Properties and Algorithms of the (n, k)-Star Graphs

Liang He

Computer Science

Submitted in partial fulfillment

of the requirements for the degree of

Master of Science

Faculty of Mathematics and Science, Brock University

St. Catharines, Ontario

Approved for the Committee:

Dr. K. Qiu

Dr. M. Winter

Dr. S. Houghten

(Computer Science)

Supervisor Dr. K. Qiu

Acknowledgments

I am deeply indebted to my supervisor, Dr. Qiu, who read my numerous revisions and giving me valuable comments. I thank him for his general and financial support and his patience over the past two years. Particularly he always gives me directions for searching a field which I am really interested in.

I thank the members of my supervisory committee (Dr. K. Qiu, Dr. M. Winter and Dr. S. Houghten) for their comments, criticisms and suggestions.

Also I would like to thank the Department of Computer Science, Brock University, for giving me the opportunity to work in a very interesting area and providing me with the financial means to complete this thesis.

Finally, many thanks to my parents, BingCheng He and ChunJie Zhang for their love and support throughout my life. I also wish to thank my wife, Ke Zhao for being a good listener. Her support and understanding helped me to finish this thesis more easily. Thank you very much and I love you all.

This document was prepared using TeXnicCenter.

Co-Authorship

Some preliminary results were reported in the following paper:

1. "Neighbourhood Broadcasting and Broadcasting on the (n, k)-Star Graph," (with K, Qiu and Z. Z. Shen), *International Conference on Algorithms and Architectures*, Cyprus, June 2008, LNCS Vol. 5022, pp. 70-78.
   (received the Best Paper Award)

Abstract

The (n, k)-star interconnection network was proposed in 1995 as an attractive alternative to the n-star topology in parallel computation. The (n, k)-star has significant advantages over the n-star which itself was proposed as an attractive alternative to the popular hypercube. The major advantage of the (n, k)-star network is its scalability, which makes it more flexible than the n-star as an interconnection network. In this thesis, we will focus on finding graph theoretical properties of the (n, k)-star as well as developing parallel algorithms that run on this network.

The basic topological properties of the (n, k)-star are first studied. These are useful since they can be used to develop efficient algorithms on this network. We then study the (n, k)-star network from algorithmic point of view. Specifically, we will investigate both fundamental and application algorithms for basic communication, prefix computation, and sorting, etc.

A literature review of the state-of-the-art in relation to the (n, k)-star network as well as some open problems in this area are also provided.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

There are several good reasons for us to study parallel computation. The primary one for using a parallel computer to solve a problem is saving much time from that required by a sequential computer; since there are many processors cooperating simultaneously on a parallel machine. So far, with the development of computer hardware, a number of commercial parallel machines have been designed and built. There are two important aspects of parallel computation, namely, the parallel computational models and the parallel algorithms.

One cannot talk about the parallel computation without mentioning the associated ***computational model*** on which parallel algorithms are designed. There are a wide range of models that have been designed and used for parallel computation. They can be divided into two major classes: *shared-memory machines* and *interconnection networks*. The difference between these two models is in the way the processors communicate among themselves, whether through a shared memory or an interconnection network. Different interconnection topologies have been proposed, such as trees, meshes, hypercubes, etc.

In this chapter, the shared-memory parallel machines are introduced first; followed by an introduction of the interconnection networks and some typical topologies. We

then present the topical network: the (n, k)-star. Next, we list the criteria to evaluate a parallel algorithm. The last section of this chapter gives an overview and organization of this thesis.

1.1 Shared-Memory Parallel Machines

Before talking about the shared-memory computers, we first introduce a basic classification of computer architectures, proposed by Michael J. Flynn [17]. The four classifications are based upon the number of concurrent instructions (or controls) and data streams available in the architecture: seen by the processor during program execution. Depending on whether there is one or several of these streams, computers can be divided in four classes:

- Single Instruction, Single Data Stream (SISD)

  A SISD computer is a general sequential machine. There is no parallelism in either the instruction or data streams in this class of computers.

- Multiple Instruction, Single Data Stream (MISD)

  In computing, a MISD computer contains N processors, each has its own control unit and all processors share a common memory unit. In this kind of computers, parallelism is achieved by using many functional units to perform different operations on the same data. But in practice, there is no known implementation of this class of computers so far.

- Single Instruction, Multiple Data Stream (SIMD)

  A SIMD computer consists of N identical processors, each with its own local memory to store data. All processors work under the control of a single instruction stream issued by a central control unit. The processors operate synchronously: at each step, all processors execute the same instruction on a

different data element. SIMD computers are much more versatile than MISD computers.

- Multiple Instruction, Multiple Data Stream (MIMD)

  In a MIMD computer, multiple autonomous processors simultaneously executing different instructions on different data, where each processor has its own control unit and local memory. Therefore, processors are potentially all executing different programs on different data while solving different sub-problems of a single problem. This makes MIMD computers more powerful than other three classes of computers.

To solve any non-trivial problems on a parallel computer, processors need to communicate with each other. This can be achieved by either through a shared memory or an interconnection network.

The class of shared-memory parallel computers is also known as the Parallel Random Access Machine (PRAM), as shown in Fig. 1.1. It consists of a number of identical processors $P_1, P_2, ..., P_n$ and a common memory which is shared by these n processors.

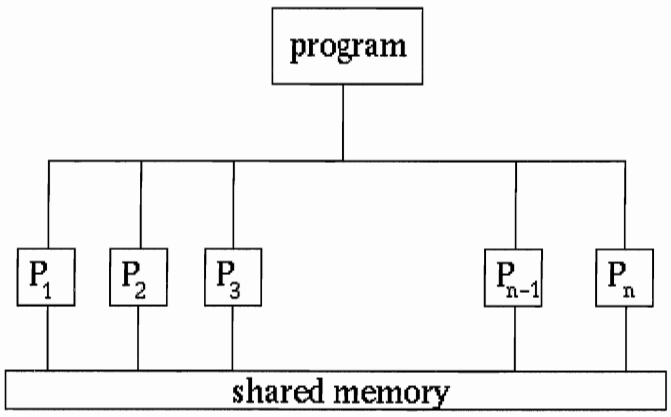

Figure 1.1: PRAM

Theoretically, all processors take the same time to access the memory (read and write). The repertoire of instructions of a synchronous PRAM can result in simultaneous access by multiple processors to the same location in the shared memory. Based on whether multiple processors accessing the same memory location simultaneously is permitted or not, there are four different possibilities to classify PRAM computers.

- **Exclusive Read, Exclusive Write (EREW):**

  In this form of parallel memory model, only one processor can read from any one memory location at a time and only one processor can write to any one memory location at a time. In other words, when this instruction is executed, n processors can simultaneously read from or write to n distinct memory locations.

- **Exclusive Read, Concurrent Write (ERCW):**

  This class of PRAM computers has the ability that only one processor can read from a memory cell but multiple processors can write to a memory cell at one time. But in practice, we do not consider this kind of memory access.

- **Concurrent Read, Exclusive Write (CREW):**

  In this form of computers, multiple processors can read a memory cell but only one can write at a time.

- **Concurrent Read, Concurrent Write (CRCW):**

  The CRCW PRAM computers allow multiple processors either to read from or write to the same memory location at the same time. When CW instruction occurs, one question is "what happens when several processors attempt to write different contents to the same memory cell? " There are several extensions ready to be used with CW in order to resolve this conflict. Some typical further divisions of CW list as follows [3]:

- **Priority CW**: only the processor with highest priority is allowed to write into the position.
- **Common CW**: only allowed if they are attempting to write the same value; otherwise, it is an illegal operation.
- **Random CW**: the processor that succeeds in writing is chosen by a random process.

In the next section, another communication mode among processors in parallel computing system which is through interconnection networks is reviewed.

1.2 Interconnection Network

In parallel computation, comparing with PRAM we mentioned earlier, another way to communicate among processors is through interconnection networks. In an interconnection network, there is no longer a shared memory; instead, each processor has its own local memory and connects with other processors via direct links between them. The links are *two-way* communication lines; in other words, two processors connected by a link can exchange data simultaneously. Therefore, mathematically, an undirected graph $G = (V, E)$ can be used to describe an interconnection network, where each processor P_i is a vertex in V and if there is a link between two processors P_i and P_j in the interconnection network, then an edge $(P_i, P_j) \in E$ exists between the two responding vertices in the graph. In this thesis, we will use the terms "interconnection network" and "graph" interchangeably.

Before introducing some existing interconnection networks, a number of criteria need to be described first. They are important in the sense that they can be used to determine the performance of a network.

Two processors directly connected by a link are said to be ***neighbours***.

Definition 1. *The* ***degree*** *of a processor is the number of neighbours of this processor. The* ***degree*** *of network topology is the maximum of all processors' degrees in the network.*

Definition 2. *The* ***distance*** *between two processors P_m and P_n is the number of links on the shortest path from P_m to P_n; then the* ***diameter*** *of network is the maximum distance among any two arbitrary processors.*

An efficient interconnection topology would usually require small diameter and low degree.

Definition 3. *The* ***connectivity*** *of a graph G is the smallest number of vertices we can delete in order to disconnect G.*

A ***regular*** graph means that all nodes in this graph have the same degree.

Definition 4. *A graph G is* ***vertex symmetric*** *if and only if for any arbitrary vertices v and w, there exists an automorphism of the graph that maps v to w.*

Definition 5. *A graph G is* ***f-fault tolerant*** *whenever f or less than f nodes are deleted from G, the remaining graph is still connected. The* ***fault tolerance*** *of the graph G is said to be the largest value of f for which it is f-fault tolerant.*

The symmetric and fault tolerance properties of a graph are very important when talking about interconnection networks. They are the basic considerations when defining and building the commercial parallel interconnection network machines.

Another aspect we need to understand about interconnection networks is the port model. Each processor in the networks may be viewed as a RAM; additionally, each processor has a number of special registers (called ***ports***) that allow it to communicate with its neighbours. In a ***single-port*** model, in each unit of time a processor is only allowed to send data to or receive data from one of its neighbours. In ***all-port*** model, the processor can communicate with one or more of its neighbours simultaneously.

In the following subsections, we will introduce some typical networks. Suppose all the networks have n processors.

1.2.1 Complete Network

The most obvious and general network topology is a graph where each node in the graph is directly connected to all other $n-1$ nodes in the graph. Such network is called *complete network (clique)*, or K_n. This is the most powerful network; the *degree* of K_n is $n-1$ and the *diameter* is 1. K_6 is shown in Figure 1.2.

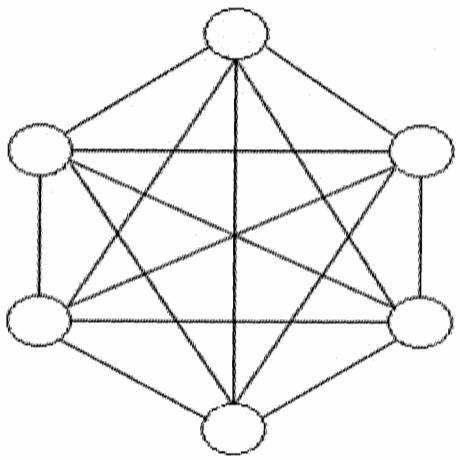

Figure 1.2: Complete Network with 6 Processors (K_6)

1.2.2 Linear Array

The *linear array* is the simplest and most fundamental topology in the interconnection networks. In this network, all n processors form an one-dimensional array. Each processor P_i $(1 < i < n)$ is connected with two neighbours which are P_{i-1} and P_{i+1}. Exceptions are the two end processors, namely, P_1 and P_n, which has only one neighbour. Obviously, the degree of the linear array is 2 and diameter is $O(n)$.

A special case of linear array is the ***ring*** network, which connect P_1 and P_n to each other. Hence, all processors in a ring network have two neighbours.

1.2.3 Mesh

Mesh is a two-dimensional network which is obtained by arranging the processors into an $r \times s$ array. The processor in row i and column j is denoted by $P_{i,j}$, where $1 \leq i \leq r$ and $1 \leq j \leq s$. The neighbours of $P_{i,j}$ will be $P_{i-1,j}, P_{i+1,j}, P_{i,j-1}$ and $P_{i,j+1}$ if they exist. Processors on the boundary rows and columns have less than four neighbours. The degree of mesh is 4 and diameter is $O(r+s)$, since the distance from $P_{1,1}$ to $P_{r,s}$ is $r-1+s-1 = r+s-2$. Figure 1.3 shows a 4×4 mesh.

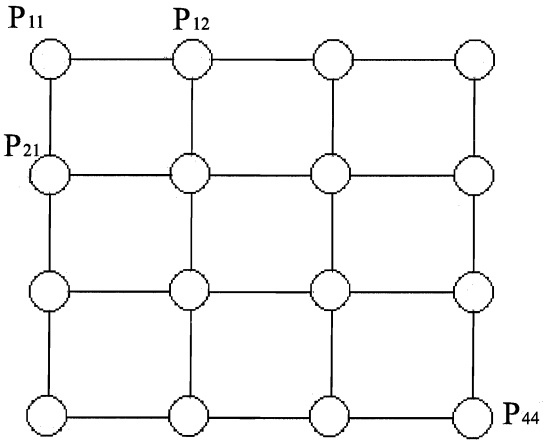

Figure 1.3: A 4×4 Mesh

In a mesh model, the dimension d of the array can also be higher than 2, such

network is called *d-dimensional mesh*. In a d-dimensional mesh, each processor is connected to two neighbours in each dimension, except the boundary processors which have fewer neighbours. Therefore, the degree of d-dimensional mesh will be $2 \times d$.

1.2.4 Perfect Shuffle

In this model, let n be a power of 2 and label n processors as $P_0, P_1, ..., P_{n-1}$. In the *perfect shuffle* topology, a one-way direct link connects P_i to P_j where

$$j = \begin{cases} 2i & 0 \leq i \leq n/2 - 1 \\ 2i + 1 - n & n/2 \leq i \leq n - 1 \end{cases}$$

Also, we can use binary representation to explain the structure in a perfect shuffle network. The binary representation of j is obtained by cyclically shifting original node i one position to the left; meaning that processor $P_{i_{n-1}i_{n-2}...i_0}$ is connected to $P_{i_{n-2}i_{n-3}...i_0i_{n-1}}$ by a shuffle line. For example, when $n = 8$, there is an one-way link from processor P_{001} to P_{010}. A variation of perfect shuffle is *shuffle-exchange* network. In this model, we switch one-way links to two-way connections. In addition, we add *exchange* links to the network, which are two-way lines connecting all even-numbered processors to their successors. Figure 1.4 shows a shuffle-exchange network with 8 processors; the shuffle edges are solid, and the exchange edges are dashed.

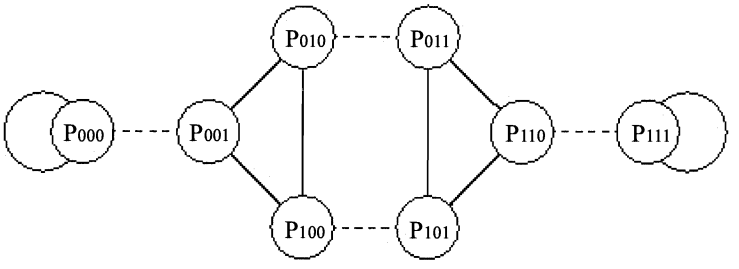

Figure 1.4: A Shuffle-Exchange with 8 Nodes

The degree of shuffle-exchange network is 3 and diameter is $O(\log n)$.

1.2.5 Hypercube

Let $n = 2^d$ for some $d \geq 0$, and we label all n processors P_i where $0 \leq i \leq n-1$ by using the binary representation of i. For each processor P_i, there exists exactly d neighbours. The neighbours of P_i are those processors P_j in which the binary representation of the indices i and j differ in exactly one bit. Here, d is also called the *dimension* of the network; thereby this network is called *d-dimensional hypercube* or *d-cube* as well. Clearly, the degree and diameter of d-cube is the dimension d. Figure 1.5 shows the hypercube networks with dimensions $1, 2$ and 3.

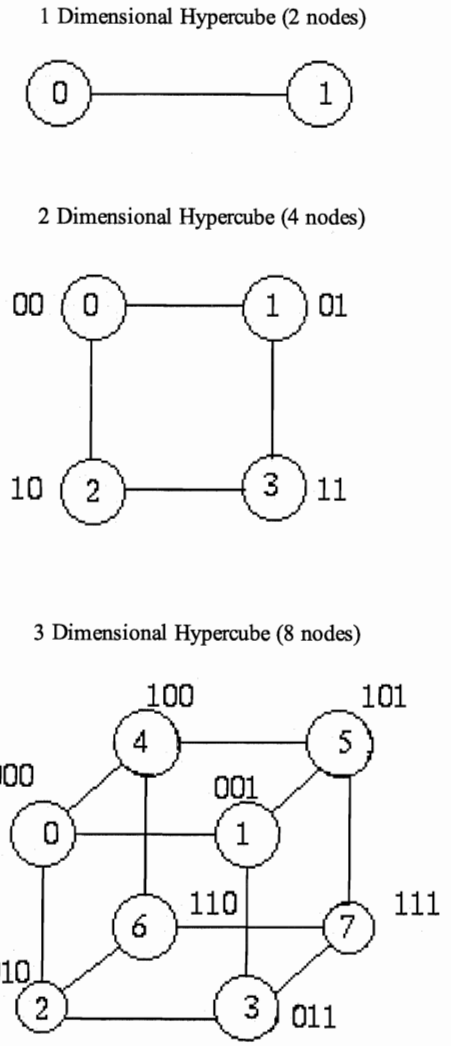

Figure 1.5: Hypercube Networks with $d = 1, d = 2, d = 3$

The *Cube-Connected Cycles* network is a variation of the hypercube. A Cube-Connected Cycles topology is similar to the d-cube, except that each of its 2^d corners is replaced with a ring of d processors. Each processor in a ring is connected to only one processor in a neighbouring ring with the same dimension. More details can be found in [32].

1.2.6 The Star

To obtain a *star* network, we begin with giving an integer n and the integer set $\{1, 2, ..., n\}$. Each processor corresponds to a distinct permutation of these n symbols. Therefore, the total number of nodes in this star network is $n!$. A processor p is connected to $n-1$ neighbours which can be obtained by interchanging the first symbol of p with the i^{th} symbol, $2 \leq i \leq n$. We call these $n-1$ connections *dimensions*. We denote this network by S_n or *n-star*. For example, if $n = 4$, and processor p_{1234} is connected with p_{2134}, p_{3214} and p_{4231} by two-way links.

The following figure shows S_3, S_4.

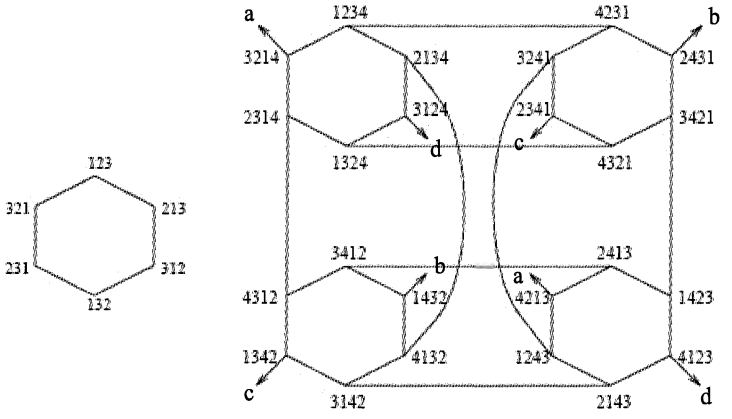

Figure 1.6: The 3-Star S_3 and 4-Star S_4.

The star interconnection network is an attractive alternative to the hypercube parallel model [1]. The most typical properties in the star can compare favorably with the hypercube, including symmetry properties, fault tolerance, etc. [2, 14, 31, 42]. For example, S_n is maximally fault-tolerant. Also, S_n is a regular graph with $n!$ nodes, but its degree is $n-1$ and the diameter is $O(n)$, i.e., sub-logarithmic in the number of vertices, while a hypercube with $O(n!)$ vertices has a degree and diameter of $O(n\ log\ n)$, i.e., logarithmic in the number of vertices.

There are also some other interconnection networks that have been proposed, like the ***Mesh of Trees*** [4], the ***Pyramid*** [29], the ***De Bruijin*** network [39], etc. Along with computer hardware improvement and continuing research, better networks are constantly being proposed and new parallel interconnection network computers being built.

Currently, the most popular non-trivial network in use is the hypercube. The reason is that a hypercube network with dimension d has a large number of processors ($n = 2^d$) and a small degree ($d = log\ n$). Also, the hypercube network structure can be recursively decomposed into successive two lower dimension hypercube. Other features of hypercube include small diameter, vertex symmetric and regular graph property, simple and optimal routing algorithm and good fault tolerance.

1.3 The (n, k)-Star Interconnection Networks

In spite of all the advantages of an n-star over the hypercube, a major drawback is its lack of scalability. As we all know, an n-star network has $n!$ number of processors, resulting in a large gap between n-star and $(n+1)$-star which has $(n+1)!$ processors.

Therefore, we may face the choice of either too many or too few available processors when solving a particular problem by using the star network. For the very popular hypercube topology, a similar problem exists since an n-cube contains 2^n nodes while an $(n+1)$-cube has 2^{n+1} nodes. The *Incomplete hypercube* was proposed by H.P. Katseff [23] in order to get over the above problem. To achieve scalability, incomplete star has also been designed [25, 38]. To overcome this restriction on n-star graph, Chiang and Chen [9] introduced another generalized version of the n-star by giving another parameter k to control the number of nodes in the topology. This new topology is called ***(n, k)-star***.

Now we can give the exact definition of (n, k)-star:

Definition 6. *An (n,k)-star graph, denoted by $S_{n,k}$, is specified by two integers n and k, where $1 \leq k < n$. The node set of $S_{n,k}$ is the set of all k-permutations of n, denoted by $< V > = \{\ p_1p_2...p_k \mid p_i \in < 1,2,...,n > \text{ and } p_i \neq p_j \text{ for } i \neq j\ \}$. The neighbours of a node $p = p_1p_2...p_i...p_k$ are defined as follows:*

1. *$p_ip_2...p_1...p_k$ through an edge of dimension i, where $2 \leq i \leq k$ (swap p_1 and p_i), this kind of edges are referred to i-**edges**.*

2. *$xp_2...p_i...p_k$ through an edge of dimension 1, where $x \in < 1,2,...,n >$ - $\{\ p_i \mid 1 \leq i \leq k\ \}$, this kind of edges are referred to 1-**edges**.*

The following Figure 1.7 shows the $(4,2)$-star or $S_{4,2}$. In this figure, the dashed lines indicate *i-edges* (there is only one choice which is $i = 2$ in $S_{4,2}$) and all other solid lines indicate 1-*edges* of the $(4,2)$-star graph.

The number of nodes in the (n,k)-star is $n!/(n-k)!$. In addition, when $k = n-1$, such $S_{n,n-1}$ is isomorphic to the n-star (S_n) graph [9]. This implies that n-star is a special case of the (n,k)-star graph. Obviously, the (n,k)-star graph allows

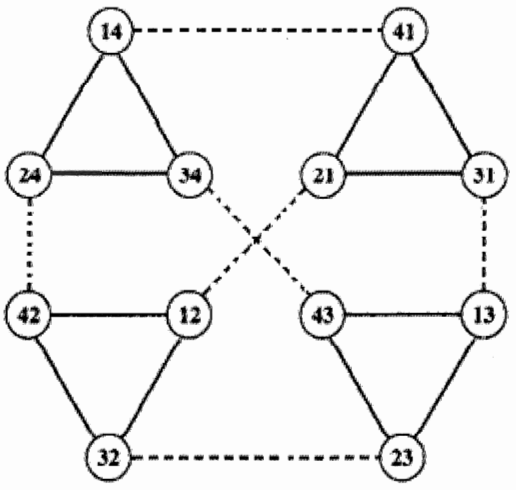

Figure 1.7: A $(4, 2)$-Star Graph

more flexibility than n-star when designing the interconnection network in parallel computation.

1.4 Evaluating Parallel Algorithms

To analyze an existing parallel algorithm, which is running on a given parallel machine (shared memory or interconnection network), we could consider a number of criteria. The most important three criteria are: **running time, the number of processors in the model** and **cost** [3].

The *running time* of a parallel algorithm is defined as the time taken by this algorithm to solve a problem on a parallel computer. Specifically, we are interested in the *worst* time, which means the time required by solving the most difficult instance of the problem using this algorithm. Usually, we count how many *elementary* steps are performed by an algorithm when solving a problem (worst case) as a measure of running time. Talking about parallel algorithm, there are two different kinds of elementary steps, which are:

1. *Computational steps*: A computational step is an arithmetic or logic operation performed on a datum within a processor, like adding two numbers.

2. *Routing steps*: A routing step takes place when a datum of constant size is transmitted from one processor to another processor via shared memory or interconnection network.

Each step (computational or routing) takes a constant number of time units, and the running time of a parallel algorithm is a function of the size of input. For a problem of size N, we use $t(N)$ to denote the worst case number of time units required by a parallel algorithm.

The *number of processors* used by a parallel algorithm is another important criterion to evaluate the performance of this algorithm. The major reason for us to focus on this factor is that fewer processors (less expensive) model is preferred when two different numbers of processors of parallel model can solve a problem with the same running time. Also, sometimes a minimum number of processors is required to guarantee the success of a parallel computation. We normally use $p(N)$ to denote the number of processors used by a parallel algorithm to solve a problem of size N. A special case is when $p(N)$ is constant, which is independent of N.

Another evaluation criterion to be considered is the *cost* of parallel algorithm, which is defined as the product of its running time and the number of processors and denoted as $c(N) = t(N) \times p(N)$. The cost of a parallel algorithm is an upper bound on the total number of elementary steps executed. If a lower bound is known as $\Omega(f(N))$ for a problem of size N, and the cost of parallel algorithm for the same problem matches the lower bound ($c(N) = \mathrm{O}(f(N))$), then this parallel algorithm is said to be *cost optimal*.

The measurement of these three criteria shows us the basic ways to evaluate the goodness of a parallel algorithm. Hence, measuring them together is referred as *algorithm analysis* in parallel computation.

1.5 Organization of the Thesis

Many interconnection networks have been discussed previously and some new ones are continuously being proposed. Since the (n, k)-star is an alternative to the n-star graph which has received much attention and is widely studied, we will investigate the (n, k)-star network, from both the graph theoretical and the algorithmic points of view.

We will discuss the following topics in future chapters:

1. a literature review of $S_{n,k}$.
2. decomposition of $S_{n,k}$ into vertex-disjoint paths or cycles.
3. presenting an optimal neighbourhood broadcasting algorithm for $S_{n,k}$, and using it to develop an optimal broadcasting algorithm in single-port model.
4. discovering a minimum dominating set of $S_{n,k}$, and using it to find a simple broadcasting algorithm on all-port $S_{n,k}$.
5. other basic algorithms for $S_{n,k}$:
   (a) prefix sums computation
   (b) sorting and merging

The remainder of the thesis is divided into six chapters. In Chapter 2, we present a literature review that includes currently available results on the (n, k)-star network. Chapter 3 discusses the graph theoretical properties of the network. The optimal

algorithms of $S_{n,k}$ for some of the most fundamental data communication problems are developed in Chapter 4. Next, in Chapter 5, we discuss some application algorithms that are designed to run on the (n, k)-star graph. The final chapter concludes the thesis and list future open problems and research directions in this network.

Chapter 2

Literature Review of the (n, k)-Star

2.1 Introduction

The (n, k)-star network has received much attention after it was first proposed in 1995. In this chapter, we offer a literature review on (n, k)-star interconnection network. We will review from two areas: existing topological properties and the parallel algorithms. All necessary terms and notations will be defined accordingly.

2.2 Properties

Proposition 1. *$S_{n,k}$ is regular of degree $n-1$[10].*

Definition 7. *Let $S_{n-1,k-1}(i)$ be a subgraph of $S_{n,k}$ induced by all the vertices with the same last symbol i , for some $1 \leq i \leq n$.*

From Definition 6, we can easily see that $S_{n-1,k-1}(i)$ is a $(n-1, k-1)$-star graph which is defined on symbols $\{1, 2, ..., n\} - \{i\}$. In other words, each subgraph $S_{n-1,k-1}(i)$ is isomorphic to $S_{n-1,k-1}$. From this property, $S_{n,k}$ can be decomposed into n $S_{n-1,k-1}$'s: $S_{n-1,k-1}(i)$, $1 \leq i \leq n$ [9]. For example, $S_{4,2}$ in Figure 2.1 contains four $(3, 1)$-stars, namely $S_{3,1}(1)$, $S_{3,1}(2)$, $S_{3,1}(3)$ and $S_{3,1}(4)$, by fixing the last symbol

at 1, 2, 3 and 4, respectively.

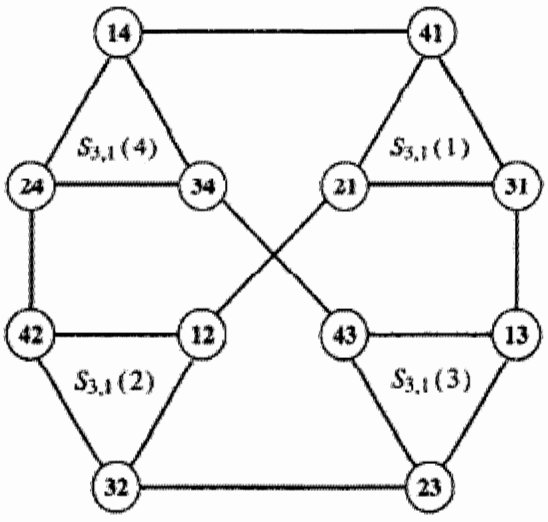

Figure 2.1: A Decomposition of $(4,2)$-Star Graph into 4 $(3,1)$-Star

This hierarchical structure of $S_{n,k}$ is one of the most important properties of the (n,k)-star graph. We are going to exploit this property in our various algorithms, for example, in our broadcasting algorithms for both single-port and all-port $S_{n,k}$ which will be presented later. Here, it should be noted that the dimension in which we fix the symbols to get $S_{n-1,k-1}$'s does not have to be the last in Definition 7, it could be any i, $2 \leq i \leq k$. Thus, in general, we can define $S^i_{n-1,k-1}(j)$ to be a $S_{n-1,k-1}$ such that all the vertices in it have the same symbol j at dimension i, $2 \leq i \leq k$ and $1 \leq j \leq n$. Formalizing above result, we have a new proposition which states:

Proposition 2. *There are $k-1$ different ways to decompose a $S_{n,k}$ into n node-joint $S_{n-1,k-1}$'s: $S^i_{n-1,k-1}(j)$, for $2 \leq i \leq k$ and $1 \leq j \leq n$ [10].*

Unless otherwise specified, in this thesis we are going to decompose the (n,k)-star at the last dimension.

Let i be any symbol from set $\{1, 2, ..., n\}$. We use the notation $i*$ to represent a permutation whose first symbol is i. Similarly, $*i$ represents a permutation whose last symbol is i.

Recall from Definition 4, a graph G is *vertex symmetric* if given any two arbitrary vertices v and w, there exists an automorphism of the graph G that maps v to w, which means the graph G viewed from any vertex looks the same. A vertex symmetric graph allows for all the processors to be identical.

Proposition 3. *The (n, k)-star graph is vertex symmetric [9].*

Proof: We need to show that for any two given vertices a and b in (n, k)-star graph, there is an automorphism of the graph that maps a to b. Suppose $a = a_1a_2...a_k$ and $b = b_1b_2...b_k$, then $< A > = \{a_1, a_2, \cdots, a_k\}$ and $< B > = \{b_1, b_2, \cdots, b_k\}$ will be two subsets of $< n >= \{1, 2, \cdots, n\}$. So we define an one-to-one onto mapping function F in $S_{n,k}$:

$F(p) = f(p_1)f(p_2)...f(p_k)$ for all $p = p_1p_2...p_k$ in $S_{n,k}$,

where f is the one-to-one onto mapping function for symbol x in $< n >$:

- if $x = a_i \in < A >$ for $1 \leq i \leq k$, then $f(x) = b_i \in < B >$.
- if $x \in < B > - < A >$, then $f(x) = y$ where one-to-one mapping for $y \in < A > - < B >$. Since $|A| = |B|$, then $|A - B| = |B - A|$.
- if $x \in < n > - < A > \cup < B >$, then $f(x) = x$.

Clearly, this function F maps a to b. Furthermore, this transformation is an automorphism of the graph. This is due to the fact that if two vertices p and q are connected, then the images of p and q, $F(p)$ and $F(q)$ are also connected by an edge. More precisely, $p = p_1p_2...p_i...p_k$ then $q = p_ip_2...p_1...p_k$ (i-neighbour) or $q = rp_2...p_i...p_k$, where $r \in < n > - \{ p_i \mid 1 \leq i \leq k\}$ (1-neighbour).

So,

$$F(p) = f(p_1)f(p_2)...f(p_i)...f(p_k)$$

and

$$\begin{aligned} F(q) &= f(p_i)f(p_2)...f(p_1)...f(p_k) \text{ or} \\ &= f(r)f(p_2)...f(p_i)...f(p_k) \end{aligned}$$

Then $F(q)$ is a neighbour of $F(p)$. ■

Similar to the definition of vertex symmetry, *edge-symmetry* means there is an automorphism of the graph that maps any two arbitrary edges. Since (n,k)-star graph has two different types of edges (1-edges and i-edges), this prevents it from being edge-symmetric. However, in $S_{n,k}$, same type of edges still have the edge-symmetric property. The following proposition gives the exact statement [10].

Proposition 4. *In $S_{n,k}$ graph,*

1. *every 1-edge is edge-symmetric with any other 1-edge.*

2. *every i-edge is edge-symmetric with any other i-edge .*

As a result of the node symmetric property, any node in $S_{n,k}$ can be mapped to the identity node $e_k = 12\cdots k$; which implies that routing between two arbitrary nodes reduces to routing from an arbitrary node to identity node e_k. Similar to the cyclic representation for a permutation of symbols $1, 2, \cdots, n$, we can represent a k-permutation (a node in $S_{n,k}$) by a product of cycles. To derive this cycle representation, we need to define *external* cycles for symbols $\in < n > - < k >$. For each external symbol p_{m_i} ($\notin < k >$) in node p we construct an external cycle

$C_i = (p_1, p_2, ..., p_{m_i})$ such that the desired position of p_j in p is held by p_{j+1} for $1 \leq j \leq m_{i-1}$, where all p_j, $1 \leq j \leq m_{i-1}$, are internal symbols ($\in < k >$). In addition, for each external cycle we define the *desired symbol* d whose desired position is held by the first element of the cycle. The rest of the cycles, called *internal* cycles, are defined the traditional way. Therefore, the routing from an arbitrary node p to the identity e_k in the (n,k)-star graph can be achieved by moving internal symbols and exchanging the external symbols with desired symbols [9]. We can correct the cycles of node p one by one in $S_{n,k}$ to demonstrate the routing scheme.

We now give an example to get a path from arbitrary node p to the identity node e_k. Suppose $p = 6472583$ in $S_{9,7}$, due to above strategy, we construct the external cycle $(6,8)$ with desired symbol $d = 1$ and internal cycles $(2,4)$, $(3,7)$. First we correct external cycle by using desired symbol 1:

$$6472583 \longrightarrow_6 8472563 \longrightarrow_1 1472563$$

Then to correct along two internal cycles:

$$\begin{aligned} 1472563 \ &\rightarrow_2 \ 4172563 \rightarrow_4 2174563 \rightarrow_2 1274563 \\ &\rightarrow_3 \ 7214563 \rightarrow_7 3214567 \rightarrow_3 1234567. \end{aligned}$$

It is easy to see that the diameter of $S_{n,k}$ is $O(k)$. More precisely, we have:

Proposition 5. *The diameter $D(S_{n,k})$ of (n,k)-star graph is [10]:*

$$D(S_{n,k}) = \begin{cases} 2k-1 & \text{if } 1 \leq k \leq \lfloor \frac{n}{2} \rfloor \\ k + \lfloor \frac{n-1}{2} \rfloor & \text{if } \lfloor \frac{n}{2} \rfloor + 1 \leq k < n \end{cases}$$

Proof: From the cycle structure and routing scheme mentioned above, the distance

between any node p to identity node e_k in $S_{n,k}$ is [9]:

$$Dis(p) = \begin{cases} c + m + x & \text{if } p_1 = 1 \\ c + m + x - 2 & \text{if } p_1 \neq 1 \end{cases} \tag{2.1}$$

where c is the number of cycles (including both external and internal) of length larger than or equal to two of node p, m is the total number of misplaced symbols of p with respect to the identity node e_k, and x is the number of external symbols in p.

Recall from Definition 2, the diameter $D(S_{n,k}) = max\{Dis(p) \mid p \in S_{n,k}\}$. We verify this function by different cases as follows.

1. if $1 \leq k \leq \lfloor \frac{n}{2} \rfloor$, then (1) $p_1 = 1, c = 1, m = k-1, x = k-1$ and (2) $p_1 \neq 1, c = 1, m = k, x = k$. So the Equation 2.1 becomes:

$$Dis(p) = \begin{cases} 1 + k - 1 + k - 1 = 2k - 1 & \text{if } p_1 = 1 \\ 1 + k + k - 2 = 2k - 1 & \text{if } p_1 \neq 1 \end{cases}$$

   Therefore, $D(S_{n,k}) = 2k - 1$.

2. if $\lfloor \frac{n}{2} \rfloor + 1 \leq k \leq n - 1$, we need to consider either n is odd or even. For n is odd number, the maximum value of $Dis(p)$ occurs only when $p_1 = 1$, then $c = 1 + (2k - n - 1)/2$, $m = k - 1$ and $x = n - k$. So,

$$Dis(p) = 1 + (2k - n - 1)/2 + k - 1 + n - k = k + (\frac{n-1}{2})$$

   For n is even number, the Equation 2.1 of $Dis(p)$ will be calculated as follows:

$$Dis(p) = \begin{cases} \underbrace{\frac{2k-n}{2}}_{c} + \underbrace{k-1}_{m} + \underbrace{n-k}_{x} = k + \frac{n}{2} - 1 & \text{if } p_1 = 1 \\ \underbrace{\frac{2k-n}{2} + 1}_{c} + \underbrace{k}_{m} + \underbrace{n-k}_{x} = k + \frac{n}{2} - 1 & \text{if } p_1 \neq 1 \end{cases}$$

Therefore, $D(S_{n,k}) = k + \lfloor \frac{n-1}{2} \rfloor$.

■

This result also provides us with another easy way to find the diameter of the n-star graph, since S_n graph is a special case of $S_{n,k}$ where $k = n - 1$. According to Proposition 5, when $k = n - 1$, the second case should apply, which is $D(S_{n,n-1}) = n - 1 + \lfloor \frac{n-1}{2} \rfloor = \lfloor \frac{3(n-1)}{2} \rfloor$.

A *Hamiltonian cycle* in a graph is a cycle that includes all the vertices of the graph exactly once. If a graph has a Hamiltonian cycle, we call such a graph *Hamiltonian.*

Proposition 6. *$S_{n,k}$ with $n \geq 3$ is Hamiltonian [20].*

Recently, more research results about the (n, k)-Star network have been discussed by some research people. For example, the fault tolerance and connectivity properties of $S_{n,k}$ are discussed in [9, 10, 20, 21] and the ring embedding property is introduced in [7].

2.3 Algorithms

2.3.1 Neighbourhood Broadcasting

Recall that in an interconnection network, communications among processors are accomplished by sending data along the interconnection links. Two possible existing communication modes of nodes are *single-port* and *all-port.* In a single-port (weak) model, a processor can send (receive) at most one fixed length datum to (from) only one of its neighbours in one time unit; on the other hand, in one time unit, a processor can send (receive) one datum of fixed length to (from) all its neighbours in the all-port (strong) model. The *neighbourhood broadcasting problem*, **NBP** for short, was

first introduced by Cosnard and Ferreira in [13] which only applies to the single-port model. More precisely, NBP is a problem in which a datum of fixed size is sent from the source node to all its neighbours in the single-port communication model. In other words, the NBP is the problem of *simulating* a single step of the all-port communication model.

For some interconnection networks with constant node degrees, obviously the time required for neighbourhood broadcasting is constant. To minimize this constant in different networks is the main problem that we are interested within this area. The lower bound of this NBP on a network with degree d is $\Omega(\log d)$, as shown below:

Theorem 1. *Any neighbourhood broadcasting algorithm on a network with degree d must require $\Omega(\log d)$ time.*

Proof: At each time unit, one processor with the messages can only send to one of its neighbours, so after every step, the number of neighbours which have received the information can at most double. The maximum number of neighbours of a node is d, so the least time to solve NBP must be $\Omega(\log d)$. ■

For example, for S_n, the best running time we can obtain is $\log(n-1)$. When talking about $S_{n,k}$, the above theorem tells us that the lower bound for NBP in $S_{n,k}$ is also $\Omega(\log n)$, since the degree of $S_{n,k}$ is $n-1$. We will discuss this later.

Previously, this problem has been studied for topologies like linear array, trees, cycles, mesh and tori in [15, 16]. Moreover, the neighbourhood broadcasting problem for hypercube [6], the star and pancake graphs [18, 30, 33, 35, 36], and a special family of Cayley graphs [24], have been discussed recently. All these algorithms are asymptotically optimal for the corresponding interconnection networks.

2.3.2 Broadcasting

Comparing with neighbourhood broadcasting, the *broadcasting problem* (**BP**) means one vertex wishes to send a message of constant size to all the vertices in the network. For a single-port model, the BP has a lower bound of $\Omega(\log N)$, where N is the total number of vertices in the network.

Theorem 2. *Any broadcasting algorithm on a single-port graph with N nodes must require time $\Omega(\log N)$ [1].*

Proof: Note that after each time unit the number of processors that have received the information being broadcast can at most double. ■

Since S_n has $n!$ nodes in the graph, the optimal algorithm for the broadcasting problem on n-star needs $O(\log(n!)) = O(n \log n)$ time. In the last few years, some broadcasting algorithms for S_n have been found [1, 19, 27]. The idea of these schemes can be described as follows. Since S_n can be decomposed as n number of S_{n-1}'s, in $O(\log n)$ time, the source node will send information to one node in each of $S_{n-1}(i)$, where $1 \leq i \leq n$. Now every $S_{n-1}(i)$ has a node with the information, it recursively carries out the algorithm on each $S_{n-1}(i)$.

Recently, the problem of broadcasting has been studied for (n, k)-star [8, 26] where $O(nk)$ time algorithms are obtained. The lower bound for this broadcasting problem on single-port $S_{n,k}$ graph is $\Omega(\log(n!/(n-k)!)) = \Omega(k \log n)$.

Talking about the broadcasting problem on interconnection networks of the all-port model, in addition to the time (the number of communication steps) required, one of the considerations of the algorithm is the *traffic*, i.e., the total number of messages exchanged [41]. This means that it is desirable to minimize both the time and traffic [41]. To minimize the traffic is equivalent to minimizing the redundancy, i.e., the

number of times a node receives the same message. This broadcasting problem has been considered before and algorithms whose running times are proportional to the diameter of the (n, k)-star have been obtained using spanning trees [26].

2.3.3 Prefix Sums

The *prefix sums* problem is a computation on a list in which each element in the result list is obtained from the sum of the elements in the original list up to its index. In parallel computation, we are given n elements a_i, $i = 1, 2, \cdots, n$, each stored in one processors P_i, $i = 1, 2, \cdots, n$ in a network, and a closed associative[1] binary operation $\oplus$, the prefix sums problem is to compute all the quantities

$$s_i = a_1 \oplus a_2 \oplus ... \oplus a_i, \ i = 1, 2, ..., n \tag{2.2}$$

At the end of the computation we want P_i to contain s_i. Since the binary operation is the usual addition operation, the word "sum" is used as a generic term for prefix computations.

With the development of parallel computing, the *prefix sums* computation has gained considerable attention in the literature and it plays a central role in parallel algorithm design. Also, prefix sums problem is illustrated using a host of examples from a variety of application areas. For example, counting sorting [12] and broadcasting could both be solved using the idea of prefix sums computation.

We will develop an optimal algorithm for the prefix sums problems on $S_{n,k}$ later.

2.3.4 Sorting

Given a sequence of elements stored in a set of ordered processors, with each processor holding one element, the sorting problem requires us to sort these numbers in non-

[1]The operation o is *associative* if $(a \ o \ b) \ o \ c = a \ o \ (b \ o \ c) = a \ o \ b \ o \ c$

decreasing order. Therefore, to properly define the sorting problem in $S_{n,k}$ graph, it is indispensable to define an ordering of the processors in the network. This way, we can say that the sorting algorithm will put the smallest element in the first processor, the second smallest element in the second processor and so on.

One way to define the order of processors in (n,k)-star is so called *reverse lexicographic order*. Since the lexicographic order[2] of permutations can be easily built, if we list all permutations backward, then we have the reverse lexicographic order of processors in (n,k)-star. For example, in $S_{4,2}$, the reverse lexicographic order of processors will be

$$43 \prec 42 \prec 41 \prec 34 \prec 32 \prec 31 \prec 24 \prec 23 \prec 21 \prec 14 \prec 13 \prec 12.$$

When sorting a sequence of elements in an interconnection network, we define the *F (Forward)* direction if for any two elements x and y held by processors p and q, respectively, $p \prec q$ implies that $x \leq y$. The *R (Reverse)* direction is defined similarly. The Sorting problem for S_n has been studied in [5, 28]. In [5], an $O(n^2)$ time sorting algorithm is given based on the $(n-1)$-dimensional lattice. The algorithm in [28] is based on *Shear Sort* which was introduced in [40] for a mesh-connected parallel computer, since a n-star can be considered as $n \times (n-1)!$ array in a row-major order. We will use this idea to find a sorting algorithm on a (n,k)-star graph.

[2]For example, the permutations of $\{1,2,3\}$ in lexicographic order are $123, 132, 213, 231, 312$, and 321

Chapter 3

Properties of the (n, k)-Star Network

3.1 Introduction

In this chapter, we present some of the properties of the (n, k)-star graph, e.g., decomposing the graph into vertex-disjoint paths and cycles, embedding the mesh into the (n, k)-star, and finding the minimum dominating set of the graph, etc. These properties are very useful in developing efficient parallel algorithms on (n, k)-star network in the next two chapters. For example, the cycle structure is used to develop the neighbourhood broadcasting and broadcasting algorithms in the single-port $S_{n,k}$, and the minimum dominating set provides us a way to broadcast messages for the all-port model.

3.2 Decomposing the (n,k)-Star Graph into Cycles by 1-Edges

Recall from the definition of the (n,k)-star graph, there are two different kinds of edges in $S_{n,k}$: i-edges and 1-edges. Hence, we can define two different kinds of neighbours of a node in $S_{n,k}$.

Definition 8. *In $S_{n,k}$, given a vertex p, then:*

1. *a vertex connected with p through the 1-edge is called a* ***1-neighbour*** *of p;*
2. *a vertex connected with p through the i-edge is called an* ***i-neighbour*** *of p.*

For example, given a node $p = 12\cdots k$ in $S_{n,k}$, then its i-neighbours are

$$21345\cdots k$$
$$32145\cdots k$$
$$42315\cdots k$$
$$\vdots$$
$$k2345\cdots 1$$

and its 1-neighbours are

$$(k+1)234\cdots k$$
$$(k+2)234\cdots k$$
$$\vdots$$
$$n234\cdots k$$

Clearly, every node in $S_{n,k}$ has $k-1$ i-neighbours and $n-k$ 1-neighbours; thereby, the degree of $S_{n,k}$ graph is $n-1$.

Given a node p in $S_{n,k}$, if we treat these two different types of neighbours of p with the node p separately, i.e., grouping p and all its 1-neighbours, then we can easily find the following graphical observation of $S_{n,k}$.

Observation 1. *In $S_{n,k}$, given an arbitrary node p, then there exists a cycle between p and all p's 1-neighbours.*

Due to the symmetry, without loss of generality, we only need to consider the node $p = e_k$, so e_k and its 1-neighbours form a cycle as follows:

$$\begin{array}{c} \mathbf{12345\cdots k} \\ \updownarrow \\ (k+1)2345\cdots k \\ \updownarrow \\ (k+2)2345\cdots k \\ \updownarrow \\ \vdots \\ \updownarrow \\ (n-1)2345\cdots k \\ \updownarrow \\ n2345\cdots k \\ \updownarrow \\ \mathbf{12345\cdots k} \end{array}$$

where "$\updownarrow$" represents a two-way link in $S_{n,k}$.

From the above, we can easily see that every node in this cycle must have the same symbols from the second to the last position; i.e. every node in the example cycle has permutation style like $x2345...k$. The number of nodes in these cycles are all the same, which is $n-k+1$, since there are $n-k$ 1-neighbours of any node in

$S_{n,k}$. Also, we can show that these cycles are disjoint from each other.

Proposition 7. *In $S_{n,k}$, given two nodes which are not connected by an 1-edge, then cycles formed with these two nodes with their 1-neighbours are disjoint from each other.*

Proof: Suppose $u = u_1u_2...u_k$ and $v = v_1v_2...v_k$ are two different nodes in $S_{n,k}$, there is no 1-edge between u and v, such that there exists at least one symbol i, $u_i \neq v_i$, for $2 \leq i \leq k$. Let u and its neighbours form a cycle C_u, v and its neighbours form a cycle C_v. Now we show C_u and C_v are disjoint from each other.

Assume these two different cycles C_u and C_v share one or more node, then let w be a node in both cycles C_u and C_v, i.e., $w \in C_u$ and $w \in C_v$. Then, if $w \in C_u$, that means $w = xu_2u_3...u_k$ which is an 1-neighbour of u. The same idea applies to $w \in C_v$, so w is an 1-neighbour of v, $x = wv_2v_3...v_k$.

$$w = xu_2u_3...u_k = xv_2v_3...v_k$$

$$\Longrightarrow$$

$$u_2u_3...u_k = v_2v_3...v_k$$

which leads to a contradiction, thus such w does not exist.

Therefore, the cycles are disjoint from each other. ■

Henceforth, using this 1-neighbour cycle structure idea of $S_{n,k}$, we can find a new way to decompose the (n,k)-star graph into different vertex-disjoint paths and cycles.

Proposition 8. *$S_{n,k}$ can be decomposed into $\frac{n!}{(n-k+1)!}$ vertex-disjoint cycles of length $n-k+1$.*

Proof: First we need to prove the number of paths (cycles) is $\frac{n!}{(n-k+1)!}$. Since we know every node p has $n-k$ 1-neighbours, the length of each path is $n-k$ (length of each

cycle is $n-k+1$), and all these nodes (including p) have the same permutation symbols from position 2 to k. Therefore, to find the total number of paths (cycles) is the same as finding how many permutations of length $(k-1)$ from the set $< n >= \{1, 2, ..., n\}$. Using combinatorial theory, the total number of choices N is

$$N = \underbrace{n \times (n-1) \times (n-2) \times ... \times (n-k+2)}_{k-1} = \frac{n!}{(n-k+1)!} \tag{3.1}$$

Secondly, from Proposition 7 we know that all these paths (cycles) are disjoint to each other. ■

So, whenever a path with start point $x_1a_2a_3...a_k$, where $x_i \in < 1, 2, ..., n >$ - $\{ a_j \mid 2 \leq j \leq k \}$, for $1 \leq i \leq n-k$, we will assume that $x_1 < x_2 < ... < x_{n-k}$, so x_1 is the minimum of all x_i. These path will go this way: exchange first symbol x_i to x_{i+1}, and so on. For example, if $n = 5$ and $k = 2$, $S_{5,2}$ can be decomposed into five $(5!/(5-2+1)! = 5)$ vertex-disjoint paths of length 4:

$$\begin{aligned}
&path\ 1: \quad 21 \leftrightarrow 31 \leftrightarrow 41 \leftrightarrow 51 \\
&path\ 2: \quad 12 \leftrightarrow 32 \leftrightarrow 42 \leftrightarrow 52 \\
&path\ 3: \quad 13 \leftrightarrow 23 \leftrightarrow 43 \leftrightarrow 53 \\
&path\ 4: \quad 14 \leftrightarrow 24 \leftrightarrow 34 \leftrightarrow 54 \\
&path\ 5: \quad 15 \leftrightarrow 25 \leftrightarrow 35 \leftrightarrow 45
\end{aligned}$$

Recall that the most powerful interconnection network is the complete graph K_n, which is also called a *clique* K_n. In fact, Observation 1 really implies that:

Theorem 3. *In $S_{n,k}$, for any node p, p and all its 1-neighbours form a clique K_{n-k+1}.*

Proof: Given any node $p = p_1p_2...p_k$ and its 1-neighbours set, which is denoted by $< p_{1-neighbours} >$, we need to prove that any two nodes in $< p_{1-neighbours} >$ are connected with each other by a two-way link.

Suppose x and y are two nodes in $< p_{1-neighbours} >$. Let $x = ip_2...p_k$ and $y = jp_2...p_k$, $x \neq y$ implies $i \neq j$. By the definition of the (n,k)-star graph, there is also a 1-edge between x and y, which means every two nodes in $< p_{1-neighbours} >$ are connected to each other.

Hence, p and its 1-neighbours form a clique with $n-k+1$ nodes. ■

Lemma 1. *There are* $\frac{n!}{(n-k+1)!}$ *cliques each with* $n-k+1$ *nodes in* $S_{n,k}$.

Proof: The proof is trivial based on Proposition 8. ■

Specifically, when $k = 1$, we have

Lemma 2. *When* $k = 1$, $S_{n,1}$ *is a clique* K_n *[7].*

Proof: By the definition of (n,k)-star. ■

3.3 Finding the Minimum Dominating Set of the (n,k)-Star Graph

Definition 9. *A dominating set of vertices in a graph* $G = (V,E)$ *is a set* $V' \subseteq V$ *such that every vertex of* G *either belongs to* V' *or has a neighbour in* V'. *The domination number is the number of vertices in* V'. *And the* ***minimum dominating set*** *is a dominating set with the smallest dominating number.*

The dominating set problem is to find a minimum dominating set D_G of a graph G with domination number $|D_G|$. In parallel computation, interconnection networks

are modeled as graphs. When talking about finding a minimum dominating set of $S_{n,k}$, we first give a lower bound of domination number in the following proposition.

Proposition 9. *Let $D_{n,k}$ be a minimum dominating set of $S_{n,k}$, $D_{n,k}$ contains at least $\frac{(n-1)!}{(n-k)!}$ vertices.*

Proof: Since $S_{n,k}$ is a regular graph of degree $n-1$, and each vertex in a minimum dominating set $D_{n,k}$ dominates itself and $n-1$ of its neighbours, then we have:

$$|D_{n,k}| \geq \frac{n!/(n-k)!}{n} = \frac{(n-1)!}{(n-k)!} \tag{3.2}$$

which implies $D_{n,k}$ contains at least $\frac{(n-1)!}{(n-k)!}$ vertices. ■

Henceforth, when considering finding the minimum dominating set of the graph $S_{n,k}$, if we can prove a dominating set D has $\frac{(n-1)!}{(n-k)!}$ vertices, then this set D will be a minimum dominating set of $S_{n,k}$.

In $S_{n,k}$, let D be a set of all the nodes whose first symbol is i, $1 \leq i \leq n$, which is denoted by $i*$. Now we can get the following result:

Theorem 4. *Every vertex set $D = \{i*\}$, for $1 \leq i \leq n$, is a minimum dominating set of $S_{n,k}$.*

Proof: To show correctness of this theorem, we need to prove two results:

1. D is a dominating set of $S_{n,k}$.
2. there are $\frac{(n-1)!}{(n-k)!}$ vertices in D.

First, to show $D = \{i*\}$ is a dominating set means any node in the graph $S_{n,k}$ is adjacent to one node of this form $i*$. Given any node $p = p_1p_2...p_k$ in $S_{n,k}$, we consider three cases:

- if $p_1 = i$, then $p \in D$.
- if $p_j = i$, $2 \leq j \leq k$, we interchange p_j and p_1 to reach a new node $p' = ip_2...p_1...p_k$, which is a neighbour of vertex p. Since $p' \in D$, then p is adjacent to a node in D.
- if $i \in < n > -\{p_j \mid 1 \leq j \leq k\}$, we can replace p_1 by i to get a neighbour of p, such as $p' = ip_2p_3...p_k$. Also, p' is a node in D, so $p' \in D$, then p is adjacent to a node in D.

Therefore, we can see that the set $D = \{i*\}$ is a dominating set of $S_{n,k}$.

Second, the number of nodes of this form $\{i*\}$ is

$$\frac{(n-1)!}{((n-1)-(k-1))!} = \frac{(n-1)!}{(n-k)!}$$

Hence, D is a minimum dominating set of $S_{n,k}$. ■

For example, in Figure 1.7, $S_{4,2}$ has four different minimum dominating sets by choosing different i:

$$i = 1: \quad D_{4,2} = \{12, 13, 14\}$$
$$i = 2: \quad D_{4,2} = \{21, 23, 24\}$$
$$i = 3: \quad D_{4,2} = \{31, 32, 34\}$$
$$i = 4: \quad D_{4,2} = \{41, 42, 43\}$$

The minimum dominating set problem of a graph is very useful, because it can be used in practical applications in data communication in networks. We will use this idea and hierarchical structure of $S_{n,k}$ to develop a broadcasting algorithm for the all-port (n,k)-star interconnection network.

3.4 Embedding the Mesh into the (n, k)-Star Graph

Definition 10. *An embedding of graph $G = (V_G, E_G)$ into $H = (V_H, E_H)$ is an one-to-one function f: $V_G \Rightarrow V_H$. G is called a guest graph and H is called a host graph.*

Embedding a guest (source) graph into a host (target) graph has been long used to model the processor allocation in the parallel and distributed computing environments [7]. The guest graph sometimes represents an existing parallel algorithm and the host graph is an interconnection network where the algorithm executes. Also, it is used to simulate a parallel algorithm of one type of interconnection network on another one.

In considering graph embedding problems, mapping of the edges of guest graph G is accomplished by a simple path of host graph H such that if $e = (a, b) \in E_G$, then there exists a single path from node $f(a)$ to $f(b)$ in H. The *edge dilation* of an edge e is defined by the distance of the single path in H, which is $dist(f(a), f(b))$. The *dilation cost* of function f is defined as $max_{(a,b)\in E_G}(dist(f(a), f(b))$. Another important feature involved in graph embedding is the so called *expansion cost*, which is defined by the ratio of size H to size G, i.e., $|V_H|/|V_G|$.

The problem of embedding a mesh into the star graph has been discussed in [22, 37]. The results shown in [22, 37] mentioned an $(n-1)$-dimensional mesh or 2-dimensional mesh both could embed into a n-star graph with expansion cost 1. The similar ideas also apply to (n, k)-star graph. We will study two mapping strategies in the following sections.

3.4.1 Embedding a k-dimensional Mesh into $S_{n,k}$

In this section, we describe a mapping of a k-dimensional mesh of size $n \times (n-1) \times ... \times (n-k+1)$, denoted by $M_{n,k}$, on the (n, k)-star graph $S_{n,k}$.

Lemma 3. *There is no embedding for $M_{n,k}$ to $S_{n,k}$ with edge dilation equal to 1 if $k > \lfloor \frac{n}{2} \rfloor + 1$.*

Proof: In any dilation 1 embedding the degree of a node in the guest graph G should be less than or equal to the degree of a node in the host graph H. Here, a node in $M_{n,k}$ can have a degree $2k-2$ and the degree of $S_{n,k}$ is $n-1$. When $k > \lfloor \frac{n}{2} \rfloor + 1$, we have $(2k-2) > n-1$, so there is no 1 dilation embedding of $M_{n,k}$ on $S_{n,k}$. ■

Let m be a node in $M_{n,k}$, then node m can be represented as $m = (m_1, m_2, ..., m_i, ..., m_k)$, $1 \leq i \leq k$, for $1 \leq m_i \leq n-i+1$. Hence, the vertex set of $M_{n,k}$ will be:

$$
\begin{aligned}
V(M_{n,k}) = \{ \; & (\overbrace{1,1,...,1}^{k}), (1,1,...,2), ..., (1,1,...,n-k+1), \\
& (1,2,...,1), (1,2,...,2), ..., (1,2,...,n-k+1), \\
& ... \\
& (n,n-1,...,1), (n,n-1,...,2), ..., (n,n-1,...,n-k+1)\}.
\end{aligned}
$$

For example, consider $n = 5, k = 3$, then all vertices in this $5 \times 4 \times 3$ 3-dimensional mesh are listed in Table 3.1.

Table 3.1: List all vertices in 3-dimensional mesh of size $5 \times 4 \times 3$

(1,1,1)	(1,1,2)	(1,1,3)	(1,2,1)	(1,2,2)	(1,2,3)
(1,3,1)	(1,3,2)	(1,3,3)	(1,4,1)	(1,4,2)	(1,4,3)
(2,1,1)	(2,1,2)	(2,1,3)	(2,2,1)	(2,2,2)	(2,2,3)
(2,3,1)	(2,3,2)	(2,3,3)	(2,4,1)	(2,4,2)	(2,4,3)
(3,1,1)	(3,1,2)	(3,1,3)	(3,2,1)	(3,2,2)	(3,2,3)
(3,3,1)	(3,3,2)	(3,3,3)	(3,4,1)	(3,4,2)	(3,4,3)
(4,1,1)	(4,1,2)	(4,1,3)	(4,2,1)	(4,2,2)	(4,2,3)
(4,3,1)	(4,3,2)	(4,3,3)	(4,4,1)	(4,4,2)	(4,4,3)
(5,1,1)	(5,1,2)	(5,1,3)	(5,2,1)	(5,2,2)	(5,2,3)
(5,3,1)	(5,3,2)	(5,3,3)	(5,4,1)	(5,4,2)	(5,4,3)

Clearly, the total number of nodes in $M_{n,k}$ is $n\times(n-1)\times(n-2)\times...\times(n-k+1)$. Since the number of nodes in $S_{n,k}$ is $\frac{n!}{(n-k)!}$, which equals to the number of nodes in $M_{n,k}$. We may consider embedding a k-dimensional mesh of size $n\times(n-1)\times...\times(n-k+1)$ into $S_{n,k}$ has an expansion cost 1.

Before we discuss embedding $M_{n,k}$ into $S_{n,k}$ graph, we give some definitions about ordering nodes in both $M_{n,k}$ and $S_{n,k}$, so called *lexicographic* order, which will be the first step of developing a one-to-one mapping function between these two graphs.

Definition 11. *In a k-dimensional mesh of size $n\times(n-1)\times...\times(n-k+1)$ or $M_{n,k}$, m_x and m_y are two processors associated with the vertices $m_x=(m_{x_1},m_{x_2},...,m_{x_k})$ and $m_y=(m_{y_1},m_{y_2},...,m_{y_k})$. The lexicographic order in $M_{n,k}$ is defined as: $m_x \preceq m_y$ if and only if the first m_{x_i} which is different from m_{y_i} satisfied as $m_{x_i}<m_{y_i}$, $1\le i\le k$.*

Using this definition, we could sort all vertices in Table 3.1 by the lexicographic order to get the following results:

Table 3.2: Lexicographic order of vertices in 3-dimensional mesh of size $5\times4\times3$

	(1,1,1)	$\preceq$	(1,1,2)	$\preceq$	(1,1,3)	$\preceq$	(1,2,1)	$\preceq$	(1,2,2)	$\preceq$	(1,2,3)
$\preceq$	(1,3,1)	$\preceq$	(1,3,2)	$\preceq$	(1,3,3)	$\preceq$	(1,4,1)	$\preceq$	(1,4,2)	$\preceq$	(1,4,3)
$\preceq$	(2,1,1)	$\preceq$	(2,1,2)	$\preceq$	(2,1,3)	$\preceq$	(2,2,1)	$\preceq$	(2,2,2)	$\preceq$	(2,2,3)
$\preceq$	(2,3,1)	$\preceq$	(2,3,2)	$\preceq$	(2,3,3)	$\preceq$	(2,4,1)	$\preceq$	(2,4,2)	$\preceq$	(2,4,3)
$\preceq$	(3,1,1)	$\preceq$	(3,1,2)	$\preceq$	(3,1,3)	$\preceq$	(3,2,1)	$\preceq$	(3,2,2)	$\preceq$	(3,2,3)
$\preceq$	(3,3,1)	$\preceq$	(3,3,2)	$\preceq$	(3,3,3)	$\preceq$	(3,4,1)	$\preceq$	(3,4,2)	$\preceq$	(3,4,3)
$\preceq$	(4,1,1)	$\preceq$	(4,1,2)	$\preceq$	(4,1,3)	$\preceq$	(4,2,1)	$\preceq$	(4,2,2)	$\preceq$	(4,2,3)
$\preceq$	(4,3,1)	$\preceq$	(4,3,2)	$\preceq$	(4,3,3)	$\preceq$	(4,4,1)	$\preceq$	(4,4,2)	$\preceq$	(4,4,3)
$\preceq$	(5,1,1)	$\preceq$	(5,1,2)	$\preceq$	(5,1,3)	$\preceq$	(5,2,1)	$\preceq$	(5,2,2)	$\preceq$	(5,2,3)
$\preceq$	(5,3,1)	$\preceq$	(5,3,2)	$\preceq$	(5,3,3)	$\preceq$	(5,4,1)	$\preceq$	(5,4,2)	$\preceq$	(5,4,3)

Recall the order of processors in the interconnection network, the *rank* of a vertex is defined as follows:

Definition 12. *Given an ordering of processors in an interconnection network, the rank of a processor p, $r(p)$, equals the number of processors that precedes p.*

Now we can describe a method about embedding a k-dimensional mesh $M_{n,k}$ into $S_{n,k}$ graph. After giving the lexicographic order of nodes in both $M_{n,k}$ and $S_{n,k}$ graphs, the mapping function from vertex set $V(M_{n,k})$ to $V(S_{n,k})$ can be found as follows: if a node $m = (m_1, m_2, ..., m_k) \in V(M_{n,k})$ and a node $p = p_1p_2...p_k \in V(S_{n,k})$ have the same rank ($r(m) = r(p)$), then m is mapped to p. Clearly, since the numbers of nodes in both graphs are equal, this mapping function is one-to-one. Table 3.3 describes the mapping between $V(M_{5,3})$ to $V(S_{5,3})$ (order from left to right).

Table 3.3: Mapping of $V(M_{5,3})$ to $V(S_{5,3})$

$M_{5,3}$	$S_{5,3}$	$M_{5,3}$	$S_{5,3}$	$M_{5,3}$	$S_{5,3}$
(1,1,1)	123	(1,1,2)	124	(1,1,3)	125
(1,2,1)	132	(1,2,2)	134	(1,2,3)	135
(1,3,1)	142	(1,3,2)	143	(1,3,3)	145
(1,4,1)	152	(1,4,2)	153	(1,4,3)	154
(2,1,1)	213	(2,1,2)	214	(2,1,3)	215
(2,2,1)	231	(2,2,2)	234	(2,2,3)	235
(2,3,1)	241	(2,3,2)	243	(2,3,3)	245
(2,4,1)	251	(2,4,2)	253	(2,4,3)	254
(3,1,1)	312	(3,1,2)	314	(3,1,3)	315
(3,2,1)	321	(3,2,2)	324	(3,2,3)	325
(3,3,1)	341	(3,3,2)	342	(3,3,3)	345
(3,4,1)	351	(3,4,2)	352	(3,4,3)	354
(4,1,1)	412	(4,1,2)	413	(4,1,3)	415
(4,2,1)	421	(4,2,2)	423	(4,2,3)	425
(4,3,1)	431	(4,3,2)	432	(4,3,3)	435
(4,4,1)	451	(4,4,2)	452	(4,4,3)	453
(5,1,1)	512	(5,1,2)	513	(5,1,3)	514
(5,2,1)	521	(5,2,2)	523	(5,2,3)	524
(5,3,1)	531	(5,3,2)	532	(5,3,3)	534
(5,4,1)	541	(5,4,2)	542	(5,4,3)	543

Next factor we need to consider about this embedding is the edge dilation cost. Assume in this embedding, a node $m = (m_1, m_2, ..., m_i, ..., m_k)$ in the original mesh is mapped to a permutation $p = p_1p_2...p_i...p_k$ in an (n,k)-star graph. Obviously, we want to consider all neighbours of m in $M_{n,k}$, the nodes $(m_1, ..., m_i + 1, ..., m_k)$ and $(m_1, ..., m_i - 1, ..., m_k)$ (if they exist), for $1 \leq i \leq k$. The only change to get a neighbour of node m is to add or subtract 1 from m_i at dimension i and keep all other dimensions the same. To get all these neighbours' corresponding nodes in $S_{n,k}$, we can consider two cases:

Plus(+) Case: if $m^+ = (m_1, ..., m_i + 1, ..., m_k)$ which is a neighbour of m by adding 1 from dimension i, then we need to find the smallest dimension j such as $m_j = m_i$ for $j > i$. If j does not exist, the corresponding node of m^+ is achieved by replacing p_i by x for $x > p_i$, such that $p' = p_1p_2...x...p_k$. If j exists, the corresponding node p' in $S_{n,k}$ is the one by interchanging the symbols p_i and p_j. In other words, if $p = p_1p_2...p_i...p_j...p_k$, then $p' = p_1p_2...p_j...p_i...p_k$.

Minus(−) Case: similar to the plus(+) case above, if $m^- = (m_1, ..., m_i - 1, ..., m_k)$ which is a neighbour of m by subtracting 1 from dimension i, then the corresponding mapped node p' of m^- in $S_{n,k}$ has two permutation forms: either $p' = p_1p_2...x...p_k$ where x is a symbol at dimension i and $x < p_i$ or $p' = p_1p_2...p_i...p_j...p_k$ where $p = p_1p_2...p_j...p_i...p_k$.

From these two cases, we can easily find the key form of corresponding node p' in $S_{n,k}$. Given m and its one neighbour m' in $M_{n,k}$, the mapping nodes are p and p' in $S_{n,k}$. Then the permutations of p and p' differ at most 2 positions, which tells us the distance between p and p' is at most 3. The special case is when m' is the neighbour by changing the first dimension m_1 of m. Then in this case, the mapping node p' will also be a neighbour of p in $S_{n,k}$. Therefore, the lemma about the dilation cost lists as follows.

Lemma 4. *The above embedding has a dilation cost* 3.

Proof: From the above description,

1. Given m and m', the two original connected nodes in the mesh, let the corresponding nodes in $S_{n,k}$ be $p = p_1p_2...p_i...p_k$ and $p' = p_1p_2...x...p_k$. If $i = 1$, these two nodes are connected by an edge in $S_{n,k}$; otherwise, an 3-length path can be built: $p = p_1p_2...p_i...p_k \rightarrow p_ip_2...p_1...p_k \rightarrow xp_2...p_1...p_k \rightarrow p_1p_2...x...p_k = p'$.

2. Given m and m', the two original connected nodes in the mesh, let the corresponding nodes in $S_{n,k}$ be $p = p_1p_2...p_i...p_j...p_k$ and $p' = p_1p_2...p_j...p_i...p_k$. Then the length 3 path from p to p' will be: $p = p_1p_2...p_i...p_j...p_k \rightarrow p_ip_2...p_1...p_j...p_k \rightarrow p_jp_2...p_1...p_i...p_k \rightarrow p_1p_2...p_j...p_i...p_k = p'$.

Hence, every edge in $M_{n,k}$ can be mapped to a 3-length or less path in $S_{n,k}$. The dilation cost of this embedding is 3. ■

As an example, consider $m = (2,2,2)$ in $M_{5,3}$ (corresponding to node 234 in $S_{n,k}$), the neighbours of m are $(1,2,2), (3,2,2), (2,1,2), (2,3,2), (2,2,1)$, and $(2,2,3)$ in the original mesh, and corresponding nodes in $S_{n,k}$ are $134, 324, 214, 243, 231, 235$. And the edges to path mapping are:

$$
\begin{aligned}
((2,2,2)(1,2,2)) &: \quad 234 \rightarrow 134 \\
((2,2,2)(3,2,2)) &: \quad 234 \rightarrow 324 \\
((2,2,2)(2,1,2)) &: \quad 234 \rightarrow 324 \rightarrow 124 \rightarrow 214 \\
((2,2,2)(2,3,2)) &: \quad 234 \rightarrow 324 \rightarrow 423 \rightarrow 243 \\
((2,2,2)(2,2,1)) &: \quad 234 \rightarrow 432 \rightarrow 132 \rightarrow 231 \\
((2,2,2)(2,2,3)) &: \quad 234 \rightarrow 432 \rightarrow 532 \rightarrow 235
\end{aligned}
$$

Combining all discussions above, we get a final theorem about embedding a k-dimensional mesh into the (n,k)-star graph.

Theorem 5. *A k-dimensional mesh of size $n \times (n-1) \times ... \times (n-k+1)$ can be embedded into $S_{n,k}$ graph with dilation cost of 3 and expansion cost of 1.*

For this embedding, since expansion cost is 1, and edge dilation is larger than 1, some nodes are used as intermediate nodes for many different pairs of adjacent nodes in the original mesh, resulting in congestions and delays of communications. A smaller dilation does not necessarily lead to smaller communication cost; actually, the communication cost is not reflected by the dilation cost.

3.4.2 Embedding a 2-dimensional Mesh into $S_{n,k}$

Recall the hierarchical structure $S_{n,k}$ we discussed in Chapter 2, each $S_{n,k}$ graph can be decomposed into n number of $S_{n-1,k-1}(i)$ subgraphs, where $1 \leq i \leq n$. We use this idea to embed a 2-dimensional mesh $n \times \frac{(n-1)!}{(n-k)!}$ into an (n, k)-star graph.

If we arrange all the vertices in $S_{n,k}$ into an $n \times \frac{(n-1)!}{(n-k)!}$ array in the row-major order (in terms of the processor ordering as reverse lexicographic order), the row i becomes $S_{n-1,k-1}(i)$. The example of $S_{4,2}$ is given in the following Table 3.4.

Table 3.4: Vertices of $S_{4,2}$

41	31	21	$S_{3,1}(1)$
42	32	12	$S_{3,1}(2)$
43	23	13	$S_{3,1}(3)$
34	24	14	$S_{3,1}(4)$

From this table we can easily see that all the vertices in the same column have the same rank in their respective $S_{n-1,k-1}$'s. For example, vertices $31, 32, 23$ and 24 are all ranked 2 in $S_{3,1}(1)$,$S_{3,1}(2)$,$S_{3,1}(3)$,$S_{3,1}(4)$, respectively.

If we exchange the 1^{st} symbol with k^{th} symbol in each vertex, we can get another $n \times \frac{(n-1)!}{(n-k)!}$ array. In this way, each column of the new array is connected to form a

simple path, and vertices in each row form a dominating set of $S_{n,k}$. Table 3.5 is the new 4×3 array after switching the first and last symbols of each vertex in Table 3.4. For example, look at column 1, the path $14 \rightarrow 24 \rightarrow 34 \rightarrow 43$ is shown; all vertices $\{41, 31, 21\}$ in row 1 form a minimum dominating set of $S_{4,2}$. Therefore, we may consider that the vertices in row-major order of $S_{n,k}$ ($n \times \frac{(n-1)!}{(n-k)!}$ array), each column as "connected" in a path directly without this constant time transformation.

Table 3.5: Vertices of $S_{4,2}$

14	13	12
24	23	21
34	32	31
43	42	41

These properties allow us to embed an $n \times \frac{(n-1)!}{(n-k)!}$ 2-dimensional mesh into $S_{n,k}$ arranged by this row-order structure. The expansion cost is still 1 since both graphs have the same number of vertices. Since every vertex in the same row belongs to one $S_{n-1,k-1}$, the edge dilation in a row edge of the mesh depends on the diameter of $S_{n-1,k-1}$. This embedding property of $S_{n,k}$ allows us to simulate the Shear Sort algorithm for a mesh-connected parallel computer in the (n, k)-star interconnection network, which will be presented in Chapter 5 later.

Chapter 4

Communication Problems on the (n, k)-Star Network

4.1 Introduction

In this chapter, we consider the neighbourhood broadcasting and broadcasting problems in the (n, k)-star interconnection network. First, we present an optimal neighbourhood broadcasting algorithm for the (n, k)-star under the single-port model. This algorithm will then be used to develop an optimal broadcasting algorithm for this interconnection network. Our neighbourhood broadcasting is the first such algorithm. For the all-port model, we develop an optimal algorithm using the minimum dominating set we found in previous chapter. The time complexity matches the results of previous work but our algorithm is much simpler.

4.2 Broadcasting on the Single-Port Model

As we mentioned in the first chapter, in a single-port (weak) model network, a node can communicate with one and only one of its neighbours in one time unit. This tells us that the broadcasting problem (BP) in such a model has a lower bound $\Omega(\log N)$,

where N is the number of nodes in the network, and neighbourhood broadcasting problem (NBP) has a lower bound $\Omega(\log d)$, where d is the degree of the network. For $S_{n,k}$, the lower bound will be $\Omega(\log(n!/(n-k)!)) = \Omega(k \log n)$ for BP and $\Omega(\log n)$ for NBP.

4.2.1 Neighbourhood Broadcasting on $S_{n,k}$

Since the (n,k)-star graph is vertex symmetric, without loss of generality, we assume that the source node, which wishes to broadcast a piece of information to all of its neighbours, is the identity node $e_k = 12\cdots k$. For this node, its $k-1$ i-neighbours are:

$$21345\cdots k, 32145\cdots k, ..., (k-1)2345\cdots 1k, k2345\cdots 1,$$

and its $n-k$ 1-neighbours are:

$$(k+1)234\cdots k, (k+2)234\cdots k, ..., (n-1)234\cdots k,\ n234\cdots k.$$

From Theorem 3 and Lemma 2 in previous chapter, we know that in $S_{n,k}$ a node p with all its 1-neighbours form a clique; in other words, all 1-neighbours of p are connected with each other. Also, $S_{n,1}$ is a clique K_n. Our neighbourhood broadcasting algorithm is based on these topological properties and the following observations on the (n,k)-star graph:

Observation 2. *For any two i-edge neighbours of p: $i*k = i23\cdots(i-1)1(i+1)\cdots k$ and $j*k = j23\cdots(j-1)1(j+1)\cdots k$ (we assume that $i<j$ without loss of generality), they are on the same cycle of length 6 as follows, where $\Leftrightarrow$ represents a two-way link*

(edge) between two nodes.

$$123\cdots i\cdots j\cdots k \Leftrightarrow$$

$$i23\cdots 1\cdots j\cdots k \Leftrightarrow$$

$$j23\cdots 1\cdots i\cdots k \Leftrightarrow$$

$$123\cdots j\cdots i\cdots k \Leftrightarrow$$

$$i23\cdots j\cdots 1\cdots k \Leftrightarrow$$

$$j23\cdots i\cdots 1\cdots k \Leftrightarrow$$

Note that, this 6-cycle contains the source node as well as two of its i-neighbours $i23\cdots(i-1)1(i+1)\cdots k$ and $j23\cdots(j-1)1(j+1)\cdots k$; and only involves i-edges. For example, in $S_{6,4}$, for $i=2$ and $j=4$, we have a 6-cycle:

$$1234 \Leftrightarrow 2134 \Leftrightarrow 4132 \Leftrightarrow 1432 \Leftrightarrow 2431 \Leftrightarrow 4231 \Leftrightarrow$$

In fact, above observation also holds true when $k+1 \leq j \leq n$.

Observation 3. *For any i-neighbour $i*k = i23\cdots(i-1)1(i+1)\cdots k$ and 1-neighbour $j*k = j23\cdots k$, where $k+1 \leq j \leq n$, they are on the same cycle of length 6 as well as the source node $e_k = 123\cdots k$,*

$$123\cdots(i-1)i(i+1)\cdots k \Leftrightarrow$$

$$i23\cdots(i-1)1(i+1)\cdots k \Leftrightarrow$$

$$j23\cdots(i-1)1(i+1)\cdots k \Leftrightarrow$$

$$123\cdots(i-1)j(i+1)\cdots k \Leftrightarrow$$

$$i23\cdots(i-1)j(i+1)\cdots k \Leftrightarrow$$

$$j23\cdots(i-1)i(i+1)\cdots k \Leftrightarrow$$

This cycle involves both i-neighbour and 1-neighbour. For example, in $S_{6,4}$, for $i = 2$ and $j = 5$, we have a 6-cycle:

$$1234 \Leftrightarrow 2134 \Leftrightarrow 5134 \Leftrightarrow 1534 \Leftrightarrow 2534 \Leftrightarrow 5234 \Leftrightarrow$$

Observation 4. *Any two 6-cycles formed as in the above two observations with distinct $2 \leq i_1, j_1, i_2, j_2 \leq n$ are disjoint except that they share the source node $123 \cdots k$.*

This observation is true because the first symbols of the first cycle are $1, i_1, j_1, 1, i_1$ and j_1 and those of the second cycle are $1, i_2, j_2, 1, i_2$ and j_2.

Note that Observations 2, 3, and 4 allow us to view the source node together with its $n-1$ neighbours as a de facto complete graph in the sense that any two nodes are connected by a path of constant length.

Based on the above observations and the technique of *recursive doubling* where at each step, we double the number of neighbours with the message by using a set of disjoint cycles of constant size in $S_{n,k}$, a simple neighbourhood broadcasting algorithm for $S_{n,k}$ can be designed.

Initially, the source node is the only one with the message. In one step, it sends the message through the direct link to one of its neighbours. Now two nodes have the message and they in turn send the message to two other neighbours of the source node in such a way that the source sends its message to a neighbour in one step and the neighbour who just received the message in previous step sends the message to another neighbour of the source node via a length-4 path that is part of a 6-cycle. The number of nodes with the message is now 4 (the source node and three of its neighbours) and these four nodes send the message to another four neighbours of

the source node in the same fashion. That is, three neighbours of the source node with the message send the message to another three neighbours of the source node by disjoint paths of length four that are parts of three disjoint 6-cycles and the source node sends its message to a neighbour directly. The algorithm continues until all neighbours of the source node receive the message.

One possible implementation is given as follows (assuming that the neighbours of node $12\cdots k$ are ordered such that $213\cdots k$ is the first, $321\cdots k$ as second, etc.).

Algorithm 1 Neighborhood Broadcasting on $S_{n,k}$

$N \leftarrow 1$ // the number of nodes currently with the message
for $i = 0$ to $\lceil \log \frac{n}{2} \rceil$ **do**
  if $1 \leq n - N \leq 3$ **then**
    source node $12\cdots k$ sends the message to the remaining nodes (neighbours) that have not received the message yet by direct links
    // nodes $2^i + j$, $1 \leq j \leq (n - N)$
    stop
  else in parallel
    each node u that has the message sends its message to node $u + 2^i$, if node $u + 2^i$ exists (source node does this through the direct link while others through paths of the form $u* \rightarrow (u + 2^i)* \rightarrow 1* \rightarrow u* \rightarrow (u + 2^i)*$ (a neighbourbour of the source) of length 4 on disjoint cycles)
    $N \leftarrow 2 \times N$
  end if
end for

Essentially, after each step, the number of nodes with the message is doubled (with the possible exception of the last step). For example, $n = 8, k = 4$ and source node $p = 1234$, the neighbourhood broadcasting is done in the following fashion:

- **Step 1:**

  $1234 \rightarrow 2134$ (direct link)

- **Step 2:**

  $1234 \rightarrow 3214$ (direct link)

  $2134 \rightarrow 4132 \rightarrow 1432 \rightarrow 2431 \rightarrow 4231$ (4-length path)

- **Step 3:**

  $1234 \rightarrow 5234$ (direct link)

  $2134 \rightarrow 6123 \rightarrow 1634 \rightarrow 2634 \rightarrow 6234$ (4-length path)

  $3214 \rightarrow 7214 \rightarrow 1274 \rightarrow 3274 \rightarrow 7234$ (4-length path)

  $4231 \rightarrow 8231 \rightarrow 1238 \rightarrow 4238 \rightarrow 8234$ (4-length path)

Another possible implementation is to first do a neighbourhood broadcasting in the $(n-k)$-clique formed by all the 1-edge neighbours of the source node, then start the recursive doubling. Many other implementations are also possible since the source and its neighbours form a de facto complete graph from a practical point of view. This algorithm works correctly since all the paths are disjoint from Observation 4.

Now we need to analyze the running time of this algorithm, we first consider the case where $n \ mod \ 2^{\lfloor \log n \rfloor} > 3$. In this case, $\lceil \log n \rceil$ steps are needed where each step requires routing of length 4 except the very first step where the source sends its message directly to node 2. Thus,

$$\begin{aligned} t(n) &= 4\lceil \log n \rceil - 3 \\ &= 4\lceil \log n \rceil - 4\log 2 + 1 \\ &= 4\left\lceil \log(\frac{n}{2}) \right\rceil + 1 \end{aligned}$$

The analysis for the other case is similar.

Therefore, the running time for the algorithm is

$$t(n) = \begin{cases} 4\lfloor \log(\frac{n}{2}) \rfloor + 1 + x & \text{if } 1 \leq x = n \bmod 2^{\lfloor \log n \rfloor} \leq 3 \\ 4\lceil \log(\frac{n}{2}) \rceil + 1 & \text{otherwise} \end{cases}$$

which is $O(\log n)$. In view of the $\Omega(\log n)$ lower bound, we can say that our algorithm is optimal.

Note that when n is relatively small, it is better for the source node to simply send its message to each of its $n-1$ neighbours one at a time, requiring $n-1$ steps.

4.2.2 Broadcasting on $S_{n,k}$

Using the neighbourhood broadcasting procedure just developed from the previous section, a broadcasting algorithm on (n,k)-star can be done easily.

Once again, without loss of generality, we assume that node $e_k = 123\cdots k$ wants to broadcast a message to all the other processors in $S_{n,k}$. In the first step, the message will be sent to all its neighbours through neighbourhood broadcasting algorithm in $O(\log n)$ time so that nodes $2*k, 3*k, ..., (k-1)*k$ and $k*1$ (all i-neighbours of source node e_k), and $(i+1)*k, (i+2)*k, ..., (n-1)*k$ and $n*k$ (all 1-neighbours of source node e_k) all have the message. Then in one more time unit, we can send the message to another $n-1$ nodes through these neighbours' k-dimension (switch 1^{st} and k^{th} symbols of these neighbours who have message). Now we have $n-1$ nodes $*2, *3, ..., *n$ with the message. Now, every $S_{n-1,k-1}(i)$, $1 \leq i \leq n$, has at least one node with the message. So we can recursively broadcast in each $S_{n-1,k-1}$ in parallel.

Algorithm 2 Broadcasting $(S_{n,k})$

if $k = 1$ **then**
simply perform a standard broadcasting algorithm. $//S_{n,1}$ is a clique of size n
else
the source node $12\cdots k$ performs a neighbourhood broadcasting. $//2*k, 3*k, ..., k*1$ (all i-neighbours) and $(i+1)*k, (i+2)*k, ..., n*k$ (all 1-neighbours) have the message.

all (except node $k*1$) send their message to their k-dimension neighbours. // at least one node of form $*2, *3, ..., *n$ with the message.
in parallel for all $1 \leq i \leq n$ **do**
Broadcasting $(S_{n-1,k-1}(i))$
end for
end if

Let $t(n,k)$ be the running time for broadcasting on the (n,k)-star graph, we know the neighbourhood broadcasting requires $O(\log n)$ time, then $t(n,k)$ is easily seen to be:

$$\begin{aligned}
t(n,k) &= C\log n + t(n-1,k-1) \\
&= C\log n + C\log(n-1) + t(n-2,k-2) \\
&\vdots \\
&= C\log n + C\log(n-1) + \cdots + C\log(n-k+2) + t(n-k+1,1) \\
&= C\log n + C\log(n-1) + \cdots + C\log(n-k+2) + C_1\log(n-k+1) \\
&= O\left(\log\left(\frac{n!}{(n-k)!}\right)\right) \\
&= O(k\ \log n)
\end{aligned}$$

where C and C_1 are two constant numbers. Hence this broadcasting algorithm is optimal in view of the $\Omega(k\log n)$ lower bound.

The key to the broadcasting algorithm is the neighbourhood broadcasting algorithm that first sends the message to $n-1$ neighbours of the source node that are of the forms $2*k, 3*k, ..., (k-1)*k, k*1, (k+1)*k, (k+2)*k, ..., n*k$. a similar idea has been used before in deriving a broadcasting algorithm for the n-star, for example, in [27]. The main difference is that instead of being neighbours of the source node, these $n-1$ nodes are in a binary tree rooted at the source node. It is also worth pointing out that there is a binomial tree rooted at the source node (thus any node due to the vertex symmetry of the star graph) containing nodes of these forms.

While our neighbourhood broadcasting is the first such algorithm designed for the (n,k)-star network, our optimal $O(\log(n!/(n-k)!))$-time broadcasting algorithm improves previous algorithms with $O(nk)$ running time [26].

4.3 Broadcasting on the All-Port Model $S_{n,k}$

Broadcasting on the all-port (n,k)-star network has been considered before and optimal algorithms whose running times are proportional to the diameter of the network have been obtained using spanning trees [26]. In this section, we present another approach to the problem using a minimum dominating set of $S_{n,k}$ to relay the message such that no node receives the same message more than once. Its running time is $O(k)$, thus optimal, and is arguably simpler.

Recall from the previous chapter, the vertex set D containing all the nodes of the form $i*$ is a minimum dominating set of $S_{n,k}$. A simple broadcasting algorithm on the all-port $S_{n,k}$ can now be found based on D as follows:

Algorithm 3 Broadcasting $(S_{n,k})$

if $n = 2$ **then**
  Source node sends the message along dimension 2.
else if $k = 1$ **then**
  Source node sends the message to all its neighbours (all 1-edge neighbours)
else
  Broadcasting $(S_{n-1,k-1}(k))$;

  Each node $*k$ (in $S_{n-1,k-1}(k)$) sends its message to neighbour $k*$ along dimension k //the set of the nodes of the form $k*$ is a minimum dominating set

  Each node in the dominating set sends its message along all dimensions except k.
end if

It is easy to see from the algorithm that each node receives the message exactly once, thus there is no message redundancy. As to the analysis, let $t(n,k)$ be the time required to broadcast in all-port $S_{n,k}$, then we have

$$t(n,k) = \begin{cases} 1 & \text{if } n = 2 \text{ or } k = 1 \\ t(n-1,k-1)+2 & \text{otherwise} \end{cases}$$

Solving this gives us $t(n,k) = 2k = O(k)$.

In this chapter, we developed three algorithms to solve the neighbourhood broadcasting and broadcasting problems on the (n,k)-star network. First two algorithms are applied to single-port model; the last one is for all-port model. All those algorithms' time complexities are optimal in view of the lower bound derived. Besides this, they are simpler to understand and easier to implement.

Chapter 5

Algorithms

5.1 Introduction

In this chapter, we present two basic algorithms that are fundamental for designing parallel algorithms on $S_{n.k}$. The algorithms presented here are prefix sums computation and sorting. To deal with prefix sums computation we will introduce the Group Copy procedure on the (n,k)-star topology. The sorting algorithm is based on the "mesh" embedding property of $S_{n,k}$ and the idea of Shear Sort. To the best of our knowledge, all the algorithms are the first to be proposed for the (n,k)-star interconnection network.

5.2 Prefix Sums Computation

Before we consider the prefix sums computation in $S_{n,k}$, we will first introduce the Group Copy procedure [34] in $S_{n,k}$.

5.2.1 Group Copy on $S_{n,k}$

Consider the following problem: given $S_{n-1,k-1}(i)$ and $S_{n-1,k-1}(j)$, where $i \neq j$, it is required to exchange the information of the processors in $S_{n-1,k-1}(i)$ with the processors in $S_{n-1,k-1}(j)$. Here, the word "exchange" means a bijective function. In other words, the information in each processor of $S_{n-1,k-1}(i)$ ($S_{n-1,k-1}(j)$) is transferred to a processor in $S_{n-1,k-1}(j)$ ($S_{n-1,k-1}(i)$), and no two processors send their contents to the same processor in the other graph. We can name this problem as copying the contents of $S_{n-1,k-1}(i)$ and $S_{n-1,k-1}(j)$ to each other in arbitrary order. The idea of solving this problem in the (n,k)-star graph is sending contents to each processor's neighbour along dimension k, so contents are now saved in all processors of the forms $i*$ and $j*$. Now we can find that given any node $a \in i*$, there exists an edge (a,b), where $b \in j*$, which connects a with one processor of the form $j*$. This is true because for each processor $a = ia_2a_3\cdots a_k \in i*$,

- if $j = a_x$, for $2 \leq x \leq k$, meaning $a = ia_2a_3\cdots a_{x-1}ja_{x+1}\cdots a_k$, then an i-edge $(ia_2a_3\cdots a_{x-1}ja_{x+1}\cdots a_k,\ ja_2a_3\cdots a_{x-1}ia_{x+1}\cdots a_k)$ exists, where $ja_2a_3\cdots a_{x-1}ia_{x+1}\cdots a_k = b \in j*$.

- if $j \neq a_x$, for all $2 \leq x \leq k$, then we have $a = ia_2a_3\cdots a_k \in i*$ and $b = ja_2a_3\cdots a_k \in j*$, so there is 1-edge (a,b) connecting a and b.

Hence, the problem of copy between $S_{n-1,k-1}(i)$ and $S_{n-1,k-1}(j)$ can be accomplished in constant time as shown in procedure **Copy**.

Procedure Copy ($S_{n-1,k-1}(i)$, $S_{n-1,k-1}(j)$)

1: **do in parallel** for all vertices $*i$ and $*j$

send contents to neighbors along dimension k.

2: **do in parallel**

for all edges $(i*,\ j*)$ exchange contents between $i*$ and $j*$;

for all edges $(i*j*,\ j*i*)$ exchange contents between $i*j*$ and $j*i*$.

3: **do in parallel** for all vertices $i*$ and $j*$

send back contends to neighbors along dimension k.

Assume that sending a message of constant size from one processor to another along an edge in $S_{n,k}$ takes one unit time and each processor can send or receive one constant size message along one and only one neighbour at a time. From these assumptions, we have:

Lemma 5. *The contents of $S_{n-1,k-1}(i)$ and $S_{n-1,k-1}(j)$, $i \neq j$, can be exchanged in a bijective way in $O(1)$ time.*

Now we extend this problem between two groups of $S_{n-1,k-1}$'s. Let group $I = \{i_1, i_2, \ldots, i_t\}$ and group $J = \{j_1, j_2, \ldots, j_t\}$ be two sequences from $\{1, 2, ..., n\}$ such that no two elements from each group are equal, and $I \cap J = \emptyset$. We want to exchange the contents in $S_{n-1,k-1}(i_1)$, $S_{n-1,k-1}(i_2)$, $\ldots$, $S_{n-1,k-1}(i_t)$ with contents in $S_{n-1,k-1}(j_1)$, $S_{n-1,k-1}(j_2)$, $\ldots$, $S_{n-1,k-1}(j_t)$. In other words, the contents of $S_{n-1,k-1}(i_r)$ are exchanged with $S_{n-1,k-1}(j_r)$, where $1 \leq r \leq t$. This problem is so called *Group Copy* of $S_{n,k}$, and can be solved by using procedure Copy given above.

Procedure Group-Copy $(I,\ J)$

1: **do in parallel** for $1 \leq r \leq t$

Copy $(S_{n-1,k-1}(i_r), S_{n-1,k-1}(j_r))$;

Since $i_r \neq j_r$, $1 \leq r \leq t$, by the given condition, we can easily show that no conflict will occur during exchanging process. This tells us the procedure Group-

Copy is correct and the time cost is the same as procedure Copy, which is $O(1)$.

This procedure enables us to develop some basic algorithms for the (n,k)-star network. In the next section, we will use it to design a prefix sums computation algorithm on the (n,k)-star topology.

5.2.2 Computing Prefix Sums

Using the Group Copy procedure from the last section, we can easily develop an algorithm to compute the prefix sums for two groups of sub-structure $S_{n-1,k-1}$'s. Suppose that we have already computed prefix sums for Groups 1 and 2, like:

$$\begin{aligned} &Group\ 1: \quad S_{n-1,k-1}(i) \cdots S_{n-1,k-1}(i+t) \\ &Group\ 2: \quad S_{n-1,k-1}(i+t+1) \cdots S_{n-1,k-1}(i+2t+1) \end{aligned}$$

Note that each processor has its own memory to save two variables $pSum$ and $tSum$, the partial prefix sums computed so far and the total sum of values in the group it is in, respectively. Since prefix sums are already computed in each group, let $tSum_1$ be the total sum for group 1 and $tSum_2$ for group 2. First, we use the Group Copy procedure to send $tSum_1$ to each processor in group 2 and $tSum_2$ to each processor in group 1. Then make no change to $pSum$ values in each processor of group 1, while the partial sum $pSum$ of a processor in group 2 becomes $pSum \oplus tSum_1$ [1]. The total sum for all processors is changed to $tSum_1 \oplus tSum_2$. After these steps, the prefix sums computation is done for all processors in groups 1 and 2. The running time is $O(1)$, since Group Copy requires only constant time.

Now it is straightforward to state the algorithm formally for computing prefix sums on $S_{n,k}$. For each $S_{n-1,k-1}$, the algorithm can be called recursively; until $k = 1$,

[1] $\oplus$ is an associative binary operation in this problem .

where $S_{n-k+1,1}$ is a clique and the prefix computation can be done in $O(\log(n-k+1))$ time.

Let $t(n,k)$ be the time required for computing prefix sums on $S_{n,k}$, and $t(n-k+1,1) = O(\log(n-k+1))$. So,

$$\begin{aligned} t(n,k) &= t(n-1,k-1) + C\log n \\ &= t(n-2,k-2) + C\log(n-1) + C\log n \\ &\cdots \\ &= t(n-k+1,1) + C(k-1)\log n \\ &= O(\log(n-k+1)) + C(k-1)\log n \\ &= O(k\log n) \end{aligned}$$

where C is a constant. Then the above algorithm of computing prefix sums needs $O(k\log n)$ time.

Now, we can reduce the problem of broadcasting to the problem of computing prefix sums as follows. Let the first processor have the value x and all the other processors have a value "0", and the binary operation $\oplus$ is the usual bit-wise OR (i.e., $x \oplus 0 = x$, for any x). Hence, after computing prefix sums on the network, each processor has the value x; the same as broadcasting x in this network. It is easy to see that the problem of computing all prefix sums has a lower bound of $\Omega(\log N)$ on an interconnection network with N nodes, where each nodes has only one value. Thus, the lower bound for this problem on $S_{n,k}$ is $\Omega(\log(\frac{n!}{(n-k)!})) = \Omega(k\log n)$. Thus, our algorithm is optimal in view of the lower bound.

5.3 Sorting

In Chapter 3, we showed a way embedding a 2-dimensional $n \times \frac{(n-1)!}{(n-k)!}$ mesh into an (n,k)-star graph. That is, consider an (n,k)-star graph, we can always think of it as arranged in an $n \times \frac{(n-1)!}{(n-k)!}$ array of the row-major order (ordering as reverse lexicographic order). Table 3.4 is an example for $S_{4,2}$. So the algorithm which is based on Shear Sort in [40] for a mesh-connected parallel computer can be used to sort values in (n,k)-star network.

The idea of this sorting algorithm is outlined below. Suppose the final sorting direction of a sequence is denoted as D, where D is either F or R direction defined in Chapter 2. And let $\overline{D}$ denote the opposite direction of D.

Procedure (n,k)**-star Sorting** (D)

1: **in parallel**

(a) sort all the odd numbered rows in the direction F

(b) sort all the even numbered rows in the direction R.

2: **for** $j = 1$ to $\lceil log\ n \rceil$ **do**

3: Start with row 1, arrange all rows into groups of 2^j consecutively numbered rows (except the last group)

4: **in parallel**

sort the columns within each group of row in direction D.

5: **in parallel**

(a) sort the rows in odd-numbered groups by calling Fill-The-Gap(D);

(b) sort the rows in even-numbered groups by calling Fill-The-Gap($\overline{D}$).

6: **end for**

And the sub-routine Fill-The-Gap is defined as follows:

Procedure Fill-The-Gap (D)

1: **if** $k \neq 1$ **then**
2: **in parallel** sort all column in direction D.
3: **in parallel** sort all rows with Fill-The-Gap(D).
4: **else**
5: **return**
6: **end if**

In the procedure of (n,k)-star Sorting, each iteration from Step 2 to 5 is the merging process. It merges two adjacent groups of sorted $S_{n-1,k-1}$'s. So the above sorting algorithm for an (n,k)-star becomes sorting on the columns of 2-dimensional array. We know that each column is connected as single path if we exchange the 1^{st} symbol with the k^{th} symbol in each processor. So the sorting algorithm for a linear array is applied to each column. This means that we first view all $S_{n-1,k-1}$'s as an $n \times \frac{(n-1)!}{(n-k)!}$ array. In the first step, each column of length n is sorted. Then procedure Fill-The-Gap is applied to each row i, $1 \leq i \leq n$, which means sorting on each $S_{n-1,k-1}(i)$. Then we consider an $(n-1) \times \frac{(n-2)!}{(n-k)!}$ array, each row is an $S_{n-2,k-2}$ and each column of length $(n-1)$ is sorted. Now, the procedure Fill-The-Gap is applied to each row $(S_{n-2,k-2})$. This process is repeated until $k = 1$ where a clique is reached. Hence, the merging is done by sorting on linear array of length $n, n-1, n-2, ..., n-k+1$, giving us the total time $n+(n-1)+(n-2)+\cdots+(n-k+1) = O(nk)$. Also, based on Richard Cole's ideas in [11], the time used to sort a clique K_{n-k+1} is $O(\log(n-k+1))$. Therefore, the above sorting algorithm for $S_{n,k}$ needs $t(n,k) = t(n-1,k-1) + \lceil \log n \rceil \times O(nk)$ with $t(n-k+1,1) = O(\log(n-k+1))$, which is $O(k^2 n \log n)$ time.

However, this sorting algorithm for $S_{n,k}$ is not time optimal, but it is easy to understand and simple to implement.

Chapter 6

Conclusion

In this thesis, we have studied the (n,k)-star interconnection network which is an attractive alternative to the n-star topology. We found some useful topological properties of the (n,k)-star and designed several parallel algorithms that could run on this network.

Specifically, we have discussed:

- decomposing the (n,k)-star into disjoint paths and cycles through 1-neighbours. In particular, $S_{n,k}$ can be decomposed into $\frac{n!}{(n-k+1)!}$ vertex-disjoint paths and each path has length $n-k+1$.
- decomposing this network into $\frac{n!}{(n-k+1)!}$ complete networks, each has $n-k+1$ nodes.
- finding the minimum dominating set of the graph (n,k)-star.
- embedding properties of the (n,k)-star graph. We embeded a k-dimensional mesh of size $n \times (n-1) \times (n-2) \times \cdots \times (n-k+1)$ into $S_{n,k}$; we then showed that a 2-dimensional mesh of size $n \times \frac{(n-1)!}{(n-k)!}$ can also be embedded into the (n,k)-star graph. Both of these two embeddings have the expansion cost 1.
- developing a neighbourhood broadcasting algorithm for the (n,k)-star network.

Such algorithm only applies to the single-port model. At the same time, using the result from this neighbourhood broadcasting algorithm, we designed a broadcasting algorithm for this network. As we know, both algorithms have optimal time complexity.

- designing a new simple broadcasting algorithm for all-port model (n, k)-star network. This algorithm is based on a minimum dominating set of an (n, k)-star graph.

- prefix sums algorithm for the (n, k)-star; in particular, we developed a prefix sums algorithm which computes the prefix sums with respect to the processor ordering in $S_{n,k}$.

- sorting and merging algorithm for the (n, k)-star. The idea is based on the embedding properties of the network and Shear Sort for a mesh structure.

The 1-neighbours path and cycle structure of the (n, k)-star graph are very simple and useful. They provide the basic ideas to developing routing and broadcasting (neighbourhood broadcasting) schemes.

When designing the prefix sums computation algorithm, we also introduced a procedure called Group-Copy which is another very important scheme for the (n, k)-star network. It enables us to develop not only the prefix sums computation, but also several other basic algorithms, such as routing.

Throughout this thesis, we have assumed that in one unit time, a processor can send or receive a datum to or from *one and only one* of its neighbours in a single-port model. On the other hand, a processor can send or receive a datum to or from *all*

of its neighbours in an all-port model. Furthermore, we can classify the (n,k)-star network into two families according to different cost assumptions, such as :

1. in one time unit, a processor in this network can only transmit at most one message of **fixed** length.

2. in one time unit, a processor in this network can transmit a message of **arbitrary** length to (from) one or all its neighbours.

In Chapter 4, we studied broadcasting algorithms for both single-port and all-port models of the (n,k)-star. With the exception of the broadcasting algorithm using minimum dominating set, all the others designed for $S_{n,k}$ are running on the single-port(weak) model.

In this thesis, we have only studies some of problems of the (n,k)-star network, many problems related to this topology remain open. Some of these problems are described below:

- Besides mesh embedding properties introduced in this thesis, more needs to be done to investigate embeddings of other well-known structures into the (n,k)-star graph, such as complete binary trees, hypercubes, and so on.

- A trivial lower bound for the problem of sorting on $S_{n,k}$ with $\frac{n!}{(n-k)!}$ nodes is $\Omega(\log(\frac{n!}{(n-k)!})) = \Omega(k \log n)$. And the algorithm we designed in Chapter 5 has a time complexity $O(k^2 n \log n)$. As we can see, there is a big gap between the trivial lower bound and our sorting algorithm. Thus one open problem is to improve sorting algorithm on $S_{n,k}$.

- Algorithms need to be designed for solving other problems on the (n,k)-star network.

As shown in this thesis, as well as other research work about the (n, k)-star interconnection network, we know that the (n, k)-star is indeed an attractive alternative to the n-star network. The (n, k)-star graph provides us better flexibility than the n-star in controlling the number of nodes in the network. However, there still exist some important algorithms on the (n, k)-star whose performances do not match those of algorithms developed for n-star. For example, sorting algorithm on n-star [5] achieved better time complexity than those on (n, k)-star so far. In despite of recent research studying on the (n, k)-star network, much work still needs to be done to make this network a serious competitor to the n-star.

Bibliography

[1] S. B. Akers, D. Harel and B. Krishnamurthy, "The Star Graph: An Attractive Alternative to the n-cube," *Proc. International Conference on Parallel Processing*, St.Charles, Illinous, August 1987, pp. 393-400.

[2] S. B. Akers and B. Krishnamurthy, "The Fault Tolerance of Star Graphs," 2^{nd} *International Conference on Supercomputing*, Vol. III, San Francisco, May1987, pp. 270-276.

[3] S. G. Akl, *Parallel Computation: Models And Methods*, Prentice Hall, Upper Saddle River, NJ, USA, 1997.

[4] S. G. Akl, *The Design and Analysis of Parallel Algorithms*, Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[5] S. G. Akl and T. Wolff, "Efficient Sorting on the Star Graph Interconnection Network", *Telecommunication Systems* 10, 1998, pp. 3-20.

[6] J. C. Bermond, A. Ferreira and J. G. Peters, "Partial Broadcasting in Hypercubes," *International Workshop on Interconnectional Networks*, Luminy, France, 1991.

[7] J. H. Chang and J. Kim "Ring Embedding in Faulty the (n, k)-star Graphs", *The 8th International Conference on Parallel and Distributed Systems*, 2001, pp. 0099.

[8] Y. S. Chen and K. S. Tai, "A Near-Optimal Broadcasting in (n, k)-Star Graphs", *ACIS International Conference on Software Engineering Applied to Networking and Parallel/Distributed Computing*, 2000, pp. 217-224.

[9] W. K. Chiang and R. J. Chen, "The (n, k)-Star Graph: A Generalized Star Graph," *Information Processing Letters*, 56, 1995, pp. 259-264.

[10] W. K. Chiang and R. J. Chen, "Topological Properties of the (n,k)-Star Graph," *International Journal of Foundations of Computer Science*, Vol. 9, No. 2, 1998, pp. 235-248.

[11] R. Cole "Parallel Merge Sort," *SIAM Journal on Computing*, Vol. 17, 1988, pp. 770 - 785.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts, 1990.

[13] M. Cosnard and A. Ferreira, "On the Real Power of Loosely Coupled Parallel Architectures", *Parallel Processing Letters*, 1, 1991, pp. 103-111.

[14] M. Dietzfelbinger, S. Madhavapeddy, and I. H. Sudborough, "Three Disjoint Path Paradigms in Star Networks," *Proc. 3^{rd} IEEE Symp. on Parallel and Distributed Processing*, Dallas, December 1991, pp. 400-406.

[15] G. Fertin and A. Raspaud, "Neighbourhood Communications in Networks", *Proc. Euro. Conference on Combinatorics, Graph Theory and Applications*, Electronics Notes on Discrete Mathematics, Vol.10, 2001.

[16] G. Fertin and A. Raspaud, "k-Neighbourhood Broadcasting", *8th International Colloquium on Structural Information and Communication Complexity*, 2001, pp. 133-146.

[17] M. Flynn, "Some Computer Organizations and Their Effectives", *IEEE Trans. Comput.*, Vol. C-21, 1972, pp. 948-960.

[18] S. Fujita, "Neighbourhood Information Dissemination in the Star Graph", *Proc. of the International Colloquium on Structural Information and Communication Complexity*, 1998, also in *IEEE Transaction on Computers*, 49(12), 2000, pp. 1366-1370.

[19] C. GowriSankaran, "Broadcasting on Recursively Decomposable Cayley Graphs", *Discrete Applied Math.*, Vol. 53, 1994, pp. 171–182.

[20] H. C. Hsu, Y. L. Hsieh, J. M. Tan and L. H. Hsu, "Fault Hamiltonicity and Fault Hamiltonian Connectivity of the (n, k)-Star Graphs", *Networks*, Vol. 42-4, 2003, pp. 189-201.

[21] H. C. Hsu, C. K. Lin, H. M. Hung and L. H. Hsu, "The Spanning Connectivity of The (n, k)-Star Graphs", *Inter. Journal of Foundations of Comp. Science*, Vol. 17, No. 2, 2006, pp. 415-434.

[22] J. S. Jwo, S. Lakshmivarahan and S. K. Dhall, "Embedding of Cycles and Grids in Star Graphs," *Proc.* 2^{nd} *IEEE Symp. Parallel and Distributed Processing*, Dallas, Texas, December 1990, pp. 540-547.

[23] H. P. Katseff, "Incomplete Hypercube," *IEEE Trans. Comput.*, C-37 (5), pp. 604-608, 1988.

[24] D. D. Kouvatsos and I. M. Mkwawa, "Neighbourhood Broadcasting Schemes for Cayley Graphs with Background Traffic", manuscript, online at *http://www.cms.livjm.ac.uk/pgnet2003/submissions/Paper52.pdf*, 2003.

[25] S. Latifi and N. Bagherzadeh, "Incomplete Star: An Incrementally Scalable Net-

work Based on the Star Graph," *IEEE Trans. on Parallel and Distributed System*, Vol. 5, No. 1, Jan. 1994, pp. 97-102.

[26] J. L. Li, M. L. Chen, Y. H. Xiang and S. W. Yao, "Optimum Broadcasting Algorithms in (n, k)-Star Graphs Using Spanning Trees", *IFIP International Conference on Network and Parallel Computing (NPC 2007)*, LNCS 4672, 2007, pp. 220-230.

[27] V. E. Mendia and D. Sarkar, "Optimal Broadcasting on the Star Graph", *IEEE Trans. on Parallel and Distributed System*, Vol.3, No. 4, 1992, pp. 389-396.

[28] A. Menn and A.K. Somani, "An Efficient Sorting Algorithm for the Star Graph Interconnection Network", *Proc. International Conference on Parallel Processing*, 1990, vol. 3, pp. 1-8.

[29] R. Miller and Q. F. Stout, "Simulating Essential Pyramids," *IEEE Trans. on Compu.*, Vol. 37, No. 12, 1988, pp. 1642-1648.

[30] I. M. Mkwawa and D. D. Kouvatsos, "An Optimal Neighbourhood Broadcasting Scheme for Star Interconnection Networks," *Journal of Interconnection Networks*, 4(1), 2003, pp.103-112.

[31] W. Najjar and P. K. Srimani, Conditional Disconnection Probability in Star Graphs, Technical Report CS-90-105, Department of Computer Science, Colorado State University, 1990.

[32] F. P. Preparata and J. Vuillemin, *The Cube-Connected-Cycle: A versatile Network for Parallel Computation*, Comm. ACM, Vol.24, No.5, 1981, pp.300-309

[33] K. Qiu and S. K. Das, "A Novel Neighbourhood Broadcasting Algorithm on Star Graphs", *IEEE 9th International Conference on Parallel and Distributed Systems*, Taiwan, Dec.2002, pp.37-41.

[34] K. Qiu, S. G. Akl and H. Meijer, "On Some Properties and Algorithms for the Star and Pancake Interconnection Networks", *Journal of Parallel and Distributed Computing*, 22, 1994, pp. 16-25.

[35] K. Qiu "A Unified Neighbourhood Broadcasting Scheme for Multiple Messages on Interconnection Networks", *ACST'06: Proceedings of the 2nd IASTED International Conference on Advances in Computer Science and Technology*, 2006, Puerto Vallarta, Mexico, pp.234-238.

[36] K. Qiu, H. Meijer and S. Akl "Decomposing a Star Graph into Disjoint Cycles", *Information Processing Letters*, 39, 1991, pp. 125-129.

[37] S. Ranka, J. Wang and N. Yeh "Embedding Meshes on the Star Graph", *Supercomputing '90: Proceedings of the 1990 conference on Supercomputing*, New York, United States, 1990.

[38] C. P. Ravikumar, A. Kuchlous and G. Manimaran, "Incomplete Star Graph: An Economical Fault-Tolerant Interconnection Network," *Proc. International Conference on Parallel Processing*, Vol. 1, 1993, pp. 83-90.

[39] M. R. Samatham and D. K. Pradhan, "The de Bruijn Multiprocessor Network: A Versatile Parallel Processing and Sorting Network for VLSI," *IEEE Trans. on Compu.*, Vol. 38, No. 4, April 1989, pp. 567-581.

[40] I. D. Scherson and S. Sen, "Parallel Sorting in Two-Dimensional VLSI Models of Computation," *IEEE Trans. on Compu.*, Vol. 38, No. 2, 1989, pp. 238-249.

[41] J. P. Sheu, C. T. Wu and T. S. Chen, "An Optimal Broadcasting Algorithm without Message Redundancy in Star Graphs", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 6, June, 1995, pp. 653-658.

[42] P. K. Srimani, "Generalized Fault Tolerance Properties of Star Graphs", *Technical Report CS-90-104*, Department of Computer Science, Colorado State University, 1990.