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

by

Yifeng Li

A Thesis Submitted to the

Department of Computer Science

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Faculty of Mathematics and Science

Brock University

St. Catharines, Ontario, Canada

August, 2009

Approved for the Committee:

Dr. K. Qiu (Supervisor)

Dr. S. Houghten

Dr. V. Wojcik

Department of Computer Science

Acknowledgements

I would like to thank my supervisor Dr. Ke Qiu for suggesting the topic of this thesis, and giving me lots of valuable comments. He has shown me how to do the research and provided me with many insightful ideas. He always gives me directions when I was lost in my research field. I thank him for his patience over past two years.

I would like to thank the members of my supervisory committee, which includes Dr. S. Houghten, and Dr. V. Wojcik, for their comments, criticisms and suggestions.

Financial support from Department of Computer Science, Brock University is gratefully acknowledged.

Finally, I would like to thank my parents, Jianpin Li and Peijun Chen for their love and support. I also wish to thank my dear friends, Liang He, Si Zhang, Fan Zhang, Xiang Yin, Jing Sun, Jimeng Sun, Yingjie Ji, Yingjue Xu. Without their help, I could not finish this thesis such successfully.

Thank you very much and I love you all.

Co-Authorship

Some preliminary results were reported in the following paper:

1. "A note on broadcasting on the arrangement graph,"(with K. Qiu), in *Proceedings of 20th IASTED International Conference on Parallel and Distributed Computing and Systems,* (Orlando,Florida,USA), pp. 141-144, November 2008.

Abstract

The (n, k)-arrangement interconnection topology was first introduced in 1992. The (n, k)-arrangement graph is a class of generalized star graphs. Compared with the well known n-star, the (n, k)-arrangement graph is more flexible in degree and diameter. However, there are few algorithms designed for the (n, k)-arrangement graph up to present. In this thesis, we will focus on finding graph theoretical properties of the (n, k)- arrangement graph and developing parallel algorithms that run on this network.

The topological properties of the arrangement graph are first studied. They include the cyclic properties. We then study the problems of communication: broadcasting and routing. Embedding problems are also studied later on. These are very useful to develop efficient algorithms on this network.

We then study the (n, k)-arrangement network from the algorithmic point of view. Specifically, we will investigate both fundamental and application algorithms such as prefix sums computation, sorting, merging and basic geometry computation: finding convex hull on the (n, k)-arrangement graph.

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

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the past 30 years, parallel computation has become a major area in computer science. The most primary reason for this is the time efficiency. Using a parallel system to solve a problem takes less time than using a sequential computer. A general idea is the more processors the faster. Unlike the sequential computation, there are two important aspects in parallel computation which are parallel computational models and parallel algorithms.

Generally, a parallel algorithm is designed for a specific parallel computational model. Parallel computational models with good properties make it easier to design efficient algorithms for parallel computation. We can classify the popular models into two major categories, namely, *shared-memory parallel machines* and *interconnection networks*, depending on how the processors communicate with each other.

In this chapter, we first introduce the basic classification of parallel computer architectures. We then give an introduction of shared-memory parallel machines followed by an introduction of the interconnection networks and some common topolo-

gies. The (n, k)-arrangement graph is defined next. The method to analyze and evaluate parallel algorithms is also presented. Finally, we give an overview and the organization of this thesis.

1.1 Classification of Computer Architectures

Depending on how instruction steams interact with data streams during program execution, we can classify computers into following four categories[23]:

- **Single Instruction, Single Data Stream (SISD)**

  In SISD computers, there is only one control unit and one memory unit. Both instruction and data work sequentially. A SISD type computer is the general sequential machine.

- **Multiple Instruction, Single Data Stream (MISD)**

  A MISD computer has multiple processors, each processor has its own control unit and all processors share one common memory unit. For the same data, different instructions are executed on it in parallel. Sometimes a pipeline of processors is considered MISD.

- **Single Instruction, Multiple Data Stream (SIMD)**

  A SIMD computer has multiple processors, all processors are controlled by a central control unit and each processor has its own memory unit. In each parallel step, all the processors execute the same instruction on its own data. All the major parallel models in this thesis are SIMD computers.

- **Multiple Instruction, Multiple Data Stream (MIMD)**

  A MIMD computer has multiple processors, each processors has its own control unit and memory unit. In each parallel step, all the processors execute the different instruction on its own data. Therefore, a MIMD computer is more powerful than other three classes of computers. Examples of MIMD computers include the Cosmic Cube, nCUBE 2, iPSC, etc[33].

On a parallel computer, processors need to communicate with each other to solve any non-trivial problems by either through a shared memory or an interconnection network.

1.2 Shared-Memory Parallel Machines

A Shared-Memory Parallel Machine contains a number of identical processors and a common memory. All the processors access the shared memory by a memory access unit (MAU). Since the sequential computer is called the random access machine (RAM), a Shared-Memory Parallel Machine is also known as parallel random access machine (PRAM). A PRAM can be seen in Figure 1.1.

Each processor in PRAM reads data from memory by using MAU and uses MAU again to write the intermediate result and the final result back. Abstractly, all the processors access the shared memory in parallel. The most serious problem here is how the processors access the same location in shared memory. It is worth mentioning that one of the design goals for parallel algorithm design is to avoid such situations, as they inevitably slow down computation. In [5], four different ways for multiple processors to read from or write to the same memory location simultaneously are

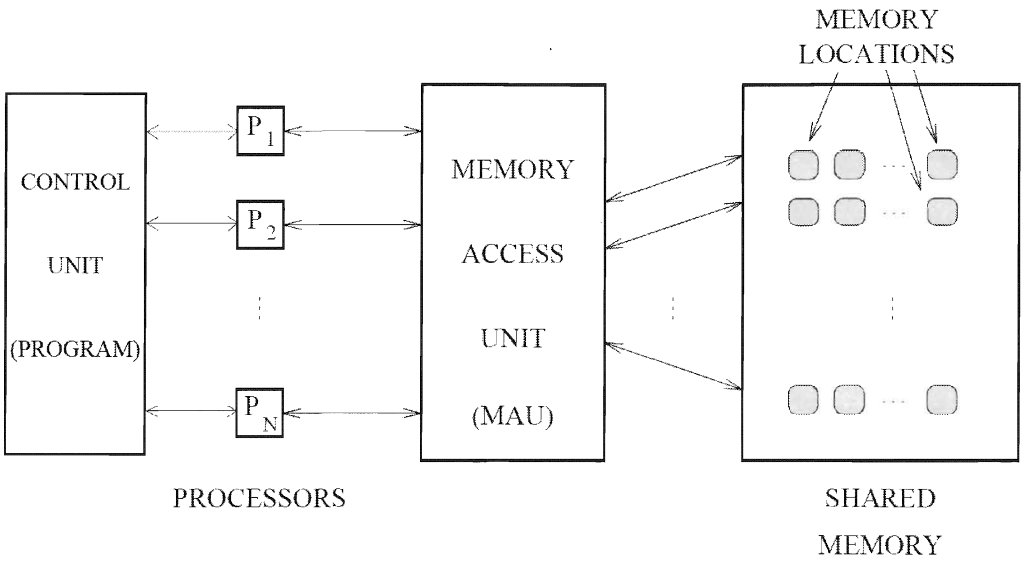


Figure 1.1: Shared-Memory Parallel Machines (PRAM)

listed, which are made possible by the PRAM's repertoire of instructions.

- **Exclusive Read (ER).** In this model, for a memory location, only one processor can read the data. In parallel, different processors can read from different memory locations.

- **Concurrent Read (CR).** In this model, for a memory location, multiple processors can read the same data simultaneously. **ER** can be seen as a special case of **CR**.

- **Exclusive Write (EW).** In this model, for a memory location, only one processor can write the data to the location.

- **Concurrent Write (CW).** In this model, for a memory location, multiple processors can write to the same location simultaneously. However, conflicts may

happen when several processors try to write to the same location at same time. Many extensions are available to be used with **CW** to resolve such conflict.

1. **PRIORITY CW.** Each processor has a priority. It allows only the highest priority processor to write the data into this memory location.
2. **COMMON CW.** Comparing the data of all the processors, if they are of the same value, then they are allowed to write into this location.
3. **ARBITRARY CW.** A deterministic algorithm is used to decide which value to save into the memory location.
4. **RANDOM CW.** A randomly selected processor is allowed to write the data into the memory location.
5. **COMBINING CW.** A combined value involving some or all the values from these processors gets written into the memory location.

Combining different Read and Write options, we can get four different types of PRAM computers which are **Exclusive Read, Exclusive Write (EREW), Exclusive Read, Concurrent Write (ERCW), Concurrent Read, Exclusive Write (CREW),** and **Concurrent Read, Concurrent Write (CRCW)**.

1.3 Interconnection Networks

In the last section, all communications in PRAM are done through a shared memory. In the interconnected networks, each processor has its own memory unit and communicates with the other processors by a topological network. In the network, if two processors are connected by a *two-way* communication link, it means they can

exchange data simultaneously. Also, two processors directly connected by a link are said to be neighbours. We can use an undirected graph to describe an interconnection network. Mathematically, given an undirected graph $G = (V, E)$, where each processor P_i is located at the vertex v_i and there exists a direct communication link between two processors P_i and P_j if and only if $(v_i, v_j) \in E$. In this thesis, we will use the terms "processor" and "node", "interconnection network" and "graph" interchangeablely.

Many interconnection networks have been proposed, built, and used as commercial system. Next, we will briefly introduce some typical networks. We assume the number of processors is N for the following networks.

Complete Graph: The complete graph is the most powerful network. In a complete graph K_N, each of the processors is adjacent to the remaining $N-1$ processors. A complete graph is also called a **Clique**.

Linear Array: The linear array is the simplest way to connect N processors, $P_0, P_1, ..., P_{N-1}$. In this network, all N processors form a one-dimensional array. Each processor P_i $(0 < i < N-1)$ is adjacent to its two neighbors P_{i-1} and P_{i+1}. The first node P_0 is adjacent to P_1 and the last node P_{N-1} to P_{N-2}. Both of them have only one neighbor.

If we connect P_0 and P_{N-1}, we get a network called **Ring**. In this case, every node has two neighbours.

Tree: In this network, all the N processors form a complete binary tree.

Two-Dimensional Array: A network is obtained by arranging the N processors into an $r \times s$ two dimensional array. The processor in row i and column j is denoted by P_{ij}, where $0 \leq i \leq r-1$ and $0 \leq j \leq s-1$. Each processor P_{ij} has two-way communication links to its four neighbours $P_{(i+1)j}$, $P_{(i-1)j}$, $P_{i(j+1)}$ and $P_{i(j-1)}$ if they

exist. Processors on the boundary rows and columns have fewer than four neighbours.

This network is also known as **Mesh**. A multi-dimensional mesh can be defined similarly. Such a network is called a d-dimensional mesh, where $d \geq 2$. Each processor in a d-dimensional mesh is adjacent to its $2 \times d$ neighbours, except the processors on the boundary. Figure 1.2 shows a $2 \times 2 \times 2$ Mesh (a) and a 3×3 Mesh (b).

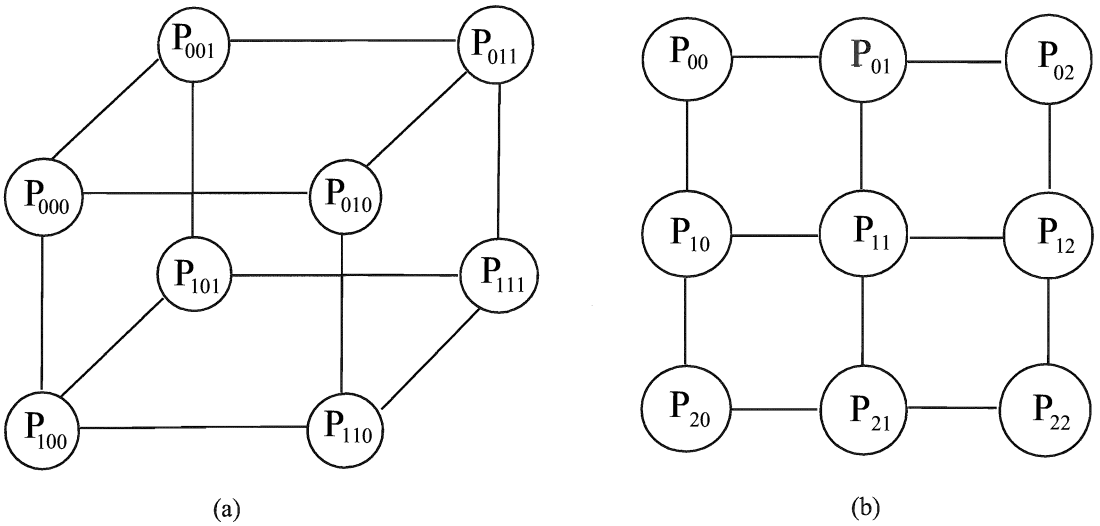


Figure 1.2: $2 \times 2 \times 2$ Mesh and 3×3 Mesh

Perfect Shuffle: Let N be a power of 2 ($N = 2^k$) and label N processors as $P_0, P_1, ..., P_{N-1}$. In the prefect shuffle interconnection network, a one-way shuffle line links P_i to P_j where

$$j = \begin{cases} 2i & 0 \leq i \leq N/2 - 1 \\ 2i + 1 - N & N/2 \leq i \leq N - 1 \end{cases} \tag{1.1}$$

In addition, we switch one-way links to two-way connections. A two-way exchange link is added between each processor with even label and its successor in the network.

Let $i_{k-1}i_{k-2}...i_1i_0$ be the binary representation of i. The i_j shows the j^{th} bit of i's binary representation, where $0 \leq j \leq k-1$. Let $\bar{i_j}$ be the binary complement of i_j. In the prefect shuffle network, the processor $P_{i_{k-1}i_{k-2}...i_1i_0}$ is adjacent to $P_{i_{k-2}i_{k-3}...i_0i_{k-1}}$ via shuffle line and to $P_{i_{k-1}i_{k-2}...i_1\bar{i_0}}$ via the exchange line. For example, Figure 1.3 shows a prefect shuffle with N = 8. P_{001} is adjacent to P_{010} by a shuffle line and to P_{000} by an exchange line.

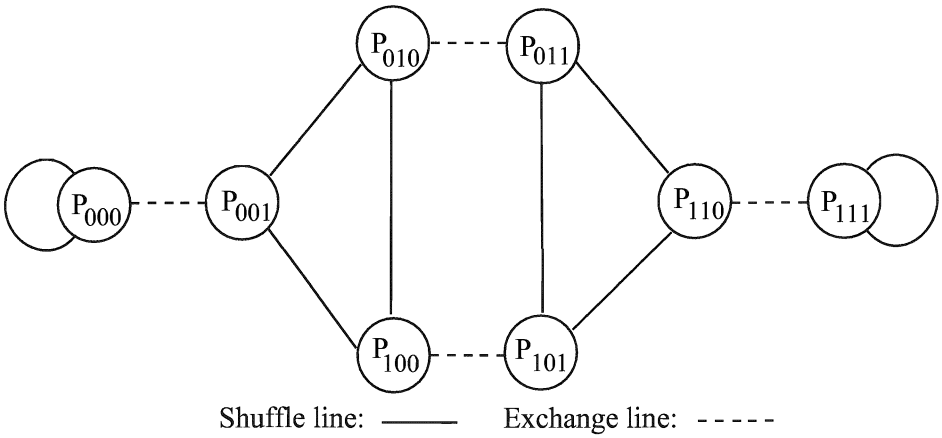
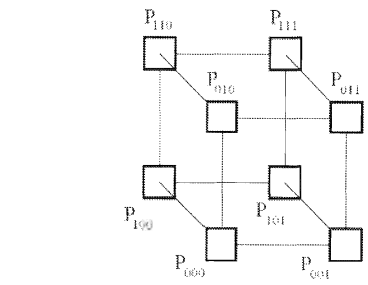


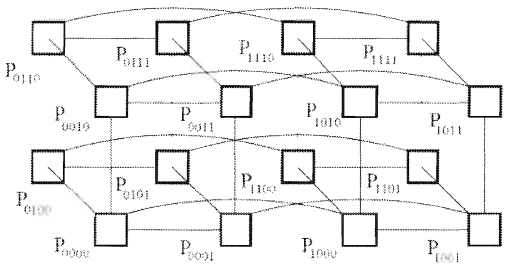
Figure 1.3: A perfect-shuffle interconnection network with $n = 8$

Hypercube: An n-dimensional hypercube is also known as an n-cube. Let N be 2^n for some $n \geq 0$ and label all processors as $P_0, P_1, ..., P_{N-1}$. In an n-cube, for P_i, let $i_0i_1...i_{n-2}i_{n-1}$ be the binary representation of i, where $0 \leq i < N$. The processors P_i and P_j are adjacent if and only if the binary representations of the indices i and j differ in exactly one bit. Figure 1.4 shows a 3-cube and a 4-cube.

The **Cube-Connected Cycles network**, or **CCC** for short, is a variation of the hypercube. A Cube-Connected Cycles topology has similar properties as the n-cube and has some additional advantage. We can obtain a Cube-Connected Cycles from



(a) Hypercube with $n = 3$



(b) Hypercube with $n = 4$

Figure 1.4: Hypercube interconnection network with $n = 3$ and $n = 4$

a n-cube by replacing each of the 2^n nodes with a ring of n processors. More details about Cube-Connected Cycles topology can be found in [25, 46].

Star: The star graph was proposed to be an attractive alternative to the hypercube topology for interconnecting processors in a interconnection network, and compares favorably with it in several aspects [3]. In a star graph, there are $N = n!$ processors. We call the network S_n or n-star. Each processor P is a permutation of n symbols. P and Q are adjacent if and only if Q can be obtained from P by interchanging the first symbol and i^{th} symbol in P, where $2 \leq i \leq n$. For example, in 4-star, processor P_{1234} is adjacent to P_{2134}, P_{3214} and P_{4231} by interchanging the first symbol 1 with the second, third and fourth symbol. Figure 1.5 shows a 4-star. More details of the n-star will be discussed in Section 5.3.

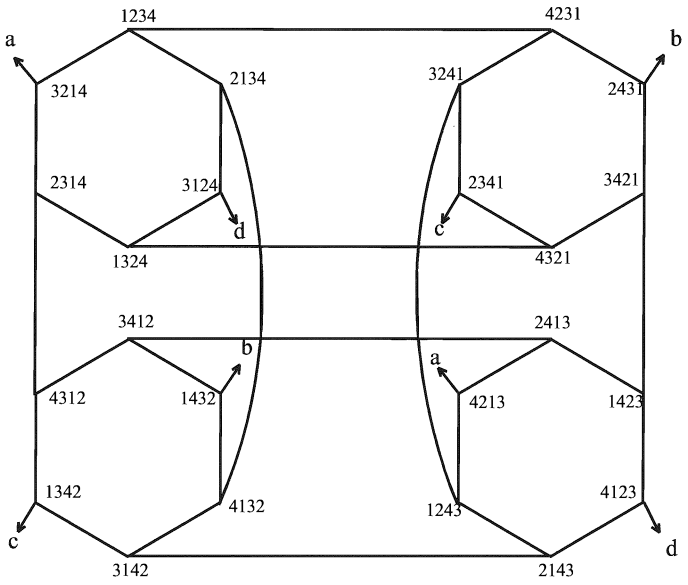


Figure 1.5: 4-Star

There are also many other interconnection networks such as the *De Bruijin* network [50], the *mesh-of-the-tree* [4], and the *pyramid* [42], etc.

A number of criteria are used to evaluate network topologies. We now introduce some of them and then use them to analyze the networks described above.

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_i and P_j is the number of links on the shortest path from P_i to P_j; the **diameter** of the network is the maximum distance among any two arbitrary processors.*

Degree is an important criterion for assessing a topology. For example, the degree of a clique K_N is $N-1$, while that of an n-cube is n. For a network, a large de-

gree is desirable in terms of diameter, connectivity, and fault-tolerant (defined later). However, having many neighbors is not only expensive, but may also be infeasible. **Diameter** is another important criterion. Since processors need to communicate among themselves and since the time for a message to go from one processor to another depends on the distance separating them, a small diameter is better than one with a large diameter in networks [5]. For example, the diameter of a linear array with N processors is $N-1$. Table 1.1 shows the degree and diameter of topologies we described earlier.

Table 1.1: Interconnection networks and their degrees and diameters

Interconnection network	Degree	Diameter	Precise Diameter
Linear Array	2	$O(N)$	$N-1$
$r \times s$ Mesh	4	$O(max(r,s))$	$(r-1)+(s-1)$
Tree	3	$O(\log N)$	$2\lfloor logN \rfloor$
Mesh of Tree	6	$O(\log N)$	$2\log N$
Pyramid	9	$O(\log N)$	$C\log N$
Shuffle-Exchange	3	$O(\log N)$	$2\log N-1$
Hypercube(n-cube)	n	$O(\log N)$	$\log N$
Cube-Connected Cycles of order n	3	$O(\log N)$	$2n+\lfloor n/2 \rfloor - 2$
n-Star	$n-1$	$O(\frac{\log N}{\log\log N})$	$\lfloor \frac{3(n-1)}{2} \rfloor$

Definition 3 *The **connectivity** of a graph G with N points is N - 1 if G is the complete graph and otherwise is the minimum number of points of G whose deletion*

results in a disconnected graph[24].

In a complete graph K_N of N nodes, each node is adjacent to all the other $N-1$ nodes and K_N has $N(N-1)/2$ lines. Therefore, the connectivity of K_N is $N-1$.

Definition 4 *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 the maximum number of f for which it is f-fault tolerant.*

The difference of connectivity and fault tolerance is 1.

Definition 5 *A graph is **regular** if and only if all nodes in this graph have the same degree.*

Definition 6 *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.*

Symmetric is very useful for routing in interconnection network because a vertex symmetric graph allows for all the node to be identical.

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.

Similar to the PRAM, depending on how many neighbours can communicate in one time unit, we can divide interconnection networks into two models.

- **single-port** (weak) model, in each unit of time a processor is only allowed to send data to or receive data from one of its neighbours.

- **all-port** (strong) model, the processor can communicate with one or more of its neighbours simultaneously.

Unless specified otherwise, all interconnection networks in this thesis are considered to be the single-port model.

1.4 (n, k)-Arrangement Graph

As mentioned before, the star graph was proposed to be an alternative to the hypercube topology for interconnecting processors in a parallel computer and compares favorably with the hypercube in terms of the degree, symmetry properties, maximal fault tolerance, etc [2, 22, 44, 52]. Specifically, a star graph of dimension n is a regular graph with degree $n-1$. It has $n!$ nodes, but both its degree and diameter are $O(n)$, i.e., sub-logarithmic in the number of vertices, while a hypercube with $O(n!)$ vertices has a degree and diameter of $O(\log(n!)) = O(n \log n)$, i.e., logarithmic in the number of vertices. Other properties include symmetry properties, as well as many desirable fault tolerance characteristics [3]. However, a major limitation to its feasibility as a topology in which processors are connected in an interconnection network is the requirement that the number of nodes in an n-star be $n!$, resulting in a huge gap between the n-star and the $(n+1)$-star. For the very popular hypercube, a similar problem exists since an n-cube contains 2^n nodes while the next one has 2^{n+1} nodes. It is for this reason that the (n, k)-star graph [16] and the (n, k)-arrangement graph [19] are proposed, both generalizations of the star graph.

Definition 7 *For $1 \leq k \leq n$, an (n, k)-arrangement graph, denoted by $A_{n,k}$, is a regular graph. The vertex set is the set of all k-permutations over $\{1, 2, \cdots, n\}$, that*

is, $\{p_1p_2\cdots p_k | 1 \leq p_i \leq n$, *and for* $i \neq j, p_i \neq p_j\}$ *such that two nodes are adjacent if their addresses differ in exactly one position.*

We will use $i*$ to represent a node in $A_{n,k}$ whose first symbol is i, $1 \leq i \leq n$. The wild card symbol in $i*j$, $i*j*$, and $*i$ are defined similarly, where $1 \leq i, j \leq n$ and $i \neq j$.

Let a node p in the (n,k)-arrangement graph be $p = p_1p_2...p_k$. Let $INT(p)$ be a **internal set** of node p defined by $INT(p) = \{p_1, p_2, ..., p_k\}$ and $EXT(p)$ be the **external set** of p defined by $EXT(p) =< n > -INT(p)$ [19]. A symbol (element) in the external set is called **external symbol (element)**. In addition, each node p can also be written as $p_1p_2\cdots p_k|e_1e_2\cdots e_{n-k}$ where $e_1 < e_2 < \cdots < e_{n-k}$ are external symbols.

For example, in a $(5,3)$-arrangement graph, node 123 is adjacent to 423, 523, 143, 153, 124, and 125. $\{4,5\}$ is the external set of node 123. Figure 1.6 shows a $(4,2)$-arrangement graph.

For each dimension j, $1 \leq j \leq k$, each node has $(n-k)$ neighbours that we call j-neighbours. Therefore, $A_{n,k}$ is a $k(n-k)$-regular graph with $n!/(n-k)!$ number of nodes. The diameter of $A_{n,k}$ is $O(k)$ [19]. In addition, it is both vertex and edge symmetric, $A_{n,1}$ is isomorphic to an n-clique K_n, and $A_{n,n-1}$ is isomorphic to the n-star [19]. In $A_{n,n-1}$, we can assume the external symbol is the first symbol in S_n and we can treat $A_{n,n-1}$ as a n-star by the definition of the n-star. This implies that n-star is a special case of (n,k)-arrangement graph. (n,k)-arrangement graph has more flexibility than an n-star when designing the interconnection network in parallel computation in terms of the number of nodes. Figure 1.7 shows a $(4,3)$-

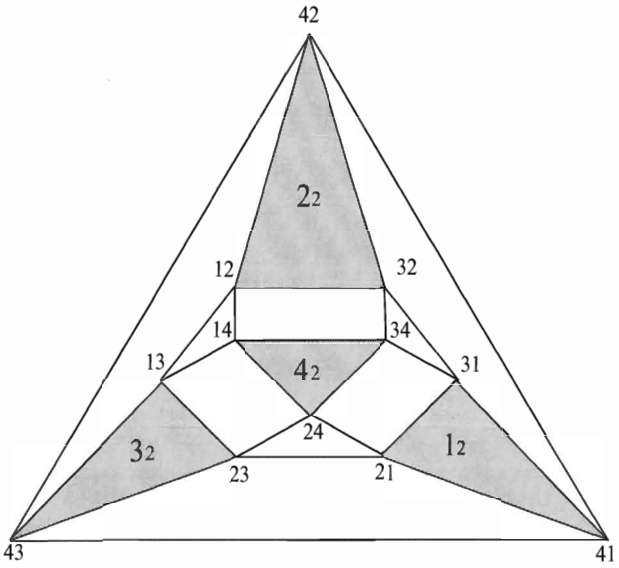


Figure 1.6: $(4, 2)$-Arrangement Graph

arrangement graph and a 4-Star and the external symbol of the arrangement graph is in parentheses.

Furthermore, there is an isomorphism between the n-alternating group graph and $(n, n-2)$-arrangement graph[15]. The n-alternating group graph $AG_n = (V, E)$ is defined as follows: Let $V = A_n$, and $E = \{(p, q)|p, q \in V$, and $q = p \circ q$ for some $g \in \Omega\}$, where A_n is the set of even permutations of n elements. An even permutation is a permutation obtainable from an even number of two-element swaps [18]. Ω is a generator set for A_n and $\circ$ is the composition operator. Here, the generator Ω is defined as $g_i^- = (12i)$, $g_i^+ = (1i2)$ and $\Omega = \left\{g_i^+|3 \leq i \leq n\right\} \cup \left\{g_i^-|3 \leq i \leq n\right\}$. The dimension of an alternating group graph is $\left\lceil \frac{3n}{2} \right\rceil - 2$ and the degree is $(n-2)$ [35]. More details about n-alternating group graph can be found in [15, 31, 35].

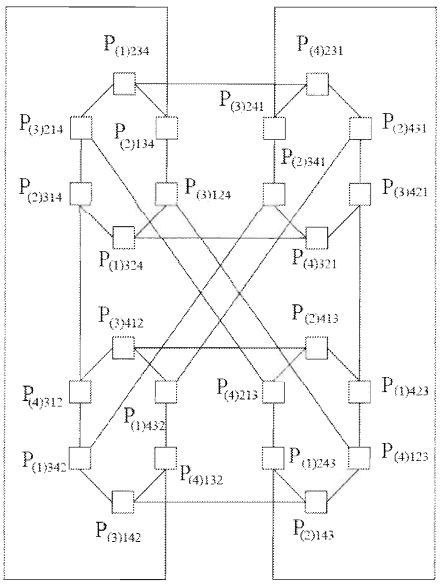


Figure 1.7: $(4, 3)$-Arrangement Graph and 4-Star

1.5 Analyzing Parallel Algorithms

When applying a parallel algorithm, the most important three criteria are: **running time**, **the number of processors used**, and **cost** [5].

The **running time** of a parallel algorithm is the time required by this algorithm when executed to solve a problem on a parallel computer. Usually, the running time of a parallel algorithm algorithm is obtained by counting elementary steps in the worst case. There are two different types of elementary steps in parallel algorithms:

- A **computational step** is a basic arithmetic or logical operation performed on one or two data within a processor. Such operations include adding, comparing, swapping, etc.

- A **routing step** is used by an algorithm to send a constant size datum from

one processor to another via the shared memory or interconnection network.

We assume that each elementary step takes a constant number of time units. The standard techniques used in analyzing sequential algorithms are applied also in parallel algorithms. We use a function $t(N)$ to represent the running time of a parallel algorithm of input size N.

Another criterion for measuring the performance of a parallel algorithm is the **number of processors used**. Since the more processors used the more expensive the cost of building the computer system is, we have to balance the cost and performance. It is very important to minimize the number of processors used while maintaining the same time complexity. We use $p(N)$ to denote the number of processors used by a parallel algorithm to solve a problem of size N.

The **cost** $c(N)$ of a parallel algorithm 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 the cost of a parallel algorithm matches a lower bound which is known for a sequential algorithm for the same problem, then this parallel algorithm is said to be **cost optimal**.

1.6 Organization of the Thesis

Many interconnection networks have been discussed previously and some new networks are continuously being proposed. The (n, k)-arrangement graph is proposed in 1992 [19] which is an alternative network to the widely studied n-star graph. As a new proposed network, some fundamental algorithms for the (n, k)-arrangement

have not been studied yet. They include prefix sums, merging, sorting, etc. We will investigate the (n, k)-arrangement network from both the graph theoretical and the algorithmic points of views in this thesis.

We will discuss the following topics in this thesis:

1. a literature review of the (n, k)-arrangement graph;
2. cyclic properties;
3. external symbols;
4. designing an optimal neighbourhood broadcasting algorithm for $A_{n,k}$, and using it to develop an optimal broadcasting algorithm;
5. a routing algorithm that exchanges the contents between two groups of $A_{n,k}$'s in constant time;
6. embedding mesh on the (n, k)-arrangement graph;
7. fundamental and application algorithms for $A_{n,k}$ including:
   (a) prefix sums;
   (b) sorting and merging;
   (c) translation and reversing;
8. basic geometry computation: convex hull problem.

This thesis is organized as follows. Chapter 2 offers a literature review of the (n, k)-arrangement graph. We will also define various problems to be studied in this thesis.

In Chapter 3, some graph theoretical properties are studied and an optimal neighbourhood broadcasting and an optimal broadcasting algorithm are presented. We discuss two constant routing algorithms for the (n, k)-arrangement graph in Chapter 4. In Chapter 5, we present several algorithms developed for the (n, k)-arrangement graph. Finally, our concluding remarks, some open problems and future research directions are offered in Chapter 6.

Chapter 2

Literature Review of the Arrangement Graph

2.1 Introduction

The (n, k)-arrangement network has received much attention lately. We will offer a literature review on the (n, k)-arrangement interconnection network. We will review the network from two aspects: its topological properties and parallel algorithms. All necessary terms, notations, and problems will be defined accordingly.

2.2 Properties

Definition 8 *A set of k-permutations is a set of permutations of the n elements of* $< n >$ *taken k at a time.*

Definition 9 *An edge of* $A_{n,k}$ *connecting two sets of k-permutations p and q which*

differ only in position i, $1 \leq i \leq k$*, is called an* ***i-edge*** *[19].*

Proposition 1 (n,k)*-arrangement graph is a regular graph with degree* $k(n-k)$ *and* $n!/(n-k)!$ *nodes [19].*

Proof: By the definition of the $A_{n,k}$. □

Theorem 1 $A_{n,k}$ *is vertex symmetric and edge symmetric [19].*

Vertex symmetric shows that given two vertices, in $A_{n,k}$ there exists an automorphism that maps one vertex into the other. Edge symmetric shows that given any two edges, there exists an automorphism that maps one edge into the other.

The (n,k)-arrangement graph has a cycle representation. The special node $12...k$ is called the **identity node** and is denoted by I_k [19]. It is always possible to find a cycle representation for an node p of the (n,k)-arrangement graph using a set of non-trivial **internal cycles** and **external cycles** of length 2 or more such that all the elements in internal cycles are in $INT(p)$ and each external cycle contains exactly one external element. For example, in $A_{9,7}$, the node $p = 6351792$. There are two cycles: $C_1 = (2,3,5,7)$, $C_2 = (4,1,6,9)$. We can see all elements in C_1 are in $INT(p)$, it is therefore an internal cycle. C_2 is an external cycle because 4 is an external element of p.

As a result of the node symmetric property, A path from p_1 to p_2 can be mapped to a path from p' to I_k. We can find the path by correcting cycles in p'. For the previous example, the path for correcting the external cycle is $6351792 \rightarrow 635\underline{4}792 \rightarrow \underline{1}354792 \rightarrow 1354\underline{7}62$. And the path for correcting the internal cycle is $1354762 \rightarrow 1354768 \rightarrow 1\underline{2}54768 \rightarrow 12\underline{3}4768 \rightarrow 1234\underline{5}68 \rightarrow 123456\underline{7}$. Let c be the number of

non-trivial cycles (both internal and external), m the number of elements in these cycles and e the number of external cycles.

Lemma 1 *The distance $D(p)$ between p and I_k in $A_{n,k}$ is $D(p) \leq c + m - 2e$ [19].*

Proof: Suppose C is a external cycle of node p', $C = (x_1, x_2, ..., x_\alpha)$, and x_α is the foreign external element (an external element of node I). C can be corrected by first moving its x_1 to its correct position which is held by x_2, then the second element, x_2, is taken to its correct position which is held by x_3. Repeat it until the element, $x_{\alpha-1}$, is corrected to its correct position, making x_α external. Therefore the correction of an external cycle C_i containing m_i elements requires $m_i - 1$ steps. On the other hand, correcting an internal cycle $C' = (y_1, y_2, ..., y_\beta)$ requires to exchange one of its elements (say y_1) with any external element, z, then y_1 is taken to its correct position which is held by y_2. Repeat it until all the elements go back to their correct position, making z external again. The correction of an internal cycle C_j containing m_j elements requires $m_j + 1$ steps. Therefore,

$$\begin{aligned} Dis(p) &\leq m_1 - 1 + ... + (m_e - 1) + (m_{e+1} + 1) + ... + (m_c + 1) \\ &= c + m - 2e \end{aligned}$$ □

Theorem 2 *The distance $D(p)$ between p and I_k in $A_{n,k}$ is $D(p) = c + m - 2e$ [19].*

Corollary 1 *The diameter of $A_{n,k}$ is $\left\lfloor \frac{3}{2}k \right\rfloor$ [19].*

Proof: Recall from Definition 2, the diameter is $max\{Dis(p) | p \in A_{n,k}\}$. The maximum distance between an arbitrary node p and the identity node I_k is $c + m - 2e$. The maximum value of the expression is obtained for $c = \lfloor k/2 \rfloor$, $m = k$, and $e = 0$. Therefore the diameter is $\left\lfloor \frac{3}{2}k \right\rfloor$. □

Definition 10 *Let i_j be a subgraph of $A_{n,k}$ induced by all the vertices with the same j^{th} symbol i, where $1 \leq i \leq n$ and $1 \leq j \leq k$.*

Proposition 2 *There are k different ways to decompose an $A_{n,k}$ into n node-joint $A_{n-1,k-1}$'s: i_j, for $1 \leq i \leq n$ and $1 \leq j \leq k$[19].*

From Definition 7, for any fixed i and j, where $1 \leq i \leq n$ and $1 \leq j \leq k$, there are $(n-1)!/(n-k)!$ nodes which have symbol i in position j. These nodes form an $A_{n-1,k-1}$. For a fixed j, $A_{n,k}$ can be decomposed into n such subgraphs 1_j, 2_j, ..., n_j, thus partitioning $A_{n,k}$ into n copies of $A_{n-1,k-1}$ [19] and i_j is an $A_{n-1,k-1}$. For example, an $A_{4,2}$ in Figure 1.6 contains four $(3,1)$-arrangement graphs by fixing the position 2, which are 1_2, 2_2, 3_2, and 4_2.

This hierarchical structure of $A_{n,k}$ is one of the most important properties of the (n,k)-arrangement graph. We are going to exploit this property in our various algorithms, for example, in our broadcasting algorithm and constant routing algorithm which will be presented later. In our algorithms we normally use the first position to do the decomposition.

Another type of partitioning is obtained by fixing an element i instead of a position j. For a fixed element i, the sub-graphs $i_1, i_2, ..., i_k$ together with the sub-graph i_0, of all nodes which do not have i in any of their k positions, form a partitioning of $A_{n,k}$ into k copies of $A_{n-1,k-1}$: $i_1, i_2, ..., i_k$, and one copy of $A_{n-1,k}$: i_0. This partitioning can be done in n different ways [19].

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 as **Hamiltonian**. Hamiltonian cycle on (n,k)-arrangement graph has been studied in

[21] and [30].

Proposition 3 *(n, k)-arrangement graph is Hamiltonian, $1 \leq k \leq n-1$. [21]*

Recently, more research results about the (n, k)-arrangement network have been discussed by some researchers. For example, the embedding property of $A_{n,k}$ is introduced in [20, 21] and the fault tolerance and connectivity properties of $A_{n,k}$ are discussed in [29, 38].

2.3 Algorithms

2.3.1 Neighbourhood Broadcasting and Broadcasting

One of important operations on a parallel computer is broadcasting where one node (source) sends a message to all nodes. A similar problem that has been studied is the problem of *neighbourhood broadcasting* which is defined as sending a fixed sized message from the source node to all its neighbours where in one time unit, a node can send to or receive from exactly one of its neighbours a datum of constant size [17]. This problem has been studied for several interconnection networks [12, 17, 26, 27, 49, 47]. Clearly, for any interconnection network with N nodes, the problem of broadcasting has a trivial lower bound of $\Omega(\log N)$ since the number of nodes receiving the message can at most double after each step. Similarly, the problem of neighbourhood broadcasting has a trivial lower bound of $\Omega(\log n)$ where n is the degree of the source node.

The problem of broadcasting on the arrangement graph has been considered previously. In [11], an optimal $O(k \log n)$ algorithm was developed for an (n, k)-

arrangement graph. The central idea is to utilize different-sized broadcasting trees for subgraphs that constitute a spanning tree for the graph. Although optimal (and without message redundancy), this algorithm is fairly complicated and the derivation is also quite involved. Chen *et al.* later found a broadcasting algorithm for the (n, k)-arrangement graph in [14] where multiple spanning trees were used. This result was improved in [37]. For single-port model, neither algorithm is asymptotically optimal.

2.3.2 Prefix Sums

Given a set $S = \{x_0, x_1, \cdots, x_{n-1}\}$ and a closed binary associative operation $\otimes$ defined over S, the prefix sums are n sums

$$\begin{aligned} s_0 &= x_0 \\ s_1 &= x_0 \otimes x_1 \\ &\cdots\cdots \\ s_{n-1} &= x_0 \otimes x_1 \otimes \cdots \otimes x_{n-1} \end{aligned}$$

The study of the prefix sums problem is important as the problem is a generalization of many problems as these problems are simply special cases of the prefix sums problems when appropriate binary operations are used. Such problems include broadcasting, interval broadcasting, and biological sequence comparison [9]. Prefix sums computation is also used in other applications such as computing the ranks of elements for sorting, computing carries for carry-lookahead addition, etc. The reader is referred to [13, 36] for in-depth study of the problem.

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 node holds one

element. This can be shown by reducing the problem of broadcasting to the problem of computing prefix sums as follows. Let the first processor have the message and let all the other processors have a value "0" (i.e., $x \otimes 0 = x$), for any x, and the operation $\otimes$ is the usual bit-wise OR. Thus, the lower bound for this problem on $A_{n,k}$ is $\Omega(\log(n!/(n-k)!)) = \Omega(k \log n)$.

We will develop an optimal algorithm for the prefix sums problems on $A_{n,k}$ in Chapter 5.

2.3.3 Sorting

Given a sequence of elements e_i stored in a set of ordered processors P_i with the ordering relation $\prec$, with each processor holding one element, we say that the sequence is sorted in nondecreasing order if $P_i \prec P_j$ then $e_i \leq e_j$. The nonincreasing order is defined similarly.

Sequentially, $\Omega(N \log N)$ number of steps are required to sort N numbers [1]. In an (n,k)-arrangement graph, $N = n!/(n-k)!$, which implies an $\Omega(\log(n!/(n-k)!)) = \Omega(k \log n)$ lower bound to sort on an (n,k)-arrangement graph. Sorting on n-star has been studied in [39], [48] and [7]. Since (n,k)-arrangement graph is a generalization of the star graph, we can get some ideas from these algorithms.

One way to define the order of processors in the (n,k)-arrangement graph is the so called *lexicographic order*. For example, the permutations of $\{1,2,3\}$ in lexicographic order are 123, 132, 213, 231, 312, and 321. We can easily list all the node in the (n,k)-arrangement graph in lexicographic order by fixing the first position. For example,

in $A_{4,2}$ the lexicographic order of nodes are

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

Furthermore, another order is called *reverse lexicographic order*, i.e., lexicographic order if we read from right to left. Node in reverse lexicographic order can be generated by decompositing the (n, k)-arrangement graph on the last position. In this thesis, without notice, we will use lexicographic order as the order of processors for some algorithms (prefix sums , sorting, etc) in the (n, k)-arrangement graph.

We will develop two sorting algorithms for the arrangement graph in Chapter 5. The first one is based on a sorting algorithm for the star graph by Menn and Somani [39] while the second one is based on a sorting algorithm on n-dimensional mesh by Kunde [34].

2.3.4 Convex Hull

The problem of finding the convex hull of a set of N points is one of the most important problems in computational geometry. It has been well studied for serial model of computation [45, 10]. Convex hull is widely used to normalize patterns in image processing, obtain triangulation of sets of points, and topological feature extraction, etc. [45, 53].

A convex set is a set in a vector space which is defined to contain line segment between any two points in the set. The convex hull of a set S of points, denoted $hull(S)$ is defined to be the minimum convex set containing S. Our algorithm is to find the convex hull of a set of N planar points on the (n, k)-arrangement graph.

Divide-and-conquer is a common strategy to find the convex hull $hull(S)$ of a set

of points S. Figure 2.1 shows a convex hull on a 2-D plane. Given an input size of $N = n!/(n-k)!$ points, initially distributed one element per processor, we will design an algorithm to find th econvex hull in Chapter 5 using the divided-and-conquer technique.

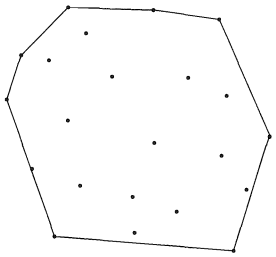


Figure 2.1: A Convex Hull on a Plane

Note that the convex hull problem can be solved sequentially in the optimal $O(N \log N)$ time. On interconnection parallel models, it has been studied in mesh [43], mesh of trees [41], pyramid [41], hypercube [40, 41] and n-star [6].

Chapter 3

Broadcasting

3.1 Introduction

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 the (n,k)-arrangement graph, the lower bound will be $\Omega(\log(n!/(n-k)!)) = \Omega(k \log n)$ for BP and $\Omega(\log n)$ for NBP.

In this chapter, we first present cyclic properties of the (n,k)-arrangement graph. An optimal neighbourhood broadcasting algorithm is then presented by using these properties. Next, we use this result to design an optimal broadcasting algorithm on the (n,k)-arrangement graph. These properties and broadcasting algorithms are very useful in developing efficient parallel algorithms on the (n,k)-arrangement network

in the next two chapters.

3.2 Cyclic Properties

Recall from the definition of the (n,k)-arrangement graph, for each dimension i, $1 \leq i \leq k$, each node has $(n-k)$ neighbours that we call i-neighbours. For example, given a node $p = 12...(i-1)i(i+1)...k$ in $A_{n,k}$, $1 \leq i \leq k$, then its i-neighbours are

$$12...(i-1)(k+1)(i+1)...k$$

$$12...(i-1)(k+2)(i+1)...k$$

$$...$$

$$12...(i-1)(n)(i+1)...k$$

Because the $A_{n,k}$ is vertex-symmetric, without loss of generality, we assume that the source node is $12\cdots k$. For this node, its neighbours are

$$\begin{cases} (k+1)23\cdots k \\ (k+2)23\cdots k \\ \cdots \\ n23\cdots k \end{cases}$$

$$\begin{cases} 1(k+1)3\cdots k \\ 1(k+2)3\cdots k \\ \cdots \\ 1n3\cdots k \end{cases}$$

$$\cdots$$

$$\begin{cases} 123\cdots(k-1)(k+1) \\ 123\cdots(k-1)(k+2) \\ \cdots \\ 123\cdots(k-1)n \end{cases}$$

The cyclic properties are based on the following observations:

Observation 1 *For any node u and $1 \leq j \leq k$, u and its $n-k$ j-neighbours form a clique K_{n-k+1}.*

Proof: By the definition of the $A_{n,k}$. □

Observation 2 *For $1 \leq i, j \leq k$ and any node u, u, any one of its i-neighbours, and any one of its j-neighbours form a cycle of length six.*

Proof: By symmetry, we assume that $u = I_k$, and without loss of generality, we assume that $i < j$. Then the following 6-cycle contains u, one i-neighbour, and one j-neighbour, where $1 \leq l, m \leq n-k$:

$$\begin{aligned} &\boldsymbol{123\cdots i\cdots j\cdots k} \leftrightarrow \\ &\boldsymbol{123\cdots(i-1)(k+l)(i+1)\cdots(j-1)j(j+1)\cdots k} \leftrightarrow \\ &123\cdots(i-1)(k+l)(i+1)\cdots(j-1)i(j+1)\cdots k \leftrightarrow \\ &123\cdots(i-1)j(i+1)\cdots(j-1)i(j+1)\cdots k \leftrightarrow \\ &123\cdots(i-1)j(i+1)\cdots(j-1)(k+m)(j+1)\cdots k \leftrightarrow \\ &\boldsymbol{123\cdots(i-1)i(i+1)\cdots(j-1)(k+m)(j+1)\cdots k} \leftrightarrow \end{aligned}$$

where $\leftrightarrow$ represents a bi-directional link (edge) between two nodes. □

Observation 3 *For any node u, any two 6-cycles formed as in Observation 2 with distinct $1 \leq i_1, j_1, i_2, j_2 \leq k$ are disjoint except at u.*

Proof: Without loss of generality, assume that $u = I_k$, $i_1 < j_1$, and $i_2 < j_2$. We can simply list the two 6-cycles and compare them to see that they are indeed disjoint:

Cycle 1

$$123..(i_1-1)i_1(i_1+1)..(j_1-1)j_1(j_1+1)..k \leftrightarrow$$

$$123..(i_1-1)(k+l_1)(i_1+1)..(j_1-1)j_1(j_1+1)..k \leftrightarrow$$

$$123..(i_1-1)(k+l_1)(i_1+1)..(j_1-1)i_1(j_1+1)..k \leftrightarrow$$

$$123..(i_1-1)j_1(i_1+1)..(j_1-1)i_1(j_1+1)..k \leftrightarrow$$

$$123..(i_1-1)j_1(i_1+1)..(j_1-1)(k+m_1)(j_1+1)..k \leftrightarrow$$

$$123..(i_1-1)i_1(i_1+1)..(j_1-1)(k+m_1)(j_1+1)..k \leftrightarrow$$

Cycle 2

$$123..(i_2-1)i_2(i_2+1)..(j_2-1)j_2(j_2+1)..k \leftrightarrow$$

$$123..(i_2-1)(k+l_2)(i_2+1)..(j_2-1)j_2(j_2+1)..k \leftrightarrow$$

$$123..(i_2-1)(k+l_2)(i_2+1)..(j_2-1)i_2(j_2+1)..k \leftrightarrow$$

$$123..(i_2-1)j_2(i_2+1)..(j_2-1)i_2(j_2+1)..k \leftrightarrow$$

$$123..(i_2-1)j_2(i_2+1)..(j_2-1)(k+m_2)(j_2+1)..k \leftrightarrow$$

$$123..(i_2-1)i_2(i_2+1)..(j_2-1)(k+m_2)(j_2+1)..k \leftrightarrow$$

□

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

3.3 Neighbourhood Broadcasting

Based on the observations in last section 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 $A_{n,k}$, a simple neighbourhood broadcasting algorithm for $A_{n,k}$ 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 1-neighbours. This 1-neighbour then sends the message to a 2-neighbour of the source node through a 6-cycle. Then the 1-neighbour sends the message to a 3-neighbour and in parallel, the 2-neighbour sends the message to a 4-neighbour, etc. In $O(\log k)$ time, for each $1 \leq i \leq k$, there exists an i-neighbour with the message. Then finally, in parallel, for all $1 \leq i \leq k$, in $O(\log(n-k))$ time, all nodes in the K_{n-k}-clique consisting of all $n-k$ i-neighbours will have the message (done by a standard broadcasting algorithm on K_{n-k}), resulting in a total time of $O(\log k + \log(n-k)) = O(\log n)$ neighbourhood broadcasting algorithm.

Essentially, after each step, the number of nodes with the message is doubled (with the possible exception of the last step). For example, $n = 6$, $k = 4$ and source node $e = 1234$, the neighbourhood broadcasting is done in the following fashion:

- **Step 1:**

  $1234 \rightarrow 5234$ (Direct link)

- **Step 2:**

  $5234 \rightarrow 5134 \rightarrow 2134 \rightarrow 2534 \rightarrow 1534$ (Six Length Cycle)

- **Step 3:**

  $5234 \rightarrow 5214 \rightarrow 3214 \rightarrow 3254 \rightarrow 1254$ (Six Length Cycle)

  $1534 \rightarrow 1532 \rightarrow 1432 \rightarrow 1435 \rightarrow 1235$ (Six Length Cycle)

- **Step 4:**

  $5234 \rightarrow 6234$ (Direct Link in K_2)

  $1534 \rightarrow 1634$ (Direct Link in K_2)

  $1254 \rightarrow 1264$ (Direct Link in K_2)

  $1235 \rightarrow 1236$ (Direct Link in K_2)

To the best of our knowledge, this neighbourhood broadcasting algorithm is the first such algorithm for the (n, k)-arrangement graph. In view of the $\Omega(\log n)$ lower bound, our algorithm is optimal.

3.4 Broadcasting

With our algorithm for neighbourhood broadcasting just developed in the last section, broadcasting on an (n, k)-arrangement graph can now be done easily. Once again, without loss of generality, assume that node $123...k$ wants to broadcast a piece of message. In the first step, the message will be sent to all its neighbours through neighbourhood broadcasting algorithm in $O(\log n)$ time so that all i-neighbours have the message, $2 \leq i \leq k$. We note that for all $2 \leq i \leq k$, each i-neighbour of node $123...k$ is adjacent to a node of the form $i*$, a k-permutation whose first symbol is i.

Since we want to make sure that at least one node of form $i*$ has the message, one more step is required. The algorithm proceeds as follows:

Broadcast$(A_{n,k})$

if $k = 1$ (the network has become a clique), simply perform a standard broadcasting algorithm

else the source node $12\cdots k$ performs a neighbourhood broadcasting.

After one more step, we have a node of the form $1*$, $2*$, $\cdots$, and $n*$,

(e is $1*$ and all 1-neighbours are $(k+1)*$,$(k+2)*$, ... $n*$) with the message

In parallel, do Broadcast(1_1), Broadcast(2_1), ..., Broadcast(n_1).

In the example of $A_{6,4}$, we know all i-neighbours, $2 \leq i \leq k$ are 1534 and 1634 (2-neighbours), 1254 and 1264 (3-neighbours), and 1235 and 1236 (4-neighbours). Each i-neighbour is adjacent to one node of $i*$ as follows:

- **2 neighbour $\rightarrow$ 2***

  $1534 \rightarrow 2534$

  $1634 \rightarrow 2634$

- **3 neighbour $\rightarrow$ 3***

  $1254 \rightarrow 3254$

  $1264 \rightarrow 3264$

- **4 neighbour $\rightarrow$ 4***

  $1235 \rightarrow 4235$

  $1236 \rightarrow 4236$

After this step, in every i_1 (an $A_{n-1,k-1}$) at least one node has the message. We can then broadcast in every sub-graph recursively.

Let $t(n,k)$ be the running time for broadcasting on $A_{n,k}$, then $t(n,k)$ is easily seen to be:

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

which is optimal in view of the $\Omega(\log(n!/(n-k)!))$ lower bound.

Compared with other broadcasting algorithms [14, 37], our novel broadcasting algorithm achieves optimality but is much simpler. However, we do want to point out that our broadcasting algorithm does allow message redundancy in that a small number of nodes receive the message more than once. In the next chapter, another broadcasting algorithm which is different from all the previous ones will be presented. This algorithm is based on our constant routing algorithms on the (n,k)-arrangement graph.

Chapter 4

Routing Problems

4.1 Introduction

In this chapter, we will first present a constant time routing algorithm that routes the contents of i_1 to the processors of j_1, $i \neq j$, $1 \leq i, j \leq n$, in a bijective fashion. Then we extend this algorithm to allow data to be exchanged between two disjoint groups of $A_{n-1,k-1}$'s. It also runs in constant time. These two constant routing algorithms help us develop several fundamental algorithms for the arrangement graph in the next chapter.

Furthermore, other than the broadcasting algorithms mentioned in the last chapter, to the best of our knowledge, no other algorithms have been developed for the arrangement graph, making our algorithms from this chapter the first such algorithms for the (n, k)-arrangement graph.

We will discuss two classes of highly parallel algorithms: ASCEND and DESCEND on the (n, k)-arrangement graph. We also present a way to embed a multidimensional

mesh into the (n, k)-arrangement graph. Later, two algorithms, translation and reversing, will be designed using this ASCEND and DESCEND idea.

4.2 Constant Routing I

Consider the following problem: Given i_1, and j_1, with $i \neq j$, it is required to exchange the contents of the processors in i_1 with the processors in j_1. By exchanging the contents of i_1 with j_1 we mean that the content of each processor in i_1 (j_1) is routed to a processor in j_1 (i_1) such that no two processors in i_1 (j_1) send their contents to the same processor in j_1 (i_1). In other words, the mapping defined by such an exchange is a bijection between i_1 and j_1. For example, in $A_{4,2}$, nodes in 1_1 are 12, 13 and 14; processors in 2_1 are 21, 23 and 24. We can view this problem as exchanging the contents of i_1 and j_1 to each other in arbitrary order. This can be accomplished in constant time as shown in Procedure Exchange. Before we discuss the procedure, we first give the Definition 11 and Lemma 2:

Definition 11 *In $A_{n,k}$, for any given pair i, j, $i \neq j$, node $ia_2 \cdots a_l j a_{l+2} \cdots a_k$ is the companion node of node $ja_2 \cdots a_l i a_{l+2} \cdots a_k$ and vice versa with respect to the pair i, j.*

Clearly, a node and its companion node have the same set of external symbols.

Lemma 2 *In $A_{n,k}$, nodes $i * j *$ and its companion node $j * i *$ are connected through a 3-length path.*

Proof: By the definition of the (n,k)-arrangement graph, it is not hard to see we can get a routing between $i*j*$ and its companion node $j*i*$ via a 3-length path:

$$i*j* \rightarrow x*j* \rightarrow x*i* \rightarrow j*i*$$

where x is an external symbol of both $i*j*$ and its companion node $j*i*$. □

In our example, 12 and 21 can be exchanged though the 3-length path, and other nodes are adjacent. the exchange can be completed in 3 steps in parallel. Procedure Exchange is given as follows:

Procedure Exchange (i_1, j_1)

1. **do in parallel** for all vertices $i*$ $(j*)$ where
   $j \notin i*$ $(i \notin j*)$, send content to its neighbour $j*$ $(i*)$.

2. **do in parallel** for all other nodes $i*j*$ and its companion node $j*i*$,
   their contents are excahnged through a 3-length path. □

The selection of the external symbol x is not important in this case, since there are only two $A_{n-1,k-1}$'s involved. For example, the algorithm could simply use the smallest external symbol in the routing. It is easy to see from the procedure that if $j \in i*$, then the node $i*$ and its companion node will exchange their information.

Assume that sending a constant-size datum from one processor to another along an edge takes unit time. Then in a single-port model, the contents of i_1 and j_1, $i \neq j$, can be exchanged in a bijective way in $O(1)$ time.

4.3 Constant Routing II

We now extend the Constant Routing I to two disjoint groups as follows. Let $I = \{i_1^1, i_1^2, ..., i_1^l\}$ and $J = \{j_1^1, j_1^2, ..., j_1^l\}$ be two groups of $A_{n-1,k-1}$'s where $i^1, i^2, \cdots, i^l$, $j^1, j^2, \cdots, j^l \in \{1, 2, \cdots, n\}$ and $|I \bigcup J| = 2l$, i.e., elements from I and J are all distinct. It is desired to exchange the contents of i_1^m with j_1^m, for all $1 \leq m \leq l$. Without loss of generality, we assume that $i_1^m < j_1^m$, for all $1 \leq m \leq l$. This task can also be achieved in constant time.

We first consider the special case where $i^1 = 1$, $j^1 = 2$, $i^2 = 3$, $j^2 = 4$, $\cdots$, $i^{n/2} = n-1$, $j^{n/2} = n$ (where, without loss of generality, we assume that n is even). Unlike the previous case when only one pair of $A_{n-1,k-1}$'s is involved, the selection of the external symbols used in the routing is important. For any node $a_1 a_2 \cdots a_k | e_1 e_2 \cdots e_{n-k} \in A_{n,k}$, where $e_1 < e_2 < ... < e_{n-k}$, let $S = \{e_1, e_2, \cdots, e_i\}$ and $T = \{e_{i+1}, e_{i+2}, \cdots, e_{n-k}\}$ where $e_i < a_1 < e_{i+1}$. The external symbol used in the routing of this node is $x = \max\{\min S, \min T\}$. That is,

$$x = \begin{cases} e_1 & T = \emptyset \\ e_{i+1} & otherwise. \end{cases}$$

In our case, it is not hard to prove that for any node $(2m-1) * (2m) * \in i_1^m$ and its companion node $(2m) * (2m-1) * \in j_1^m$ have the same external symbol because there is no integer between $2m-1$ and $2m$. With this choice for the external symbol used in the routing, let us examine, for each node, how many other nodes use it in their routing path (of length 3). For a node $(2m-1) * \in i_1^m$ such that $(2m) \notin (2m-1) *$, it is routed to $(2m) * \in j_1^m$ via the direct link between the two nodes. Otherwise, as noted before, a node $a_1 a_2 \cdots a_k = i * (i+1) *$ and its companion node $(i+1) * i *$,

where i is an odd number, are on a length-3 path

$$i*(i+1)* \leftrightarrow x*(i+1)* \leftrightarrow x*i* \leftrightarrow (i+1)*i*.$$

For any node $a_1a_2\cdots a_k|e_1e_2\cdots e_ie_{i+1}\cdots e_{n-k}$ such that $e_i < a_1 < e_{i+1}$, it has $n-k$ neighbours at the position 1:

$$\begin{aligned}
e_1a_2\cdots a_k &\mid e_2e_3\cdots e_ia_1e_{i+1}\cdots e_{n-k},\\
e_2a_2\cdots a_k &\mid e_1e_3\cdots e_ia_1e_{i+1}\cdots e_{n-k},\\
&\cdots\\
e_ia_2\cdots a_k &\mid e_1e_2\cdots e_{i-1}a_1e_{i+1}\cdots e_{n-k},\\
&\cdots\\
e_{n-k}a_2\cdots a_k &\mid e_1e_2\cdots e_ia_1e_{i+1}\cdots e_{n-k-1}.
\end{aligned}$$

Among them, only $e_ia_2\cdots a_k|e_1e_2\cdots e_{i-1}a_1e_{i+1}\cdots e_{n-k}$ has a_1 as its external symbol for the routing purpose, assuming that $i \geq 1$. If $S = \emptyset$, no node will use $a_1a_2\cdots a_k|e_{i+1}\cdots e_{n-k}$ as its second node in the routing since no node will have a_1 as its external symbol used for the routing. Therefore, we can claim that if the routing algorithm Exchange is used for the group exchange, each node u is used by at most one other node v in v's routing (as the second node in a length-3 path for v). Of course, u is the first node in its own routing. Therefore, for the special case, the contents of nodes in I and J can be exchanged in constant time by procedure Group-Exchange given below.

Procedure Group-Exchange (I, J)

1. **do in parallel** for $1 \leq m \leq l$

   Exchange (i_1^m, j_1^m). □

When the elements of I and J are arbitrary, we can do the following:

Let $I = \{i_1^1, i_2^2, \cdots, i_1^l\}$ and $J = \{j_1^1, j_2^2, \cdots, j_1^l\}$ such that $i^1 < i^2 < \cdots < i^l$ and $i^m < j^m$ for all $1 \leq m \leq l$. We can define a new linear order

$$i^1 < j^1 < i^2 < j^2 < \cdots < i^l < j^l.$$

The selection of the external symbol used for the routing of each node is carried out with the new linear order. For example, when $I = \{1, 2, 4, 5\}$ and $J = \{6, 3, 8, 7\}$, the new linear order is $1 < 6 < 2 < 3 < 4 < 8 < 5 < 7$. For node $13564 \in A_{8,5}$, its external symbol is $\max\{\min \emptyset, \min\{2, 8, 7\}\} = 2$ and has a routing path $13564|2 \leftrightarrow 23564|1 \leftrightarrow 23514|6 \leftrightarrow 63514|2$. Note that for node 63514, its external symbol is also 2. As another example, consider node 87413, which can be written as $87413|625$ and its external symbol is $\max\{\min\{6, 2\}, \min 5\} = \max\{6, 5\} = 5$ and has a routing path $87413|5 \leftrightarrow 57413|8 \leftrightarrow 57813|4 \leftrightarrow 47813|5$. Similarly, for node $47813|625$, its extended symbol is also 5. Then the procedure Group-Exchange can be used to exchange the contents of nodes in I with nodes in J in a bijective way in constant time.

Here, a few words about the running time of the routing algorithms Exchange and Group-Exchange are in order. If the external symbols have been determined before the routings, then clearly, both routing algorithms require constant time. However, the selection of external symbols for all the nodes is not a constant-time operation. In fact, according to our routing algorithms, these external symbols require $O(n-k)$ $= O(n)$ time to be found. On the other hand, it is worth pointing out that for many applications, the pattern of data exchanges is pre-determined, such as the cases for

all of our algorithms in the next section that are direct applications of the routing algorithm Group-Exchange. For example, for the broadcasting algorithm in next section and prefix sums algorithm in next chapter, we will be doing exchanges as follows:

- 1_1 to 2_1;
- 1_1 to 3_1

  2_1 to 4_1;
- 1_1 to 5_1

  2_1 to 6_1

  3_1 to 7_1

  4_1 to 8_1;
- .....

Therefore, the external symbols could be computed before hand by each node in a pre-processing step. In addition, the computation time required to obtain these external symbols is negligible compared to the routing/communication steps.

4.4 Broadcasting Using Constant Routing

Broadcasting on (n, k)-arrangement has been studied in the last chapter. The broadcasting algorithm for the arrangement graph is a trivial application of the constant-time routing Group-Exchange.

First of all, the message is broadcast to all other processors in 1_1 recursively. Note that the recursion stops when k becomes 1. In this case the graph becomes a clique

K_{n-k+1} so that broadcasting can be done in $O(\log(n-k+1)) = O(\log n)$ time. Then the contents of 1_1 are copied to the rest of the $A_{n,k}$'s by the technique of recursive doubling and the routing procedure. In other words, the contents of 1_1 are copied to the vertices in 2_1, the contents of 1_1 and 2_1 are subsequently copied to the vertices in 3_1 and 4_1, then the contents of 1_1, 2_1, 3_1, and 4_1 are copied to the vertices in 5_1, 6_1, 7_1, and 8_1. This process continues until the message is broadcast to all the vertices in the $A_{n,k}$.

Let $t(n,k)$ be the time complexity of the broadcasting algorithm for $A_{n,k}$, then $t(n,k) = t(n-1,k-1) + c\log n$, resulting in $t(n,k) = O(k\log n)$ which is optimal in view of the $\Omega(k\log n)$ lower bound.

4.5 ASCEND and DESCEND

In this section, we will discuss two classes of highly parallel algorithms: **ASCEND** and **DESCEND** on the (n,k)-arrangement graph. In order to design an ASCEND and DESCEND algorithm, way to embed a multi-dimensional mesh into an arrangement graph will be introduced first. We then present a coordinate rank of the (n,k)-arrangement graph which is based on the embedding. Two example algorithms, **Translation** and **Reversing**, will be designed. Finally, we will discuss the general case of ASCEND and DESCEND.

4.5.1 Embedding Meshes into Arrangement Graph

Definition 12 *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.

In considering graph embedding problems, two costs, *dilation cost* and *expansion cost*, are involved [28]. The *edge dilation* of edge $(i,j) \in E_G$ is $d(f(i), f(j))$, $f(i), f(j) \in V_H$. The *dilation cost* of f is defined as $max_{(i,j)\in E_G}(d(f(i), f(j)))$. In addition, the *expansion cost* is the ratio of the size of H to the size of G, i.e., $|V_H|/|V_G|$.

In parallel computation, interconnection networks are presented as graphs. One of the objectives of embedding a guest (source) graph into a host (target) graph is to simulate a parallel algorithm for the guest graph on the host graph. The guest graph sometimes represents an existing parallel algorithm and the host graph is an interconnection network where the algorithm is executed. The difference of running time for the same algorithm between guest graph and host graph is dependent on the *dilation cost* and *expansion cost.*

The problem of embedding into the (n,k)-arrangement graph has been studied in [20, 21]. In [20] the problem of embedding meshes, hypercubes, and trees has been considered and in [21] the problem of embedding cycles is discussed. The results about embedding meshes in [20] is stated in the following theorem:

Theorem 3 *An* $(n-k+1) \times (n-k+2) \times ... \times (n-1) \times n$ *mesh can be embedded in* $A_{n,k}$ *with unit expansion and dilation 3.*

The number of nodes for both the $(n-k+1) \times (n-k+2) \times ... \times (n-1) \times n$ mesh and the (n,k)-arrangement is $n!/(n-k)!$. The expansion cost is 1. In embedding an $(n-k+1) \times (n-k+2) \times ... \times (n-1) \times n$ mesh, two nodes are connected though either a direct link or a 3 length path. For example, in $A_{4,3}$, a node 234 is connect to node 134, 231, 214, 243, and 324. Nodes 134, 231, 214 are adjacent to

234. Nodes 243 and 324 are the companion nodes of 234 which are connected to 234 by 3 length routing paths. This 3 length path has been discussed in our Constant Routing algorithm in Section 4.2. Here the **Constant Routing II** is used and leads to a constant communication cost with embedding. Therefore the dilation cost of the embedding is 3. Figure 4.1 show a $4 \times 3 \times 2$ mesh on a $(4,3)$-arrangement graph.

Table 4.1: Coordinate Ranks (C. R) of nodes in $A_{5,3}$

C. R	$A_{5,3}$	C. R	$A_{5,3}$	C. R	$A_{5,3}$	C. R	$A_{5,3}$	C. R	$A_{5,3}$
111	123	211	213	311	312	411	412	511	512
112	124	212	214	312	314	412	413	512	513
113	125	213	215	313	315	413	415	513	514
121	132	221	231	321	321	421	421	521	521
122	134	222	234	322	324	422	423	522	523
123	135	223	235	323	325	423	425	523	524
131	142	231	241	331	341	431	431	531	531
132	143	232	243	332	342	432	432	532	532
133	145	233	245	333	345	433	435	533	534
141	152	241	251	341	351	441	451	541	541
142	153	242	253	342	352	442	452	542	542
143	154	243	254	343	354	443	453	543	543

Suppose that all the vertices $u_0, u_1, ..., u_{n!/(n-k)!-1}$ in $A_{n,k}$ have been ordered such that $u_k \prec u_j$ if $k < j$. For each node u_p, let x_i be $\left\lfloor \frac{p}{(i-1)!/(n-k)!} \right\rfloor \bmod i$, $n-k+1 \leq i \leq n$, and $0 \leq x_i \leq i-1$. Thus, each vertex u_p, $0 \leq p \leq (n!)/(n-k)!-1$, is associated with

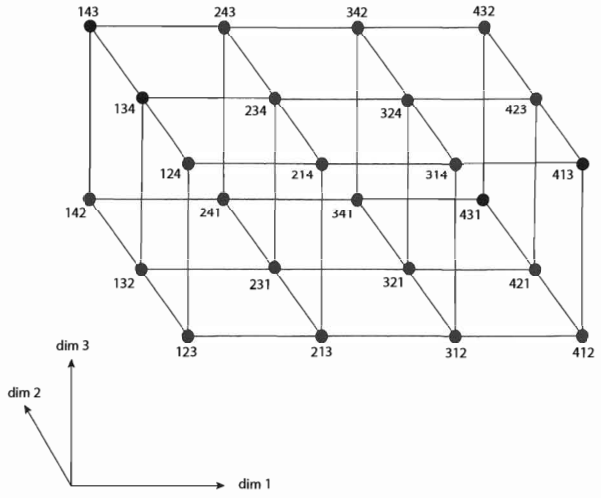


Figure 4.1: $4 \times 3 \times 2$ mesh on $A_{4,3}$

k unique values x_i, $n-k+1 \leq i \leq n$. The coordinate rank of node u_p in $A_{n,k}$ can be represented as the address $x_n x_{n-1} ... x_{i+1} x_i x_{i-1} ... x_{n-k+1}$. In this representation, each x_i, $n-k+1 \leq i \leq n$, is a coordinate of the k-dimensional mesh. We call x_i the i^{th} coordinate of u_p. Table 4.1 shows the nodes on a $(5,3)$-arrangement graph and their coordinate rank.

From the result of the **Constant Routing II**, we can see that any i nodes with coordinates $x_n x_{n-1} ... x_{i+1} x_i x_{i-1} ... x_{n-k+2} x_{n-k+1}$, $x_i = 0, 1, 2, ..., i-1$, are "connected" in a linear array of length i in $A_{n,k}$ after a constant time routing, $n-k+1 \leq i \leq n$. We can see these i nodes form a column along dimension i of the k-dimensional mesh. In $A_{4,3}$, the coordinate ranks 122, 222, 322, 422 are nodes 134, 234, 324, 423 which are connected in a linear array along dimension 1.

4.5.2 Translation and Reversing

Suppose that all the vertices $u_0, u_1, ..., u_{n!/(n-k)!-1}$ in $A_{n,k}$ are in lexicographical order and each node compute its corresponding coordinate rank. Given some integer s, vertex u_p has to send its message to u_q, where $q = (p+s) \bmod (n!/(n-k)!)$, for all p, $0 \leq p \leq n!/(n-k)!$. This task is referred to as a **translation**. The operation translation is also known as **cyclic shift**.

In Section 4.5.1 we have defined coordinate rank for the (n,k)-arrangement graph. Let coordinate rank of u_p before translation be $x_n x_{n-1}...x_{n-k+2}x_{n-k+1}$ and $y_n y_{n-1}...$ $y_{n-k+2}y_{n-k+1}$ after the translation. The translation problem can be reduced to correcting the value for each coordinate from x_i to y_i. Since we can treat i nodes with coordinates $x_n x_{n-1}...x_{i+1}x_i x_{i-1}...x_{n-k+1}$, $x_i = 0, 1, 2, ..., i-1$, as "connected" in a linear array of length i, a coordinate correction can be obtained after running a sorting algorithm on each column. A sorting algorithm is executed on i^{th} column and the value used in the comparisons are y_i, for all $i = n-k+1, n-k+2, ..., n$. The time of sorting array in size i is $O(i)$. As the result, the running time equals total steps of sorting all the columns which is $O(\Sigma_{i=n-k+1}^{n} i) = O(k(n+(n-k+1))/2) = O(nk)$.

Suppose that the element in $A_{n,k}, u_p$, is represented as the coordinate rank, $0 \leq p \leq n!/(n-k)!-1$. The information of node u_p wants to move to the vertex u_q where $q = n!/(n-k)!-p$. This problem is called **reversing**. Similar to the translation, let $x_n x_{n-1}...x_{n-k+2}x_{n-k+1}$ be the original coordinate rank and $y_n y_{n-1}...y_{n-k+2}y_{n-k+1}$ the coordinate rank after reversing. We can use the same method used in translation to correct each column in each dimension. The time complexity for the reversing is the same as translation which is $O(nk)$. Reversing is needed when we want to merge two

sorted sequences of the same direction, that is, reversing one sequence into opposite direction then apply merge. Such a merging algorithm will be discussed in the next chapter.

Generally, we can see **Translation** and **Reversing** as the same type of algorithm. We can sort coordinate from left to right and from right to left. These kinds of algorithms are similar to the usual ASCEND-DESCEND algorithms for the hypercube [46]. The general case is given as follows:

Procedure ASCEND$(A_{n,k})$

for $i = (n-k+1)$ **to** n **do**

OPER $(x_n x_{n-1}...x_{i+1}0x_{i-1}...x_{n-k+1},$

$x_n x_{[n-1]}...x_{i+1}1x_{i-1}...x_{n-k+1},$

$...,$

$x_n x_{n-1}...x_{i+1}(i-1)x_{i-1}...x_{n-k+1})$

where OPER could be any computation on i elements on the linear array of length i. For the dual algorithm DESCEND, the main loop of the algorithm is changed to run from n to $(n-k+1)$. In our cases, Translation and Reversing, the OPER is sorting. These are two similar algorithms using same idea. In these algorithms, each vertex has a record which includes a non-empty destination. We can use the ASCEND or DESCEND to match the coordinate from the original vertex by the coordinate of the destination vertex. Since the size of the longest linear array is n, each OPER can be done in $O(n)$, the running time is sum of all the OPER which is $O(nk)$.

Chapter 5

Algorithms

5.1 Introduction

In this chapter, we present several basic algorithms that are fundamental to designing parallel algorithms on $A_{n,k}$. The algorithms presented here are prefix sums computation, sorting, and computing convex hull on the arrangement graph.

The prefix sums algorithm uses the constant routing algorithms introduced in last chapter. We will present two sorting algorithms, both of them are based on "mesh" embedding property of $A_{n,k}$. As a part of sorting algorithm, a merging algorithm is also presented. Finally, a convex hull algorithm, an important algorithm in computational geometry, on the arrangement graph is presented which will use almost all algorithms we designed in this thesis.

5.2 Prefix Sums Computation

Once again, for the purpose of computing prefix sums, we define the processor ordering $\prec$ lexicographically, that is, $123\cdots(k-1)k \prec 123\cdots k(k-1) \prec ... \prec n(n-1)\cdots(n-k+1)$.

An $O(k \log n)$ time algorithm for computing all prefix sums on $A_{n,k}$ with respect to the processor ordering of $\prec$, using procedure Group-Exchange, is given below.

Suppose that we have computed prefix sums for two groups of sub-structures as follows:

$$\begin{array}{lllll} \textit{Group } 1: & 1_1, & 2_1, & \cdots, & l_1 \\ \textit{Group } 2: & (l+1)_1, & (l+2)_1, & \cdots, & (2l)_1 \end{array}$$

and that each processor holds two variables s and t, for storing the partial prefix sum computed so far with respect to the substructure it is in and the total sum of values in the group it is in, respectively. Let the total sum in Group 1 be t_1 and the total sum in Group 2 be t_2. We first use Group-Exchange to send t_1 to every processor in Group 2, and t_2 to every processor in Group 1, then the prefix sums in processors in Group 1 remain the same, while the prefix sum s in a processor in Group 2 becomes $s \otimes t_1$. The total sum for all the processors in both groups becomes $t_1 \otimes t_2$. All these steps can be accomplished in $O(1)$ time. When a group contains only one $A_{n-1,k-1}$, the algorithm is called recursively. Once again, the recursion stops at $k = 1$ when the prefix sums on $A_{n-k+1,1} = K_{n-k+1}$, are computed in $O(\log n)$ time. This leads to a running time of $O(k \log n)$ for the prefix sums algorithm, which is optimal in view of the $O(k \log n)$ lower bound. It is straightforward to state the algorithm formally.

However, care must be taken since n is not necessarily a power of 2.

As discussed in Section 3.2.2, this prefix sums algorithm implies yet another $O(k \log n)$ broadcasting algorithm. In addition, **Interval Broadcasting** is a special case of prefix sums. In $A_{n,k}$, m vertices are marked as leaders $l_1, l_2, ..., l_m$, where $l_i < l_j$ if $i < j$, and $m \leq n!/(n-k)! - 1$. In terms of the processor ordering (lexicographical ordering), each leader node l_i has to broadcast its information to all the nodes between l_i and l_{i+1}. Interval broadcasting can be solved by running prefix sums. In this special case of prefix sums, initially each leader node holds an index 1 as well as its information, and each non-leader node sets its index to 0 and a blank message. The binary associative operation of the prefix sums is as follows: the node with index 0 is assigned to the maximum of two indices and copies the message from the node with larger index.

It is easy to see that the lower bound of interval broadcasting is the same as prefix sums on an (n, k)-arrangement graph. Since the problem of prefix sums can be solved in $O(k \log n)$, the running time of interval broadcasting is also $O(k \log n)$. It can be reduced to the broadcasting problem by having only one leader and one interval.

5.3 Sorting Algorithms

We will develop two sorting algorithms for the arrangement graph in this section. The first one is based on a sorting algorithm for the star graph by Menn and Somani [39] while the second one is based on a sorting algorithm for the multi-dimensional mesh by Kunde [34].

To describe our sorting algorithms, we will first briefly describe the algorithm by

Table 5.1: Vertices of S_4 with reverse lexicographical order

4321	3421	4231	2431	3241	2341
4312	3412	4132	1432	3142	1342
4213	2413	4123	1423	2143	1243
3214	2314	3124	1324	2134	1234

Menn and Somani. This will offer us a good idea about why their, and subsequently Kunde's algorithm, can be adapted to run efficiently on the arrangement graph.

Let $S_{n-1}(i)$ be a sub-graph of S_n induced by all the nodes of the form $*i$, for some $1 \leq i \leq n$. It can be seen that $S_{n-1}(i)$ is an $(n-1)$-star defined on symbols $\{1, 2, \cdots, n\} - \{i\}$. Thus, S_n can be decomposed into n S_{n-1}'s: $S_{n-1}(i)$, $1 \leq i \leq n$ [3]. For example, S_4 in Figure 1.5 contains four 3-stars, namely $S_3(1)$, $S_3(2)$, $S_3(3)$, and $S_3(4)$, respectively.

For the star sorting algorithm, the processors are ordered in a reverse lexicographical order, that is, $n(n-1)(n-2)\cdots 321 \prec (n-1)n(n-2)\cdots 321 \prec \ldots \prec 213\cdots(n-1)n \prec 123\cdots(n-1)n$.

If we arrange all the vertices in S_n into an $n \times (n-1)!$ array in the row-major order (in terms of the processor ordering), then row i becomes $S_{n-1}(i)$ [39]. The vertices in S_4 are given in Table 5.1. From the processor ordering, we can see that all the vertices in the same column of the $n \times (n-1)!$ array (Table 5.1) have the same rank in their respective S_{n-1}'s. For example, vertices 2431, 1432, 1423, and 1324 are all ranked third in $S_3(1)$, $S_3(2)$, $S_3(3)$, and $S_3(4)$, respectively.

In S_n, if we exchange the 1^{st} symbol with the n^{th} one in each vertex, we get

Table 5.2: Vertices of S_4 after exchange

1324	1423	1234	1432	1243	1342
2314	2413	2134	2431	2143	2341
3214	3412	3124	3421	3142	3241
4213	4312	4123	4321	4132	4231

another $n \times (n-1)!$ array (Table 5.2) in which, by the definition of S_n, each column is connected to form a simple path, i.e., a linear array of processors [39]. Therefore, we may consider the vertices in each column of S_n (whose vertices are arranged in an $n \times (n-1)!$ array in row-major order) as "connected" in a path directly without this constant time transformation.

Given a sequence of elements stored in a set of processors, with each processor holding one element, we say that the sequence is sorted in 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 $\Omega((n!)\log(n!))$ number of steps required to sort $n!$ numbers sequentially implies an $\Omega(\log(n!)) = \Omega(n \log n)$ lower bound to sort on S_n. Sorting on S_n was first studied in [39] where an $O(n^3 \log n)$ time algorithm was given. This algorithm is based on a sorting algorithm (called *Shear Sort*) in [51] for a mesh-connected parallel computer, and is outlined below. In it, we denote by D the direction of the final sorted sequence, where D can be either F or R. We use $\bar{D}$ to denote the direction opposite to D. Also, whenever we are considering a k-star, $2 \leq k \leq n$, we always think of it as arranged in a $k \times (k-1)!$ array in a row-major order. See Table 5.1 for an example.

Procedure n-Star Sort (D)

- 1. **in parallel** sort all the odd numbered rows in the direction F and all the even numbered rows in the direction R recursively.
- 2. **for** $j = 1$ **to** $\lceil \log n \rceil$ **do**
    1. Starting with row 1, arrange all rows into groups of 2^j consecutively numbered rows (the last group may not have all 2^j rows).
    2. **in parallel** sort the columns within each group of rows in the direction D.
    3. **in parallel**
        (a) sort the rows in odd-numbered groups by calling FTG (D);
        (b) sort the rows in even-numbered groups by calling FTG ($\bar{D}$). □

Procedure FTG is defined as follows:

Procedure FTG (D)

- 1. **If** this is not a 1-star, **do** Step 2 and 3, **else** return;
- 2. **in parallel** sort all columns in direction D;
- 3. **in parallel** sort all rows with FTG (D). □

In procedure n-Star Sort, each iteration of Step 2 is the merging process. It merges two adjacent groups of sorted S_{n-1}'s. From the above algorithms we can see that sorting or merging on S_n is reduced to sorting on the columns. Since each

column is connected as a linear array, the *odd-even transposition sort* [32] can be applied. This means that given two sorted sequences stored in two groups of S_{n-1}'s:

$$A: \ S_{n-1}(i), \ S_{n-1}(i+1), ..., \ S_{n-1}(j)$$
$$B: \ S_{n-1}(j+1), \ S_{n-1}(j+2), ..., \ S_{n-1}(l)$$

$i < l$, (A and B do not necessarily contain the same number of S_{n-1}'s), such that the two sequences are in opposite directions, they can be merged into a sorted sequence stored in

$$S_{n-1}(i), S_{n-1}(i+1), \cdots, S_{n-1}(j), S_{n-1}(j+1), \cdots, S_{n-1}(l),$$

in either direction as follows. We first view S_{n-1}'s in A and B as an $(l-i+1)\times(n-1)!$ array. In the first step, each column of length $l-i+1$ is sorted. Then procedure FTG is applied to each row $S_{n-1}(t)$, $i \leq t \leq l$. The latter is considered as an $(n-1)\times(n-2)!$ array, in which each row is an $(n-2)$-star, and each column of length $(n-1)$ is sorted. Now, procedure FTG is applied to each row $S_{n-2}(km)$ (an $(n-2)$-star whose nodes all have km as their last two symbols), $1 \leq k \leq n$ and $k \neq m$, $i \leq m \leq l$. The latter is considered as an $(n-2) \times (n-3)!$ array, in which each row is an $(n-3)$-star, and each column of length $(n-2)$ is sorted. This process is repeated until a 1-star is reached. As we can see, the merging is done by sorting on linear arrays of length $l-i+1$, $n-1$, $n-2$, ..., 2, 1. Thus the total time is $(l-i+1)+(n-1)+(n-2)+\cdots+2$ $= O(n^2)$.

Let $t(n)$ be the time to sort $n!$ elements on S_n, then

$$t(n) = t(n-1) + \lceil \log n \rceil \times O(n^2) = O(n^3 \log n).$$

Obviously, this performance is far from the lower bound $\Omega(n \log n)$.

To adapt this algorithm to run on the arrangement graph, we examine whether the graph has similar structure as the star in terms of the sorting algorithm. As defined before, the processors are ordered in lexicographic order. Therefore, all processors in i_1 precede processors in j_1 for $j > i$. We can arrange all processors in $A_{n,k}$ as an n by $(n-1)!/(n-k)!$ array such that rows are 1_1, 2_1, ..., n_1 and processors in each row are also ordered. Now consider each column. For any two nodes $i* \in i_1$ and $(i+1)* \in (i+1)_1$ with the same rank in their corresponding $A_{n-1,k-1}$, if the node in i_1 is $ia_2 \cdots a_s(i+1)a_{s+2} \cdots a_k$, then the node in $(i+1)_1$ with the same rank must be $(i+1)a_2 \cdots a_s i a_{s+2} \cdots a_k$. Similarly, if the node in i_1 is $ia_2 \cdots a_k$ (i.e., (i+1) is not there), then the node in $(i+1)_1$ with the same rank must be $(i+1)a_2 \cdots a_k$. This is so because there is no integer between i and $i+1$. This observation shows that any two neighbouring nodes in each column is connected by a path of length either 1 or 3. So like the star graph, each column can be considered as a linear array. This shows that the star sorting algorithm can run on the arrangement graph directly with time

$$t(n,k) = t(n-1,k-1) + \log n \times O(nk) = O(k^2 n \log n).$$

Note that $A_{p,1}$ is a p-clique K_p, which can be sorted in $O(\log p)$ time.

Since each iteration of Step 2 of sorting procedure is the merging process, as the result of adapting the algorithm, we can **merge** two adjacent groups of sorted $A_{n-1,k-1}$'s (in opposite directions) in $O(nk)$ time. In addition, given three groups of $A_{n-1,k-1}$, A, B, and C defined on $A_{n,k}$, C is $Merge(A,B)$. Suppose each element after merging into C knows its original rank in A or B, the problem of **unmerging** is to permute the list to move each element in C back to its original vertex. It can be solved by running the merging algorithm in reverse order using the given rank

information. The operation unmerging also takes $O(nk)$.

In [34], an efficient sorting algorithm was presented for sorting on a multi dimensional mesh. For a r-dimensional mesh of dimensions $n_1 \times n_2 \times \cdots \times n_r$, its running time is $O(n_1 + n_2 + \cdots + n_r)$. With the lexicographical processor ordering and the discussion above, for $A_{n,k}$, we know that each column can be viewed as a linear array. Since each row is an $(n-1, k-1)$-arrangement graph, this property holds recursively. Therefore, an (n,k)-arrangement graph can be viewed as an $n \times (n-1) \times \cdots \times (n-k+1)$ mesh (We have shown this embeding property in Section 4.5), thanks to our routing algorithm developed in Section 4.3. Fig. 4.1 shows the nodes of $A_{4,3}$ as organized in a $4 \times 3 \times 2$ mesh. Note that the connection between any two neighbouring nodes is through the routing from Section 4.3. Thus, Kunde's algorithm implies an $O(n+(n-1)+\cdots+(n-k+1)) = O(nk)$ sorting algorithm on the arrangement graph $A_{n,k}$. Of course, the processor ordering for the arrangement graph has to be the same as that defined on the corresponding mesh. An $O(n^2)$ sorting algorithm on S_n based on a similar idea was given in [7].

5.4 Convex Hull on the (n,k)-Arrangement Graph

Given an input size of $N = n!/(n-k)!$ planar points, initially distributed in an arbitrary fashion, one point per processor, we first sort the points by their x-coordinates. Recall that an $A_{n,k}$ can be partitioned into n $A_{n-1,k-1}$'s. Therefore, n disjoint convex hull of $(n-1)!/(n-k)$ can be found recursively in parallel. Then we merge these hulls repeatedly into a final convex hull.

Procedure CONVEXHULL on $A_{n,k}$

1. **do in parallel for** $1 \leq i \leq n$: CONVEXHULL on $A_{n-1,k-1}$

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

   (a) Starting with row 1, arrange all rows into groups of 2^j consecutively numbered rows.

   (b) for all the groups do **in parallel**: merge two convex hulls within the group.

□

We can see that all the Step 2 does in the procedure is merging. The merging is discussed below. Let hull(A) and hull(B) be two disjoint convex hulls of two sets of points A and B. From Figure 5.1 hull(A) and hull(B) are merged into hull($A \bigcup B$) by finding two common tangent lines between hull(A) and hull(B).

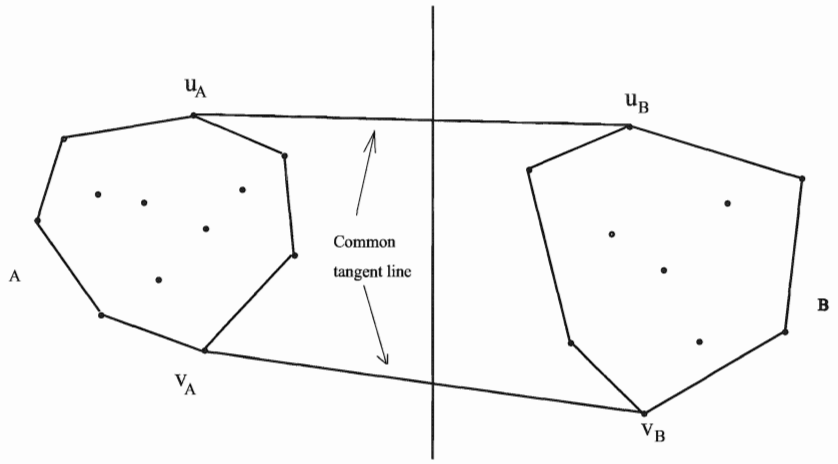


Figure 5.1: Two common tangent lines between two convex hulls

We call the points on the hull **extreme points** and an **edge** is connected by two neighbouring extreme points. We also define an edge e_A as an external edge in hull(A) if e_A belongs to both hull(A) and hull($A \bigcup B$); e_A is internal edge in hull(A)

if e_A belongs to hull(A) but not hull($A \bigcup B$). There are two important observations which are used to indicate if an edge is external edge or internal edge and to find a common tangent line.

Observation 4 *If hull(A) and hull(B) are in the same half-plane bounded by the edge, the edge is a external edge.*

Proof: By the definition of the convex hull. □

Observation 5 *If an extreme point is shared by an internal edge and an external edge, the extreme point is on the common tangent line.*

Proof: By the definition of internal edge and external edge, the extreme point p_A shared by an internal edge and an external edge of hull(A) is a extreme point which belong to hull($A \cup B$). In the hull($A \cup B$), two edges share p_A. One is the external edge of Hull(A). Since an internal edge of hull(A) is not a edge of hull($A \cup B$),p_A has to cut off the internal edge and connect to an edge, $e_{A,B}$, in hull($A \cup B$) shared by another extreme point of hull(B), p_B, which has the same situation as p_A. Since $e_{A,B}$ is a boundary edge of hull($A \cup B$), hull(A) and hull(B) are in the same half-plane bounded by the edge, $e_{A,B}$ is a common tangent line of hull(A) and hull(B). □

By the two observations, from Figure 5.2 we can clearly see that e_1 is an external edge since all points in hull(A) and hull(B) are in the same half-plane. We notice that e_2 cuts hull(B) and the points are in different half-plane, e_2 therefore is an internal edge. Also for the extreme point v which is shared by e_1 and e_2, where e_1 is an external edge and e_2 is an internal edge, we can see that v is on the upper common tangent line.

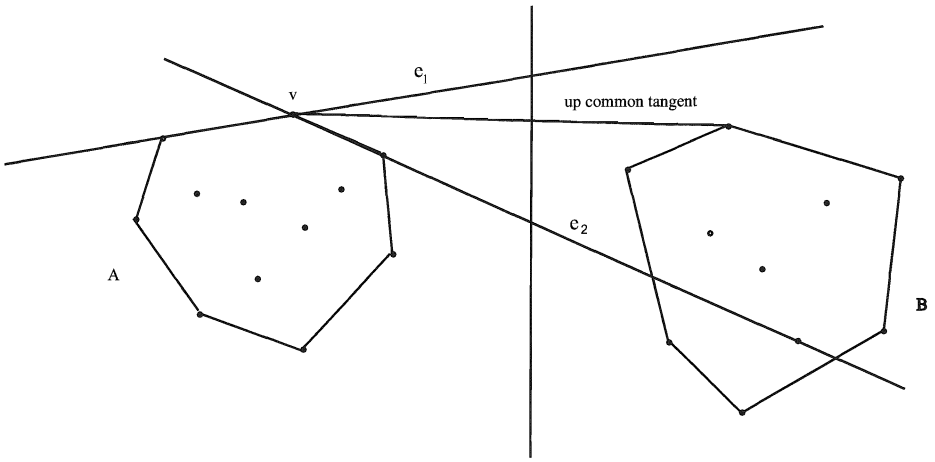


Figure 5.2: External edge and Internal edge

However, instead of testing all the vertices of hull(B) with an edge e in hull(A), we can only test two representatives that are the nearest and farthest extreme points from hull(B) to e. The following definitions and a procedure Extremal Search are used to find these two representatives, all angles are measured with respect to the x-axis.

First of all, We want to give the definition of the cousins as follows:

Definition 13 *The **cousins** of $a \in A$ in B are two consecutive elements b_1 and b_2 in B, such that a lies between b_1 and b_2 in the sorted list $A \bigcup B$ resulting from merging A and B.*

By the definition of cousins, the cousins in B of each element in A can be determined by running merging and interval broadcasting in $O(nk)$ time on the (n, k)-arrangement graph.

Definition 14 *The distance of a point to an oriented edge p is the distance from the point to a line containing p; if the point is to the left (alternatively, right) of p, then*

the distance is said to be positive (alternatively, negative).

Definition 15 *The α-distance of a point to an edge p is its distance to the edge p' obtained by rotating p by the angle α in a counterclockwise direction around a point.*

Let A and B be two convex hulls in the plane, each containing $O(m(n-1)!/(n-k)!)$ edges stored in groups of m $A_{n-1,k-1}$'s, $n-k+1 \leq m \leq n-1$, given in counterclockwise order. An extremal search problem $ES(A, B, \alpha)$ is described as follows: For each edge $p \in A$ find a vertex $v_p \in B$ with the smallest α-distance to p among vertices from B. v_p is called an associated point of p in direction α. It is easy to see that for $\alpha = 0$ v_p is of the smallest distance from p among vertices of B and for $\alpha = \pi$ v_p is of the greatest distance from p among vertices of B.

Proposition 4 *Let $s(e)$ denote the angle of an edge e. The associated point $v_p \in B$ (in direction α) of an edge $p \in A$ belongs to an edge $p' \in B$ such that $|s(p)+\alpha-s(p')|$ is minimized on B.*

The **Proposition 4** of associated points shows that the associated point of an edge in A belongs to an edge that is its *cousin* in B.

Now we give the procedure $ES(A, B, \alpha)$ as follows:

Procedure $ES(A, B, \alpha)$

1. **In parallel**, increase the angles of edges of A by α.

2. Let the edges with minimal angles in A and B be the first vertices of corresponding groups of $A_{n-1,k-1}$'s by running translation.(we can do a prefix sums first to determine how many translations need to do.)

3. Merge A and B into C (needs to reverse B first)

4. Running an interval broadcasting to determine the cousins of every edges in A

5. Unmerge C back to A and B □

In the procedure $ES(A, B, \alpha)$, Step 1 runs in constant time. In Step 2, translation requires $O(nk)$ and the prefix sums requires $O(k \log n)$ time, therefore the total running time is $O(nk)$. Since both edges in A and B are sorted in increasing order, the merging and reversing both requires $O(nk)$ time. As we discussed in Section 3.4 and Section 5.3, interval broadcasting and unmerge take $O(k \log n)$ and $O(nk)$ time. As a result, after we add up all the steps, the procedure $ES(A, B, \alpha)$ runs in $O(nk)$ time.

For every edges in hull(A), procedure $ES(hull(A), hull(B), 0)$ will be executed first to find the point with the smallest distance in hull(B). Then let $\alpha = \pi$, every edge can know the greatest distance point in hull(B). After $O(1)$ time, each edge in hull(A) can determine if it is external edge or internal edge and find out four extreme points which can form two common tangent lines. We can see that the merging procedure to merge two groups takes $O(nk)$ time and it is repeated $O(\log n)$ times.

For the procedure CONVEX HULL, the recursion stops at $k = 1$ when $A_{n-k+1,1}$ is a clique K_{n-k+1}. Convex hull problem with n nodes can be solved for EREW PRAM with n processors in $O(\log n)$ time [41]. An algorithm designed for EREW PRAM can be simulated on a clique with cost $O(\log n)$ [8]. Therefore, the base case of finding convex hull on $A_{n,k}$ requires $O(\log^2(n - k + 1))$ time. Let $t(n, k)$ be the time to find the convex hull of $n!/(n - k)!$ planar points on $A_{n,k}$, then

$$t(n, k) = t(n - 1, k - 1) + O(nk \log n) = O(k^2 n \log n)$$

Therefore the convex hull of $n!/(n-k)!$ planar points can be found on $A_{n,k}$ in $O(k^2 n \log n)$ time.

We have designed a parallel algorithm for computing the convex hull of $n!/(n-k)!$ planar points on $A_{n,k}$ and the running time is $O(k^2 n \log n)$. Since the convex hull problem has a lower bound of $\Omega(N \log N)$ and can be solved sequentially in $O(N \log N)$ time, on our parallel interconnection network, the lower bound is $\Omega(\log N) = \Omega(\log(n)!/(n-k)!) = \Omega(k \log n)$. Our result is far from the lower bound. However, since we have to use the merging operation, this performance matches our sorting algorithm on $A_{n,k}$ which is adapted from n-star using the idea of Menn and Somani in Section 5.3.

Chapter 6

Conclusions

In this thesis, we have studied the (n, k)-arrangement graph (interconnection network) which is proposed as an attractive alternative to the n-star network. We found some useful topological properties of the (n, k)-arrangement graph. We also designed several parallel algorithms for this network. In particular, we studied:

- finding cyclic properties for $A_{n,k}$, which allows us to view the source node together with all its neighbours as a de facto complete graph in the sense that any two nodes are connected by a path of constant length;
- developing a single-port neighbourhood broadcasting algorithm for the (n, k)-arrangement network; as a result of this neighbourhood broadcasting algorithm, we designed a broadcasting algorithm. Both these two algorithms are optimal;
- developing a Constant Routing I algorithm for exchanging the contents of the processors in two sub-graphs; we then build Constant Routing II which extend Constant Routing I to exchange contents between two groups of disjoint sub-

graphs;

- designing a broadcasting algorithm which is based on Constant Routing I/II; this algorithm is different from the one we discussed before and it is also optimal.

- embedding properties of the (n, k)-arrangement graph; we embed an $(n-k+1) \times (n-k+2) \times ... \times (n-1) \times n$ mesh into $A_{n,k}$ with unit expansion and dilation 3 and a 2-dimensional mesh;

- prefix sums algorithm for the (n, k)-arrangement graph;

- sorting and merging algorithm for the (n, k)-arrangement graph; we discussed two sorting algorithms; one based on the 2-dimensional mesh embedding properties of the network and Shear Sort; another one is based on multi-dimensional mesh structure; neither algorithm is optimal;

- discussing ASCEND-DESCEND algorithms; in particular we designed Translation and Reversing algorithms based on ASCEND-DESCEND idea from the hypercube;

- designing a basic computational geometry algorithm: convex hull algorithm on the arrangement graph.

So far, only a few algorithms have been developed for the arrangement graph. We would like to find algorithms for solving more problems on the network in the future. Some of them are listed below:

- Designing some basic algorithms such as broadcasting, prefix sums, etc for the all-port model.

- The $\Omega((\frac{n!}{(n-k)!})\log(\frac{n!}{(n-k)!}))$ number of steps required to sort $n!/(n-k)!$ numbers sequentially implies an $\Omega(\log(n!/(n-k)!)) = \Omega(k \log n)$ lower bound to sort on $A_{n,k}$ in parallel. Although we have designed a sorting algorithm which has a time complexity $O(nk)$, it still does not reach the trivial lower bound. Thus one open problem is to improve the sorting algorithm on the (n,k)-arrangement graph.

- In this thesis, some basic data permutation problems, translation and reversing, run in $O(nk)$ time. It remains to find a solution for them to be achieved in $O(k \log n)$ time, or better, in $O(n)$ time.

- The convex hull problem can be solved in $O(N \log N)$ sequentially. The lower bound on the (n,k)-arrangement graph is $\Omega(k \log n)$ in parallel. Our convex hull algorithm does not reach the trivial lower bound. However, a faster merging algorithm would immediately improve a faster convex hull algorithm. We believe this could be achieved.

As shown in this thesis, as well as other research work on the (n,k)-arrangement interconnection network, we know that the (n,k)-arrangement graph is indeed an attractive alternative to the popular n-star network. The (n,k)-arrangement graph provides us better flexibility than the n-star in controlling the number of nodes in the network. Some algorithms the (n,k)-arrangement graph such as broadcasting, prefix sums and sorting match the performance of those of algorithms developed for the n-star. In spite of recent research studying on the (n,k)-arrangement graph, much work still needs to be done to make this network a serious competitor to the n-star.

Bibliography

[1] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms.* Massachusetts: Addison Wesley, 1974.

[2] S. Akers and B. Krishnamurthy, "The fault tolerance of star graphs," in 2^{nd} *International Conference on Supercomputing*, San Francisco, May 1987, pp. 270–276.

[3] ——, "A group theoretic model for symmetric interconnection networks," *IEEE Transactions on Computers*, vol. c-38, no. 4, pp. 555–566, 1989.

[4] S. Akl, *The Design and Analysis of Parallel Algorithms.* Englewood Cliffs, NJ, USA: Prentice Hall, 1989.

[5] ——, *Parallel Computations: Models and Methods.* Upper Saddle River, NJ, USA: Prentice Hall, 1997.

[6] S. Akl, K. Qiu, and I. Stojmenvlc, "Data communication and computational geometry on the star and pancake interconnection networks," in *Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing*, 1991, pp. 415–422.

[7] S. Akl and T. Wolff, "Efficient sorting on the star graph interconnection network," *Telecommunication Systems*, vol. 10, pp. 3–20, 1998.

[8] H. Alt, T. Hagerup, K. Mehlhorn, and F. P. Perparata, "Deterministic simulation of idealized parallel computers on more realistic ones," *SIAM J. Comput.*, vol. 16, no. 5, pp. 808–835, 1987.

[9] S. Aluru, N. Futamura, and K. Mehrotra, "Parallel biological sequence comparison using prefix computations," *Journal of Parallel and Distributed Computing*, vol. 63, pp. 264–272, 2003.

[10] D. Avis, "On the complexity of finding the convex hull of a set of points," School of Comput. Sci., McGill Univ., Tech. Rep. FOCS 79.2, 1979.

[11] L. Bai, H. Maeda, H. Ebara, and H. Nakano, "A broadcasting algorithm with time and message optimum on arrangement graphs," *Journal of Graph Algorithms and Applications*, vol. 2, no. 2, pp. 1–17, 1998.

[12] J. C. Bermond, A. Ferreira, S. Pérennes, and J. G. Peters, "Neighborhood broadcasting in hypercubes," *SIAM J. Discret. Math.*, vol. 21, no. 4, pp. 823–843, 2008.

[13] G. Blelloch, "Scans as primitive parallel operations," *IEEE Transactions on Computers*, vol. 38, no. 11, pp. 1526–1538, 1989.

[14] Y. Chen, T. Juang, and Y. Shen, "Multi-node broadcasting in an arrangement graph using multiple spanning trees," in 7^{th} *IEEE International Conference on Parallel and Distributed Systems (ICPADS'00).* IEEE Computer Society Press, Japan, July 2000, pp. 213–220.

[15] W. K. Chiang and R. Chen, "On the arrangement graph," *Information Processing Letters*, vol. 37, pp. 215–219, 1998.

[16] W. Chiang and R. Chen, "The (n, k)-star graph: A generalized star graph," *Information Processing Letters*, no. 56, pp. 259–264, 1995.

[17] M. Cosnard and A. Ferreira, "On the real power of loosely coupled parallel architectures," *Parallel Processing Letters*, vol. 1, pp. 103–111, 1991.

[18] J. D'Angelo and D. West, *Mathematical Thinking: Problem-Solving and Proofs.* Upper Saddle River, NJ: Prentice-Hall, 2000.

[19] K. Day and A. Tripathi, "Arrangement graphs: A class of generalized star graphs," *Information Processing Letters*, no. 42, pp. 235–241, 1992.

[20] ——, "Embedding grids, hypercubes, and trees in arrangement graphs," in *Int. Conf. on Parallel Processing*, no. III, 1993, pp. 65–72.

[21] ——, "Embedding of cycles in arrangement graphs," *IEEE Transactions on Computers*, vol. 42, no. 8, pp. 1002–1006, 1993.

[22] M. Dietzfelbinger, S. Madhavapeddy, and I. H. Sudborough, "Three disjoint path paradigms in star networks," in *Proc. Third IEEE Symp. on Parallel and Distributed Processing*, Dallas, December 1991, pp. 400–406.

[23] M. Flynn, "Some computer organizations and their effectives," *IEEE Trans. Comput.*, vol. C-21, pp. 948–960, 1972.

[24] H. Frank, "The maximum connectivity of a graph," in *Proceedings of the National Academy of Sciences of the United States of America*, vol. 48, no. 7, 1962, pp. 1142–1146.

[25] I. Friš, I. Havel, and P. Liebl, "The diameter of the cube-connected cycles," *Inf. Process. Lett.*, vol. 61, no. 3, pp. 157–160, 1997.

[26] S. Fujita, "Neighbourhood information dissemination in the star graph," *IEEE Transaction on Computers*, vol. 49, no. 12, pp. 1366–1370, 2000.

[27] ——, "Optimal neighborhood broadcast in star graphs," *Journal of Interconnection Networks*, vol. 4, no. 4, pp. 419–428, 2003.

[28] J. Hong, K. Mehlhorn, and A. Rosenberg, "Cost trade-offs in graph embeddings with applications," *J. ACM*, vol. 30, no. 4, pp. 709–728, 1983.

[29] S. Hsieh, G. Chen, and C. Ho, "Fault-free hamiltonian cycles in faulty arrangement graphs," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 10, no. 3, pp. 223–237, 1999.

[30] H.-C. Hsu, T.-K. Li, J. J. M. Tan, and L.-H. Hsu, "Fault hamiltonicity and fault hamiltonian connectivity of the arrangement graphs," *IEEE Trans. Comput.*, vol. 53, no. 1, pp. 39–53, 2004.

[31] J. Jwo, S. Lakshmivarahanm, and S. Dhall, "A new class of interconnection networks based on the alternating group," *NETWORKS*, vol. 23, no. 4, pp. 315–326, 1993.

[32] D. Knuth, *The Art of Computer Programming.* Massachusetts: Addison-Wesley, Reading, 1973, vol. 3.

[33] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms.* Redwood City, California: The Benjamin/Cummings Publishing Company, Inc., 1994.

[34] M. Kunde, "Routing and sorting on mesh-connected arrays," in *Proc.of the Aegean Workshop on Computing.* LNCS 319, 1988, pp. 423–433.

[35] C. Lai and J. Tsay, "Communication algorithms on alternating group graphs," in *Proceedings of the* 2^{nd} *AIZU International Symposium on Parallel Algorithms*, 1997, pp. 104–110.

[36] S. Lakshmivarahan and S. Dhall, *Parallel Computing Using the Prefix Problem.* New York: Oxford University Press, 1994.

[37] J. Li, Y. Xiang, M. Chen, and Y. Zhou, "Broadcasting in (n, k)-arrangement graph based on an optimal spanning tree," in *Proc. of the First Asia International Conference on Modelling and Simulation (AMS 2007).* IEEE Computer Society Press, March 2007, pp. 27–30.

[38] R. Lo and G. Chen, "Embedding longest fault-free paths in arrangement graphs with faulty vertices," *Networks*, vol. 37, no. 2, pp. 84–93, 2001.

[39] A. Menn and A. Somani, "An efficient sorting algorithm for the star graph interconnection network," in *Proc. International Conference on Parallel Processing*, August 1990, pp. 1–8.

[40] R. Miller and Q. Stout, "Computational geometry on hypercube computers," in *Proceedings of the third conference on Hypercube concurrent computers and applications.* New York, NY, USA: ACM, 1988, pp. 1220–1229.

[41] ——, "Efficient parallel convex hull algorithm," *IEEE Trans. Comput.*, vol. 37, no. 12, pp. 1605–1618, 1988.

[42] ——, "Simulating essential pyramids," *IEEE Transactions on Computers*, vol. 37, no. 12, pp. 1642–1648, 1988.

[43] ——, "Mesh computer algorithms for computational geometry," *IEEE Trans. Comput.*, vol. 38, no. 3, pp. 321–340, 1989.

[44] W. Najjar and P. Srimani, "Conditional disconnection probability in star graphs," Department of Computer Science, Colorado State University, Tech. Rep. CS-90-105, 1990.

[45] F. Preparata and M. Shamos, *Computataional Geomentry.* Berlin, Germany: Springer-Verlag, 1985.

[46] F. Preparata and J. Vuillemin, "The cube-connected-cycle: A versatile network for parallel computation," *Comm. ACM*, vol. 24, no. 5, pp. 300–309, 1981.

[47] K. Qiu, "On a unified neighbourhood broadcasting scheme for interconnection networks," *Parallel Processing Letters*, vol. 17, no. 4, pp. 425–437, 2007.

[48] K. Qiu, S. Akl, and H. Meijer, "A parallel sorting algorithm on the star graph," Department of Computing and Information Science, Queens University, Kingston, Ontario, Canada, Tech. Rep. 90-286, 1990.

[49] K. Qiu and S. Das, "A novel neighbourhood broadcasting algorithm on star graphs," in *9^th IEEE International Conference on Parallel and Distributed Systems (ICPADS'02).* IEEE Computer Society Press, Taiwan, 2002, pp. 37–41.

[50] M. Samatham and D. Pradhan, "The de bruijn multiprocessor network: A versatile parallel processing and sorting network for vlsi," *IEEE Transactions on Computers*, vol. 38, no. 4, pp. 567–581, 1989.

[51] I. Scherson and S. Sen, "Parallel sorting in two-dimensional vlsi models of computation," *IEEE Transactions on Computers*, vol. 38, no. 2, pp. 238–249, 1989.

[52] P. Srimani, "Generalized fault tolerance properties of star graphs," Department of Computer Science, Colorado State University, Tech. Rep. CS-90-104, 1990.

[53] G. Toussaint, "Pattern recognition and geomerical complexity," in *Proc 5th Int. Conf. Pattern Recognition*, 1980, pp. 1324–1347.