangement of S_is in an S_{i+2}. The S_is are denoted by their fixed symbol representation. 98

6.1 Allowed pairings of star clusters and their association with root FBT leaf nodes. s^1: determining symbol of star cluster SC1, s^2: determining symbol of star cluster SC2, l_1: the first symbol required in the leaf node l associated with the pair, link-1: link edges to root of child FBT in SC1, link-2: link edges to root of child FBT in SC2. 145

6.2 Star clusters required for embedding FBTs of various sizes. S-CDS: Single-CDS, D-CDS: Double-CDS. h_{opt} is the theoretical optimal height of an FBT in an S_n. 148

6.3 A comparison of the heights of the full binary trees embedable with the various schemes. *Embeddings with this scheme have node congestion higher than one. 149

7.1 Comparison of communication complexity of the algorithms with the lower bounds. 174

7.2 Processor assignment for 3.3!-point FFT computation on S_3. 177

7.3 Comparison of n!-point FFT on star graph and hypercube. 180

Chapter 1

Introduction

Ever since the first computer became operational there have been a growing demands for more and more computing power. There have been demands for solving problems in less and less time and to solve problems with larger and larger input instances. This has been considerably matched by a tremendous growth in technology. However, there has been a general recognition of the fact that it is not possible to sustain the delivery of speed by centering around a single processor computer. The obvious way to overcome this problem was to build computers that have more than one processor. High speed is attainable by using many processors to solve problems in parallel.

A parallel computer consists of an interconnected set of processors that can cooperate with each other to solve a problem in parallel. It is possible to construct parallel computers with very high raw computing power in a cost effective manner with the high performance microprocessors available off-the-shelf.

For parallel processing, a computation is decomposed into multiple sub-computations to be performed by processes running concurrently on processors of a parallel computer. The processes interact through interprocess communication. The cost of interprocess communication, which is determined by the decomposition and distribution of the sub-computations, is a significant overhead in parallel processing. This overhead is non-existent in its sequential counterpart. For efficient parallel processing it is necessary to keep this overhead due to communication low.

Interprocess communication is achieved in a shared memory parallel computer through the sharing of memory by the communicating processes. Unfortunately, the shared memory has a limited access bandwidth that becomes a bottleneck. The communication cost becomes excessive after the number of processors exceed a limit. The upward scalability of such systems is poor.

The alternative to shared memory systems is to distribute the memory among the processors in the system. In a distributed memory system the processors are interconnected, typically, by a static interconnection network. In such systems, the interprocessor communication is achieved by passing messages through the interconnection network. It is believed that the distributed memory systems provide the path to high scalability in parallel computing.

Normally, the goal in algorithm design for parallel processing is to extract the maximum possible parallelism in the computation. This potentially makes the computation suitable to be performed on the largest possible number of processors, simultaneously, to achieve maximum overlap in computation. However, the interprocess communication becomes more and more intense as higher levels of parallelism is attempted. As a result the communication overhead also becomes more and more significant and parallelism above a limit becomes fruitless. This limit occurs earlier for higher communication costs. To achieve large scale parallelism, it is necessary that the communication cost is kept low. Ideally, the communicating processes should reside in adjacent processors in the network as the cost of communication is higher between non-adjacent processors. However, to satisfy this requirement, in general, it will be necessary to provide complete pairwise communication links between processors. But this would imply an enormous hardware cost. Therefore, in practice, interconnection networks are built with nodes having limited number of communication links.

A practical interconnection network has a characteristic topology. In general, communication between a pair of processors is achieved through a path in the network. The cost of communication is often determined by lengths of such paths. To minimize the communication cost, the processes need to be assigned to processors such that the communicating processes are not far apart. Moreover the communication paths between processes should preferably not overlap with each other to avoid additional communication delay due to contention. In general, for the communication cost to be small, the communication structure of the computation should suit the topology of the interconnection network. A topology with rich interconnectivity suits more easily to computations and allows efficient parallel processing. The role of the topology of the network therefore is critical to the efficiency of parallel processing. Further, topology is also a major factor in determining the physical network construction cost. The number of communication links in the network and the complexity of the communication processors depend on it. As a result, the study of interconnection network topologies has been an important area of research in the field of parallel computing.

1.1 Interconnection network

Interconnection networks can be static or dynamic [Fen81]. Static networks consist of point-to-point communication links among processors. They are also referred to as direct networks and are typically used to construct distributed memory computers. Dynamic networks are built using switches and communication links. Communication links are connected to one another dynamically by the switching elements to establish paths among processors and memory banks. Dynamic networks are also referred to as indirect networks.

1.1.1 Static interconnection networks

Distributed memory systems typically use static interconnection networks to interconnect the processors. The topology of the network determines the cost of message passing in the network to a great extent. The topology is important in development of efficient algorithms. The parameters that influence the choice of a topology can be listed as follows.

1. **Symmetry:** Symmetry is an important criteria used in the choice of the topology. Symmetric networks lend themselves more easily to development of efficient algorithms. A *vertex symmetric* network allows the development of uniform distributed routing algorithms. When the network is *edge symmetric*, subnetworks with different orientations can be allocated to computations, as the communication links can be used interchangeably. A recursive structure is suitable for developing scalable algorithms.

2. **Degree:** The degree of the network determines the number of communication ports a processor has. Therefore the cost of the communication processors in the network increases with the degree. Hence it is desirable to keep the network degree small.

3. **Diameter:** The distance between two processors is defined as the length (in terms of number of links) of the shortest path between them. The diameter of a network is the maximum such distance in the network. Since distance determines the communication cost to a large extent, it is preferable to have a topology with small diameter.

4. **Bisection width and bisection bandwidth:** The bisection width of a network is defined as the minimum number of communication links that have to be removed to partition the networks into two equal halves. The number of bits that can be communicated simultaneously over a communication link between two processors is the channel width. Channel width is equal to the number of physical wires in each communication link. The peak rate at which a single physical wire can deliver bits is called the channel rate. The

peak rate at which data can be communicated between the ends of a communication link is called channel bandwidth. Channel bandwidth is the product of channel rate and channel width. The bisection bandwidth of a network is defined as the maximum possible volume of communication across a cut with the bisection width. It is the product of bisection width and channel bandwidth. A network with a larger bisection width normally has lesser contention for communication resources.

The bisection width of a network provides a lower bound on the space required in packaging the network. If the bisection width of a network is w, the lower bound on the area in a two dimensional packaging is $\theta(w^2)$, and lower bound on the volume in a three dimensional packaging is $\theta(w^{\frac{3}{2}})$.

To avoid congestion in the network it is desirable to have a topology with higher bisection width. But, from the view point of packaging space it is preferable to have the bisection width small.

5. **Connectivity:** *Edge connectivity* is the minimum number of edges that must be removed to disconnect the network. Another measure of connectivity of a network is *node connectivity*. It is the minimum number of nodes that must be removed to disconnect the network. High connectivity reduces contention for communication resources. *Fault tolerance* of a network is one less than the node connectivity and indicates the maximum number of nodes that can be removed and still have the network connected. The connectivity of a network can be at most its degree and its fault tolerance is at most one less than the degree. A network is said to be *optimally fault tolerant* if its fault tolerance is one less than the degree.

Routing mechanisms

Efficient routing of messages is critical in achieving a good performance in a distributed-memory parallel computer system. The routing mechanism determines the path a message takes through the network from the source to the destination. Routing mechanisms can be minimal or nonminimal. A minimal routing mechanism always selects one of the shortest paths between the source and the destination. Minimal routing sometimes leads to congestion in the network. A nonminimal routing scheme, in contrast, may route the message along a longer path to avoid congestion in the network.

A routing mechanism can also be deterministic or adaptive. In deterministic routing, the route used depends only on the source and destination nodes and not on the state of the network. An adaptive routing, on the other hand, uses the state of the network to determine a path for the message. It tries to avoid creation of congestion in the network.

Communication costs

Time spent in communicating messages from one processor to another is a major source of overhead in the execution of a program in a message passing parallel computer. The time taken in the transfer of a message between two processors is called the *communication latency.* Communication latency is the sum of the time required to prepare the message for transmission and the time taken by the message to traverse the path to reach the destination. The components of the communication latency in an interconnection network are as follows.

1. **Start-up time (t_s):** Start-up time is the time required to handle the message at the source processor. This includes the time to prepare the message (adding header, trailer, error-correcting information), the time to execute the routing algorithm, and to establish an interface between the local processor and the router. This start-up time delay occurs only once for a single message transfer.

2. **Per-hop time (t_h):** After a message leaves a processor, it takes finite amount of time to reach the next processor in its path. The time for the header of the message to travel between two directly connected processors is known as the per-hop time. This time is related to the latency within the router for determining which output buffer or channel is to be used to forward the message.

3. **Per-word transfer time (t_w):** If the channel bandwidth is b words per second, then each word takes a time $t_w = \frac{1}{b}$ to traverse the link. This is called the per-word transfer time.

Routing techniques

Routing techniques influence the communication latency in the network to a large extent. Two of the techniques that are commonly used in static interconnection networks are: *store-and-forward routing* and *wormhole routing.*

Store-and-forward routing: In store-and-forward routing, when a message is traversing a path with multiple links, each intermediate processor on the path forwards the message to the next processor after it has received and stored the entire message.

Suppose that a message of length m is being transmitted over a path of ℓ links. At each link, the message incurs a cost t_h for the header and mt_w for the rest of the message to traverse the link. Since there are ℓ such such links, the total time is $(t_h + mt_w)\ell$. Therefore when store-and-forward routing is used, the total communication cost for a message of size m words to traverse a path of ℓ communication links is:

$$t_{comm} = t_s + (mt_w + t_h)\ell \tag{1.1}$$

The store-and-forward routing makes poor utilization of the communication resources. A message is sent from one processor to the next only after the entire message has been received.

Wormhole routing: In wormhole routing, the message is broken into small units called *flow control digits* or *flits*. The flits are pipelined through the network. An intermediate processor does not wait for the entire message to arrive before forwarding it. As soon as a flit is received at an intermediate processor, it is passed onto the next processor. All flits in a message are routed along the same path.

Consider a message that has to traverse a path of ℓ links. The header flit of the message takes ℓt_h time to reach the destination. If the message is m words long, then the entire message will arrive in time mt_w after the arrival of the header. Therefore the total communication time for wormhole routing of the message is:

$$t_{comm} = t_s + \ell t_h + mt_w \tag{1.2}$$

It may be noted that for a communication involving paths of a single link the cost of communication for both the techniques is identical.

1.1.2 Interconnection topologies

Many different topologies have been used as interconnection networks for building parallel computer systems. Many others have been proposed as alternatives. We briefly review some of these topologies below.

1. Completely-connected topology

In a network with the completely-connected topology, each processor has a direct communication link to every other processor in the network. Figure 1.1(a) illustrates a completely-connected network. The completely-connected network is ideal from diameter (= 1) point of view. However, it is not feasible to construct a large network with this topology. In a network of p processors, each processor requires $p-1$ communication ports and the network will have $\binom{p}{2}$ communication links.

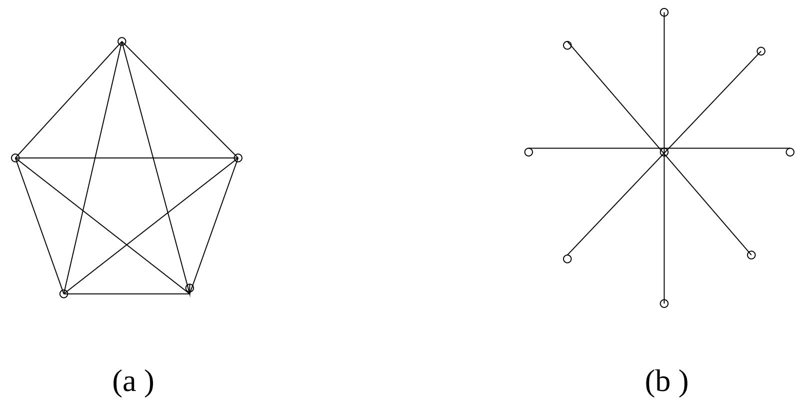


Figure 1.1: (a) A completely connected network, (b) A star-connected network.

2. Star-connected topology

In a network with star-connected topology, one processor acts as the central processor. Every other processor has a single communication link which is connected to this central processor. Figure 1.1(b) illustrates a star-connected network. The diameter of a star-connected network is two. The central processor is the bottleneck in the star-connected network.

3. Linear array topology

In a linear array network each processor, except the processors at the ends, has a direct communication link to two other processors. See Figure 1.2(a). The diameter of a linear array network of p processors is $p - 1$.

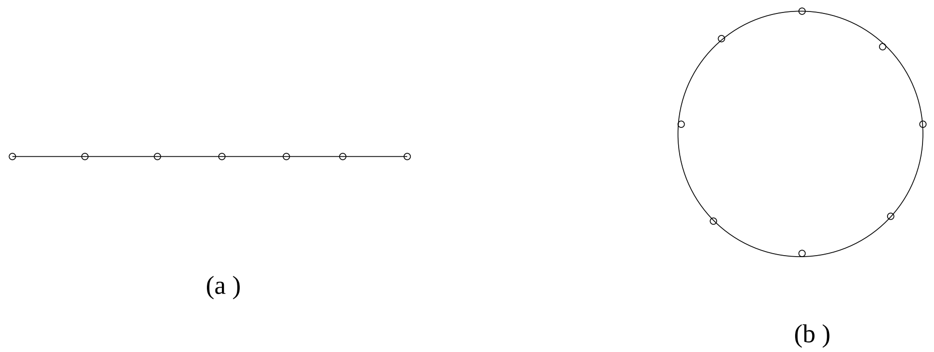


Figure 1.2: (a) A linear array network, (b) A ring network.

4. Ring topology

A linear array with a wraparound connection is referred to as a ring. See Figure 1.2(b).

The diameter of a ring with p processors is $\frac{p}{2} - 1$. The degree and bisection width are two.

5. Mesh topology

An n-dimensional mesh can be considered to be an n-dimensional arrangement of the processors. Each processor in the network, except those at the periphery, has direct communication links to $2n$ processors. Figure 1.3(a) illustrates a two-dimensional mesh network. The minimum diameter of an n-dimensional mesh of p processors is $n(p^{\frac{1}{n}} - 1)$. Many commercial machines have been built on the mesh topology. These include the DAP which uses a two-dimensional mesh, and the J-machine which uses three dimensional mesh. The Tera computer is based on a sparse three dimensional mesh network.

6. Torus topology

A torus consists of a mesh, in which the processors at the periphery are connected by wraparound connections. See Figure 1.3(b). A two-dimensional $nr \times nc$ torus can be considered to consist of nc rings of nr processors each with the processors occupying identical positions in the rings are connected into rings. The diameter of an n-dimensional torus is half of that of an n-dimensional mesh of the same size. The Paragon XP/S is based on a two-dimensional torus and the Cray T3D uses a three-dimensional torus.

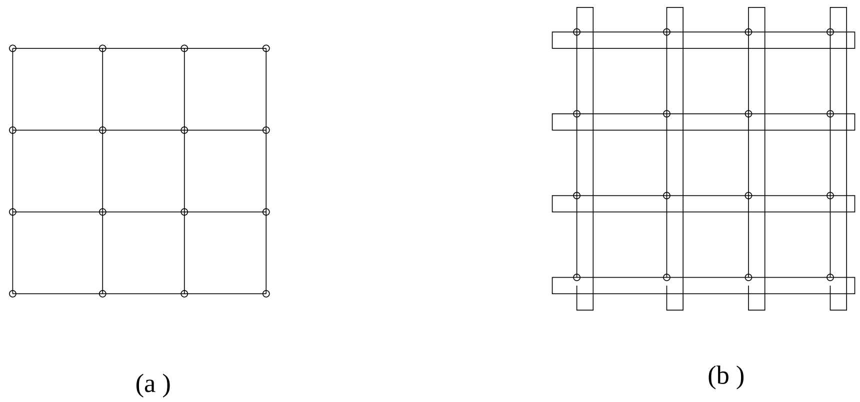


Figure 1.3: (a) A two-dimensional mesh network, (b) A two-dimensional torus network.

7. Tree topology

A tree network is one in which there is only one path between any two processors. Both linear array and star-connected network are special cases of the tree network. Figure 1.4(a) shows a full binary tree. A full binary tree of height h has $2^{h+1} - 1$ processors and diameter $2h$.

To route a message in a tree, the source processor sends the message up the tree until it reaches the processor at the root of the smallest subtree containing both the source and the destination processors. Then the message is sent down the tree towards the destination processor. As a result, the tree networks suffers from a communication bottleneck at higher levels of the tree. This problem can be alleviated by increasing the number of communication links between processors that are closer to the root. Such a tree is called a *fat-tree* [Lei85]. See Figure 1.4(b).

The DADO parallel computer uses a static binary tree network. The CM-5 is based on a dynamic fat-tree network.

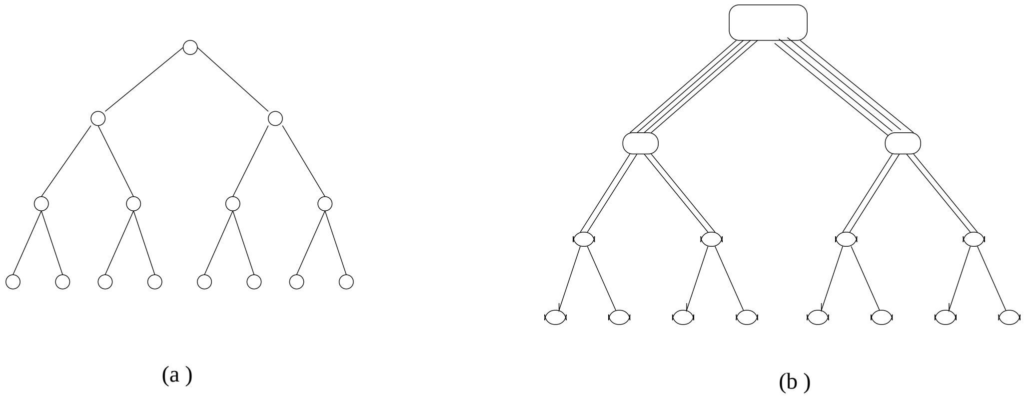


Figure 1.4: (a) A full binary tree network, (b) A fat-tree network.

8. Hypercube topology

A hypercube is a multidimensional mesh of processors, with exactly two processors in each dimension. A d-dimensional hypercube contains $p = 2^d$ processors. It has a highly symmetric and strongly hierarchical structure. A hypercube of dimension $d+1$ is constructed by connecting the corresponding processors of two d-dimensional hypercubes. Both degree and diameter of a d-dimensional hypercube is d. Figure 1.5 illustrates the structures of a two-dimensional, and a three-dimensional hypercube. Some of the parallel computers constructed based on the hypercube network include the Cosmic cube, nCUBE2, iPSC-1, and iPSC-2.

7. k-ary n-cube topology

Hypercube, ring, and square torus are special cases of the general class of networks called k-ary n-cube. An n-dimensional hypercube is a binary n-cube, a p processor ring is a p-ary 1-cube and a two-dimensional square torus with p processors is a $\sqrt{p}$-ary 2-cube. A k-ary n-cube can be constructed from k k-ary (n-1)-cubes by connecting the processors that occupy identical positions in the cubes into a ring.

There are many other topologies such as pyramid [Te80], cube-connected cycles [PV81], de

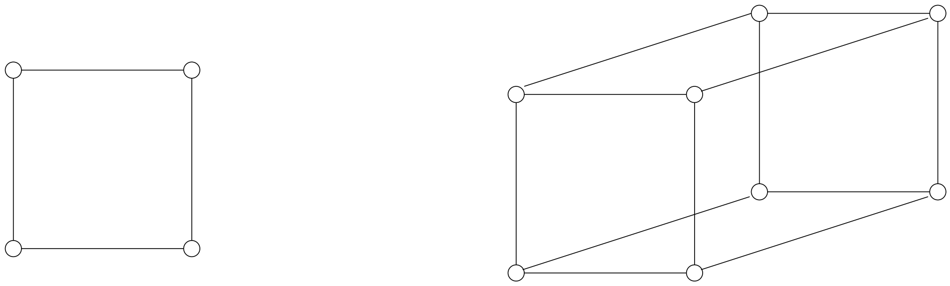


Figure 1.5: (a) A two-dimensional hypercube network, (b) A three-dimensional hypercube network.

Bruijn network [SP89], steirling network [DGD92], fibonacci cube [Hsu93] etc. that have been proposed as interconnection networks. However, none of them fully satisfy all the requirements of an ideal interconnection network topology for large parallel computers. The search for better topologies continues.

Recently the star graph [AK89, AHK87] topology has attracted many researchers because it has many properties desirable for building a large parallel computer.

1.2 The star graph

Notations:

$(i\ j)$: An operator for the transposition of the i-th and the j-th symbols in a permutation.
g_i : The generator for an edge of dimension i. $g_i = (1\ i)$.
ug_i : Equivalent to $g_i(u)$, *i.e.*, the operator g_i applied on u.

DEFINITION 1.1 *A n-star graph, $S_n = (V_n, E_n)$ is a Cayley graph [AK89, LJD93] in which the set of nodes $V_n = \{v | v$ a permutation of $< n >= \{1, 2, ..., n\}\}$. The set of edges $E_n = \{(u, v) | u, v \in V_n$, and $v = ug_i$, for $2 \le i \le n\}$.*

In other words, the set V_n of all permutations of $< n >$ constitute the set of nodes of S_n. Two nodes u and v are connected if and only if the permutation v can be obtained from that of u by transposing the symbol at the i-th position with that at the first (leftmost) position, i.e. $v = ug_i$. The edge is undirected as the transposition is its own inverse and $v = ug_i$ implies $u = vg_i$. The degree of a node is $n - 1$, each element of the generator set accounting for one of the $n-1$ edges incident on a node. Thus, S_n is a regular graph with size $|V_n| = n!$ and degree $n-1$.

The edges incident on a node can be labeled with the index i of the associated generator g_i . There are $n-1$ edges incident on a node of S_n. These will have the labels 2,3,..., and n . These are referred to as the dimensions of the S_n.

The star graph has a hierarchical structure. An S_n, can be decomposed into n S_{n-1}s. This can be done by deleting all the edges of one of the $n-1$ dimensions. This is equivalent to obtaining a particular S_{n-1} from its parent S_n by permuting $n-1$ symbols while one of the n symbols is kept fixed.

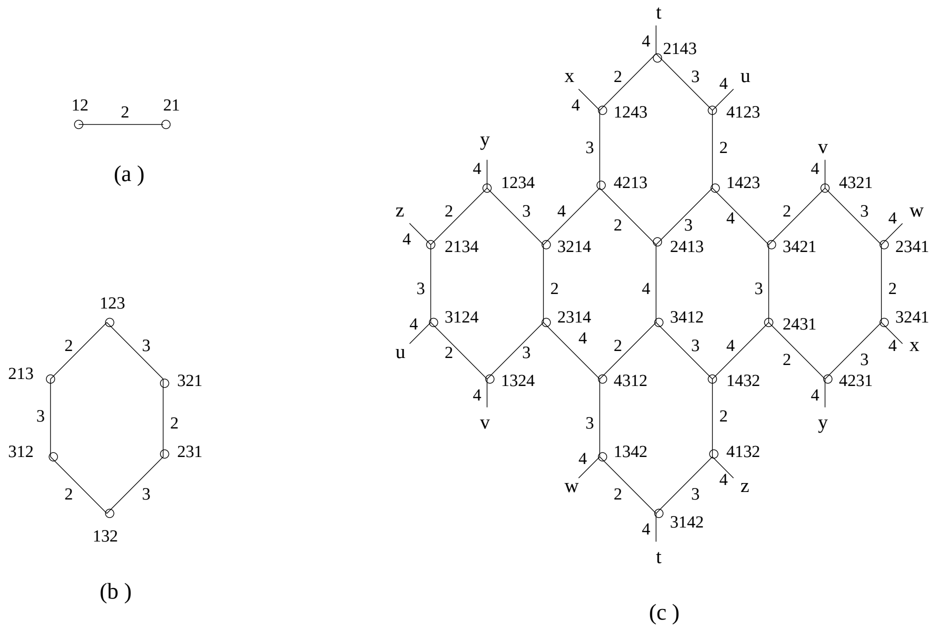


Figure 1.6: The Star Graph. (a) S_2, (b) S_3, (c) S_4.

The star graph has many desirable structural properties. Some of these we list below.

- It is vertex and edge symmetric.
- It is strongly hierarchical.
- Its degree and diameter are sublogarithmic in the number of vertices in the network.
- Its edge connectivity is same as its degree and it is optimally fault tolerant.
- It is bipartite.

Some of the basic parameters of the star graphs are listed below.

Basic parameters of S_n:

Dimension	n
Degree	$n-1$
No. of vertices, N	$n!$
No. of edges	$\frac{n!(n-1)}{2}$
Diameter, D_n	$\lfloor \frac{3}{2}(n-1) \rfloor$
Average Distance, d_a	$n-4+\frac{2}{n}+\sum_{i=1}^{n}\frac{1}{i}$
Bisection width	$\approx \frac{n!}{4}$

One of the best known interconnection network topologies that have been used in building distributed memory parallel computers is the hypercube. Table 1.1 gives a comparison of some of the parameters, that influence the choice of interconnection networks in a parallel system, of the star and the hypercube.

Dimension		Vertices		Edges		Degree		Diameter	
Star n_s	n-cube n_h	Star $n_s!$	n-cube 2^{n_h}	Star	n-cube	Star n_s-1	n-cube n_h	Star D_{n_s}	n-cube n_h
5	7	120	128	240	448	4	7	6	7
7	12	5040	4096	15120	24586	6	12	9	12
9	18	362880	262114	1451520	2359026	8	18	12	18
10	21	3628800	2097152	16329600	21009096	9	21	13	21

Table 1.1: Comparison of the some of the cost parameters (number of edges, degree, diameter) of star graphs and hypercubes of comparable sizes (number of nodes).

It can be seen that for networks of comparable sizes the star graph has much smaller degree and a proportionately smaller number of edges. These reduce the communication hardware cost of a star compared to a hypercube for a large scale parallel system. The smaller diameter can make the communication cost in computations much smaller. The other classes of topologies that find wide use are the constant degree networks such as the mesh. However, the diameter of the constant degree networks are significantly larger. That makes the communication cost in these networks much higher. Considering the large degree of the hypercube and large diameter of the constant degree networks, the star graph is a good alternative interconnection network for large parallel systems.

Significant research has taken place over the past decade in investigating the star graph as a possible interconnection network topology for future parallel computers. Researchers have looked into the problems of designing algorithms for the star graph topology [AQS91, AQ93, MS90, FA94, MS92, RW93a, SLC93, SWC95, FA93b], studying its topological properties [LJD93, DT93a, SS93b, JLD90, QMA93, QA91, HB94], embedding of other interconnection structures on star graph and also the embedding of the star graph in other interconnection structures [NSK90, JLD91, RWY93, MPS94, SHC+93, BHOS94, HOS95, SV94], modifying the star graph to provide a more cost effective solution to building parallel computers [DT92, LB94, Lat93a, SS93a, LY91, RM94, DT93b], studying its fault-tolerance properties [AK87, Lat93b, JM94, RS93, SS91, FA93a], developing routing algorithms [AHK87, DT93a, MJ94b, BNL93, DMS91, AQ93, LBT94, RG94] and processor allocation on star graph based networks [Lat94].

With the growing demand for higher and higher levels of real parallelism there is a rising need for building cost effective large scale parallel computers. The star graph seems to provide a suitable topology for building such machines. However, the star graph is more recent and its broad capabilities are not yet fully understood. *There seems to be a number of general issues that need to be further investigated before the star graph can be used as an interconnection network topology.*

1. The permutation labels of the nodes in a star graph, as such, do not define an ordering of the nodes in the network. However in many computations it is often required to use an ordering of the nodes. That is, the nodes are required to be ranked. It is necessary to investigate for an elegant way to rank the nodes of a star graph which preserves its inherent recursive structure and also allows *order preserving communications* in constant time as in a hypercube. Such a ranking scheme will give the star graph an equivalence to the hypercube and will make it more adaptive to algorithm development, simulation of other interconnection structures, processor allocation and so on.

2. Ability to support efficient communication paradigms indicates the richness of an interconnection structure and allows easy development of algorithms on the structure. It is necessary to investigate for a suitable set of communication paradigms on the star graph with low communication overheads so that efficient algorithms for a wide variety of problems can be developed based on these.

3. The star graph grows very rapidly with dimension. The size (number of nodes) of the star graph of dimension $n+1$ is $n+1$ times that of a star of dimension n. Due to this, it is difficult to utilize the star graph efficiently. Further, its extensibility is poor. Therefore, it is necessary to devise a scheme for efficient utilization of star graph based

networks. Further, it is necessary to find a way to expand star graph based networks at a near linear fashion.

4. The ability to embed other interconnection structures on the star graph enables implementation of algorithms developed for those structures on the star. There have been several results on such embeddings. While for some of these embeddings are not very efficient, for several other important structures embedding techniques are unknown. It is necessary to develop techniques for efficient embedding of these interconnection structures on the star graph.

The primary objective of the research reported in this thesis is to look into the issues raised above. The results obtained would hopefully throw more light on the suitability of the star graph as an interconnection topology for future parallel computers.

1.3 Overview of the thesis

In this thesis we address the above issues of the star graph as an interconnection topology for parallel computers. We deal with these issues in an integrated fashion with the focus on efficient parallel computing. We adopt two different decomposition schemes for our various studies. The first is the hierarchical decomposition in which a star is decomposed in to a set of disjoint substars called *sibling substars*. A sequence of dimensions of the star graph, called a *composition dimension sequence*, defines a decomposition (or conversely, a composition) of substars with a given orientation. This decomposition helps in defining ranking schemes, developing embedding schemes and algorithms, and also in developing processor allocation schemes for efficient utilization of star graph based networks. The other one is a domain decomposition which is based on the concepts of *(i,k)-centre* and the *domain of a node*. This decomposition gives a good understanding of the communication structure of the star graph and is suitable for defining and analyzing several communication paradigms on the star graph.

Two classes of isomorphic transformations on the star graph called *symbol swap transformation* and *dimension swap transformation*, and a special class of symbol swap transformation called the *i-th symbol swap transformation* form the basis for most of the work presented in this thesis. The i-th symbol swap transformation is a minimal bijective mapping between the nodes of two sibling S_{i-1}s.

Based on the i-th symbol swap transformation we develop a scheme for hierarchical ranking of the nodes and the substars of a star graph. This scheme allows order preserving communications on the star graph in constant time and gives a mixed radix generalized cube [BA84] structured view to the star graph. The hierarchical ranking scheme also leads to another isomorphic transformation called *peer node imaging.* We also define another scheme for ranking the nodes of the star graph, based on the i-th symbol swap transformation, which gives a pipelined view to the star graph and provides a convenient framework for developing pipelined algorithms for the star graph.

We develop four different communication paradigms for the star graph, namely, i) hierarchical order preserving communication (HOPC), ii) adjacent domain communication (ADC), iii) transmit-and-domain-distribute (TDD) and iv) domain-distribute-and-transmit (DDT). The HOPC paradigm is based on the hierarchical ranking scheme and the hierarchical decomposition of the star graph. A HOPC consists of communication between identically ranked nodes in sibling substars. There are two alternative routing schemes for HOPC. The edge congestion with one of the schemes, in general, is two. However, many operations can be performed without edge congestion. This communication paradigm is relevant to the order preserving ascend/descend class [PV81, AQ93] of algorithms which include important algorithms such as fast fourier transformation, bitonic sort etc. There are also several fundamental operations such as translation, concentration and distribution, which are required in algorithms in the areas such as computational geometry [AQS91], can be efficiently implemented with HOPC. It is also relevant to several embeddings schemes on the star graph.

The ADC, TDD and DDT paradigms are based on the domain decomposition of the star graph. The ADC paradigm consists of communication between i-th symbol swap image nodes in the domains of two adjacent nodes in sibling substars. Special cases of the ADC paradigm are: i) the *group-copy* paradigm defined by Akl et al. [AQ93] and ii) pipelined order preserving communication (POPC), a communication between identically ranked nodes in the pipeline structured ranking scheme. Like HOPC, there are two alternative routing schemes in the ADC paradigm too. The edge congestion in one of them is two, while in the other it is $O(n)$ on ans S_n. The ADC paradigm is relevant to non-order preserving ascend/descend algorithms and pipelined algorithms on the star. It is also relevant to dilation three embeddings of grids on the star graph.

The TDD and DDT paradigms are suitable for developing global communication algorithms such as broadcasting, gossiping, scattering and accumulation on the star graph.

One of the main obstacles in efficient parallel processing on a star graph based interconnection network is its extremely rapid rate of expansion. We propose the concept of *star cluster* to address this issue. A star cluster, $SC_{i,k}$ is a subgraph of an S_{i+1} induced by k sibling S_is. We analyze some of its topological aspects and show that it preserves most of the desirable properties of the star graph. As a result, the algorithms and embedding schemes developed for the star graph can be implemented on the star clusters with minor or no degradation.

Based on the hierarchical ranking of the substars and the hierarchical decomposition scheme, we devise a two dimensional arrangement of substars which leads to simple schemes for recognizing star clusters in star graph based networks. Based on this, we develop efficient schemes for dynamic processor allocation, in terms of star clusters, on star graph based networks for supporting multi-tasking parallel computing environment. Processor allocation in terms of star clusters leads to efficient utilization of processors in a star graph based interconnection network. Simulation studies have been carried out on these schemes and found to have good performance characteristics.

Based on the symbol swap transformation and the hierarchical ranking scheme we develop embedding schemes for several important interconnection structures. These include schemes for embedding cycles in a minimal star cluster, embedding torus of any dimension and size, and embedding pyramid. The embedding scheme for two dimensional torus is based on obtaining a required set of parallel cycles and arranging them suitably to form the torus. For multidimensional torus, the scheme consists of obtaining a suitable set of lower dimensional tori through i-th symbol swap transformation or peer node imaging and arranging them appropriately. We have also developed two schemes for embedding full binary trees with dilation two. The schemes are recursive in terms of embeddings in star clusters. The two dimensional arrangement of substars is used to identify star clusters with desired characteristics for the embeddings. With one scheme the embeddings are free of both edge and node congestion. With the second scheme the embeddings are free of edge congestion and the height is at most two less than the theoretical optimum. All the embedding schemes are applicable to star clusters.

We develop several algorithms based on the communication paradigms introduced. Based on the transmit-and-domain-distribute and the domain-distribute-and-transmit communication paradigms we develop simple asymptotically optimal global communication algorithms for broadcasting, gossiping, and scattering on star graphs. Based on a concept called *orthogonal set of composition dimension sequence* we develop a method of constructing $n-1$ mutually contention-free spanning trees in an S_n with a given root node. These spanning trees are used in a multipath broadcasting algorithm suitable for large sized messages. Based on the hierarchical order preserving communication paradigm and the hierarchical ranking scheme we develop order preserving ascend/descend type of algorithms for FFT and matrix multiplication. The FFT algorithm has a lower communication cost and thus higher scalability compared to an earlier algorithm [FA94]. Based on the pipeline structured ranking scheme and the adjacent domain communication paradigm we have developed pipelined algorithms for 1-D and 2-D FFT. We also develop a congestion-free optimal algorithm for prefix computation on the star graph based on the transmit-and-domain-distribute communication paradigm. All the algorithms developed here can be implemented on star clusters with minor or no modifications.

1.3.1 Organization of the thesis

Significant research has been carried out on the star graph. In Chapter 2 we briefly review these works so that the background to the present work can be understood.

In Chapter 3 we lay down the theoretical foundation of the work reported in this thesis. We first discuss the hierarchical decomposition of the star graph. This is followed by the presentation of the isomorphic transformations. We present the two ranking schemes– i) *hierarchical ranking scheme* and ii) *pipeline structured ranking scheme*. We also introduce the domain decomposition of the star graph based on the concept of the *domain of a node*.

In Chapter 4 we develop the communication paradigms. We present alternative routing schemes for these communication paradigms and characterize them with respect to path length, edge and node congestion, and path distribution. We also look into the requirements of the communication processor of a node with respect to communication model and routing information, to support these paradigms in a star graph based interconnection network.

In Chapter 5, we present the concept of *star cluster*. We study some of its topological properties to show that the star cluster preserves most of the desirable characteristics of the star graph. We develop a shortest path routing scheme for the star cluster and also look into the adaptability of the communication paradigms on the star cluster. We present a *two dimensional arrangement of substars* to enable efficient recognition of star clusters. We also introduce a concept called *distance-k set of star clusters* and present algorithms for obtaining a distance-2 set of star clusters.

In Chapter 6 we present the embedding schemes. These include embedding schemes for cycles, multidimensional tori, full binary trees and pyramids.

In Chapter 7 we present the various algorithms developed. We first present the set global communication algorithms. These are followed by the order preserving ascend/descend algorithms, pipelined algorithms and the prefix computation algorithm.

In Chapter 8 we present two efficient heuristics for processor allocation in terms of star clusters. These two are compared by simulation studies.

Finally, Chapter 9 summaries the current work and sketches future directions for investigation.

Chapter 2

Study of the star graph: A brief Survey

The many interesting properties of the star graph topology have attracted many researcher to study the topology for understanding its potential as an interconnection network for future multicomputers. The various aspects of the graph that have been studied can be classified into the following categories: (a) structural and topological characteristics, (b) the question of embedding existing interconnection structures, (c) algorithm development, (d) alleviation of its shortcomings due to the very rapid rate of expansion, (e) processor allocation and (f) other miscellaneous issues. In this chapter we briefly review these works.

2.1 Decomposition of the star graph

In an S_n if the symbol at the n-th position is fixed we get $(n-1)!$ permutations. The fixed symbol at the n-th position can be any one of the n symbols. For each of these n symbols we get a distinct set of $(n-1)!$ permutations. Each of these n groups of $(n-1)!$ permutations represent a subgraph of S_n and is isomorphic to an S_{n-1}. The S_{n-1}s so obtained are connected by edges of dimension n. Therefore fixing the n-th symbol represents a way to decompose an S_n into S_{n-1}s by deleting the dimension n edges. The S_{n-1}s are called the substars of the S_n. Similar decomposition of an S_n into S_{n-1} substars can also be achieved by fixing a symbol at any positions i, $2 \leq i \leq n$, which is equivalent to partitioning of the S_n by deleting the edges of dimension i.

The above decomposition of the star graph can be performed hierarchically. For example, we get n S_{n-1}s within an S_n by fixing the n symbol. Within these S_{n-1}s, if we fix the $(n-1)$-th symbol we will get $n-1$ S_{n-2}s in each. Since the star graph is strongly hierarchical, any of the dimensions of the graph can be chosen for the decomposition.

Berthome et al. [BFP92] uses an interesting decomposition of the star graph based on the first symbols of the nodes. In this, an S_n is decomposed into n partitions, $\Delta_i,\ i = 1, 2, \ldots, n$. The set of nodes with the first symbol $i,\ 1 \leq i \leq n$, constitutes the partition Δ_i. It is interesting to note that the nodes in such a partition are not connected. However, the nodes in each of the partitions from a perfect matching with the nodes in one of the S_{n-1}s in the S_n.

2.1.1 Substar labeling

In general, by fixing symbols in any $n-i$ positions or in other words by removing all the edges of any $n-i$ dimensions we can get a complete set of $\frac{n!}{i!}$ distinct S_is in the S_n. It is possible to identify these substars in different ways. One way to denote a substar is by its fixed symbols. For example, in the S_is obtained by fixing the symbols from $(i+1)$-th to n-th positions, the sequence of these $n-i$ fixed symbols is distinct for each S_i. Therefore, the sequence of fixed symbols can be used for denoting the S_{i-1}. Such a notation for the substars we shall refer to as the *fixed symbol representation* of the substars. In this representation to indicate the fixed symbols and their the positions of the permuting symbols are usually filled with the '$*$' symbol [AQS91, AQ93]. For example, $***3$ represents an S_3 with the fixed symbol '3' at position 4. Similarly, $**25*$ represents an S_3 in S_5 with the fixed symbols '2' and '5' at positions 3 and 4 respectively. An S_{n-1} in an S_n can be represented by a single fixed symbol. Lakshmivarahan et al. [JLD91] uses the notation $G[i,k]$ to represent the S_{n-1} with the fixed symbol k at position i. Latifi [Lat93a] uses the notation $S_{n-1}(k)$ to denote the S_{n-1} with the n-th fixed symbol k. The fixed symbol representations of substars have also been used by several other authors [AK89, AQS91]. However, at times this representation becomes cumbersome. It would be very useful to represent a substar with a unique integer.

We shall use the notation S_{n-1}^j to represent a (n-1)-substar of S_n obtained by deleting the edges of dimension n. The index j of the substar is given by $j = n - s_n^j$, where s_n^j is the n-th symbol in the nodes of S_{n-1}^j.

2.1.2 Ranking of the nodes

In a star graph, S_n, the nodes are labeled with permutations of n symbols. The permutation labels, as such, do not define an ordering of the nodes in the network. However, in computations it is often required to define an ordering to the nodes. That is, the nodes are required to be ranked. For example, in sorting, it is required to have an ordering of the processors. Other examples are algorithms, such as FFT, that require order preserving communications. In computations involving order preserving communication, nodes of the same rank in subnetworks communicate. Order preserving communication is specifically relevant in hierarchical networks where it can be used in several types of algorithms [AQS91, AQS93].

Two schemes for ranking of the nodes in the star graph have been proposed in the literature [AQS91, JLD91, MS90]. Menn and Somani [MS90] defines a reverse lexicographic ordering of the nodes in the star graph.

DEFINITION 2.1 *In S_n, let p denote the processor associated with the node $a_1a_2...a_n$ and q denote the node $b_1b_2...b_n$. The ordering, $\prec$, on the processors is defined as follows: $p \prec q$ if there exists an i, $1 \leq i \leq n$, such that $a_j = b_j$ for $j > i$, and $a_i < b_i$. In other words, the processors are ordered in reverse lexicographic order. If $p \prec q$, p is said to preceed q. The rank $r(u)$ of a node u is the number of nodes v such that $v \prec u$, i.e., $r(u) = |\{v | v \prec u, v \in V_n\}|$. Clearly, $0 \leq r(u) \leq n! - 1$.*

This ordering of the nodes leads to a hierarchical ranking of the nodes in the star graph. However, an order preserving communication, with this ranking scheme, requires $O(n)$ time on an S_n [AQS91]. To make the communication cost of a computation comparable to that in a hypercube, it is necessary that an order preserving communication does not take more than $O(1)$ time.

The second ranking scheme for the star graph is based on obtaining a Hamiltonian cycle on it [JLD91]. The linear order of the nodes defined by the cycle is used for the ranking. As a result, in this scheme, the hierarchical structure of the star graph is lost.

In a recent work, Bagherzadeh et al. [BDL95] develops an enumeration scheme for star graphs. An interesting property of this scheme is that the graph induced by the nodes in any prefix or suffix in the enumeration is connected. The order preserving communication aspect of this ranking scheme has not been studied.

2.2 Routing

Routing in the star graph involves sorting of the source permutation into the destination permutation. Since the star graph is vertex symmetric, routing between any two arbitrary nodes u and v reduces to routing from some node R to the node labeled with the identity permutation.

DEFINITION 2.2 **Relative permutation:**
We define R as the relative permutation of u with respect to v such that $vR = u$, i.e. $R = v^{-1}u$.

From the expression $vR = u$ it can be seen that when R reduces to the identity permutation $I = 123...n$, u reduces to v. R can be reduced to I by sorting its symbols in one or more steps of exchange of the first symbol with a symbol at some position i, $2 \leq i \leq n$. Each of these exchange steps is equivalent to traversal of an edge of dimension i on the star graph. Thus the problem of routing from a node u to node v on the star graph reduces to the problem of sorting the relative permutation R of u with respect to v by applying the transposition operation $(1\ i), 2 \leq i \leq n$.

2.2.1 Shortest path routing

A permutation can be viewed as a product of disjoint cycles [Deo74]. For example, the permutation $P = 43251876$ consists of the cycles $(1\ 5\ 4), (3\ 2), (6\ 8)$ and (7) and we can express $P = (1\ 5\ 4)(3\ 2)(6\ 8)(7)$. The symbols contained in the unit length cycles represent the invariants in the permutation and therefore can be ignored in a cyclic representation of permutations. Cycles of length greater than 1 shall be referred to as $p-cycles$ [DT93a]. Shortest path routing on the star graph consists of step-by-step reduction of these p-cycles in the relative permutation. Each reduction step consists of correcting the positions of a symbol in the cycle. For the reduction of a cycle not containing '1', it has to be first merged with the cycle containing '1'. This merging consists of moving one of the symbols in the cycle to position 1. The reduction of the p-cycles can be done in any order[DT93a, MJ94a] independent of each other. These can also be merged before reduction [RG94].

Day and Tripathi [DT93a] obtained the set of rules for shortest path routing on the star graph. These rules in terms of relative permutation are given below. The shortest path routing from a node u to v can be achieved by sorting the relative permutation R of u with respect to v. This can be achieved by applying the rule set below to successively modify R until it is identity. Recollect that application of a transposition $(1\ i)$ means use of an edge of dimension i as it represents the generator g_i.

Rule-1: If $R(1) = 1$, apply transposition $(1\ i)$, where $R(i) \neq i$.
Rule-2: If $R(1) = i \neq 1$, then either,

(a) apply transposition $(1\ i)$,

or,

(b) apply transposition $(1\ j)$, where
$R(j) \neq j$ and j is not in the cycle containing 1.

The rules above can be used for finding any of the shortest paths between two nodes. Therefore, these can be used for minimal adaptive routing on the star graph. The application of Rule-1 implies merging of a p-cycle with the cycle (1), when $R(1) = 1$. The Rule-2(a) consists of correcting the position of symbol i as a step towards reduction of the corresponding p-cycle. The Rule-2(b) consists of merging the p-cycle containing symbol 'j' with the p-cycle containing '1'. In the shortest path routing process, the invariant symbols other than '1' are not touched.

A shortest path between two nodes in a star graph obtained through the above set of rules has the following properties:

PROPERTY 2.1 *a. An edge of dimension d does not occur in the path if the symbol at position d is invariant, i.e. $R(d) = d$.*
b. Edges of the same dimension do not occur more than twice in the path. If there are c p-cycles in R, not involving symbol 1, edges of c dimensions will occur twice in the path.

The occurance of two edges of a dimension in Property 2.1(b) is due to the requirement that a cycle not involving the symbol 1 needs to be merged before being reduced.

2.2.2 Deadlock-free routing

Misić and Jovanović [MJ94a, MJ94b] have developed an algebraic method for finding the shortest path between any two nodes in the star graph in terms of a sequence of generators. The method can also be used for finding alternative paths from a given generator sequence. They develop a scheme for deadlock-free routing in the star graph based on the virtual channel concept [DS87, DS89]. In their scheme, a route uses the next higher indexed buffer/virtual channel if the dimension of the physical channel to be used is higher than that of the preceeding physical channel used. Otherwise, the virtual channel of the same index is used in the next step. It is shown that the minimum number of buffers/virtual channels required for the purpose, with minimal routing, adaptive or otherwise, is $n-1$. Ravikumar and Goel [RG94] propose two schemes for deadlock-free wormhole routing. The first scheme uses a non-minimal deterministic routing called *e-star routing* and requires $\frac{n}{2}$ virtual channels, on the average, per physical channel. The second scheme uses minimal deterministic routing and on the average requires $\lfloor \frac{5n}{4} \rfloor - 1$ virtual channels per physical channel.

2.3 Fault tolerance

The fault tolerance is an important aspect in an interconnection network for a large scale parallel system. The *fault tolerance* [AK87] of the star graph, S_n, is $n-2$, that is one less than its degree. Therefore, it is said to be optimally fault tolerant. Connectivity, parallel paths, fault diameter etc. are some other measures of fault tolerance of the interconnection network. Connectivity of the star graph S_n is its degree $n-1$.

2.3.1 Fault diameter

The fault diameter of a network G with connectivity n is the maximum diameter of any network obtained from G by removing $n-1$ nodes [AK87]. The fault tolerance of the network is judged upon how close is its fault diameter to the diameter of the fault-free network. The fault diameter of the star graph, S_n has been shown to be D_n+1 for any n, $n \neq 4$ and $n \neq 6$, where D_n is the diameter of a fault-free S_n [Lat93b, JM94, RS93]. For $n = 4$ and $n = 6$, the fault diameter of S_n is $D_n + 2$.

2.3.2 Parallel paths

Day and Tripathi [DT93a], Jwo et al. [JLD90] and Ditezfelbinger et al. [DMS91] analyzed the parallel paths between nodes in a star graph. It has been found that in total there are $n-1$ parallel paths between any two nodes in an S_n. We state the results from these studies in the theorem below.

THEOREM 2.1 *Let u and v be two nodes in S_n and let R be the relative permutation of u with respect to v. Let c be the number of p-cycles in R and m be the total number of symbols in these p-cycles. Then,*
Case-1: If $R(1) = 1$, there are m parallel paths of length $c+m$ (minimum length) and $n-m-1$ parallel paths of length $c+m+2$ between the two nodes.
Case-2: If $R(1) \neq 1$, there are c parallel paths of length $c+m-2$ (minimum length), $m-c-1$ parallel paths of length $c+m$ and $n-m$ parallel paths of length $c+m+2$ between the two nodes.

The strategy followed in obtaining the set of parallel paths is to place the first symbol in the $n-1$ different possible positions in the first step, for the $n-1$ parallel paths and then correct the rest of the symbols before correcting this possibly misplaced symbol. The fact that the first symbol of the source node occupies a fixed position in the intermediate nodes in a path and that its position for the paths are distinct, ensures that the paths are node-disjoint, *i.e.* parallel. The exact procedures to obtain these parallel paths are presented in Appendix A. Some characteristics of the parallel paths obtained through these procedures are:

1. In the case of $R(1) = 1$, in each of the parallel paths the starting and terminating edges have the same dimension.

2. In the case of $R(1) \neq 1$,

   - The parallel paths of length $c+m-2$ correspond to the first symbols in the p-cycles in R. The starting edge of such a path correspond to the position of the first symbol in one of the p-cycles while the terminating edge corresponds to the position of the first symbol of another p-cycle or the last symbol of the p-cycle involving symbol '1'.
   - A path of length $c + m$ correspond to a symbol in the p-cycles not occuping the position one and the starting and terminating edges of such a path have a dimension corresponding to the position of this symbol in R.
   - A path of length $c+m+2$ correspond to the invariant symbols in R and the starting and terminating edges of such a path have a dimension corresponding to the position of this symbol in R.

2.4 Embedding schemes

The embedding of well known interconnection structures has traditionally been studied for various topologies. Ability to embed other interconnection structures, efficiently, implies richness in interconnectivity of the topology and allows efficient implementation of the algorithms available for the embedded interconnection structure on the topology under study. Embedability on the star graph has been studied by several investigators. Schemes for embedding various existing interconnection structures such as mesh, hypercube, full binary trees and cycles on the star graph have been proposed [JLD91, MPS94, NSK90, RWY93]. We discuss the results below.

2.4.1 Cycle embedding

Nigam et al. [NSK90] gave an algorithm for obtaining Hamiltonian cycle on a star graph. They also presented a scheme for obtaining disjoint cycles on the star graph. It was shown that an

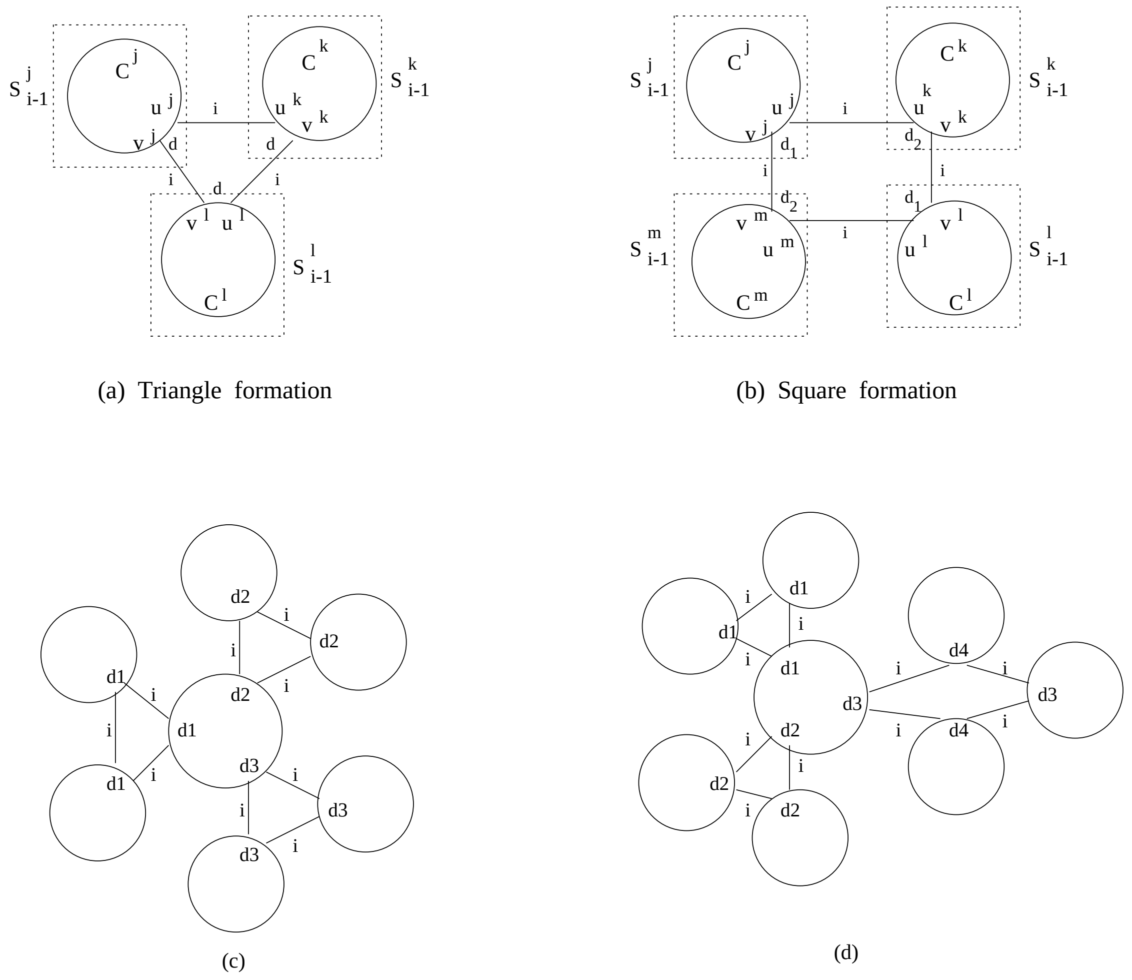


Figure 2.1: Jwo's method of cycle construction.

S_n can be decomposed into $\frac{n!}{(n-i)(n-i+1)}$ vertex disjoint cycles of length $(n-i)(n-i+1), 1 \leq i \leq n-1$.

THEOREM 2.2 *An S_n can be decomposed into $(n-2)!$ vertex disjoint cycles of length $(n-1)n$. [NSK90]*

COROLLARY 2.1 *An S_n can be decomposed into $\frac{n!}{(n-i)(n-i+1)}$ vertex disjoint cycles of length $(n-i)(n-i+1), 1 \leq i \leq n-1$. [NSK90]*

Jwo et al. [JLD91] proposed recursive schemes for embedding cycles of even length in star graph. There are no odd cycles in a star [AHK87, JLD91]. The scheme is based on the following observations.

LEMMA 2.1 *If there exists a cycle C in S_n, then for any given edge $(p, q) \in E_n$, it is possible to find a new even cycle C^* such that $|C| = |C^*|$ and C^* contains the edge (p, q). [JLD91]*

LEMMA 2.2 *Given any three substars $S^i_{n-1}, S^j_{n-1}, S^k_{n-1}$ of S_n, where $i \neq j \neq k$, it is possible to connect them in a triangle as shown in Figure 2.1(a). [JLD91]*

LEMMA 2.3 *Given any four substars $S^i_{n-1}, S^j_{n-1}, S^k_{n-1}, S^m_{n-1}$ of S_n, where $i \neq j \neq k \neq m$, it is possible to connect them in a square as shown in Figure 2.1(b). [JLD91]*

Based on Lemma 2.2 and Lemma 2.3 any number of sibling S_{i-1}s can be connected together. Figure 2.1 illustrates some examples. According to Jwo's method a cycle, C, can be constructed in an S_i from smaller cycles $C^j, C^k, \ldots$, in $S^j_{i-1}, S^k_{i-1}, \ldots$., that include the edges $(u^j, v^j), (u^k, v^k), \ldots$ respectively, such that $|C^j| + |C^k| + \ldots = |C|$, by connecting them with the inter-S_{i-1} edges (dimension i edges) from the nodes $u^j, v^j, u^k, v^k, \ldots$ and deleting the edges $(u^j, v^j), (u^k, v^k), \ldots$. If the cycles $C^j, C^k, \ldots$ are Hamiltonian cycles in the substars $S^j_{i-1}, S^k_{i-1}, \ldots$ in S_i then C is a Hamiltonian cycle of S_i.

The scheme also leads to a constructive proof that star graph S_n is Hamiltonian for $n > 2$. It should be noted that in the Jwo's method of construction three sibling S_{i-1}s are required for constructing a cycle of length l_c, $(i-1)! < l_c \leq 2(i-1)!$. Further, for cycles of length $\frac{i}{2}(i-1)! > l_c \leq \frac{i!}{2}$, for i odd, it is possible to construct only one cycle in an S_i with this method. In such situations, the scheme leads to non-optimal embeddings.

2.4.2 Mesh embedding

Embedding schemes for meshes have been proposed by Ranka et al. [RWY93], Jwo et al. [JLD91], and Qiu et al. [QMA91]. A mesh cannot be a subgraph of a star graph due to the fact that the smallest cycle in a star graph is of length six [JLD91]. Hence the minimum dilation possible in an embedding of mesh on the star is two. Ranka et al. developed a scheme for embedding an $n-1$ dimensional mesh of size $2 \times 3 \times \ldots \times n-1 \times n$ in an S_n with dilation three and expansion one. Based on a scheme given by Atallah [Ata88] the authors suggest that SIMD algorithms on uniform mesh of size $N^{\frac{1}{n-1}} \times \ldots \times N^{\frac{1}{n-1}}$ can be simulated on S_n with dilation $O(N^{\frac{n}{logN}})$.

Jwo et al. [JLD91] also gave a scheme for embedding a $n-1$ dimensional mesh of size $2 \times 3 \times \ldots \times n-1 \times n$ on an S_n with unit expansion and dilation 3. Based on unfolding of the $n-1$ dimensional mesh they gave a method for embedding any two dimensional mesh $k_1 \times k_2$, for $k_1 k_2 = n!$, in an S_n with the same dilation and expansion. Based on the construction of Hamiltonian cycles [JLD91] on the S_n a method was given for the embedding of $2 \times (\frac{n!}{2})$ mesh in S_n with dilation 2. In this the odd ranked nodes in the Hamiltonian cycle form the first row while the even ranked nodes form the second row. The authors also showed that there are special cases (*e.g.* 4×6) of meshes for which dilation 2 embeddings exists on the star graph.

Qiu et al. [QMA93] presented methods for embedding of $(n-2)! \times n$ mesh with dilation $n-1$ and expansion $n-1$ and of $(n-2)! \times (n-1)n$ mesh with dilation $2\lfloor \frac{n}{2} \rfloor + 1$ and unit expansion in S_n. These schemes are based on a modified method of obtaining disjoint cycles on the star graph [NSK90]. These results also hold for torus networks of identical sizes.

2.4.3 Hypercube embedding

Nigam et al. [NSK90] presented schemes for embedding hypercubes in star graphs with dilations 2, 3 and 4. It was shown that an n dimensional cube, $Q(n)$ can be embedded in a star graph of dimension 2^n with dilation 2 and in a star graph of dimension n with dilation 3. Further, the authors present a scheme for embedding a hypercube of dimension $d + \lfloor log_2(n+1) \rfloor$ in an S_{n+1} with dilation four from a dilation four embedding of an d-dimensional hypercube on an S_n. The results are embodied in the following theorem.

THEOREM 2.3 *[NSK90] The following embeddings of hypercubes on star graphs exists.*
(a) $Q(n)$ in S_{2^n} with dilation 2,
(b) $Q(n)$ in S_n with dilation 3, and
(c) If $Q(d)$ embedable in S_n with dilation 4 then $Q(d + \lfloor log_2(n+1) \rfloor)$ can also be embedded in S_{n+1} with dilation 4.

Miller et al.[MPS94] presented three schemes for dilation 3 embedding of hypercubes in star graphs. The first scheme is for embedding a hypercube of dimension k_n in an S_n. Here, k_n is defined recursively as $k_1 = 0$ and $k_n = k_{\lfloor \frac{n}{2} \rfloor} + k_{\lceil \frac{n}{2} \rceil} + \lfloor \frac{n}{2} \rfloor$. The second scheme embeds a hypercube of dimension $k_n + \lfloor log_2(n+1) \rfloor)$ in an S_{n+1}. The third scheme embeds a hypercube of dimension $n2^{n-1}$ in a star graph of dimension 2^n.

THEOREM 2.4 *[MPS94] Let the sequence $k_i, i \geq 1$, be defined recursively by $k_1 = 0$ and $k_n = k_{\lfloor \frac{n}{2} \rfloor} + k_{\lceil \frac{n}{2} \rceil} + \lfloor \frac{n}{2} \rfloor$ and let $L_n = \lfloor log_2(n+1) \rfloor$. Then it is possible to have the following dilation 3 embeddings:*
(a) $Q(k_n)$ in S_n for $n \geq 1$,
(b) $Q(k_n + L_n)$ in S_{n+1} for $n \geq 1$, and
(c) $Q(n2^{n-1})$ in S_{2^n} for $n \geq 0$.

The authors in [MPS94] also present two schemes for one-to-many embedding of hypercubes on the star graph. With one scheme a hypercube of dimension $11k + 2$ can be embedded on a star graph of dimension $13k + 2$ with dilation 2. With the other scheme a hypercube of dimension $k + k' + 1$ can be embedded on a star graph of dimension $n + n'$ with dilation 1 from the embeddings of k-cube in an S_n and k'-cube in an $S_{n'}$.

THEOREM 2.5 *[MPS94] It is possible to obtain a one-to-many embedding of $Q(11k + 2)$ in S_{13k+2} with dilation 2.*

THEOREM 2.6 *[MPS94] Assume that it is possible to obtain a dilation 1 one-to-many embedding of $Q(k)$ in S_n and that of $Q(k')$ in $S_{n'}$. Then it is possible to obtain a one-to-many embedding of $Q(k + k' + 1)$ in $S_{n+n'}$ with dilation 1.*

2.4.4 Full binary tree embedding

Bouabdallah et al. [BHOS94] and Heydemann et al. [HOS95] have developed schemes for embedding full binary trees (FBT) on the star graph. In [BHOS94] the authors present schemes for embedding FBTs with dilation 1 and 2. In the dilation 2 scheme, it is possible to embed an FBT of height $(n+1)\lfloor log_2 n\rfloor - 2^{\lfloor log_2 n\rfloor+1} + 2$, in an S_n. The embeddings have node-congestion higher than 1. In the dilation 1 embedding scheme, the height of the FBT embedable in an S_n is lesser by a factor of at least $\frac{n}{2}$ [HOS95].

Heydemann et al. [HOS95] present a scheme for congestion-free embedding of FBTs of height $(n + 1)\lfloor log_2 n\rfloor - 2^{\lfloor log_2 n\rfloor+1} + 1$, in an S_n with dilation 4. The authors also present another scheme for embedding FBTs with dilation higher than 4. The height of the FBT embedable in an S_n in this case is at least $\log_2(n!) - 3\lfloor log_2 n\rfloor + 3$.

2.5 Algorithms

Significant research on the star graph topology has been carried out on the algorithm development front. These works include some basic algorithms such as translation, concentration, distribution, reversing, prefix computation etc.; global communication algorithms such as broadcasting, gossiping, scattering etc., which are used as components of other more complex algorithms; and other classes of algorithms for problem domains such as sorting, graph, linear systems, computational geometry etc.

2.5.1 Algorithms based on order preserving communication

Menn and Somani [MS90] define a reverse lexicographic ordering of the nodes on the star graph. This leads to a hierarchical ranking of the nodes in it [AQS91]. Based on this ranking of the nodes the authors in [AQS91] define *order preserving communications* on the star graph which consist of communications between nodes of identical ranks in substars. Such an order preserving communication on an S_n takes $O(n)$ time. This order preserving communication is used in a set of fundamental algorithms: translation (cyclic shift), concentration, reversing and distribution. Each of these require $O(n)$ order preserving communications so that the over all complexity of each of these is $O(n^2)$.

2.5.2 Algorithms based on group-copy paradigm

In [AQ93] the authors define a communication paradigm called the *group-copy* on the star graph. It consists of interchanging of data by the nodes in two sets of substars, S_{n-1}s in an S_n in a bijective manner. The one-to-one correspondence between the node sets of the two sets of substars are defined such that a corresponding node pair u and v in the two sets are related as $u = v(l\ n)$, $1 \leq l < n$, so that the distance between them is at most 3. The pair of nodes u and v communicate using a path (l, n, l). The group-copy paradigm has a congestion of $O(n)$ as $O(n)$ pair of nodes will use an edge of dimension n in the communication.

This group-copy paradigm is used to develop an algorithm for parallel prefix computation of $n!$ elements on an S_n requiring $O(nlogn)$ steps [AQ93]. It is easy to see that segmented prefix [Lei92] can also be performed on the star graph in $O(nlogn)$ time. As a result interval broadcasting [AQS91] can be performed on the S_n in $O(nlogn)$ time.

Using the group-copy paradigm Akl and Qiu [AQ93] adapt a hypercube algorithm for computation of *minimum spanning forest* in a graph [DDP90b]. The algorithm belongs to the *non-order preserving* ascend/descend class [AQ93] of algorithms. The algorithm runs in $O(nlogn)$ time in an S_n. This is of the same order as for the hypercube. Other hypercube algorithms for graph problems such as connected components, fundamental cycle set, the bridges of a graph, bipartiteness checking [DDP90a] can also be adapted using the same technique to run on S_n in $O(nlogn)$ time.

2.5.3 Sorting and selection

Sorting is an important operation which is very often required in computer applications. Several sorting algorithms have been developed for the star graph interconnection network [QA, MS90, RW93b]. Menn and Somani [MS90] proposed a sorting algorithm for the star which has a time complexity of $O(n^3 logn)$ for sorting $n!$ elements on the S_n. The algorithm is based on odd-even transposition sort. In the algorithm the S_n is viewed as a multidimensional grid. The algorithm involves merging of sorted lists in two groups of substars, S_{n-1}s and the merging operation requires $O(n^2)$ time.

Rajasakaran et al. [RW93a] give a bitonic sort based algorithm with the same order of complexity but involving a smaller constant multiplicative factor. A randomized sorting algorithm was proposed in [RW93b] that runs in $O(n^3)$ time with high probability. The algorithm is based on repeated selection. It uses prefix computation and selection algorithms that require $O(n^2)$ time for n prefix computations and $O(n^2)$ time for n selections. The prefix computation is performed using a tree like subgraph.

Qiu et al. [QA] proposed an algorithm for sorting $N \geq n!$ numbers distributed evenly with each processor holding not more than $M = \lceil \frac{N}{p} \rceil, p = n!$, numbers can be sorted in time $O(MlogMnlogn + Mn^3 logn)$, with respect to reverse lexicographic processor ordering. When $N = n!$, the complexity becomes $O(n^3 logn)$. An asymptotically faster sorting algorithm was presented in [QA] which requires $O(\frac{N}{p}nlogp + log(\frac{N}{p})n^4 log^2 n)$ time. This algorithm uses a selection algorithm that has a time complexity of $O(\frac{N}{p}loglogp + log(\frac{N}{p})n^3 logn)$.

2.5.4 Computational geometry algorithms

Based on some of the above algorithms, Akl et al. [AQS91] presented several computational geometry algorithms for the star graph. They develop an algorithm for the merging slope technique that runs in $O(n^2)$ time on an S_n. Using this they develop algorithms for convex hull computation and for finding the smallest enclosing box, each requiring $O(n^3 logn)$ time and for finding critical support line requiring $O(n^2)$ time. The authors also suggest methods, based on the merging slope technique, for computing diameter, width, minimum linear fit of a set of points, maximum distance between two convex polygons, and vector sum of two convex polygons.

General prefix computation (GPC) is a generic problem that captures the most common difficult 'kernel' of many problems [SS89b]. It is shown in [SS89b] that GPC can be computed on the hypercube of N processors in $O(log^2 N)$ time. Akl et al. [AQS91] suggest that the GPC algorithm for the hypercube can easily be modified to run in $O(n^3 logn)$ time on S_n, for $N = n!$, using the merging and the interval broadcasting algorithms. Some of the problems that can be solved using GPC are ECDF searching, 2-set dominance counting and 3-D maximal elements computation, triangulating a set of points in a plane etc [SS89b].

2.5.5 Fast Fourier transform

Fragopolou et al. [FA94] have developed an algorithm for computing $n!$-point FFT in an S_n. The time complexity of the algorithm is $O(n^2)$. The computation is performed in $n-1$ stages and the i-th stage consists of computing $(n-i+1)$-point DFTs on a linear array. A method of decomposing the star graph into disjoint cycles [QMA91] is used for obtaining these linear arrays on the star. Since a systolic algorithm is used for computing the $(n-i+1)$-point DFTs the communication complexity of the algorithm is also $O(n^2)$.

2.5.6 Global communication algorithms

Global communication operations are often required in various algorithms for multicomputers [JH89, SS89a]. Some of these operations are:

- One-to-all broadcast– a node transmits a message to all other nodes.
- One-to-all scatter– a node transmits a possibly distinct message to each node in the network.
- All-to-all broadcast (Gossip)– Each node in the network performs its own broadcast.
- All-to-all scatter (Total exchange)– each node in the network performs its own scatter operation.
- Single-node accumulation– All the nodes in the network sends a message to a common node.

It has been shown [AHK87] that minimum number of steps required for one-to-all broadcasting on the S_n with the single-port model is $O(nlogn)$. Akers et al. [AHK87] gave an algorithm for one-to-all broadcasting with the single-port model requiring $3(nlogn - \frac{n}{2})$ steps. Mendia and Sarkar [MS92] also give an algorithm for one-to-all broadcasting on the star graph with the single-port model which requires $O(nlogn)$ steps. The multiplicative factor in this case is lower than that in [AHK87]. Berthome et al. [BFP92] give algorithms for broadcasting and gossiping with the single-port model. The broadcast operation requires $2\sum_{k=2}^{\frac{n}{2}} log2k + \lceil\frac{3n}{4}\rceil$ steps and the gossiping operation requires $\sum_{k=1}^{n}\lceil logk\rceil + n$ steps. Sheu et al. [SLC93] give an algorithm for broadcasting with the multi-port model that requires $2n-3$ steps. The algorithm uses the same set of paths as in [MS92]. Sheu et al. [SWC95] give another algorithm with single-port model that requires $O(nlogn)$ steps. The algorithm is free of redundant message traffic. Akl and Qiu [AQ93] give a broadcasting algorithm based on the group-copy paradigm. This algorithm also runs in $O(nlogn)$ time but has a multiplicative factor of 3.

Misić and Jovanović [MJ94a] give recursive algorithms for global communication operations assuming the single port communication model for the nodes. The algorithms are congestion-free. The one-to-all broadcast algorithm runs in $\frac{n(n-1)}{2}$ steps. The one-to-all scatter and all-to-all broadcast algorithms require $O(n!-1)$ steps while the total exchange algorithm runs in $(n!-1)\hat{d_n}$ steps, where $\hat{d_n}$ is the average distance in an S_n. The later three algorithms are optimal for the single-port model.

Fragopolou and Akl [FA93b, FA93a] give methods for construction of various types of spanning trees on the star graph based on which global communication algorithms are developed. With single-port model all-to-all broadcast and one-to-all scatter require $n!-1$ steps while total exchange require $n!\hat{d_n}$ steps. On the other hand, with the multi-port model all-to-all broadcast and one-to-all scatter require $\lceil\frac{n!-1}{n-1}\rceil$ steps and total exchange requires $\frac{n!\hat{d_n}}{n-1}$ steps. The authors also give methods for construction of multiple edge disjoint spanning trees which are used to define fault tolerant algorithms for global communication operations.

2.6 Addressing the problem of extremely rapid expansion rate

The star graphs grow very rapidly with dimension. The size (number of nodes) of a star graph of dimension $n+1$ is $n+1$ times that of a star of dimension n. This extremely rapid rate of expansion manifests as shortcomings of the network in two ways. The first is the possibility of poor utilization. For instance, consider a job that requires a thousand processor nodes. A 7-star, the smallest star having a thousand nodes, consists of 5040 nodes. The utilization will be less than 20% in this case. Poor extensibility is the other major problem resulting from the high expansion rate of the network. The cost of upgrading an existing network to its next higher size is prohibitive. Several schemes have been proposed in the literature to over come these problems of the star graph.

2.6.1 Arrangement graph

A generalization of the star graph, called the *arrangement graph* was proposed by Day and Tripathi [DT92] to address the rapid expansion problem of the star graph.

DEFINITION 2.3 **Arrangement graph**
The (n,k)-arrangement graph $A_{n,k} = (V, E)$, $1 \leq k \leq n$, *is an undirected graph given by:*
$V = \{p_1, p_2, ..., p_k | p_i \in <n> \text{ and } p_i \neq p_j \text{ for } i \neq j\}$ *and*
$E = \{(p, q) | p, q \in V \text{ and for some } i \in <k>,\ p_i \neq q_i \text{ and } p_j = q_j \text{ for } j \neq i\}$.

That is, the vertices of $A_{n,k}$ are the arrangements (permutations) of k elements out of n elements of $< n >$, and the edges of $A_{n,k}$ are links between arrangements which differ in exactly one of their k positions. For $k = n - 1$, the (n,k)-arrangement graph reduces to the star graph S_n and for $k = 1$, it is a complete graph on n vertices.

The arrangement graph is symmetric, hierarchical and has sublogarithmic degree and diameter. However the algorithm implementability aspect of the graph is unknown.

2.6.2 Super star graph

Sur and Srimani [SS93a] proposed an incrementally extensible network topology called the *super star*. The graph can be defined for any arbitrary number of nodes. The construction of the graph consists of appropriate interconnection of different sized star graphs. If N is the number of nodes, $n! < N < (n+1)!$, the proposed graph is a super graph of several star graphs of dimensions less than or equal to n. The topology has the following interesting properties:

- Additional nodes can be added to an existing network with no or minimal reorganization.
- The topology has sublogarithmic degree and diameter. The diameter of a super star graph $H_n(N)$, $n! < N(n+1)!$, is $\lfloor \frac{3n}{2} \rfloor + 1$. The maximum degree of any node is $n + 1$.
- It is optimally fault tolerant in the sense that the vertex connectivity of the graph is equal to the minimum degree of a node.
- The number of edges in the network is linear in the number of nodes and the difference between maximum degree and minimum degree of a node is 2, a constant independent of the number of nodes, $i, e.$ the graph is *almost regular.*

Although the topology is incrementally extensible, the algorithm implementability aspect of the graph is totally unknown.

2.6.3 Incomplete star

Recently, Latifi and Bagherzadeh [Lat93b] proposed the *incomplete star* in the line of the *incomplete hypercube* [Kat89] to address the problem of extremely rapid rate of expansion of the star graph. They propose a selection method for designing an N processor network, $(n-1)! < N < n!$, such that it remains a subgraph of the S_n regardless of N. For the construction, a coefficient vector as an (n-1)-tuple $b_{n-1}b_{n-2}...b_2b_1, (1 \leq b_i \leq i)$ is defined such that- $N = b_{n-1}(n-1)! + b_{n-2}(n-2)! + ...b_2 2! + b_1 1! = \sum_{i=1}^{n-1} b_i(i!)$. In principle, the coefficient vector gives the number of complete stars of dimension $n-1$, or lower, contained in the original structure, where the construction is maximal with respect to the higher dimension.

A special case of the topology for $N = b_{n-1}(n-1)!$, $2 < n$, is a graph referred to as C^{n-1} graph. This special case of incomplete star has been referred to as the *clustered-star* in [Lat93a].

DEFINITION 2.4 **Clustered-star:** *[Lat93a]*
An (n-1)-dimensional clustered-star(CS) of order m, denoted by CS^m_{n-1} is an n-dimensional star with $n-m$ of its S_{n-1}s missing. In other words, CS^m_{n-1} is a collection of m (n-1)-stars which are connected to each other according to the basic connection rules defined for the star graph. The n-th symbol (fixed symbol) for each (n-1)-star, $S_{n-1}(k)$, is k, where $n-m < k \leq n$. Thus, $CS^m_{n-1} = \cup_k S_{n-1}(k)$, for $n-m < k \leq n$; furthermore, $CS^m_{n-1} \subset S_n$.

In the above definition, the notation $S_{n-1}(k)$ denotes the S_{n-1} with the n-th symbol fixed to k.

The authors have studied the topological properties- diameter, average distance and Hamiltonicity of CS^m_{n-1} graph. The diameter is $\lfloor \frac{3(n-1)}{2} \rfloor$, same as that for an S_n. The average distance $d_{CS^m_{n-1}} \approx d_{n-1} + 2 - \frac{2n+m-3}{m(n-1)}$, where d_{n-1} is the average distance in an S_{n-1}. It is also shown that CS^m_{n-1} is Hamiltonian for $m = 4$ and $m = 3k$, where $k \neq 2$ is a positive integer.

According to the definition, there is restriction on the choice of the substars S_{n-1}s of an S_n that can be included in a clustered-star. This restriction limits the utility of clustered-star to near linear expansion of the interconnection structure. It can not be used for efficient utilization of the star graph based networks through processor allocation in terms of clustered-stars. Further, the algorithm implementation aspect of the incomplete star graph, except for broadcasting on clustered-star, has not been studied.

2.7 Processor allocation

Processor allocation in an interconnection network is an important problem in a multicomputer system. The problem has extensively been studied for the hypercube and other interconnection networks [CC87, DD95, DH91, RTS95]. Support for a multi-tasking environment in a multicomputer system is usually achieved through sharing of the processor resources in contrast to time sharing used in conventional sequential computers. Sharing of processor resource implies allocation of portions of the network to individual concurrent tasks (programs/jobs). To ensure that a task can be implemented efficiently and that the tasks do not interfere with each other, the allocated processors should form a connected subgraph having a suitable structure and concurrently allocated subgraphs do not overlap with each other. Usually, in a hierarchical interconnection network a sub-network which forms a lower level structure of the same family is allocated. For example, in a hypercube based system distinct subcubes are allocated.

The techniques used for processor allocation can be divided into two categories: Bit mapped technique (BMT) and available subgraphs technique (AST). In a BMT a bit is maintained [CC87] for each processor to indicate whether it is free while in an AST a list of free subgraphs (*e.g.* subcubes in case of hypercube) is maintained. The ASTs differ in whether the list maintained consists of distinct (non-overlapping) subgraphs or overlapping subgraphs. We shall briefly review these techniques, as used for processor allocation in hypercube based systems.

In BMT, for allocation in a hypercube system, it is necessary to find 2^k free processors that form a k-subcube on request for a k-subcube. The buddy strategy (BS) [CC87] is a BMT. It is based on systematic halving and doubling of subcubes for allocation and deallocation. The buddy strategy has been proved to be optimal under a LIFO release scheme. However, its performance deteriorates under dynamic conditions. The schemes based on BMT are also slow in allocation.

The AST offers the advantage of quickly recognizing whether or not a request can be satisfied. This takes $O(n)$ time, in a n-dimensional hypercube, in the worst case, in contrast to $O(2^n)$ in BMT. Dutt and Hayes [DH91] maintains a non-overlapping set of subcubes called the maximal set of subcubes (MSS). An MSS is formed through coalescing of smaller subcubes using a consensus graph. The coalescing algorithm has a time complexity of $O(n3^{2n})$. An approximate heuristic algorithm has a time complexity of $O(n2^n)$. The coalescing needs to be performed for every deallocation and most allocations.

Wang and Yang [WY91] proposed an overlapping AST based strategy for allocation in Hypercubes. They suggested that maintaining an overlapping rather than disjoint set of available subcubes result in less fragmentation for non-cubic allocation, e.g. in case of embedding of rectangular grids, binary trees etc. Rai et al. [RTS95] also presented a scheme based on overlapping AST for allocation in hypercubes. The selection of a subcube for allocation is based on a measure of disjointness of the subcube from the other subcubes (separation factor) and a measure of overlap of the subcube with the other subcubes (overlap syndrome). A list of maximal set of subcubes is computed to compute the separation factor and the overlap syndrome for the subcube. The time complexity of both allocation and deallocation algorithm is $O(n3^{2n})$.

The problem of processor allocation on the star graph was also considered in [Lat94]. In this, the allocation is considered in terms of substars. The author introduces the concept of *star code* for recognizing substars, like the *gray code* used in the case of hypercube. The author derives the optimal number of star codes required for complete substar recognition in an S_ns to be 2^{n-2}. A method for construction of such a set of star codes is also proposed. Allocation of processors in a star graph in terms of substars, however, leads to inefficient utilization of the processor resources.

Chapter 3

Foundation

In this chapter we introduce and characterize the theoretical concepts that are used in the various schemes and algorithms that we present in this thesis. The basic tool used in the work is a set of isomorphic transformations on the star graph. We define two classes of isomorphisms, namely, symbol swap transformation and dimension swap transformation. The symbol swap transformation allows a subgraph in the star to be mapped to another subgraph, having the same orientation with reference to the dimensions of the edges. On the other hand, the dimension swap transformation allows a subgraph in the star to be mapped to another subgraph with a different orientation.

The first application of the symbol swap transformation is in developing schemes for ranking the nodes and the substars of the star graph. We propose two ranking schemes. These will be used for developing algorithms as well as embedding schemes for the star graph.

Another set of concepts called (i,k)-centre and the domain of a node results in a domain decomposition of the star graph and allows us to define a set of communication paradigms on the star. We also characterize these concepts. The (i,k)-centre concept is also used in the embedding schemes.

We start with a list of notation used in this thesis. This is followed by a discussion on the hierarchical decomposition of the star graph and some related concepts. The isomorphic transformations are presented in the next section. The two ranking schemes are presented in Section 3.4. In Section 3.5, we present the domain decomposition of the star.

3.1 Notations

In this thesis we define several new terms and introduce some new notations. We list these notations below. The definitions will be introduced in appropriate places when necessary.

Notation	Description
$< S_i >$	: The set of permuting symbols in an S_i.
$(i\ j)$	: An operator for the transposition of the i-th and the j-th symbols of a permutation.
g_i	: The generator for an edge of dimension i. $g_i = (1\ i)$.
$u \circ v$	: Equivalent to $v(u)$ if u and v are permutations. Where clear we will drop the " $\circ$ " and write uv.
ug_i	: Equivalent to $g_i(u)$ or $u \circ g_i$.
u_i	: The i-th symbol in a permutation u.
$pos(s, u)$	: Position of the symbol s in the permutation u.
s_n^j	: The fixed symbol in nodes of S_{n-1}^j at position n.
$r_i(u)$	: Rank of the node u in S_i.
$r_k(S_i)$	: Rank of an S_i in S_k $(k > i)$.
S_{n-1}^j	: The j-th ranked S_{n-1} in S_n, where $j = n - s_n^j$.
$*pq$	: A substar S_{n-2} in an S_n with the fixed symbols p and q at positions $n-1$ and n.
u^k	: A node in a k-th ranked substar, S_i^k.
$SC_{i,k}$	: A star cluster induced by k sibling S_is.
$(d_1, d_2, .., d_k)$	: A dimension sequence implying a path.
T_h	: An FBT of height h, or of $2^{h+1} - 1$ nodes.
t_s	: Start-up time in a message communication.
t_h	: Per-hop time in a message communication.
t_w	: Word transfer time over a communication channel.

3.2 Hierarchical decomposition of the star graph

There are various alternative ways to decompose a star graph [AHK87]. The hierarchical decomposition is the most suitable one for algorithm development. Also for the hierarchical decomposition, due to the strongly hierarchical nature of the star graph, there are large number of options.

Definition 3.1 **Composition dimension sequence**
An S_{i+1} can be constructed by taking $i+1$ S_i and connecting them appropriately with a set of $\frac{(i+1)!}{2}$ edges of the same dimension (not already in the S_is). This dimension will be called the composition dimension for the S_{i+1}. To construct an S_n from the S_1s (i.e. the nodes) step by step hierarchically, we can chose a sequence of $n-1$ composition dimensions one for each of the substar levels 2 to n. Such a dimension sequence we shall refer to as a composition dimension sequence(CDS) for an S_n.

Due to the strongly hierarchical nature of the star graph the composition dimension sequence can be chosen in any order. There are $(n-1)!$ possible sequences of composition dimensions of an S_n. The most natural sequence is the monotonic ascending sequence $(2, 3, 4, \ldots, n)$. We refer to this as the *primary composition dimension sequence* (PCDS). This sequence corresponds to the construction of an S_2 by connecting two S_1s with a dimension 2 edge followed by that of an S_3 by connecting three S_2s with a set of three dimension 3 edges and so on. We refer to an S_i composed using the PCDS as a PCDS S_i. In a PCDS S_i only the symbols at the first i positions permute in the permutation representation of its nodes. The rest of the symbols remain fixed in their positions.

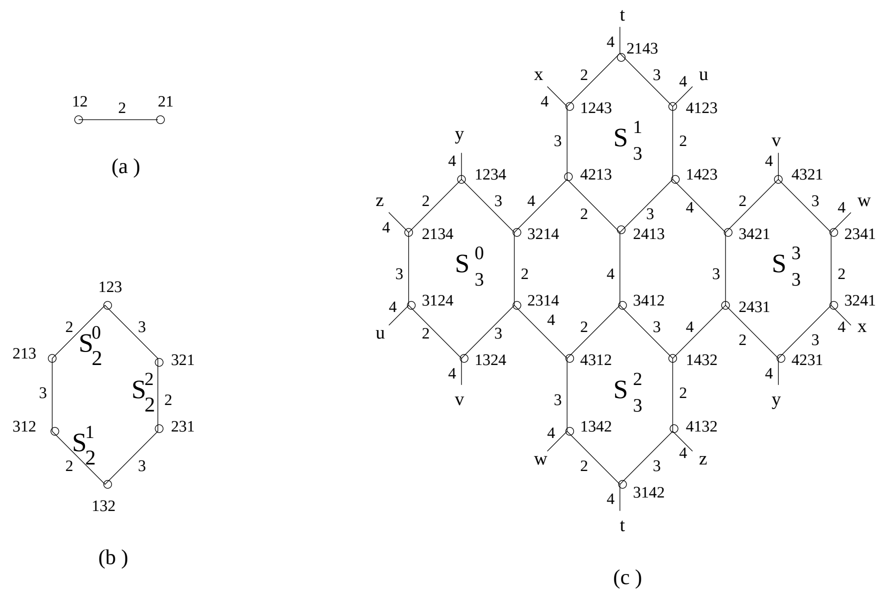


Figure 3.1: Sibling substars and their labeling. (a) S_2, (b) S_3, (c) S_4.

A CDS that is not a PCDS is referred to as a *secondary composition dimension sequence* (SCDS). If the composition dimension sequence chosen is $(d_2, d_3, \ldots, d_n)$, where $d_i \in \{2, 3, 4, \ldots, n\}$ and $d_i \neq d_j$ for $i \neq j$, for $2 \leq i, j \leq n$, then the composition dimension for S_i is d_i and in the S_i the symbols in positions $1, d_2, d_3, \ldots, d_i$ permute while the rest remain fixed.

In this thesis we shall normally assume the PCDS. In case of a departure from this we shall specify the SCDS assumed. It may be noted that the above definition of composition dimension sequence is equivalent to the definition of *star code* defined in [Lat94].

We have seen that an S_i within an S_n can be composed of i S_{i-1}s.

DEFINITION 3.2 **Sibling substar**
In an S_n the S_{i-1}s with a common parent S_i are called sibling substars.

We use a new scheme for denoting the substars which is based on ranking of the substars. The general scheme for the ranking shall be discussed later in the chapter. The ranking of the PCDS S_{i-1}s in the PCDS S_i containing the identity permutation node is done as follows. An S_{i-1} in the S_i is assigned a rank $j = i - s_i^j$, where s_i^j is the i-th symbol in the nodes of the S_{i-1}. As $s_i^j \in \{1, 2, \ldots, i\}$, $j \in \{0, 1, \ldots, i-1\}$. The S_{i-1} will be denoted by S_{i-1}^j. For example, an S_4 (Figure 3.1) consists of S_3^0, S_3^1, S_3^2 and S_3^3. The 4-th symbols in the permutations of S_3^0, S_3^1, S_3^2, and S_3^3 are 4,3,2,1 respectively. It should be noted that the rank of the S_{i-1} containing the identity permutation node is zero. This is because the i-th symbol of the identity node and the S_{i-1} containing it is i.

An SCDS S_{i-1} in an S_i will also sometimes be denoted as S_{i-1}^j. However, in this case, the rank j assigned to the S_{i-1} will be arbitrary, though distinct from its siblings.

3.3 Isomorphic transformations on the star graph

In this section we discuss two classes of isomorphic transformations, namely the *symbol swap transformation* and the *dimension swap transformation*, on the star graph and study their characteristics. These transformations form the basic tools for the synthesis and analysis of the various schemes and algorithms developed in this thesis.

3.3.1 Symbol swap transformations

Due to the vertex and edge symmetric nature of the star graph it is possible to map a subgraph in one portion of the star to another. For this it is necessary to have an isomorphic transformation. The symbol swap transformation is one such transformation.

Definition 3.3 **Symbol swap transformation**
Let $< S_n >= \{1, 2, \ldots, n\}$, *the set of symbols in the nodes of* S_n. *Consider a one-to-one mapping* $t :< S_n >\rightarrow< S_n >$ *and define* $t(u_1u_2\ldots u_n) \triangleq t(u_1)t(u_2)\ldots t(u_n)$, *where* $u_1u_2\ldots u_n$ *implies a node* u *in* S_n. *It is evident that* $t(u)$ *is also a node in* S_n. *The node* $t(u)$ *is said to be a symbol swap transformation image of* u *and* t *is called a symbol swap transformation operator.*

For example, let $u = u_1u_2\ldots u_j\ldots u_k\ldots u_l\ldots u_n$ and a symbol swap transformation operator t is defined as:
$(t(u_j) = u_l, t(u_l) = u_k, t(u_k) = u_j$, and $t(u_m) = u_m$ for $1 \leq m \leq n$, and $m \neq j, k, l)$
Then $t(u) = t(u_1)t(u_2)\ldots t(u_j)\ldots t(u_k)\ldots t(u_l)\ldots t(u_n) = u_1u_2\ldots u_l\ldots u_j\ldots u_k\ldots u_n$. In this transformation, the symbols u_m are invariant. The non-invariant symbols are said to *participate* in the transformation. Since specifying the invariant symbols is redundant, these can be omitted. For example, $(t(5) = 3, t(3) = 5)$ is a symbol swap transformation on the nodes of an S_5 and $t(23154) = 25134$. If μ symbols participate in a symbol swap transformation we shall sometimes refer to it as a *μ-symbol swap transformation*. Symbol swap transformation differs from usual *positional* permutation operation in that here a set of symbols permute irrespective of their positions. In a positional permutation, on the other hand, the permuting positions are specified. Irrespective of what symbols occupy these positions, they permute. From this point of view, a symbol swap transformation can be called a *symbolic* permutation.

The application of a symbol swap transformation on an edge is equivalent to its application on the corresponding end nodes. Similarly, application of a symbol swap transformation on a subgraph of S_n is equivalent to its application on all the nodes of the subgraph. The Lemma below follows from the definition of symbol swap transformation.

Lemma 3.1 *A symbol swap transformation is an isomorphism on the star graph.*

Proof. Let (u, v) be an edge in S_n. Let $u = s_1s_2...s_j...s_l...s_n$ and $v = ug_l = u(1\ l) = s_ls_2...s_j...s_1...s_n$. In the two nodes u and v all the symbols $s_j, 1 < j \leq n, j \neq l$ are identical. When a transformation operator t is applied on u and v to get u' and v' a symbol s_j in u will be replaced by $t(s_j)$ in u' and that in v will also be replaced by $t(s_j)$ in v'. Thus we will have $u' = t(s_1)t(s_2)...t(s_j)...t(s_l)...t(s_n)$ and $v' = t(s_l)t(s_2)...t(s_j)..t(s_1)..t(s_n)$. It is easy to see that $v' = u'(1\ l) = u'g_l$. Therefore, (u', v') is an edge and hence transformation is adjacency preserving [1]. Also, the dimension of the image edge (u', v') is same as that of the edge (u, v). By definition, a symbol swap transformation is an one-to-one transformation from $< S_n >$ onto $< S_n >$ which implies that it is also an one-to-one transformation on S_n. Therefore, it is

[1]A transformation in a graph is said to be adjacency preserving if when it is applied on an edge the resulting image also is an edge.

an isomorphism on the star graph. ☐

One notable fact resulting from the discussion of the proof above is that a symbol swap transformation preserves paths. That is, the image of a path comprises of the same sequence of dimensions as the original path.

It is often necessary to map the nodes of two sibling substars in a bijective manner. The following theorem gives us clue to obtaining a symbol swap transformation for such a mapping.

THEOREM 3.1 *The necessary and sufficient condition for mapping a node in* S^j_{i-1} *to a sibling substar* S^k_{i-1} *through a symbol swap transformation is that- i) the transformation involves the substitution operation* $t(s^j_i) = s^k_i$*, and ii) none of the fixed symbols of the parent* S_i *participate in the transformation.*

PROOF. The i-th symbols in the nodes of S^j_{i-1} and S^k_{i-1} are s^j_i and s^k_i respectively. Therefore it is a must that the symbol s^k_i takes the position of the symbol s^j_i. Hence, in the mapping of the nodes in S^j_{i-1} to a sibling substar S^k_{i-1} the transformation must involve the substitution operation $t(s^j_i) = s^k_i$. Secondly, if any of the fixed symbols of the parent S_i (*i.e.* those in positions $i+1$ to n) are altered the node will map to some other substar outside the parent S_i. Therefore these symbols must not participate in the transformation. Hence the above conditions are *necessary* for the mapping.

The above conditions are also *sufficient* because the involvement of any other symbol substitution operation participated by the permuting symbols of the parent S_i do not affect the mapping, in the sense that the image node will belong to S^k_{i-1}. ☐

From the above theorem it can be observed that if a symbol swap transformation involves the two symbol substitution operations $(t(s^j_i) = s^k_i, t(s^k_i) = s^j_i)$ and the symbols $s^j_m, i > m \leq n$, do not participate in the transformation, the nodes in the two sibling substars S^j_{i-1} and S^k_{i-1} will map to each other in a bijective way. It is also easy to see that the transformation $(t(s^j_i) = s^k_i, t(s^k_i) = s^j_i)$ is the minimal symbol swap transformation for a bijective mapping between the nodes in the two sibling substars.

DEFINITION 3.4 **i-th symbol swap transformation**
The minimal symbol swap transformation $(t(s^j_i) = s^k_i, t(s^k_i) = s^j_i)$ *that defines a bijective mapping between the nodes of two sibling substars* S^j_{i-1} *and* S^k_{i-1} *is referred to as an i-th symbol swap transformation.*

The i-th symbol swap transformation on a node v in S_{i-1}^k is basically a transposition operation $(l\ i)$, where $1 \leq l < i$, is the position of s_i^k in v. The value of l differs from node to node in S_{i-1}^j. Since the i-th symbol swap transformation is a special case of symbol swap transformation, it is also an isomorphism. Consequently, it can be used to generate isomorphic copies of subgraphs in sibling substars.

The following lemmas and theorems characterize some of the properties of the i-th symbol swap transformation. These properties of the transformation will find use in the later chapters of the thesis.

LEMMA 3.2 *If an i-th symbol swap transformation is performed on a subgraph G in an S_i to map its portion in S_{i-1}^j to S_{i-1}^k the portion of G in S_{i-1}^k maps to S_{i-1}^j. However, the image of the portion of G in $S_{i-1}^l, l \in \{0, 1, ..., i\} \backslash \{j, k\}$ remains within S_{i-1}^l.*

PROOF. The i-th symbol swap transformation that maps a subgraph in S_{i-1}^j to S_{i-1}^k is $(t(s_i^j) = s_i^k,\ t(s_i^k) = s_i^j)$. The participating symbols s_i^j and s_i^k will occupy positions 1 to $i-1$ in the nodes of an $S_{i-1}^l, l \neq j \neq k$, but not the position i. Therefore the swapping of these two symbols will leave the i-th symbol in the nodes of S_{i-1}^l unchanged so that a node in S_{i-1}^l to a node in S_{i-1}^l itself. Hence we have the lemma. ◻

Consider the mapping of a path that spreads across a set of sibling S_{i-1}s to another set of sibling S_{i-1}s through i-th symbol swap transformation. As per Lemma 3.2, the mapping has to be performed between the S_{i-1}s in the two sets, pair-wise.

THEOREM 3.2 *The dilation of the i-th symbol swap transformation image of a path that spreads across multiple sibling S_{i-1}s is three. The specific edges having dilation three are the images of the inter-S_{i-1} edges of the original path.*

PROOF. Without loss of generality let us consider a path that spreads across two sibling substars with only one inter-substar edge. Let P be a path that spreads across two sibling substars S_{i-1}^j and S_{i-1}^k and let (u^j, u^k) be the inter-substar edge on it. See Figure 3.2.

The node u^j belongs to S_{i-1}^j and u^k belongs to S_{i-1}^k. Let P^j be the section of the path which resides entirely in S_{i-1}^j and P^k be the section of the path that resides entirely in S_{i-1}^k. We obtain an i-th symbol swap transformation image P', of the path P, in sibling substars S_{i-1}^l and S_{i-1}^m by performing one i-th symbol swap transformation on P^j on to S_{i-1}^l and another i-th symbol swap transformation on P^k on to S_{i-1}^m. Due to the adjacency preserving property of an i-th symbol swap transformation the two image path sections P^l and P^m in S_{i-1}^l and S_{i-1}^m respectively are of unit dilation. However, the image of the inter-substar edge (u^j, u^k) will

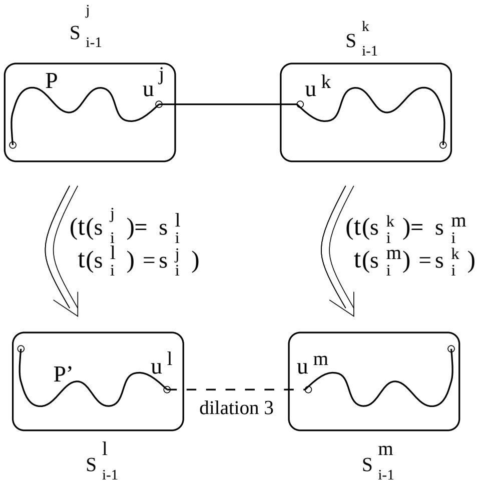


Figure 3.2: Effect of i-th symbol swap transformation on inter-substar edge.

not be a single edge as the transformations applied to the two nodes u^j and u^k are different. Let u^l and u^m be the images of u^j and u^k respectively. Therefore, $u^l = u^j(l_1\ i)$, where l_1 is the position of s^l_i in u^j, $u^m = u^k(l_2\ i)$, where l_2 is the position of s^m_i in u^k and $u^j = u^k(1\ i)$. Observe that, $u^l = u^k(1\ i)(l_1\ i) = u^m(l_2\ i)(1\ i)(l_1\ i) = u^m(1\ l_1)(1\ i)(1\ l_2) = u^m g_{l_1} g_i g_{l_2}$ Therefore, the i-th symbol swap image of the inter-substar edge (u^j, u^k) is a path of length three. $\square$

Lemma 3.3 *The distance between a node and its i-th symbol swap transformation images is at most three.*

Proof. Let u and v be a pair of i-th symbol swap transformation image nodes in sibling S_{i-1}s. Then they are related as $u = v(l\ i)$, where $1 \leq l < i$. When $l = 1$, $u = v(1\ i) = vg_i$, and therefore u and v are separated by a single edge path of dimension i. When $l > 1$, $u = v(1\ l)(1\ i)(1\ l)$ (or $v(1\ i)(1\ l)(1\ i)$) $= vg_l g_i g_l$ (or $vg_i g_l g_i$). Therefore, there are two alternative paths (shortest) of distance three between them, namely, *(l,i,l)* and *(i,l,i)*. Hence follows the lemma. $\square$

Theorem 3.3 *Let u^j be a node in S^j_{i-1} and let $u^k, 0 \leq k < i$ be the i-th symbol swap image node of u^j in the sibling substar S^k_{i-1}. The distance between any two nodes in the i-th symbol swap transformation image node set $IS = \{u^k\ |k = 0, 1, \ldots, i-1\}$ [2] of u^j is at most four.*

[2]Note that for $k = j$, $u^k = u^j$ and therefore $u^j \in IS$.

PROOF. Let u^p, u^q be two distinct nodes in IS.
Case-1: $p = j, q \neq j$.
By definition, u^p and u^q are i-th symbol swap transformation image nodes of each other. Therefore, according to Lemma 3.3 the distance between them is at most three.
Case-2: $p \neq j$ and $q \neq j$.
Let $u^p = u^j(l_p\ i)$ and $u^q = u^j(l_q\ i)$. Therefore, we can write $u^p = u^q(l_q\ i)(l_p\ i) = u^q(l_q\ l_p\ i)$.
For $l_p = 1,\ u^p = u^q(1\ i\ l_q) = u^q g_i g_{l_q}$. For $l_q = 1,\ u^p = u^q(1\ l_p\ i) = u^q g_{l_p} g_i$.
For $l_p \neq l_q \neq 1,\ u^p = u^q(l_q\ l_p\ i) = u^q g_{l_q} g_{l_p} g_i g_{l_q} = u^q g_{l_p} g_i g_{l_q} g_{l_p} = u^q g_i g_{l_q} g_{l_p} g_i$.

Therefore, there is a path of length two from u^q to u^p : (i, l_q) if $l_p = 1$ and (l_p, i) if $l_q = 1$.
There are three alternative paths of length four from u^q to u^p for $l_p \neq l_q \neq 1$.
They are: (l_q, l_p, i, l_q), (l_p, i, l_q, l_p) and (i, l_q, l_p, i).

Thus, the distance between any two nodes in the set IS is at most four. $\square$

3.3.2 Dimension swap transformation

With the symbol swap transformation one can map a graph in a PCDS substar to another PCDS substar. However, it is sometimes necessary to map a subgraph in a PCDS substar to an SCDS substar. To achieve this, we define another class of isomorphic transformations on the star graph called the *dimension swap transformation* based on the edge symmetric nature of the star graph.

DEFINITION 3.5 **Dimension substitution operator**
Let $< D_n >= \{2, 3, \ldots, n\}$*, the set of dimensions of* S_n*. Consider a one-to-one mapping* $t_d :< D_n > \rightarrow < D_n >$ *and define* $t_d(d_1, d_2, \ldots, d_k) \triangleq (t_d(d_1), t_d(d_2), \ldots, t_d(d_k))$*, where* $(d_1, d_2, \ldots, d_k)$ *implies a path consisting of a sequence of* k *edges of dimensions* $d_1, d_2, \ldots, d_k$ *respectively.* t_d *is referred to as a dimension swap transformation operator.*

Consider a dimension sequence ds for some path in S_n such that $ds = (ds_1, ds_2, \ldots, ds_l)$. From the above definition, it is clear that $t_d(ds) = (t_d(ds_1), t_d(ds_2), \ldots, t_d(ds_l))$.

Traversing an edge of dimension d_i from a node u is equivalent to operating with the generator g_{d_i} on u. Therefore, if $v = u(g_{d_1} g_{d_2} \cdots g_{d_k})$ we can loosely write, $v = u(d_1, d_2, \ldots, d_k)$ implying that node v is reachable from node u by traversing a path of k edges of dimensions d_1,d_2,...,d_k respectively. Moreover, if $v = u(ds)$, where $ds = d_1, d_2, \ldots, d_k$, then $u = v(d_k, \ldots, d_2, d_1)$. We shall use the notation $\overline{ds}$ to imply the reverse of the dimension sequence ds so that $u = v(\overline{ds})$. Note that $\overline{t_d(ds)} = t_d(\overline{ds})$.

Definition 3.6 **Dimension swap transformation**
Define dimension swap transformation $T_d : [u, t_d] : S_n \to S_n$ *such that given* $u(ds_v) = v$, *where* (ds_v) *is a path from node* u *to* v *in* S_n, $T_d(v) = u(t_d(ds_v))$.

It should be noted that the node u is invariant in the transformation and therefore will be referred to as the *pivot node* of the transformation. A dimension swap transformation image G' of a graph G in S_n consists of the nodes $T_d(v)$, for every node v in G with respect to a given pivot node u.

Definition 3.7 **Inverse dimension swap transformation**
For a given dimension swap transformation $T_d : [u, t_d]$ *the inverse dimension swap transformation is* $T'_d : [u, t'_d]$ *such that if* $t_d(d_j) = d_k$, *then* $t'_d(d_k) = d_j$.

It is easy to see that $t'_d(t_d(ds)) = ds$.

Lemma 3.4 *A dimension swap transformation is an isomorphism on the star graph.*

Proof. Consider two paths with dimension sequences ds_1 and ds_2 to a node v from node u. That is, $v = u(ds_1) = u(ds_2)$. Then $u(ds_1, \overline{ds_2}) = u$, or the dimension sequence $(ds_1, \overline{ds_2})$ forms a cycle. Now, given a dimension swap transformation $T_d : [u, t_d]$, u is invariant. Therefore, $T_d(u(ds_1, \overline{ds_2})) = u$. Or, $u(t_d(ds_1, \overline{ds_2})) = u$. That is, $t_d(ds_1, \overline{ds_2})$ is a cycle too.

Now we shall prove that the transformation is one-to-one. Let $v = u(ds_v)$ and $w = u(ds_w)$ be two nodes in S_n. For a given dimension swap transformation $T_d : [u, t_d]$, let us assume that $T_d(v) = T_d(w)$, that is, v and w map to the same node. Since $T_d(v) = u(t_d(ds_v))$ and $T_d(w) = u(t_d(ds_w))$, the composite dimension sequence $(t_d(ds_v), \overline{t_d(ds_w)})$, therefore, forms a cycle. Now if the inverse transform T'_d is applied we will get the dimension sequence $(ds_v, \overline{ds_w})$ as the image of the dimension sequence $(t_d(ds_v), \overline{t_d(ds_w)})$. However, since u is the invariant node in T'_d too the image $(ds_v, \overline{ds_w})$ must also form a cycle. That is, v and w must be the same node. Therefore, T_d is one-to-one.

We now need to prove that the transformation is adjacency preserving. Consider two nodes v and w separated by an edge of dimension $d_{v,w}$, so that, $w = v(d_{v,w})$. Let $v = u(ds_v)$ and $w = u(ds_w)$. Then $u(ds_v, d_{v,w}, \overline{ds_w}) = u$. That is, the composite dimension sequence $(ds_v, d_{v,w}, \overline{ds_w})$ forms a cycle. Therefore, for any given dimension swap transformation $T_d : [u, t_d]$, $T_d(u(ds_v, d_{v,w}, \overline{ds_w})) = u$. That is, $u(t_d(ds_v), t_d(d_{v,w}), t_d(\overline{ds_w})) = u$. Or, $u(t_d(ds_v)), (t_d(d_{v,w})) = u(t_d(ds_w))$. Or, $T_d(v)(t_d(d_{v,w})) = T_d(w)$. That is, the image of w is separated from the image of v by a single edge of dimension $t_d(d_{v,w})$ implying that a dimension swap transformation is adjacency preserving.

Thus, a dimension swap transformation is an isomorphism on the star graph. □

In a dimension substitution operator some of the dimensions in $< D_n >$ remain invariant. The non-invariant dimensions are said to *participate* in the substitution operation. As in the case of a symbol swap transformation in a dimension swap transformation too, it is not necessary to specify the invariant dimensions. Therefore, the invariant dimensions will be omitted in the specification of a dimension swap transformation.

LEMMA 3.5 *If all the dimensions participating in the transformation are $\leq i$, then the image of a subgraph of a PCDS S_i remains confined within the S_i.*

PROOF. Let G be a subgraph of an S_i. Naturally, all the edges of G are of dimensions less than or equal to i. Since all the participating dimensions in the transformation are less than or equal to i the dimensions of the edges of the transformed graph, say $T_d(G)$, will also be less than or equal to i. Therefore, $T_d(G)$ will also be confined within an S_i. Since the invariant node u is contained within the same S_i as that containing G, $T_d(G)$ will also be contained within the same S_i. □

Due to the edge symmetry property of the star graph we have the following corollary.

COROLLARY 3.1 *If $i-1$ dimensions remain invariant, the image of the portion of the subgraph confined within the S_i composed of the $i-1$ invariant dimensions and containing the invariant node u also remains invariant.*

LEMMA 3.6 *If all the dimensions participating in a dimension swap transformation applied on a subgraph G are higher than i, then the portions of the subgraph confined within a PCDS S_i will map to a PCDS S_i and the image of the portion of the subgraph confined within the PCDS S_i containing the pivot node u will remain invariant.*

PROOF. Let G be a subgraph of an S_n. The portion of G which is confined within a PCDS S_i must have edges with dimensions less than $i+1$. Therefore, the dimensions of none of the edges of such a portion of the graph will change in $T_d(G)$ and will remain confined within an S_i. Only for some of the edges linking such portions of the graph there will be dimension swap as they are of higher dimensions than i. Due to this, the portions of G confined within PCDS S_is will be shifted to other PCDS S_is. However, the portion of G confined within the PCDS S_i containing the invariant node u will not obviously be shifted to any other S_i and therefore it will remain invariant. □

3.4 The ranking schemes

In this section we present two node ranking schemes that assign integer labels to the nodes of the star. The first ranking scheme called the *hierarchical ranking scheme* gives a mixed radix generalized cube [BA84] structured view to the star graph. It allows order preserving communications in constant time in contrast to $O(n)$ time required in a node ranking scheme proposed earlier [AQS91] for the star graph. As a result many algorithms that use order preserving communications [AQS93] can be implemented more efficiently using this ranking scheme. The second ranking scheme called the *pipelined structured ranking scheme* gives a pipeline structured view to star graph. It provides a convenient framework for developing pipelined algorithms for the star graph.

For the ranking of the nodes of an S_n an $n-1$ digit address $A = a_{n-1}a_{n-2}\ldots a_2a_1$ is assigned to each node of S_n. The i-th digit a_i of the address of a node indicates the PCDS S_i within a PCDS S_{i+1} to which the node belongs. It distinguishes the $i+1$ S_is within the S_{i+1} by assuming one of the $i+1$ values from $\{0,1,2,\ldots,i\}$. For example, a_1 takes a value of 0 or 1 to distinguish the two nodes (a single node consists an 1-star) within a 2-star, a_2 assumes a value of 0,1,or 2 to distinguish the 2-star within a 3-star to which it belongs.

The assigning of the addresses to the nodes is based on the minimal bijective mapping between the node sets of two sibling substars defined by the i-th symbol swap transformation.

3.4.1 The hierarchical ranking scheme

In this scheme the address assigning process is recursive. The $n-1$ digit addresses of the nodes of an S_n are assigned by following the procedure Assign_Address_Hr given below.

ALGORITHM 3.1 **Assign_Address_Hr(S_n)**

if $n > 1$ *then*
For each node belonging to $S^j_{n-1}, j = 0,1,..n-1$, *assign* $a_{n-1} \leftarrow j$.
For the nodes of S^0_{n-1} *to assign the address digits* a_1 *to* a_{n-2}
perform Assign_Address_Hr(S_{n-1}).
For each node in $S^j_{n-1}, j = 1$ *to* $n-1$, *determine the n-th symbol swap transformation image in* S^0_{n-1} *and copy the digits* a_1 *to* a_{n-2} *of the image node.*

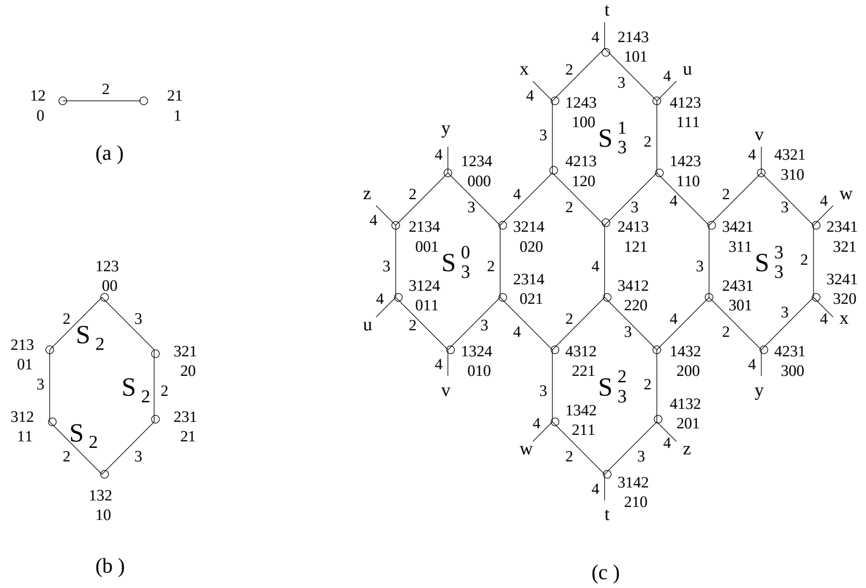


Figure 3.3: Address assignment to the nodes of star graph. The nodes are labeled with two numeric sequences; the upper one is the permutation representation and the lower one is the address assigned to it. (a) S_2, (b) S_3, (c) S_4.

Figure 3.3 illustrates this address assignment. Let us look at the scheme in the bottom-up manner. In an S_2 (Figure 3.3(a)) the node addresses consist of a single digit a_1. The node with permutation 12 constitutes S_1^0 and the node with permutation 21 constitutes S_1^1. Therefore, the address of node 12 is 0 and that of node 21 is 1. In S_3 (Figure 3.3(b)) the second digit a_2 of the addresses assigned to the nodes belonging to S_2^0, S_2^1, and S_2^2 are 0,1, and 2 respectively. The first digit a_1 for the nodes of S_2^0 is assigned value in the same way as in S_2. But, the first address digit for the S_2^1 and S_2^2 nodes is decided by the i-th symbol swap transformation image of these nodes in S_2^0. This recursive process is repeated for assigning addresses to the nodes of higher level stars.

Definition 3.8 **Rank of a node**
With the nodes of an S_n assigned addresses as described above, the rank of a node v in an S_i, $(2 \le i \le n)$, is computed as

$$r_i(v) \quad = \quad (i-1)! \times a_{i-1} + (i-2)! \times a_{i-2} + \ldots + 2! \times a_2 + a_1 \tag{3.1}$$

Two nodes in two sibling substars S_i^j and S_i^k are of the same rank if the first $i-1$ digits of the addresses are identical (or in other words, they are i-th symbol swap transformation images of the same node in S_i^0). For example, in Figure 3.3(c), the nodes with the addresses 010, 110, 210, and 310 in S_3^0, S_3^1, S_3^2, and S_3^3 respectively are of rank 2 while their ranks in the S_4 are 2, 8, 14, and 20 respectively.

As the i-th symbol in the identity permutation node is i the i-th digit of its address is $a_i = 0$, for $i = 1$ to $n-1$. Hence we have the following property.

PROPERTY 3.1 *The rank of the identity permutation node is zero.* □

Two nodes, u and v in the substars S_i^j and S_i^k respectively are said to have the same rank in S_is if and only if $r_i(u) = r_i(v)$.

DEFINITION 3.9 **Substar peer nodes**
A set of nodes of the same rank in sibling substars are said to be peer nodes in the substars.

It is easy to see that a set of $(i+1)$-th symbols swap transformation image nodes in sibling S_is of a node in S_i^0 constitutes a set of peer nodes in the S_is.

From the address assignment scheme it is clear that for the addresses of all the nodes of a PCDS S_i the $n-i$ most significant digits a_{n-1} to a_i are identical. These we shall use for the address of the PCDS S_is.

DEFINITION 3.10 **Address of a substar**
The address of a PCDS S_i, consists of the string of $n-i$ most significant digits (i.e. $a_{n-1}a_{n-2}\dots a_{i+1}a_i$) of the addresses of its nodes.

DEFINITION 3.11 **Rank of a substar**
The rank of a PCDS S_i within a PCDS S_k, $i < k \leq n$, is given by

$$r_k(S_i) = \frac{1}{i!}((k-1)! \times a_{k-1} + (k-2)! \times a_{k-2} + \dots + i! \times a_i) \tag{3.2}$$

It should be noted that the rank of a PCDS S_j within a PCDS S_{j+1} is a_j, the j-th digit in the addresses of its nodes. Therefore, we can denote a PCDS S_j within any PCDS S_{j+1} (not restricted to the one containing the identity node) as S_j^l where $l = a_j$. A given PCDS S_j^l in an S_{j+1}^k is an image of the S_j^l in the S_{j+1} containing the identity permutation node which is obtained through a series of i-th symbol swap transformations for $i = j+1$ to m, where an S_{m+1} is the smallest substar that contains the given S_j^l as well as the identity permutation node.

From the Property 3.1 it follows that the rank of a PCDS substar containing the identity permutation node is always zero.

DEFINITION 3.12 **0-th ranked substar**
A PCDS S_i containing the identity permutation node is referred to as the 0-th ranked S_i.

Two properties of the substar ranking scheme are-

PROPERTY 3.2 *A set of $i+1$ S_i with ranks in a range ρ to $\rho+i$, where $\rho \bmod (i+1) = 0$, forms a PCDS S_{i+1}.*

PROPERTY 3.3 *The ranks of the substars of S_i^j will be lower than that of the substars of S_i^k if $j < k$.*

From the address assignment procedure it is clear that a set of peer nodes in S_{i-1}s also constitutes a set of i-th symbol swap transformation image nodes. Therefore we have the following corollary to the Theorem 3.3 regarding the distance between any pair of peer nodes.

COROLLARY 3.2 *With the nodes of the star graph ranked according to the hierarchical ranking scheme there is a path of length at most four between any pair of peer nodes in two sibling substars.* □

The diameter of an S_2 is *one* an that of an S_3 is *three*. Thus, we have the following theorem.

THEOREM 3.4 *There exists a path of length at most one (three) between peer nodes of sibling S_1s (S_2s).*

FACT 3.1 *A communication between a set of peer nodes constitutes an order preserving communication. Therefore, with the nodes ranked using the hierarchical ranking scheme an order preserving communication on the star graph can be performed in constant time.* □

Peer node imaging

Given a subgraph G^j in a substar S_i^j the peer nodes of the nodes in G in a sibling substar S_i^k forms a subgraph G^k which is isomorphic to G^j. This is due to the fact that both G^j and G^k are (i+1)-th symbol swap transformation images of a subgraph G^0 in sibling substar S_i^0. This fact can be utilized for generating isomorphic copies of a subgraph in an S_i in sibling S_is through *peer node imaging*. Peer node imaging is a derivative of the i-th symbol swap transformation. The advantage of the peer node imaging technique is that the image of a subgraph G in S_i^j can be obtained in S_i^k by adding a constant $(k-j)i!$ to the rank of each node in G. This fact directly follows from the definitions of peer nodes and the ranking of substars. One obvious but important property of peer node imaging is that the distance between a node and its image is at most four. In that sense, the isomorphic copies of the graph generated using peer node imaging are *parallel*.

Ranking and unranking of a node

The algorithm Assign_Address_Hr computes the addresses of all the nodes in an S_n for the ranking. However it is useful to be able to compute the address of a node for its ranking given its permutation. Let u be a node in S^0_{n-1} with address $0a_{n-2}\ldots a_1$. Then the node $v = u \circ (i\ n)$ has address $a_{n-1}a_{n-2}\ldots a_1$ where $a_{n-1} = n - v_n$ and $i = pos(v_n, u) = pos(u_n, v)$. Similarly, if a node u has address $00a_{n-3}\ldots a_1$ then the node $v = u\circ(j\ \ (n-1))$ has the address $0a_{n-2}a_{n-3}\ldots a_1$ where $a_{n-2} = (n-1) - v_{n-1}$ and $j = pos(v_{n-1}, u) = pos(u_{n-1}, v)$. In general, if the address of a node u is $0\ldots 0a_{i-1}\ldots a_1$ then the node $v = u \circ (k\ \ (i+1))$ has the address $0\ldots 0a_ia_{i-1}\ldots a_1$ where $a_i = (i+1) - v_{i+1}$ and $k = pos(v_{i+1}, u) = pos(u_{i+1}, v)$. It should be noted that the node u with address $0\ldots 0a_{i-1}\ldots a_1$ belongs to the S_{i+1} containing the identity permutation node. As a result $u_j = j$, for $j > i$. Thus $k = pos(i+1, v)$. The address of a node, therefore, can be computed iteratively from its permutation using the algorithm Compute_Address_Hr given below.

ALGORITHM 3.2 **Compute_Address_Hr**(v, n)

Input : Permutation v of a node in S_n.
Output: Address $A = a_{n-1}a_{n-2}\ldots a_1$ of the node.
1. for $i = n-1$ to 1
  1.1 $a_i = i + 1 - v_{i+1}$
  1.2 $k = pos(i+1, v)$
  1.3 $u = v \circ (k\ \ (i+1))$
  1.4 $v = u$

Sometimes it is also necessary to compute the permutation label of a node from its rank, or equivalently, its address. Recollect that if the permutation of a node with address $0\ldots 0a_{i-1}\ldots a_1$ is u then the permutation of the node with address $0\ldots 0a_ia_{i-1}\ldots a_1$ is $u\circ(i+1\ \ pos(s, u))$ where $s = i+1-a_i$. Therefore, the permutation of any node with address $a_{n-1}a_{n-2}\ldots a_2a_1$ can be computed iteratively from the identity permutation node which has the address $0\ldots 0$ using the algorithm Compute_Permutation_Hr given below.

ALGORITHM 3.3 **Compute_Permutation_Hr**(A, n)

Input : Address $A = a_{n-1}a_{n-2}\ldots a_1$ *of a node in* S_n.
Output: Permutation u *of the node.*
1. Initialize $u =$ "$123\ldots n$".
2. for $i = 1$ *to* $n-1$
    2.1 $s = i + 1 - a_i$
    2.2 $u = u \circ (i+1 \;\; pos(s,u))$

The time complexity of the above algorithm is $O(n^2)$ as the step 2.2 requires a search for the symbol s in u. This search can be avoided and the complexity can be reduced to $O(n)$ based on the following observations.

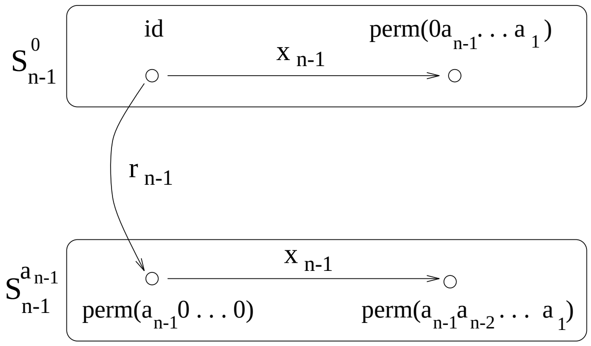


Figure 3.4: Computation of permutation of a node given the address $a_{n-1}a_{n-2}\ldots a_1$.

We can express the permutation of a node with address $a_{n-1}a_{n-2}\ldots a_1$ as

$$perm(a_{n-1}a_{n-2}\ldots a_1) = id \circ perm(a_{n-1}a_{n-2}\ldots a_1)$$

where id represents the identity permutation $12\ldots n$. This implies that $perm(a_{n-1}a_{n-2}\ldots a_1)$ is equivalent to a path from the identity permutation node to the node $perm(a_{n-1}a_{n-2}\ldots a_1)$. One such path can be through the n-th symbol swap image node of identity permutation node in the substar $S_{n-1}^{a_{n-1}}$, to which the node $perm(a_{n-1}a_{n-2}\ldots a_1)$ belongs. See Figure 3.4. The address of this image node is $a_{n-1}0\ldots 0$. That is

$$perm(a_{n-1}a_{n-2}\ldots a_1) = id \circ r_{n-1} \circ x_{n-1}$$

where $id \circ r_{n-1} = perm(a_{n-1}0\ldots 0)$ and $perm(a_{n-1}0\ldots 0) \circ x_{n-1} = perm(a_{n-1}a_{n-2}\ldots a_1)$. Since $perm(a_{n-1}0\ldots 0)$ is the n-th symbol swap image of the identity permutation node,

$$r_{n-1} = id \circ (n \;\; pos(n - a_{n-1}, id)) = (n \;\; n - a_{n-1})$$

This reduces the computation of $perm(a_{n-1}a_{n-2}\dots a_1)$ to computing x_{n-1}. The n-th symbol swap transformation image node of $perm(a_{n-1}a_{n-2}\dots a_1)$ in S^0_{n-1} is $perm(0a_{n-2}\dots a_1)$. Recall that a path from any node u to any other node v is equivalent to the relative permutation of u with respect to v. From the discussion of Lemma 3.1 we know that a symbol swap transformation (and therefore an n-th symbol swap transformation) is an isomorphism that also preserves paths, *i.e.* the image of a path comprises of the same sequence of dimensions as the original path. Therefore the relative permutation of the node $perm(a_{n-1}0\dots 0)$ with respect to the node $perm(a_{n-1}a_{n-2}\dots a_1)$ is identical to the relative permutation of the identity permutation node id with respect to the node $perm(0a_{n-2}\dots a_1)$. (See Figure 3.4.) In other words, since $perm(a_{n-1}0\dots 0)\circ x_{n-1} = perm(a_{n-1}a_{n-1}\dots a_1)$ we also have $id\circ x_{n-1} = perm(0a_{n-2}\dots a_1)$. Thus the problem reduces to computing the $perm(0a_{n-2}\dots a_1)$ which we can rewrite as

$$perm(0a_{n-2}\dots a_1) = id\circ r_{n-2}\circ x_{n-2}$$

where $r_{n-2} = ((n-1)\ \ (n-1-a_{n-2}))$, $id\circ x_{n-2} = perm(00a_{n-3}\dots a_1)$ and $x_{n-1} = r_{n-2}\circ x_{n-2}$ so that

$$perm(a_{n-1}a_{n-2}\dots a_1) = id\circ r_{n-1}\circ r_{n-2}\circ x_{n-2}$$

The above process of rewriting can be repeated $n-1$ times to finally arrive at the expression

$$perm(a_{n-1}a_{n-2}\dots a_1) = id\circ r_{n-1}\circ r_{n-2}\circ\cdots\circ r_i\cdots\circ r_1$$

where $r_i = ((i+1)\ \ (i+1-a_i))$. It is easy to see that the desired permutation can be computed directly from the address using this expression in $O(n)$ steps.

3.4.2 The pipeline structured ranking scheme

To view an S_n as a sequence of n S_{n-1}s forming a pipeline of n stages (Figure 3.5) the address assignment is done by the following the procedure Assign_Address_Pr given below.

ALGORITHM 3.4 **Assign_Address_Pr(S_n)**

1. *For each node belonging to* $S^j_{n-1}, j = 0,1,..i$ *assign* $a_{n-1}\leftarrow j$.
2. *For the nodes of* S^0_{n-1} *to assign the address digits* a_1 *to* a_{n-2} *perform*
   Assign_Address_Hr(S_{n-1}).
3. *For each node in* $S^j_{n-1}, j = 1$ *to* n, *determine the n-th symbol swap transformation*
   image in S^{j-1}_{n-1} *and copy the digits* a_1 *to* a_{n-2} *of the image node.*

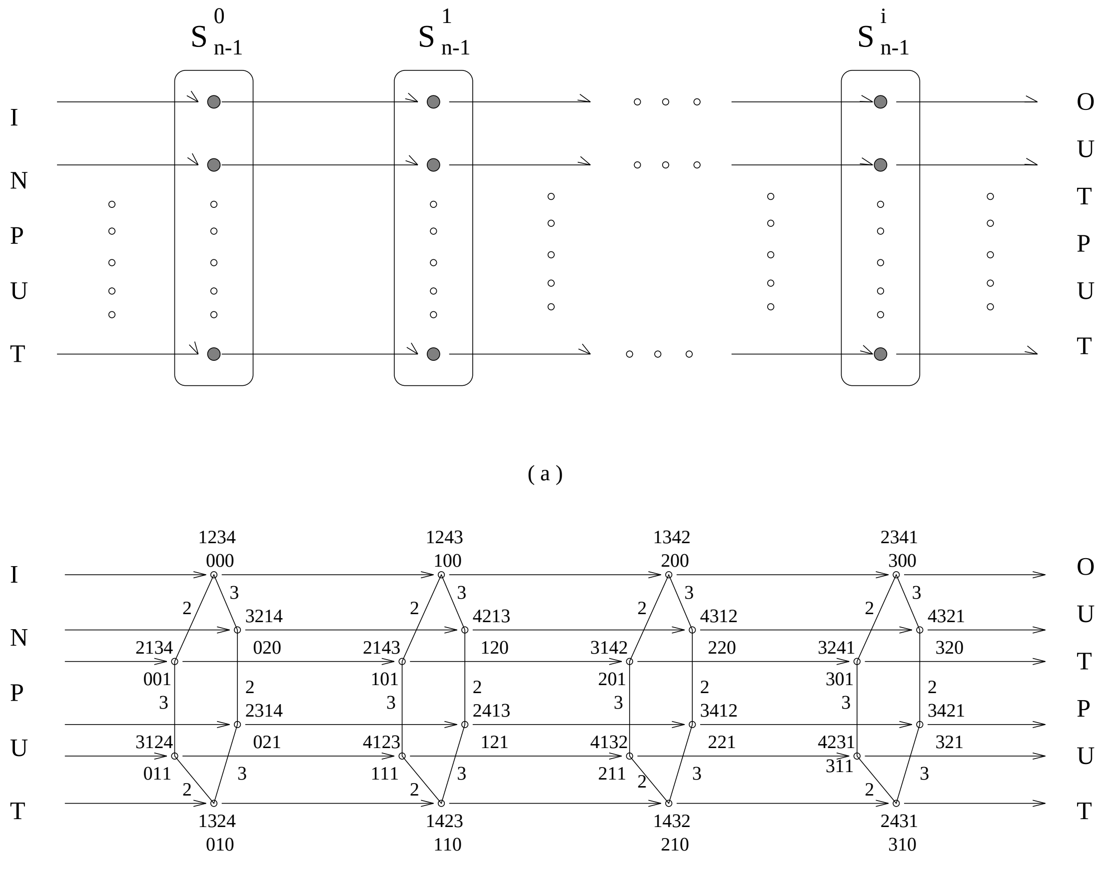


Figure 3.5: Pipeline organization of star graph. (a) S_n, (b) S_4 with address assignment for pipeline structured ranking.

After assigning the addresses to the nodes the rank of a node can be computed in the same way as in the case of the hierarchical scheme using equation 3.1. In an S_n, with the nodes ranked using this scheme, for any S^j_{n-1}, S^{j-1}_{n-1} is its predecessor and S^{j+1}_{n-1} is its successor. It should be noted that the ranking of the nodes of the substars (S_{n-1}s) is done using the hierarchical scheme. From the address assignment procedure it is clear that a pair of peer nodes in S^j_{n-1} and S^{j+1}_{n-1} are n-th symbol swap transformation images of each other. Therefore, we have the following corollary to the Lemma 3.3, regarding the distance between such a pair of peer nodes.

Corollary 3.3 *When the nodes of an n-star are ranked using the pipeline structured ranking scheme, there exists a path from any node in S^j_{n-1} to its peer node in S^{j+1}_{n-1} (or S^{j-1}_{n-1}) having a length of at most three.* □

Therefore, the peer nodes in adjacent (successor and predecessor) S_{n-1}s can be reached from each other within three communication steps and thus, forms a pipeline of S_{n-1}s of n stages (Figures 3.5(a) and Figure 3.5(b)).

3.5 Domain decomposition of the star graph

In this section we introduce a new decomposition scheme for the star graph. It is based on the concepts of (i,k)-centre and the domain of a node and referred to as a *domain decomposition*. This decomposition provides a good understanding of the communication structure of the star graph and allows us to define and analyze several elegant and useful communication paradigms on the star graph. Some of the ideas presented in this section will also be useful in our embedding schemes.

Consider two sibling substars S_i^j and S_i^k. Both contain $i!$ nodes. However, between the nodes of these two distinct substars only $(i-1)!$ direct edges are present. Note that these are dimension $i+1$ edges. On the average i nodes in an S_i share such a dimension $i+1$ edge in the communication between nodes of two sibling S_is and the end points of such edges act as a sort of gateway nodes between the two sibling substars.

DEFINITION 3.13 **(i,k)-centre**
A node of S_i^j is an (i,k)-centre if it is directly connected to a node in sibling substar S_i^k by an edge of dimension $i+1$. The node in S_i^k is called the corresponding (i,j)-centre.

For example, in Figure 3.6 the node 2413 in S_3^1 is a (3,2)-centre. The node 3412 is the corresponding (3,1)-centre in S_3^1.

PROPOSITION 3.1 *a. The first symbol in the permutation associated with an (i,k)-centre in S_i^j is the (i+1)-th symbol (i.e. s_{i+1}^k) in the permutations associated with the nodes of S_i^k.*
b. There are (i-1)! (i,k)-centres in an S_i^j.
c. Each node in S_i^j is an (i,k)-centre for some value of $k \in \{0,1,\ldots,i\}\backslash j$.
d. The distance between any two (i,k)-centres in S_i^j is at-least three edges.

PROOF. (of 3.1a:)
Two nodes joined by a dimension $i+1$ edge implies that their associated permutations are related by the transposition of their 1-st and (i+1)-th symbols. If u is an (i,k)-centre in S_i^j and v is the node in S_i^k connected to u, then the 1-st symbol of u is the (i+1)-th symbol of v. Since the (i+1)-th symbol of all the nodes of S_i^k are identical, the Proposition 3.1(a) is follows. $\square$

PROOF. (of 3.1b:)
There are $i!$ nodes in S_i^j, one associated with each permutation of first i symbols of their permutation representations. Out of these $i!$ nodes those having s_{i+1}^k as their first symbol are (i,k)-centres. There are $(i-1)!$ such nodes in S_i^j, one associated with each of the permutations of the rest of $i-1$ symbols. □

PROOF. (of 3.1c:)
The property follows from the decomposition of an S_{i+1} into its substars S_is. The dimension $i+1$ edge from any node in an S_i^j leads to a node in a sibling S_i^k for some value of $k \in \{0, 1, \ldots, i\}\backslash j$. □

PROOF. (of 3.1d:)
The permutations associated with any two nodes of S_n must differ in two symbol positions. The first symbol in the permutations associated with all the (i,k)-centres in S_i^j is identical. Therefore, two (i,k)-centres in S_i^k must differ in at-least two positions other than the first. Let u and v be two (i,k)-centres in S_i^j and their permutations differ in positions l and m such that $l \neq m \neq 1$;
i.e. $v = u(l\ m)\ = u(1\ l)(1\ m)(1\ l)\ = ug_l g_m g_l$
Thus, any two (i,k)-centre nodes u and v in S_i^k must be separated by at-least three edges. □

PROPOSITION 3.2 *All the S_{i-2}s, except one, in a substar S_{i-1}^j have $(i-1, k)$-centres for $k \in \{0, 1, \ldots, i-1\}\backslash j$.*

PROOF. The i-th fixed symbol s_i^k of S_{i-1}^k is a permuting symbol in the sibling substars S_{i-1}^j, $j \in \{0, 1, \ldots, i-1\}\backslash k$. However, s_i^k will be a fixed symbol ($(i-1)$-th symbol) in one of the S_{i-2}s in S_{i-1}^j. In the remaining S_{i-2}s in the S_{i-1}^j s_i^k will be a permuting symbol. Therefore in all these S_{i-2}s in the S_{i-1}^j there will be nodes with s_i^k as the first symbol which will be $(i-1, k)$-centre nodes. Thus we have the proposition. □

The above proposition will be useful in our cycle embedding schemes.

DEFINITION 3.14 **Domain of a node**
Let u be a node in S_i^j. The set of nodes in S_i^j consisting of the immediate neighbours of u and itself is defined as the domain of u in S_i. The node u is called the centre of its domain.

In other words, the domain of a node in S_i^j consists of its immediate neighbour nodes in S_i^j and itself. For example, in Figure 3.6, the domain of the node 2314 in S_3 is the three member set {2314, 3214, 1324}

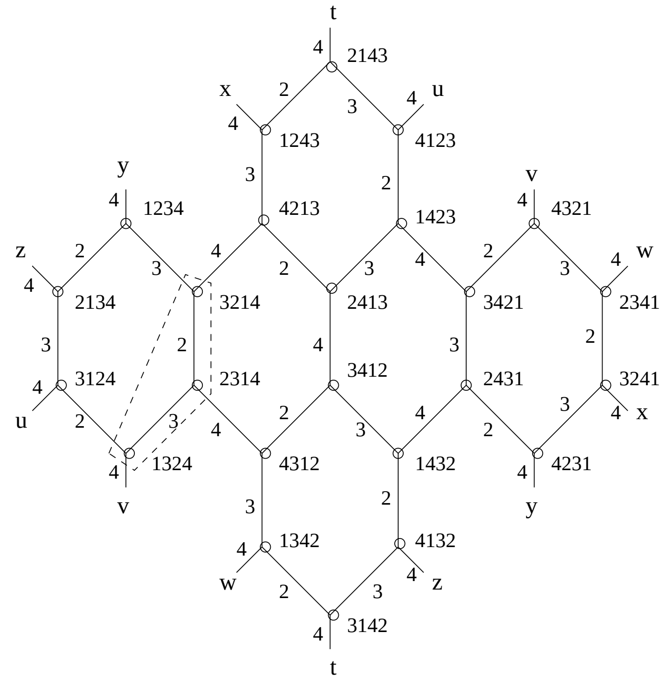


Figure 3.6: Domain of 2314 in S_3 is $\{$2314,3214,1324$\}$.

PROPOSITION 3.3 a. *The size of the domain of a node in S_i is i.*
b. *The domains of the $(i-1)!$ (i,k)-centres in S_i^j are disjoint.*
c. *The union of the domains of all the (i,k)-centres of S_i^j covers the node set of S_i^j.*

PROOF. (of 3.3a:)
In an S_i a node has $i-1$ neighbours, one along each of the $i-1$ dimensions 2 to i. Hence, an (i,k)-centre has $i-1$ neighbours in S_i^j. Including itself, therefore, an (i,k)-centre has i nodes in its domain. $\Box$

PROOF. (of 3.3b:)
Neither an (i,k)-centre nor its neighbours can be in the domain of another (i,k)-centre as the minimum distance between any two (i,k)-centres is three. Hence, the domains of the (i,k)-centres must be disjoint. $\Box$

PROOF. (of 3.3c:)
There are (i-1)! (i,k)-centres in S_i^j each having i nodes in its domain and the domains are disjoint. Thus, their union has $i(i-1)! = i!$ nodes, i.e. the entire set of nodes of S_i^j. $\Box$

For example, in Figure 3.6, the S_3^2 has two (3,0)-centres, 4312 and 4132. The domain of 4312 in S_3 consists of {4312, 3412, 1342} and the domain of 4132 in S_3 consists of {4132, 1432, 3142}. Clearly, the two domains are disjoint and their union contains the entire set of nodes of S_3^2. In general, the domains of the (i,k)-centres in $S_i^j, j \neq k$, will constitute a decomposition of S_i^js. In this decomposition, the nodes in S_i^k are not included in any of the domains. However, if we consider the domain of an (i,k)-centre in the S_{i+1}, it will also include the corresponding (i,j)-centre node. As a result the domains of the (i,k)-centre nodes in an S_{i+1} constitutes a proper decomposition of the S_{i+1}.

DEFINITION 3.15 **Domain decomposition**
The decomposition of an S_{i+1} consisting of the domains of the (i,k)-centres, $0 \leq k \leq i$, is referred to as a domain decomposition.

PROPOSITION 3.4 *The domain of a node in S_i^j contains one (i,k)-centre for each $k \in \{0, 1, \ldots, i\}\backslash j$.*

PROOF.
Let u be a node in S_i^j. A neighbour v of u is given by $v = u(1\ l), 2 \leq l \leq i$. Thus, the first symbol of each of the neighbours of u is different and is different from the first symbol of u itself. Therefore the dimension $i+1$ edge from each of the i nodes in the domain of u leads to a different sibling S_i. Hence the domain of u contains an (i,k)-centre for each $k \in \{0, 1, \ldots, i\}\backslash j$.
$\square$

We shall use the idea of *adjacent domains* to define and analyze the *adjacent domain communication* paradigm for star graph based networks.

DEFINITION 3.16 **Adjacent domains**
Consider two sibling substars S_i^j and S_i^k. Let u be an (i,k)-centre in S_i^j and v be its corresponding (i,j)-centre in S_i^k. The set of nodes in the domain of u in S_i^j and the set of nodes in the domain of v in S_i^k are (i+1)-th symbol swap images of each other. The domains of u and v are said to be a pair of adjacent domains.

From the definition it is evident that the centres of two adjacent domains are separated by a single edge path.

In the following we discuss some properties and concepts that concern peer nodes. These will be useful in our analysis of the *hierarchical order preserving communication* paradigm and in some embedding schemes.

PROPOSITION 3.5 *A complete set of peer nodes (in sibling S_is) contains an (i,0)-centre which* *is also called the* peer 0-centre. *If that (i,0)-centre belongs to S_i^k, then all other peer nodes in the set are (i,k)-centres.*

PROOF. Let u be a node in S_i^0. The set of peer nodes of u consists of $PS = \{u, u(l\ i+1)|l = 1, 2, \ldots, i\}$. Let v be the peer node such that $v = u(1\ i+1)$. According to the definition v is an (i,0)-centre. Let v belong to S_i^k. Obviously, u is an (i,k)-centre in S_i^0. Consider a node $w \in PS$ and $w \neq u \neq v$. According to definition of peer nodes $w = u(l\ i+1)$, where l is the position of s_{i+1}^0 in w, $2 \leq l < i+1$. Therefore, $v = w(l\ i+1)(1\ i+1) = w(1\ i+1)(1\ l)$. We can write $v = x(1\ l)$, where $x = w(1\ i+1)$. Obviously, x belongs to the domain of v in S_i^k and therefore w is an (i,k)-centre. □

For example, in the set of peer nodes in S_4s $\{24135, 25134, 24153, 54132, 24531\}$, 54132 is a (4,0)-centre and it belongs to S_4^3. Therefore, 54132 is the peer 0-centre in the set. All the other node in the set are (4,3)-centres.

PROPOSITION 3.6 *Let v in S_i^k be the (i,0)-centre of a set of peer nodes in S_is. Let D_v be its domain in S_i^k. Let w be the peer of v in S_i^j and let D_w be its domain in S_i^j. Then the peer in S_i^j of each node in D_v belongs to D_w. The peer node pairs in the two domains are reachable along the same dimension edge from the corresponding domain centres.*

PROOF. Let u be an (i,k)-centre in S_i^j and v be its peer 0-centre in S_i^k. Then we know that, $v = u(1\ i+1)(l\ i+1) = u(l\ 1\ i+1)$, where l is the position of s_{i+1}^0 in u.
Let x be a node in the domain of the (i,k)-centre u and y be its peer node in S_i^k.
Then, $x = u(1\ m)$ and $u = x(1\ m)$, where m is the position of s_{i+1}^k in x.
Therefore, $v = x(1\ m)(l\ 1\ i+1) = x(1\ m)(1\ i+1)(1\ l)$.
As per definition of peer nodes in the hierarchical ranking scheme, $y = x(l\ m\ i+1)$.
We can rewrite, $y = x(1\ m)(1\ i+1)(1\ l)(1\ m) = v(1\ m)$.
That is, y is a neighbour of v and hence belongs to the domain of the peer 0-centre v. Moreover, x and y are reachable along the dimension m edge from their domain centres u and v respectively, where m is the position of s_{i+1}^k in x and also the position of s_{i+1}^0 in y. □

DEFINITION 3.17 **Peer domains**
The domain of an (i,0)-centre in S_i^k and the domain of its peer node in S_i^j, which is an (i,k)-centre, are referred to as a pair of peer domains.

The domain of the (4,0)-centre 54132 in S_4^3 is {54132, 45132, 34152, 14532}. Its peer node in S_4^1 is 25134 which is a (4,3)-centre and its domain in S_4^1 is {25134, 51234, 35124, 15234}. These two domains consist a pair of peer domains.

PROPOSITION 3.7 *The distance between a node in S_i^j ($j \neq 0$) to its peer 0-centre in S_i^k is two, if $j \neq 0$, and 1, if $j = 0$. When $j \neq 0$, the intermediate node is an (i,j)-centre in S_i^k.*

PROOF. Let u be a node in S_i^j and v be its peer 0-centre in S_i^k.
Then, $v = u(l\ 1\ i+1)$, where l is the position of s_{i+1}^0 in u.
We can rewrite, $v = u(1\ i+1)(1\ l) = ug_{i+1}g_l$.
For j=0, l=1 and $v = u(1\ i+1) = ug_{i+1}$.
Hence, the distance between u and v is two if $k \neq 0$, and 1 if $j = 0$.
For $j \neq 0$, let the intermediate node between u and v be x.
According to Proposition 3.6, u is an (i,k)-centre. Therefore, x must be in S_i^k and hence an (i,j)-centre. □

3.6 Summary

We have introduced several new concepts which are used in developing the various schemes and algorithms presented in this thesis. We have introduced the concept of *composition dimension sequence* (CDS) to specify a hierarchical decomposition of the star graph. We defined a *primary* composition dimension sequence (PCDS) to consist of the natural monotonic sequence of dimensions and the *secondary* composition dimension sequence (SCDS) to consist of any other sequence of dimensions.

Two classes of isomorphic transformations on the star graph called *symbol swap transformation* and *dimension swap transformation* have been defined and characterized. We have also defined a special class of symbol swap transformation called the *i-th symbol swap transformation*. Most of the schemes presented in this thesis are based on these transformations.

We have presented a ranking scheme called the *hierarchical ranking scheme* for the star graph that ranks its nodes and the substars. The scheme preserves the hierarchical structure of the star and allows order preserving communication in constant time. The scheme also leads to an isomorphic transformation called the *peer node imaging*. We have also presented another ranking scheme called the *pipeline structured ranking scheme* which gives a pipelined view to the star graph and provides a framework for developing pipelined algorithms for the star graph.

We have developed another decomposition scheme based on the concepts of *(i,k)-centre* and the *domain of a node* introduced in this thesis. This decomposition scheme is useful in defining and analyzing several communication paradigms for the star graph topology. The properties of (i,k)-centre and the domain of a node have been characterized. In the process we have also defined several other useful terms such as *adjacent domains*, *peer 0-centre* and *peer domains*.

Chapter 4

Communication Paradigms

The concepts introduced in the previous chapter give us four different communication paradigms on the star graph. These communication paradigms provide the framework for development of efficient algorithms for the star graph for a wide variety of problems. They are also relevant to embeddings of several interconnection structures on the star graph. The four communication paradigms we present here are– i) the hierarchical order preserving communication (HOPC), ii) the adjacent-domain communication (ADC), 3. the transmit-and-domain-distribute (TDD), and 4. the domain-distribute-and-transmit (DDT).

For each communication paradigms alternative routing strategies are possible. The important parameters of these paradigms, such as, congestion, path length and path distribution are influenced by the routing. Algorithms that use these communication paradigms inherit this influence. In the following we discuss these communication paradigms and the alternative routing strategies and characterize them. The results can be directly used in analyzing the communication cost of the algorithms.

An order preserving communication (OPC) in a star graph based interconnection network consists of a communication between a set of peer nodes in sibling substars. When the substars are defined according to the hierarchical ranking scheme, we have a hierarchical OPC and when the peer nodes are defined according to the pipelined structured ranking scheme, we have a pipelined OPC. The pipelined OPC is a special case of the ADC paradigm. The hierarchical ranking scheme provides a framework for efficient implementation of order preserving ascend/descend algorithms on the star. This class includes many important algorithms such as the *fast fourier transform* and the *bitonic sort*. Several fundamental algorithms like translation, concentration, distribution etc. which are required computational geometry [AQS93], for example, can also be implemented easily with HOPC. The hierarchical ranking scheme also leads to dilation four embeddings of several important interconnection networks such as the torus, pyramid, hypercube, cube connected cycle etc.

From the definition of adjacent domains (in Section 3.5) we know that the node sets in two adjacent domains in sibling S_is are (i+1)-th symbol swap transformation images of each other. The communication between (i+1)-th symbol swap transformation image nodes in adjacent-domains in sibling S_i is referred to as the *adjacent domain communication.* When the set of S_is in an S_{i+1} are logically placed in a linear array and the communication occurs between adjacent S_is through the adjacent-domain communication paradigm, we refer to it as the *pipelined OPC*. The communications, in this case, occur between identically ranked nodes in the substars as defined by the pipeline structured ranking scheme. When all the pairs of adjacent domains in a pair of sibling substars communicate through the adjacent-domain communication paradigm it takes the *group-copy* form as defined by Akl et al. [AQ93] for implementation of non-order preserving ascend/descend algorithms such as the minimum spanning forest [AQ93] on the star. The pipelined OPC provides a framework for implementation of pipelined algorithms such as the two dimensional FFT on the star. Thus the ADC is relevant to implementations of pipelined algorithms and non-order preserving ascend/descend algorithms. The dilation three embeddings of $x_1 \times x_2 \times ... \times x_d$ grids, where $x_1 \times x_2 \times ... \times x_d = n!$, on an S_n [NSK90, RWY93] can be thought to be a pipelining of the substars. Therefore this paradigm is also relevant to the dilation three embeddings of grids on the star graph.

The transmit-and-domain-distribute (TDD) paradigm and the domain-distribute-and-transmit (DDT) paradigm provide simple methods for data distribution in global communication operations such as broadcasting, gossiping, scattering etc. Through reversal of data paths the paradigm can also be used for global data collection operations such as single node accumulation. The TDD paradigm can also be used for of an optimal congestion-free parallel prefix algorithm on the star graph based interconnection networks.

4.1 Hierarchical order preserving communication

The order preserving communication (OPC) consists of a communication between a set of peer nodes in sibling substars. In HOPC the peer nodes are defined by the hierarchical ranking scheme. There are three different alternative set of paths between a pair of peer nodes. Based on these alternative path sets we define three different HOPC schemes which can be classified into two categories. The first category is called the *domain collection and domain distribution HOPC* or the *confined routing HOPC* and the second category is called the *peer 0-centre routed HOPC* or the *third substar routed HOPC*. The first category is called confined routing as the intermediate nodes on the path in a communication between a pair of peer nodes are confined within the two sibling substars containing the communicating peer node pair. On the other hand, in the second category the communication path between a pair of peer nodes uses a third substar not containing the two peer nodes. The first category again consists of two types: the *0-domain collection HOPC* and the *0-domain distribution HOPC*.

4.1.1 Confined routing or domain collection and domain distribution HOPC

From the Proposition 3.5 in the previous chapter we know that the domain of an (i,k)-centre in S_i^j and the domain of its peer 0-centre in S_i^k form a pair of peer domains. From Proposition 3.6 it is known that the centres of these two domains are separated by a path of length 1 or 2. It is possible that the data generated in the nodes of one of these domains can be collected at the centre of the domain and then these can be relayed to the centre of the peer domain which can then distribute the data to the destination peer nodes. Refer to Figure 4.1 for illustrations. Since the distance from the centre to any node in a domain is at most one (by definition of a domain) and the domain centres are at a distance of at most two (Proposition 3.6), the total distance traveled by any of the data elements, in the process, is at most four.

Thus, the HOPC schemes on the star with the domain collection and domain distribution paradigm have the following common features:

1. There is a collection centre in a collection domain and a distribution centre in a distribution domain.

2. The HOPC takes place in three steps: the collection step, the relay step and the distribution step. In the collection step, a node in the collection domain hands over its data to the collection centre. The collection centre transmits the data to the distribution centre in the relay step. In the distribution step the distribution centre hands over the data to the destination peer node in the distribution domain.

Moreover, it is known that the peer nodes are reachable along the same dimension edge from their respective domain centres (Proposition 3.6). This fact can be used to simplify the routing process as follows. On arrival of a data packet from a source node, the collection centre adds a tag to it marking it with its arrival edge dimension. The distribution centre uses this tag information for handing over of the data packet to the destination peer node.

Let us assume that all the nodes in S_i^j communicate with respective peer nodes in S_i^k using a domain-collect-and-domain-distribute HOPC scheme. We can cluster the nodes of S_i^j in

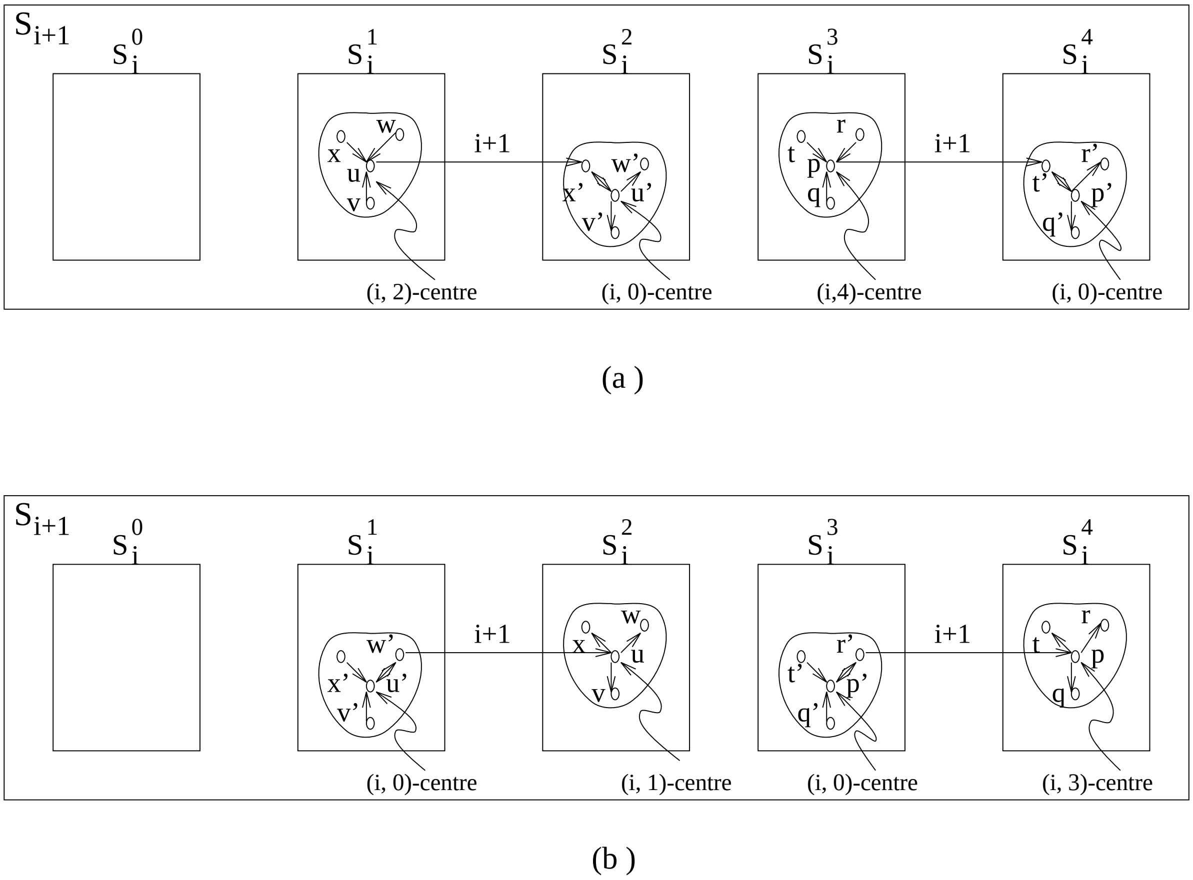


Figure 4.1: HOPC with confined routing. (a) 0-domain distribution scheme, (b) 0-domain collection scheme.

---

the domains of the (i,k)-centre nodes. In that case the peer clusters in S_i^k will consist of the domains of the respective peer 0-centres. As a result, the domains of the (i,k)-centre nodes in S_i^j will act as the collection domains and the domains of the (i,0)-centres in S_i^k will act as the distribution domains. Since, in this case, the domains of the peer 0-centres act as the distribution domains it is referred to as the *0-domain distribution* scheme. On the other hand, if we cluster the nodes of S_i^j in the domains of the (i,0)-centre nodes the peer clusters in S_i^k will consist of the domains of the respective peer (i,j)-centres. As a result, the domains of the (i,0)-centre nodes in S_i^j will act as the collection domains and the domains of the (i,j)-centres in S_i^k will act as the distribution domains. Since in this case the domains of the peer 0-centres act as the collection domains it is referred to as the *0-domain collection* scheme. It should be noted that since an S_i^0 can not have an (i,0)-centre the 0-domain collection scheme is not applicable for a source node in S_i^0.

Edge congestion

It can be seen that in both the above routing schemes all the i nodes of the domain of a collection centre use the same edges during the relay stage while communicating with the peer nodes in the same sibling substar. Hence there is a congestion of $O(i)$ over the path between the collection centre and the distribution centre.

THEOREM 4.1 *The worst case congestion in HOPC on S_n using confined routing is $O(n)$.*

Hot spot will occur on these edges when more than one node of a collection domain simultaneously communicate with their peer nodes in the same sibling substar. These hot spots can be eliminated by making use of the fact that the collection and distribution operations are performed by the same pair of nodes. A collection centre can combine the data originating simultaneously in the nodes of its domain into a single packet along with appropriate markers regarding their arrival dimensions. The distribution centre can break-up the combined packets into the individual data packets and distribute them along the marked dimensions.

4.1.2 Third substar or peer 0-centre routed HOPC

From Proposition 3.5 we know that each set of peer nodes contains an (i,0)-centre which is also called the peer 0-centre of the set. If a peer 0-centre belongs to S_i^k its peer nodes in the sibling S_is are (i,k)-centres in the respective S_is. Figure 4.2 depicts the paths from a peer 0-centre to its peer nodes in sibling S_is. From the figure it is clear that the peer 0-centre can act as a linkage node for communication between a set of peer nodes. The distance between any pair of peer nodes via the peer 0-centre is at most four.

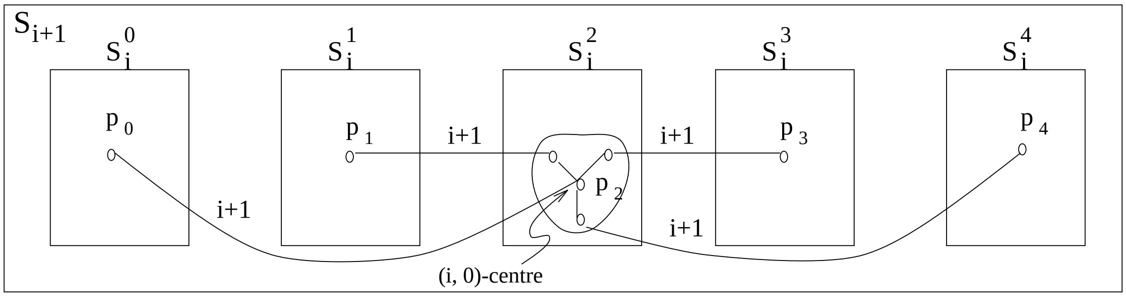


Figure 4.2: Peer 0-centre routed HOPC.

The communication between a pair of peer nodes, in this scheme, consists of two phases. In phase-1, the source node sends the packet to the peer 0-centre within two steps. The first step consists of a transmission along dimension $i+1$ and the second step a transmission to the neighbouring (i,0)-centre. In phase-2, the peer 0-centre relays the packet to the destination node in S_i^k again within two steps. The first step consists of a transmission to the neighbouring (i,k)-centre and the second step a transmission along dimension $i+1$ edge. When an (i,0)-centre is the destination node, the communication consists of only phase-1. On the other hand, if the source node is an (i,0)-centre the communication consists of only phase-2.

Edge congestion

We shall consider a set of situations that may occur in the various algorithms or embedding schemes that use the HOPC paradigm. In all such cases it is assumed that the communication with the HOPC paradigm is not performed at more than one substar level at a time. Further we shall assume that if an edge is shared by two paths in which the communication occur in opposite directions, the edge is not considered to suffer congestion.

One common type of communication operation commonly studied is the data permutation operation. Consider a data permutation operation in which nodes within peer node sets interchange data. Let us call it a *peer permutation* operation.

THEOREM 4.2 *A synchronous peer permutation operation, if performed in two phases using peer 0-centre routed HOPC, is free of edge-congestion.*

PROOF. Consider the Figure 4.3 which depicts a set of peer 0-centre routed HOPC paths between S_i peer nodes. In the figure, p_2, q_2, r_2 and t_2 are the nodes in the domain of the node q_2 in S_i^2. The S_i peer node sets of these four nodes are $P = \{p_0, p_1, p_2, p_3, p_4\}$, $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $R = \{r_0, r_1, r_2, r_3, r_4\}$ and $T = \{t_0, t_1, t_2, t_3, t_4\}$ respectively. The peer 0-centres of these sets are p_1, q_2, r_3, and t_4. The fact that the peer 0-centres of the nodes in the domain of q_2 are distributed over the sibling substars follows from Proposition 3.3(d)[1] and Proposition 3.5 [2]. It can be seen that the domain nodes p_2, r_2, t_2 are intermediate nodes in the paths between the nodes in the peer node set Q while each of these belong to a distinct peer node set. Due to this, the dimension $i+1$ edges from these nodes participate in the communication between peer nodes in two sets each. However, these dimension $i+1$ edges appear in opposite directions in the two paths in a phase of the peer 0-centre routed HOPC. For example, consider simultaneous peer permutation in the two peer node sets P and Q. The edge (q_1, p_2) is common to paths in the communication of both the sets. q_1 is the intermediate node in the path from p_2 to its peer 0-centre p_1 and p_2 is the intermediate node in the path from q_1 to its peer 0-centre q_2. In the first phase of peer 0-centre routed HOPC the data

[1]Each node in the domain of a node in S_i is an (i,k)-centre for a distinct value of k.

[2]The peer 0-centre node of an (i,k)-centre resides in S_i^k.

from the peer nodes are collected in the peer 0-centre. Clearly, the data from the two nodes q_1 and p_2 will travel in opposite directions over the edge (q_1, p_2) in this phase. In the second phase, the peer 0-centres distribute the data to the peer nodes. Therefore in this phase too the data will travel in opposite directions over the edge. The edge (q_1, p_2) does not lie in the communication path of any other set of peer nodes. Therefore it will be congestion-free in the two phase communication. The same will happen for all other dimension $i + 1$ edges. The edges of other dimensions will not participate in the path of more than one set of peer nodes. Therefore, the communication will be free of edge-congestion. □

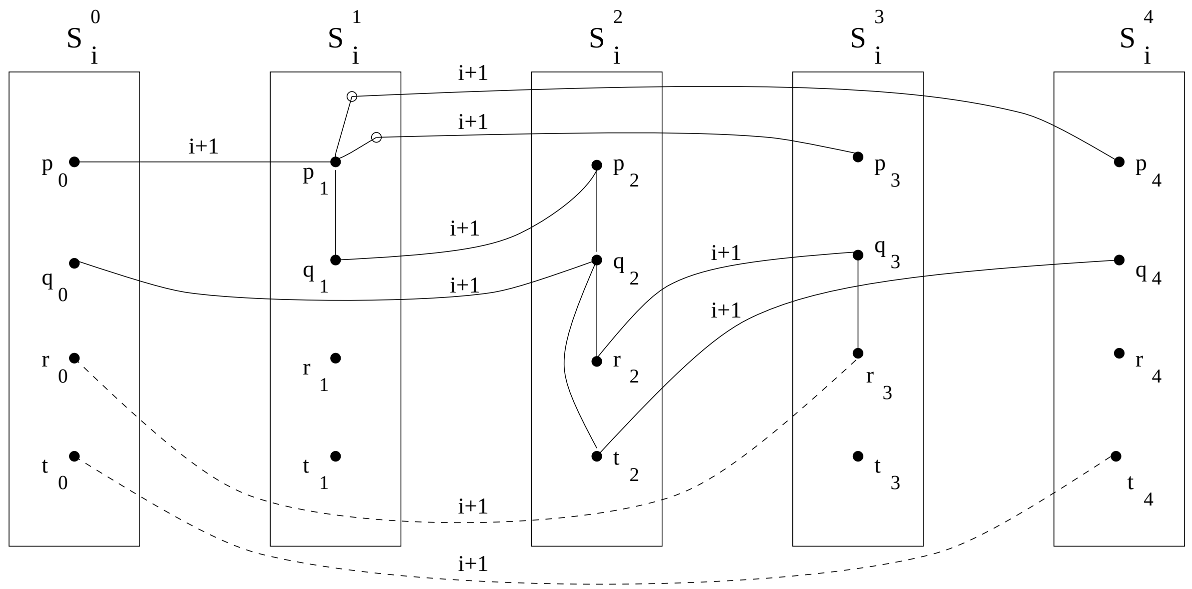


Figure 4.3: Edge congestion in peer 0-centre routed HOPC.

From the above proof discussion, it is however clear that if the two phase restriction is not enforced the edge congestion in peer 0-centre routed HOPC will be two. A special case of the peer permutation through HOPC occurs when the sibling substars are arranged in a linear or a ring formation in which the peer nodes in adjacent substars communicate. This arrangement will be used in our torus and pyramid embedding schemes. Obviously the edge-congestion in the communication will be one or two depending upon whether the two phase restriction is enforced or not.

THEOREM 4.3 *Consider an all-to-all peer broadcast operation in which each node in an S_{i+1} simultaneously sends a message to all its S_i peer nodes. The operation can be performed without edge-congestion.*

PROOF. Consider performing the all-to-all broadcast operation in two phases. In the first phase each of the peer nodes in a set transmits its message to the peer 0-centre. The peer 0-centre combines these messages into a single packet and sends a copy of it to each of the peer nodes in the second phase. The first phase will be free of edge-congestion because each node will use a distinct dimension $i + 1$ edge and the domains of the peer 0-centre nodes are distinct. Similarly, the second phase will also be free of edge-congestion. □

Node congestion

Consider a situation in which each of the nodes in an S_i peer node set communicate simultaneously with one or more of its peer nodes. See Figure 4.2. In this case the peer 0-centre will receive $O(i)$ packets simultaneously to be relayed to the desired destination peer node. Such a situation leads to a node congestion of $O(i)$ at the peer 0-centre node.

THEOREM 4.4 *The worst case node congestion in peer 0-centre routed HOPC on an S_n is $O(n)$.* □

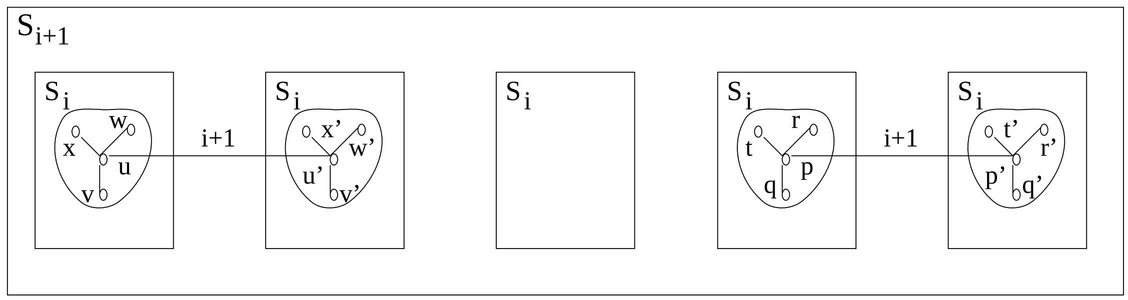
Path distribution

There are $i!$ sets of S_i peer nodes in an S_{i+1} and therefore there are $i!$ peer 0-centres. These $i!$ peer 0-centres are distributed equally among the S_i^ks, $k \in \{1, 2, ..., i\}$, such that each of these S_is contain $(i-1)!$ peer 0-centres. Consider a situation where each node in an S_i transmits a packet to one of its S_i peer nodes using peer 0-centre routed HOPC. Since the peer 0-centres are distributed over all the sibling S_is, except S_i^0, the communication paths will go through all the sibling S_is except may be S_i^0. Even if some of the sibling S_is are not involved in the computation, they will be used in the communication paths. This fact sometimes leads to an additional *expansion*, in case of an embedding scheme, as we shall see in a later chapter, when third substar routing is used.

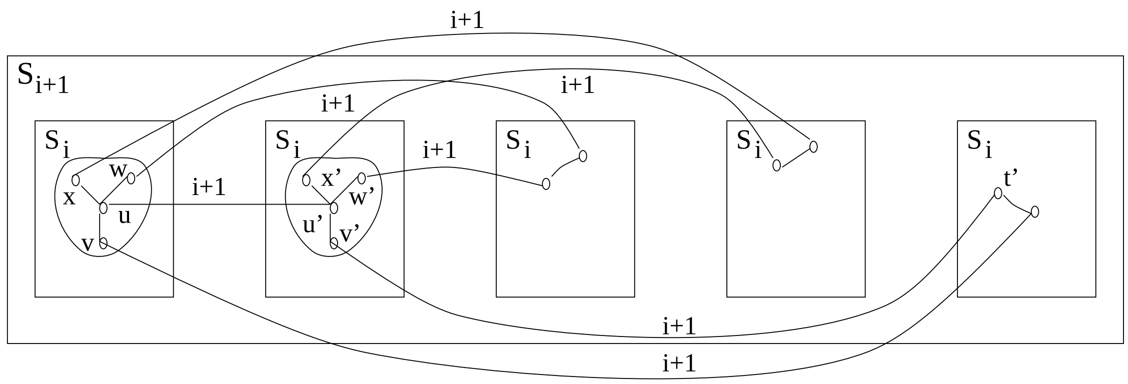
4.2 Adjacent-domain communication

An adjacent-domain communication consists of communication between a pair of (i+1)-th symbol swap image nodes in adjacent domains in sibling S_is. Let S_i^j and S_i^k be two sibling substars. The set of (i,k)-centres in S_i^j contains $(i-1)!$ nodes and their domains are disjoint and cover the entire set of nodes in S_i^j. Similarly, the corresponding set of (i,j)-centres in S_i^k also contain $(i-1)!$ nodes and their domains cover the entire set of nodes in S_i^k. Thus, through adjacent-domain-communication the two sibling substars S_i^j and S_i^k can perform ((i+1)-th symbol swap transformation) image-pair-wise communication. See Figure 4.4.

The relationship between two (i+1)-th symbol swap transformation image nodes u and v is given by $v = u(l\ i+1) = u(1\ l)(1\ i+1)(1\ l) = u(1\ i+1)(1\ l)(1\ i+1)$. Thus, there are two alternative paths– *path-1:* $(l,\ i+1,\ l)$ and *path-2:* $(i+1,\ l,\ i+1)$, between u and v and leads to two different schemes for the ADC paradigm. When *path-1* is used, the intermediate nodes in the path are confined to S_i^j and S_i^k, the substars containing the two communicating nodes. Therefore we refer to it as *confined routing* as in the case of HOPC. On the other hand, when *path-2* is used, the two intermediate nodes reside in a third substar S_i^m. Therefore, it is referred to as *third substar routing.* There is an exception however. When u and v are (i,k)-centre and (i,j)-centre respectively (*i.e.* $v = (1\ i+1)$), they are directly linked by a dimension $i+1$ edge and there are no alternative shortest paths. Since the path between the two nodes is a single-edge path, it does not involve a third substar.



a. Confined routing



b. Third substar routing

Figure 4.4: Adjacent domain communication.

4.2.1 Confined routing or domain collection and domain distribution ADC

In the adjacent domain communication, it is possible that the centres of the two domains act as collection and distribution centres in their respective domains. The data generated in the nodes of a domain can be collected at the centre of the domain and then can be relayed to the centre of the adjacent domain to be distributed to the destination nodes in that domain. A packet in this process will follow *path-1* above. In *path-1* the first edge and the third edge, that is the collection edge and the distribution edge, have the same dimension. The domain-collection-and-domain-distribution ADC between two nodes u and v in adjacent domains of sibling substars S_i^j and S_i^k respectively can therefore be performed in three steps as follows.

Collection step: The source node u hands over the data packet to its neighbouring (i,k)-centre.

Relay step: The (i,k)-centre hands over the data packet to the (i,j)-centre along the dimension $i+1$ edge.

Distribution step: The (i,j)-centre hands over the packet to the destination node.

Edge congestion

The communication paths in the domain-collection-domain-distribution ADC between the nodes in a pair of adjacent domains is depicted in Figure 4.4(a). It is evident that the dimension $i+1$ edge connecting the two centres of the communicating domains is shared in the communication and leads to an $O(n)$ edge congestion in an S_n.

THEOREM 4.5 *The worst case edge congestion in ADC using confined routing is $O(n)$ on S_n.* □

As in the case of confined routing for HOPC hot-spots on these congested edges can be eliminated through combining of the data at the collection centre.

Node congestion

In this case the node congestion is limited by the edge congestion. Therefore node congestion on S_n is $O(n)$.

4.2.2 Third substar routed ADC

In third substar routed ADC, a node u in S_i^j communicates to its (i+1)-th symbol swap transformation image node v in S_i^k as follows:

Step-1: The source node u transmit the data packet along dimension $i+1$ edge.
Step-2: The receiving node transmits the packet to the neighbouring (i,j)-centre.
Step-3: The (i,j)-centre hands over the packet to the destination node along the dimension $i+1$ edge.

Path distribution

The communication paths in the third substar routed ADC between the nodes in a pair of adjacent domains is depicted in Figure 4.4(b). It can be seen that the intermediate nodes in the paths are distributed to all the sibling S_is.

Congestion

As in the case of HOPC, for the congestion analysis we shall assume that the communication with the ADC paradigm is not performed at more than one substar level at a time. This assumption is valid for the algorithms developed based on this paradigm.

THEOREM 4.6 *Third substar routed ADC between nodes in a single pair of sibling substars is free of both edge and node congestion.*

PROOF. This is the case in which nodes in one pair of adjacent domains in two S_is in an S_{i+1} communicate as depicted in Figure‘4.5. Each of the source nodes has a distinct dimension $i+1$ edge and therefore a distinct adjacent node along dimension $i+1$ edge. The same is true for the destination nodes too. Therefore the pair of intermediate nodes in each of the paths is distinct. This implies that the communication is free of both edge and node congestion. □

THEOREM 4.7 *Third substar routed ADC between multiple disjoint pairs of sibling substars has edge-congestion two.*

PROOF. The pair of intermediate nodes in a path in ADC belongs to a third substar. If this third substar also participates in the communication, then these intermediate nodes will become source or destination nodes.

In the ADC between nodes in S_is, the dimensions of the edges in any path is less than of equal to $i+1$. Consider a dimension $i+1$ edge (v', t') (Figure 4.5) such that the two end

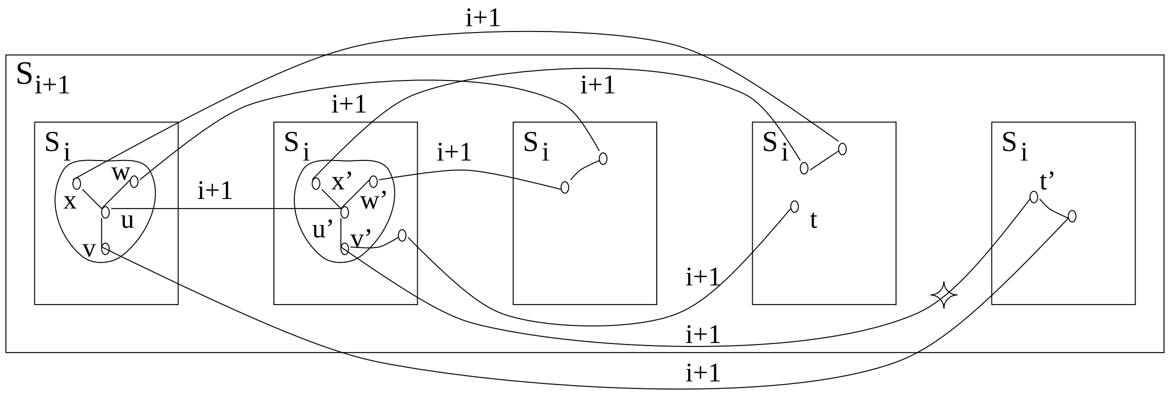


Figure 4.5: Congestion in pair-wise adjacent domain communication.

nodes v' and t' are source and destination nodes respectively in their communications. In such a situation the edge (v', t') will be shared by the two paths in the same direction. Since an edge can have only two end nodes, no more than two paths will share such an edge. Therefore congestion on edges of dimension $i + 1$ will be at most two.

Now let us consider the congestion on edges of dimension less than $i + 1$. An edge (u, v) of dimension less than $i + 1$ is used only by a link path of length 3. This occurs where u has dimension $i + 1$ edge to one of the end nodes of the link path and v has dimension $i + 1$ edge to the other end node of the link path. There can be at most one such link path for the edge (u, v). Hence, the congestion on all such edges will be at most one. Thus the over all congestion in the communication is two. $\square$

The *group-copy* [AQS93] paradigm is a special case of the ADC in which communication occurs between disjoint pairs of substars. From the above theorem it is clear that if group-copy is performed through third substar routing it will have an edge-congestion of two.

4.2.3 Pipelined order preserving communication (POPC)

The pipelined order preserving communication is a special case of the adjacent domain communication. Consider, the S_{n-1}s in S_n are logically organized in a pipeline as discussed in Section 3.4.2. In this if the peer nodes (according to the pipeline structured ranking scheme) in S_{n-1}s communicate in the pipelined fashion, the communication is referred to as *pipelined order preserving communication*. As the peer nodes in the adjacent S_{n-1}s in the pipeline are n-th symbol swap transformation images of each other, the communication corresponds to ADC between nodes in the adjacent S_{n-1}s.

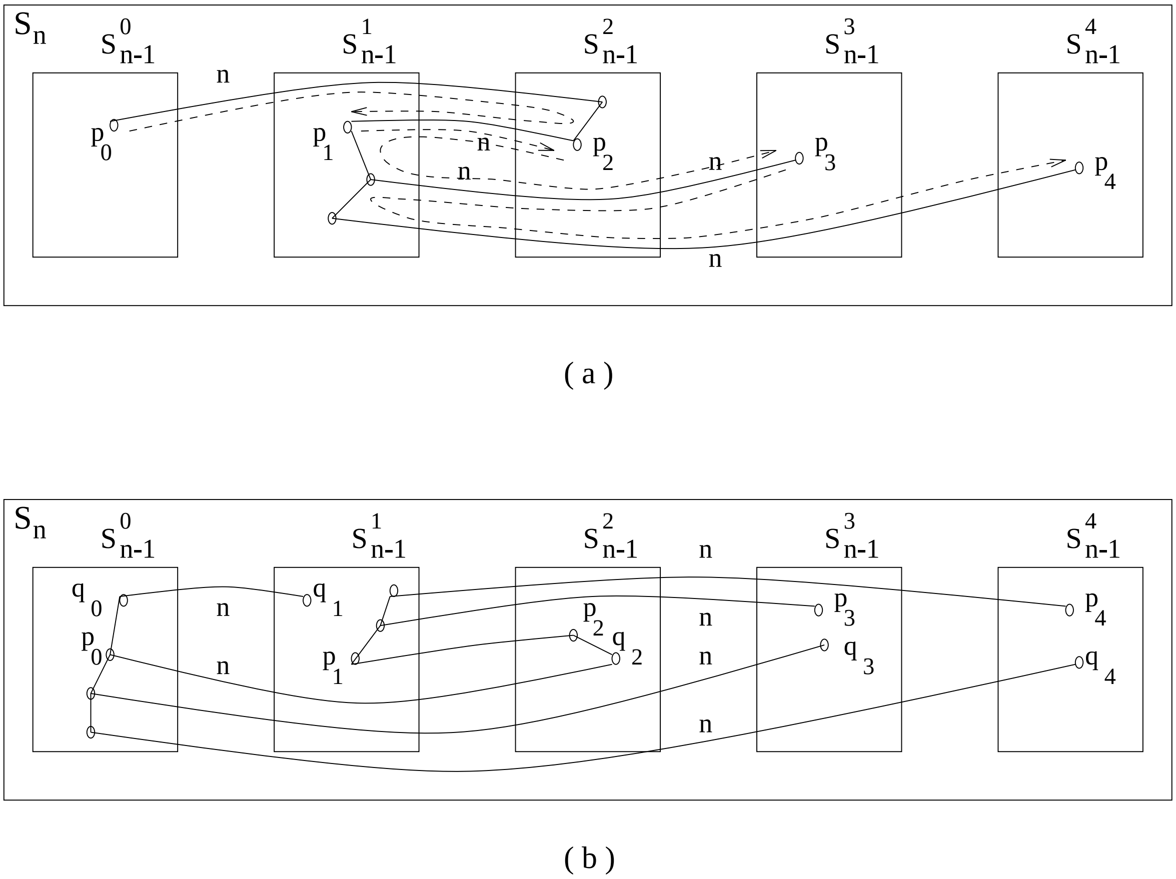


Figure 4.6: Pipelined order preserving communication (POPC). (a) A single thread of the pipeline, (b) two threads of the pipeline: $(p_0, p_1, p_2, p_3, p_4)$ and $(q_0, q_1, q_2, q_3, q_4)$

The two routing schemes, confined routing and third substar routing, are valid for POPC too. It is obvious that edge-congestion in POPC with confined routing is $O(n)$. In the following theorem we consider the edge-congestion aspect of POPC with third substar routing. We shall refer to the path between two peer nodes in adjacent S_{n-1}s as a *link path.* The chain of link paths starting from a node in the S^0_{n-1} to its peer node in S^n_{n-1}, will be called a *thread* in the pipeline. Figure 4.6 illustrates the POPC.

THEOREM 4.8 *If unidirectional pipelined order preserving communication is performed through third substar routing, the edge-congestion is two.*

PROOF. Let us first consider the congestion in the edges of dimension n. A dimension n edge, (u, v) can be used by a link path if u, v or both of them are its end nodes. When u and v are end points of two different link paths, then (u, v) will be used in both the link paths. In

POPC, a node can be end point of two link paths. As a result the edge (u, v) can be used in as many as four link paths. However, in unidirectional POPC, in the two link paths that a node can be an end point, the communication in one will be into the node and that in the other will be out of the node. Therefore, the four link paths in which the edge (u, v) can be used, the communication through (u, v) in two of them will be in the opposite direction to that in the other two. Hence the congestion on a dimension n edge, (u, v), can be at most two.

Following the same argument as in Theorem 4.7 the congestion on the edges of dimensions less than n will be 1. Therefore, the overall congestion in the communication is two. $\square$

4.3 Transmit-and-domain-distribute paradigm

The transmit-and-domain-distribute (TDD) paradigm is based on the concepts of (i,k)-centre and the domain of a node. It follows from the theorem below.

THEOREM 4.9 *If each node in an S_i contains a copy of a data packet, a copy of it can reach every node in the parent S_{i+1} in two communication steps.*

PROOF. Let S_i^k be the substar which contains a copy of the data packet in each of its nodes. From Proposition 3.1(b) we know that each sibling substar S_i^j, $0 \leq j \leq i$, $j \neq k$, has $(i-1)!$ (i,k)-centres. These (i,k)-centres in S_i^js can be reached by the packet in one step along the dimension $i+1$ edges from the nodes of S_i^k. Since, the domains of the (i,k)-centres in an S_i^j cover the entire set of nodes in S_i^j (Proposition 3.3(c)), all the other nodes in the S_i^j can be reached in the second step. $\square$

In a TDD communication the nodes in an S_i, say S_i^k, distribute data to the nodes in the parent S_{i+1}. Following discussion of the above theorem, the communication takes place in two steps as follows. In the first step the nodes in the S_i^k *transmit* the data to the (i,k)-centres in the sibling S_is along the dimension $i+1$ edge. Then the (i,k)-centres *distribute* the data among the nodes in their respective domains in S_i second step.

For example, in Figure 4.7(a), let each of the nodes in S_3^0 have a copy of the message. If each of these nodes transmit a copy of the message to the neighbours along the dimension 4 edges it will be received by the (3,0)-centres 4123 and 4213 in S_3^1, 4312 and 4132 in S_3^2 and 4321 and 4231 in S_3^3 in the first step. Now, these (3,0)-centres can spread copies of the message in the second step in their respective domains in S_3 to cover all the nodes in the S_4. Figure 4.7(b) illustrates the communication.

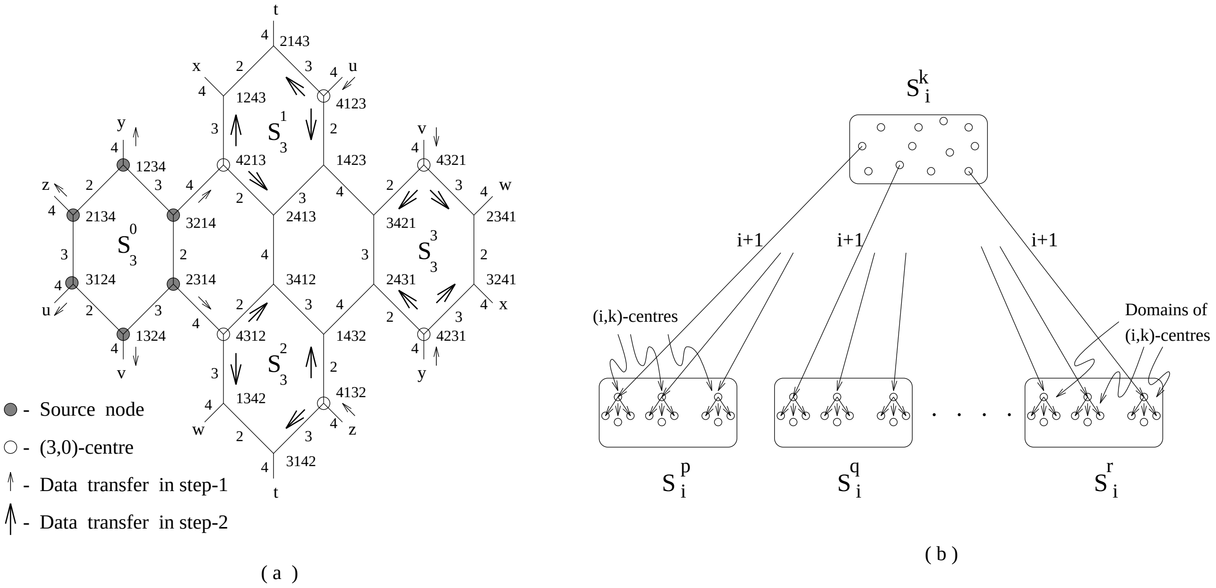


Figure 4.7: Transmit and domain distribute (TDD) communication paradigm. (a) Data distribution by the nodes in S_3^0 to the nodes in the parent S_4. (b) Data distribution by the nodes in S_i^k to the nodes in the parent S_{i+1}.

Based on the TDD paradigm data in a single node can be distributed to the entire S_n following a bottom-up approach. That is, iterating the above process for each S_{i+1} starting at substar level $i = 1$ and terminating at substar level $i = n - 1$. In an iteration the receiving node of the previous iteration in the S_{i+1} is to be made the source node. Therefore, the TDD paradigm can be used for global communication operations such as broadcasting, gossiping etc. and for prefix-like computations on the star graph based interconnection network.

4.4 Domain-distribute-and-transmit paradigm

The domain-distribute-and-transmit (DDT) paradigm is also based on the concepts of (i,k)-centre and the domain of a node and it follows from the theorem below.

THEOREM 4.10 *Given a node u in S_i^j, there is at least one node v, in any given sibling substar, S_i^k, which is within a distance of two edges from u.*

PROOF. According to Proposition 3.4, the domain of a node u in S_i^j has an (i,k)-centre to any of its sibling substars S_i^k, $k \in \{0, 1, \ldots, i\} \backslash j$. From an (i,k)-centre its corresponding (i,j)-centre v is single edge distance away. Therefore, there is a node v in S_i^k within two edge distance

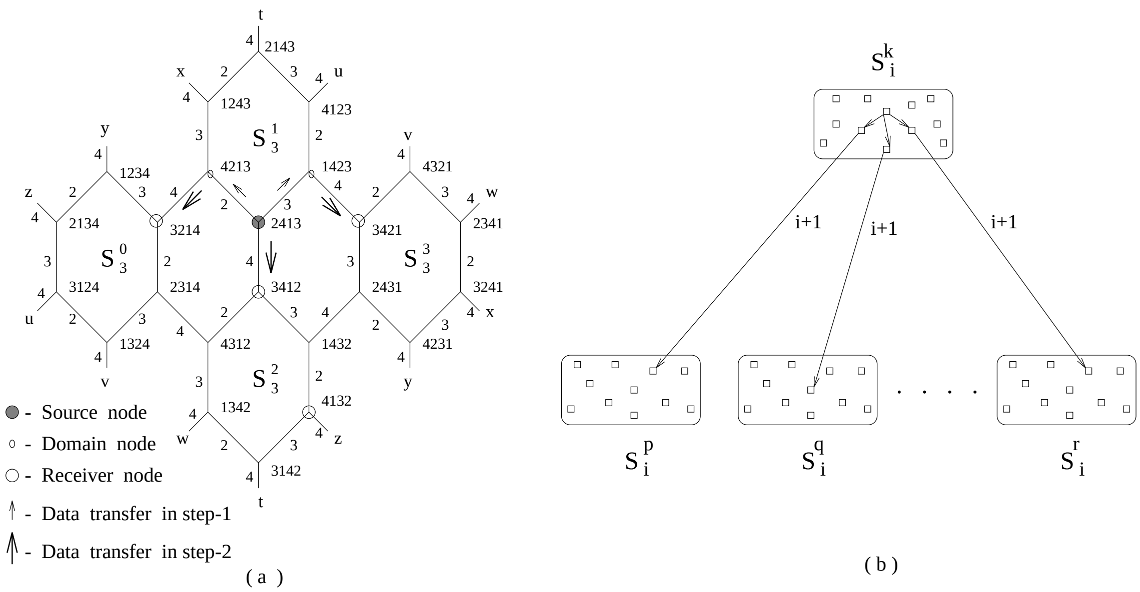


Figure 4.8: Domain distribute and transmit (DDT) communication paradigm. (a) Data distribution by a node in S_3^1 to one node in each of the S_3 in the parent S_4. (b) Data distribution by one node in S_i^k to one node in each of the S_is in the parent S_{i+1}.

away from u in S_i^j. $\square$

In a domain-distribute-and-transmit communication a node in an S_i, say S_i^j, distributes its data to one node in each of the sibling S_is. Following the above theorem the communication takes place in two steps as follows. In the first step the source node *distributes* the data among the nodes in its domain in S_i^j. Each of these i domain nodes *transmits* the data along the dimension $i+1$ edge to a node in one of the i sibling S_i.

For example, in Figure 4.8(a), let the node 1234 in S_3^0 has the data element to be transmitted to one node in each of the sibling substars S_3^1, S_3^2, and S_3^3. If the node 1234 distributes the data element to all the nodes in its domain in S_3^0 in the first step, the nodes 1234, 2134, and 3214 each will have a copy of the data element. In the second step if these nodes transmit the copy along dimension 4 edges the nodes 4231, 4132, and 4213 which belong to S_3^3, S_3^2, and S_3^1 respectively, will receive a copy of the data element. Figure 4.8 illustrates the communication.

Based on the DDT paradigm data in a single node can be distributed to the entire S_n following a top-down approach. That is, iterating the above process for each S_{i+1} starting at substar level $i = n-1$ and terminating at substar level i=1. In an iteration, the receiving node of the previous iteration in the S_{i+1} is to be made a source node. The DDT paradigm can be used for certain global communication operations such as broadcasting and scattering.

4.5 The communication processor

In this section we shall analyze the requirements of the communication processor of a node in a star graph interconnection network for supporting the communication paradigms discussed above. We particularly, look at two aspects of the communication processor: i) the communication model and ii) the routing information.

4.5.1 Communication models

Two common communication models used for the nodes of an interconnection network are the *one-port* model and the *multi-port* model. In the *one-port* model a node can transmit on at most one port and receive on at most one port at a time. The transmitting and the receiving ports need not be the same. In the *multi-port* model, on the other hand, a node can simultaneously transmit and receive on any number of its ports. The model supported in a parallel computer depends, among other things, on the communication hardware and the memory bus bandwidth. For a network with a relatively small degree it is not difficult to support the multi-port model. While majority of the commercially available systems support the one-port model, many systems supporting the multi-port model are also available.

In the communication paradigms presented, the processors may require to transmit and also receive messages on multiple ports simultaneously. For example, in the TDD paradigm, the distributing node requires to transmit the message to all the nodes in its domain. For example, in the case of DDT paradigm, the source node needs to transmit the message to all its domain nodes. Similar situation occurs for the HOPC and the ADC paradigms too. The multi-port communication model will allow these communication paradigms to be supported efficiently on a star graph based machine. Due to the sublogarithmic degree of the star graph, in a large network, the number of communication ports required in a node is considerably less than networks based on topologies such as the hypercube. (Refer Table 1.1). Therefore, it is possible to support the multi-port model in a star graph based model.

4.5.2 Routing information

It is necessary for the communication processor at a node to maintain a minimum set of the routing information to support the communication paradigms. It is sufficient for a communication processor in a system with a star graph based interconnection network to maintain the permutation and the address of the processor node. In the communication paradigms discussed, it is often required for a node to forward the message to the adjacent (i,k)-centre, where $0 \leq k \leq i$. For this the node should know the dimension of the edge that links it to the (i,k)-centre. This dimension is nothing but the position of s_{i+1}^k, the (i+1)-th symbol in the permutation of nodes of the sibling S_i^k, in the permutation of the node. Therefore, it is necessary to know s_{i+1}^k. The node can compute s_{i+1}^k from its address by computing the permutation of any node in the S_i^k. The cost will be least to compute the permutation of the 0-th ranked node in the S_i^k. If the address of the current node is $A = a_{n-1} \dots a_{i-2} a_{i-1} a_{i-2} \dots a_1$, the address of the 0-th ranked node in the S_i^k will be $A^0 = a_{n-1} \dots a_i k 0 \dots 0$. We know from Section 3.4 that the permutation u of a node with address $A = a_{n-1} a_{n-2} \dots a_1$ can be computed in $O(n)$ steps using the following expression.

$$u = id \circ (n \ \ (n - a_{n-1})) \circ ((n-1) \ \ ((n-1) - a_{n-2})) \circ \cdots \circ (2 \ \ a_1)$$

Since the address digits $a_{i-2}, \dots, a_1$ are 0s in A^0, the expression for the permutation u^0 of the node with address A^0 reduces to

$$u = id \circ (n \ \ (n - a_{n-1})) \circ ((n-1) \ \ ((n-1) - a_{n-2})) \circ \cdots \circ ((i+1) \ \ ((i+1) - a_i)) \circ (i \ \ (i-k))$$

As a result the cost of the computation will be reduced by half on the average. Further, for all practical purposes the value of n is small for a star graph based network.

4.6 Summary

We have developed four different communication paradigms for the star graph interconnection networks. They are - i) hierarchical order preserving communication (HOPC), ii) adjacent domain communication (ADC), iii) transmit-and-domain-distribute (TDD), and iv) domain-distribute-and-transmit (DDT). These communication paradigms are suitable for developing algorithms for a wide variety of problems. We have developed alternative routing schemes for these communication paradigms and have characterized them with respect to path length, edge and node congestion, and path distribution. The HOPC paradigm is based on the hierarchical ranking scheme. The other three paradigms, namely, ADC, TDD and DDT are based on a decomposition of the star graph which is based on the concepts of *(i,k)-centre* and the *domain of a node* that were introduced in the previous chapter. We have also analyzed the requirements of the communication processor of a node, namely, the communication model and routing information, to support these communication paradigms in star graph based interconnection networks.

Chapter 5

The Star Cluster

Several modifications to the star graph have been proposed to alleviate the problem of extremely rapid rate of expansion [DT93a, Lat93b, SS93a]. While these topologies have good characteristics from incremental growth point of view, the algorithm implementability aspect is not very encouraging or totally unknown. Very recently, Latifi [Lat93a] proposed the *clustered-star*, CS^m_{n-1}. This topology has near-linear rate of growth and also has suitable characteristics for algorithm implementation. However, there is restriction in the choice of the substars (S_{n-1}s) included in a CS^m_{n-1}. As a result, though clustered-star addresses the expansion problem, the problem of efficient utilization of processor resources in a star graph based interconnection network remains unresolved.

We propose a modification to the definition of clustered-star and call it the *star cluster*.

DEFINITION 5.1 **Star cluster**
A star cluster $SC_{n,k}$ *is a subgraph of an* S_{n+1} *induced by any* k, $1 \leq k < n+1$, *sibling* S_n*s.*

From the definition it is obvious that a star cluster $SC_{n,k}$ is a subgraph of S_{n+1} with some of the S_ns excluded from it. The value of k lies in the range $[1, \ldots, n]$. For $k = 1$, $SC_{n,k}$ becomes an S_n. A star cluster can grow from an $SC_{n,1}$ with $n!$ nodes to an $SC_{n,2}$ with $2n!$ nodes from which it can grow to an $SC_{n,3}$ with $3n!$ nodes and so on.

The degree of a node in $SC_{n,k}$ is either n or $n-1$. If a node is an (n, ℓ)-centre and the S^ℓ_n is not included in the $SC_{n,k}$, its degree is $n-1$ with the edge of dimension $n+1$ missing. Otherwise, its degree is n. There are $k(n-k+1)(n-1)!$ nodes with degree $n-1$ and the rest $k(k-1)(n-1)!$ nodes with degree n.

From the definition of the *star cluster* it is clear that the k S_ns for an $SC_{n,k}$ can be chosen

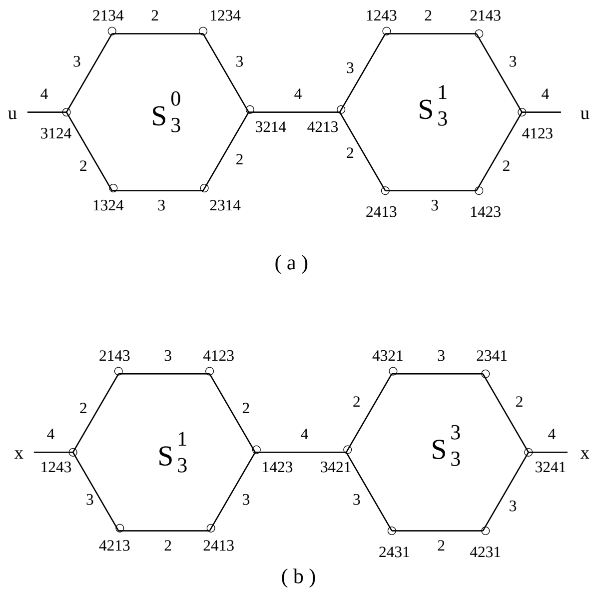


Figure 5.1: Two star clusters $SC_{3,2}$s.

arbitrarily from the set of $n+1$ S_ns in an S_{n+1}. From the expansion point of view the set of S_ns could have been restricted to the set, say, $\{S_n^0, S_n^1, \ldots, S_n^{k-1}\}$ as is done in case of the clustered star. Figure 5.1 shows two $SC_{3,2}$, star clusters. In Figure 5.1(a), the $SC_{3,2}$ consists of S_3^0 and S_3^1 and in Figure 5.1(b) the $SC_{3,2}$ consists of S_3^1 and S_3^3. This flexibility of arbitrary choice of S_ns in an $SC_{n,k}$ greatly enhences its utility. It can be used for efficient utilization of the processor resources in star graph based parallel machines through processor allocation in terms of star clusters. For example, for the job requiring one thousand processor nodes an $SC_{6,2}$ consisting of only 1440 nodes can be allocated leaving the rest of the network free for use by the other jobs. In contrast, if a substar is allocated to the job, it will require an S_7 consisting of 5040 nodes resulting in less than 20% utilization. Further, the star cluster concept can be used for developing schemes for efficient embeddings on star graph based networks.

Most of the desirable properties of the star graph are preserved in a star cluster. As a result most algorithms developed for the *complete star* can be implemented, with minor or no modifications, on star clusters. This will be demonstrated while developing algorithms for the star graph.

In the following, we characterize the *valid paths* in a star cluster and develop a scheme for shortest path routing on it. We analyze some topological aspects of the *star cluster* to show that an $SC_{n,k}$ preserves many of the characteristics of the *complete star*. We look into the adaptability of the communication paradigms defined in the previous chapter for the star cluster.

For some of our embedding schemes and for processor allocation schemes it will be necessary to recognize star clusters in a star graph based network. We present a two dimensional arrangement of substars which leads to simple ways for recognition of star clusters with two different CDSs. With reference to the two dimensional arrangement, we also define a concept called *distance-k set of star clusters* which is useful in the development of embedding schemes for certain interconnection structures. We devise methods for obtaining such sets of star clusters.

5.1 Routing in a star cluster

In a star cluster $SC_{n,k}$, some of the sibling S_n substars of S_{n+1} do not exist. A path between two nodes can not include edges in these non existent S_ns. Therefore, a routing scheme for star clusters must have restrictions to avoid such edges. We shall call a path found using such a scheme a *valid path.*

LEMMA 5.1 *(a) A path between two nodes in a star cluster $SC_{n,k}$ is a* valid path *if there is at most one dimension $n+1$ edge in it.*
(b) A path with two dimension $n+1$ edges is valid in an $SC_{n,k}$, if source and destination nodes are of degree n and have identical first symbols and the dimension $n+1$ edges are starting and terminating edges in the path.
(c) A path with three dimension $n+1$ edges is valid in an $SC_{n,k}$ if both source and destination nodes are of degree n and their first symbols differ and the starting and terminating edges are of dimension $n+1$.

PROOF. Part-(a):
From the scheme adapted for factoring of the star we know that to move from one S_n to another S_n a dimension $n+1$ edge must be used. Let u and v be two nodes in S_n^p and S_n^q respectively and π be a path from u to v. Let S_n^r be a substar excluded from the $SC_{n,k}$. Let us assume that π goes through S_n^r. Then π must have a dimension $n+1$ edge to cross from S_n^p to S_n^r and another dimension $n+1$ edge to cross from S_n^r to S_n^q. Therefore, there must be at least two dimension $n+1$ edges if π is to use edges in S_n^r, a non existent substar. Hence, a path with at most one dimension $n+1$ edge in it is a valid path in $SC_{n,k}$.

Part-(b):
Since both source node and destination node have the same first symbol, both must be

(n, j)-centre to the same S_n^j. Since the nodes are of degree n the substar S_n^j must be a part of the $SC_{n,k}$. In the proposed path the first and the last edges are of dimension $n+1$ and no other edge in the path is of dimension $n+1$. Therefore, the rest of the path will be confined in the substar S_n^j. Thus the path is valid in the $SC_{n,k}$.

Part-(c):
Since the first symbols of the two nodes differ, they have edges of dimension $n+1$ to two different S_ns and these two S_ns exist as the two nodes are of degree n. The path under question has the starting and terminating edges of dimension $n+1$. Therefore it enters one of these two S_ns and emerges out of the other S_n. The third edge of dimension $n+1$ edge in the path must be to cross from the first S_n to the second S_n. Therefore, the path does not use any S_n that does not exist in the $SC_{n,k}$ and hence it is valid. □

Thus we have a set of rules to find *valid paths* between nodes in an $SC_{n,k}$. We shall adapt these in our routing schemes for $SC_{n,k}$.

5.1.1 Shortest path routing

We can have a shortest path routing scheme for the $SC_{n,k}$ from the shortest path routing scheme for the star graph discussed in Section 2.2.1 by imposing the restrictions due to the rules discussed above. It may be noted that the third rule in Lemma 5.1 is not relevant to shortest path routing as a shortest in a star graph can have at most two edges of the same dimension (Property 2.1). The modified shortest path routing rules for star clusters are given below. Recollect that the shortest path routing from a source node u to destination node v consists of sorting the relative permutation R of u with respect to v. The application of a transposition $(1\ i)$ at node u implies use of an edge of dimension i to reach the node $u(1\ i)$. Moreover, $R(1) = 1$ means that the source and destination nodes have the same first symbol.

Rule-1: If $R(1) = 1$, apply transposition $(1\ i)$, where $R(i) \neq i$. Allow i to be
equal to $n+1$ only if the degree of the current node is n,
i.e. when the dimension $n+1$ edge exists in the node.

Rule-2: If $R(1) = i \neq 1$, then either,
(a) apply transposition $(1\ i)$,
or,
(b) apply transposition $(1\ j)$, where
$R(j) \neq j$ and j is not in the cycle containing 1.
Allow j equal to $n+1$ only if the degree of the current node is n,

i.e. when the dimension $n+1$ edge exists in the node.

THEOREM 5.1 *Sorting of the relative permutation R using the rules above gives valid shortest paths from the source to the destination node in a star cluster.*

PROOF. The rules above are same as those given in [AHK87, DT93a] except for the restrictions on the use of the dimension $n+1$ edges. A dimension $n+1$ edge is used through Rule-1 when the first symbol of the source and the destination nodes are the same and the source node has a dimension $n+1$ edge. Therefore, it will lead to an S_n that exists in the $SC_{n,k}$. It is used through Rule-2(a) when $R(1) = n+1$, *i.e.* the first symbol of the source node is the (n+1)-th symbol of the destination node. This move places the (n+1)-th symbol in the correct position so that the path will enter the S_n of the destination node and there will not be any more of dimension $n+1$ edges in the path. Therefore, the resulting paths are *valid* for the *star cluster*. A dimension $n+1$ edge is also allowed to be used through Rule-2(b) when the current node is of degree n. Since the path does not violate the rules for shortest path routing in a star graph the resulting paths are also of shortest distance in the *star cluster*. □

LEMMA 5.2 *A path between two nodes in $SC_{n,k}$ obtained by applying the shortest path routing rules discussed above is of the same length as the shortest distance from each other in S_{n+1}.*

PROOF. No move that follows the shortest path routing rules discussed above violates the rules for finding the shortest paths for S_{n+1}. (Refer to Section 2.2.1). Therefore, the paths derived through these rules have the same length as a shorest path in S_{n+1}. □

5.2 Topological properties of the star cluster

In this section, we investigate some topological aspects of the star cluster.

THEOREM 5.2 *All $SC_{n,k}$s, for a given pair of n and k values, are isomorphic.*

PROOF. Let X and Y be two star clusters, $SC_{n,k}$s in an S_{n+1} and G be a subgraph of X. Let us assume that k' of the S_ns in the two $SC_{n,k}$s are common and let the two sets of S_ns in the two $SC_{n,k}$s be:
$X: \{S_n^{c_1}, S_n^{c_2}, \ldots, S_n^{c_{k'}}, S_n^{x_1}, \ldots, S_n^{x_{k-k'}}\}$ and
$Y: \{S_n^{c_1}, S_n^{c_2}, \ldots, S_n^{c_{k'}}, S_n^{y_1}, \ldots, S_n^{y_{k-k'}}\}$.

Consider a sequence of $k-k'$ (n+1)-th symbol swap transformations $(t(s^{x_j}_{n+1}) = s^{y_j}_{n+1}, t(s^{y_j}_{n+1}) = s^{x_j}_{n+1})$, $j = 1, 2, \ldots, k-k'$ applied successively on G. According to Lemma 3.2, the transformation $(t(s^{x_j}_{n+1}) = s^{y_j}_{n+1}, t(s^{y_j}_{n+1}) = s^{x_j}_{n+1})$, will map the portion of G in $S_n^{x_j}$ to $S_n^{y_j}$ while the portions of G in the remaining S_ns will remain confined in their respective S_ns. Therefore, application of each of the (n+1)-th symbol swap transformations of the sequence will map only the portion of G in one S_n in X but not in Y to one S_n in Y but not in X and at the end of the sequence of $k-k'$ (n+1)-th symbol swap transformations G will be completely mapped to the S_ns in Y. If $G = X$ then the above sequence of (n+1)-th symbol swap transformations will perform a one-to-one mapping of X onto Y. Hence, any two $SC_{n,k}$s are isomorphic. $\square$

LEMMA 5.3 *A star cluster $SC_{n,k}$ is bipartite and both the partitions contain equal number of nodes.*

PROOF. A star graph S_n is bipartite where the even parity nodes form one partition while the odd parity nodes form the other partition [LJD93] [1]. Each of the k S_ns constituting the $SC_{n,k}$ therefore is bipartite with equal number of nodes in the two partitions. If an odd parity node u in one of the S_ns is connected to a node v in another S_n in the $SC_{n,k}$ then v must be an even parity node. Thus in the $SC_{n,k}$ too the even parity nodes will form one partition and the odd parity nodes will form the other partition. Since each of the S_ns contain equal number of even and odd parity nodes, the two partitions in $SC_{n,k}$ too will have equal number of nodes. $\square$

THEOREM 5.3 *(a) The star cluster, $SC_{n,k}$ is Hamiltonian for $n > 3$. It is also Hamiltonian for $2 \leq n \leq 3$ when $k > 2$.*
(b) The star cluster, $SC_{n,2}$, $n \leq 3$, has a Hamiltonian path.

PROOF. The part-(a) of the theorem follows from the Jwo's method of construction of cycles in $SC_{n,k}$, for $k \geq 3$ and $n \geq 3$, by combining Hamiltonian cycles in the S_ns of the $SC_{n,k}$ (refer Section 2.4.1) and our method of construction of a cycle in an $SC_{n,2}$, $n > 3$, discussed in Section 6.1.2.

Part-(b) :
It is known that S_n has a Hamiltonian path. Due to symmetry, it can start at any chosen node or terminate at any chosen node. Now in the two S_ns constituting the $SC_{n,2}$, if we construct one Hamiltonian path in each such that the end point in one has a dimension $n+1$ edge to

[1] The parity of a node in S_n is defined as odd or even depending upon whether its shortest distance from the identity permutation node is odd or even.

the starting point of the other, what we get is a Hamiltonian path for the $SC_{n,2}$. □

Form Lemma 5.2, it directly follows that the diameter of a star cluster $SC_{n,k}$ is same as that for an S_{n+1}. Hence, we have the following theorem.

THEOREM 5.4 *The diameter of an* $SC_{n,k}$ *is* $\lfloor \frac{3n}{2} \rfloor$. □

5.2.1 Parallel paths

In an n-star there exist $n-1$ parallel paths [DT93a] between a pair of nodes u and v. The number of parallel paths, it may be noted, is equal to the degree of a node in S_n. In an $SC_{n,k}$ some of the nodes have degree $n-1$ and the rest have degree n. It is easy to see that if either of the two nodes u and v in $SC_{n,k}$ have degree $n-1$ the number of parallel paths between them cannot exceed $n-1$. On the other hand, if both the nodes u and v are of degree n, it is possible that there will be n parallel paths between them. The exact number of parallel paths and their lengths are expressed in the Propositions below.

Let R be the relative permutation of u with respect to v and it has c p-cycles and containing a total of m symbols.

PROPOSITION 5.1 *If the first symbol in the two nodes* u *and* v *are identical and also the (n+1)-th symbols are identical, that is,* $R(1) = 1$ *and* $R(n+1) = n+1$, *then,*
i) there are m *parallel paths of length* $c+m$ *between* u *and* v.
ii) If both u *and* v *are of degree* n, *there are* $n-m$ *parallel paths of length* $c+m+2$. *Otherwise, there are* $n-m-1$ *parallel paths of that length.*

Recollect that when the first symbols of the source and the destination nodes are identical, the parallel paths in an S_n are obtained such that the starting and the terminating edges in a path are identical (Section 2.3.2). Since the nodes have identical (n+1)-th symbol, both belong to the same S_n. Therefore, $n-1$ of the parallel paths between u and v can be constructed as in an S_n. The remaining parallel path with starting and terminating edges of dimension $n+1$ will exist if the nodes are of degree n. The length of this path will be $c+m+2$ as the symbol $n+1$ is invariant in R (refer to Section 2.3.2). The construction of the parallel paths is depicted in Figure 5.2.

PROPOSITION 5.2 *If the first symbols of the two nodes* u *and* v *are identical but the symbols at (n+1)-th position differ, i.e.* $R(1) = 1$ *and* $R(n+1) \neq n+1$, *then*
i) there are m *parallel paths of length* $c+m$ *if both* u *and* v *have degree* n. *Otherwise, there are* $m-1$ *parallel paths of length* $c+m$.

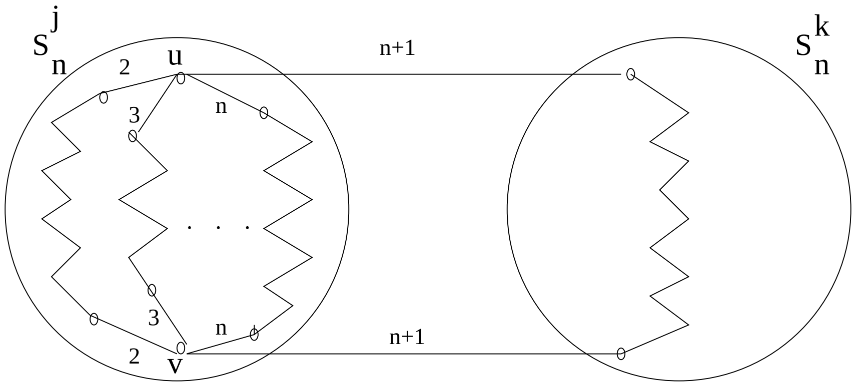


Figure 5.2: Construction of the parallel paths when the first symbols and the (n+1)-th symbols of the source and destination nodes u and v are identical.

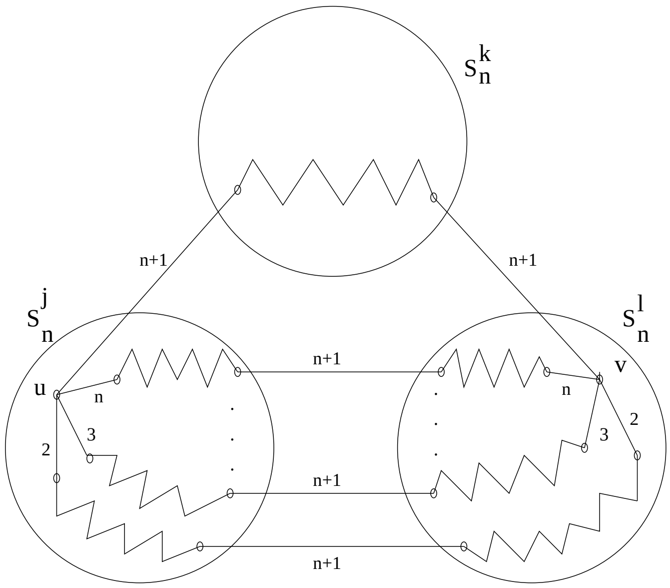


Figure 5.3: Construction of the parallel paths when the first symbols are identical but the (n+1)-th symbols are different.

ii) there are $n-m$ parallel paths of length $c+m+2$.

As the first symbols of the two nodes are identical, both the nodes will be (n,j)-centre to the same S_n^j. The construction of the parallel paths can be done in the same way as in an S_{n+1}. The dimensions of the starting and the terminating edges in a path will be identical. The path with the starting and terminating edges of dimension $n+1$ will exist if the the two nodes are of degree n. The length of this path will be $c+m$ as the symbol $n+1$ belongs to a p-cycle in R. Figure 5.3 depict the construction.

PROPOSITION 5.3 *If the first symbols in u and v differ but their (n+1)-th symbols are identical, that is, $R(1) \neq 1$ but $R(n+1) = n+1$, then*
i) there are c parallel paths of length $c+m-2$.
ii) there are $m-c$ parallel paths of length $c+m$ between u and v.
iii) there are $n-m$ parallel paths of length $c+m+2$ if both u and v have degree n. Otherwise there are $n-m-1$ parallel paths of length $c+m+2$.

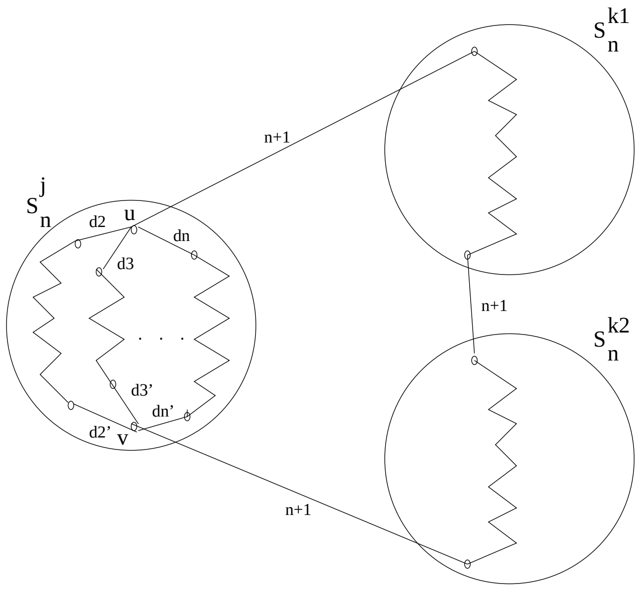


Figure 5.4: Construction of the parallel paths when the first symbols are different but the (n+1)-th symbols are identical.

In this case both the nodes u and v belong to the same S_n. Moreover, the symbol $n+1$ is invariant in R. In the construction of parallel paths in S_n, when $R(1) \neq 1$, the parallel paths of length $c+m+2$ correspond to the invariant symbols in R and the starting and terminating edges of such a path have dimensions same as that of the position of an invariant symbol in R (refer to Section 2.3.2). Further, none of the other parallel paths contain an edge of dimension corresponding to the position of an invariant symbol. Thus, if the parallel paths are constructed as in an S_{n+1}, one of the paths will have starting and terminating edge of dimension $n+1$. No other path will contain an edge of dimension $n+1$ and therefore the remaining paths will be confined within the S_n containing u and v. The construction of the parallel paths for this case is depicted in Figure 5.4.

PROPOSITION 5.4 *If the symbols at the first position and also the symbols at (n+1)-th position differ, i.e.* $R(1) \neq 1$ *and* $R(n+1) \neq n+1$, *then*
i) there are c *parallel paths of length* $c+m-2$,
i) there are $m-c$ *parallel paths of length* $c+m$ *between* u *and* v *if both have degree* n. *Otherwise there are* $m-c-1$ *parallel paths of length* $c+m$ *between them.*
ii) there are $n-m$ *parallel paths of length* $c+m+2$.

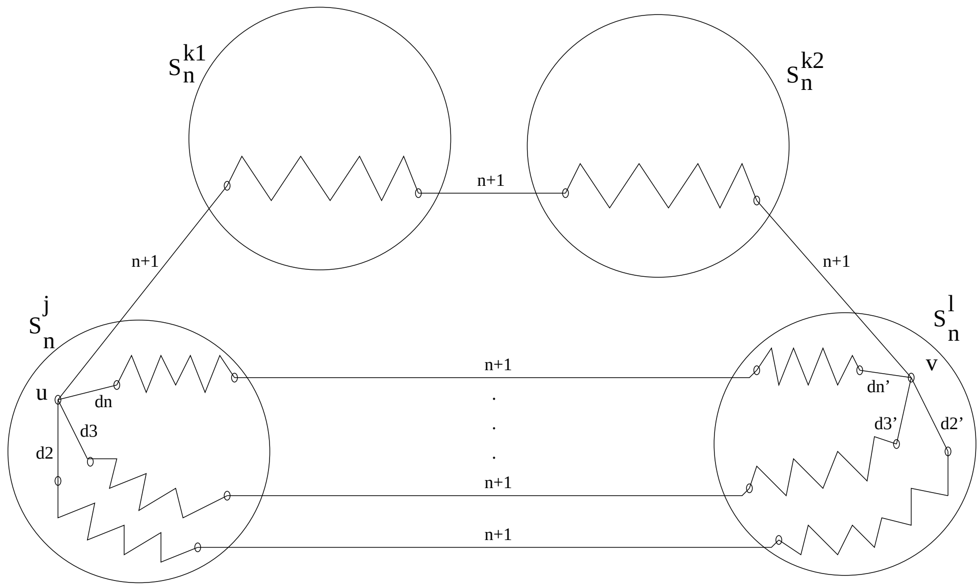


Figure 5.5: Construction of the parallel paths when the first symbols and also the (n+1)-th symbols are different.

In this case both the nodes u and v belong to two different S_ns. Moreover, the symbol $n+1$ belongs to a p-cycle in R. In the construction of parallel paths in S_n, when $R(1) \neq 1$, the parallel paths of length $c+m-2$ correspond to the first symbols in the p-cycles in R (Section 2.3.2). The starting edge of such a path corresponds to the position of the first symbol in one of the p-cycles while the terminating edge corresponds to the first symbol of another p-cycle. On the other hand, a path of length $c+m$ correspond to a symbol in the p-cycles not occupying the position one and the starting and terminating edges of such a path have a dimension corresponding to the position of this symbol in R. In all the paths other than this, the dimension corresponding to the symbol will occur only once. Since the positions of the symbols in a p-cycles can be cyclically shifted, we can arrange the p-cycle containing $n+1$ such that it does not occupy the first position in the cycle. Now if the parallel paths between u and v are constructed as in an S_{n+1}, there will be a path of length $c+m$ with starting and terminating edges of dimension $n+1$. This path will exist in the $SC_{n,k}$ if both the nodes u and v are of degree n. In all the other parallel paths the dimension $n+1$ will occur only once and these will be valid paths in the $SC_{n,k}$. The construction of the parallel paths for this case is illustrated in Figure 5.5.

From the above, we can conclude the following.

THEOREM 5.5 *The parallel paths between any two nodes in $SC_{n,k}$ are identical in number and length to those in an $SC_{n,k}$ as those in an S_{n+1}, if the both the nodes are of degree n. If either of the nodes is of degree $n-1$, one of the paths in an S_{n+1} will not exist in the $SC_{n,k}$.* □

5.3 Adaptation of communication paradigms

Adaptation of the various communication paradigms for the star graph on star clusters is necessary for implementation of various star graph algorithms on star cluster. In the following, we discuss this issue of adapting the communication paradigms: HOPC, ADC, TDD, and DDT of the star on star clusters.

HOPC and ADC:

The communication paradigms HOPC and ADC both have two alternative communication schemes, namely, confined routing and third substar routing. It may be observed that in a path obtained through confined routing the dimension $n+1$ occurs at most once. Therefore, confined routing always gives valid path in a star cluster $SC_{n,k}$. However, while performing HOPC or ADC at the S_n level, it is possible that the third substar does not exist. In that case the third substar routing will be disallowed. This problem occurs only at the S_n level of HOPC or ADC. At all the levels below S_n communication with these paradigms can be performed using both confined routing and third substar routing. Therefore, valid paths can be ensured in HOPC and ADC by avoiding third substar routing at the S_n level.

TDD and DDT:

In the TDD or the DDT communication paradigm the dimension $n+1$ can occur in a path between two sibling S_ns and only once. Therefore, the resulting path is always valid for the star cluster. Hence there is no restriction on using these two paradigms in a star cluster $SC_{n,k}$.

5.4 Recognition of star clusters

We shall use the star cluster concept for some compact embedding schemes and for efficient processor allocation schemes for star graph based networks. In these it will be necessary to obtain smaller star clusters within larger star clusters. For this we use a two dimensional arrangement of substars.

5.4.1 Two dimensional arrangement of substars

It is possible to represent an S_{i+2} as a two dimensional array of its S_is. This representation allows $SC_{i,k}$s to be obtained in the S_{i+2} using two CDSs. This ability allows more $SC_{i,k}$s to be obtained in an S_{i+2} than it is possible with one CDS. As a result it is possible to develop schemes for compact embedding of certain interconnection structures such as the full binary tree. This representation also simplifies identification and allocation of star clusters in processor allocation for a multitasking environment. The representation is based on the hierarchical ranking of the substars.

An S_{i+2} has $i+2$ S_{i+1}s and each of these S_{i+1}s has $i+1$ S_is. In total there are $(i+2)(i+1)$ S_is in an S_{i+2}.

THEOREM 5.6 *Consider a logical two dimensional $(i+2)\times(i+1)$, row major, array arrangement of the S_is (with PCDS) of an S_{i+2} (also with PCDS) in the ascending order of their ranks in the S_{i+2} as shown in Table-5.1. In this array arrangement the $i+1$ S_is in each of the $i+2$ rows forms a PCDS S_{i+1}. Each of the $i+1$ columns has one S_{i+1} with the SCDS $\{2,3,\ldots,i,i+2\}$. One element in each column is not included in the S_{i+1} of that column. These excluded S_is from the column S_{i+1}s lie in the diagonal of the array consisting of the $(j,j-1)$-th S_i, $1\leq j\leq i$, in the array. These diagonal S_is form another S_{i+1} with the SCDS $\{2,3,\ldots,i,i+2\}$.*

0	1	2	...	$i-1$	i
i+1	i+1+1	i+1+2	...	$i+1+i-1$	i+1+i
.	.	.	...	.	.
.	.	.	...	.	.
(i+1)(i+1)	(i+1)(i+1)+1	(i+1)(i+1)+2	...	$(i+1)(i+1)+i-1$	(i+1)(i+1)+i

Table 5.1: Two dimensional arrangement of S_is in an S_{i+2}. The S_is are represented with their ranks in the S_{i+2}.

PROOF. We shall prove the theorem for the PCDS S_{i+2} that contains the identity permutation node. As the ranking of the substars in any other S_{i+2} is obtained through i-th symbol swap transformation from this S_{i+2} and i-th symbol swap transformation is isomorphic in nature, the result will hold good for any PCDS S_{i+2}.

From property 3.2, it follows that the S_i in each of the rows in the arrangement form a PCDS S_{i+1}. Further, from property 3.3 it follows that the S_{i+1} of row j will be of rank j in the S_{i+2}, assuming that the rows are numbered starting from 0.

In the S_{i+2} containing the identity permutation node, the (i+2)-th symbol in the nodes of the j-th ranked S_{i+1} is $i+2-j$. Therefore, the S_is in the j-th row of the arrangement will have the (i+2)-th symbol as $i+2-j$. Again, in the S_{i+1} containing the identity permutation node, the (i+1)-th symbol in the nodes of the k-th ranked S_i will be $i+1-k$. Thus the k-th ranked S_i in S^0_{i+1} will be $*(i+1-k)(i+2)$.

According to the ranking scheme, the k-th ranked S_i in $S^j_{i+1}, j \neq 0$ is the (i+2)-th symbol swap transformation image of the k-th ranked S_i in S^0_{i+1} as they are of same rank in S_{i+1}s. The symbols swapped in the transformation will be $i+2$ and $i+2-j$. Therefore, the S_i in the k-th column and j-th row in the arrangement will be $*(i+1-k)(i+2-j)$ except when $i+1-k = i+2-j$, *i.e.* $k = j-1$. For the case where $k = j-1$ the S_i will be $*(i+2)(i+2-j)$ as in this case of the (i+2)-th symbol swap transformation the (i+1)-th and the (i+2)-th symbols (both fixed symbols) in the two S_is will be swapped. Therefore, all the S_is in the k-th column, except for the S_i in the j-th row, where $k = j-1$, will have the same (i+1)-th symbol $i+1-k$. However the (i+2)-th symbol in these S_is vary as it is $i+2-j$. Therefore, the $i+1$ symbols in the positions $1, 2, \ldots, i$ and $i+2$ in the $i+1$ S_i of the k-th column, $k \neq j-1$, with the (i+1)-th symbol remaining fixed. Hence, by definition, the $i+1$ S_i in a column form an S_{i+1} with the SCDS $\{2, 3, \ldots, i, i+2\}$.

The (j,k)-th S_is in the array arrangement, where $k = j-1$ for $j = 1, 2, \ldots, i+1$, are not included in the S_{i+1}s of the columns. These S_is form a diagonal in the array as $k = j-1$. In these S_is too the (i+1)-th symbol is fixed, $i+2$, and the (i+2)-th symbol permute. Therefore, the S_is in the diagonal form an S_{i+1} with the SCDS $\{2, 3, \ldots, i, i+2\}$. □

We shall use the two dimensional arrangement of S_is in S_{i+2} to identify the star clusters $SC_{i,k}$ with either PCDS or the SCDS $\{2, 3, ..., i, i+2\}$. In this thesis, we shall use the term *diagonal* to mean the set of (j,k)-th S_is, in the two dimensional arrangement, where $k = j-1,\ j = 2, 3, \ldots, i+2$.

THEOREM 5.7 *Let G be a subgraph of an S_{i+1} in a row (i.e. with PCDS) and u be a node in*

S_i rank / S_{i+1} rank	0	1	2	...	$i-1$	i
0	$*(i+1)(i+2)$	$*(i)(i+2)$	$*(i-1)(i+2)$	...	$*(2)(i+2)$	$*(1)(i+2)$
1	$*(i+2)(i+1)$	$*(i)(i+1)$	$*(i-1)(i+1)$	...	$*(2)(i+1)$	$*(1)(i+1)$
2	$*(i+1)(i)$	$*(i+2)(i)$	$*(i-1)(i)$	...	$*(2)(i)$	$*(1)(i)$
.	.	.	.	...	.	.
.	.	.	.	...	.	.
i	$*(i+1)(2)$	$*(i)(2)$	$*(i-1)(2)$	...	$*(i+2)(2)$	$*(1)(2)$
$i+1$	$*(i+1)(1)$	$*(i)(1)$	$*(i-1)(1)$	...	$*(2)(1)$	$*(i+2)(1)$

Table 5.2: Two dimensional arrangement of S_is in an S_{i+2}. The S_is are denoted by their fixed symbol representation.

G. A dimension swap transformation $T_d =< u, t_d >$ with the dimension substitution operator $(t_d(i+1) = i+2, t_d(i+2) = i+1)$ will map it to the S_{i+1} in the column (i.e. with SCDS) to which u belongs or to the diagonal if u belongs to the S_{i+1} of the diagonal.

Proof. The graph G can be partitioned into subgraphs contained in the sibling S_is of the S_{i+1} to which it belongs. These subgraphs are linked by edges of dimension $i+1$ as the S_{i+1} is composed with PCDS. In the image, say G', these edges will be replaced by edges of dimension $i+2$. According to Lemma 3.6, the subgraph of G in the S_i containing the invariant node u remains invariant. The subgraphs in the other S_is will shift but remain confined within S_is. However, the images of these subgraphs must be connected by edges of dimension $i+2$. By definition, the S_is connected by dimension $i+2$ edges must be siblings in an S_{i+1} with the SCDS $\{2, 3, \dots, i, i+2\}$. According to Theorem 5.6 an S_{i+1} with the SCDS lies in a column or the diagonal of the two dimensional arrangement. Therefore, since u remains invariant, the image G' will be in the S_{i+1} in the column containing u or the S_{i+1} in the diagonal if u belongs to an S_i of the diagonal. ◻

5.4.2 Distance-k set of star clusters

In an SC_{n-1,k_n}, it is possible to obtain a set of as many as $\lfloor\frac{(n-1)k_n}{k_{n-1}}\rfloor$ $SC_{n-2,k_{n-1}}$ disjoint star clusters. There are many possibilities for such a set of star clusters. Figure 5.6 shows some such examples. However, in some of our embedding schemes we will be interested in a obtaining a set of star clusters in which each star cluster has at least one node within a distance of k edges from a node in a given S_{n-2} called the *reference substar*. Such a set of star clusters we shall refer to as a *distance-k set* of star clusters with respect to the reference S_{n-2}.

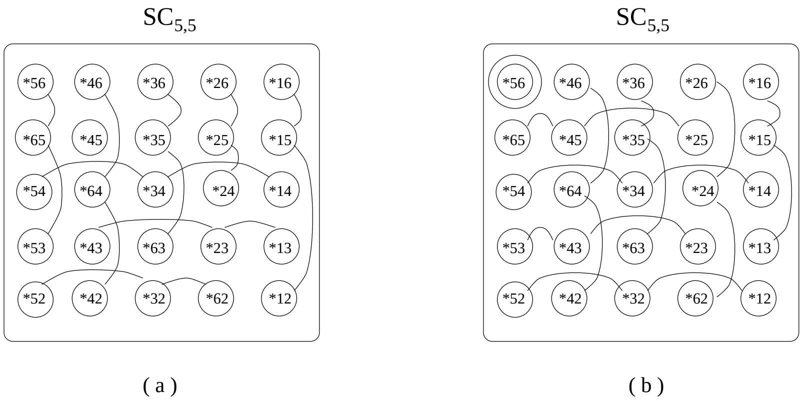


Figure 5.6: Sets of eight $SC_{4,3}$s in $SC_{5,5}$. (a) an arbitrary set, (b) a distance-2 set with reference substar *56. The S_4s are represented with fixed symbols. The S_4s included in an $SC_{4,3}$ are connected by arcs.

THEOREM 5.8 *In an SC_{n-1,k_n} there can be at most $n+k_n-4$ $SC_{n-2,k_{n-1}}$s in a distance-1 set and $2(n+k_n-4)$ $SC_{n-2,k_{n-1}}$s in a distance-2 set.*

PROOF. Let the reference S_{n-2} in an SC_{n-1,k_n} be $*pq$. There are nodes in another S_{n-2}, say $*xy$, in the S_{n-1,k_n} which are within two edges away from nodes in $*pq$, if either $x=p$ or $y=q$. When $x=p$, $(n-3)!$ nodes in $*pq$ which have the first symbol y have dimension n edge to corresponding nodes in $*xy$ (*i.e.* in $*py$). For an S_{n-2}, $*py$, we say that y is the *deciding symbol*. In an SC_{n-1,k_n} the n-th symbol can assume k_n different values. For $x=p$, y can be any of these apart from p and q. Each choice of y gives a distinct $*xy$ for a total of k_n-2 S_{n-2}s which are single edge away from nodes in $*pq$. Similarly, when $y=q$, $*xy$ is single edge (of dimension $n-1$) away from the $(n-3)!$ nodes in $*pq$ which have the first symbol x. The deciding symbol for an S_{n-2}, $*xq$, therefore, is x. In this case, x can assume any of the $n-2$ symbols in the permuting set in SC_{n-1,k_n} other than p and q. This leads to a total of $n-2$ distinct S_{n-2}s that are single edge away from nodes in $*pq$. Over all there are $n+k_n-4$ S_{n-2}s in an SC_{n-1,k_n}, single edge from nodes in the reference S_{n-2}, $*pq$. These S_{n-2}s we shall refer to as *distance-1 key elements* with reference to the given S_{n-2} within the SC_{n-1,k_n}.

On the other hand, there are nodes in $*pq$ from which nodes in $*xy$ are two edges away if either $x=q$ or $y=p$. Here again, when $x=q$ (or $y=p$), nodes in $*xy$ are two edges away from $(n-3)!$ nodes in $*pq$ which have the first symbol y (or x). The *deciding symbol*, therefore, is y (or x) and the dimensions of the edges are n and $n-1$. In this case too there will be $n+k_n-4$ distinct S_{n-2}s which have nodes two edges away from nodes in $*pq$. These

S_{n-2}s we shall refer to as *distance-2 key elements.*

It is a must for an $SC_{n-2,k_{n-1}}$ in a distance-1 set to include at least one distance-1 key element. Therefore there can be at most $n+k_n-4$ $SC_{n-2,k_{n-1}}$s in a distance-1 set. On the other hand, an $SC_{n-2,k_{n-1}}$ in a distance-2 set must include a distance-1 key element or a distance-2 key element. Therefore there can be at most $2(n+k_n-4)$ $SC_{n-2,k_{n-1}}$s in a distance-2 set. $\square$

In a two dimensional arrangement of S_{n-2}s in an S_n, if the 0-th ranked S_{n-2} $*(n-1)(n)$ is taken as the reference substar, the single edge away S_{n-2}s will consist of the elements in the first row and the first column, excepting $*(n)(n-1)$. These we shall refer to as the *first row key elements* and the *first column key elements* respectively. The two edges away S_{n-2}s will consist the second row and the diagonal, excepting $*(n)(n-1)$ again. These we shall refer to as the *second row key elements* and the *diagonal key elements* respectively. See Table 5.2. It should be noted that the *deciding symbol* for a first row key element or a second row key element is the (n-1)-th symbol and that for a first column key element or a diagonal key element is the n-th symbol.

An SC_{n-1,k_n} is isomorphic to any other SC_{n-1,k_n} (Theorem 5.2) and one can be mapped to the other through a sequence of n-th symbol swap transformations. Therefore, while obtaining a distance-k set of $SC_{n-2,k_{n-1}}$s, for convenience we shall assume an SC_{n-1,k_n} to consist of the set $\{S^0_{n-1}, S^1_{n-1}, \ldots, S^{k-1}_{n-1}\}$ and that the reference S_{n-2} is $*(n-1)(n)$. In terms of the two dimensional arrangement, then, the problem of obtaining a distance-1 set of $SC_{n-2,k_{n-1}}$s consists of obtaining required number of $SC_{n-2,k_{n-1}}$s from the rows or the columns such that each contains at least one S_{n-2} from the first row or the first column. Similarly, the problem of obtaining a distance-2 set of $SC_{n-2,k_{n-1}}$ consists of obtaining required number of $SC_{n-2,k_{n-1}}$s from the rows or the columns such that each contains at least one S_{n-2} from the first row, first column, second row or the diagonal. For simplification we ignore the possibility of obtaining an $SC_{n-2,k_{n-1}}$ from the diagonal.

5.4.3 Obtaining a distance-2 set of $SC_{n-2,k_{n-1}}$s

In this section we present methods for obtaining a distance-2 set of $SC_{n-2,k_{n-1}}$s in an SC_{n-1,k_n}. We will be interested in the cases with k_{n-1} values in the range $\frac{k_n}{2} < k_{n-1} \leq n-1$. This range we divide into two subranges for $k_{n-1} \leq k_n$: (1) $max(\frac{k_n}{2}, \frac{n-1}{2}) < k_{n-1} \leq n-1$ and (2) $\frac{k_n}{2} < k_{n-1} < \frac{n-1}{2}$. We also consider two special cases: (1) $k_{n-1} > k_n$, $\frac{n-1}{2} < k_{n-1} \leq n-1$ and (2) $\rho \leq k_n$; $\frac{n-1}{2} < k_{n-1} \leq n-1$, where ρ is the number of $SC_{n-2,k_{n-1}}$s required.

It is easy to see that the maximum possible number of $SC_{n-2,k_{n-2}}$s available in an SC_{n-1,k_n}

is $\rho_{max} = \lfloor \frac{(n-1)k_n - 1}{k_{n-1}} \rfloor$, after leaving one S_{n-2} for the reference substar. In the following we shall refer to the PCDS $SC_{n-2,k_{n-1}}$s as *row clusters* and the SCDS $SC_{n-2,k_{n-1}}$s as *column clusters*.

Case-1: $k_{n-1} \leq k_n$; $max(\frac{n-1}{2}, \frac{k_n}{2}) < k_{n-1} \leq n-1$

By substituting $k_{n-1} = \frac{n-1}{2}$ and $k_{n-1} = \frac{k_{n-1}}{2}$, we get the inequalities $\rho_{max} < 2k_n$ and $\rho_{max} < 2(n-1)$ respectively. Hence, $\rho_{max} < 2\,min(n-1, k_n)$ or $\rho_{max} < (n-1) + k_n$. Therefore, while obtaining $\rho \leq \rho_{max}$ $SC_{n-2,k_{n-1}}$s, if we obtain k_n row clusters, one from each row, we will have to pick $\rho - k_n$ $(< n-1)$ column clusters. This can be done using the algorithm D2_Clusters below. At first we shall present the algorithm for obtaining an arbitrary set of ρ $SC_{n-2,k_{n-1}}$s which will be followed by the modifications required in the algorithm to obtain a distance-2 set.

Algorithm D2_Clusters

1. Mark the last $r = \rho - k_n$ columns to be used for the column clusters. Allocate the elements of the rest $n-1-r$ columns to the row clusters of the respective rows. Therefore $k_{n-1} - (n-1-r)$ elements remain to be allocated to each of the rows.

2. Allocate elements from one row at a time starting with the first row.

   (a) Allocate the elements of a row in the r columns marked for the column clusters one by one as follows.

   - Check if the corresponding row cluster requires any more elements.
     - If yes, check if sufficient number of elements remains in the corresponding column, excluding the element under consideration, to satisfy the cluster of the column.
       * If yes, allocate the element to the row cluster.
       * Otherwise, allocate it to the column cluster.
     - Otherwise, if the column cluster is still not complete, allocate the element to the column cluster.

   (b) Check if there are any unallocated elements in the row.

   - if yes, check for the elements in the row, allocated to the row cluster one by one from right to left. If the column cluster corresponding to the column of the element is still not complete, free the element from the row cluster and allocate it to the column cluster. Allocate one of the free elements in the row to the row cluster.

The star clusters obtained through the above algorithm do not consist a distance-2 set as some of the column clusters do not contain key elements. The algorithm can be made to produce only a distance-2 set of star clusters if the key elements are preallocated. For this, the algorithm can be modified by rewriting the first step as follows.

1. Mark the last $r = \rho - k_n$ columns to be used for the column clusters. Allocate the elements of the rest $n-1-r$ columns to the row clusters of the respective rows.
   - Allocate the first row elements in the last $n-1-k_{n-1}-1$ columns to the respective column clusters.
   - Allocate the second row elements in the remaining $r-(n-1-k_{n-1}-1)$ columns of those marked for the column clusters to the respective column clusters.
   - Allocate the first column element and the diagonal element in each row, excepting the first row, to the respective row clusters.
   - Reserve the first element of the first row as the reference substar.

The correctness of the algorithm is discussed in the appendix B.

Case-2: $k_{n-1} \leq k_n$; $\frac{k_n}{2} < k_{n-1} < \frac{n-1}{2}$.

In this case it is possible to obtain ρ, $k_n + n - 1 < \rho < 2k_n + n - 1$, star clusters by obtaining $2k_n$ clusters from the rows, two from each, and the remaining from the columns, at most one per column. This can be done by using the algorithm of case-1 with the modification that initially the row clusters are considered to consist of $2k_{n-1}$ elements. After obtaining the star clusters each $2k_{n-1}$ row cluster is broken into two row clusters of size k_{n-1} each. In the cases of those in the rows other than the first and the second, the elements should be divided such that a cluster contains either the first column element or the diagonal element.

Case-3: $k_{n-1} > k_n$; $\frac{n-1}{2} < k_{n-1} < n-1$

In this case $SC_{n-2,k_{n-1}}$s can be obtained only from the rows and not from the columns. Moreover, as $k_{n-1} > \frac{n-1}{2}$, only one $SC_{n-2,k_{n-1}}$ can be obtained from each row. Therefore, the number of $SC_{n-2,k_{n-1}}$s that can be obtained in SC_{n-1,k_n} is k_n. The obtaining of the star clusters in this case is therefore trivial.

Case-4: $\rho \leq k_n$; $\frac{n-1}{2} < k_{n-1} \leq n-1$

In this case the number of $SC_{n-2,k_{n-1}}$s required is less than or equal to the number of rows in the SC_{n-1,k_n}. Therefore, the $SC_{n-2,k_{n-1}}$s can be easily obtained by picking one from each of ρ rows.

5.5 Summary

In this chapter we have proposed the concept of star cluster to address the problem of extremely rapid rate of expansion of the star graph. The star cluster concept allows near linear expansion of star graph based interconnection networks. It also allows efficient utilization of such a network. In addition, it retains most of the desirable properties of the star graph. As a result most of the algorithms developed for the star can be implemented on star clusters with minor or no modifications.

We have characterized the paths valid for star clusters and developed a scheme for shortest path routing on the star graph. The restrictions required to be imposed on the communication paradigms discussed in chapter 4 have also been analyzed.

We have also studied some of the topological properties of star cluster. Some of the results of the study are as follows:

1. All $SC_{n,k}$s for given n and k values are isomorphic.
2. It is Hamiltonian for $n > 3$. It is also Hamiltonian for $2 \leq n \leq 3$ when $k > 2$.
3. The diameter of an $SC_{n,k}$, $k > 1$, is same as that of an S_{n+1}.
4. It is bipartite and both the partitions have equal number of nodes.
5. There are n or $n-1$ parallel paths between any two nodes depending upon whether they are of degree n or $n-1$. The lengths of these paths are identical to those in an S_{n+1}.

We have presented a two dimensional arrangement of substars that can be used to recognize star clusters, $SC_{i,k}$s in an S_{i+2} with two different CDSs. This ability is useful in obtaining compact embeddings and in efficient processor allocation in terms of star clusters. In this context, a concept called *distance-k set* of star clusters have also been introduced. Methods for obtaining distance-2 sets of star clusters have been developed. It is useful in developing embedding schemes for certain interconnection structures such as the full binary tree.

Chapter 6

Embedding Schemes

Simulation of one interconnection network in another is an important problem in parallel computing. This problem of simulation is usually formulated as the problem of graph embedding, or simply embedding, where the simulated network is called the *guest graph* and the network on which the simulation is to be carried out is called the *host graph*. The guest graph is said to be *embedded* in the host graph.

DEFINITION 6.1 *Let $G = (V, E)$ and $H = (W, F)$ be two graphs. Let P_H be the set of all paths in H. An embedding of G (a guest graph) in H (a host graph) is a pair of mappings $\phi : V \to W$ and $\rho : E \to P_H$, denoted by (ϕ, ρ), such that for any edge (v_1, v_2) in E, $\rho(v_1, v_2)$ connects $\phi(v_1)$ and $\phi(v_2)$.*

The mapping ρ is often referred to as the routing of the edges.

The quality of an embedding is measured in terms of four parameters, namely, dilation, congestion, expansion and load. The length of the path $p \in P_H$ to which an edge $e \in E$ is mapped is said to be the dilation of the edge e. Often the maximum dilation of any edge $e \in E$ is referred to as the dilation of the embedding (ϕ, ρ). In an embedding it is desirable to have the dilation as small as possible.

Let $P'_H \subseteq P_H$ be the subset of paths to which the set of edges E is mapped. In general, the paths in this subset P'_H may not be edge disjoint. The number of paths in P'_H that share an edge $f \in F$ is the congestion of the edge f. Similarly, the number of paths in P'_H that share a node $w \in W$ is the congestion of the node w. The maximum congestion of any edge $f \in F$ is referred to as the *edge congestion* and the maximum congestion of any node $w \in W$ is referred to as the *node congestion* of the embedding. If the edge congestion of an embedding is one it is said to be edge congestion-free. A node congestion of one implies that the edge congestion is also one. Therefore an embedding with node congestion one is said to be *congestion-free*. It is desirable to keep the congestion of an embedding low as congestion leads to contention in communication and results in additional communication overheads.

The *expansion* of an embedding is defined to be the ratio of the size of the host graph to that of the guest. In case of hierarchical networks such as the hypercube the size of the host graph is taken as that of the subcube used. In case of the star graph, therefore, the tendency is to consider size the substar used as the size of the host graph. However, in a substar many of its immediately lower level substars may remain unused and result in poor utilization of the network. In this thesis we find that star clusters are suitable for performing computations. Processor allocation in terms of star clusters can also be performed efficiently (Chapter 8). Therefore, we consider the size of the host graph in terms of star clusters.

The *load* of a node $w \in W$ is the number of nodes $v \in V$ that are mapped to w. The maximum of the load of any node $w \in W$ is called the load of the embedding. It is desirable to keep the load in an embedding minimum.

In this chapter we present new techniques for embedding cycles, tori, full binary trees and pyramids as guest on star graph as host. Cycles are basic components of many regular interconnection structures such as torus, hypercycles and cube connected cycles. There are also many algorithms that assume cycle or linear array as the interconnection structure. The cycle embedding technique developed by Jwo et al.[JLD91] fails to construct cycles of some specific sizes in optimal star cluster. In Section 2.4.1 we develop cycle embedding techniques for some such cases. We also introduce the concepts of parallel cycles and approximate cycles.

The torus structure is also highly suitable for many scientific and engineering applications which require matrix based computations such as solution of systems of equations and graph algorithms. Commercial systems such as Intel's Paragon have been built with the torus interconnection structure. A large base of algorithms exists for the torus structure. In Section 6.2 we present a torus embedding scheme.

The binary tree is an important interconnection structure for parallel computing. There are many computations which naturally fit into the binary tree structure. Many parallel algorithms have been developed assuming the binary tree structure. The embedding schemes developed earlier [BHOS94, HOS95] for embedding full binary tree are not very efficient. In Section 6.3 we present two new schemes for embedding full binary trees.

The pyramid structure is known to be highly suitable for implementing low and intermediate level operators in vision applications [Te80, Ce86]. In Section 6.4 we present a scheme for embedding pyramid on the star graph.

In all the techniques presented in this chapter, the load of the embeddings is one. Further, the schemes presented here are also applicable to star clusters.

6.1 Embedding of cycles

In this section we discuss methods for the construction of cycles of different lengths in a star graph. We are especially interested in constructing these cycles in minimal star clusters. There are certain situations in which it is required to have a set of disjoint cycles. Sometimes these cycles need to be parallel. A set of cycles are said to be *parallel* if there is a one-to-one correspondence between the nodes in any two such cycles and the distance between any pair of corresponding nodes is bounded by a small constant. A set of such corresponding nodes in a set of parallel cycles we shall refer to as *cycle peer nodes*. To construct a set of parallel cycles in a star graph it will sometimes be necessary that some of these are *approximate cycles*.

Consider the construction of a cycle by joining a set of paths. It is possible that the edges joining the paths have some small dilation. If the dilation of these edges do not exceed some small constant d, we can consider the cycle formed to be of dilation d and of length equal to the sum of lengths of the paths.

DEFINITION 6.2 **Approximate cycle**
A cycle constructed as above is defined as an approximate cycle of dilation d, if the number of dilated edges, or equivalently, the number of paths forming the cycle is small, say less than $\frac{1}{10}$ of the length of the cycle.

One way to construct an approximate cycle of dilation three is through i-th symbol swap transformation of a cycle (exact). Suppose, C^o is an exact cycle that spreads across a set of r sibling S_{i-1}s of an S_i and we take an image of this cycle onto another distinct set of sibling S_{i-1}s in the S_i through i-th symbol swap transformation. Then we get an approximate cycle C^a in which the inter-substar edges of the original cycle C^o are dilated to three edge paths (Theorem 3.2). The number of such dilated edges is only a very small fraction ($\approx \frac{1}{(i-1)!}$)of the edges in C^a. The length of the approximate cycle, C^a is considered to be same as that of C^o. Figure 6.1 illustrates the generation of such an approximate cycle.

In general, in an approximate cycle, generated through i-th symbol swap transformation, the dilated edges may go through some of the nodes already used in the cycle. Due to this, there may be a small congestion on a few edges in such a cycle. Experimentally it has been

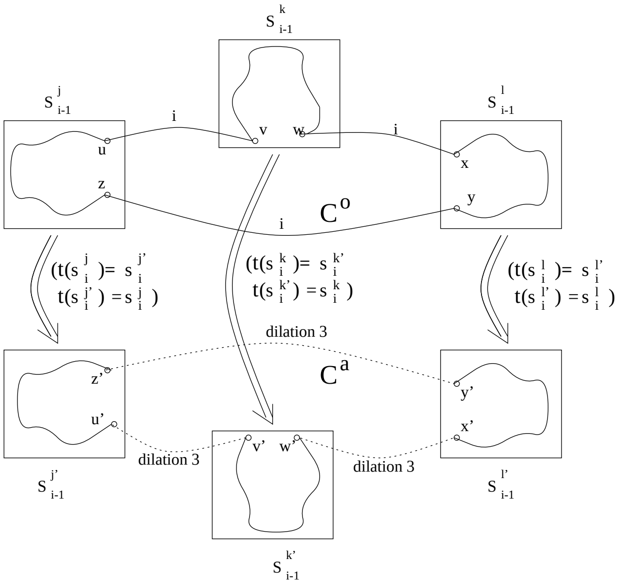


Figure 6.1: Generation of an approximate cycle.

---

found out that this congestion is at most two and it occurs for about 1% of the edges in the graph.

In Jwo's method of construction [JLD91], a cycle of length $(i-1)! > l_{c1} \leq 2(i-1)!$ requires an $SC_{i-1,3}$. Further, for cycles of length $\frac{i}{2}(i-1)! > l_{c2} \leq \frac{i!}{2}$, for i odd, it is possible to construct only one cycle in an S_i with the Jwo's method. This leads to a very unsatisfactory utilization of a star in certain situations. We present a method of constructing a cycle of length l_{c1} in an $SC_{i-1,2}$ and two cycles of length l_{c2} in an S_i. For the second, we also give a method of construction such that the two cycles are parallel.

In our construction it is necessary that a Hamiltonian cycle in an S_i has a special edge called a *[p,q]-adjacent edge* for any pair of symbols $p, q \in < S_i >$. Recollect that $< S_i >$ is the set of permuting symbols of the S_i.

DEFINITION 6.3 **[p,q]-adjacent edge**
Let (u, v) be an edge in a star graph such that the first symbol in u is p and that in v is q. The edge (u, v) is said to be a [p,q]-adjacent edge.

For example, in an S_4 the edge (2413, 3412) is a [2,3]-adjacent edge. It is obvious that if (u, v) is a [p,q]-adjacent edge in S^j_i and p, q are the (i+1)-th (fixed) symbols in S^k_i and S^l_i respectively, then u is an (i,k)-centre and v is an (i,l)-centre.

LEMMA 6.1 *In the construction of a Hamiltonian cycle in an S_i from the Hamiltonian cycles of S_{i-1}s using Jwo's method, at most two [p,q]-adjacent edges, for any (p,q) symbol pair, are deleted from the collection of edges in the Hamiltonian cycles of the sibling S_{i-1}s.*

PROOF. The construction of a Hamiltonian cycle in an S_i through Jwo's method consists of combination of triangle and square formation of the Hamiltonian cycles in the constituent S_{i-1}s as shown in Figure 2.1 in Section 2.4.1. It may be observed that at most one square formation is required in this construction. Let us first consider a triangle formation of the sibling S_{i-1}s as shown in Figure 2.1(a). In the illustration the deleted edges are (u^j, v^j), (u^k, v^k) and (u^l, v^l). It can be seen that (u^j, v^j) is an $[s_i^k, s_i^l]$-adjacent edge, (u^k, v^k) is an $[s_i^j, s_i^l]$-adjacent edge and (u^l, v^l) is an $[s_i^j, s_i^k]$-adjacent edge. Hence, no two deleted edges are [p,q]-adjacent edges for the same (p,q) pair.

Now let us consider a construction with square formation as illustrated in Figure 2.1(b). Here the deleted edges are (u^j, v^j), (u^k, v^k), (u^l, v^l) and (u^m, v^m). It can be seen that the edges (u^j, v^j) and (u^l, v^l) are $[s_i^k, s_i^m]$-adjacent edges and (u^k, v^k) and (u^m, v^m) are $[s_i^j, s_i^l]$-adjacent edges. Hence, in the square formation there are two pairs of deleted edges with identical (p, q) symbol pairs. Clearly, in a general construction, Figure 2.1(c) and Figure 2.1(d), no more than two deleted edges can have identical (p,q) symbol pair. $\square$

6.1.1 Hamiltonian cycle with a given edge

Consider the problem of constructing a Hamiltonian cycle in an S_i that contains a specific edge (u, v). Let d be the dimension of (u, v) in S_i. Assume, H is a Hamiltonian cycle in the S_i constructed using Jwo's method and (u, v) is not in H. Let (w, x) be an edge of dimension d [1] in H. Obtain a symbol swap transformation image H' of H such that w in H maps to u in H'.

THEOREM 6.1 *H' is a Hamiltonian cycle in the S_i that contains the edge (u, v).*

PROOF. From Lemma 3.1 we know that the symbol swap transformation is an isomorphism on the star graph. Therefore H' is a cycle of the same length as H. Also according to Lemma 3.1 the image of the edge (w, x) in H' will have the same dimension as that of (w, x) in H, that is d. Therefore the image of (w, x) will be (u, v). Hence, H' contains the edge (u, v). Moreover, the symbol swap transformation required to map w to u will not involve any of the fixed symbols in the S_i as both the nodes are contained in it. Therefore, on application of this transformation

[1] Note that a Hamiltonian cycle in an S_i must contain edges of each dimension in the S_i. Otherwise, the cycle will be confined to an S_{i-1} or a lower level substar.

on a node in S_i will map it to a node in S_i. Thus, H' will be contained within the S_i. In other words, H' is a Hamiltonian cycle in the S_i containing the edge (u, v). □

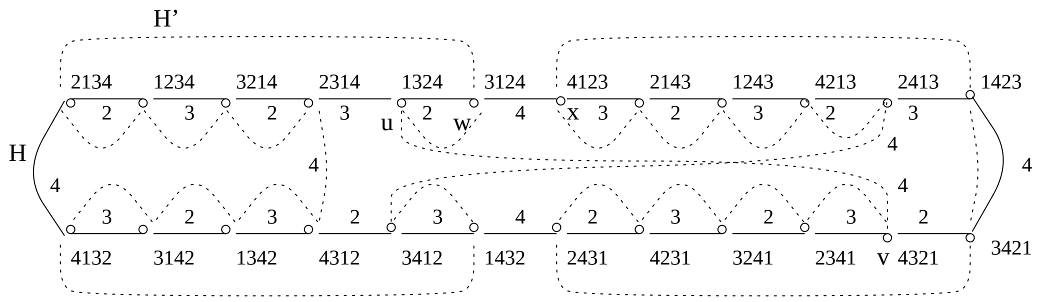


Figure 6.2: Generation of Hamiltonian cycle H' with the edge $(1324, 4321)$ using symbol swap transformation $(t(1) = 3, t(3) = 1)$ of a Hamiltonian cycle H in an S_4. The dotted arcs form the cycle H'.

Figure 6.2 illustrates the construction of a Hamiltonian cycle in S_4 using the above method. Here, H is the initial Hamiltonian cycle. It is desired to have a Hamiltonian cycle containing the edge $(1324, 4321)$ which is of dimension 4. The edge (3124,4123) in H is of dimension 4. Consider the 2-symbol swap transformation $(t(1) = 3,\ t(3) = 1)$ which maps the node 3124 to the node 1324. If this transformation is applied on H the node 4123 will map to the node 4321 and we get the desired Hamiltonian cycle H' as shown in the figure.

THEOREM 6.2 *In an $S_i, i \geq 3$, a Hamiltonian cycle, constructed by either Jwo's method or the modified method given above, has [p,q]-adjacent edges for any $p, q \in < S_i >$.*

PROOF. The proof consists of two parts. The first part gives the proof for a Hamiltonian cycle constructed using Jwo's method. In the second part, the proof is extended for a Hamiltonian cycle constructed using the modified method given above.
Part-1:
Let η_i be the least number of [p,q]-adjacent edges in a Hamiltonian cycle in an S_i for any $p, q \in < S_i >$. It can be observed that an S_3 has only one Hamiltonian cycle and it has two [p,q]-adjacent edges for any $p, q \in < S_3 >$. That is, $\eta_3 = 2$.

Let us consider the general case of $i > 3$. In all the S_{i-1}s of an S_i, except the two in which p or q is the fixed (i-th) symbol, have both p and q in their respective $< S_{i-1} >$s. Therefore, in the S_i, there are $i - 2$ S_{i-1}s having $[p, q]$-adjacent edges. By assumption, a Hamiltonian cycle in each of these S_{i-1}s will have at-least η_{i-1} $[p, q]$-adjacent edges. A Hamiltonian cycle in the S_i is constructed from these S_{i-1} Hamiltonian cycles and at most 2 $[p, q]$-adjacent edges are deleted in the process (Theorem 6.2). Therefore, $\eta_i \geq \eta_{i-1}(i-2)-2$. We know that $\eta_3 = 2$. Therefore it can be easily verified that $\eta_i > 0$, for $i \geq 3$.

Part-2:
We shall now prove that the property of having [p,q]-adjacent edges for any $p, q \in< S_i >$ holds for a Hamiltonian cycle obtained by the modified method. The proof is by contradiction. Let t be the transformation operator used for obtaining the Hamiltonian cycle H' from H, obtained using Jwo's method, and let t^{-1} be the inverse transformation. Let us assume that there exists two symbols $a, b \in< S_i >$ such that there is no [a,b]-adjacent edge in H'. For this assumption to be true, according to the definition of t, it is necessary that there is no [$t^{-1}(a)$,$t^{-1}(b)$]-adjacent edge in H. But $t^{-1}(a), t^{-1}(b) \in< S_i >$. Therefore, it contradicts the result of Part-1 above. □

The significance of Theorem 6.2 is that we can construct a Hamiltonian cycle in an S_i^j, containing any given edge, which has the property that given any two distinct siblings S_i^k and S_i^m other than S_i^j, we can find at least one edge in it whose one end point is an (i,k)-centre and the other end point is an (i,m)-centre. The method of construction of a cycle involving two sibling substars discussed in the following section uses this fact. It is also used in the method of construction of two cycles in an S_i that share an S_{i-1}. The latter is discussed in Section 6.1.3.

6.1.2 Construction of a cycle in an $SC_{i-1,2}$

In this section we present a method for obtaining even cycles of length l_c, $(i-1)! < l_c \leq 2(i-1)!$, for $i > 4$, in an $SC_{i-1,2}$. Recall that Jwo's method requires at-least three sibling S_{i-1}s for constructing such a cycle. Let S_{i-1}^j and S_{i-1}^k be the two sibling substars in which the cycle is to be obtained. The construction algorithm **Cycle_2Si** is given below (refer to Figure 6.3 for illustration):

ALGORITHM 6.1 **Cycle_2Si**

1. *Let S_{i-2}^p and S_{i-2}^q be two sibling S_{i-2}s in S_{i-1}^j both of which have (i-1,k)-centres. Let u be an (i-1,k)-centre in S_{i-2}^p and v be an (i-2,q)-centre in S_{i-2}^p adjacent to u. Obtain a Hamiltonian cycle H^p in S_{i-2}^p with the edge (u, v) in it.*

2. *Let w be the (i-2,p)-centre in S_{i-2}^q corresponding to u and let x be (i-1,k)-centre adjacent to w. Obtain a Hamiltonian cycle H^q in S_{i-2}^q with the edge (x, w) in it.*

3. *Let $S_{i-2}^{r_1}, S_{i-2}^{r_2}, ..., S_{i-2}^{r_{i-2}}$ be the rest of the sibling S_{i-2}s in S_{i-1}^j. Obtain appropriate cycles*

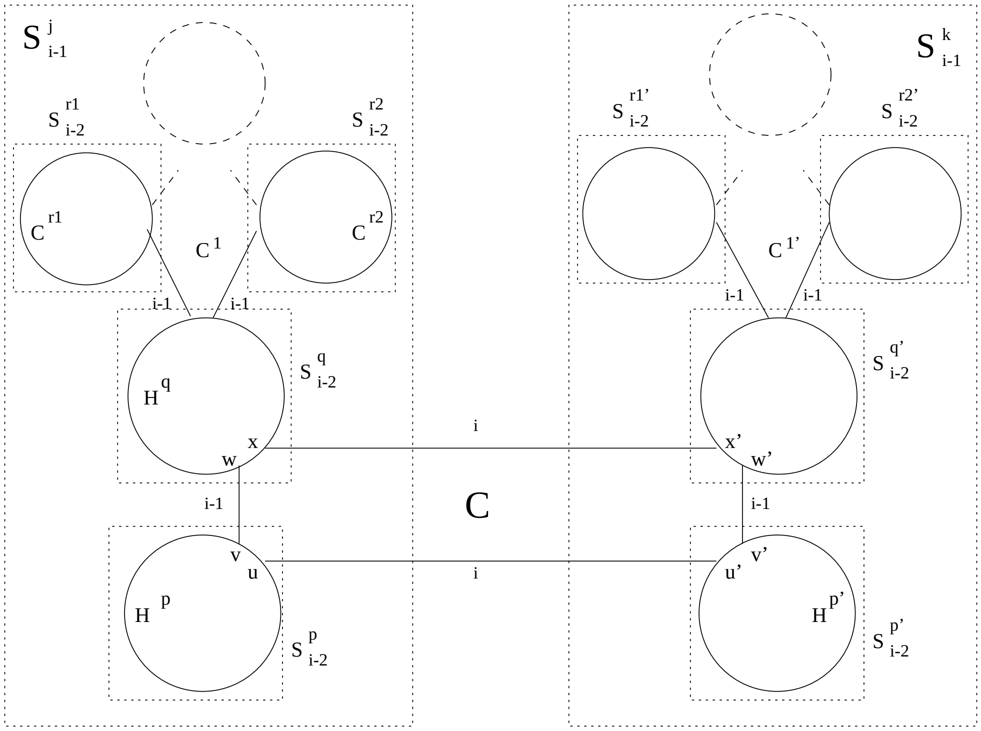


Figure 6.3: Construction of a cycle in an $SC_{i,2}$s.

C^{r_1}, C^{r_2},...,$C^{r_{i-2}}$ in these substars such that they can be combined with H^q in S^q_{i-2} to form a single cycle C^1 of length $\frac{l_c}{2} - (i-2)!$.

4. *Take i-th symbol swap images of H^p and C^1 in S^k_{i-1} to obtain the image cycles $H^{p'}$ and $C^{1'}$. Let u', v', w' and x' be the image nodes of u, v, w and x respectively. By definition, u' and x' are (i-1,j)-centres corresponding to u and x respectively. Also v' and w' are adjacent nodes belonging to $H^{p'}$ and $C^{1'}$ respectively.*

5. *Combine H^p, $H^{p'}$, C^1 and $C^{1'}$ to form a cycle C by deleting the edges (u,v), (w,x), (u',v'), (w',x') and including the edges (v,w), (v',w'), (u,u'), (x,x').*

THEOREM 6.3 *The algorithm Cycle_2Si constructs an even cycle of length l_c, $(i-1)! < l_c \leq 2(i-1)!$, in an $SC_{i-1,2}$.*

PROOF. Let us look at the validity of the algorithm step by step. According to Proposition 3.2, all S_{i-2}s, except one, in a substar S^j_{i-1} has $(i-1,k)$-centres, for $k \in \{0,1,...,i-1\} \backslash j$. Therefore it is possible to find two S_{i-2}s in S^j_{i-1} having $(i-1,k)$-centres in step-1. It is also straight forward to find the nodes u and v in S^p_{i-2} and to obtain the Hamiltonian cycle H^p. The fact that S^q_{i-2} has $(i-1,k)$-centre ensures that u and v are distinct nodes. Finding the $(i-1,k)$-centre x adjacent to w in S^q_{i-2} is possible due to the Proposition 3.3(d). The step-3 is based on Theorem 6.2 and Jwo's cycle construction method (refer Section 2.4.1). These allow that H^q is combined with the cycles (of any length $\leq (i-2)!$) in the sibling S_{i-2}s in S^j_{i-1} through method of triangle and square formation of Jwo where H^q participates in all the formations. Moreover, there are sufficient number of nodes in these substars to form the cycle C^1 of length $\frac{l_c}{2} - (i-2)!$. The step-4 is self explanatory. Note that $H^{p'}$ and $C^{1'}$ are isomorphic images of H^p and C^1 and therefore are of lengths $(i-2)!$ and $\frac{l_c}{2} - (i-2)!$ respectively. Thus the cycle C obtained by combining H^p, $H^{p'}$, C^1 and $C^{1'}$ will be of length l_c. □

6.1.3 Construction of two cycles in an S_i sharing an S_{i-1}

In this section we discuss methods of constructing two disjoint cycles in an S_i. The length of the cycles, l_c, is such that they share an S_{i-1}. That is, $\lfloor \frac{i}{2} \rfloor (i-1)! < l_c \leq \frac{i!}{2}$. Necessarily, i is an odd integer here. First we present a scheme for construction of two exact (not approximate) disjoint cycles of length l_c. This is followed by a modified algorithm to construct two parallel cycles of length l_c in S_i. One of these two cycles is approximate, that is, a small fraction of the edges in one of the cycles have a small dilation. This parallel cycle construction scheme will be useful in our torus embedding scheme to be discussed later in this chapter. In what follows, we present scheme for the construction of two disjoint cycles in S_i, for $i > 5$, in algorithm Cycle_ShSi_General and the modified algorithm for the construction of two parallel cycles in the S_i in algorithm ParCycle_ShSi_General. The corresponding schemes for the special case of $i = 5$ is presented in the algorithms Cycle_ShSi_5 and ParCycle_ShSi_5 respectively.

ALGORITHM 6.2 **Cycle_ShSi_General**
{ Refer to Figure 6.4 for illustration.}

1. *Take S^s_{i-1} in the S_i to be shared by the two cycles.*
2. *Group the rest of the S_{i-1}s in the S_i in two sets A and B each containing $\frac{i-1}{2}$ S_{i-1}s. Also group the S_{i-2}s in S^s_{i-1} into two sets D and E each containing $\frac{i-1}{2}$ S_{i-2}s.*
3. *Let S^a_{i-1} and S^b_{i-1} be two of the members of A and (u,v) be an edge in $S^e_{i-2} \in D$ such that u is an (i-1,a)-centre and v be an (i-1,b)-centre. Construct a cycle C^1 of length $\frac{i-1}{2}(i-2)!$ using Jwo's method by combining Hamiltonian cycles in the S_{i-2}s of the set D such that it has the edge (u,v).*

4. *Construct appropriate cycles in the S_{i-1}s in the set A such that they can be connected together with C^1 as shown in Figure 6.4 to form a cycle C^o of the desired length l_c. To achieve the desired cycle length, the length of cycle in one of the S_{i-1}s other than S^a_{i-1} or S^b_{i-1} can be appropriately chosen and the rest can be made Hamiltonian cycles.*

5. *Repeat the steps 3 and 4 above to construct the second cycle C^a in the substars of the sets B and E.*

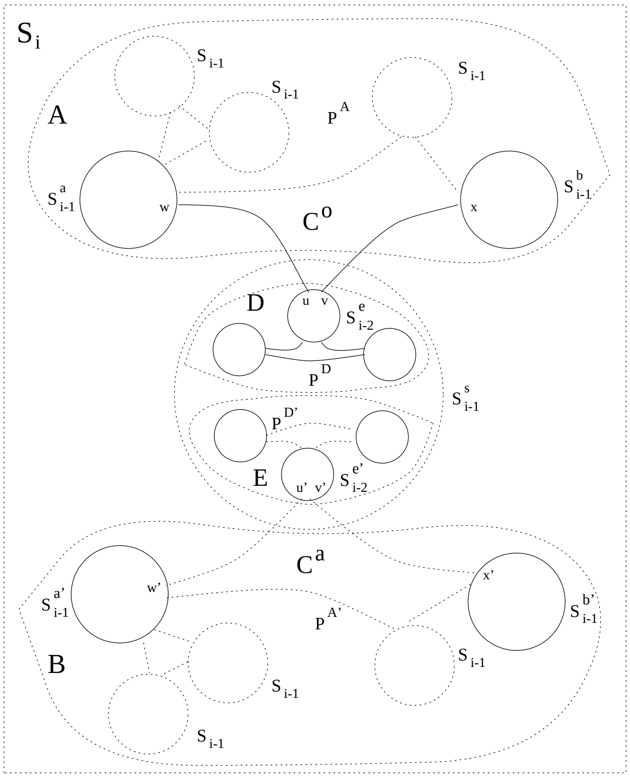


Figure 6.4: Construction of two disjoint cycles in an S_i.

THEOREM 6.4 *The algorithm Cycle_ShSi_General constructs two disjoint cycles of length l_c, $\lfloor\frac{i}{2}\rfloor(i-1)! < l_c \leq \frac{i!}{2}$, in an S_i.*

PROOF. The steps 1 and 2 in the algorithm are straight forward. The step 3 follows from Proposition 3.2 and the Jwo's cycle construction method. The step-4 can be performed as follows. Construct Hamiltonian cycles in $\frac{i-1}{2} - 1$ of the S_{i-1}s including S^a_{i-1} and S^b_{i-1} of the set A. Construct a cycle of length $l_c - (\frac{i-1}{2} - 1)(i-1)! - \frac{i-1}{2}(i-2)!$ in the remaining one S_{i-1}

in the set A. Align the cycles in S^a_{i-1} and S^b_{i-1} such that they can be connected with C^1 to make a single cycle in a triangle formation with (u, v) in C^1 as an edge to be removed. Merge the cycles in the rest of the S_{i-1}s to it by combining them to the Hamiltonian cycle in S^a_{i-1} through triangle and square formations which is allowed by Theorem 6.2. Step-5 repeats this process with the substar sets B and E. Since B and E are distinct from A and B and have equal numbers of S_{i-1}s and S_{i-2}s respectively a second and distinct cycle is obtained. □

The algorithm ParCycle_ShSi_General below is a modification of the above algorithm to obtain two parallel cycles of length l_c in an S_i.

ALGORITHM 6.3 **ParCycle_ShSi_General**

1. *Same as in Algorithm 6.2*
2. *Same as in Algorithm 6.2*
3. *Same as in Algorithm 6.2*
4. *Same as in Algorithm 6.2*
5. *Let P^A be the section of C^o that lies in the set A and P^D be the section of the cycle that lies in the set D. Take an i-th symbol swap transformation image $P^{A'}$ of P^A on to the S_{i-1}s in set B by selecting a distinct member of B for each member in set A. Similarly, take an (i-1)-th symbol swap transformation image $P^{D'}$ of P^D on to the S_{i-2}s in set E by selecting a distinct member of E for each member in set D. Combine $P^{A'}$ and $P^{D'}$ to get the approximate cycle C^a.*

Since the cycle obtained in the sets B and E is an i-th symbol swap transformation (i-th and (i-1)-th) of the cycle in the sets A and D, the corresponding image nodes in the two cycles are within a distance bound of three. Thus they are parallel. The cycle in the substar sets B and E is approximate.

THEOREM 6.5 *Dilation of the inter-substar edges of approximate cycle C^a constructed through the algorithm ParCycle_ShSi_General is at most five.*

PROOF. From Theorem 3.2 we know that the dilation of the image path sections (refer to Figure 6.4) $P^{A'}$ and $P^{D'}$ of C^a is three. We need to prove that the edges (w', u') and (v', x') that connect $P^{A'}$ and $P^{D'}$, in Figure 6.4, to form C^a are of dilation at most five.

From the construction we know that:
$w = u(1\ i)$; $v = u(1\ l_1)$, where $l_1 < i-1$, so that u and v belong to the same S_{i-2};
$x = v(1\ i)$; $w' = w(l_2\ i)$; $u' = u(l_3\ i-1)$; $v' = v(l_4\ i-1)$; $x' = x(l_5\ i)$.
Therefore, we can write

$$w' = u(1\ i)(l_2\ i) = u'(l_3\ i-1)(1\ i)(l_2\ i) = u'(l_3\ i-1)(1\ l_2\ i) = u' g_{l_3} g_{i-1} g_{l_3} g_{l_2} g_i \quad \dots (1)$$

$$x' = v(1\ i)(l_5\ i) = v'(l_4\ i-1)(1\ i)(l_5\ i) = v'(l_4\ i-1)(1\ l_5\ i) = v' g_{i-1} g_{l_4} g_{i-1} g_{l_5} g_i \quad \dots (2)$$

Thus, the edges (u', v') and (v', x') are of dilation five. □

In the expressions (1) and (2) above, if we have $l_2 = l_3$ and $l_5 = i-1$, we get $w' = u' g_{l_2} g_{i-1} g_i$ and $x' = v' g_{i-1} g_{l_4} g_i$ which reduces the dilation of the approximate cycle C^a to three. This is stated in the following theorem.

THEOREM 6.6 *Dilation of the inter-substar edges of the approximate cycle C^a obtained through the algorithm ParCycle_ShSi_General can be reduced to three through appropriate matching of the substars.*

PROOF. Refer to the construction illustrated in Figure 6.4. Without loss of generatity, let us assume u to be the identity node so that S^0_{i-1} is the shared S_{i-1}. Let $v = u(1\ m)$, $m < i-1$, so that $u, v \in S^0_{i-2}$ in S^0_{i-1}. According to the construction, $w = u(1\ i)$ and $x = v(1\ i)$ so that $w \in S^{i-1}_{i-1}$ and $x \in S^{i-m}_{i-1}$ as $w_i = 1$ and $x_i = m$.

Now, let $w' = w(l\ i)$ for some $l < i-1$, $l \neq m$ and let $x' = x(i-1\ i)$, $u' = u(l\ i-1)$, and $v' = v(l\ i-1)$ so that $l_2 = l_3 = l_4 = l$ and $l_5 = i-1$. Then $u', v' \in S^{i-1-l}_{i-2}$ in S^0_{i-1}, $w' \in S^{i-l}_{i-1}$, $x' \in S^1_{i-1}$ as $u'_{i-1} = v'_{i-1} = l$, $w'_i = l$, and $x'_i = i-1$.

If we choose l and m values in the range $[2, i-2]$ such that $l \neq m$, then we will have S^{i-1-l}_{i-2} distinct from S^0_{i-2} and the S_{i-1}s S^0_{i-1}, S^{i-1}_{i-1}, S^{i-m}_{i-1}, S^{i-l}_{i-1}, and S^1_{i-1} also distinct from each other. It is trivial to make such a choice as $i > 5$. Thus the construction of the cycles is possible.

Now, $w' = w(l\ i) = u(1\ i)(l\ i) = u'(l\ i-1)(1\ i)(l\ i) = v'(1\ l)(1\ i-1)(1\ i) = u' g_l g_{i-1} g_i$ and $x' = x(i-1\ i) = v(1\ i)(i-1\ i) = v'(l\ i-1)(1\ i)(i-1\ i) = v'(1\ i-1)(1\ l)(1\ i) = v' g_{i-1} g_l g_i$. That is, the edges (u', w') and (v', x') are of dilation three. Thus we have the theorem. □

The corresponding algorithm for construction of the parallel cycles for the special case of $i = 5$, that is, $48 < l_c \leq 50$, is presented below. The construction is illustrated in Figure 6.5.

ALGORITHM 6.4 **ParCycle_ShSi_5**

1. *Break l_c into three components n_1, n_2, and n_3 such that $n_1 = 12$ $n_2 = 24$ and $n_3 = l_c - 36$.*

2. *Obtain two node disjoint 12-cycles C^1 and $C^{1'}$, in S_4^2 using the method given in the following section, such that they are $(t(2) = 4,\ t(4) = 2)$ swap transformation images of each other. Let edge (u, v) be an edge on C^1, where u is a (4,0)-centre and v is a (4,1)-centre.*

3. *Obtain a Hamiltonian cycle C^2 in S_4^0 such that it contains an edge (w, x) where w is the (4,2)-centre corresponding to u, and x is a (4,1)-centre.*

4. *Obtain a cycle C^3 of length n_3 in S_4^1 such that the edge (y, z) is on it, where y is the (4,0)-centre corresponding to x, and z is the (4,2)-centre corresponding to v.*

5. *Connect the cycles C^1, C^2 and C^3 by including the inter-S_4 edges (u, w), (v, z), and (x, y) and deleting the edges (u, v), (w, x), and (y, z) to obtain a cycle C^0 of length l_c.*

6. *Take 5-th symbol swap transformation image of C^2 in S_4^0 on to S_4^3 and C^3 in S_4^1 onto S_4^4 and the image through the 2-symbol swap transformation $(t(2) = 4,\ t(4) = 2)$ of C^1 on to $C^{1'}$ in S_4^2 to construct the approximate image cycle C^a.*

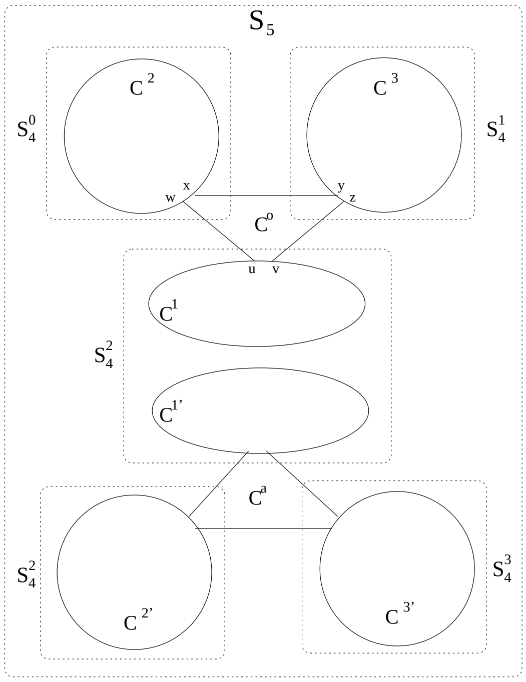


Figure 6.5: Construction of two disjoint cycles in an S_5.

THEOREM 6.7 *The algorithm ParCycle_ShSi_5 constructs two distinct l_c length cycles C^o and C^a. The cycle C^o is of unit dilation. The second cycle C^a has three edges of dilation three while the rest of the edges are of unit dilation.*

PROOF. In the algorithm the step-1 is trivial and the step-2 follows from construction of 12-cycles in S_4 as discussed in Section 6.1.4. Validity of step-3 and step-4 follows from Theorem 6.2. Step-5 is straight forward. We have to prove that the inter-S_4 edges in C^a are of dilation three. That is, we have to show that the images of the edges (u, w), (v, z) and (x, y) in C^a are paths of length not exceeding three.

Let the images of the nodes u, v, w, x, y and z be u', v', w', x', y' and z' respectively and let the notation p_s^v to denote the position of a symbol s in a permutation v. Then,
$u = w(1\ 5) = w(1\ p_5^w)$ since 5 is the 5-th symbol in the nodes of S_4^0.
$w = w'(p_2^w\ p_5^w)$
$u' = u(p_2^u\ p_4^u) = w(1\ p_5^w)(p_2^u\ p_4^u) = w'(p_2^w\ p_5^w)(1\ p_5^w)(p_2^u\ p_4^u) = w'(1\ p_5^w)(1\ p_2^w)(p_2^u\ p_4^u)$
But $p_2^u = p_2^w$. Therefore, $u' = w'(1\ p_5^w)(1\ p_4^u)(1\ p_2^u)$
Hence, the length of the shortest path between the nodes u' and w' is three.
Similarly, it can be shown that the length of the shortest paths between the nodes v' and z' and that between the nodes x' and y' are also three.
Thus there are three edges of dilation three in C^a, the rest edges are of unit dilation. □

In the above constructions, a node in C^o and its image node in C^a become cycle peer nodes. By Lemma 3.3, the distance between such a pair of cycle peer nodes is at most three.

6.1.4 Construction of multiple cycles in S_4

In this section we consider the construction of cycles of length greater than six but less than or equal to twelve. Two S_3s should be large enough to accommodate the nodes of such a cycle. However, it is not possible to combine two S_3s to form such a cycle. Therefore, special methods, described below are required for construction of such cycles. Let $i, j, k \in \{2, 3, 4\}$ and $i \neq j \neq k$.

- Two node disjoint 12-cycles can be obtained by operating on two nodes u and v, where $v = ug_j$, with the generator sequence g_i, g_j, g_k four times in succession. This construction has the property that the two cycles obtained are 2-symbol swap transformation images of each other. For example, if we construct the two cycles with $u = 1234$ and $i = 2, j = 3, k = 4$, they are $(t(2) = 4,\ t(4) = 2)$ swap transformation images of each other. If the image node pairs are made cycle peer nodes, the distance between them will be at most three.

- Two node disjoint 10-cycles can be obtained by operating on two nodes u and v, where $v = ug_ig_kg_i$, with the generator sequence $g_i, g_j, g_k, g_i, g_k, g_i, g_k, g_j, g_i, g_j$. Such a pair of disjoint 10-cycles in an S_4 has the property of being 2-symbol swap transformation images of each other. If the image node pairs in the cycles are made peer node pairs the distance between any two peer nodes will be three.

- Three node disjoint 8-cycles with dilation four peer node adjacency can be obtained by operating on three nodes u, v, and w with the generator sequences $\{g_i, g_j, g_i, g_k, g_i, g_j, g_i, g_k\}$, $\{g_k, g_j, g_k, g_i, g_k, g_j, g_k, g_i\}$, and $\{g_j, g_i, g_j, g_k, g_j, g_i, g_j, g_k\}$ respectively, where $u = vg_ig_j$ and $v = wg_ig_k$ and $i, j, k \in \{2, 3, 4\}$, $i \neq j \neq k$. u, v, and w will be the first nodes in their respective cycles.

6.2 Embedding of torus

In this section we present a scheme for embedding multidimensional torus on the star graph. We first present the scheme for embedding a two dimensional torus. The scheme is then generalized for embedding multidimensional torus.

The scheme for embedding of a $n_r \times n_c$-torus in a star graph, presented here, consists of obtaining a set of n_c node disjoint cycles (some of which may be *approximate cycles*) of length n_r, following the cycle construction schemes discussed in Section 6.1. These cycles form the columns of the torus. The column cycles are placed in a cyclic arrangement such that the cycle peer nodes in adjacent cycles are within bounded (four edges) distance. This results in a dilation four embedding of the torus in the star graph with each set of cycle peer nodes forming a row cycle of the torus.

6.2.1 Generation of the first cycle set

The first step in the embedding is to determine the *optimal substar* for a cycle of length n_r. Substar S_i is said to be optimal if $(i-1)! < n_r \leq i!$. It may be possible to obtain one or more cycles in an optimal substar. The cycles in an optimal substar are referred to as *sibling cycles*. Initially, one set of such sibling cycles are obtained in an optimal substar S_i. This set is referred to as the *first cycle set*. To achieve a good bound on the expansion of the embedding we try to have as many cycles as possible in the first cycle set. For this, the method for the construction of the first cycle set is chosen appropriately depending upon the cycle length. The various cases are discussed below.

Let λ be the number of S_{i-1}s required for the construction of a cycle, *i.e.* $(\lambda-1)(i-1)! < n_r \leq \lambda(i-1)!$.

Case-1: $\frac{i!}{2} < n_r \leq i!$, *i.e.* $\lambda > \frac{i}{2}$.
This is the case where S_i can accommodate only one cycle of length n_r. Jwo's method can be used, in this case, for construction of the first cycle.

Case-2: $i \geq 5$, $\lambda \leq \lfloor \frac{i}{2} \rfloor$.
This is a case where the optimal substar can accommodate at least two cycles without any S_{i-1} being shared by the cycles and λ sibling S_{i-1}s are required for the construction of one of the cycles. Therefore, if $\lambda > 2$, use Jwo's method to obtain the first cycle, call it the *original cycle*, of length n_r in an $SC_{i-1,\lambda}$s. If $\lambda = 2$, use the method discussed in the Subsection 6.1.2 to obtain an original cycle of length n_r in an $SC_{i-1,2}$. By i-th symbol swap transformation construct $\lfloor \frac{i}{\lambda} \rfloor - 1$ approximate cycles in that many distinct $SC_{i-1,\lambda}$s in the S_i. Thus a total of $\lfloor \frac{i}{\lambda} \rfloor$ n_r-length sibling cycles are obtained in the first set. A node in the original cycle and its image nodes in the image cycles become *cycle peer nodes.*

Case-3: $i \geq 5$, i odd and $\lfloor \frac{i}{2} \rfloor (i-1)! < n_r \leq \frac{i!}{2}$, $\lambda = \lceil \frac{i}{2} \rceil$.
This is a case where S_i can accommodate two n_r length cycles but they must share one of the S_{i-1}s. Therefore, construct the first set of two cycles of length n_r in an S_i using the method discussed in the Subsection 6.1.3.

Case-4: $i = 4$ and $n_r \leq \frac{i!}{2}$, *i.e.* n_r =8, 10, or 12. In this case use the methods given in Section 6.1.4 for construction of multiple cycles of length n_r in an S_4.

From the construction and the Theorem 3.3, it is evident that the distance between the cycle peer nodes of any two sibling cycles is at most four. Hence, we shall say that sibling cycles are *adjacent with dilation four* or simply, *adjacent* to each other.

6.2.2 Generation of additional cycles

The first set has ρ (say) cycles contained in optimal substar S_i. The value of n_r determines S_i and ρ, $\rho \leq \frac{i!}{n_r}$. We require a total of n_c cycles each of length n_r corresponding to the column cycles of the torus. If $\rho < n_c$, the additional cycles are obtained as follows. Through substar *peer node imaging* (refer to Section 3.4.1) generate additional cycles in sibling S_is within an S_{i+1}. The same imaging technique may be used to generate additional number of cycles in sibling S_{i+1}s and higher levels of substars, if necessary, for obtaining a total of n_c cycles. The peer nodes in the substars become cycle peer nodes. The n_c n_r-length cycles thus generated shall be referred to as the column cycles as they will form the columns of the torus.

DEFINITION 6.4 **Peer cycles**
Let C^j and C^k be two cycles such that they are substar peer node images of each other in sibling S_ms. C^j and C^k are said to be a pair of peer cycles in S_ms.

According to Theorem 3.3, the distance between the cycle peer nodes of any two peer cycles is at most four. As in the case of sibling cycles for a pair of peer cycles too we shall say that they are *adjacent with dilation four* or simply, *adjacent*.

6.2.3 Arrangement of cycles

In order to embed the $n_r \times n_c$-torus we now need to arrange the n_c column cycles each of length n_r in a cyclic adjacency arrangement. A linear arrangement of k cycles $C_0, C_1, \ldots, C_j, \ldots, C_{k-1}$, is said to be a cyclic adjacency arrangement of the cycles if a cycle C_j is adjacent to $C_{(j+1) mod\ k}$, for $0 \leq j < k$. In the cyclic adjacency arrangement of the column cycles each set of cycle peer nodes in the n_c column cycles form a row cycle of the torus with dilation four.

DEFINITION 6.5 **Adjacency graph of column cycles**
Let G be a graph in which the column cycles form the vertices. Two vertices in G are connected if the corresponding cycles are adjacent. G is defined to be an adjacency graph of the cycles.

A cyclic adjacency arrangement of column cycles corresponds to a Hamiltonian cycle in the adjacency graph of the cycles. Two vertices in an adjacency graph are said to be siblings if the corresponding cycles are siblings. They are said to be peers if the corresponding cycles are peers.

DEFINITION 6.6 **Non-critical vertex**
A vertex in a path of an adjacency graph of cycles is said to be non-critical if both of its adjacent (in the path) vertices are either its siblings or its peers (at the same substar level).

A non-critical vertex can be dropped with out causing a break in the path.

LEMMA 6.2 *Assuming S_i to be the optimal substar for an column cycle, it is possible to construct a Hamiltonian path, with any two given vertices as end points, on the adjacency graph of the column cycles in an $SC_{l,k}$, for $l \geq i$, $k \geq 1$.*

Proof. Let u and v be the two given vertices. We prove the lemma by induction on the number of column cycles. Three different basis cases arise depending on the value of ρ.

Case-1: $\rho > 2$; $l = i$ and $k = 1$.
Since the cycles within the optimal substar are siblings, the corresponding adjacency graph is complete. Therefore, the lemma holds.

Case-2: $\rho = 2$; $l = i + 1$, where $i \geq 4$, and $k = 1$.
The condition of $i \geq 4$ is reasonable because S_4 is the smallest star that can have two cycles. However it excludes the cases of tori with number of column cycles less than ten for $\rho = 2$. These will be considered as special cases in Lemma 6.3. Figure 6.6 shows the construction of Hamiltonian paths in the adjacency graph corresponding to the cycles in an S_{i+1} for various cases of u and v.

Case-3: $\rho = 1$; $l = i + 1$ and $k = 1$.
Since the cycles within an S_{i+1}, in this case, are peers, the corresponding adjacency graph is complete. Therefore, the lemma holds.

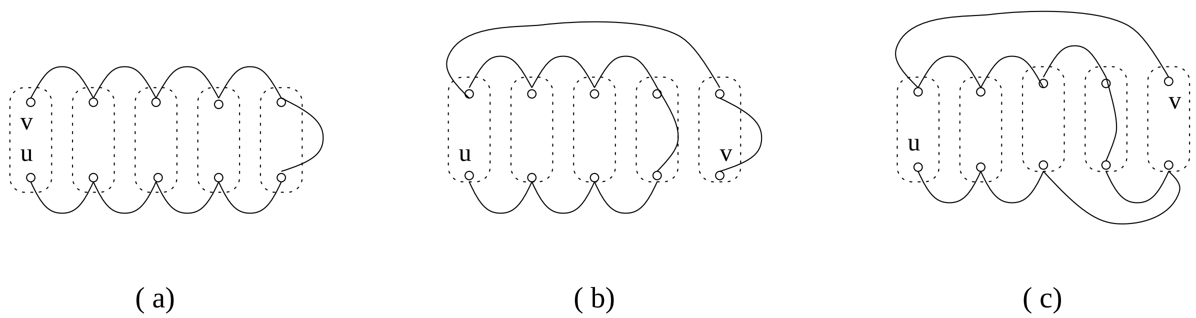


Figure 6.6: Construction of Hamiltonian path in the adjacency graph of cycles: the basis cases for $\rho = 2$. (a) $v = sibling(u)$, (b) $v = peer(u)$, (c) $v \neq sibling(u) \neq peer(u)$.

It may be noted that in each of the above basis cases there is at-least one non-critical vertex in the Hamiltonian path.

Let us now assume that the lemma holds for the adjacency graph corresponding to an S_m, $i \leq m < l$. We need to prove that it holds for $SC_{m,k}$ which will also imply that it holds for S_{m+1}. There are two different constructions depending upon the relative locations of u and v.

Case-(a): u and v belong to two different S_ms.

1. Arrange the S_ms such that the one containing u is in the first position and the one containing v is in the k-th position in the sequence.

2. Construct the Hamiltonian path in the subgraph (of the adjacency graph) corresponding to the first S_m with u as the first vertex.

3. For $j = 2$ to $k-1$, if $k > 2$

   - Suppose H_j is the Hamiltonian path of the subgraph corresponding to the S_m at the j-th position. Construct a Hamiltonian path H_j such that the peer of the last vertex of H_{j-1} is the first vertex of H_j. Let w be the peer of the last vertex of H_{k-1} in the k-th S_m. The case of $w = v$ can be avoided by interchanging the positions of the last two vertices in the path H_{k-1}.

4. Construct a Hamiltonian path in the subgraph corresponding to the k-th S_m with w and v as the end vertices.

5. Connect the last vertex and the first vertex respectively of two consecutive paths (in subgraphs) to get the required Hamiltonian path.

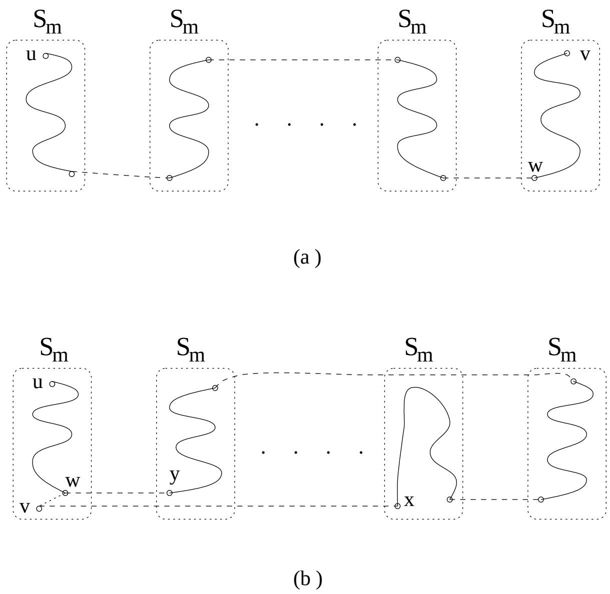


Figure 6.7: Construction of Hamiltonian path in the adjacency graph of cycles in an $SC_{m,k}, i \leq m < l$: (a) u and v belongs to two different S_ms, (b) u and v belong to the same S_m.

Case-(b): u and v belong to the same S_m.

1. Construct a Hamiltonian path H' in the S_m containing u and v with u as the first and v as the last vertex. Let w be the vertex adjacent to v in the path.

2. Let x be the peer of v and y be the peer of w in two different sibling S_ms if $k > 2$. If $k = 2$ x and y are taken in the same S_m.

3. Construct a Hamiltonian path, using the method of Case-(a), in the $k-1$ S_ms not containing u and v, with x and y as the end vertices.

4. Drop v from H' and then connect v to x and w and y to get the desired Hamiltonian path.

□

We have considered the case of obtaining a Hamiltonian path in the adjacency graph corresponding to the column cycles in an $SC_{l,k}$, $l > i$, $1 \leq k$. We shall now consider the construction of a Hamiltonian cycle in the adjacency graph of n_c column cycles obtained in the smallest number of S_{n-1}s that can accommodate the column cycles obtained according to the methods discussed above.

LEMMA 6.3 *It is possible to obtain n_c column cycles in an $SC_{n-1,k}$, where $(k-1)(n-1)! < \lceil \frac{n_c}{\rho} \rceil i! \leq k(n-1)!$, such that the adjacency graph corresponding to these n_c cycles is Hamiltonian, except for the specific case of $n_c = 3$ when $\rho = 2$.*

PROOF. Let us consider that a maximum possible $(k-1)\rho\frac{(n-1)!}{i!}$ column cycles are accommodated in the first $k-1$ S_{n-1}s so that the remaining $\Delta n_c = n_c - (k-1)\rho\frac{(n-1)!}{i!}$ are accommodated in the k-th S_{n-1}.

Case: $\Delta n_c > 1$

1. Take Δn_c vertices in the adjacency graph corresponding to the k-th S_{n-1} such that they are connected in a path. Let w and x be the end vertices of the path.

2. Let u and v be peer vertices of w and x respectively in the adjacency graph corresponding to $k-1$ S_{n-1}s. Construct a Hamiltonian path in this adjacency graph with u and v as the end vertices.

3. Connect u to w and v to x to get the Hamiltonian cycle.

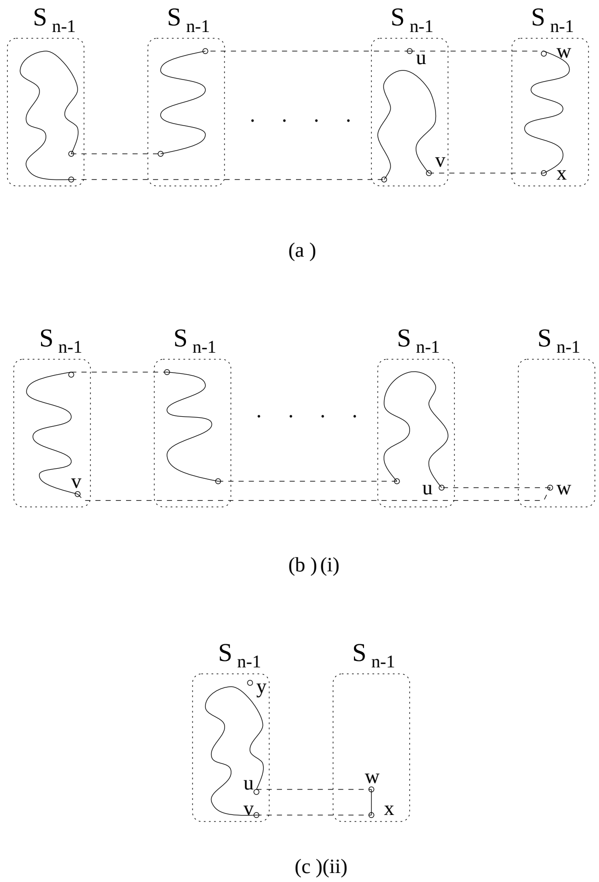


Figure 6.8: Construction of Hamiltonian cycle in the adjacency graph of cycles in an $SC_{n-1,k}$: (a) $\Delta n_c > 1$, (b) $\Delta n_c = 1$: (i) $k > 2$, (ii) $k = 2$.

Case: $\Delta n_c = 1$

If $k > 2$

1. Let w be a vertex in the adjacency graph corresponding to the k-th S_{n-1}.
2. Let u and v be two peer peer vertices of w in the adjacency graph corresponding to the cycles in the first $k-1$ S_{n-1}s. Construct a Hamiltonian path in this adjacency graph with u and v as the end vertices.
3. Connect w to u and v to get the Hamiltonian cycle.

If $k = 2$

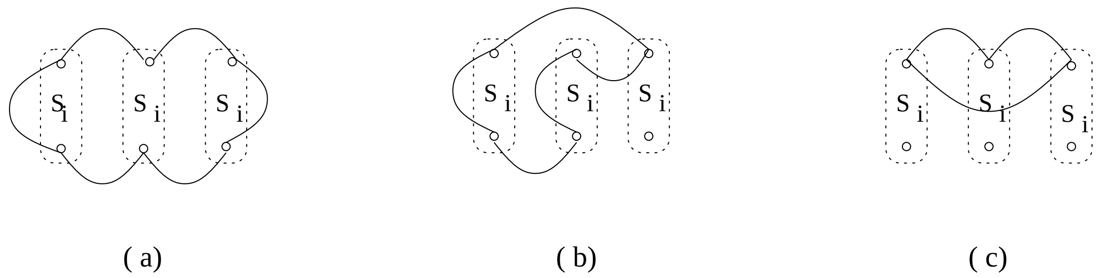


Figure 6.9: Construction of Hamiltonian cycles for $3 \leq n_c < 10$, when $\rho = 2$. (a) n_c even, (b) n_c odd > 3, (c) $n_c = 3$.

---

1. Let w and x be two adjacent vertices in the adjacency graph corresponding to one of the two S_{n-1}s.

2. Let u and v respectively be two peer vertices of w and x in the adjacency graph of the other S_{n-1}. Construct a Hamiltonian path H' in this adjacency graph with u and v as the end vertices.

3. Delete one of the non-critical vertices, say y, in H' to get H''. This is to balance the one excess vertex taken in the second S_{n-1}.

4. Connect u to w and v to x to get the desired Hamiltonian cycle.

Special case of $n_c < 10$ when $\rho = 2$:
Figure 6.9 illustrates the construction of Hamiltonian cycles for $3 \leq n_c < 10$. It can be observed that in the case of $n_c = 3$, three S_{n-1}s (S_is in this case) are required instead of two. The case of $n_c \leq 2$ is trivial. □

THEOREM 6.8 *It is possible to obtain a set of n_c cycles in an $SC_{n-1,k}$, where $(k-1)(n-1)! < \lceil \frac{n_c}{\rho} \rceil i! \leq k(n-1)!$, which can be arranged to form a $n_r \times n_c$-torus with dilation four, except for $n_c = 3$ when $\rho = 2$.*

PROOF. The theorem follows from the Hamiltonicity of the adjacency graph of n_c column cycles ($i.e.$ Lemma 6.2. As mentioned earlier, a Hamiltonian cycle in the adjacency graph of the column cycles corresponds to a cyclic adjacency arrangement of the column cycles which is required to form the torus. □

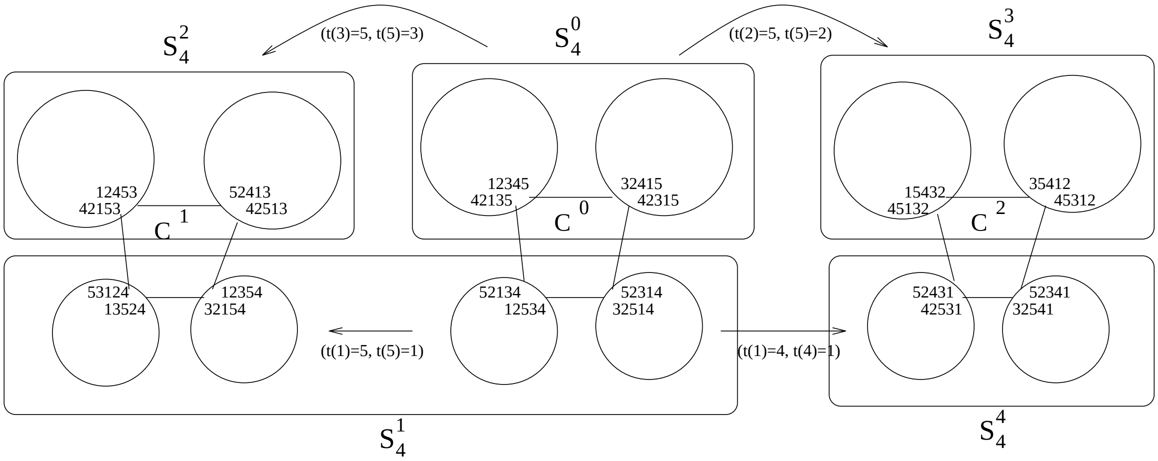


Figure 6.10: Special construction of three cycles in an S_i.

6.2.4 Expansion

Let us assume that ρ is the number of column cycles accommodated in an optimal substar S_i such that we require $\lceil \frac{n_c}{\rho} \rceil$ S_is to embed a torus of size $n_r \times n_c$. If we consider the size of the host graph required for the embedding in units of optimal substars, it is $\lceil \frac{n_c}{\rho} \rceil i!$. The corresponding expansion in this case will be $E_i = \frac{\lceil \frac{n_c}{\rho} \rceil i!}{n_r n_c}$.

Let $(k-1)(n-1)! < \lceil \frac{n_c}{\rho} \rceil i! \leq k(n-1)!$, $1 < k \leq n$, so that k S_{n-1}s are required to contain the $\lceil \frac{n_c}{\rho} \rceil$ S_is. It may be required to consider $k(n-1)!$ as the size of the host graph used in the embedding even though many of the S_is in one of the S_{n-1}s may not be assigned with any guest graph node according to the embedding scheme. In that case the expansion of the embedding scheme is $E_{n-1} = \frac{k(n-1)!}{n_r n_c} = \frac{k(n-1)!}{\lceil \frac{n_c}{\rho} \rceil i!} E_i < \frac{k(n-1)!}{(k-1)(n-1)!} E_i = \frac{k}{k-1} E_i$. In the worst case, for $k = 2$, $E_{n-1} = 2E_i$. On the other hand, if we consider the size of the host graph to be $n!$ the corresponding expansion, $E_n = \frac{n!}{n_r n_c} < \frac{n!}{(k-1)(n-1)!} E_i = \frac{n}{k-1} E_i$. The worst case occurs for $k = 2$ when $E_n < nE_i$.

The expansion measure E_i can be used if it is acceptable to consider an S_i to be available for other purposes when its nodes are used only as intermediate nodes in some routes for the embedded torus edges. Otherwise, the measure E_{n-1} should be used for expansion if confined routing is used. However, if third substar routing is used some of the embedded torus edges will be routed through the S_{n-1}s, in the S_n, that are not assigned with any guest node. In that case E_n should be used as the expansion measure.

For determining the expansion measure E_i we need only to consider the utilization of the optimal substar S_i in constructing the first cycle (set). In case-1, more than half of each S_i is used in the embedding. Therefore in this case the expansion $E_i < 2$. In case-2, there are $\lfloor \frac{i}{\lambda} \rfloor$ cycles in an S_i each having at-least $(\lambda - 1)(i-1)! + 2$ nodes. Therefore, in the worst case, expansion $E_i = \frac{i!}{\lfloor \frac{i}{\lambda} \rfloor ((\lambda-1)(i-1)!+2)}$. For $\lambda = 2$, $E_i = \frac{i!}{\frac{i}{2}((i-1)!+2)} < 2$, if i even, and $E_i = \frac{i!}{\frac{i-1}{2}((i-1)!+2)} \approx \frac{2i}{i-1}$, if i odd. That is, for i odd, E_i may exceed 2. This occurs due to nonutilization of one S_{i-1} in the S_i. The expansion can be brought down to within $\frac{2i}{i+1}$, *i.e.* < 2, by constructing one more cycle within S_i utilizing the unused S_{i-1} as shown in Figure 6.10. In case-3, in each S_i at-least $2\lfloor \frac{i}{2} \rfloor (i-1)! + 2$ nodes are used in the torus. Therefore, $E_i < \frac{i!}{2\lfloor \frac{i}{2} \rfloor (i-1)!}$. The worst case of $E_i \approx 1.25$ occurs for i=5. In case-4, all the S_is, S_4s to be specific, except may be one, will be completely used. Therefore, here too, in the worst case $E_i < 2$.

Thus, for the embedding scheme the bound for E_i is 2. The corresponding bounds for the other two expansion measures E_{n-1} and E_n are, consequently, 4 and $2n$ respectively.

6.2.5 Embedding of multidimensional torus

Consider the problem of embedding a D-dimensional ($D > 2$) torus in a star graph. In the embedding of 2-dimensional torus we consider the adjacency graph of the column cycles (a cycle forms an 1-dimensional torus) and find a Hamiltonian cycle in it to obtain a torus formation. For the embedding of a 3-dimensional torus we can similarly have the concepts of sibling and peer tori and define adjacency (dilation four) of 2-dimensional tori. Then consider the adjacency graph of the 2-dimensional tori in which we can find a Hamiltonian cycle to obtain a dilation four embedding of a 3-dimensional torus. In general, for the embedding of a D-dimensional torus, we can define the concepts of sibling and peer $(D-1)$-dimensional tori which also have dilation four adjacency. Then consider the adjacency graph of the $(D-1)$-dimensional tori in which we can find a Hamiltonian cycle leading to the dilation four embedding of D-dimensional torus.

6.2.6 Example constructions of 2-dimensional tori

In the following we consider a few example constructions of tori in star clusters of appropriate sizes.

i) A 20×20 torus:

The optimal substar for a cycle of size 20 is S_4. An S_4 can accommodate only one 20-cycle.

Therefore, the first cycle of length 20 should be obtained in an S_4 using Jwo's method. Five such 20-cycles can be obtained in an S_5 through peer node imaging in sibling S_4s. Then we can use four sibling S_4s to obtain 20 such 20-cycles through peer node imaging in sibling S_5s.

In this construction none of the column cycles is approximate. Therefore the columns will be of unit dilation. The rows are of dilation four for obvious reason. The expansion measures are- $E_i = 1.2$, $E_{n-1} = 1.2$ and $E_n = 1.8$.

ii) A 360×360 torus:

The optimal substar is S_6. An S_6 can accommodate exactly two 360-cycles each requiring three S_5s. Therefore, the first set of two 360-cycles should be obtained by first constructing the original cycle in three S_5s and then generating the approximate cycle in the remaining three S_5s through i-th symbol swap transformation. By taking substar peer node images of these two cycles in sibling S_6s we can then obtain fourteen 360-cycles in an S_7. Then, we can obtain 112 such cycles in an S_8 through peer node imaging in sibling S_7s. Three sibling S_8s can then be used completely to obtain 336 such cycles through peer node imaging. The rest 24 cycles can be obtained in two S_7s of a fourth S_8 through peer node imaging in S_8.

In this construction half of the column cycles are approximate and therefore of dilation three. Here, $E_i = 1$, $E_{n-1} \approx 1.25$ and $E_n = 2.8$.

iii) A 30×30 torus:

The optimal substar for a 30-cycle is S_5. Three 30-cycles can be constructed in an S_5 as shown in Figure 6.10 using the special construction method. Then the required thirty 30-cycles can be obtained in two S_6 through appropriate peer node imaging. Ten of the column cycles are of unit dilation. The remaining twenty are approximate and therefore of dilation three. Here, $E_i \approx 1.34$, $E_{n-1} = 1.6$ and $E_n = 5.6$.

iv) A 50×50 torus:

The optimal substar for a 50-cycle is S_5. An S_5 can accommodate two 50-cycles with the sharing of an S_4. Therefore, the first set of two 50-cycles should be obtained in an S_5 as discussed in section-3.4. The required fifty 50-cycles can then be obtained in five S_6s through appropriate peer node imaging.

Half the column cycles in this construction are of unit dilation. The rest half are approximate and therefore of dilation three. Here, $E_i = 1.2$, $E_{n-1} \approx 1.44$ and $E_n \approx 2$.

6.2.7 Some communication primitives and associated costs

In this section we discuss the four important communication primitives required in many of the torus algorithms [KGGK93].

Cyclic shift in a row

An edge in a row consists of a peer (a cycle peer) communication path. Therefore, a single step cyclic shift operation in a row consists of a HOPC or an ADC operation requiring at most four communication steps. There is a trade-off in the choice of paths for the row edges. We have seen in Chapter 4 that we can choose either confined routing or third substar routing for HOPC and ADC. The use of confined routing will have an edge congestion $O(n)$. Use of third substar routing will have a small edge congestion of 2. However, this routing will increase the expansion of the embedding as the row edge paths will use all the substars of the top level even if they are otherwise not used in the embedding. Increase in expansion does not occur when an entire substar is used in the embedding. For example, a 10×10 torus can be embedded in an S_5 with expansion 1.2. Third substar routing does not increase expansion here.

Cyclic shift in a column

Most of the column edges are of unit dilation. Therefore the cyclic shift in such cases is straightforward. In case of rows that consists of approximate cycles, there are a few dilation three edges. For these column edges shortest path routing can be used. These dilated edges cause a small congestion. Experimentally this congestion has been found to be limited to 2 which occurs for approximately 1% of the edges.

Broadcast in a row

The broadcast in a row will consist of one or more cycle peer communication steps. Let us assume that an $SC_{n-1,k}$s is used for construction of the torus and S_i be the optimal substar which accommodates one or more cycles. The row broadcast will need the following steps.

1. Transmit the message to the peer nodes in the $k-1$ sibling S_{n-1}s.
2. For j = n-2 to i, each node in the row having the message, transmit it to all the peer nodes in sibling S_js.
3. Each node in the row having the message transmits it to all the cycle peer nodes within

the S_i.

The total number of peer communication steps will be $n - i + 1$ if the optimal substar accommodates more than one cycle, otherwise, it will be $n - i$. The edge congestion in case of row broadcasting, considering the entire set of nodes in a column performs a broadcast using confined routing, is $O(n)$. The associated edge congestion is 2 when third substar routing is used.

Broadcast in a column

The columns of the torus are embedded in distinct set of sibling substars. Therefore, any of the available broadcasting techniques (refer to Section 7.1) can be used, with possible minor modifications, in $O(i)$ time, where S_i is the optimal substar.

6.3 Embedding of FBTs

A full binary tree (FBT), T_h is defined to be a binary tree of height h in which at any level i, $0 \leq i \leq h$, there are 2^i nodes. Sometimes such a tree is also referred to as a complete binary tree. In this section we present two recursive schemes for embedding FBTs on star graphs. In these schemes a decomposition of an FBT into smaller level FBTs is used. An FBT of height h, T_h, can be decomposed into $\rho + 1$ FBTs, $\rho = 2^j$ for some small integer j, as shown in Figure 6.11. One of these FBTs is of height $log_2\rho - 1$. The remaining ρ FBTs are of height $h' = h - log_2\rho$. The FBT of height $log_2\rho - 1$ contains the root of the parent tree which is also its root. This FBT we shall refer to as the *root FBT*. The leaf nodes of this root FBT are the parent nodes for the roots of rest of the ρ FBTs which we shall refer to as the *child FBTs*. An edge between a leaf node of the root FBT and the root of a child FBT shall be referred to as a *link edge*.

In the schemes, an FBT embedding in S_{n-1} is used for obtaining the embeddings in S_n. Assume that the embedding of a $T_{h_{n-1}}$ in $SC_{n-2,k_{n-1}}$ is known. The embedding of T_{h_n} in S_n is then obtained as follows. The FBT of height h_n is decomposed into a root FBT of height $h_n - h_{n-1} - 1$ and $\rho = 2^{h_n - h_{n-1}}$ child FBTs of height h_{n-1}. An S_{n-2} is taken as reference and a distance-2 set of ρ $SC_{n-2,k_{n-1}}$s is obtained in the S_n. The root FBT is embedded in the reference S_{n-2} such that there is a distinct pair of $SC_{n-2,k_{n-1}}$s in the set for each leaf of the root FBT with at least one node in each SC_{n-1,k_n}, of the pair, within two edges from the leaf. Each of the ρ child FBTs is embedded in one of the $SC_{n-2,k_{n-1}}$s in the set with such a node as the root.

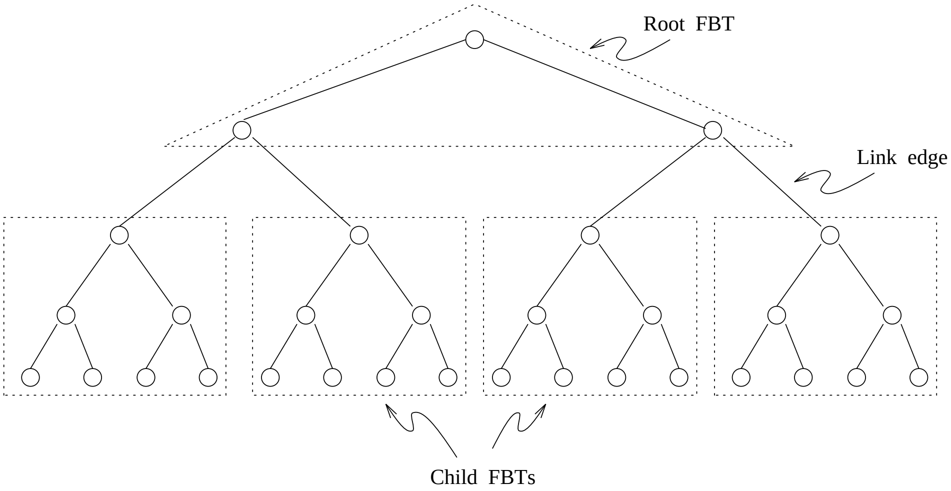


Figure 6.11: Decomposition of an FBT into smaller FBTs.

The two embedding schemes differ depending upon how the star clusters $SC_{n-2,k_{n-1}}$s are obtained in an S_n. In the first scheme, the $SC_{n-2,k_{n-1}}$s are obtained only from the rows of the two dimensional arrangement of S_{n-2}s in S_n (refer Section 5.4.1). In other words, only the PCDS is used for obtaining the star clusters. Therefore, it is referred to as the *single-CDS embedding scheme.* In the second scheme, on the other hand, the $SC_{n-2,k_{n-1}}$s are obtained from both rows and columns of the two dimensional arrangement. That is, two CDSs, the PCDS and an SCDS, are used. Hence, it is referred to as the *double-CDS embedding scheme.* In the single-CDS scheme exactly one $SC_{n-2,k_{n-1}}$ is obtained in an S_{n-1} and therefore the ρ value is limited to $2^{\lfloor log_2 n \rfloor}$. Since the $SC_{n-2,k_{n-1}}$s are obtained form both rows and columns in the double-CDS scheme, the ρ value can be $2^{\lfloor log_2 \frac{n(n-1)-1}{k_{n-1}} \rfloor}$, after keeping one S_{n-2} reserved as the reference substar. As a result more compact embeddings can be achieved through the double-CDS embedding scheme.

As the embedding scheme is recursive, given an FBT, T_h, its embedding needs to be performed in one or more steps in a bottom-up manner starting from the basis embedding of FBT of height 3 in $SC_{3,3}$. The process of constructing the embedding of the T_{h_n} from the embedding of $T_{h_{n-1}}$ is only one such in a sequence of similar steps. Each such step in the sequence we shall refer to as an *embedding step.*

An embedding of T_3 in an $SC_{3,3}$ is used as the basis in both the schemes.

LEMMA 6.4 *A T_3 can be embedded in an $SC_{3,3}$ with dilation two and without congestion.*

PROOF. A T_3 can be decomposed into five T_1s. One of these is the root FBT while the other

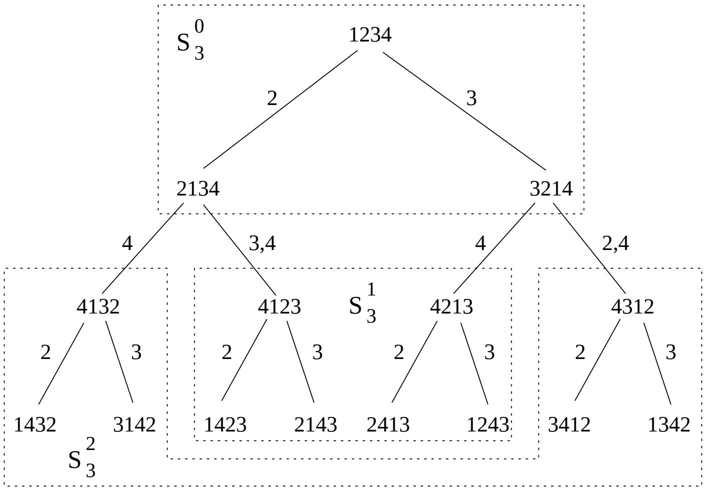


Figure 6.12: Embedding of T_3 in $SC_{3,3}$.

four are the child FBTs. The embedding is shown in Figure 6.12. The root FBT can be placed in one of the S_3s of the $SC_{3,3}$. The four 3-node child FBTs are embedded in the remaining two S_3s. It is easy to see that the embedding is free of both edge and node congestion. $\square$

The $SC_{3,3}$ used in the embedding can consist of any three of the four S_3s of an S_4. If S_3^k is not included in the $SC_{3,3}$, the root of the T_3 is to be assigned to one of the nodes in the chosen $SC_{3,3}$ that has a dimension 4 edge to S_3^k.

6.3.1 Construction of the root FBT

At first we shall present a scheme in which the construction of an FBT of height $log_2\rho - 1$ ($i.e.$ with $\frac{\rho}{2}$ leaves) that requires edges of $\frac{\rho}{2}$ dimensions. This is then modified to construct FBTs of the same height with edges of $\frac{\rho}{2} - 1$ dimensions. In the construction, the FBT has leaf nodes of distinct first symbols. In the following we shall assume that the nodes of an FBT are numbered in the usual manner. That is, the root node is numbered #1. The left child of a node $\#x$ is numbered $\#2x$ and its right child is numbered $\#(2x+1)$. The construction is as follows. Let the dimensions to be used in the first construction be $\{d_1, d_2, ..., d_{\frac{\rho}{2}}\}$.

- Use a distinct dimension from the set $\{d_1, d_2, ..., d_{\frac{\rho}{2}}\}$ for an edge of the $\frac{\rho}{2}$ edges to the leaf nodes.

- Choose the dimensions to the remaining edges as follows:

- Choose the dimension of the edge $(\#x, \#2x)$ to be same as that of the edge $(\#(2x+1), \#2(2x+1))$.
- Choose the dimension of the edge $(\#x, \#(2x+1))$ to be same as that of the edge $(\#2x, \#4x)$.

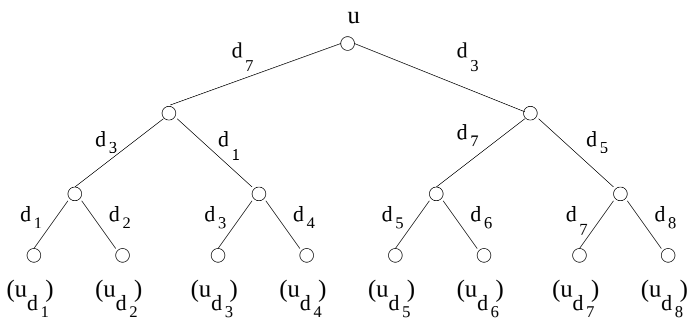


Figure 6.13: Construction of FBT of height $log_2\rho - 1,\ \rho = 16$, with dilation one using $\frac{\rho}{2}$ dimensions. First symbols of the leaf nodes are shown within parentheses.

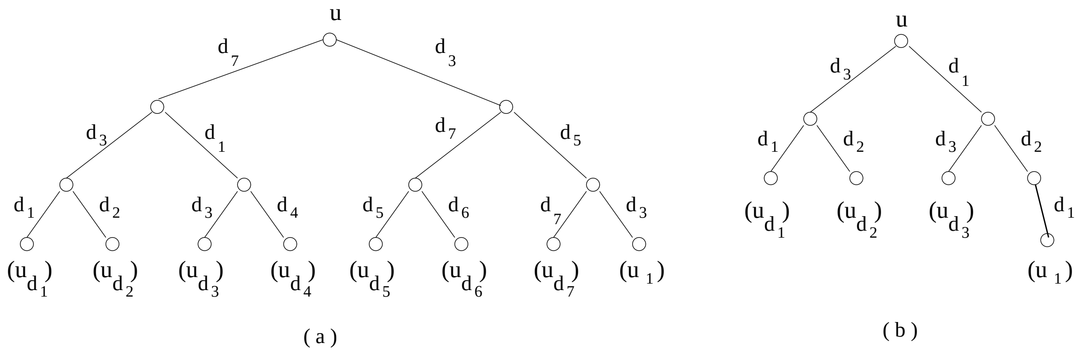


Figure 6.14: Modified construction of FBT of height $log_2\rho - 1,\ \rho = 16$, with dilation one using $\frac{\rho}{2} - 1$ dimensions. (a) for height ≥ 3, (b) for height $=2$.

Figure 6.13 shows such an FBT. The FBTs constructed with this method has the following properties.

1. No two edges in the path from the root to a leaf node are of the same dimension.
2. The first symbols of the $\frac{\rho}{2}$ leaf nodes are distinct. If the root node is u and the dimension of the edge to a leaf from its parent is d, then its first symbol is u_d.
3. First symbol of none of the leaf nodes is u_1.

4. The FBT is congestion-free.

5. The FBT requires an $S_{\frac{\rho}{2}+1}$.

The construction for the FBT with edges of $\frac{\rho}{2} - 1$ dimensions, $\rho > 4$, depends upon the height of the tree. For an FBT of height ≥ 3, replace the dimension of the edge to the leaf node $\#(\rho - 1)$, in the above construction, with the dimension of the edge from the root towards the leaf node as shown in Figure 6.14(a). In case, height = 2, construct the FBT as shown in Figure 6.14(b). The substar required in this case is an $S_{\frac{\rho}{2}}$. Moreover, the first symbol of the node $\#(\rho - 1)$ in this case is u_1. Figure 6.14 shows FBTs with the modified construction.

6.3.2 The single-CDS embedding scheme

In this scheme an $SC_{n-1,\rho+1}$ is taken in an S_n for the embedding of the FBT if n is not a power of two. Else, the entire S_n is taken. The ρ child FBTs are embedded in ρ S_{n-1}s one in each. The root FBT is embedded in the remaining S_{n-1} when n is not a power of 2. Other wise, it is embedded in an S_{n-2}, in the S_n, not used in the embedding of the child FBTs. The general scheme for the construction of the embeddings is embodied in the two lemmas below.

LEMMA 6.5 *For $2^l < n < 2^{l+1}$, if an FBT of height h_{n-1} can be embedded in an S_{n-1} with dilation two and without congestion, then an FBT of height $h_{n-1} + log_2\rho$, $\rho = 2^{\lfloor log_2 n \rfloor}$ can be embedded in an $SC_{n-1,\rho+1}$ with the same dilation and without congestion.*

PROOF. The FBT of height $h_{n-1} + log_2\rho$ can be decomposed into ρ child FBTs of height h_{n-1} and a root FBT of height $log_2\rho - 1$. One child FBT can be embedded in each of ρ S_{n-1}s of the $SC_{n-1,\rho+1}$. The root FBT can be embedded in the remaining S_{n-1}.

Let u be the node in S^p_{n-1} chosen for the root node. Apart from S^p_{n-1}, let the other ρ available S_{n-1}s for the $SC_{n-1,\rho+1}$ are such that their n-th symbols are $s_1, s_2, ..., s_\rho$. Further, let the positions of these symbols in the permutation u be $p_1, p_2, ..., p_\rho$ respectively. Choose $\frac{\rho}{2}$ values from the positions p_1 to p_ρ such that none of these is 1. Without loss of generality, assume that these chosen values be $\{p_1, p_2, ..., p_{\frac{\rho}{2}}\}$. Construct a dilation one FBT of height $log_2\rho - 1$ in S^p_{n-1} with u as the root using the $\frac{\rho}{2}$ dimensions $\{p_1, p_2, ..., p_{\frac{\rho}{2}}\}$ with the scheme discussed in the previous section. The first symbols of the leaf nodes will be $s_1, s_2, ...$ and $s_{\frac{\rho}{2}}$. Therefore, these nodes of the root FBT will have adjacent nodes along dimension n in the S_{n-1}s having n-th symbols $s_1, s_2, ...$ and $s_{\frac{\rho}{2}}$ respectively. Construct a child FBT of height h_{n-1} in each of these S_{n-1}s with the respective adjacent node of the leaf as the roots. This leaves $\frac{\rho}{2}$ more child FBT embeddings to be constructed.

There are two cases we have to consider for the embedding of the remaining $\frac{\rho}{2}$ child FBTs.

Case-1: $p_j \neq 1$, for $j = 1, 2.., \rho$
In this case, none of the n-th symbols of the S_{n-1}s occupy the first position in u. This is possible when $n \geq \rho + 2$. The routing in this case can be performed as follows.

To each of the leaf nodes of the root FBT (there are $\frac{\rho}{2}$ leaves) add an edge of a dimension from the set $\{p_{\frac{\rho}{2}+1}, p_{\frac{\rho}{2}+2}, ..., p_\rho\}$. These dimensions do not appear in the root FBT. Therefore, the added edges are congestion-free. The first symbols of the new nodes will be $s_{\frac{\rho}{2}+1}$, $s_{\frac{\rho}{2}+2}$, ..., and s_ρ. The node with the first symbol $s_{\frac{\rho}{2}+j}$ will be adjacent to the S_{n-1} with $s_{\frac{\rho}{2}+j}$ as the n-th symbols via dimension n edge. These adjacent nodes are at distance of two edges from the leaf nodes of the root FBT. Construct child FBTs with these nodes as roots, in the respective S_{n-1}s.

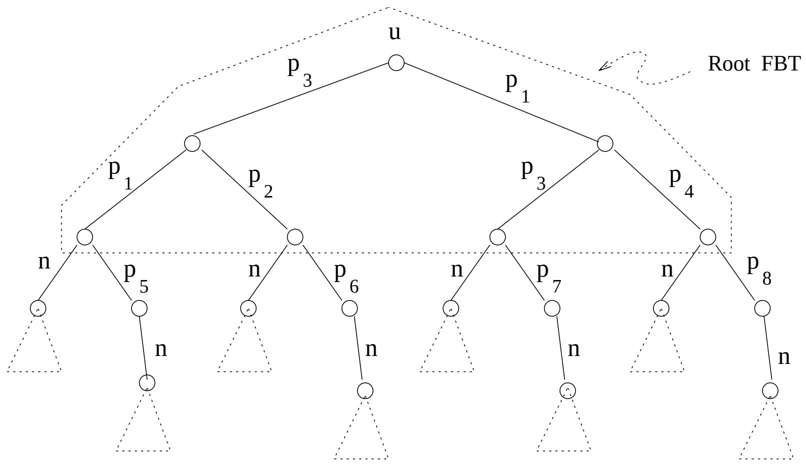
Case-2: one of the positions, say p_ρ, is 1.
In this case, $s_\rho = u_1$ and the position p_ρ can not represent a dimension of S_{n-1}.

To each of the leaf nodes, except one, add an edge of a dimension from the set $\{p_{\frac{\rho}{2}+1}, p_{\frac{\rho}{2}+2}, ..., p_{\rho-1}\}$. These dimensions do not appear in the root FBT. Therefore, the added edges are congestion-free. The first symbols of the new nodes will be $s_{\frac{\rho}{2}+1}$, $s_{\frac{\rho}{2}+2}$, ..., and $s_{\rho-1}$. Therefore, these nodes will have adjacent nodes in the S_{n-1}s along dimension n in the S_{n-1}s having these as n-th symbols. These adjacent nodes are at distance two from the leaf nodes. Construct child FBTs with these nodes as roots, in the respective S_{n-1}s.

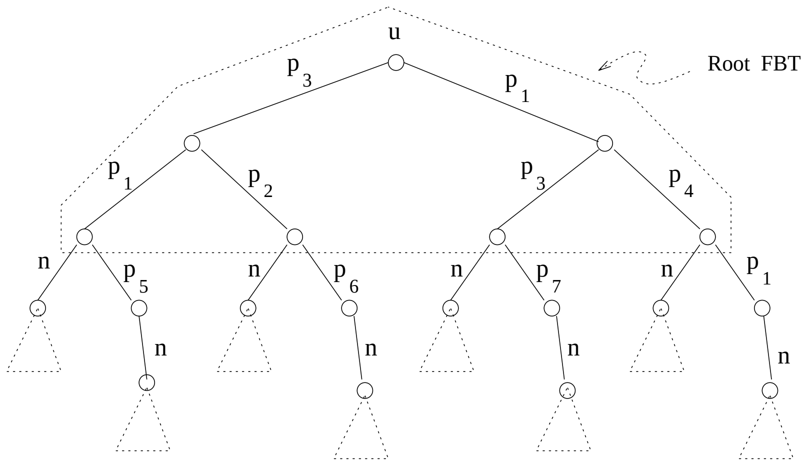
In the remaining leaf node of the root FBT s_ρ will occupy the position d, where d is the dimension of the first edge from the root u to the leaf. Add an edge of dimension d to this leaf node if $\rho > 4$. The new node will have the first symbol s_ρ. The root u is the only other node in the root FBT with first symbol s_ρ. Therefore, it is distinct from all other nodes in the root FBT. Further, the new node will have an adjacent node in the remaining S_{n-1} the n-th symbol of which is s_ρ. Take this adjacent node as the root and construct a child FBT in the S_{n-1}. This will complete the construction of the embedding.

If $\rho = 4$, then construct a dilation two root FBT as shown in the Figure 6.15(b)(ii).

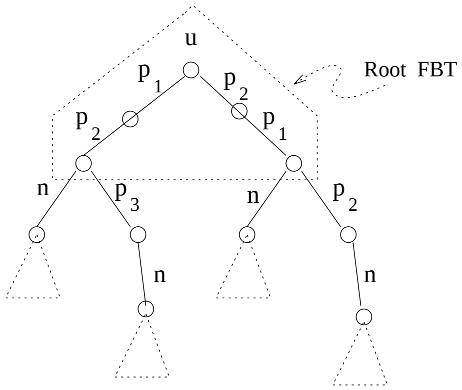
As the child FBTs and the root FBT are embedded in disjoint substars or star clusters, and the link edges do not use any node used by these embeddings, the embedding is congestion-free. Moreover, the root FBT is of dilation two (in the worst case) and the link edges are also of dilation two. Therefore, the dilation of the embedding is 2. □



(a)



(b)(i)



(b)(ii)

Figure 6.15: Single-CDS embedding of FBTs in $SC_{n-1,\rho+1}$, $2^l > n > 2^{l+1}$, $\rho = 2^l$; (a) $p_\rho \neq 1$; (b) $p_\rho = 1$: (i) $\rho > 4$, (ii) $\rho = 4$.

LEMMA 6.6 *For $n = 2^l$, if an FBT of height h_{n-2} can be embedded in an S_{n-2} with dilation two and without congestion, then an FBT of height $h_n = h_{n-2} + log_2 n + \lfloor log_2(n-1) \rfloor$ can be embedded in an S_n with the same dilation and congestion.*

PROOF. The FBT of height h_n can be decomposed into n child FBTs of height $h_{n-1} = h_{n-2} + \lfloor log_2(n-1) \rfloor$ and a root FBT of height $log_2 n - 1$. Since an FBT of height h_{n-2} can be embedded in an S_{n-2}, as per Lemma 6.5, an FBT of height $h_{n-1} = h_{n-2} + \lfloor log_2(n-1) \rfloor$ can be embedded in $SC_{n-2,2^{\lfloor log_2(n-1) \rfloor}+1}$ of an S_{n-1}. There will be $n - 1 - 2^{\lfloor log_2(n-1) \rfloor} - 1 = n - 2 - \frac{n}{2}$ S_{n-2}s free in the S_{n-1} after the embedding. Thus $n-1$ of the child FBTs of height h_{n-1} can be embedded in $n-1$ of the S_{n-1}s and the other child FBT of height h_{n-1} can be embedded in an $SC_{n-2,2^{\lfloor log_2(n-1) \rfloor}+1}$ of the remaining S_{n-1}. The root FBT can be embedded in a free S_{n-2} of this S_{n-1}.

Let $S^{p'}_{n-2}$ in S^p_{n-1} be the substar chosen for hosting the root FBT and u be the node in it chosen for the root of the FBT. Construct a dilation one FBT of height $log_2 n - 1$ in $S^{p'}_{n-2}$ with u as the root and with the dimensions $d_1, d_2, ...d_{\frac{n}{2}}$ with the scheme discussed in Section 6.3.1. The first symbols of the leaf nodes will be $u_{d_1}, u_{d_2}, ...$ and $u_{d_{\frac{n}{2}}}$. Therefore, they will have adjacent nodes along dimension n in the S_{n-1}s having n-th symbols $u_{d_1}, u_{d_2}, ...$ and $u_{d_{\frac{n}{2}}}$ respectively. Construct $T_{h_{n-1}}$ height child FBTs in these S_{n-1}s with the respective adjacent nodes of the leaf nodes as the roots. This leaves $\frac{k}{2}$ more child FBT embeddings to be constructed.

There are $n-3$ distinct dimensions in an S_{n-2}, Therefore, there are $\frac{n}{2} - 3$ dimensions in S_{n-2} that has not been used in the root FBT. Add an edge of one distinct dimension of these unused dimensions to each of $\frac{n}{2} - 3$ leaf nodes. As these dimensions do not appear in the root FBT the added edges are congestion-free. The first symbol of such a new node with dimension d edge from leaf will be u_d. The first symbols of the new nodes will be distinct within themselves and also distinct from those of the leaf nodes. Therefore, the new nodes will have adjacent nodes along dimension n edges in distinct S_{n-1}s. These adjacent nodes are at distance two from the leaf nodes. Construct child FBTs with these nodes as roots, in the respective S_{n-1}s.

There are embeddings for three child FBTs that remain to be constructed. The S_{n-1}s that remain to be used are as follows:

1. In S^p_{n-1} only one of the S_{n-2}s has been used. Of the remaining S_{n-2}s $\frac{n}{2} + 1$ are to be used for embedding one of the child FBTs.

2. The S_{n-1} that has the n-th symbol same as the (n-1)-th symbol (fixed symbol) of $S^{p'}_{n-2}$ has not been used.

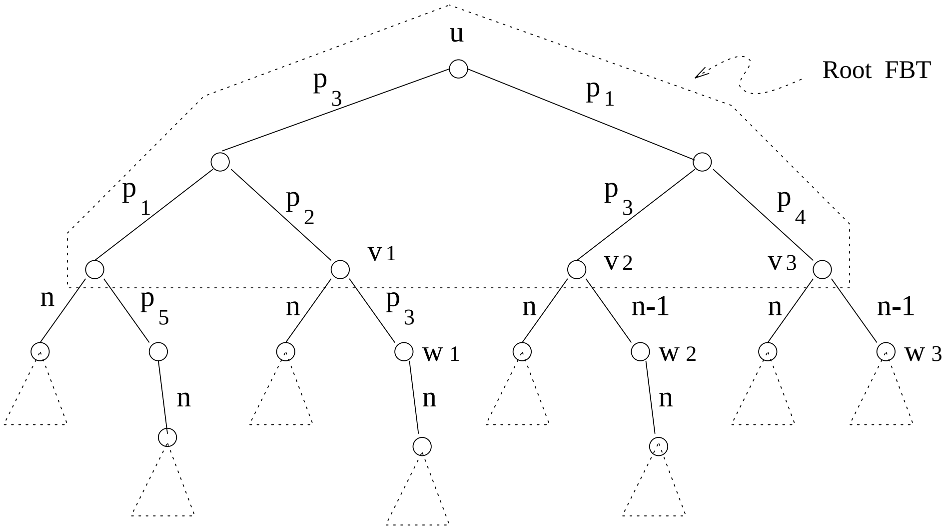


Figure 6.16: Single-CDS embedding of FBT in $SC_{n-1,n}$, $n = k = 2^l$.

3. The first symbol of u, u_1, does neither appear as first symbol in any of the leaf nodes nor as first symbols in the new nodes as dimension 1 does not exist. Therefore, the S_{n-1} with the n-th symbol u_1 remains to be used.

Let $v1$, $v2$ and $v3$ be the leaf nodes for which only one child FBT has been constructed. Let d be the dimension of the edge from the root node u on the path to $v1$. Then u_1 will occupy the position d in $v1$. Hence, a node $w1$ adjacent to $v1$ along dimension d will have u_1 as its first symbol. The root node is the only other node in the root FBT with the first symbol u_1. Therefore $w1$ is distinct from any other node in the root FBT and the edge form $v1$ to $w1$ is congestion free. A node in the S_{n-1}, with u_1 as the n-th symbol, can be reached from $v1$ within two edges via $w1$. With this node as the root embed a child FBT in the S_{n-1}.

The node $v2$ has an adjacent node $w2$ along dimension $n-1$ in a sibling (say S^q_{n-2}) of $S^{p'}_{n-2}$. The first symbol of $w2$ is $s^{p'}_{n-1}$. Therefore, $w2$ has an adjacent node along dimension n in the S_{n-1} with the n-th symbol same as the (n-1)-th symbol of $S^{p'}_{n-2}$. Use this node, which is within two edges from $v2$, as the root for embedding a child FBT in the S_{n-1}. The node $w2$ in S^q_{n-2} is used as an intermediate node in the path. Therefore, to avoid congestion, S^q_{n-2} can not be included in the $SC_{n-2,\frac{n}{2}+1}$ that is used for hosting the child FBT in S^p_{n-1}.

The node $v3$ has an adjacent node $w3$ along dimension $n-1$ in a sibling, say S^r_{n-2}, of $S^{p'}_{n-2}$ which is distinct from S^q_{n-2}. Form an $SC_{n-2,2^{\lfloor log_2(n-1)\rfloor+1}}$, in S^p_{n-1}, that includes S^r_{n-2} but excludes $S^{p'}_{n-2}$ and S^q_{n-2}. As there are sufficient number of S_{n-2}s in S_{n-1} to spare, this can be easily achieved. Embed a child FBT in the star cluster as per Lemma 6.5 with $w3$ as the root. This will complete the construction of the embedding.

As the child FBTs and the root FBT are embedded in disjoint substars or star clusters, and the link edges do not use any node used by these embeddings, the embedding is congestion-free. Moreover, the root FBT is of dilation one and the link edges are of dilation two. Therefore, the dilation of the embedding is 2. □

THEOREM 6.9 *An FBT of height $h_n = (n+1)\lfloor log_2 n\rfloor - 2^{\lfloor log_2 n\rfloor+1} + 1$ can be embedded in an S_n with dilation two and without congestion.*

PROOF. According to Lemma 6.5, for $n \geq 5$, n not a power of 2, the height of the FBT embedable in an S_n is $\lfloor log_2 n\rfloor$ more than the height of the tree embedable in an S_{n-1}. According to Lemma 6.6, for $n \geq 8$ and a power of 2, the height of FBT embedable in an S_n is $\lfloor log_2(n-1)\rfloor + log_2 n$ more than the height of the FBT embedable in S_{n-2}. In effect, the height of FBT embedable in S_n is $\lfloor log_2 n\rfloor$ higher than the height of the FBT embedable in S_{n-1}, for $n \geq 5$. Therefore, $h_n = h_4 + \sum_{j=5}^{n}\lfloor log_2 j\rfloor$ for $n \geq 5$. Since a 15 node FBT can be embedded in an S_4, $h_4 = 3$. Therefore, for $n \geq 4$, $h_n = 3 + \sum_{j=5}^{n}\lfloor log_2 j\rfloor = \sum_{j=0}^{n}\lfloor log_2 j\rfloor - 1 = (n+1)\lfloor log_2 n\rfloor - 2^{\lfloor log_2 n\rfloor+1} + 1$. It is trivial to construct dilation one FBTs of heights 0 and 1 in S_2 and S_3 respectively. Therefore, the height of the FBT embedable in S_n, $n \geq 2$, with dilation two and congestion is $(n+1)\lfloor log_2 n\rfloor - 2^{\lfloor log_2 n\rfloor+1} + 1$. □

6.3.3 Double-CDS embedding scheme

In this embedding scheme, to make the embedding compact, the smallest possible star cluster, SC_{n-1,k_n} in S_n is used for the embedding of T_{h_n}. In an SC_{n-1,k_n} there are $(n-1)k_n$ S_{n-2}s and k_{n-1} S_{n-2}s are required in an $SC_{n-2,k_{n-1}}$ for embedding each child FBT. When $k_n \geq k_{n-1}$, it is possible to obtain a distance-2 set of ρ $SC_{n-2,k_{n-1}}$s and a reference S_{n-2} in an SC_{n-1,k_n}, where $k_n = \lceil \frac{\rho k_{n-1}+1}{n-1} \rceil$ using the algorithm D2_Clusters presented in Section 5.4.3.

In an embedding step, when k_{n-1} is close to $n-1$ it is possible that $k_n \geq \rho$. In such a case, only one $SC_{n-2,k_{n-1}}$ can be obtained from an S_{n-1}, and therefore, only the PCDS needs to be used for obtaining the set of star clusters $SC_{n-2,k_{n-1}}$s. Such an embedding step we shall refer to as a *single-CDS* embedding step. When $k_n < \rho$, the star clusters have to be obtained from both rows and columns of the two dimensional arrangement discussed in Section 5.4.1. That is, both PCDS and the SCDS need to be used for obtaining the set of star clusters. Such an embedding step we shall refer to as a *double-CDS* embedding step.

Single-CDS embedding step

In a single-CDS embedding step, a complete S_{n-1} is available for the embedding of each of the $\rho - 1$ of the child FBTs. The remaining child FBT may share an S_{n-1} with the root FBT if $k_{n-1} < n-1$. Since an S_{n-1} is vertex symmetric, the root node of a child FBT can be placed at any node, in the S_{n-1}, which is within two edges from its associated leaf node. As a result, the root of the $T_{h_{n-1}}$ also can be placed at any node in the S_{n-1}.

In a single-CDS step, always the largest FBT embedding in S_{n-1} is used for obtaining an embedding in S_n. This is for the obvious reason of keeping the unused portion of the S_{n-1}s minimum.

In an S_n there are n S_{n-1}s. Therefore, there can be at most $2^{\lfloor log_2 n \rfloor}$ child FBTs in a single-CDS step of embedding in an S_n. As a result, the upper bound on the height of the root FBT in a single-CDS embedding step of embedding in an S_n is $\lfloor log_2 n \rfloor - 1$. Hence, an $S_{2^{\lfloor log_2 n \rfloor - 1}}$ is sufficient to construct the root FBT required for the embedding.

The construction of the embedding in a single-CDS step depends upon the value of k_{n-1}. When $k_{n-1} < n-1$, $k_n = \rho$ and therefore the root FBT and a child FBT has to share one of the S_{n-1}s of the SC_{n-1,k_n}. On the other hand, when $k_{n-1} = n-1$, $k_n = \rho + 1$ and an additional S_{n-1} is available for the embedding of the root FBT.

Case-1: $k_{n-1} < n-1$ ($i.e.$, $k_n = \rho$)
Let u be the node chosen for the root and assume that it belongs to $S^{p'}_{n-2}$ of S^{p}_{n-1} in the SC_{n-1,k_n}. Construct a root FBT of height $log_2 k_n - 1$ in $S^{p'}_{n-2}$ and u as the root using the dimensions 2,3,...,$\frac{k_n}{2} + 1$ using the method discussed in Section 6.3.1. Associate a pair of the S_{n-1}s of the SC_{n-1,k_n} to a leaf of the root FBT such that the n-th symbol of neither of the S_{n-1}s is same as the first symbol of the leaf or its parent. For each S_{n-1}, other than S^{p}_{n-1}, find the node which is two edges from the associated leaf node and construct a h_{n-1} height child FBT embedding with this node as the root. Let l be the leaf node of the root FBT which is to be the parent of the child FBT in S^{p}_{n-1}. Find the node w adjacent to l along dimension $n-1$ which belongs to, say S^{q}_{n-2}, a sibling of $S^{p'}_{n-2}$. Take an $SC_{n-2,k_{n-1}}$ that includes S^{q}_{n-2} but excludes $S^{p'}_{n-2}$. Construct the child FBT in the $SC_{n-2,k_{n-1}}$ with w as the root.

Case-2: $k_{n-1} = n-1$ ($i.e.$, $k_n = \rho + 1$)
Let u be the node chosen for the root and assume that it belongs to S^{p}_{n-1} in the SC_{n-1,k_n}. Construct a root FBT of height $log_2(k_n - 1) - 1$ in S^{p}_{n-1} and u as the root using the dimensions 2,3,...,$\frac{k_n - 1}{2} + 1$. Associate a pair of the S_{n-1}s, not including S^{p}_{n-1}, of the SC_{n-1,k_n} to a leaf of the root FBT such that the n-th symbol of neither of the S_{n-1}s is same as the first symbol of the leaf or its parent. Find the nearest node (two edges away) in each S_n from the associated

leaf node and construct a h_{n-1} height child FBT embedding with this node as the root.

It may be worth noting that the embeddings in the above cases differ from those in an embedding step of the single-CDS scheme presented in the previous section. In the above, the embedding of $T_{h_{n-1}}$ in $SC_{n-2,k_{n-1}}$ may be obtained through a double-CDS embedding step. In an embedding step in the single-CDS scheme when $k_n = \rho$ *i.e.* $n = 2^l$, there are sufficient number of free S_{n-2}s in the S_{n-1} embedding the root FBT and a child FBT to allow the embedding to be congestion-free. However, the same cannot be ensured in a single-CDS step of the double-CDS embedding scheme. This fact leads to the possibility of node congestion in this scheme (In the Case-1 above). As the requirement of avoiding node congestion is not imposed in this scheme, the embedding for the Case-2 above is simpler compared to the single-CDS scheme in a similar situation.

It is clear that if the height of the FBT embedable in S_{n-1} is h_{n-1} than the height of the FBT embedable in an S_n through a single-CDS step is $h_{n-1} + \lfloor log_2 n \rfloor$.

Lemma 6.7 *In a single-CDS embedding step the root FBT and the link edges are free of edge-congestion.*

Proof. The path used by a link edge can share an edge with (a) the path used by the second link edge from the same leaf node, or (b) the path used by the leaf node and its parent, or (c) the path used by an edge between the child root and one of its child nodes.

Let the w be a root FBT leaf node, x be its parent node and S^j_{n-1} and S^k_{n-1} be the substar pair associated to w. The paths to be used by the link edges from w to the child FBT root nodes will be $(pos(s^j_{n-1}, w), n)$ and $(pos(s^k_{n-1}, w), n)$. Since S^j_{n-1} and S^k_{n-1} are distinct, $s^j_{n-1} \neq s^k_{n-1}$. Therefore, the case-(a) does not occur. The path between w and its parent node x consists of the single edge $(pos(x_1, w))$. According to the embedding scheme $x_1 \neq s^j_{n-1} \neq s^k_{n-1}$. Therefore, the case-(b) does not occur. The case-(c) will not occur because the root nodes of the child FBTs are the only nodes in the paths used by the link edge that belong to the S_{n-1}s hosting the child FBTs.

In the case where the child FBT shares the S_{n-1} with the root FBT, the link edge uses a single edge path $(n-1)$, which is an edge linking the S_{n-2} hosting the root FBT to the $SC_{n-2,k_{n-1}}$ hosting the child FBT. Therefore, this link edge is also congestion-free. □

Double-CDS embedding step

In a double-CDS embedding step, an $SC_{n-2,k_{n-1}}$ is available for the embedding of a child FBT, $T_{h_{n-1}}$ but an $SC_{n-2,k_{n-1}}$ is not vertex symmetric. Therefore, while embedding a $T_{h_{n-1}}$ in an $SC_{n-2,k_{n-1}}$, it may not be possible to place the root at an arbitrary host node. There may be some specific characteristics that the host of the root node must possess, in the embedding of $T_{h_{n-1}}$ in $SC_{n-2,k_{n-1}}$, with reference to the S_{n-2}s included in the star cluster. For example, in the embedding of T_3 in $SC_{3,3}$ the root node must be placed at a node which has a dimension 4 edge to the sibling S_3 excluded from the $SC_{3,3}$. However, if there is no such requirement, the roots of the child FBTs can also be positioned at will. In that case the root of T_{h_n} can also be placed at any node. It may be noted that there is no specific requirement on the placement of the root node in an FBT embedding obtained through a single-CDS step. Therefore, if the child FBT embeddings are obtained through single-CDS step, the parent FBT embedding, even if obtained through a double-CDS step, will not have restrictions on the placement of the root node. We know that the embedding of T_3 in $SC_{3,3}$ has restriction on the choice of the root node. However, an FBT embedding in S_5 through the double-CDS scheme, which has to use the embedding of T_3 in $SC_{3,3}$, can be performed only with single CDS steps. (See Table 6.2.) As a result an FBT embedding in S_5 does not have restriction on the placement of the root node. Due to this the restriction on the placement of the root node of T_3 in $SC_{3,3}$ does not propagate to the embedding of larger FBTs.

It should be noted that a column can be used for obtaining an $SC_{n-2,k_{n-1}}$ in an SC_{n-1,k_n} only if $k_n \geq k_{n-1}$. If $k_n < k_{n-1}$ and also $k_n < \rho$, it is not possible to obtain ρ $SC_{n-2,k_{n-1}}$s in an SC_{n-1,k_n}. In that case, it is not be possible to embed T_{h_n} in an SC_{n-1,k_n} using the embedding of $T_{h_{n-1}}$ in $SC_{n-2,k_{n-1}}$. In this situation it is necessary to embed T_{h_n} in an $SC_{n-1,k_{n-1}}$ or attempt to get a more compact embedding using the embedding of a $T_{h_{n-1}-1}$. If $T_{h_{n-1}-1}$ is to be used, the number of child FBTs will be 2ρ. The embedding of $T_{h_{n-1}-1}$ can be obtained, say in an $SC_{n-2,k'_{n-1}}$, using the embedding of $T_{h_{n-2}}$, through a single-CDS step [2]. The star cluster required for embedding T_{h_n} will then be an SC_{n-1,k'_n}, where $k'_n = \lceil \frac{2\rho k'_{n-1}+1}{n-1} \rceil$. We can go for obtaining the embedding of T_{h_n} in SC_{n-1,k'_n} using the embedding of $T_{h_{n-1}-1}$ only if $k'_n < k_{n-1}$. If $k'_n \geq k_{n-1}$, then it will be simpler and possibly less expensive to obtain the embedding of T_{h_n} in an $SC_{n-1,k_{n-1}}$ using the embedding of $T_{h_{n-1}}$.

THEOREM 6.10 *In a double-CDS step, if the largest possible embedding in S_{n-1} is used for obtaining an embedding in S_n, the number of child FBTs required will be less then $2(n-1)$.*

[2]Restriction to single-CDS step, in this case, is to avoid recurrence of similar situation in the embedding of $T_{h_{n-1}-1}$ too.

PROOF. We wish to obtain the embedding of T_{h_n} in SC_{n-1,k_n} using the embedding of $T_{h_{n-1}}$ in $SC_{n-2,k_{n-1}}$ using a double-CDS step. The number of child FBTs required is $\rho \leq \frac{(n-1)k_n}{k_{n-1}}$. The value of k_{n-1} is such that $\frac{k_n}{2} < k_{n-1} \leq k_n$. Here $k_{n-1} > \frac{k_n}{2}$ because we are considering the largest possible embedding in S_{n-1}. Substituting $k_{n-1} = \frac{k_n}{2}$, therefore, we have the number of child FBTs $\rho < 2(n-1)$. □

The above theorem is important because it tell us that the number of star clusters required in a distance-2 set for an embedding is less than $2(n-1)$ which is within the upper limit of $2(n+k_n-4)$, for $k_n \geq 3$ (Theorem 5.8). Further, it also gives us the size of the largest substar required for embedding the root FBT. We have seen in Section 6.3.1 that the root FBT required for an embedding with ρ child FBTs can be constructed in an $S_{\frac{\rho}{2}}$. Therefore, the root FBT required for embedding T_{h_n} can be constructed within an S_m, where $m \leq n-2$.

A double-CDS embedding step consists of the following substeps.

1. Obtain a distance-2 set of ρ $SC_{n-2,k_{n-1}}$s in an SC_{n-1,k_n} with respect to a given reference S_{n-2} as discussed in Section 5.4.3.

2. Form pairs of the star clusters in the distance-2 set as discussed below.

3. Construct an embedding of a root FBT $T_{log_2\rho-1}$ in the reference S_{n-2}, with the required set of first symbols in the leaf nodes, as per the star cluster pairing, and taking a given node as its root. Using the method discussed in Section 6.3.1.

4. Associate the star cluster pairs to leaf nodes of the tree appropriately to ensure that the link edges are edge-congestion-free and their dilation do not exceed 2.

5. Identify the closest node in each $SC_{n-2,k_{n-1}}$ from the associated leaf node and construct an embedding of a $T_{h_{n-1}}$ in the $SC_{n-2,k_{n-1}}$ with the identified node as the root.

Two star clusters $SC_{n-2,k_{n-1}}$s hosting a pair of sibling child FBTs are said to form a pair. The star cluster in such a pair must have a node within two edges from the same leaf node of the root FBT. In addition to the dilation constraint, the pairing also affects the edge congestion of the embeddings. To avoid edge-congestion on the paths used by the link edges it is necessary to perform pairing of the star clusters and their association with leaf nodes carefully. Table 6.1 gives a certain set of allowed pairings of star clusters. Each is subject to a given condition on the associated leaf node of the root FBT. The pairings satisfy our congestion and dilation requirements. Priority levels are attached to the allowed pairings. At any stage the feasible pairing with highest priority is preferred. Priority of level 1 is considered to be the highest.

The pairings are specified in terms of the key elements in the $SC_{n-2,k_{n-1}}$s and their deciding symbols. If a star cluster includes one or more key elements from first row or first column any one of these can be considered to be the key element of the star cluster. If there is no first row or first column key element in the star cluster and there are one or more key elements from the second row or the diagonal, then any one of these can be considered to be the key element of the star cluster. One additional condition that should be followed in the association of the leaf node with a star cluster pair is that the determining symbol of neither of the clusters should be $parent(l)_1$, where $parent(l)_1$ is the first symbol of the parent node of the associated leaf node l.

In the table, the paths to be used by the two link edges from a leaf node are also indicated. The link edges are of dilation two and the paths used by the pair of link edges from a leaf to the two roots of the child FBTs do not share a host host edge.

Priority level	Allowed pairing			l_1	Link edge paths	
	Key element of SC1 (s^1)	Key element of SC2 (s^2)	Condition		link-1	link-2
1	2nd row	diagonal	$s^1 = s^2$	$s^1 = s^2$	$(n-1, n)$	$(n, n+1)$
2	1st row	diagonal	none	s^2	$(pos(s^1, l), n-1)$	$(n, n-1)$
2	1st column	2nd row	none	s^2	$(pos(s^1, l), n)$	$(n-1, n)$
2	1st row	2nd row	$s^1 \neq s^2$	s^2	$(pos(s^1, l), n-1)$	$(n-1, n)$
2	1st column	diagonal	$s^1 \neq s^2$	s^2	$(pos(s^1, l), n)$	$(n, n-1)$
3	1st column	1st column	none	any	$(pos(s^1, l), n)$	$(pos(s^2, l), n)$
3	1st row	1st row	none	any	$(pos(s^1, l), n-1)$	$(pos(s^2, l), n-1)$
3	1st row	1st column	none	s^1, if $s^1 = s^2$, else, any	$(pos(s^1, l), n-1)$	$(pos(s^2, l), n)$

Table 6.1: Allowed pairings of star clusters and their association with root FBT leaf nodes. s^1: determining symbol of star cluster SC1, s^2: determining symbol of star cluster SC2, l_1: the first symbol required in the leaf node l associated with the pair, link-1: link edges to root of child FBT in SC1, link-2: link edges to root of child FBT in SC2.

THEOREM 6.11 *If star cluster pairings are performed such that higher priority pairings are obtained first, at most one leaf node of a given first symbol will be required in the root FBT.*

PROOF. It is necessary for a leaf node to have a given first symbol only if in the associated star cluster pair the key element in at least one of the star clusters is from the second row or from the diagonal. In an SC_{n-1,k_n} there is at most one second row key element having a given deciding symbol and also at most one diagonal key element having the same deciding symbol. Star clusters with such a pair of key elements are paired with the highest priority. Therefore, there can be at most one star cluster pair which requires a leaf node with a given first symbol. As a result, it will be sufficient to have at most one leaf node, in the root FBT, with a given first symbol.

There is one case that we have ignored. It is necessary for an associated leaf node to have a given first symbol if the key elements in the star cluster pair are from first row and first column and their deciding symbols are identical. This leads to the possibility of requiring two leaf nodes with the same first symbol. But such a requirement can occur only if there is one row cluster with first row key element, one row cluster with diagonal key element, one column cluster with first row key element and one column cluster with second column key element and all four with the same deciding symbol. For this there must be two column clusters from the same column. But we never obtain two column clusters from the same column as $k_{n-1} > \frac{k_n}{2}$. Thus the possibility of requiring two leaf nodes with the same first symbol does not exist. $\square$

This theorem ensures that the method discussed in Section 6.3.1 for construction of root FBT satisfies the first symbols requirements of the leaf nodes.

LEMMA 6.8 *With the above pairing of the star clusters and the corresponding association of the leaf nodes with the pairs, the paths used by the link edges are free of edge-congestion.*

PROOF. The path used by a link edge can share an edge with - (a) the path used by the second link edge from the same leaf node or (b) the path used by the leaf node and its parent or (c) the path used by an edge between the child root and one of its child nodes. We have already seen from Table 6.1 that the case-(a) does not occur.

Let x be the parent of a leaf node l and S^j_{n-1} and S^k_{n-1} be the pair of substars associated to l. The path between a leaf node, l, and its parent node x consists of the single edge $(pos(x_1, l))$. According to the embedding scheme, $x_1 \neq s^j_{n-1} \neq s^k_{n-1}$. Therefore, the case-(b) does not occur. The case-(c) will not occur because the root nodes of the child FBTs are the only nodes in the paths used by the link edge that belong to the $SC_{n-2,k_{n-1}}$s hosting the child FBTs. $\square$

It is worth noting that the intermediate node on the path used by a link edge to an $SC_{n-2,k_{n-1}}$ remains within the S_{n-2} containing the root FBT, if the key element of the $SC_{n-2,k_{n-1}}$ from the 1st row or the 1st column. In that case, the intermediate node is congestion-free. However, when the key element in the $SC_{n-2,k_{n-1}}$ is from the 2nd row or the diagonal, the intermediate node belongs to an S_{n-2} which may be used in another $SC_{n-2,k_{n-1}}$. In that case, there is possibility for the intermediate node in the path to have congestion (node-congestion).

LEMMA 6.9 *If a $T_{h_{n-1}}$ is embedable in an $SC_{n-2,k_{n-1}}$ with dilation two and without edge-congestion, a T_{h_n}, $h_n = h_{n-1} + \lfloor log_2 \frac{n(n-1)-1}{k_{n-1}} \rfloor$ can be embedded in an S_n with the same dilation and without congestion.*

PROOF. We have seen that it is possible to obtain a distance-2 set of $\rho_{max} = 2^{log_2 \lfloor \frac{n(n-1)-1}{k_{n-1}} \rfloor}$ $SC_{n-2,k_{n-1}}$s in an S_n. As a result, an FBT of height $h_n = h_{n-1} + log_2 \lfloor \frac{n(n-1)}{k_{n-1}} \rfloor$ can be embedded in an S_n using either a single-CDS or a double-CDS step. In the embedding step, a congestion-free root FBT is embedded in the reference S_{n-2} and a child FBT is embedded one in each of the ρ $SC_{n-2,k_{n-1}}$s of the distance-2 set. Thus the child FBTs and the root FBT do not share any host edge. Further, the root FBT and the child FBTs are free of edge-congestion. According to Lemma 6.7 and Lemma 6.8, the link edges in the single-CDS embedding steps and double-CDS embedding steps are free of edge-congestion. Therefore, the embedding of the FBT of height h_n in S_n will be free of edge-congestion. Since the root FBT is of dilation one and the child FBTs are of dilation two and the link edges are of dilation two, therefore, the dilation of the embedding will be 2. ◻

THEOREM 6.12 *The FBT embeddings obtained through the double-CDS embedding scheme are free of edge-congestion.*

PROOF. The theorem follows from Lemma 6.4 and Lemma 6.9. ◻

The Lemma 6.9, gives the height of the largest FBT embedable in an S_n in terms of the height of the FBT embedable in S_{n-1} and the size of the star cluster, $SC_{n-2,k_{n-1}}$ required for it. Therefore, to compute the height of the largest FBT embedable in S_{n+1} it is necessary to know the size of the star cluster, SC_{n-1,k_n} required for the embedding in S_n. In Section 6.3.3, we have seen that $k_n = \lceil \frac{\rho k_{n-1}+1}{n-1} \rceil$, for $\rho > k_n \geq k_{n-1}$ or $k_n \geq \rho$. However, in case $\rho > k_n$ but $k_n < k_{n-1}$ it is not possible to obtain ρ $SC_{n-2,k_{n-1}}$s in an SC_{n-1,k_n} for the embedding of the ρ child FBTs, $T_{h_{n-1}}$s. In that case it is necessary to obtain 2ρ $SC_{n-2,k'_{n-1}}$s for embedding that many $T_{h_{n-1}-1}$s for the construction of the embedding in SC_{n-1,k'_n}. It is possible that $k_{n-1} < 2k'_{n-1}$. As a result, k'_n can be greater than k_n or even k_{n-1}. In that case it is preferable to perform the embedding in $SC_{n-2,k_{n-1}}$ as discussed earlier. We will presently see an example of such an exception. Due to such exceptions the expression $k_n = \lceil \frac{\rho k_{n-1}+1}{n-1} \rceil$ does not always hold.

We construct the Table 6.2 showing the heights of the FBTs embedable in star graphs of various sizes up to S_{19}. The star clusters required for the embeddings, the corresponding embedding step types are indicated. The table shows the embedding used in each step. It can be seen that in all the cases, except one, the largest embedding in the immediately lower level

Star	$h_{opt} =$	Double-CDS embedding			Embedding used	
S_n	$\lfloor log_2(n!) - 1 \rfloor$	h_n	SC_{n-1,k_n}	step type	h_{n-1}	$SC_{n-2,k_{n-1}}$
S_4	3	3	$SC_{3,3}$	basis	-	-
S_5	5	5	$SC_{4,4}$	S-CDS	3	$SC_{3,3}$
S_6	8	7	$SC_{5,4}$	S-CDS	5	$SC_{4,4}$
S_7	11	10	$SC_{6,6}$	D-CDS	7	$SC_{5,4}$
S_8	14	13	$SC_{7,7}$	D-CDS	10	$SC_{6,6}$
S_9	17	16	$SC_{8,8}$	S-CDS	13	$SC_{7,7}$
S_{10}	20	19	$SC_{9,8}$	S-CDS	16	$SC_{8,8}$
S_{11}	24	22	$SC_{10,7}$	D-CDS	18	$SC_{9,4}$
S_{12}	27	26	$SC_{11,11}$	D-CDS	22	$SC_{10,7}$
S_{13}	31	29	$SC_{12,8}$	S-CDS	26	$SC_{11,11}$
S_{14}	35	33	$SC_{13,10}$	D-CDS	29	$SC_{12,8}$
S_{15}	39	37	$SC_{14,12}$	D-CDS	33	$SC_{13,10}$
S_{16}	43	41	$SC_{15,13}$	D-CDS	37	$SC_{14,12}$
S_{17}	47	45	$SC_{16,14}$	D-CDS	41	$SC_{15,13}$
S_{18}	51	49	$SC_{17,14}$	D-CDS	45	$SC_{16,14}$
S_{19}	55	53	$SC_{18,14}$	D-CDS	49	$SC_{17,14}$

Table 6.2: Star clusters required for embedding FBTs of various sizes. S-CDS: Single-CDS, D-CDS: Double-CDS. h_{opt} is the theoretical optimal height of an FBT in an S_n.

star is used. It is found that the smallest star for which the this exception occurs is S_{19} for the embedding of T_{53}. The largest FBT embedding in an S_{18} is a T_{49}, which requires an $SC_{17,14}$. Based on the requirement of S_{17}s, 16 $SC_{17,14}$s should be available in an $S_{18,13}$. However, in this case, $k_n < k_{n-1}$. Therefore, it is not possible. On the other hand, the embedding of a T_{48} requires an $SC_{17,8}$ and to obtain 32 $SC_{17,8}$ an $SC_{18,15}$ is required. Therefore, the T_{49} cannot be embedded in an $SC_{18,13}$, instead an $SC_{18,14}$ has to be used.

Table 6.2 also shows h_{opt}, the theoretical optimal height of the FBT, going by the size of the star. For an S_n, $h_{opt} = \lfloor log_2(n!) \rfloor - 1$. It can be seen that the height of the FBT embedable with the scheme is 2 less than h_{opt}, in the worst case, for all practical purposes.

Table 6.3 gives a comparison of the heights of the full binary trees embedable using our single-CDS and double-CDS schemes with that using existing schemes [BHOS94, HOS95].

Star	Optimal	Schemes in [BHOS94, HOS95]				Our schemes (dilation 2)	
S_n	height	dilation 1	dilation 2*	dilation 4	dilation > 4	single-CDS	double-CDS
10	20	15	19	18	-	18	19
13	31	21	28	27	28	27	29
16	43	27	38	37	39	37	41
19	55	33	50	49	51	49	53

Table 6.3: A comparison of the heights of the full binary trees embedable with the various schemes. *Embeddings with this scheme have node congestion higher than one.

6.4 Embedding of pyramids

In the embedding scheme presented here a pyramid of height k, $P(k)$, is embedded in an S_{k+2}. The scheme is recursive in nature. An embedding of a quad $P(1)$ in S_3 is taken as the basis. The scheme makes extensive use of the peer node imaging technique.

6.4.1 The structure of a pyramid

A pyramid $P(L)$ is an $L+1$ layered interconnection network in which each layer consists of a mesh. The top most layer is the L-th layer and consists of a single node called the *apex*. The lower most layer (0th layer) is called the base of the pyramid. The size of the mesh in layer $l-1$ is normally larger than that of the mesh in layer l. The ratio between the sizes of the meshes in layer l and $l-1$ is called the *convergence factor* of level l of the pyramid. The convergence factor is the product of the row convergence factor and the column convergence factor. The most common type of pyramid is the *quad-pyramid* which has a convergence factor of four. Both row convergence factor and column convergence factor being two. The size of the mesh at layer l is $l^2 \times l^2$. A node (i, j) in the l-th layer is connected to four nodes $(2i-1, 2j-1), (2i-1, 2j), (2i, 2j-1)$, and $(2i, 2j)$ on the layer $l-1$ and the node $((i+1)/2, (j+1)/2)$ on the layer $l+1$ in addition to the four adjacent nodes in the mesh of the layer l.

In general, a pyramid can have a convergence factor other than four. For example, a *bin-pyramid* has a convergence factor of two. It is also possible for the convergence factor to be variable so that the convergence factor may differ for different pairs of layers. The row convergence factor and the column convergence factor may also differ. For example, in a bin-pyramid the row and column convergence factors are 1 and 2 alternately for alternate pairs of layers.

Considering a pyramid $P(k)$ of variable convergence let us use the notation c_l to denote the convergence factor between the levels l and $l-1$. Loosely, we shall refer to c_l as the convergence factor of level l and sometimes as the convergence factor of $P(l)$. Note that c_l is the product of the row and column convergence factors of level l. A pyramid $P(k)$ can be thought to be consisting of c_k $P(k-1)$s. A $P(k)$ can be factored into its constituent $P(k-1)$s by removing the edges to its apex and also other edges in all levels below apex. The $P(k-1)$s we shall refer to as the *child* $P(k-1)$s of the $P(k)$. The edges from the apex nodes of a pyramid we shall refer to as the *apex edges*. The apex nodes of the child $P(k-1)$s are directly connected to the apex of the $P(k)$ by the apex edges. The apex nodes of the child $P(k-1)$s form a mesh.

6.4.2 The basis embedding

An embedding of a quad $P(1)$ in S_3 is used as the basis for the embeddings of the higher level pyramids. A quad $P(1)$ is embedded in an S_3 with dilation three as shown in Figure 6.17. The rank 1 node hosts the apex node of $P(1)$ and the nodes $2, 3, 4$, and 5 host the four base nodes. The rank 0 node is kept unused.

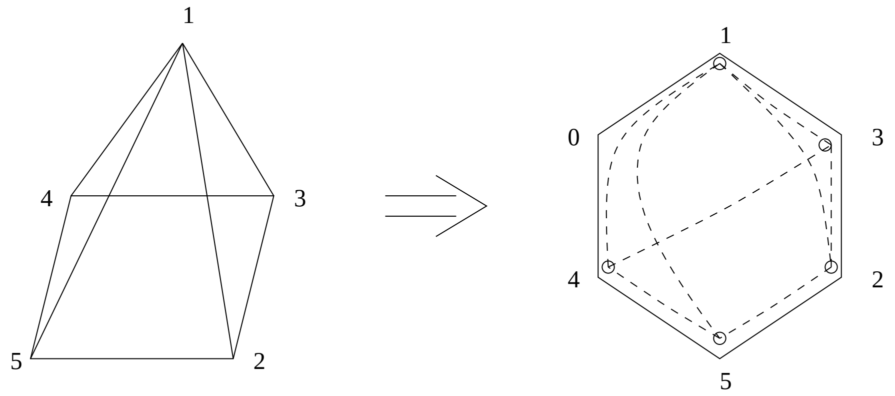


Figure 6.17: The basis embedding: $P(1)$ in S_3.

6.4.3 Embedding of the child pyramids

In the embedding of a $P(k)$, $k > 1$, the c_k, $c_k \leq k+2$, chlidren $P(k-1)$s are placed in that many sibling S_{k+1}s such that they are substar peer node images of each other. Therefore the corresponding nodes in the $P(k-1)$ are at a distance of at most four from each other. As a result the apex nodes of the $P(k-1)$s can be arranged in an arbitrary logical mesh of dilation

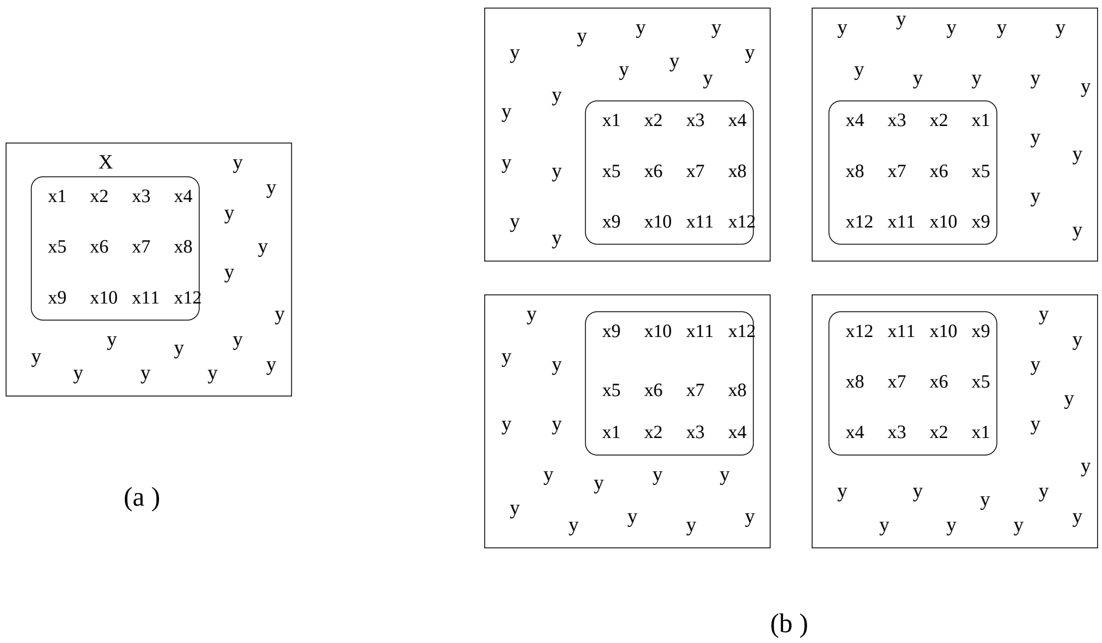


Figure 6.18: Peer adjacency arrangement. (a) An array arrangement of 'x' elements in an S_i. (b) A 2×2 array arrangement of four S_i.

---

four. For the meshes at different levels of the embedded $P(k)$, the peer adjacency arrangement as defined below is used to achieve a dilation of four.

DEFINITION 6.7 **Peer adjacency arrangement**
Consider a two dimensional logical array arrangement of a set of nodes X in an S_i as shown in Figure 6.18(a). Now if a set of sibling S_is are arranged logically in a two dimensional array form such that the peer node sets of X in the S_is from a logical mesh and the nodes in two S_is adjacent in the arrangement are peers. Such an arrangement of node sets in sibling S_is is called a peer adjacency arrangement.

Figure 6.18(b) shows such an arrangement. It is easy to see that in the logical mesh of the arrangement an edge between two nodes in two different S_is has a dilation of at most four. By using the technique recursively, it is possible to form logical meshes of dilation four in a star graph.

6.4.4 Placement of the apex node

As per the embedding of the child $P(k-1)$s their apex nodes are peer nodes in S_{k+1}s. In the embedding of $P(2)$, chose the S_3 containing the peer-0 centre of the hosts of the apex nodes of the $P(1)$s to host the apex. It may be recalled that the node 0 in the S_3 is not used in the $P(1)$ embedding and it is also adjacent to the node hosting the apex of the $P(1)$ (the peer 0-centre of the apex host set). Therefore, this node is at a distance of at most three from the $P(1)$ apex nodes. The apex of the $P(2)$ is placed at this node.

In the embedding of $P(3)$ in S_5 the value of c_3 is limited to four so that one of the S_4s has no child $P(2)$. The S_4 chosen to be kept free is the one that contains the peer-0 centre of the hosts of the apex nodes of the child $P(2)$s. The apex of the $P(3)$ is placed at this peer 0-centre node. As a result the dilation of the apex edges is at most two.

Placement of the apex nodes for $k > 3$

In the embeddings of any $P(k)$, $3 < k \leq 26$, the apex node is placed in one of the S_4s hosting the apex nodes of the embedded $P(3)$s. None of the other nodes in the S_4 is used. A Hamiltonian path can be constructed in this S_4 with the host of the $P(3)$ apex node as the starting node [JLD91]. In the embedding of $P(4)$ in S_6 the apex node is placed in one of the c_4 S_4s hosting the apex nodes of the $P(3)$s. The chosen S_4 is the one that contains the peer 0-centre of the hosts of the $P(3)$ apex nodes. The free node in the Hamiltonian path adjacent to the host of the $P(3)$ is used to host the apex of the embedded $P(4)$. In the embedding of $P(5)$ in S_7 the apex node is placed at the free node in the Hamiltonian path adjacent to the node hosting the apex of the $P(4)$ embedded in one of the c_5 S_4s hosting the apex nodes of the child $P(4)$. Here again the S_4 chosen is the one that contains the peer 0-centre of the hosts of the apex nodes of the $P(4)$. In general, in the embedding of $P(k)$, $3 < k \leq 26$, the S_4 containing the peer 0-centre of the hosts of the $P(k-1)$ apex nodes is chosen for placement of the $P(k)$ apex. Further, the apex is placed at the free node in the Hamiltonian path and adjacent to the node which is a peer 0-centre and hosts the apex of one of the $P(k-1)$s. The dilation of the apex edges will be at most three as the peer 0-centre of the host of the apex nodes of child $P(k-1)$s is adjacent to the host of the $P(k)$ apex. The k value is limited to 26 as the Hamiltonian path in the S_4 has 24 nodes so that it can host the apex nodes of $P(3)$ to $P(26)$.

A $P(26)$ is large enough for all practical purposes. However, the scheme can be extended for larger k values based on the following observation. Five of the S_4s in an S_6, in the embedding of a $P(4)$, hosting the apex nodes of $P(3)$s form an S_5 with the composition dimension sequence

$\{2,3,4,6\}$. In each of these S_4s only one node is used by a $P(3)$ and these used nodes are peers. It is simple to construct a sufficiently long (beyond 100) path in the S_5 excluding the peer nodes hosting the apex nodes of the $P(3)$s. In the path, the peer-0 centre among the used nodes should be the starting node. After obtaining such a path, the rest of the embedding scheme will remain the same.

6.4.5 Edge-congestion

THEOREM 6.13 *The edge-congestion in the embeddings of pyramids is c_{max}, the maximum of the convergence factors of the pyramid.*

PROOF. While obtaining the embedding of a $P(k)$ from that of $P(k-1)$, the apex edges are routed through the $P(k-1)$ apex which is a peer 0-centre. Therefore the c_{k-1} apex edges may use the edge between the apex of the $P(k)$ and the peer 0-centre node. It is possible that one of the mesh edges between two $P(k-1)$ apex nodes (one being the peer 0-centre) may also be routed through the edge under consideration. In that case, one of the apex edge of the $P(k)$ will not be routed through the peer 0-centre. Therefore the congestion of the edge between the peer 0-centre and the apex of $P(k)$ will be c_k.

The third substar routing between peer nodes can be used for the mesh edges. The edge-congestion in this case is two. Therefore, the congestion of the embedding is c_{max}. □

For a quad pyramid, the edge-congestion will therefore be four.

6.4.6 Dilation and expansion

In the construction of the pyramid embedding the apex nodes are placed such that in no case the dilation of an apex edge is more than three. It has been seen that the dilation of a mesh edge is at most four. Therefore the over all dilation of the embeddings through the scheme is four.

The expansion of an embedding through this scheme is dependent on the convergence factors of the embedded pyramid. The size N_k of the pyramid $P(k)$ with expansion factor c_l at level l is approximately $\prod_1^k c_l$. For a quad pyramid $N_k \approx 4^k$. The size of S_{k+2} required for embedding a $P(k)$ is $(k+2)!$. Therefore, the expansion of the embedding of a general pyramid $P(k)$ is $E_g \approx \frac{(k+2)!}{\prod_1^k c_l}$ and that for a quad pyramid is $E_q \approx \frac{(k+2)!}{4^k}$. The expansion in the embedding of a quad pyramid grows rapidly with the size of the pyramid. This is expected due to the mismatch in the growth rates of the two structures. If the convergence factor of the pyramid is allowed to grow at the same rate as the star graph the expansion factor remains close to unity as can be seen from the expression for E_g.

6.5 Summary

In this chapter we have presented embedding new techniques for cycles, tori, full binary trees and pyramids. We have presented two schemes for embedding cycles. The first scheme allows construction of cycles of length l_c, $(i-1)! < l_c \leq 2(i-1)!$ in an $SC_{i-1,2}$. An earlier method given by Jwo et al. [JLD91] required at least an $SC_{i-1,3}$ for construction such a cycle. The second scheme allows construction of two cycles of length $l_c \leq \frac{i!}{2}$ each in an S_i. The scheme is also modified to construct two parallel cycles of the same length in an S_i. Here, one of the cycles is an approximate one, in the sense that a small fraction of the edges in the cycle are of dilation three. The method of construction of parallel cycles is useful in obtaining embeddings for other cycle based structures such as torus, cube connected cycles etc.

We have presented a scheme for embedding torus of arbitrary size and dimension on the star graph. A set of methods, namely, i-th symbol swap transformation and substar peer node imaging, for generating parallel cycles are used to obtain the column cycles of the torus. The row cycles of the torus are obtained by joining the peer nodes of the parallel cycles. Some of the column cycles are approximate and therefore have some edges dilated to three edge paths. But such edges constitute a very small fraction ($\approx \frac{1}{i!}$). The row cycles are of dilation four. The congestion and expansion depend upon the routing of the row edges. We present two schemes for the routing of the row edges- confined routing and third substar routing. With the confined routing the embedding has edge congestion of $O(n)$ and expansion four. In the third substar routing the embedding has edge congestion two and expansion $O(n)$.

We have proposed two schemes for embedding full binary trees on the star graph with dilation two. The schemes use the concepts of composition dimension sequence an a two dimensional arrangement of subgraphs. The embeddings with the first scheme are congestion-free. In this scheme, FBTs of height $h(n) = (n+1)\lfloor log_2 n\rfloor - 2^{\lfloor log_2 n\rfloor+1} + 1$ can be embedded in an S_n. This scheme improves in dilation over an earlier scheme [BHOS94] by a factor of two. The second scheme produces embeddings free of edge-congestion. With this scheme, FBTs of height at least $\lfloor log_2(n!)\rfloor - 3$ can be embedded in an S_n for values of n up to 19. That is, the height is at most 2 less than the theoretical optimal height.

We have presented a recursive scheme for embedding pyramids in star graphs based on the peer node imaging technique. The scheme requires an S_{k+2} for embedding a pyramid $P(k)$.

The dilation of the embedding is four and congestion is c_{max}, where c_{max} is the maximum of the convergence factors of the pyramid. The expansion depends on the convergence factor(s) of the pyramid. In the embedding of a quad pyramid in an S_n the expansion is $\approx \frac{n!}{4^{n-2}}$. The expansion of the embedding of a quad pyramid grows with the size of the pyramid. This is due to the mismatch in the growth rates of the two structures.

Chapter 7

Algorithms

In this chapter, we present algorithms belonging to three different classes. These are: global communication algorithms, order-preserving ascend/descend algorithms and pipelined algorithms. In all the algorithms presented here, we assume the *multi-port* communication model for the nodes.

In a message passing oriented parallel computer, the efficiency of an algorithm depends to a large extent upon communication efficiency. There are certain global communication operations such as broadcast, scatter, gossip, accumulate etc. [SS87] that are often required in parallel algorithms. The cost of performing these operations depends upon the interconnection structure used in building the computer system. The importance of the problem has motivated large number of studies in this direction for different interconnection networks such as the hypercube [SS87, JH89]. In this chapter we consider the problem of performing these operations in a star graph based interconnection network and present simple and efficient algorithms for these.

In Chapter 4 we presented four different communication paradigms for the star graph based interconnection network. These can be used for the global communication operations on the star. The two paradigms transmit-and-domain-distribute (TDD) and domain-distribute-and-transmit (DDT) are particularly suitable for some of these communication operations. We use the TDD paradigm to develop two asymptotically optimal broadcast algorithms, one for short messages and the other for long messages. It is also used for an asymptotically optimal gossiping algorithm. The DDT paradigm is used for developing asymptotically optimal algorithms for the broadcasting and the scattering operations. We use the HOPC paradigm for a simple and elegant algorithm for the single node accumulation operation.

A well known class of parallel algorithms that uses the divide and conquer strategy is the ascend/descend [PV81] class. In an algorithm of this class the computation is carried out in multiple hierarchical levels. Computation at one hierarchical level forms a parallel step of the algorithm. An entire ascend phase of the algorithm consists of climbing up the hierarchy step by step from bottom to top. Similarly, a descend phase consists of the steps of climbing down the hierarchy from top to bottom. In general, an ascend/descend algorithm consists of one or more ascend and/or descend phases. The general ascend/descend class can be further categorized into two subclasses based on the communication pattern at any level of the hierarchy. In general, at a level, two or more groups of nodes communicate. If this occurs in an order preserving manner, the corresponding algorithm is referred to as an order preserving ascend/descend algorithms. It contains algorithms such as FFT, matrix multiplication, and bitonic sort. In the other case of the ascend/descend class, nodes of non-identical ranks in the groups are allowed to communicate. This class of algorithms form the non-order preserving ascend/descend class. Instances of efficient algorithms of non-order preserving ascend/descend class for the star graph based interconnection networks include minimum spanning forest(MSF) [AQ93] and prefix sum computation [AQS91]. But, not much success, so far, has been achieved in the development of algorithms of comparable efficiency for the order preserving class. Interestingly, our hierarchical ranking scheme and the HOPC paradigm provides scope for development of efficient algorithms of this class. As examples, in Section 7.2, we present order preserving ascend/descend algorithms for FFT and matrix multiplication.

An equally well known class of parallel algorithm is that of the pipelined algorithms. The pipeline structured ranking scheme presented in Chapter 3 provides a convenient framework for developing pipelined algorithms based on the adjacent domain communication (ADC) paradigm for the star. In Section 7.3, we present pipelined algorithms for one dimensional $n!$-point FFT and two dimensional $n! \times n!$-point FFT on the star as examples.

A common component of many parallel computation is the well known prefix computation. An efficient implementation of prefix computation is therefore very essential. Akl et al.[AQ93] present an algorithm for prefix computation on star graph based on the group-copy paradigm with the multi-port communication model. Berthome et al. [BFP92] present an efficient algorithm for prefix computation with the single-port communication model. In Section 7.4, we present a congestion-free algorithm for prefix computation on the star based on the TDD communication paradigm with the multi-port communication model. The time complexity of the algorithm is $O(nlogn)$ for input size $n!$, which is same as that for the algorithm in [AQ93]. However, the algorithm presented here is congestion-free.

7.1 Global Communication Algorithms

Some of the important considerations in global communication algorithms are minimization of communication time, avoidance of channel contention and simplicity. Avoidance of channel contention is important because it may otherwise significantly increase the actual communication delays Moreover, contention may even lead to deadlock situations. In addition to these, there are also other special considerations specific to the operation involved. For instance, in the gossiping and accumulation operations one message unit from each of the nodes in the network gets collected at a node. In a large network the collected data size is expected to be large. It is not always sufficient to collect the message units in arbitrary order. For ensuring efficient processing it is often necessary that the message units are collected at a node in an easily accessible order. For a good gossiping (or accumulation) algorithm this ordering should be integrated into the collection operation. In the scattering operation, on the other hand, a node has a possibly unique message unit for each of the nodes in the network. The sequence of message units need to be partitioned and different blocks of messages transmitted to different destinations. If the algorithm does not allow easy partitioning of the message blocks these operations may lead to significant processing cost

The global communication operations can be categorized into two classes: data distribution and data collection. The operations such as one-to-all broadcasting and one-to-all scattering come under the data distribution class. The operations such as single-node accumulation come under the data collection operation. However, the operations such as gossiping can be placed in either of the classes. Note that the operation of gossiping can be thought to be consisting of each processor node performing a broadcasting operations. As an alternative, it can also be thought to be consisting of each processor node performing an accumulation operation.

Data distribution and collection operations can be performed on the star graph using any of the four communication paradigms discussed in Chapter 4. However, the two paradigms *transmit and domain distribute* (TDD) and *domain distribute and transmit* (DDT) are particularly suited for these operations, except for single node accumulation. The *hierarchical order preserving communication* paradigm can be used for an elegant algorithm for the single node accumulation operation.

Data distribution by a single node using the TDD paradigm (see Section 4.3) can be performed hierarchically starting at substar level 1 and terminating at substar level $n-1$ when the entire set of nodes in S_n has received the data. From this point of view the operation with the TDD paradigm can be said to follow an *ascend* approach. Using the DDT paradigm (Section 4.4) also data distribution by a single node can be performed hierarchically. But in this case the operation starts at substar level $n-1$ and terminates at substar level 1. Therefore, with the DDT paradigm the data distribution operation can be said to follow a *descend* approach.

We shall assume the *multi-port* communication model for the nodes in deriving the time complexity of the algorithms presented. Further, we will ignore the start-up time as its contribution is going to be a constant additive factor.

7.1.1 One-to-all broadcast

One-to-all broadcast is a data distribution operation in which a source node sends a message to all the other nodes in the network. Since the diameter of an S_n is $\lfloor \frac{3}{2}(n-1) \rfloor$ the length of the longest path the broadcast message has to traverse is at least $\lfloor \frac{3}{2}(n-1) \rfloor$. Therefore the total hop time required is at least $\lfloor \frac{3}{2}(n-1) \rfloor t_h$ (refer Section 1.1.1). For a message of length M the total transfer time is at least $\lfloor \frac{3}{2}(n-1) \rfloor M t_w$. Hence the lower bound on time complexity of the broadcast operation on S_n is $\lfloor \frac{3}{2}(n-1) \rfloor (t_h + M t_w)$. This lower bound holds for broadcast of very short messages. For long messages it is possible to break-up the the message into smaller sub-units and transmit them along parallel paths to achieve a smaller time complexity. We shall discuss this in the following section.

In this section we present two algorithms suitable for broadcasting short messages on the star graph. The first algorithm is based on the TDD paradigm while the second uses the DDT paradigm.

THEOREM 7.1 *A message of length M can be broadcast on an S_n using the TDD paradigm in time* $(2n-3)(t_h + M t_w)$.

PROOF. The source node that contains the message constitutes an S_1. It can spread the message to the nodes in its parent S_2 in two steps (Actually in one step, because the domain of a node in an S_1 consists of itself and therefore the second step is not required) using the TDD paradigm. The S_2 nodes then can spread the message to the nodes in the parent S_3 in two steps. This process can be repeated $n-1$ times. In the $(n-1)$-th iteration the nodes in the S_{n-1} containing the source node will spread the message to the entire S_n. The longest path traversed by a copy of the message is $2(n-2)+1 = 2n-3$. Therefore the total hop time is $(2n-3)t_h$. The total transfer time will be $(2n-3)M t_w$. Hence, the total time required in the operation is $T = (2n-3)(t_h + M t_w)$. □

Figure 7.1 shows the tree of broadcast paths for S_4 with the node 1234 as the source. Such

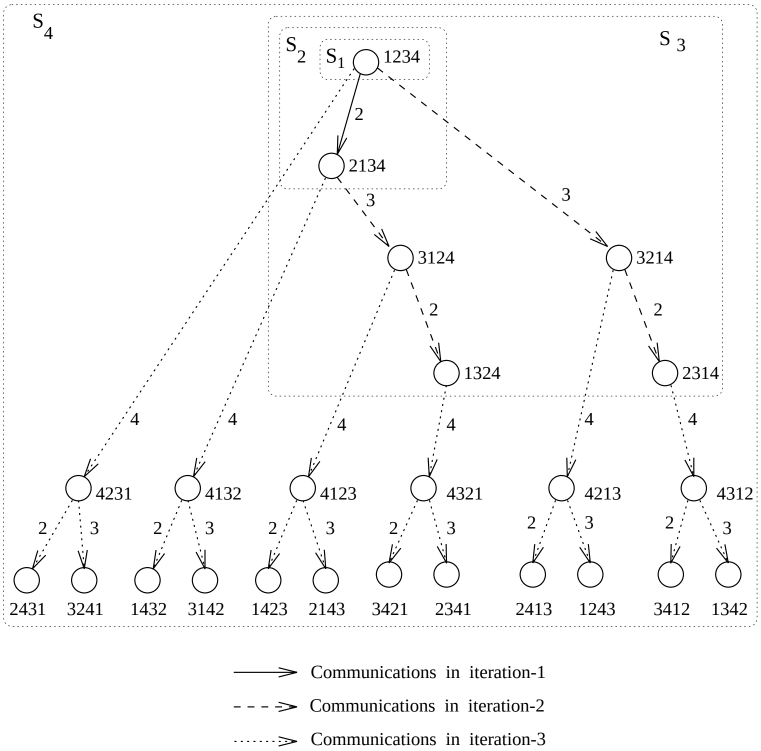


Figure 7.1: TDD-broadcast tree in S_4 with 1234 as the source node.

---

a tree shall be called a *TDD-broadcast tree.* The corresponding algorithm Star_Broadcast_TDD is given below.

Algorithm 7.1 **Star_Broadcast_TDD**

Input : A source node having a message.
Output : Each node in S_n containing a copy of the message in the source node after 2n-3 parallel communication steps.

1. for i = 1 to $n-1$ do

1.1 Each node having a copy of the message transmits a copy of it to the adjacent (i,k)-centre in sibling S_is along dimension $i+1$ edge.

1.2 if $i > 1$ then

Each node receiving a copy of the message along the dimension $i+1$ edge in step 1.1 transmits a copy of it in parallel to each of the nodes in its domain in S_i.
*/*i.e. transmit to all the adjacent nodes along dimensions 2 to i. */*

Now let us consider performing the operation using the *domain distribute and transmit* paradigm.

THEOREM 7.2 *A message of length M can be broadcast on an S_n using the DDT paradigm in time* $(2n-3)(t_h + Mt_w)$.

PROOF. The source node can spread the message to one node in each of the S_{n-1}s in two steps using the DDT paradigm (see Section 4.4). Each of these n nodes receiving a copy of the message will spread it to one node in each of the S_{n-2}s in two steps. The process can be repeated for $n-1$ times so that after the final iteration each S_1 (*i.e.* each node) will have a copy of the message.

It is easy to see that the length of the longest path traversed by the message will be $2n-3$. Therefore it follows that the time taken by the operation will be $(2n-3)(t_h + Mt_w)$. □

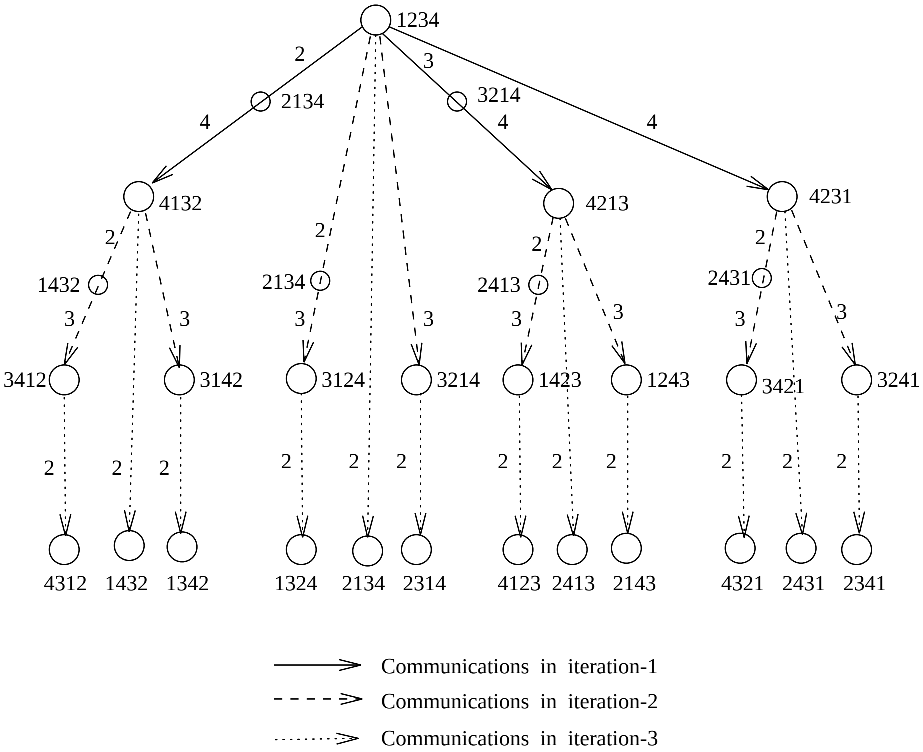


Figure 7.2: Graph of broadcast paths obtained through DDT paradigm in S_4 with 1234 as the source node.

The graph of broadcast paths in an S_4 with the node 1234 as the source is shown in Figure 7.2. It can be seen that some nodes receive the message multiple number of times. A node receiving the message as a domain node only forwards the message without accepting it. A node accepts the message only when it receives the message as an (i,k)-centre in the $(n-i)$-th iteration when the source node belong to S_i^k (*i.e.* the message arrives along dimension $i+1$ edge). This aspect of the algorithm makes it simpler compared to the one presented in [SLC93] even though both use the same set of paths for a given source node.

The corresponding algorithm Star_Broadcast_DDT is given below.

ALGORITHM 7.2 **Star_Broadcast_DDT**

Input : A source node having a message.
Output : Each node in S_n containing a copy of the message in the source node after 2n-3 parallel communication steps.

1. for $i = n-1$ to 1 do
  *1.1 Each node having a copy of the message transmits in parallel a copy to each of the nodes in its domain in S_i /*i.e. along dimensions 2 to i */*
  1.2 Parallel do
    1.2.1 Each node that receives a copy of the message in step-1.1 forwards it to the adjacent node along dimension $i+1$ edge.
    1.2.2 Rest of the nodes having a copy of the message transmits a copy to the adjacent node along dimension $i+1$ edge.

7.1.2 Multi-path one-to-all broadcast

In broadcasting very long messages the transfer time determines the total time taken by the operation. In such a case it is necessary to break the message into smaller sub-units and transmit them along parallel paths so as to achieve a minimal communication time. The minimum time required for broadcasting a message of length M in a symmetric interconnection network of diameter D, if performed with a single path to a single destination, is $D(t_h + Mt_w)$. On the other hand, if d parallel paths to each destination are used and over each path a message sub-unit of length $\frac{M}{d}$ is transmitted, the time required will be $D(t_h + \frac{M}{d}t_w)$. This is the lower bound on communication cost for a broadcasting operation on a symmetric interconnection network with degree d. On an S_n this lower bound is $\lfloor \frac{3}{2}(n-1) \rfloor (t_h + \frac{M}{n-1}t_w)$.

We shall use the concept of *orthogonality* of composition dimension sequences to obtain a set of $n-1$ TDD-broadcast-trees. These broadcast trees are then used in a multi-path broadcast algorithm for long messages.

Definition 7.1 **Orthogonal set of composition dimension sequences**
A set of composition dimension sequences for S_n is said to be orthogonal if the composition dimension for S_i, $2 \leq i \leq n$, is distinct for each of the sequences in the set.

Since an S_n has $n-1$ dimensions, in an orthogonal set of composition dimension sequences there can be at most $n-1$ members. Such an orthogonal set can be obtained through rotation of any given composition dimension sequence. For example, for S_4, through rotation of the primary composition dimension sequence (2,3,4) we can get the orthogonal set {(2,3,4),(3,4,2),(4,2,3)}. Figure 7.3 shows S_4 and its substars for this orthogonal set of composition dimension sequences.

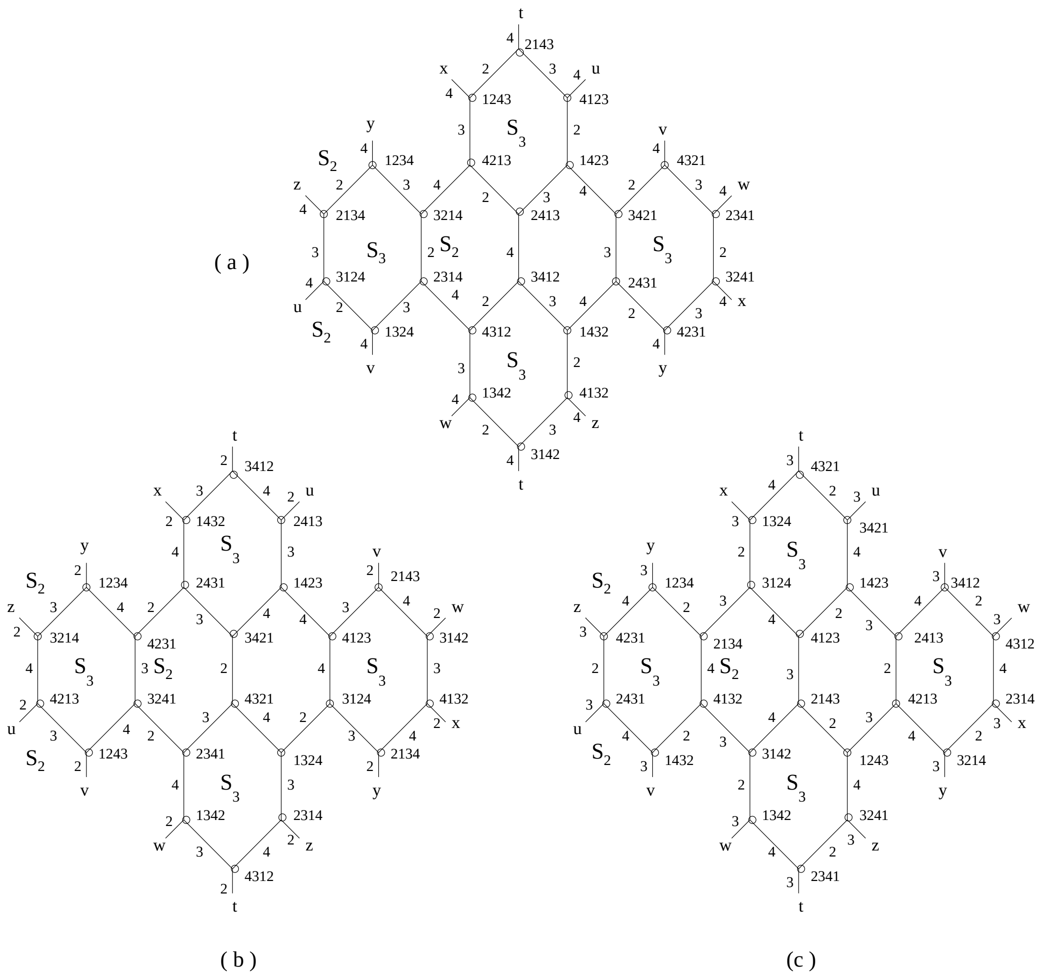


Figure 7.3: S_4 and its substars with the orthogonal set of CDSs $\{(2,3,4),(3,4,2),(4,2,3)\}$. (a) CDS $(2,3,4)$, (b) CDS $(3,4,2)$, (c) CDS $(4,2,3)$. In each case we show the S_3s in the S_4 and the S_2s in one of the S_3s.

In the algorithm Star_Broadcast_TDD in iteration i an S_i spreads the message to the parent S_{i+1} in two steps. In the transmit step the dimension d_{i+1}, the composition dimension for S_{i+1}s, is used[1]. We shall refer to the dimension d_{i+1} edges used in the *transmit step* as the *transmit edges* of iteration-i. In the *distribute step* the edges along dimensions d_2 to d_i are used by a distributing node for distributing the message among the domain nodes. We shall refer to the nodes that perform the distribution operation in the distribute step as the *distribute nodes* of the iteration.

LEMMA 7.1 *The first symbol of the distribute nodes in iteration-i of the algorithm Star_Broadcast_TDD is $v_{d_{i+1}}$, where $v_{d_{i+1}}$ is the d_{i+1}-th symbol of the source node v.*

PROOF. It can be observed that the dimension d_{i+1} is used for the first time in the operation in the transmit step of the i-th iteration. The generator $g_{d_{i+1}}$ is used for the first time at this step in the generation of the TDD-broadcast tree with the source node as the root. This will lead to positioning of $v_{d_{i+1}}$ as the first symbol in the distribute nodes as the transmit step leads to the distribute nodes. Hence follows the lemma. $\square$

For the multi-path broadcast algorithm we need to have $n-1$ broadcast trees so that the message can be divided into $n-1$ sub-units and each of the trees used to broadcast one of these sub-units. To achieve optimality it is also necessary that these broadcast trees are mutually contention-free so that additional delays are avoided.

DEFINITION 7.2 *Any two broadcast trees are said to be* **mutually contention-free** *for a synchronous operation if at any given transmission step the two trees have no common edge.*

THEOREM 7.3 *If an orthogonal set of composition dimension sequences is used for composing the substars at the different levels, the TDD-broadcast trees obtained using these sequences are mutually contention-free.*

PROOF. Due to the orthogonality of the composition dimension sequence set the dimension of the edges used in the transmit step (*i.e.* the composition dimension) at any given iteration is distinct for each of the trees. Therefore, no two trees have a common transmit edge in any given iteration i.

Now if we can prove that in any given iteration i each tree has a distinct distribute node set the proof will be complete. From Lemma 7.1 we know that the first symbol of the distribute

[1]Here the substars are not restricted to PCDS ones. The notation d_{i+1} is used to denote the composition dimension of the S_{i+1}s (refer Section 3.2). Due to symmetry of the star graph the Theorem 4.9 and the TDD paradigm will be valid for SCDS substars also.

nodes in iteration-i in a TDD-broadcast tree is $v_{d_{i+1}}$. Since d_{i+1} is distinct for each of the set of trees considered, the first symbol of the set of distribute nodes for the iteration will also be distinct for each of them. Therefore the distribute node set for any given iteration i will be distinct in each tree. Thus the transmit and domain distribute broadcast trees obtained using the orthogonal set of composition dimension sequences are mutually contention-free. $\square$

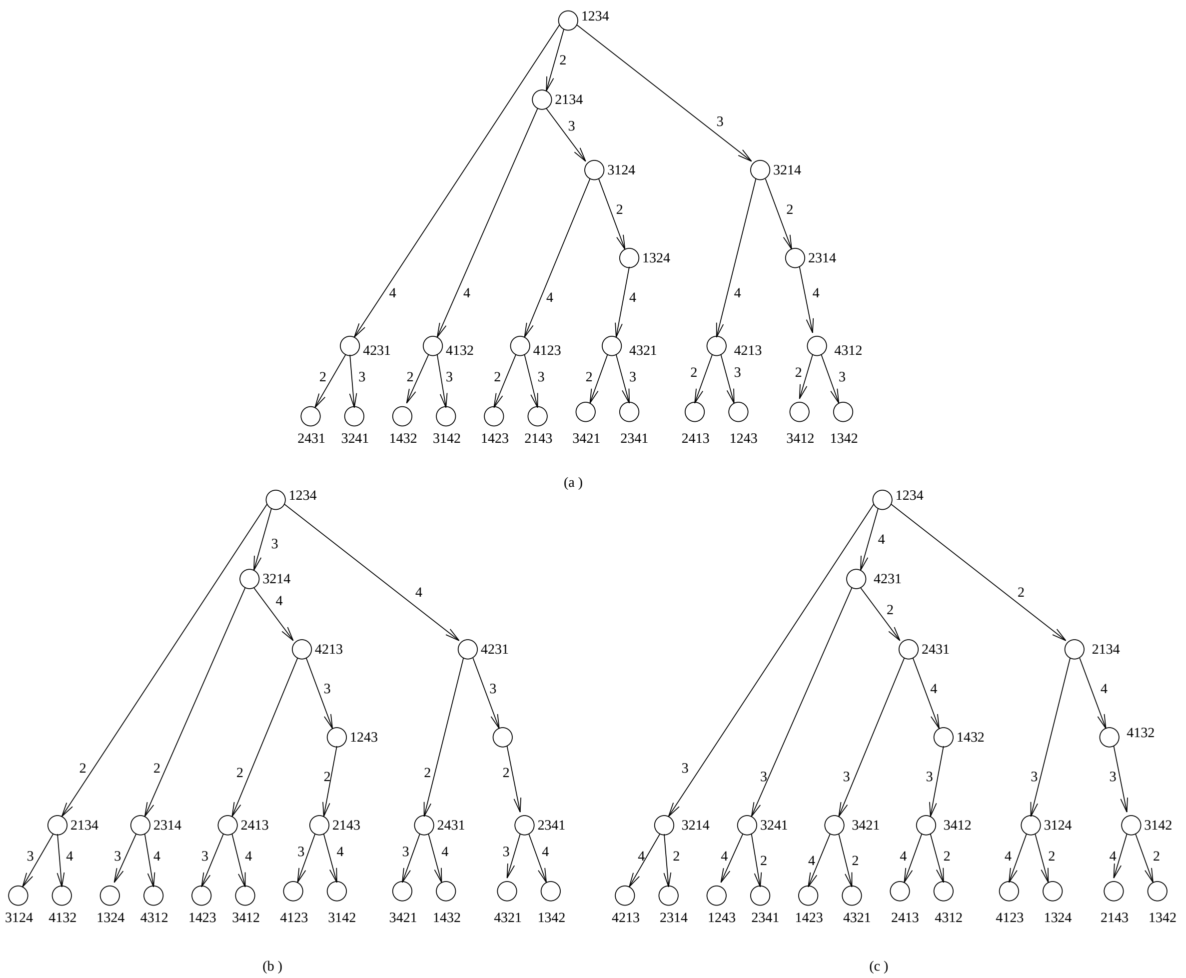


Figure 7.4: TDD-broadcast trees in S_4 with the orthogonal set of CDSs $\{(2,3,4),(3,4,2),(4,2,3)\}$. (a) with CDS (2,3,4), (b) with CDS (3,4,2), (c) with CDS (4,2,3).

Figure 7.4 shows three contention free TDD-broadcast trees for S_4 obtained using the orthogonal set of composition dimension sequences {(2,3,4),(3,4,2),(4,2,3)}. The synchronous algorithm Star_Broadcast_Mpath using such a set of mutually contention-free TDD-broadcast trees is given below.

ALGORITHM 7.3 **Star_Broadcast_Mpath**

Input : A source node having a message of length M.
Output : Each node in S_n containing a copy of the message in the source node after $2n-3$ parallel communication steps.

1. The source node partitions the message into $n-1$ sub-units each of length $M/(n-1)$ and each of these is placed in a separate packet. Each packet is assigned a composition dimension sequence from a chosen orthogonal set and is added as a header to the packet along with the sequence number of the message sub-unit. The source node transmits a copy of each of these packets to the adjacent node along the corresponding dimension d_2 edge.

2. for $i = 2$ to $n-1$ do

2.1 Each node having a copy of the message transmits a copy to the adjacent (i,k)-centre in sibling S_i along dimension d_{i+1} edge.

2.2 Each node receiving a copy of the message along the dimension d_{i+1} edge transmits a copy of it in parallel to each of the nodes in its domain in S_i. / i.e. transmit to all the adjacent nodes along dimensions d_2 to d_i. */*

Since $n-1$ TDD-broadcast trees are used and the trees are mutually contention-free, we have the following theorem.

THEOREM 7.4 *The time required for broadcasting a message of length M with the algorithm Star_Broadcast_Mpath is* $(2n-3)(t_h + \frac{M}{n-1}t_w)$.

7.1.3 One-to-all scatter

In the one-to-all scatter operation a particular node sends data elements of equal size to every other node in the network. It differs from broadcast in that the data element transferred to each node is possibly unique. Therefore, the number of data elements involved in this operation in an S_n is $n!$ (including one data element for the source node itself).

In a one-to-all scatter operation on S_n the source node has to transmit $n!-1$ message units, one to each of the other nodes in the network. If the transmission of these $n!-1$ message units is divided equally over the $n-1$ channels from the source nodes, the transfer time will be at least $\frac{n!-1}{n-1}Mt_w$. For a message unit to reach the farthest node from the source it will require $\lfloor\frac{3}{2}(n-1)\rfloor$ hops. Therefore the minimum hop time required is $\lfloor\frac{3}{2}(n-1)\rfloor t_h$. The lower bound on the time complexity of the scatter operation on S_n is $\frac{n!-1}{n-1}Mt_w + \lfloor\frac{3}{2}(n-1)\rfloor t_h$.

The scattering operation can be performed on an S_n using the DDT paradigm as follows. Initially the source node contains $n!$ data elements, one for each of the nodes in the S_n. The source node can partition the data into n blocks one for each S_{n-1} and each containing $(n-1)!$ data elements. Then it distributes these data blocks to the sibling S_{n-1}s by transferring a block to one of the nodes in each S_{n-1} through a DDT step. After a block reaches one of the nodes in an S_{n-1}, it is partitioned into $n-1$ sub-blocks each containing $(n-2)!$ data elements meant for the nodes of an S_{n-2} in the S_{n-1}. Once these sub-blocks reach their respective S_{n-2}s they are further partitioned for the nodes in the S_{n-3}s in the S_{n-2}s. This process of partitioning and transmitting is continued until a block contains only a single data element and for a single node which receives it. The partitioning problem is trivial if the data elements in the source are initially ordered according to the ranks of the destination nodes. In this arrangement the required number of consecutive data elements constitutes a block. The j-th block is meant for the j-th sibling substar. This follows from the Property 3.2 and Property 3.3 of the hierarchical ranking scheme in Section 3.4.1. The corresponding synchronous algorithm Star_Scatter_DDT is given below.

ALGORITHM 7.4 **Star_Scatter_DDT**

Input : A source node having $n!$ message units one for each node in S_n and sorted in the ascending order of the destination nodes.

Output : Each node in S_n receiving the appropriate message unit from the source node within $2n-3$ parallel communication steps.

1. for $i = n-1$ to 1 do

1.1 Let v be a node with a message block. Let us say that it belongs to S_i^j and is an (i,ℓ)-centre. Each such node v partitions its message block into $i+1$ sub-blocks, each sub-block consisting of $i!$ consecutive message units, and transmits the k-th sub-block, $0 \leq k \leq i$, $k \neq j$, $k \neq \ell$, to the (i,k)-centre in its domain in S_i.
*/*i.e. along dimensions 2 to i */*

1.2 if $i > 1$ then
parallel do

1.2.1 Each node receiving a message block along dimension k, $k \leq i$, transmits it to the adjacent node along dimension $i+1$ edge

1.2.2 Rest of the nodes, which are (i,ℓ)-centres, having two message sub-blocks transmits the ℓ-th sub-block to the adjacent node along dimension $i+1$ edge.

In the algorithm in step-1 out of the $i+1$ sub-blocks a node transmits $i-1$ sub-blocks to the respective (i,k)-centres to be forwarded to the S_i^ks in step-2. The j-th sub-block is not transmitted as it contains message units for the nodes of S_i^j to which the transmitting node belongs. On the other hand the ℓ-th sub-block is not transmitted in step-1 as the node itself is the (i,ℓ)-centre. It is transmitted in step-2.2.

THEOREM 7.5 *The communication time complexity of the scattering algorithm Star_Scatter_DDT is bounded above by* $(2n-3)t_h + 3(n-1)!Mt_w$.

PROOF. In the first iteration of the algorithm the message sub-blocks consist of $(n-1)!$ message units. In the second iteration the size of each sub-block is $(n-2)!$. In general, in the i-th iteration the size of a sub-block is $(n-i)!$. The algorithm consists of $n-1$ iterations. In the first $n-2$ iterations there are 2 parallel communication steps each and in the last iteration there is a single parallel communication step. Therefore, the total transfer time for the algorithm will be $Mt_w(2\sum_{n-1}^{2} i! + 1) \leq 3(n-1)!Mt_w$.

The length of the longest path in the algorithm is $2n-3$. Assuming that a sub-block can be accommodated in a single packet, the total hop time for the entire operation will be $(2n-3)t_h$.

Thus, the over all time complexity of the algorithm is bounded above by $(2n-3)t_h + 3(n-1)!Mt_w$. □

7.1.4 All-to-all broadcast

In the all-to-all broadcasting or gossiping operation each node in the network simultaneously performs a broadcast operation. Since there are $n!$ nodes in an S_n the gossiping operation will involve movement of $n!$ message units. Each processor will receive $n!-1$ message units, one from every other processor in the network.

In a gossiping operation each node in an S_n will receive $n!-1$ message units each of size M words. If the receiving of these message units by a node are divided equally over the $n-1$ edges connected to it, the minimum transfer time will be $\frac{n!-1}{n-1}Mt_w$. The message unit from the farthest node will have to traverse a path of $\lfloor\frac{3}{2}(n-1)\rfloor$ hops before reaching the receiving node. This will involve a minimum hop time of $\lfloor\frac{3}{2}(n-1)\rfloor t_h$. The lower bound on the time complexity of a gossiping operation on S_n is $\frac{n!-1}{n-1}Mt_w + \lfloor\frac{3}{2}(n-1)\rfloor t_h$.

The gossiping operation can be performed using the TDD paradigm. Initially each node in the network has its own data element. If each of them transmits its data element to its sibling in the S_2, gossiping will be complete for each of the S_2s in the network. Each S_2 then can transmit (that is, its nodes) its 2! data elements to its sibling S_2s using a TDD step. This will complete gossiping for each of the S_3s in the network. Each S_3 then can transmit its 3! data elements to its sibling S_3s using a TDD step so that gossiping for each of the S_4s is completed. If the process is repeated for $n-1$ times the the gossiping will be complete for the entire S_n. The corresponding algorithm Star_Gossip_TDD is presented below.

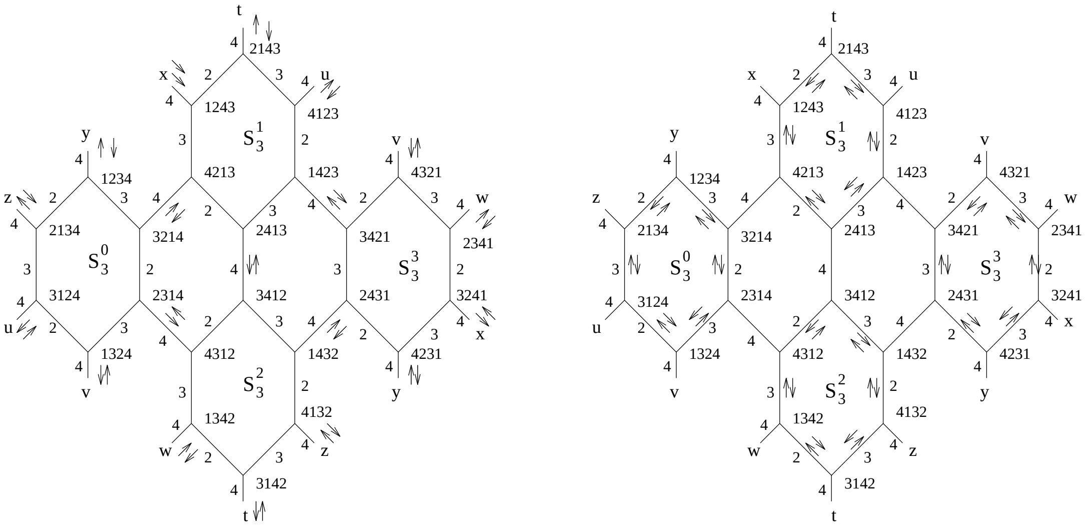
ALGORITHM 7.5 **Star_Gossip_TDD**

Input : Each node in S_n having one message unit.
Output: Each node in S_n having $n!$ message units, one from each node in the S_n.
1. for i:=1 to $n-1$ do
1.1 Each node transmits a copy of the data contained in it along dimension $i+1$ edge to the adjacent (i,k)-centre in sibling S_i.
1.2 if $i > 1$ then
1.2.1 Each node receives the packet arriving along dimension $i+1$ edge and transmits in parallel a copy of it to each of the nodes in its domain in S_i. / i.e. transmit to the nodes along dimensions 2 to i */*
1.2.2 Each node receives in parallel $i-1$ packets, one from each of its neighbours in its domain in S_i corresponding to step-1.2.1. It combines these $i-1$ packets and two others which are already present in it (one received in the step-1.1 and one of its own) into a single packet. The packet from j-th sibling substar S_i^j is placed in the j-th slot of the new packet.

The algorithm performs the operation hierarchically as in the case of the one-to-all-broadcast algorithm. Gossiping at the S_i substar level is completed before completing the gossiping at the S_{i+1} level in the next iteration. Once the gossiping at the substar level S_i are completed an S_i distributes its messages to all the nodes in the sibling S_is following the same pair of communication steps as in the case of one-to-all-broadcast. Message of a node in this iteration consists of the collection of the $i!$ message units, one from each of the nodes of the S_i. The data transfer process in iteration-3 that completes gossiping in an S_4 is illustrated in Figure 7.5.

THEOREM 7.6 *The gossiping algorithm Star_Gossip_TDD is free of channel contention.*

PROOF. In the step-1 of iteration $i, 1 \leq i \leq n-1$, of Star_Gossip_TDD each node has a single packet to transmit. This packet is transmitted only along the dimension $i+1$ edge. Therefore, in step-1 each node receives only one packet and that along the dimension $i+1$ edge. This



(a) Step-1 (b) Step-2

Figure 7.5: Data transfer in iteration-3 of gossiping in an S_4.

---

received packet is the only one to be transmitted by a node in step-2. Hence, in neither of the two steps there is channel contention. Therefore the the algorithm is contention-free. □

THEOREM 7.7 *The message units collected at the nodes through the algorithm Star_Gossip_TDD are arranged according to the ranks of their source nodes.*

PROOF. We shall give an inductive proof.
Case S_1: As S_1 consists of only one node, the case is trivial.
Case S_i: After the (i-1)-th iteration of the algorithm each node has the complete set of message units from the nodes in its S_i. Let us assume that they are ordered according to the ranks of their source nodes.
Case S_{i+1}: In the step 1.2.2 of the algorithm in the i-th iteration the message blocks that arrive from sibling S_is are ordered according to the *ranks* of the source substars. According to the *hierarchical ranking scheme* the ranks of the nodes in S_i^j are higher than those in sibling S_i^k if their ranks j and k are related as $j > k$ (Property 3.3). Therefore, after the i-th iteration the message units collected at a node from all the nodes of its S_{i+1} will be ordered according to the ranks of the source nodes. Thus follows the theorem. □

THEOREM 7.8 *The time complexity of the algorithm Star_Gossip_TDD is bounded above by*

$(2n-3)t_h + 3(n-1)!Mt_w$.

PROOF. The total hop time as in the earlier algorithms is $(2n-3)t_h$. In the i-th iteration the message from a node consists of the collection of $i!$ message units, one from each of the $i!$ nodes of its S_i. Each iteration, except the first, consists of two steps. The first iteration consists of one step. There are totally $n-1$ iterations. Thus, the total message transfer time is $Mt_w + 2Mt_w \sum_2^{n-1} i! < 3(n-1)!Mt_w$. Therefore, the total communication cost of the gossip algorithm is bounded above by $(2n-3)t_h + 3(n-1)!Mt_w$. $\square$

7.1.5 Single node accumulation

The single-node-accumulation operation is the reverse process of the one-to-all scattering operation. Through this operation data elements of equal size from all the nodes in the network are accumulated at a particular node. The lower bound on the time complexity of an accumulation operation on an S_n is also $\lfloor \frac{3}{2}(n-1) \rfloor t_h + \frac{n!-1}{n-1} Mt_w$.

It is possible to perform an accumulation operation by tracing the same set of paths as a data scattering operation in the reverse direction where the source node takes the role of the collector node. Since the scattering operation is performed using the domain-distribute-and-transmit paradigm, the accumulation operation has to be performed using a reverse DDT paradigm what we may call *transmit-and-domain-collect*. However, performing the accumulation operation through *transmit-and-domain-collect* approach is not straight forward due to the lack of knowledge about the identity of the some of the intermediate nodes (intermediate substar level accumulator nodes). This can be overcome by *precomputing* the entire path to the destination node from each source node. A source node can determine such a path by computing the reverse path, that is, the path followed in a domain distribute and transmit operation from the destination to the source node.

A more elegant way to perform the accumulation operation is by using the HOPC paradigm. The use of the HOPC paradigm leads to a simple algorithm for the single node accumulation algorithm though at a some what higher cost. The algorithm Star_HOPC_Accumulate is given below.

ALGORITHM 7.6 **Star_HOPC_Accumulate(A)**

Input : One data element in each node of S_n and the address, A of the node at which the data elements have to be accumulated.
Output : Data elements from each node in S_n accumulated at the node with address A.

1. for i:= 1 to n-1 do

1.1 parallel do for each node of the same rank in an S_i as that of the destination node

1.1.1 if the node is not of the same rank in its S_{i+1} as that of the destination node

1.1.1.1 then

it transmits the data to its peer node in the sibling S_i of the same rank as that of the destination node.

1.1.1.2 else

it receives the data arriving from its peer nodes in the rest of the sibling S_is.

If performed synchronously, the data elements accumulated at the collector node will be in the ascending order of the source nodes. In the algorithm the data in the substars S_{i-1}s are collected into a single node in each of them before the data from sibling substars S_{i-1}s are collected into a single node in each parent S_i. While collecting the data blocks from the sibling substars the blocks are placed in slots according to the ascending order of the *ranks* of the substars, S_{i-1}s in the S_i. As a result, the data gathered in the collector node will be ordered according to the ranks of the source nodes in the S_i. This follows from Property 3.3.

THEOREM 7.9 *The time complexity of the algorithm Star_HOPC_Accumulate is bounded above by $4(n-2)t_h + 6(n-1)!Mt_w$.*

PROOF. The first iteration for accumulation in S_2s require 1 step and the second iteration for accumulation in S_3s require 3 step. The remaining $n-3$ iterations require 4 steps each. Therefore the maximum length of a path is $4(n-2)$. The total hop time in this algorithm is therefore $4(n-2)t_h$. In the i-th iteration the message from a node consists of the collection of $i!$ message units, one from each of the $i!$ nodes of its S_i. Therefore, the total message transfer time is $Mt_w + 3M(2!) + 4Mt_w \sum_3^{n-1} i! < 6(n-1)!Mt_w$. The total communication cost of the gossip algorithm is bounded above by $(2n-3)t_h + 6(n-1)!Mt_w$. □

7.1.6 Optimality of the algorithms

Table 7.1 gives a comparison of time complexities of the algorithms that we have presented in this section with that of optimal algorithms for the corresponding operations. Considering the fact that t_s and t_w are constants for a system and $(n-1)! < \frac{n!-1}{n-1}$, it is easy to see that the algorithms are asymptotically optimal. Some optimal algorithms for the global communication operations have been developed by Fragopolou and Akl [FA93b, FA93a] and Misić and

Operation	Time complexity	
	Lower bound	Presented algorithm
One-to-all broadcast (single-path)	$\lfloor\frac{3}{2}(n-1)\rfloor(t_h + Mt_w)$	$(2n-3)(t_h + Mt_w)$
One-to-all broadcast (multi-path)	$\lfloor\frac{3}{2}(n-1)\rfloor(t_h + \frac{M}{n-1}t_w)$	$(2n-3)(t_h + \frac{M}{n-1}t_w)$
One-to-all scatter	$\lfloor\frac{3}{2}(n-1)\rfloor t_h + \frac{n!-1}{n-1}Mt_w$	$< (2n-3)t_h + 3(n-1)!Mt_w$
All-to-all broadcast	$\lfloor\frac{3}{2}(n-1)\rfloor t_h + \frac{n!-1}{n-1}Mt_w$	$< (2n-3)t_h + 3(n-1)!Mt_w$
Single node accumulation	$\lfloor\frac{3}{2}(n-1)\rfloor t_h + \frac{n!-1}{n-1}Mt_w$	$< (2n-3)t_h + 6(n-1)!Mt_w$

Table 7.1: Comparison of communication complexity of the algorithms with the lower bounds.

Jovanović [MJ94a]. The algorithms presented here are simple and easy to implement. Further, the algorithms are readily adaptable on star clusters introduced in Chapter 5.

7.2 Ascend/descend algorithms

In this section, we present order preserving ascend/descend algorithms for Fast Fourier Transformation (FFT) and matrix multiplication based on the HOPC paradigm discussed in Chapter 4.

7.2.1 Fast Fourier Transform

We present two algorithms for computing $n!$-point FFT on an S_n, namely, *monotonic factored FFT* and *prime factored FFT*.

Monotonic factored FFT

In this algorithm, the input size $N = n!$ is split into monotonically increasing factors (i.e., $N = 2.3.....(n-1).n$) to implement the $n!$-point FFT in $n-1$ stages with the i-th stage consisting of (i+1)-point DFTs. For example, a 24-point (4!-point) FFT computation in this scheme will be computed in 3 stages. The first stage will consist of 12 2-point DFTs, the second stage of 8 3-point DFTs and the third stage of 6 4-point DFTs. Figure 7.6 shows the flow graph for the 24-point FFT for decimation-in-time in-place computation.

The nodes of a row in the FFT flow graph (corresponding to one signal sample) are allocated to the same processor in the S_n. For example in Figure 7.6 the nodes in the row corresponding to signal sample $x(23)$ is allocated to the processor ranked 23. The address of the processor of the S_n allocated to the ρ-th signal sample can be computed using equation 7.1 given below. In the equation a_i is the i-th digit of the processor address $A = a_{n-1}a_{n-2}....a_1$.

$$a_i = (\rho \ div \ \frac{n!}{(i+1)!}) \ mod \ (i+1) \tag{7.1}$$

where, the operator *div* stands for integer division and the operator *mod* stands for the modulo operator. Figure 7.6 illustrates the processor assignment for 24-point FFT computation on S_4.

An examination of the FFT flow graph in Figure 7.6 shows that all internode communications in stage-1 are confined within S_2. In stage-2 all such communications are confined within S_3s and further only the peer nodes in the sibling S_2s within a S_3 communicate. Similarly, in stage-3 all such communications are confined within S_4s. Also, only the peer nodes in the sibling S_3s within a S_4 communicate. In general, for an n!-point FFT, in the i-th stage $(1 \leq i < n)$ all the internode communications are confined within S_{i+1}s. All the peer nodes in the sibling S_is within the parent S_{i+1} would communicate.

In stage-1 as the communications are confined within S_2s, the routing is trivial. The node pairs of S_2s interchange information along the dimension-2 edges to compute the 2-point DFTs.

In the i-th stage $(2 \leq i \leq n-1)$, all the $(i+1)$ sibling S_is within an S_{i+1} send information to each other (between peer nodes). This is basically an all-to-all broadcast between peer nodes in S_is. It can be implemented using peer 0-centre routed HOPC without edge-congestion (Theorem 4.3). The communication cost in each stage is $O(1)$. It is clear that as there are $n-1$ stages the overall communication steps required is $O(n)$. In the i-th stage $i+1$ processors together compute an $(i+1)$-point DFT. The computational load per processor in the i-th stage is $O(i)$. The overall time complexity, obtained by adding them over all the $n-1$ stages, is $O(n^2)$.

An earlier algorithm for $n!$-point FFT on S_n proposed by Fragopolou and Akl [FA94] uses disjoint cycles in S_n. The computation time complexity of that algorithm is $O(n^2)$. An existing systolic DFT algorithm on linear array [ZY84] has been used giving a communication cost of $O(n^2)$ time steps.

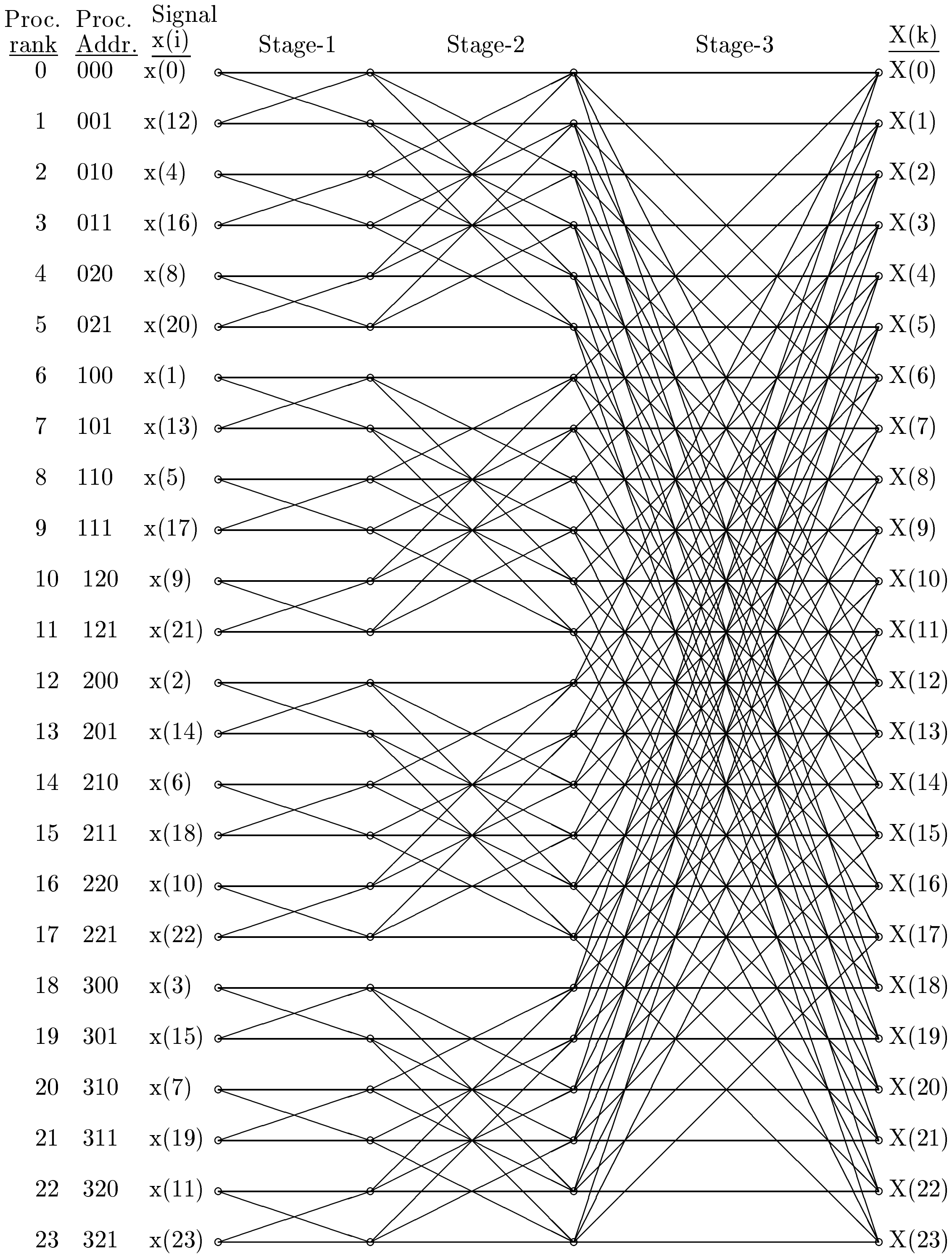


Figure 7.6: 24-point FFT flow diagram with processor assignment on S_4.

Contraction for Computation of (k.n!)-point FFT on a S_n

Suppose it is necessary to perform a (k.n!)-point FFT computation on an S_n. This can be carried out by performing k n!-point FFTs on the S_n and then adding an additional stage of k-point FFTs to complete the computation. If k is not a prime number, it can be suitably factorized to reduce the cost of the final k-point FFT computation. For the processor assignment, in this case, eqation 7.1 is modified to:

$$a_i = (\rho \; div \; \frac{kn!}{(i+1)!}) \; mod \; (i+1) \tag{7.2}$$

This processor assignment avoids additional communication cost in the final stage of k-point FFTs. Table 7.2 gives the resulting processor assignment for 18-point (3.3!-point) FFT computation on S_3.

Signal sample	Processor assigned	
no.	Address	rank
0	00	0
1	00	0
2	00	0
3	10	2
4	10	2
5	10	2
6	20	4
7	20	4
8	20	4
9	01	1
10	01	1
11	01	1
12	11	3
13	11	3
14	11	3
15	21	5
16	21	5
17	21	5

Table 7.2: Processor assignment for 3.3!-point FFT computation on S_3.

It can be observed that the computation of FFT with an arbitrary input size can also be performed on a star. Suppose we have an input size $m \neq k.n!$. To compute the FFT one should find a suitable $k.n! > m$. Then, append the input with 0's to makeup the size to $k.n!$. After this the procedure for computation of $k.n!$-point FFT can be used.

Prime factored FFT

The computation cost associated with n!-point FFT on the star is $O(n^2)$. The computation cost for the same size is $O(nlogn)$ on a hypercube. It is possible to reduce the cost of the FFT computation on S_n and make it quite comparable to that on the hypercube. This can be done by using the prime factors of n! and each stage of the computation performing DFTs of length of one of the factors. For example, 4! can be factorized as $2 \times 3 \times 2 \times 2$, leading to 4 stages of computations for 4!-point FFT. The first, third and the fourth stages will be of 2-point DFTs and the second stage will be of 3-point DFTs. Figure 7.7 shows the FFT flow graph for this computation with processor assignment on the S_4.

In general, if f_i denotes the i-th prime factor then the computation cost is proportional to $\sum_i (f_i - 1)$. Table 7.3 gives a comparison of the computation costs for n!-point FFT with monotonic factored and prime factored implementations on S_n and that for radix-2 implementation on equivalent sized hypercube for different values of n.

In this case the computational cost for the star is marginally higher than that for the hypercube implementation. However, considering the other benefits of the star such as reduced communication hardware cost, due to the lower degree of the nodes, this implementation is quite meaningful. For example, in implementing 10!-point FFT the ratio of computation costs on the star and the hypercube is $\frac{30}{22} \approx 1.36$ and the ratio of node degrees of the corresponding hypercube and star is $\frac{22}{9} \approx 2.44$. This means that part of the communication hardware cost can be reduced by a factor of 2.44 if we permit an increase in computational cost by 36 percent.

The prime factored implementation of n!-point FFT on the S_n, however, needs a minor change in the internode communication strategy. For example, while performing 2-point DFTs the direct peer to peer HOPC has to be performed. For larger DFTs the collection and distribution scheme through the peer 0-centre can be used.

7.2.2 Matrix multiplication

In this section we present a simple order preserving ascend/descend matrix multiplication algorithm. The computation is broken-up recursively into smaller matrix multiplications until they get reduced to a set of $(2\times2)\times(2\times3)$ matrix multiplications. These $(2\times2)\times(2\times3)$ matrix

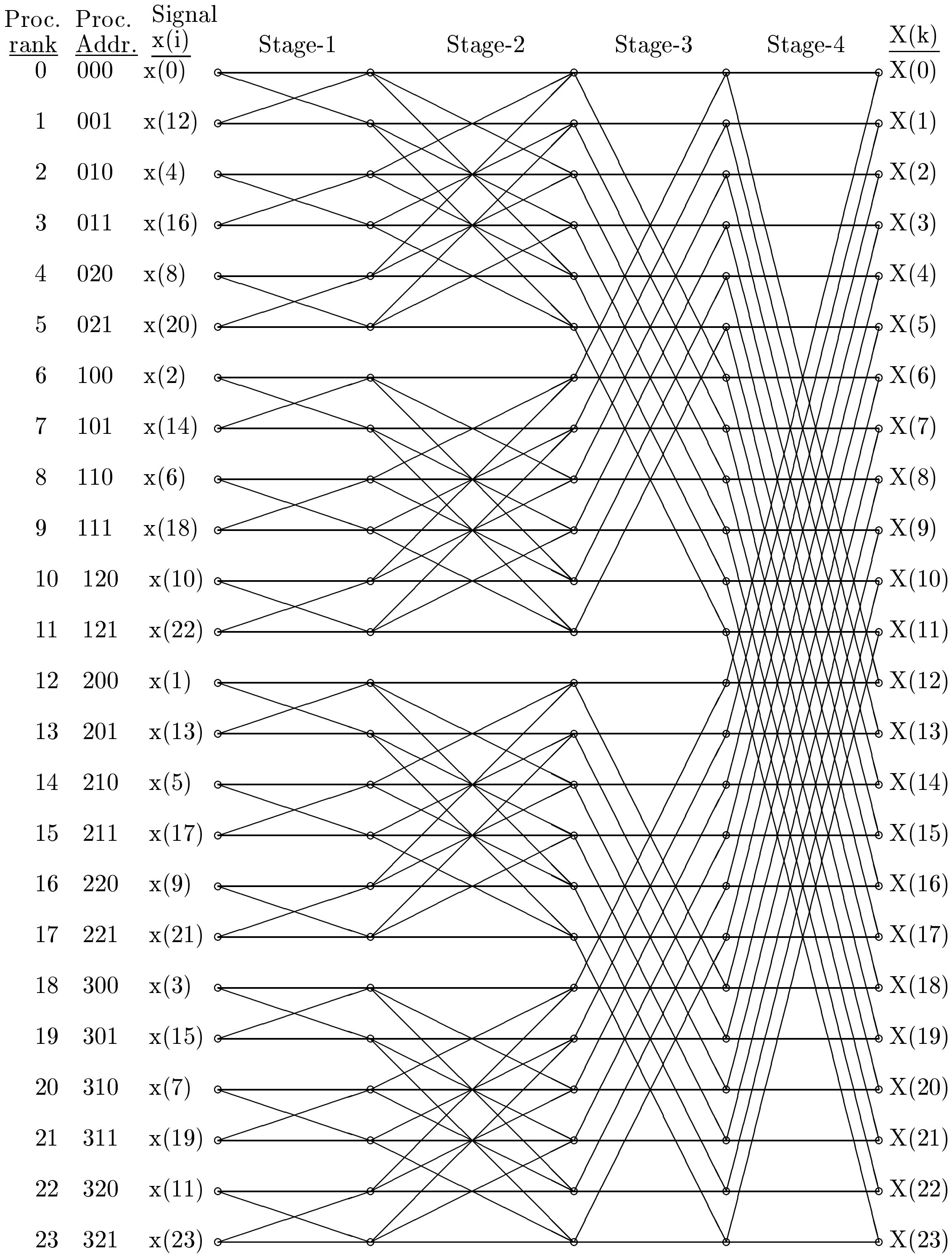


Figure 7.7: Prime factored 24-point FFT flow diagram with processor assignment on S_4.

n	Network size	Star degree	Hypercube degree	FFT computation time		
				Hypercube	Star with prime factorization	Star with monotonic factorization
	n!	n-1	$\lceil logn! \rceil$	$\lceil logn! \rceil$	$\sum_i (f_i - 1)$	$(n^2 - n)/2$
4	24	3	5	5	5	8
5	120	4	7	7	9	12
6	720	5	10	10	12	17
7	5040	6	13	13	18	21
8	40320	7	16	16	21	28
9	362880	8	19	19	25	36
10	3628800	9	22	22	30	45

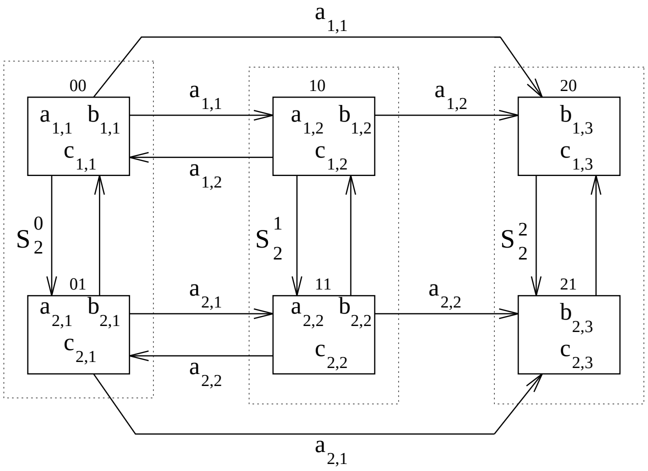
Table 7.3: Comparison of $n!$-point FFT on star graph and hypercube.

multiplications are performed in parallel in the S_3s of the star network. For the $(2 \times 2) \times (2 \times 3)$ matrix multiplication the processors of a S_3 are arranged logically in a (2×3) array form in which each column constitutes an S_2 as shown in Figure 7.8.

The communications required in the computation are shown with directed edges with labels of the data elements to be transferred. A communication within a column can be carried out in a single step over the dimension 2 edge of the corresponding S_2. The communication in the rows are between the peer processors in the S_2. Hence, these can be performed using either confined routing or the third substar routed HOPC. After the communications are performed the computation becomes straight forward (Figure 7.8).

For a $(4 \times 4) \times (4 \times 6)$ matrix multiplication an S_4 can be used. Let A be a (4×4) matrix, B be a (4×6) matrix and the product be C, a (4×6) matrix. For this the S_3 of the S_4 are logically arranged in a (2×2) array as shown in Figure 7.9. In the figure the data to be transferred between the substars are indicated with labeled directed edges. Here, $A_{p,q}$ ($B_{p,q}$) represent the entire set of A elements (B elements) in the S_3 corresponding to the (p,q)-th element of the logical array of the substars. A close look at the arrangement will reveal that if the S_3s in each row of this array interchange the A elements between their peer nodes. Similarly the S_3s in each column interchange their B elements in an order preserving manner. The problem then gets reduced to two $(2 \times 2) \times (2 \times 3)$ matrix multiplications and one matrix addition in each S_3. The matrix addition is essentially the summing of the resulting product elements at each node.

In general, by suitably arranging the substars, multiplication of matrices of various sizes



$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \end{bmatrix} = \begin{bmatrix} a_{1,1}b_{1,1}+a_{1,2}b_{2,1} & a_{1,1}b_{1,2}+a_{1,2}b_{2,2} & a_{1,1}b_{1,3}+a_{1,2}b_{2,3} \\ a_{2,1}b_{1,1}+a_{2,2}b_{2,1} & a_{2,1}b_{1,2}+a_{2,2}b_{2,2} & a_{2,1}b_{1,3}+a_{2,2}b_{2,3} \end{bmatrix}$$

Figure 7.8: Data distribution and communication structure for $(2 \times 2) \times (2 \times 3)$ matrix multiplication on S_3.

---

can be performed on the star. If the substars, S_is within the S_{i+1}s, $3 \leq i < n$, are logically arranged in arrays of $x_i \times y_i$ form the matrix multiplication that can be performed on the S_n is $(p \times q) \times (q \times r)$, where $p = 2\prod_3^{n-1} x_i$, $q = 2\prod_3^{n-1} min(x_i, y_i)$ and $r = 3\prod_3^{n-1} y_i$. In case the matrix size is larger than the size of the star one can assign blocks of matrix elements obtained through suitable block-checkerboard partitioning [KGGK93] of the matrices. Multiplication of matrices of arbitrary sizes can be performed by suitably partitioning the matrices. Some examples are shown in Figure 7.10. Figures show the partitioning of the matrices and also the associated communication structure.

For a matrix multiplication performed on an S_n, the communication consists of $n-2$ phases each with $O(1)$ communication steps. This gives an $O(n)$ communication complexity for matrix multiplication on an S_n. The number of scalar multiplications per processor is equal to the number of columns in the first matrix or the number of rows in the second matrix. The number of additions is one less than the number of multiplications. Thus the parallel computation cost is $O(m)$ for a $(\ell \times m) \times (m \times p)$ matrix multiplication.

It is evident that the matrix multiplication scheme given here has a high memory requirement. Our motivation in presenting this scheme was to show the use of the hierarchical ranking scheme and the HOPC paradigm in implementing order preserving ascend/descend algorithms. It may be noted that the constant time order preserving communications can be used to im-

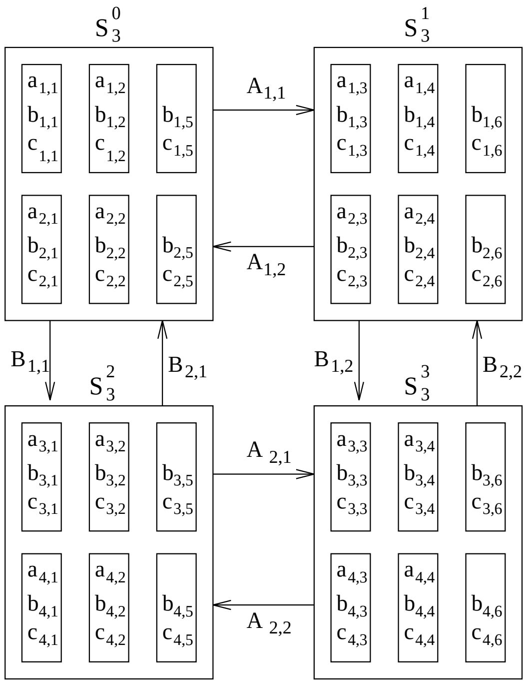


Figure 7.9: Data distribution and communication structure for $(4 \times 4) \times (4 \times 6)$ matrix multiplication on S_4.

---

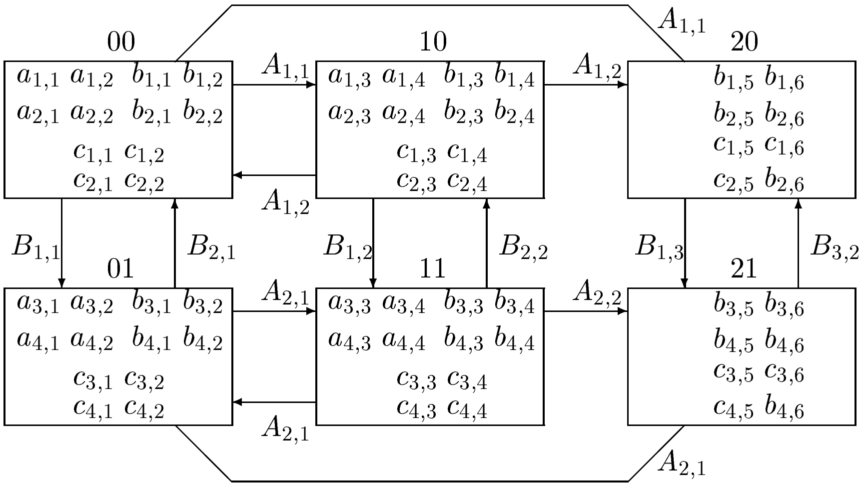
plement Cannon's and Fox's memory efficient matrix multiplication algorithms [KGGK93] for a mesh on the star graph.

7.3 Pipelined algorithms

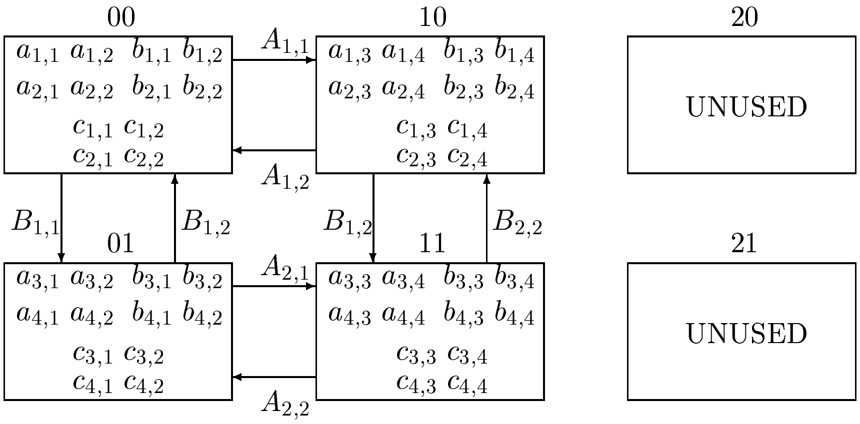
The pipeline structured ranking scheme enables peer nodes in sibling substars S_n^i and S_n^{i+1} to communicate using the ADC paradigm. This creates a pipeline of $n+1$ $S_n^i, i = 0, 1, ..., n$ where the peer nodes of the i-th stage and (i+1)-th stage are connected by a path of length at most three. See Figure 7.11. This view of the star can be used to implement pipelined algorithms for it. As examples we present pipelined algorithms for $n!$-point 1-D FFT and $n! \times n!$-point 2-D FFT on S_{n+1}.

7.3.1 Pipelined implementation of n!-point 1-D FFT on S_{n+1}

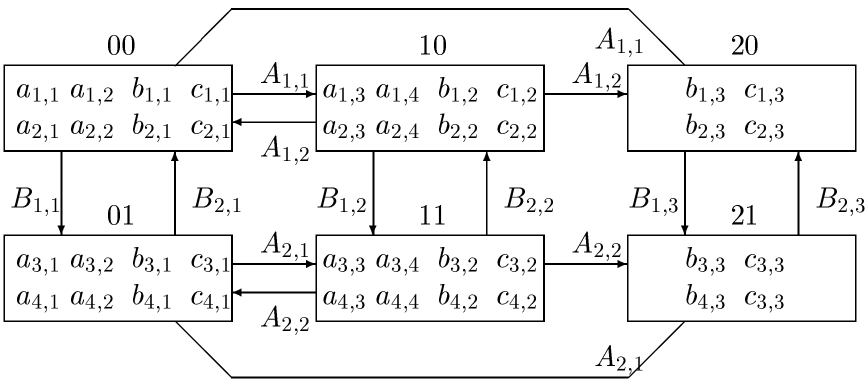
The pipelined implementation of the n!-point FFT on an S_{n+1} is quite simple. Following the algorithm in Section 7.2 we can compute an n!-point FFT on an S_n. The computation consists



(a) for $(4 \times 4) \times (4 \times 6)$ matrix multiplication.



(b) for $(4 \times 4) \times (4 \times 4)$ matrix multiplication.



(c) for $(4 \times 4) \times (4 \times 3)$ matrix multiplication.

Figure 7.10: Data distribution and communication structure for multiplication of matrices of various sizes on S_3.

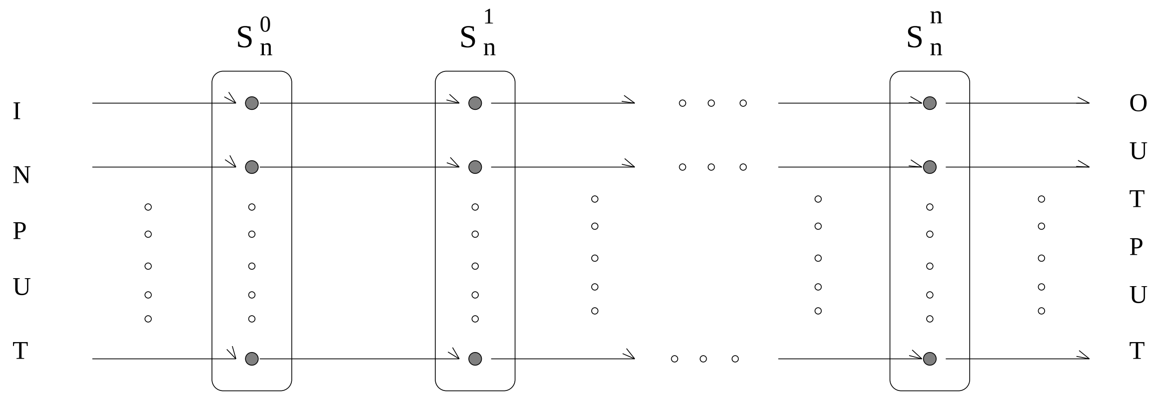


Figure 7.11: Pipelined organization of S_ns in an S_{n+1}.

of $n-1$ stages. Instead of allocating all the $n-1$ stages of the computation to a single S_n, we can distribute the computation by allocating the i-th stage to S_n^i of the S_{n+1}. After performing the i-th stage of computation S_n^i will transfer the result to S_n^{i+1} in an order preserving manner. Finally, at the end of the pipe S_n^{n-1} will produce the final result. The two stages S_n^0 and S_n^n can be used to perform input and output operations.

From the discussion in Section 7.2.1 we know that the time complexity of the i-th stage of computation is $O(i)$. Summing up the costs of the $n-1$ stages, we have the overall time complexity as $O(n^2)$.

7.3.2 Pipelined implementation of $n! \times n!$-point 2-D FFT on S_{n+1}

The pipelined implementation of $n! \times n!$-point 2-D FFT presented here is based on the pipelined implementation of n!-point FFT on S_{n+1} discussed above. In addition, it uses the HOPC based data accumulation algorithm, Star_Accumulate_HOPC, discuss in Section 7.1.5, for the matrix transposition operation of the $n! \times n!$-point data array. The S_{n+1} is used as a bidirectional pipeline of the S_ns contained in it. S_n^0 forms the input as well as the output stage of the bidirectional pipeline. S_n^n performs the matrix transposition operation. At first, n! elements of a column of the input data array are allocated to each processor of S_n^0. The individual addresses of the processors of the S_n which is assigned to the ρ-th column can be computed using equation 7.1.

Each processor of S_n^0 passes on the first element of the column allocated to it to its peer node in S_n^1 so that S_n^1 will have all the first row data elements. S_n^1 then computes the 1st stage of the n!-point FFT of the first row elements and passes the result to S_n^2 which computes the second stage of n!-point FFT. At the same time the S_n^0 hands over the second row data elements to S_n^1. S_n^1 repeats the 1st stage of computation on the second row and hands over the results to S_n^2 while S_n^2 hands over its results to S_n^3. Similarly, S_n^{n-1} receives the partially computed data from S_n^{n-2} for one row at a time and performs the (n-1)-th and final stage of the n!-point FFT computation of the row and then hands over the result to S_n^n. On receiving the FFT of a row S_n^n accumulates the data using the algorithm Star_Accumulate_HOPC at the processor the address of which can be obtained by using equation 7.1. S_n^n performs the data accumulation operation for each row of the $n! \times n!$ array resulting in its transposition. At the end of it each processor in S_n^n has a complete row (instead of a column) of the array. After completion of the row FFT computation phase and the transposition operation the column FFT computation phase begins. This is performed in a similar manner. However, in this phase data flows in the reverse direction where S_n^n acts as the input stage and S_n^0 acts as the output stage. S_n^{n-i} computes the i-th, $1 \leq i \leq n-1$, stage of the n!-point FFT. Finally, S_n^0 contains the result with the k-th ranked processor holding the k-th row FFT points.

The time taken by the $(n-1)$-th stage of the $n!$-point 1D-FFT computation is $O(n)$. The data accumulation operation performed by the n-th stage of the pipeline also requires $O(n)$ time. Since the pipeline consists of n-stages, it therefore requires a total of $O(n^2)$ time to fill the pipeline. After the pipeline is filled, one 1-D FFT computation is completed in every $O(n)$ time steps. It is necessary to compute $n!$ 1-D FFTs in the row FFT computation phase and another $n!$ 1-D FFTs in the column FFT computation phase. As a result the overall time complexity of the algorithm to compute $n! \times n!$-point 2-D FFT on an S_{n+1} is $O(n.n!)$.

7.4 Parallel prefix algorithm

In this algorithm each node holds two variables, t and s. The variable t maintains the total sum for a group of nodes and s maintains a partial prefix for the node. A group consists of the nodes of a sequence of consecutive sibling substars. The algorithm requires $n-1$ iterations for a data size of $n!$ on an S_n, one iteration for each substar level 1 to $n-1$. In the i-th iteration of the algorithm sibling S_is form groups. The i-th iteration consists of $\lceil log(i+1) \rceil$ steps. In each of these steps adjacent pairs of groups of sibling S_is combine to form new groups and reduce the number of groups to about half and at the end of the iteration all the $i+1$ S_is in an S_{i+1} form a single group. In the ρ-th step there are 2^ρ S_is in each group except for the last, which may have less number of S_is. The combining process is performed as follows. Let the two groups to be combined, say $C1$ and $C2$, consist of the two sets of S_is $\{S_i^{k1}, \ldots, S_i^{k1+\Delta 1-1}\}$ and $\{S_i^{k2}, \ldots, S_i^{k2+\Delta 2-1}\}$ respectively such that $k2 = k1 + \Delta 1$. The first S_i in a group is called a leader. For example, S_i^{k1} and S_i^{k2} are the leaders in the above groups. For combining, each (i,k)-centre, $k1 \leq k < k1 + \Delta 1$, in S^{k1_i} transmits the t value to the group $C2$ through the TDD paradigm. Similarly, each (i,k)-centre, $k2 \leq k < k2 + \Delta 2$, in S^{k2_i} transmits the t value to the group $C1$ through the TDD paradigm. Each node in the two groups thus receives the total sum t of the other group. It then updates its t value as $t \leftarrow t \oplus received\ t$. The nodes in the group $C2$ also update its s value as $s \leftarrow s \oplus received\ t$. The result is the formation of a new group with the set of S_is $\{S_i^{k1}, \ldots, S_i^{k1+\Delta 1+\Delta 2}\}$. Figure 7.12 illustrates the regrouping process and the associated communication for $i = 6$. After completion of the i-th iteration a group consists of a single (i+1)-substar. The algorithm is given below.

ALGORITHM 7.7 **Star_Prefix(S_n)**

Input: $n!$ data elements one in each node of S_n.
Output: Prefix sum computed in the nodes of S_n. The order of the nodes is assumed to be as ranked by the hierarchical ranking scheme.

1. for i := 1 to n-1 do
    - 1.1 for each S_{i+1} parallel do
        - 1.1.1 for step := 1 to $\lceil log(i+1) \rceil$ do
            - 1.1.1.1 parallel do
                - 1.1.1.1.1 for l := 1 to $\lfloor (i+1)/(2 * group\_size) \rfloor$ parallel do
                  Regroup(l,group_size,l+group_size,group_size,i)
                - 1.1.1.1.2 $l := (i+1) \bmod (2 * group\_size)$
                - 1.1.1.1.3 if $l > group\_size$ then
                  Regroup(l,group_size,l+group_size,l-group_size,i)

ALGORITHM 7.8 **Regroup($l1, sz1, l2, sz2, i$)**

Input: $l1, l2$ - the ranks of the leaders of the two groups of substars;
$sz1, sz2$ - the number of substars in the two groups;
i - the dimension the substars.
Output: To produces a single group of $sz1 + sz2$ S_is with l1 as the rank of the leader and the s and t values appropriately updated.

1. parallel do
    - 1.1 Each (i,k)-centre, $l2 \leq k \leq l2 + sz2 - 1$, node in S_i^{l1} transmits its t value to the sibling S_is of rank $l2$ to $l2 + sz2 - 1$ through a TDD operation.
    - 1.2 Each (i,k)-centre, $l1 \leq k \leq l1 + sz1 - 1$, node in S_i^{l2} transmits its t value to the sibling S_is of rank $l1$ to $l1 + sz1 - 1$ through a TDD operation.
2. parallel do
    - 2.1 Each node in the S_is of rank $l1$ to $l1 + sz1 - 1$ updates its t value
      as $t \leftarrow t \oplus$ received t.
    - 2.2 Each node in the S_is of rank $l2$ to $l2 + sz2 - 1$ updates its t and s values
      as $t \leftarrow t \oplus$ received t
      $s \leftarrow s \oplus$ received t

In the algorithm all the communications occur through the TDD communication paradigm. Moreover, at one time the communication occur only at one S_i level. Therefore, the algorithm

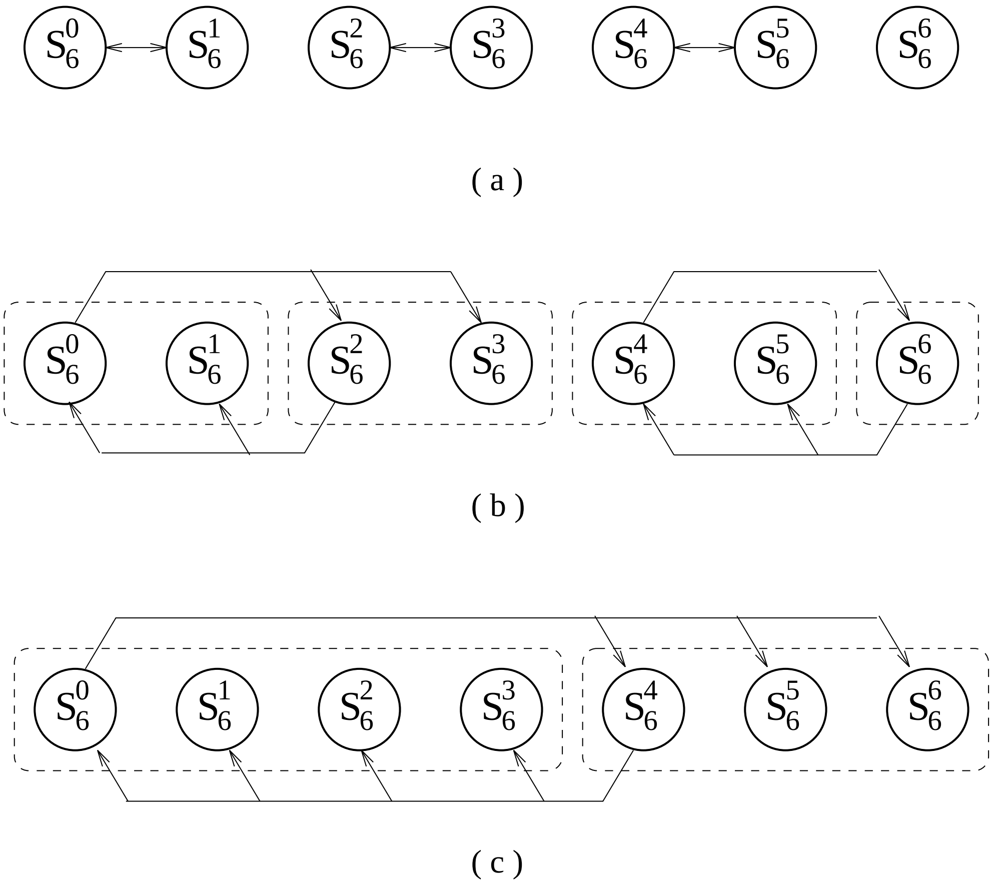


Figure 7.12: Regrouping in prefix computation; Iteration, $i = 6$. (a) Step-1: each S_6 is a leader, (b) step-2: S_6^0, S_6^2, S_6^4 and S_6^6 are leaders, (c) step-3: S_6^0 and S_6^4 are leaders.

is congestion free. The algorithm consists of $n-1$ iterations one for each of the substar levels. The i-th iteration consists of $\lceil log(i+1) \rceil$ steps. As a result the computation complexity of the algorithm is $O(nlogn)$. In each step of the computation the communication consists of two parallel steps. Therefore the communication consists of $2\sum_1^{n-1}\lceil log(i+1)\rceil$ congestion free parallel steps. Hence, the communication complexity is also $O(nlogn)$.

7.5 Algorithm implementation on star clusters

In this section we discuss how existing algorithms for the star graph can be implemented on *star clusters*. There are basically three requirements for algorithms developed for the star to be implementable on a star cluster. First, the routing schemes used should be such that it would be possible to avoid a path through a non-existent substar. Second, a computation with the given algorithm should be easily contractable. Third, the algorithm should not use the strongly hierarchical properties of the star. The first requirement is easy to satisfy as there is always a minimal path, between two nodes, that does not use a third substar. Most of the algorithms developed for the star so far are hierarchical in nature and belong to the divide and conquer category. The divide and conquer strategy seem to suit the star interconnection topology well. This class of algorithms are easily contractable and satisfies the second requirement. If an algorithm does not satisfy the third requirement, it should still be possible to implement the algorithm on a star cluster with minor modifications and some amount of degradation as the S_ns of the $SC_{n,k}$ are still strongly hierarchical. In the following we shall briefly illustrate how three types of algorithms developed for the star can be implemented on the star cluster. These are i) a set of global communication algorithms, ii) the ascend/descend class of algorithms, and ii) the pipelined algorithms in which a substar is considered a stage in the pipeline.

7.5.1 Global communication algorithms

We shall consider a broadcasting algorithm to illsutrate how the global communication algorithms of hierarchical nature can be implemented on a star cluster. The algorithm Stcl_Broadcast_TDD given below uses the *transmit and domain distribute* approach for broadcasting on a star cluster $SC_{n,k}$. The for loop in step-1 performs broadcasting in the S_n containing the source node. Steps-2 and step-3 spread the message to the rest of the S_ns to complete the broadcasting operation.

ALGORITHM 7.9 **Stcl_Broadcast_TDD**

Input : A source node in S_n^j of an $SC_{n,k}$ having a message.
Output : Each node in the $SC_{n.k}$ containing a copy of the message in the source node after 2n-1 parallel communication steps.

1. for i = 1 to $n-1$ do

1.1 Each node having a copy of the message transmits in parallel a copy of it to the adjacent (i,k)-centre in sibling S_is along dimension $i+1$ edge.

1.2 if $i > 1$ then

Each node receiving a copy of the message along the dimension $i+1$ edge transmits a copy of it in parallel to each of the nodes in its domain in S_i.
*/*i.e. transmit to all the adjacent nodes along dimensions 2 to i. */*

2. Each node in S_n^j that has the dimension $n+1$ edge transmits in parallel a copy of the message to the adjacent (n,j)-centre in sibling S_n along the dimension $n+1$ edge.

3. Each node receiving a copy of the message along the dimension $n+1$ edge transmits a copy of it in parallel to each of the nodes in its domain in S_n.

Since the absence of some of the S_ns do not affect TDD and DDT communication paradigms (Section 5.3), there is no degradation in the performance of algorithms based on these paradigms when implemented on a star cluster, $SC_{n,k}$.

7.5.2 Ascend/descend algorithms

We illustrate the adaptation of ascend/descend algorithms on star clusters with the example of an $n!$-point FFT algorithm. Figure 7.13 shows the computation graph for 24!-point FFT to be implemented on an $SC_{3,2}$. The Figure also shows the processor allocation. The entire set of computation nodes in a row corresponding to a single signal sample is allocated to a single processor node. The address of the processor node of the S_n allocated to the row of nodes corresponding to the ρ-th signal sample is computed using the equation 7.3 given below. In the equation a_i the i-th digit of the processor address $A = a_{n-1}a_{n-2}....a_1$.

$$a_i = (\rho \; div \; \frac{n!}{(i+1)!}) \; mod \; (i+1) \tag{7.3}$$

where, the operator *div* stands for integer division and operator *mod* stands for the modulo operator.

The other ascend/descend algorithms such as matrix multiplication, MSF, prefix computation etc. can also be easily contracted for implementation on star clusters. If we have an $SC_{n-1,k}$ instead of an S_n the $\frac{n}{k}$ rows will be allocated to the same processor and the address of the processor, within an S_{n-1} of the $SC_{n-1,k}$, allocated to the row corresponding to the ρ-th signal sample can be computed using the equation 7.4 below.

$$\begin{aligned} a_i &= (\rho \; mod \; k(n-1)!) \; div \; \frac{k(n-1)!}{(i+1)!}) \; mod \; (i+1), \quad for \;\; 1 \le i \le n-2 \quad and \\ &= \rho \; mod \; k, \qquad\qquad for \;\; i = n-1 \end{aligned} \tag{7.4}$$

The processor allocation is done such that the communication requirement is minimized while the parallelism is kept at a maximum for the given number of processors.

In the ascend/descend algorithms, the communications at the top most substar level has to be restricted to confined routing (Section 5.3). As a result, there may be edge-congestion in communication in the corresponding stage of computation. This may lead to minor degradation in performance of the computations.

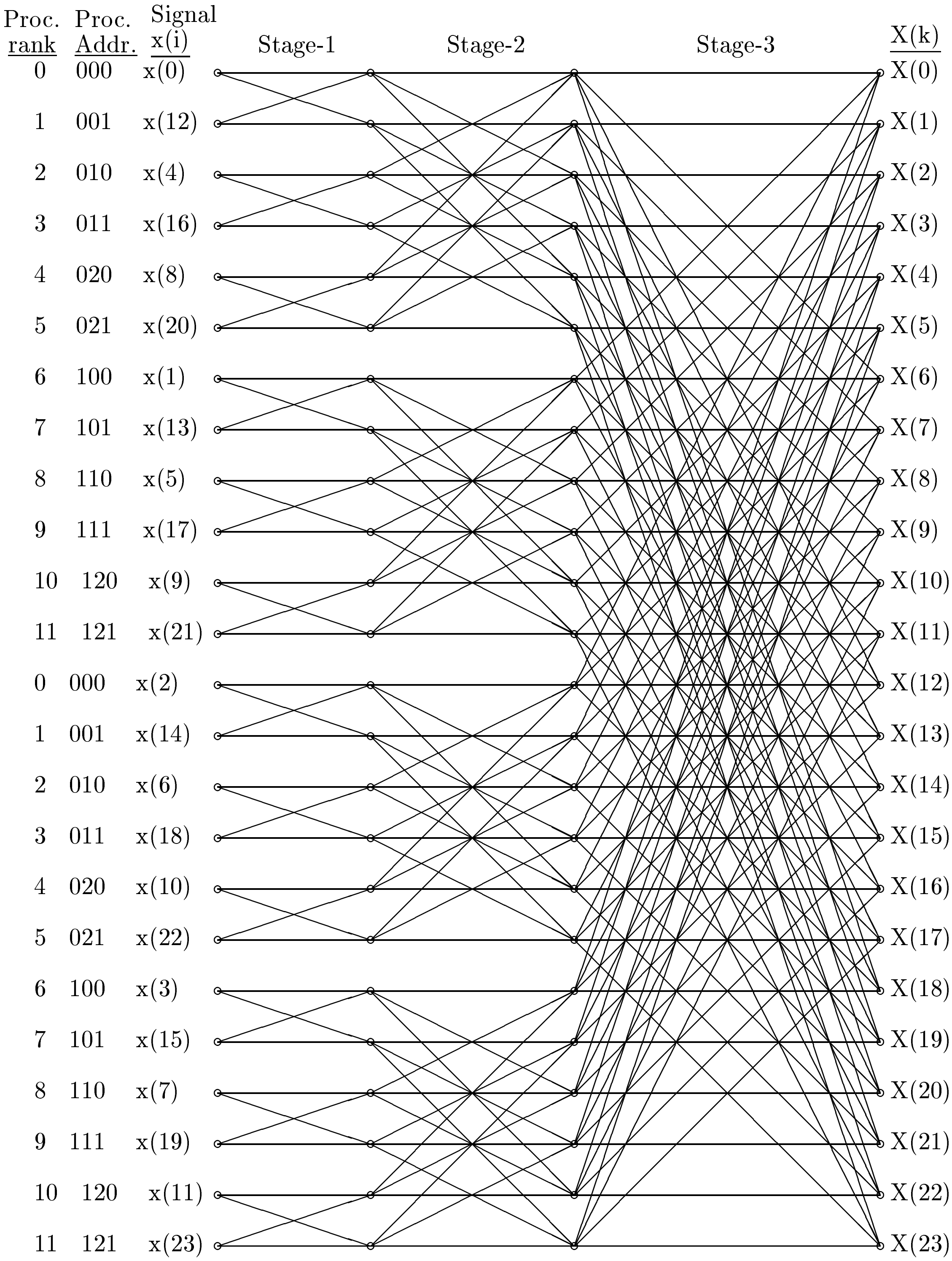


Figure 7.13: 24-point FFT flow diagram with processor allocation on an $SC_{3,2}$.

7.5.3 Pipelined Algorithms

Pipelined algorithms have been developed for the star graph interconnection network using the pipelined view of the star provided by the *pipelined structured ranking scheme* and the *adjacent domain communication paradigm*. In these pipelined algorithms an S_{n+1} is viewed as a pipeline of $n+1$ stages in which S_n^j consists the i-th stage of the pipeline. In a similar manner an $SC_{n,k}$ can be viewed as a pipeline of k stages, an S_n in it consisting a stage of the pipeline. Since the pipelined view of S_{n+1} and $SC_{n,k}$ remain basically the same, any pipelined algorithm developed for S_{n+1} using this view can be implemented on an $SC_{n,k}$ through suitable allocation of the computation stages. It will only require that more than one stage of the computation is allocated to single stage of the pipeline. However, there will be some amount of degradation in performance due to congestion in communication as the communication in the pipeline has to be restricted to confined routing (Section 5.3).

7.6 Summary

In this chapter we have presented efficient algorithms belonging to three different classes. These are global communication algorithms, ascend/descend algorithms and pipelined algorithms.

The global communication operations that we have considered are: broadcasting, gossiping, scattering and accumulation. Based on the TDD and DDT communication paradigms we have developed two broadcasting algorithms for broadcasting on the star. The time complexity for both these algorithms is $(2n-3)(t_h + Mt_w)$. We have also developed a multi-path broadcast algorithm based on the concept of *orthogonal set of composition dimension sequence* and the TDD paradigm. The algorithm is *contention-free* and has a time complexity of $(2n-3)(t_h + \frac{M}{n-1}t_w)$.

We have presented a scattering algorithm based on the DDT paradigm. The time complexity of this algorithm is bounded above by $(2n-3)t_h + 3(n-1)!Mt_w$. One important aspect about this algorithm is the simplicity with which data partitioning is performed for directing the appropriate data elements to the destination nodes.

We have presented a gossiping algorithm based on the TDD paradigm. The algorithm is free of channel contention. The time complexity of the algorithm is same as that for the scattering algorithm. We have also presented an elegant algorithm for the single node accumulation operation based on the HOPC paradigm. The time complexity of this algorithm is bounded above by $4(n-2)t_h + 6(n-1)!Mt_w$.

The global communication algorithms developed here are asymptotically optimal in time for these operations. These cost marginally higher the existing optimal algorithms [FA93b, FA93a, MJ94b] for these operations. However, our algorithms are simple amd easy to implement, and are readily adaptable in star clusters.

Based on the HOPC paradigm and the hierarchical ranking scheme we have developed order preserving ascend/descend type of algorithms for FFT and matrix multiplication. The time complexity of the FFT algorithm for computation of $n!$-point FFT on an S_n is $O(n^2)$. This is identical to an earlier algorithm [FA94]. However, the communication cost of our algorithm is $O(n)$ compared to $O(n^2)$ time required in [FA94]. The time complexity of the matrix multiplication algorithm for $(\ell \times m) \times (m \times p)$ matrix multiplication is $O(m)$. Based on the pipeline structured ranking scheme and ADC paradigm we have developed pipelined algorithms for computation of $n!$-point 1-D FFT and $n! \times n!$-point 2-D FFT on an S_{n+1}. The time complexity of the 1-D FFT algorithm is $O(n^2)$ and that of 2-D FFT algorithm is $O(n.n!)$. We have also developed a time optimal algorithm for prefix computation on the star graph based on the TDD communication paradigm and with the multi-port communication model. This algorithm is congestion-free and has a lower communication cost and thus a higher scalability compared to an earlier algorithm [AQ93].

We have also illustrated, with examples, how the algorithms can be adapted for running on star clusters.

Chapter 8

Processor Allocation

Processor allocation in an interconnection network is an important problem in multicomputer systems. It has extensively been studied for the hypercube and other interconnection networks [CC87, DD95, DH91, RTS95]. Processor allocation is required for supporting a multitasking parallel computing environment. Support for a multitasking environment in a multicomputer system is usually achieved through sharing of the processor resources in contrast to time sharing used in conventional sequential computers. Sharing of processor resource implies allocation of portions of the network to individual parallel programs/jobs. To ensure that a task can be implemented efficiently and that the tasks do not interfere with each other, the allocated processors should form a connected subgraph having a suitable structure and concurrently allocated subgraphs should not overlap. Usually, in a hierarchical interconnection network a sub-network which forms a lower level structure of the same family is allocated. For example, in a hypercube based system distinct subcubes are allocated.

Ideally, a processor allocation scheme should be able to allocate a subgraph of the required size and structure whenever the desired number of free processors are available. For this to be possible the first requirement is that the free processors should not be fragmented in the network. This implies that the allocation strategy should retain large substructures whenever possible. The second requirement is that the allocation scheme should not fail to recognize the structure when it is available. We can in fact view the allocation scheme to consist of two components. The first component consists of a recognizer which, given a set of free processor nodes, recognizes all the possible free subgraphs of the desired structure and size. The second component consists of a selector which selects the subgraph, from the given set of free subgraphs, the allocation of which causes least fragmentation. Another important requirement to a processor allocation scheme is that the scheme should be simple so that it does not involve excessive processing cost as it will add to the parallel processing cost overhead. Due to this requirement it is not always possible to achieve a perfect subgraph recognizer which will recognize all the subgraphs of a given structure and size among the free processors. It also may not be possible to have a perfect selector which causes the least possible amount of fragmentation in the free set of processors. Thus, there will be situations in which the allocation scheme will not be able to allocate subgraphs of desired structure and size in spite of having sufficient number of free processors. Such a situation in the allocation process is referred to as a *miss* or a *failure*. Misses in the allocation process lead to delays in execution of jobs and poor utilization of the processor resources in the network. It is always desirable to minimize the delays in the execution of jobs and the inefficiency in the utilization of the processor resources. Therefore it is a goal of a processor allocation scheme to minimize the number of misses in the allocation process.

A processor allocation scheme can be static or dynamic. In a static allocation scheme the incoming requests are considered for allocation only at some specific time intervals. It does not consider deallocation (processor relinquishment) at any arbitrary time. On the other hand, in a dynamic allocation scheme processor allocation and deallocation are performed at any time depending upon the arrival and completion of jobs. The static allocation schemes introduce delays in the allocation process and lead to poor utilization of the processor resources. Therefore, a dynamic processor allocation scheme is usually preferred. In dynamic allocation the allocator does not have prior knowledge of the future releases of processors by the jobs and therefore it cannot have any control over the fragmentation created by the release process. Hence, it is not possible to have a dynamic allocation scheme which can guarantee failure-free allocation. One has to rely on sound heuristics to minimize the percentage of misses.

In a star graph based interconnection network, substars are the most natural units for allocation. However, the star interconnection network has an extremely rapid rate of expansion. This is evident from the fact that the size of an S_{n+1} is $n+1$ times larger than that of an S_n. This rapid expansion rate, sometimes, leads to poor utilization of the processor resource. For instance, say a task requires a thousand processor nodes. The smallest substar that has a thousand processors is an S_7 which actually consists of 5040 nodes. Thus, if substar allocation is used, in the above instance, the processor utilization will be less than 20%. An alternative is to allocate star clusters. With the star cluster concept it is possible to make better utilization of the processor resource. For example, for the problem requiring one thousand processor nodes one can use an $SC_{6,2}$ consisting of only 1440 nodes. This will lead to a utilization of about 70%. It is easy to see that in no case the utilization will fall below 50%. In fact it is possible to make a better utilization of the processor resources in a star graph, with the star cluster concept, than in a hypercube. This is due to the fact that hypercube size always doubles from one level to the next, whereas star cluster sizes can grow at smaller increments, for example, from $SC_{n,k}$ to $SC_{n,k+1}$. We have already seen in Chapter 5 that star clusters preserve most of the desirable characteristics of the star graph and as a result algorithms developed for the star can be implemented on star clusters with minor or no modifications. In this thesis, therefore, we consider the problem of processor allocation in a star graph based network as the problem of star cluster allocation.

The schemes proposed here adopts the available subgraph technique (AST) and maintains a list of available star clusters (overlapping). We also introduce the concept of a bucket that holds a set of overlapping (available) star clusters. An S_{i+2} bucket consists of a two dimensional bit map each bit representing an S_i in an S_{i+2}. Through use of the hierarchical ranking scheme the problem of recognizing a star cluster $SC_{i,k}$ is converted to a simple process of finding k '1's in a row or a column in an S_{i+2} bucket. We develop two simple strategies for the selection of star clusters with minimal fragmentation. Due to the simplicity of the scheme both allocation and deallocation operations can be performed very efficiently. The worst case computation time for these operations on an S_n is $O(nR)$, where R is the maximum number of requests that may be concurrently allocated.

8.1 The star cluster recognition problem

Due to the hierarchical nature of the star graph an S_{i+1} can be factored into $i+1$ disjoint S_is. Similarly, an S_n can be factored into $\frac{n!}{i!}$ disjoint S_is. The S_is obtained this way are composed with a CDS of $i-1$ members from the set $\{2, 3, ..., n\}$. However, the star graph is also strongly hierarchical. Therefore, it is possible to chose any $i-1$ members for the CDS of the S_is. There are $\binom{n-1}{i-1}$ such CDSs possible. Thus there are in total $\binom{n-1}{i-1}\frac{n!}{i!}$ choices for an S_i in an S_n. Similarly, there are $\binom{n-1}{i}\frac{n!}{(i+1)!}$ choices for an S_{i+1} in an S_n. Note that these S_{i+1}s though distinct are not disjoint.

Now let us consider the choices for a star cluster $SC_{i,k}$ in an S_n for $k > 1$. An S_{i+1} has $i+1$ S_i with a given CDS. By picking any k of these we can get an $SC_{i,k}$ in the S_{i+1}. Therefore, there are $\binom{i+1}{k}$ choices for an $SC_{i,k}$ with a given CDS within an S_{i+1}. However, we can choose any $i-1$ member CDS from the i members of the CDS of an S_{i+1} for the S_is of the $SC_{i,k}$. Therefore, there are $i.\binom{i+1}{k}$ choices for an $SC_{i,k}$ in an S_{i+1} with a given CDS. Hence, in total there are $i.\binom{i+1}{k}\binom{n-1}{i}\frac{n!}{(i+1)!}$ choices for an $SC_{i,k}$ in an S_n for $k > 1$.

The job of a star cluster recognizer is to recognize a star cluster from the available set of free processors in a star graph based network. Ideally it should be possible to recognize a star cluster whenever any of the above choices of star clusters is free. However, in dynamic star cluster allocation, and sub-graph allocation in general, an ideal recognition scheme alone cannot produce good performance. It is also very important that the associated selection scheme matches the recognition scheme. There is a general tendency that a recognition scheme that uses a larger number of CDSs causes more fragmentation to the free set of processors unless a selection scheme of matching sophistication is used. This means that the use of a larger number of CDSs for recognition not only means more processing cost for recognition, but also for the selection process. In fact use of a larger number of CDSs for recognition without suitable modification in the selection process often lead to deterioration in the allocation performance.

In the following we present a single-CDS and a double-CDS method for star cluster recognition. These methods are simple and easy to implement and exhibit reasonably good performance. The selection methods associated with these are also rather simple to implement. For these we use the two dimensional array arrangement of S_is in an S_{i+2} discussed in Section 5.4.1.

8.1.1 Single-CDS and double-CDS recognition of star clusters

In the two dimensional array arrangement of substars presented in Section 5.4.1, the S_{i+1}s lie in the rows, columns and a diagonal of the array. Therefore, the recognition of S_{i+1}s get reduced to a matter of simply picking rows, columns, or the diagonal. The problem of recognizing an $SC_{i,k}$, in turn, consists of picking k S_is from a row, a column or the diagonal of the array. If the S_is are picked only from a row, it corresponds to the *single-CDS* (PCDS) recognition of star clusters. On the other hand, if the S_is are picked from a row, a column or the diagonal of the array, it corresponds to the *double-CDS* (PCDS and an SCDS) recognition of star clusters.

THEOREM 8.1 *The total number of $SC_{i,k}$s recognizable in an S_n with single-CDS is $\frac{n!}{(i+1)!}\binom{i+1}{k}$ and that with double-CDS is $2\frac{n!}{(i+1)!}\binom{i+1}{k}$.*

PROOF. From the rows it is possible to recognize $i+2$ S_{i+1}s and from the columns we have $i+1$ S_{i+1}s, while the diagonal gives another S_{i+1}. Therefore, while with single-CDS we can recognize $i+2$ S_{i+1}s in an S_{i+2}, with double-CDS it is possible to recognize $2(i+2)$ S_{i+1}s. Within each S_{i+1} there are $\binom{i+1}{k}$ choices for an $SC_{i,k}$. Therefore, with single-CDS and double-CDS we can recognize $(i+2)\binom{i+1}{k}$ and $2(i+2)\binom{i+1}{k}$ $SC_{i,k}$s respectively within an S_{i+2}. It has been assumed that the S_{i+2}s are composed of PCDS. Therefore, there are $\frac{n!}{(i+1)!}\binom{i+1}{k}$ such S_{i+2}s in an S_n. Hence, the total number of $SC_{i,k}$s recognizable with single-CDS and double-CDS are $\frac{n!}{(i+1)!}\binom{i+1}{k}$ and $2\frac{n!}{(i+1)!}\binom{i+1}{k}$ respectively. □

8.2 Selection of star clusters

Given the method of recognition of free star clusters, single-CDS or double-CDS, the problem of allocation of star clusters reduces to the problem of selection of the star clusters from a row/column or the diagonal of the array arrangement such that the fragmentation is minimal. Here onwards we shall use the terms allocation and selection interchangeably. Before we discuss the selection methods it is necessary to understand the problem of fragmentation. Fragmentation of a free set of nodes $V_f(S_n)$ in an S_n leads to many small sized star clusters which can not be combined to form larger star clusters. Due to this, at times, request r for a star cluster of size $|r|$ significantly smaller than $|V_f(S_n)|$ also can not be satisfied. To avoid such a situation it is necessary that large star clusters are not broken, whenever possible, to allocate a smaller sized star cluster. That is, the *maximal set of star clusters* [1] should be available after every allocation to ensure minimal fragmentation. For $|V_f(S_n)| = \sum_{i=1}^{n} i!$, the maximal set of free star clusters will consists of the set $\{SC_{i,k_i} | i = 1, 2, \ldots, n\}$. Certain amount of fragmentation can never be avoided due to the sizes of the preceding requests and the order of their occurrence. The situation is further aggravated in dynamic allocation by the randomness in the relinquishing time of the previously allocated star clusters by completed jobs. We do not have control over the fragmentation created by the deallocation process. However, through the use of suitable structures for maintaining the free processor information it is possible to recognize all the available free star clusters, with single-CDS or double-CDS as the case may be, with minimum effort.

8.2.1 The free list

As in any other processor allocation scheme in our star cluster allocation/selection scheme too the structure of the free processors list assumes importance. In this scheme the free processor information is maintained in lists of buckets. Each bucket represents a substar at some level and consists of a two dimensional bit map. An S_{i+2} bucket contains $(i + 2) \times (i + 1)$ bits and each bit represents S_is in the S_{i+2}. A bucket in fact corresponds to the two dimensional organization of the S_is in an S_{i+2}. A bit indicates whether the corresponding S_i is available or not. The bucket is identified by the rank of the S_{i+2} in the S_n. Buckets are maintained only for partially available substars S_{i+2}s. An exception is the bucket for the S_n which is always maintained, whether full or empty. A separate list of buckets is maintained for every substar level. The highest and the lowest substar levels for which bucket lists need to be maintained are n and 3 respectively. The bucket list at a substar level will be empty if there is no partially

[1] the concept of *maximal set of subcubes* (MSS) has been used in the context of subcube allocation in [DH91].

available substar at that level. Initially there is only one full bucket and that is in the level n list. To ensure that none of the available star clusters go unrecognized, the information about the availability of a substar is maintained in a bucket at the highest possible substar level. For example, if there is a fully free S_i, it is maintained as a single element in an S_{i+2} level bucket, not as a row in an S_{i+1} bucket or as a full bucket at the S_i level list. Care is taken in this regard both at the time of allocation and deallocation. At the time of deallocation when a complete row of S_is become available in an S_{i+2} bucket, the corresponding S_{i+1} is transferred to the parent S_{i+3} bucket.

8.2.2 Single-CDS allocation

Assume that there is an allocation request for an $SC_{i,k}$. The star cluster selection algorithm for single-CDS allocation is as follows.

ALGORITHM 8.1 **Allocate_Single-CDS($SC_{i,k}$)**

1. *If the $i+2$ level contains free PCDS $SC_{i,k}$s take the first bucket in the list that has a PCDS $SC_{i,k}$. Pick a least free row, in the bucket, having a free $SC_{i,k}$. Allocate the $SC_{i,k}$ consisting of k elements in the row corresponding to the least free columns.*

2. *In case the $i+2$ level bucket list does not have a free PCDS $SC_{i,k}$, then go to the lowest level above $i+2$ that has a non-empty bucket list. If such a level exists let it be l. Otherwise, reject the request.*

   (a) *Pick an element (S_{l-2}) that belongs to a least free row and corresponds to the least free column in the first bucket in the list.*

   (b) *Repeat the following steps for $j = l$ down to $j = i+4$.*

      i. *Create a new bucket at level $j-1$ for the parent of S_{j-2} picked in the previous step and fill the row corresponding to the S_{j-2}.*

      ii. *Decrement j.*

      iii. *Pick an element (S_{j-2}) from the row in the newly created bucket.*

   (c) *Create a new S_{i+2} bucket for the parent of the S_{i+2} and fill the row corresponding to the S_{j-2}. Allocate an $SC_{i,k}$ from the newly filled row by picking k elements.*

3. *If any bucket becomes empty in the process, remove it from the list.*

THEOREM 8.2 *The worst case time complexity of the algorithm single-CDS allocation on an S_n is $O(nR)$, where R is the maximum number of requests that may be concurrently allocated.*

PROOF. In the step-1 the algorithm searches the S_{i+2} bucket list for the availability of an $SC_{i,k}$. The complexity of search in a bucket is $O(n)$. Assuming the maximum length of a bucket list is d, the worst case complexity of the search is $O(nd)$. The cost of picking an $SC_{i,k}$ from a bucket in the step is $O(n)$. Therefore, the over all complexity of step-1 is $O(nd)$.

The worst case complexity of finding the non-empty bucket list in step-2 is $O(n)$. The complexity of picking the element from the least free row and corresponding to the least free column in step-2(a) is $O(n)$. The cost of creating a new bucket and filling a row in it in step-2(b)(i) is $O(n)$. and that of picking an element in step-2(b)(iii) is $O(n)$. Therefore the overall complexity of step-2 is $O(n^2)$.

The cost of searching the parent bucket in step-2(c) is $O(d)$. Inserting a new row in the bucket costs $O(n)$ time. Therefore step-2(c) costs $O(n + d)$. The picking of the k S_is in step-2(d) involves a sorting so that its complexity is $O(n^2)$. Thus the overall complexity of step-2 is $O(n^2)$.

The complexity of checking for whether a bucket is empty and removing it from the list in step-3 is $O(1)$.

A bucket is maintained in a list only if it is partially free. The partially free number of buckets in a list cannot exceed the number of requests allocated at the time. A loose upper bound on the number of buckets in a list is the maximum number of requests that can be concurrently allocated. Consider that all the requests are for SCi, is. An S_{i+2} can satisfy $(i + 2) + 1 = i + 3$ such requests, for $i > 1$, and one S_i will remain free in it so that the bucket will be maintained. A new S_{i+2} bucket will be created for every $i + 3$ requests and will be added to the list. Let the maximum number of requests that may be concurrently allocated be R. Then the maximum possible length of the S_{i+2} bucket list will be $\frac{R}{i+3}$. The maximum possible bucket length will be for $i = 2$, $i.e.$ $d = \frac{R}{5}$. This will occur when there is a continuous sequence of R requests for $SC_{2,2}$s. It is assumed that the network size is sufficiently large. In general, d can be greater than n. Therefore the overall worst case complexity of the algorithm is $O(nR)$. □

The single-CDS deallocation algorithm is as given below.

ALGORITHM 8.2 **Deallocate_Single-CDS($SC_{i,k}$)**

1. *Search for the parent S_{i+2} bucket in the appropriate list.*

2. *If found, insert the $SC_{i,k}$ in the bucket.*

   (a) *If the corresponding PCDS S_{i+1} has become fully free remove it from the bucket and perform deallocation on the S_{i+1}.*

   (b) *If the S_{i+2} bucket becomes empty in the process remove it from the list.*

3. *If the parent S_{i+2} bucket not found, Create it and insert the $SC_{i,k}$ in it.*

THEOREM 8.3 *The worst case time complexity of single-CDS deallocation algorithm is $O(nR)$.*

PROOF. The worst case complexity of searching for the parent bucket in step-1 is $O(d)$ where d is the maximum length of a bucket list. As the upper bound on the value of d is R the cost reduces to $O(R)$. The cost of insertion in step-2 is $O(n)$. The step-2(a) constitutes a recursion in the algorithm and the depth of recursion can be at most $n-3$, as the lowest and the highest levels in the bucket list are 3 and n respectively. The cost of each of step-2(b) is $O(1)$. The cost of bucket creation and insertion in step-3 is $O(n)$.

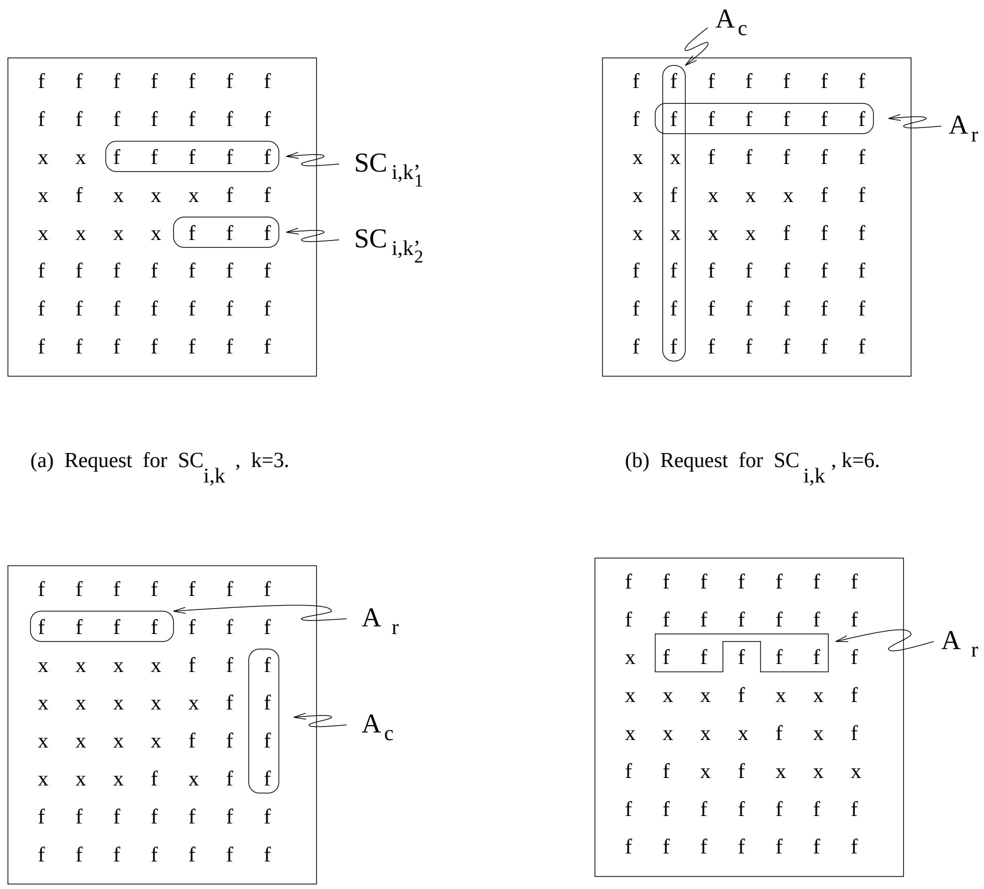
Since, in general, $R > n$, and the upper bound on the depth of recursion is n the overall worst case complexity of the algorithm is $O(nR)$. □

8.2.3 Double-CDS allocation

Let us first look at some example scenarios that influence our strategy for the selection of free star clusters in double-CDS allocation.

Observation 1:

Consider the partially allocated bucket depicted in Figure 8.1(a). It is necessary to allocate an $SC_{i,k}$ from the bucket. The allocation can be done from any row or column that has k or more free S_is. A row or column with k' free S_is give a free $SC_{i,k'}$. Let SC_{i,k'_1} and SC_{i,k'_2} be two such free star clusters in the bucket such that $k'_1 > k'_2 \geq k$. If the $SC_{i,k}$ is allocated from the row/column contributing the SC_{i,k'_1} the free SC_{i,k'_1} will be lost and if it is allocated from the row/column contributing the SC_{i,k'_2} the free SC_{i,k'_2} will be lost. Since SC_{i,k'_1} is larger than SC_{i,k'_2} preserving the SC_{i,k'_1} is preferable. Therefore, the observation is that it is preferable to perform an allocation from the least free row/column in a bucket.



(a) Request for $SC_{i,k}$, k=3.

(b) Request for $SC_{i,k}$, k=6.

(c) Request for $SC_{i,k}$, k=4.

(d) Request for $SC_{i,k}$, k=4.

Figure 8.1: A set of observations on good star cluster allocations. x : allocated S_i, f : free S_i.

Observation 2:

Consider a situation in which there are $k + \delta$ fully free rows in an S_{i+2} bucket as shown in Figure 8.1(b). None of the partially free rows has more than $f_{max} < k$ free S_is and it is required to allocate an $SC_{i,k}$ from the bucket. The largest available free star cluster in the bucket is an $SC_{i+1,k+\delta}$. If the strategy is to allocate from the smallest free row/column in the bucket then the allocation will obviously be from a column, with an allocation A_C, as shown in the figure. This will reduce the largest free star cluster in the bucket to $SC_{i+1,max\{\delta,f_{min}\}}$, where f_{min} is the size of the smallest partially free row in the bucket. If instead, the allocation is done from a row with an allocation, A_r as shown in the figure, the largest free star cluster remaining in the bucket will be an $SC_{i+1,k+\delta-1}$. It is easy to see that $k+\delta-1 > max(\delta, f_{min})$. Consequently, the row allocation is superior in this case. It may be recollected that there are equal number of PCDS and SCDS S_{i+1}s in an S_{i+2} bucket. The above asymmetry in allocation is occurring due to the fact that the initial allocations from the bucket were PCDS allocations. If the initial allocation from the bucket was an SCDS one, it would be preferable to make SCDS allocations for the succeeding requests too, in a similar circumstance.

Observation 3:

Consider the partially allocated bucket depicted in Figure 8.1(c). Here, k or more rows have partially been allocated and all these partially allocated rows have less than k free S_is. There are ρ rows fully free and the largest free star cluster is $SC_{i+1,\rho}$. The partially allocated rows can contribute to form one or more SCDS $SC_{i,k}$s. It is required to allocate an $SC_{i,k}$. This allocation can be from either a row, *e.g.* the allocation A_r, or from a column, *e.g.* the allocation A_c, as shown in the figure. It can easily be seen that the allocation A_c leaves the largest free star cluster in the bucket intact while after the allocation A_r the largest free star cluster will be reduced to an $SC_{i+1,\rho-1}$. Therefore, in this case the column allocation is preferable.

Observation 4:

Consider a situation as depicted in Figure 8.1(d) in which an allocation from a row is preferable and the allocation should be from the least free row that has a free $SC_{i,k}$. To cause minimum fragmentation in the SCDS star clusters the S_is should be picked from the least free row such that the corresponding free SCDS star clusters are of the least sizes. That is, the corresponding columns are the least free ones. From symmetry we can say that while allocating an $SC_{i,k}$ from a row (column) the S_is should be picked from the least free columns (rows).

The allocation strategy

Based on the above observations we formulate the strategy for star cluster allocation from a given bucket as follows.

1. Allocate a star cluster from the smallest available free row in the bucket, whenever possible, except for the following cases. Here the preference is given to row allocation based on observation 2.

2. If none of the free rows is sufficiently large for the allocation of an $SC_{i,k}$, allocate from a column, if possible. We shall not use the diagonal for simplicity of the algorithm. A diagonal element will be used in a row star cluster whenever available.

3. If there are k or more partially free rows none of which have k free S_is but these contribute to form an SCDS $SC_{i,k}$, perform the allocation from column. This is based on

observation 3.

4. In all the above cases in the allocation of an $SC_{i,k}$ from a row (column) pick the S_is from the row (column) that have the least number of free sibling S_is in the columns (rows). This rule follows from observation 4.

Based on the above strategy we come up with the following algorithm for double-CDS star cluster allocation.

ALGORITHM 8.3 **Allocate_Double-CDS(**$SC_{i,k}$**)**

1. *If the $i+2$ level bucket list contains free PCDS/SCDS $SC_{i,k}$s take the first bucket in the list that has an $SC_{i,k}$.*

   (a) *Ignore the largest free PCDS $SC_{i+1,k'}$ in the bucket, if any, and search the remaining rows in the bucket for a free PCDS/SCDS $SC_{i,k}$.*

      i. *If available allocate an $SC_{i,k}$ from the least free row/column.*

      ii. *Otherwise, take an S_{i+1} from the ignored $SC_{i+1,k'}$ and and allocate an $SC_{i,k}$ from it by picking k S_is corresponding to the least free columns.*

2. *In case the $i+2$ level bucket list is empty or does not have a free PCDS $SC_{i,k}$, then go to the lowest level above $i+2$ that has a non-empty bucket list. If such a level exists let it be l. Otherwise, reject the request.*

   (a) *Pick an element (S_{l-2}) from the first bucket in the list.*

   (b) *Repeat the following steps for $j = l$ down to $j = i+4$.*

      i. *Create a new bucket at level $j-1$ for the parent of S_{j-2} picked in the previous step and fill the row corresponding to the S_{j-2}.*

      ii. *Decrement j.*

      iii. *Pick a free element (S_{j-2}) from the newly created bucket.*

   (c) *Create a new S_{i+2} bucket for the parent of the S_{i+2} and fill the row corresponding to the S_{j-2}. Allocate an $SC_{i,k}$ from the newly filled row by picking k elements.*

THEOREM 8.4 *The worst case time complexity of the algorithm double-CDS allocation on an S_n is $O(nR)$, where R is the maximum number of requests that may be concurrently allocated.*

PROOF. The double-CDS algorithm differs from the single-CDS one in step-1. The complexity of searching for a PCDS/SCDS free $SC_{i,k}$ in an $i+2$ level bucket is $O(n)$ because it will consist of first checking the rows for free $SC_{i,k}$ and then checking the columns for free $SC_{i,k}$. In the worst case it will be necessary to check all the d buckets in the $i+2$ level bucket list resulting in a complexity of $O(nd)$.

The complexity of ignoring the largest $SC_{i+1,k'}$ and searching the remaining rows for free $SC_{i,k}$ in step-2(a) is again $O(n)$. The cost of allocation of an $SC_{i,k}$ in step-1(a)(i) is $O(n^2)$ and that in step-1(a)(ii) is $O(n)$. Therefore, the overall complexity of step-1 is $O(n^2 + nd)$. Based on the same argument as in Theorem 8.2 this reduces to $O(nR)$. Hence, the overall complexity of this algorithm is also $O(nR)$. $\square$

The double-CDS deallocation algorithm differs from the single-CDS algorithm in that the $SC_{i,k}$ to be deallocated may also be an SCDS one. As a result more than one (say k') S_{i+1}s may become completely free which will need to be removed from the bucket and deallocated as an SCDS $SC_{i+1,k'}$. The algorithm is given below.

ALGORITHM 8.4 **Deallocate_double-CDS($SC_{i,k}$)**

1. *Search for the parent S_{i+2} bucket in the appropriate list.*

2. *If found, insert the $SC_{i,k}$ in the bucket.*

    (a) *If one or more of the corresponding PCDS S_{i+1}s become fully free remove the corresponding PCDS $SC_{i+1,k'}$ from the bucket and perform a deallocation on it.*

    (b) *If the S_{i+2} bucket becomes empty in the process remove it from the list.*

3. *If the parent S_{i+2} bucket not found, Create it and insert the $SC_{i,k}$ in it.*

THEOREM 8.5 *The worst case time complexity of double-CDS deallocation algorithm in $O(nR)$.*

PROOF. The double-CDS deallocation algorithm differs only in the step-2(a) from that of the single-CDS algorithm. This step costs $O(n)$ instead of $O(1)$ in the single-CDS case. However, the order of overall complexity of the algorithm remains $O(nR)$. $\square$

8.3 Simulation results

Both the allocation schemes have been simulated to study their performance in terms of number of misses at different utilization levels. The simulator generates job requests requiring different star cluster sizes in a given range. For this purpose each star cluster size is assigned an index. The index value for an $SC_{i,k}$ is $\frac{i(i-1)}{2}+k$. A uniform distribution is assumed over the range of given index values. The time duration T_d for a job is distributed uniformally over a given range $[T_{d\_min}, T_{d\_max}]$. The inter-arrival time T_i, between job requests, is also assumed to be uniformally distributed over a given range $[0, T_{i\_max}]$. The control over the job load on the system is achieved by controlling the frequency of arrival of jobs. This is done by making T_{i_max} dependent on T_{d_max}. We define a ratio, $\rho = \frac{T_{d\_max}}{T_{i\_max}}$. An increase in the value of ρ reduces the inter-arrival time of jobs with respect to the duration of the jobs, thus increasing the frequency of job requests. A job request on arrival is considered for allocation and if it can not be satisfied, due to miss or unavailability of sufficient number of free processors; it is not reconsidered for allocation. Reconsideration of missed requests may lead to counting of the same missed request multiple times.

Each run of the simulation is carried out until 10,000 successful allocations are performed. The utilization percentage of the processor resource is computed as $\frac{\sum_{j=1}^{10,000} T_{d\_j}|r_j|}{Tn!} \times 100\%$, where n is the order of the star graph, T is the total duration of the experiment, T_{d_j} is the duration of the job r_j and $|r_j|$ is its size.

The miss versus utilization level plots for different ranges of input request sizes are given in Figure 8.2 and Figure 8.3. Both miss and utilization are in percentage. The miss percentage is computed with respect to successful number of requests.

The number of misses in case of the double-CDS scheme on the average is smaller than that in case of the single-CDS scheme at any utilization level even though the difference is marginal at low utilization levels, which is expected. The performance difference grows with increase in the level of utilization of the processor resources. At higher utilization levels the number of misses grow more sharply for the single-CDS scheme than for the double-CDS scheme. The superior performance of the double-CDS scheme becomes more visible when the request sizes are restricted to a smaller range. The performance difference is maximum when the allocations are always from the S_n level bucket. That is, the requests are restricted to the range $SC_{n-2,1}$ to $SC_{n-1,1}$. See Figure 8.3. This is due to the fact that in this case the advantage of double-CDS recognition can be exploited to the maximum.

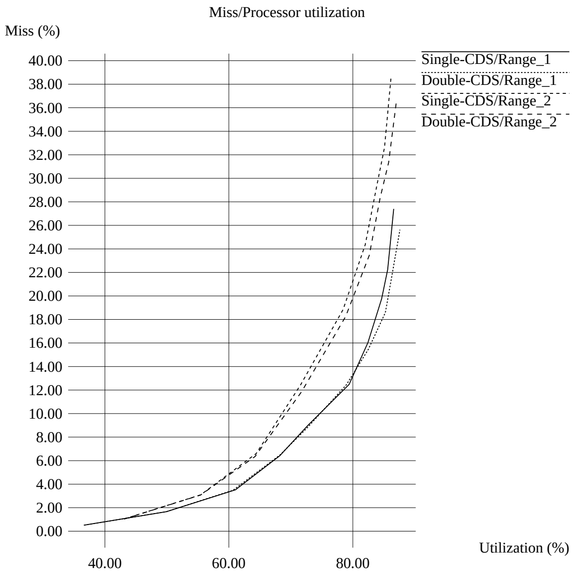


Figure 8.2: The miss/processor utilization plot for single-CDS and double-CDS allocation schemes for allocations in two different ranges of star clusters in an S_9. Range_1: $SC_{1,1}$ to $SC_{9,1}$, Range_2: $SC_{5,1}$ to $SC_{9,1}$.

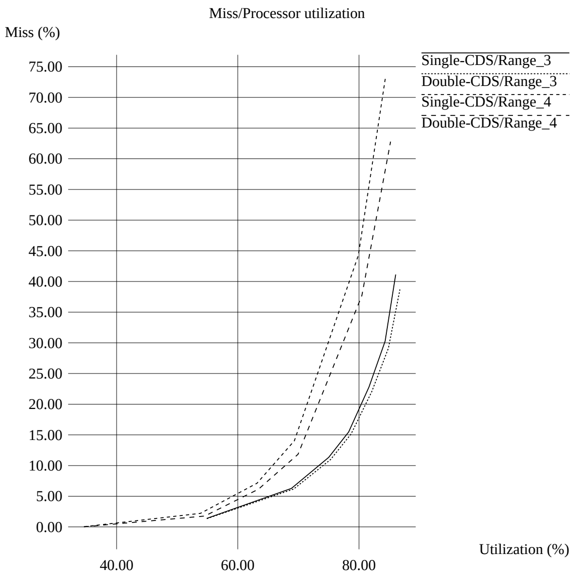


Figure 8.3: The miss/processor utilization plot for single-CDS and double-CDS allocation schemes for allocations in two different ranges of star clusters in an S_9. Range_3: $SC_{5,1}$ to $SC_{8,1}$, Range_4: $SC_{7,1}$ to $SC_{8,1}$.

8.4 Summary

In this chapter we have presented two schemes for processor allocation in star graph based interconnection networks. The schemes allocate processors in terms of *star clusters*. The schemes are simple and have time complexity of $O(nR)$ for both allocation and deallocation on an S_n if R is the maximum number of allocation requests that can be satisfied concurrently. The free processor information is maintained as *overlapping* list of star clusters. A two dimensional arrangement of S_is in an S_{i+2} is used to convert the problem of star cluster ($SC_{i,k}$) allocation to a problem of picking S_is from rows and columns of the arrangement. A concept of bucket is used to maintain such two dimensional arrangements so that the allocation problem gets vastly simplified. The single-CDS scheme allocates star clusters only from the rows while the double-CDS scheme allocates a star cluster either from a row or from a column depending upon availability and suitability. Simulation studies show reasonably good performance for both the schemes. The double-CDS scheme performs better than the single-CDS scheme at higher utilization levels. The performance difference is more prominent when the input request size is limited to a small range of star clusters near the higher end. It is believed that increasing the number of CDSs above two will not lead to performance improvement of the allocation scheme compared to the increase in the complexity of the scheme. This is due to the fact that use of larger number of CDSs tend to cause more fragmentation in the free set of processors.

Chapter 9

Conclusions

In this thesis we have presented several results concerning the use of the star graph as an interconnection topology for parallel computing. We believe that these results have extended and also contributed to our knowledge of the star graph as an interconnection topology. We have introduced a fundamental set of theoretical ideas. These help in dealing with several issues concerning efficient parallel processing on star graph based networks. We have presented results that demonstrate the utility of these concepts.

The key set of ideas includes an isomorphic transformation called the *symbol swap transformation*, a concept called *composition dimension sequence* (CDS) for specifying a hierarchical decomposition of the star graph and a *domain decomposition* of the star graph based on the ideas of *(i,k)-centre* and the *domain of a node.*

Based on a special case of the symbol swap transformation called the *i-th symbol swap transformation*, we have developed a scheme for ranking the nodes and the substars of a star graph called the *hierarchical ranking scheme.* This gives a mixed radix generalized cube structured view to the star graph and allows order preserving communications to be performed in constant time, as in a hypercube, in contrast to $O(n)$ time required in an earlier node ranking scheme [AQS91]. The hierarchical ranking scheme also leads to an isomorphic transformation called *peer node imaging.*

We have developed another ranking scheme called the *pipeline structured ranking scheme* which gives a pipelined view of the star graph and provides a convenient framework for developing pipelined algorithms for the star graph.

Based on the domain decomposition of the star graph and the hierarchical ranking scheme we have developed four different communication paradigms, namely, i) hierarchical order preserving communication (HOPC), ii) adjacent domain communication (ADC), iii) transmit-and-domain-distribute (TDD), and iv) domain-distribute-and-transmit (DDT). These paradigms have low communication overheads due to path length and edge-congestion and are suitable for developing a variety of algorithms for star graph based networks. Based on the TDD and DDT communication paradigms we have developed simple asymptotically optimal global communication algorithms for broadcasting, gossiping, and scattering on star graphs. Based on a concept called *orthogonal set of composition dimension sequences* we have developed a method of constructing $n-1$ mutually contention-free broadcast trees in an S_n with a given root node. These trees are used in a multipath broadcasting algorithm suitable for large sized messages. Based on the HOPC paradigm and the hierarchical ranking scheme we have developed order preserving ascend/descend type of algorithms for FFT and matrix multiplication. The FFT algorithm has a lower communication cost and thus higher scalability compared to an earlier algorithm [FA94]. Based on the pipeline structured ranking scheme and ADC paradigm we have developed pipelined algorithms for 1-D and 2-D FFT. We have also developed a time optimal algorithm for prefix computation on the star graph based on the TDD communication paradigm. This algorithm is congestion-free and has a lower communication cost and thus a higher scalability compared to an earlier algorithm [AQ93].

We have proposed the idea of *star cluster* as a subgraph of a star graph induced by a set of sibling substars. It is shown that a star cluster retains most of the desirable properties of the star graph. As a result most of the algorithms developed for the star graph can be adapted to a star cluster with minor or no modifications. We can achieve near linear growth rate by building our interconnection network based on the star cluster. Further, smaller star clusters within such a network can be allocated to computations according to their processor requirements. As a consequence, the concept of star cluster is also useful in efficient utilization of star graph based interconnection networks.

Based on the isomorphic transformations, hierarchical ranking scheme and the idea of star cluster we have developed new techniques for embedding several important interconnection structures on the star graph. We have developed several new schemes for the generation of even length cycles in minimal star clusters. We have introduced the concept of *approximate cycles*. A very small fraction of the edges in an approximate cycle is of dilation three. Approximate cycles can be generated from unit dilation cycles through i-th symbol swap transformation. Such approximate cycles have the property of being *parallel* to the original cycle in the sense that the peer nodes have a distance bound of three. Based on the method of generation of parallel cycles and on the peer node imaging technique mentioned earlier, we have developed a scheme for embedding a torus of arbitrary dimension and size on the star graph. The embeddings have a dilation bound of four. The congestion on the column edges is at most two. The congestion on the row edges and the expansion of the embedding depends upon the routing of the row edges. With a routing scheme called *confined routing* the row edge congestion is $O(n)$ and worst case expansion is four. With another routing scheme called *third substar routing* the edge congestion is two and the worst case expansion is $O(n)$. However, in most cases the expansion is small for both the routing schemes.

Based on symbol swap transformation and a *two dimensional arrangement of substars* we have developed two schemes for embedding full binary trees in star graphs with dilation two. The first scheme constructs a congestion-free embedding of a full binary tree of height $(n+1)\lfloor log_2 n\rfloor - 2^{\lfloor log_2 n\rfloor+1} + 1$ in an S_n. This scheme is an improvement in dilation by a factor of two over an earlier work [BHOS94]. The second scheme constructs an edge-congestion-free embedding of full binary tree of height at least $log_2(n!) - 3$ in an S_n, for $n \leq 19$. This is the most compact known embedding of full binary tree on the star graph for any dilation for a wide practical range.

We have also developed a scheme for embedding pyramids on the star graphs with dilation four based on peer node imaging. An attractive feature of this embedding scheme is that the convergence factor(s) of the embedded pyramid can be varied to simulate a dynamically varying structure. The expansion of the embedding depends upon the convergence factor(s) of the pyramid. For a quad pyramid the expansion is $\approx \frac{n!}{4^{n-2}}$. The edge congestion in the embeddings is c_{max}, the maximum of the convergence factors of the pyramid.

Based on the hierarchical ranking scheme and the associated decomposition of the star graph, we have developed a concept of two dimensional recognition of substar/ star clusters. Based on this concept we have developed two efficient schemes for allocation of star clusters in a star graph for a general purpose multi-tasking parallel computing environment. The time complexity for allocation/deallocation on S_n with both the schemes is $O(nR)$, where R is the maximum number of requests that can be concurrently allocated. Simulation studies have been carried out on both the schemes and found to have good performance characteristics.

To summarise, we have proposed new ranking schemes, developed communication paradigms and some classes of algorithms for the star graph. We have also shown how to embed a variety of popular interconnection topologies on the star. The star cluster concept addresses the problems arising from the rapid expansion rate of the star graph. The results reported in the thesis give us positive indications that the star graph topology can be utilized in the construction of parallel computer systems.

9.1 Future work

The general approach undertaken in this work may be used to investigate properties of graphs that share similar properties with the star. This mainly addresses the Cayley graphs. However, with reference to the star graph alone there are a number of particular issues that can be taken up for further research. We outline some of these issues below.

1. We have defined and characterized the *dimension swap transformation*. Unlike the symbol swap transformation, the dimension swap transformation has not been used in our schemes. However, this transformation seems to have potential application in the embedding area. It may be worth while to investigate for application of this isomorphic transformation.

2. Due to the isomorphic nature of the star clusters ($SC_{n,k}$s) in a star (S_{n+1}) it seems possible that the star cluster concept can be used for developing fault-tolerant systems on computers built with star graph based networks.

3. We have used the concept of *distance-2 set of star clusters* to develop efficient embedding scheme for FBTs. The general *distance-k set of star clusters* may possibly be applicable in the development of embedding schemes for other interconnection structures.

4. In general it is possible to recognize more number of star clusters by using higher number of composition dimension sequences. Therefore in may be worthwhile to investigate for more compact embedding of FBTs through use of more than two CDSs.

5. Use of larger number of CDSs for processor allocation may also lead to more efficient processor utilization. However, to make the allocation and deallocation process efficient it will be necessary to investigate for suitable data structures for maintaining the free list and appropriate heuristics for the selection of free star clusters for allocation.

Appendix A

Parallel paths between any two nodes in an S_n

The following are the procedures for computing the parallel paths between two nodes in S_n with relative permutation R [DT93a, JLD90, DMS91]. It is assumed that there are c p-cycles in R and these cycles contain a total of m symbols.

Case-1: $R(1) = 1$

A. Minimal paths of length $c + m$.

1. Give an arbitrary order to the p-cycles.

2. Start the path by moving the first symbol of the first p-cycle to position 1. Leave the reduction of this cycle incomplete.

3. Reduce the rest of the p-cycles.

4. Complete reduction of the first cycle to obtain a path of length $c + m$.

5. Make a left rotation of the symbols of the first p-cycle so that the second symbol occupies the first position. Repeat the steps 2-4 to obtain another parallel path.

6. Repeat the step 5, so that each of the symbols in the p-cycle takes the first position once.

7. Make a left rotation of the cycle positions so that the second cycle takes the first position. Repeat the steps 2-6, to obtain another set of parallel paths.

8. Repeat the step 7, so that each p-cycle takes first position once, to obtain the remaining parallel paths of length $c+m$.

B. Paths of length $c+m+2$.

1. Move one of the invariant symbols to the first position.
2. Reduce the p-cycles in any order.
3. Correct the invariant symbol moved in step 1 to obtain a path.
4. Repeat the steps 1-3 for each of the invariant symbols, except '1', to obtain the remaining parallel paths of length $c+m+2$.

In the set of parallel paths obtained as above, the dimensions of the starting and the terminating edges of a path are identical. There will be m parallel paths of length $c+m$ and $n-m-1$ parallel paths of length $c+m+2$.

Case-2: $R(1) \neq 1$

A. Minimal paths of length $c+m-2$.

1. Give an order to the p-cycles.
2. Reduce the cycles one after another according to the order to obtain a path.
3. Make a left circular shift of the p-cycle positions in the order so that the second cycle becomes the first. Repeat step 2, to obtain another parallel path of length $c+m-2$.
4. Repeat step 3, so that each cycle occupies the first position once, to obtain the remaining paths of length $c+m-2$.

There will be c parallel paths of length $c+m+2$, one corresponding to each p-cycle occupying the first position once. The dimension of the starting and the terminating edges of a path will differ. The dimension of the starting edge will correspond to the position in R of the first symbol in the first p-cycle. The dimension of the terminating edge will correspond to the position in R of the first symbol in the last p-cycle, except when the last p-cycle contains '1'. In this case the dimension of the terminating edge will correspond to the last symbol in the last p-cycle.

B. Paths of length $c + m$.

1. Move one of the symbols, say 'x', $x \neq 1$, not occupying the first position in its p-cycle to merge the p-cycle with the p-cycle containing '1'.

2. Move symbol '1' to position 1 to bring it out of the new p-cycle.

3. Make the new p-cycle the last cycle with 'x' occupying the first position in it.

4. Reduce the cycles one by one, in order to obtain a path of length $c + m$.

5. Repeat the steps 1-4, for each of the symbols in the p-cycles not occupying the first position in its cycle, to obtain the remaining parallel paths of length $c + m$.

C. Paths of length $c + m + 2$.

1. Move one of the invariant symbols, say 'x', $x \neq 1$, to position 1 to merge it with the p-cycle containing '1'.

2. Move symbol '1' to position 1 to bring it out of the new p-cycle.

3. Make the new p-cycle the last cycle with 'x' occupying the first position in the cycle.

4. Reduce the cycles one by one, in order, to obtain a path.

5. Repeat the steps 1-4, for each of the invariant symbols, to obtain the remaining parallel paths of length $c + m + 2$.

In the paths of length $c+m$ and $c+m+2$, the dimensions of the starting and the terminating edges of a path are identical. These dimensions correspond to the position in R of 'x', the symbol moved in step 1. There will be $m - c - 1$ parallel paths of length $c + m$ and $n - m$ parallel paths of length $c + m + 2$.

Appendix B

Correctness proof of algorithm D2_Cluster

In the following, we present the proof of correctness of the algorithm D2_Clusters, presented in Section 5.4.3, for obtaining a distance-2 set of star clusters. Recollect that the algorithm seeks to obtain ρ, $k_n < \rho < n-1+k_n$, $SC_{n-2,k_{n-1}}$s in an SC_{n-1,k_n}.

PROOF. According to the algorithm an element is allocated to a row cluster only if sufficient number of free elements remain in the corresponding column to fully satisfy the column cluster. Therefore, all the column clusters are, finally, always satisfied by the algorithm. We have only to prove that allocation to none of the row clusters remain incomplete.

Let us first consider the algorithm ignoring the step-2(b). In this step the algorithm checks for the unused elements in a row. In case of availability of unused elements some of these are allocated to the row cluster so that some already allocated elements in the row can be freed to be allocated to incomplete column clusters.

According to the allocation of the elements the two dimensional matrix can be divided into several distinct partitions as shown in Figure B.1. In step-1 of the algorithm, all the elements of the first $n-1-(\rho-k_n)$ columns are allocated to the row clusters. These correspond to the partition R which is of size $k_n \times (n-1-(\rho-k_n))$. The elements in the partition A, of size $(k_n-k_{n-1}) \times (k_{n-1}-(n-1-(\rho-k_n)))$, are allocated to the first k_n-k_{n-1} row clusters. These row clusters are fully satisfied combinedly by the partitions A and R1. The elements in the partition B, of size $k_{n-1} \times (k_{n-1}-(n-1-(\rho-k_n)))$, are allocated to the first $k_{n-1}-(n-1-(\rho-k_n))$ column clusters. The corresponding column clusters are fully satisfied by the partition B. The elements in area C, of size $(k_n-k_{n-1}) \times (n-1-k_{n-1})$, are

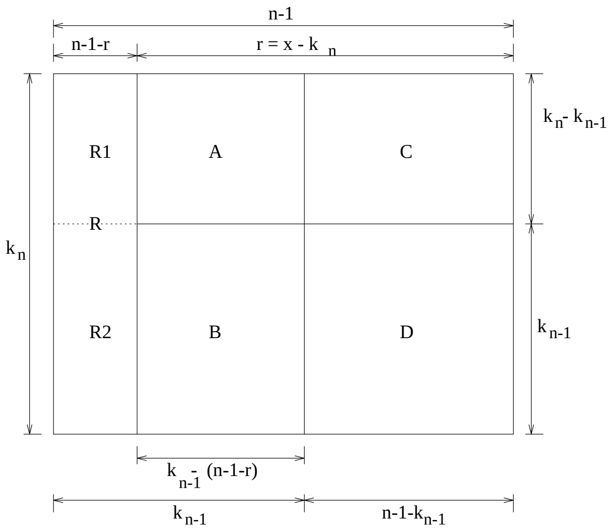


Figure B.1: Partitioning of the array of S_{n-2}s in SC_{n-1,k_n} according to allocation ignoring the step-2(b). In the figure, $x = \rho$.

---

allocated to the corresponding column clusters. Each of these column clusters are allocated $k_n - k_{n-1}$ elements from this partition. As $k_{n-1} > \frac{k_n}{2}$, $k_n - k_{n-1} < k_{n-1}$. Therefore, these column clusters will not be fully satisfied by the partition C. Let us call these *C-clusters*. The remaining $2k_{n-1} - k_n$ elements to each of these C-clusters are to be allocated from the partition D. Similarly, the partition R2 provides $n - 1 - (\rho - k_n)$ elements to each of the k_{n-1} row clusters corresponding to the area. Let us call these the *R-clusters*. The remaining $k_{n-1} - (n - 1 - (\rho - k_n))$ elements to each of the R-clusters are to be allocated from the partition D.

Recall that finally all the column clusters will definitely be fully allocated. This means that only some of the R-clusters may not be fully allocated. Let us now look at when such failures may occur. We need to look at the remaining allocation of the C-clusters and the R-clusters from the partition D. Each of the C-clusters requires $2k_{n-1} - k_n$ elements from partition D while each of the R-clusters requires $k_{n-1} - (n - 1 - (\rho - k_n))$ elements. The size of the partition is $k_{n-1} \times (n - 1 - k_{n-1})$.

The partition D can again be divided into five distinct partition as shown in Figure B.2. The partition D2 is reserved for allocation to the corresponding C-clusters to fully satisfy them. The partition D1 will be allocated to the corresponding R-clusters to fully satisfy them. The partition D3 will be allocated to the corresponding C-clusters to fully satisfy them. The area D4 will remain free as the corresponding C-clusters and the R-clusters are fully satisfied by the partitions D1 and D3. The partition D5 will be allocated to R-clusters. The

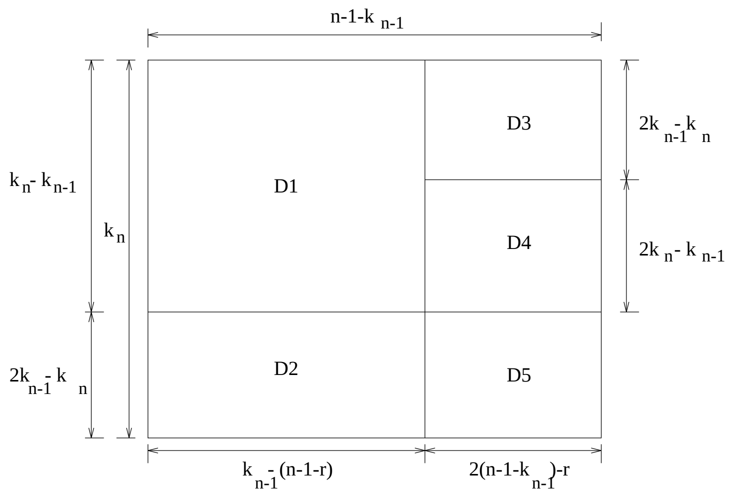


Figure B.2: Allocation pattern in partition D ignoring step-2(b) in the algorithm.

---

corresponding $2k_{n-1} - k_n$ R-clusters will be fully satisfied only if $k_{n-1} - (n - 1 - (\rho - k_n)) \leq 2(n - 1 - k_{n-1}) - (\rho - k_n)$. Otherwise, the algorithm will fail to satisfy these R-clusters.

The choking of the last $2k_{n-1} - k_n$ R-clusters may occur due to D4 remaining free. The elements in D4 can not be allocated to these R-clusters. However, the required number of elements can be obtained in D2 by freeing them from the C-clusters. The C-clusters can the be allocated elements from D1 which are freed from R-clusters corresponding to the partition which in turn can be allocated from the partition D4 which was entirely free. The step-2(b) in the algorithm perform this very task. The resulting allocation pattern is shown in Figure B.3. The step-2(b) checks if there is any free element in the row that has been allocated. If there is, it checks if there is any C-cluster yet to be satisfied. If so, it considers the first unsatisfied C-cluster from right and takes the element in the corresponding column in the row, frees it from the R-cluster to allocate to the C-cluster. The first free element (from left) is allocated to the R-cluster. This is repeated for each free element in the row if there are C-clusters remaining to be satisfied. Thus an element in a row is left free only if no more C-cluster remains to be satisfied. Assuming that there is an element is left free, then all the elements to its right will also be left free. Similarly, all the elements below will also be left free as all the R-clusters require the same number of elements from the partition D. Thus, no R-cluster will finally remain unsatisfied if there are free elements left. This means that some R-clusters may remain unsatisfied only if there is no free elements left. But this can not happen because $(n - 1)k_n \geq \rho k_{n-1} + 1$. That is, there are sufficient number of elements to satisfy all the ρ clusters. Hence, the algorithm satisfies all the ρ clusters.

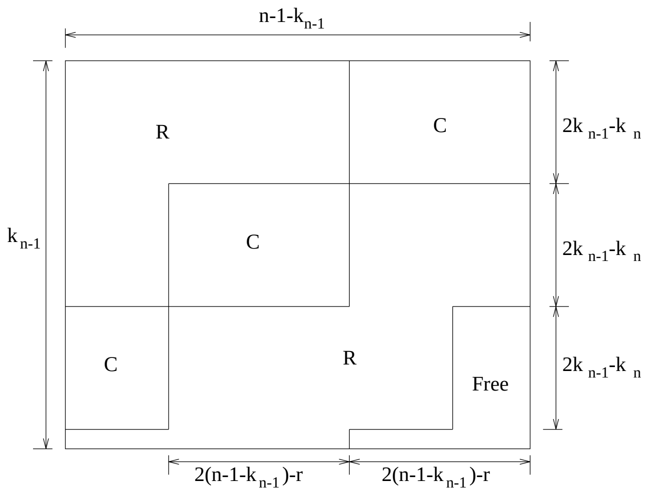


Figure B.3: Allocation pattern in partition D considering step-2(b) of the algorithm.

Now consider the correctness of the algorithm with the modification to make it produce only a distance-2 set. The modified algorithm preallocates second row key elements to the column clusters corresponding to the partition B. The result is shifting of allocation to the row cluster corresponding to the second row from the partition A to partition C. The algorithm also preallocates the diagonal elements to the row clusters. This preallocation compensates for the preallocation of the second row key elements to the column clusters as far as the partition B is concerned and therefore does not effect the correctness of the algorithm. However, there could be diagonal elements in the partition D too. This is equivalent to shifting of some of the allocations to row clusters to the D partition in advance. This only eases the chocking of the R-clusters. Satisfying of the column clusters is still ensured as the algorithm allocates any of the rest of the elements (non-preallocated elements) only if sufficient number of elements remain in a column for the column cluster.

$\sqcup\!\!\!\!\sqcap$

Bibliography

[AHK87] S.B. Akers, D. Harel, and B. Krishnamurthy. The Star Graph : An attractive alternative to the n-cube. In *Proc. Intl. Conf. on Parallel Processing*, pages 333–340, 1987.

[AK87] S.B. Akers and B. Krishnamurthy. The Fault Tolerance of the Star Graphs. In *Proceedings of the second Intl. Conf. on Supercomputing, Vol-III, SanFrancisco*, pages 270–276, May 1987.

[AK89] S.B. Akers and B. Krishnamurthy. A group theoretic model for symmetric interconnection networks. *IEEE Trans. on Computers*, C-38(4):555–563, 1989.

[AQ93] S.G. Akl and K. Qiu. A novel routing scheme on the star and pancake networks and its applications. *Parallel Computing*, 19:95–101, 1993.

[AQS91] S.G. Akl, K. Qiu, and Stojmenovic. Data Communication and Computational Geometry on Star and Pancake Interconnection Networks. In *Proc. IEEE Symposium on Parallel and Distributed Processing*, 1991.

[AQS93] S.G. Akl, K. Qiu, and Stojmenovic. Fundamental Algorithms for the Star and Pancake Interconnection Networks with Applications to Computational Geometry. *NETWORK*, 23:215–225, 1993.

[Ata88] M.J. Atallah. On multidimensional arrays of processors. *IEEE Trans. on Computers*, 37(10):1306–1309, October 1988.

[BA84] L.N. Bhuyan and D.P. Agrawal. General Hypercube and Hyperbus structures for a computer network. *IEEE Trans. on Computers*, 33:323–333, 1984.

[BDL95] N. Bagherzadeh, M. Dowd, and S. Latifi. A Well-Beheaved Enumeration of Star Graphs. *IEEE Trans. on Parallel and Dist. Systems*, 6(5):531–535, May 1995.

[BFP92] P. Berthome, A Ferreira, and S. Perennes. Decomposing Hierarchical Cayley Graphs, with Applications to Information Dissemination and Algorithm Design.

Technical report, TR-92-38, Laboratoire de l'Informatique du Parallelisme, Ecole Normale Superieure De Lyon, July 1992.

[BHOS94] A. Bouabdallah, M.C. Heydemann, J. Opatrny, and D. Sotteau. Embedding Complete Binary Trees into Star and Pancake Graphs. In *Proc. of the Conf. on Mathematical Foundation of Computer Sc., Lect. Notes in Computer Sc. 841*, pages 266–275, 1994.

[BNL93] N. Bagherzadeh, N. Nassif, and S. Latifi. A Routing and Broadcasting Scheme on Faulty Star Graphs. *IEEE Trans. on Computers*, C-42(11), November 1993.

[CC87] M. S. Chen and K.G. Chin. Processor Allocation in an N-cube Multiprocessor Using Gray Codes. *IEEE Trans. on Computers*, C-36(12):1396–1407, December 1987.

[Ce86] V. Cantoni and S.Levialdi (eds). *Pyramidal Systems for Image Processing, and Computer Vision*. Springer Verlag, 1986.

[DD95] N.J. Dimopoulos and V.V. Dimakopoulos. Optimal and Suboptimal Processor Allocation for Hypercycle-based Multiprocessors. *IEEE Trans. on Parallel and Dist. Systems*, 6(2), February 1995.

[DDP90a] S.K. Das, N. Deo, and S. Prasad. Parallel graph algorithms for hypercube computers. *Parallel Computing*, 13:143–158, 1990.

[DDP90b] S.K. Das, N. Deo, and S. Prasad. Two minimum spanning forest algorithms on fixed-size hypercube computers. *Parallel Computing*, 15:179–187, 1990.

[Deo74] Narsing Deo. *Graph Theory: with Application to Engineering and Computer Science*. Prentice Hall Inc., 1974.

[DGD92] S.K. Das, J. Ghosh, and N. Deo. Steirling Networks: A Versatile Combinatorial Topology for Multiprocessor Systems. *Discrete Applied Math*, 38, 1992.

[DH91] S. Dutt and J.P. Hayes. Processor Allocation in an N-cube Multiprocessor Using Gray Codes. *IEEE Trans. on Computers*, 40(3):341–352, March 1991.

[DMS91] M. Dietzfelbinger, S. Mahavapeddy, and I.H. Sudhborough. Three Disjoint Path Paradigms in Star Networks. In *Proc. IEEE Symposium on Parallel and Distributed Processing*, pages 400–406, December 1991.

[DS87] W.J. Dally and C.L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. on Computers*, 36:547–553, May 1987.

[DS89] W.J. Dally and C.L. Seitz. Virtual-channel flow control. *IEEE Trans. on Computers*, 36, 1989.

[DT92] K. Day and A. Tripathi. Arrangement graphs : A class of generalized star graphs. *Information Processing Letters*, 42:235–241, 1992.

[DT93a] K. Day and A. Tripathi. A comparative study of topological properties of hypercubes and star graph. *Journal of Parallel and Distributed Computing*, 1993.

[DT93b] K. Day and A. Tripathi. Unidirectional star graphs. *Information Processing Letters*, 45:123–129, 1993.

[FA93a] P. Fragopolou and S.G. Akl. Edge-disjoint Spanning Trees on the Star Network with Application to Fault Tolerance. Technical report, TR-93-354, Dept. of Computing and Info' Sc., Queen's University,Kingston, Canada, 1993.

[FA93b] P. Fragopolou and S.G. Akl. Optimal Communication Algorithms on Star Graphs using Spanning Tree Constructions. Technical report, TR-93-346, Dept. of Computing and Info' Sc., Queen's University,Kingston, Canada, 1993.

[FA94] P. Fragopolou and S.G. Akl. A Parallel Algorithm for Computing Fourier Transform on the Star Graph. *IEEE Trans. on Parallel and Dist. Systems*, 5(5):525–531, May 1994.

[Fen81] T.Y. Feng. A Survey of Interconnection Networks. *Computer*, 14(12):12–27, December 1981.

[HB94] D.A. Hoelzeman and S. Bettayeb. On the Genus of Star Graphs. *IEEE Trans. on Computers*, C-43(6), June 1994.

[HOS95] M.C. Heydemann, J. Opatrny, and D. Sotteau. Embeddings of complete binary trees into Star Graphs with congestion-1. In *Proc. of the Twenty-Eighth Hawaii Intl. Conf. on Systems Sciences*, volume 2, pages 546–554, 1995.

[Hsu93] W.J. Hsu. Fibonacci Cubes - A New Interconnection Topology. *IEEE Trans. on Parallel and Dist. Systems*, 4(1):3–12, January 1993.

[JH89] S.L. Johnsson and C.T. Ho. Optimum Broadcasting and personalized communication in Hypercube. *IEEE Trans. on Computers*, 38(9):1249–1268, 1989.

[JLD90] J. Jwo, S. Lakshmivarahan, and S.K. Dhall. Characterization of Node Disjoint (Parallel) Paths in Star Graphs, preprint, 1990. Manuscript, Parallel Processing Institute, School of Elect. Eng. and Computer Sc., Univ. of OKlahama.

[JLD91] J. Jwo, S. Lakshmivarahan, and S.K. Dhall. Embedding of cycles and grids in star graphs. *Journal of Circuits, systems and computers*, 1(1):43–74, 1991.

[JM94] Z. Jovanović and J. Misić. Fault tolerance of the star graph interconnection network. *Information Processing Letters*, 49, February 1994.

[Kat89] H.P. Katseff. Incomplete Hypercubes. *IEEE Trans. on Computers*, 37(5):604–608, May 1989.

[KGGK93] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms.* The Benjamin-Cummings, Redwood City, CA, 1993.

[Lat93a] S. Latifi. Clustered-star graph: a new topology for large interconnection networks. In *Proc. of Seventh Intl. Parallel Processing Symposium*, pages 514–518, 1993.

[Lat93b] S. Latifi. On the fault-diameter of the star graph. *Information Processing Letters*, 46:143–150, 1993.

[Lat94] S. Latifi. Task allocation in star graph. *IEEE Trans. on Parallel and Dist. Systems*, 5(11):1220–1224, May 1994.

[LB94] S. Latifi and N. Bagherzadeh. Incomplete Star Graph : An incrementally scallable network based on the star graph. *IEEE Trans. on Parallel and Dist. Systems*, 5(1), January 1994.

[LBT94] C. Liang, S. Bhattacharyya, and J. Tan. Performance evaluation of fault-tolerant routing on star graphs. In *Proc. of the Scalable High-performance Computing Conference*, pages 650–657, 1994.

[Lei85] C.E. Leiserson. Fat-trees: Universal networks for hardware efficient supercomputing. In *Proc. Intl. Conf. on Parallel Processing*, pages 393–402, 1985.

[Lei92] F.T. Leighton. *Introduction to Parallel Algorithm and Architecture : Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, San Mateo, California, 1992.

[LJD93] S. Lakshmivarahan, J. Jwo, and S.K. Dhall. Symmetry in interconnection networks based on Cayley graph of permutation groups: A survey. *Parallel Computing*, 19:361–407, 1993.

[LY91] K.Y. Lee and H. Yoon. Indirect Star-Type Networks for Large Multiprocessor Systems. *IEEE Trans. on Computers*, C-40(11), November 1991.

[MJ94a] J. Misić and Z. Jovanović. Communication Aspects of the Star Graph Interconnection Networks. *IEEE Trans. on Parallel and Dist. Systems*, 5(7), July 1994.

[MJ94b] J. Misić and Z. Jovanović. Routing Function and Deadlock Avoidance in a Star Graph. *Journal of Parallel and Distributed Computing*, pages 216–228, August 1994.

[MPS94] Z. Miller, D. Pritikin, and I.H. Sudhborough. Near Embeddings of Hypercubes into Cayley graphs on the symmetric group. *IEEE Trans. on Computers*, 43(1), January 1994.

[MS90] A. Menn and A.K. Somani. An efficient sorting algorithm for the star graph interconnection network. In *Proc. Intl. Conf. on Parallel Processing*, volume III, pages 1–8, 1990.

[MS92] V.E. Mendia and D. Sarkar. Optimal Broadcasting on the Star Graph. *IEEE Trans. on Parallel and Dist. Systems*, 3(4), July 1992.

[NSK90] M. Nigam, S. Sahni, and B. Krishnamurthy. Embedding Hamiltonians and hypercubes in star graphs. In *Proc. Intl. Conf. on Parallel Processing*, pages 340–343, August 1990.

[PV81] F.P. Preparata and J. Vuillemin. The cube-connected cycle: A versatile network for parallel computation. *Comm. ACM*, 24(5):300–309, 1981.

[QA] K. Qiu and S.G. Akl. Load Balancing, Selection, and Sorting on the Star and Pancake Interconnection Networks. *Journal of Parallel Algorithms and Applications.*

[QA91] K. Qiu and S.G. Akl. Properties of Breadth First Spanning Trees of the Star Graph, 1991. Manuscript, Dept. of Computing ond Info', Sc., Queen's University.

[QMA91] K. Qiu, H. Meijer, and S.G. Akl. Decomposing a star graph into disjoint cycles. *Information Processing Letters*, 39(3):125–129, 1991.

[QMA93] K. Qiu, H. Meijer, and S.G. Akl. On the Cycle Structure of Star Graphs. In *Proc. of the 24th Southwestern Conf. on Combinatorics, Graph theory and Computing, Boca Raton, FL*, 1993.

[RG94] C.P. Ravikumar and A.M. Goel. Deadlock-free Wormhole Routing Algorithm for Star Graph Topology. In *Proc. of Fourth National Seminar on Theoretical Computer Science, Kanpur*, pages 1–19, June 1994.

[RM94] C.P. Ravikumar and G. Maniraman. Star-Graph based multistage interconnection network for ATM switch fabric. In *Proc. IEEE Symposium on Parallel and Distributed Processing*, 1994.

[RS93] Y. Rouskov and P.K. Srimani. Fault diameter of the star graph. *Information Processing Letters*, 48:243–251, 1993.

[RTS95] S. Rai, J.L. Trahan, and T. Smailus. Processor Allocation in Hypercube Multiprocessors. *IEEE Trans. on Parallel and Dist. Systems*, 6(6), June 1995.

[RW93a] S. Rajasekaran and D.S.L. Wei. Selection, Routing and Sorting on the Star Graph, 1993. Presented in the Int'l Parallel Processing Symposium, also available as Tech. Report, Dept. of CIS, University of Pennsylvania, Philadelphia.

[RW93b] S. Rajasekaran and D.S.L. Wei. Selection, Routing and Sorting on the Star Graph, 1993. Manuscript.

[RWY93] S. Ranka, J. Wang, and N. Yeh. Embedding meshes on the star graph. *Journal of Parallel and Distributed Computing*, 19:131–135, 1993.

[SHC+93] X. Shen, Q. Hu, B. Cong, I.H. Sudhborough, M. Girou, and S. Bettayeb. The 4-star graph is not a subgraph of any Hypercube. *Information Processing Letters*, 43:199–203, 1993.

[SLC93] J. Sheu, W. Liaw, and T. Chen. A broadcasting algorithm in the star interconnection networks. *Information Processing Letters*, 48:237–241, 1993.

[SP89] M.R. Samantha and D.K. Pradhan. The de Bruijn Multiprocessor Network: A Versatile Parallel Processing and Sorting Network for VLSI. *IEEE Trans. on Computers*, 38(4):567–581, April 1989.

[SS87] Y. Saad and M.H. Schultz. Topological Properties of Hypercubes. *Journal of Parallel and Distributed Computing*, 1987.

[SS89a] Y. Saad and M.H. Schultz. Data communication in Hypercube. *Journal of Parallel and Distributed Computing*, 6:115–135, 1989.

[SS89b] F. Springsteel and I. Stojmenovic. Parallel General Prefix Computations with Geometric, Algrabic and other Applications. *Intl. Journal of Parallel Programming*, 18(6):485–503, 1989.

[SS91] S. Sur and P.K. Srimani. A Fault Tolerant Routing Algorithm in Star Graphs. In *Proc. Intl. Conf. on Parallel Processing*, volume III, pages 267–270, August 1991.

[SS93a] S. Sur and P.K. Srimani. Super Star : A New Optimally Fault Tolerant Network Architecture, 1993. Manuscript.

[SS93b] S. Sur and P.K. Srimani. Topological Properties of the Star Graphs. *Computers and Mathematics with Applications*, 24(25):87–98, June 1993.

[SV94] O. Sykora and I. Vrto. On VLSI layout of the star graph and related networks. *Integration, The VLSI journal*, 17(1):83–93, August 1994.

[SWC95] J. Sheu, C. Wu, and T. Chen. An Optimal Broadcasting Algorithm without Message Redundancy in Star Graphs. *IEEE Trans. on Parallel and Dist. Systems*, 6(6):653–658, June 1995.

[Te80] S.L. Tanimoto and A. Klinger (eds). *Structured Computer Vision: Machine Perception through Hierarchical Computation Structures.* Academic Press, New York, 1980.

[WY91] H. Wang and Q. Yang. Prime cube graph approach to processor allocation. In *Proc. Intl. Conf. on Parallel Processing*, pages I–25–I–32, August 1991.

[ZY84] C. Nian Zhang and D.Y.Y. Yun. Multi-dimentional systolic networks for discrete fourier transform. In *Proc. 11th Ann. Intl. Synp. Comput. Arch.*, pages 215–222, 1984.