Sorting on Hypercube and its variants

Dissertation submitted to the Jawaharlal Nehru University in partial fulfillment of the requirement for the award of the degree of

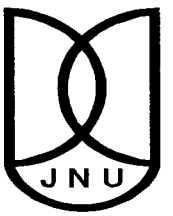
MASTER OF TECHNOLOGY

IN

COMPUTER SCIENCE AND TECHNOLOGY

By

Shalendra Kumar



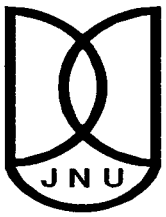
SCHOOL OF COMPUTER AND SYSTEMS SCIENCES

JAWAHARLAL NEHRU UNIVERSITY

NEW DELHI-110067, INDIA

JULY 2008

JAWAHARLAL NEHRU UNIVERSITY
School of Computer and Systems Sciences
New Delhi – 110067, INDIA



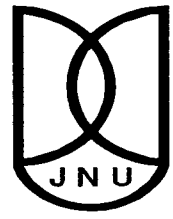
Declaration

I hereby declare the dissertation entitled **"Sorting on Hypercube and its variants"** submitted for the degree of Master of Technology in Computer Science engineering is my original work and the dissertation has not formed the basis for the award of any degree, diploma or similar other title. It has not been submitted to any other University or institute for award of any degree or diploma.

Shalendra Kumar

Shalendra Kumar

JAWAHARLAL NEHRU UNIVERSITY
School of Computer and Systems Sciences
New Delhi – 110067, INDIA



Certificate

This is to certify that **Shalendra Kumar** has carried out his dissertation work entitled "**Sorting on Hypercube and its variants**" as a partial requirement for the award of **Master of Technology** by **Jawaharlal Nehru University**, as a record of his work carried out and completed under my supervision during the academic session **2007-08**. To the best of my knowledge the dissertation has not previously formed the basis for the award of any degree and diploma or similar other titles.

.....8/8/08

Prof. Parimala N.
(Dean of School of Computer and Systems Sciences, JNU)

Prof. C. P. Katti
(Supervisor)

For

My Parents

They added their great contribution and sacrifice for my education especially for higher education.

Acknowledgements

A work becomes easier and done nicely if the working environment is in favour of the work and the person doing the work. Here I am saying this because I have felt it. It starts from this University, from where I am getting the facilities. It provides very knowledgeable, dynamic and world fame teachers who adore this university.

I am thanking my parents, my brothers and my sisters who supported and encouraged me through their voice on my phone. They could not do more than that and I was not expecting more than that from their side. So I am extremely grateful for all of them.

Since last two years or even more than that I am suffering from my health problem continuously but I didn't say anything to anybody other than my classmate friend Mr. Sanjeev and the Doctor in Health Center, JNU. Although it is a big obstacle in my carrier but they cooperated me a lot. So I am thanking them from the bottom of my heart.

Shalendra Kumar

Shalendra Kumar

Contents

List of Figures

Index of tables and Graphs

Abstract

Parallel Computing provides a perfect and high level framework to solve bigger problems with the help of multiple processors. Parallel computing is contributing its great time saving performance. Many scientific and engineering problems are now giving better solution and obviously time saving.

Sorting data is very important activity in the field of computer science. From the beginning many researchers have given many different ways to tackle it easily with perfection. But when single processor computer goes to its highest level of performance than parallel computers come to picture. Because the fast growing industry needs fast access of data in a particular order.

Network topologies like Hypercube, Supercube etc. are good architecture to implement different types of sorting. Most of the sorting algorithms have been implemented on Hypercube. Supercube is better architecture because it can be constructed with any number of processing nodes. It gives better sorting performance on the Supercube. In my dissertation I have discussed how to implement sorting on Supercube.

Chapter 1

1.1 Introduction

In Parallel Computing a single task is divided into multiple smaller subtasks and executes simultaneously on multiple processors to obtain the result faster. Conventionally the software has been written with the concept of *serial computation*:

Serial computation needs:

a. A single Central Processing Unit (CPU).

b. A Process to split a problem into smaller sub problems (P_1, P_2, ...P_{n-1}, P_n).

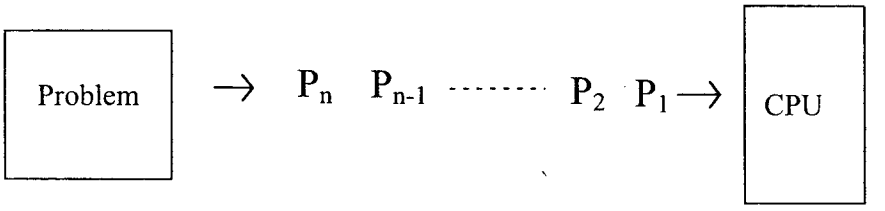


Figure 1.1: Serial execution

Using parallel computing the same problem can be solved in lesser time by splitting and assigning the problems among multiple processors.

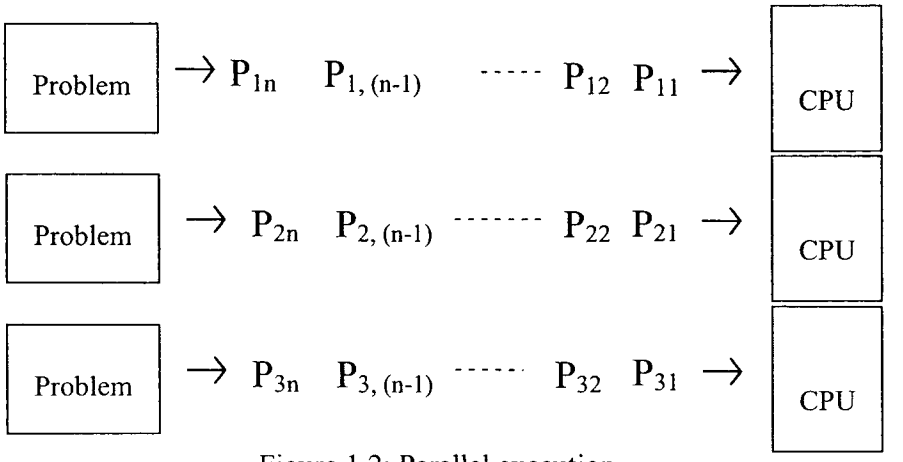


Figure 1.2: Parallel execution

1.2 Program Partitioning Method

Partitioning a program into smaller sub programs is an important issue. The programmer has to figure out how to break the problem into pieces, and has to figure out how the pieces relate to each other. Programmers have to take care that if one processor has already evaluated up to a particular position, then others would not waste time duplicating the effort. On the type of partitioning the parallelism is categorized as:

a. Instruction level parallelism: A computer program is a stream of instructions executed by a processor. These instructions can be re-ordered and combined into groups which are then executed in parallel without changing the result of the program.

b. Bit-level parallelism: Parallelising bits increases the size of word. When word size increases reduces the number of instructions. Example- If an 8-bit processor adds two 16-bit integers, then the processor adds 8 lower-order bits from each integer, when it adds the 8-heigher-order bits, which requests two instruction to accomplish a single a single operation. But a 16-bit processor is able to accomplish the operation in single instruction.

c. Data parallelism: Data parallelism is a parallelism in the loops in a program, which focuses on distributing the data across different computing nodes to be processed in parallel.

d. Task parallelism: In task parellelism entirely different calculations can be performed on either the same or different sets of data.

1.3 Architecture

Various hardware and software architectures are used for parallel computing. At a lower level, it is necessary to interconnect multiple CPUs with some sort of network, regardless of whether that network is printed onto a circuit board or made up of loosely-coupled devices and cables. At a higher level, it is necessary to interconnect processes running on those CPUs with some sort of communication system.

Distributed programming typically falls into one of several basic architectures or categories: Client-server, 3-tier architecture, N-tier architecture, Distributed objects, loose coupling, or tight coupling.

a. Client-server - Smart client code contacts the server for data, then formats and displays it to the user. Input at the client is committed back to the server when it represents a permanent change.
b. 3-tier architecture - Three tier systems move the client intelligence to a middle tier so that stateless clients can be used. This simplifies application deployment. Most web applications are 3-Tier.
c. N-tier architecture - N-Tier refers typically to web applications which further forward their requests to other enterprise services. This type of application is the one most responsible for the success of application servers.
d. Tightly coupled (clustered) - refers typically to a set of highly integrated machines that run the same process in parallel, subdividing the task in parts that are made individually by each one, and then put back together to make the final result.
e. Peer-to-peer - an architecture where there is no special machine or machines that provide a service or manage the network resources. Instead all responsibilities are uniformly divided among all machines, known as peers. Peers can serve both as clients and servers.

1.4 Von Neumann Architecture

Any computer architecture ever built has been rooted in Von Neumann Architecture. This architecture is based in the concept of stored program. A computer with von Neumann architecture having:

a. Arithmetic Logic Unit (ALU), Control Unit, Memory, and I/O devices.

b. A Shared memory with one data bus and one address bus.

c. The program is encoded numerically and stored in the memory along with the data.

d. One instruction and data has to be fetched in sequential order i.e. at one time only one instruction is executed.

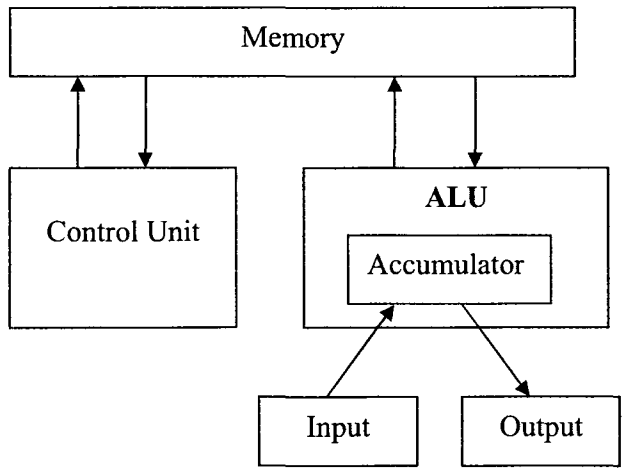


Figure 1.3: Von Neumann Architecture

This architecture is built as a sequential machine executing scalar data, because of this, these machines are slow. Later look ahead techniques were developed to prefetch instructions in order to develop I/E (instruction fetch/ decode and execute) operations and to enable functinal parallelism. Functional parallelism was supported by two approaches: One is to use multiple functional units simultanesously and the other is to practice pipelining at various precessing levels.

1.5 Parallel Computer Memory Architecture

Two categories of parallel computers are architecturally modeled:

1. Shared Memory
2. Distributed Memory

1.5.1 Shared memory:

- There are various types of parallel computers with shared memory, but in general these have the ability for all processors to access all memory as global address space.
- More than one processor can operate independently but share the same memory resources.
- If there is a change of memory location by one processor, it is visible to all other processors.

Shared memory machines can be categorized into two classes on the basis of memory access time: UMA and NUMA.

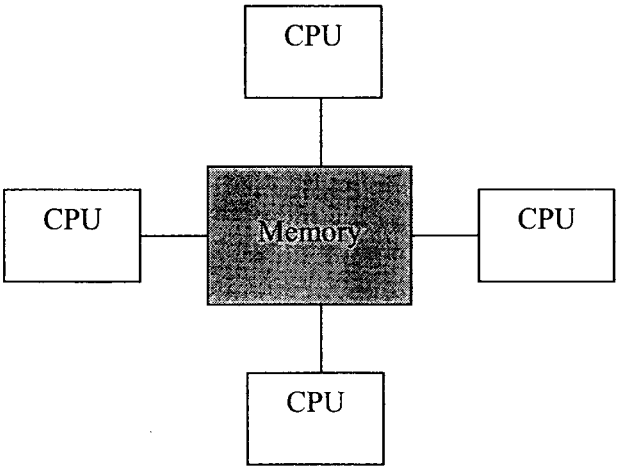


Figure 1.4: Shared Memory Architecture

1. Uniform Memory Access (UMA): It has been represented by Symmetric Multiprocessor (SMP) machine. In a UMA model, the physical memory is uniformly shared by all the processors. All processors have equal access time to all memory. Sometimes it is called CC-UMA (Cache coherent UMA) because if one processor updates a location in shared memory, all the other processors know about the update.

2. Non-Uniform Memory Access (NUMA): It is made by connecting two or more SMPs phyically. One SMP can directly access the memory of another SMP. Access time varies with the location of the memory. Accessing memory across the link is slower.

Advantages:

a. Global address spacing provides user-friendly programming environment.

b. Data sharing between tasks is fast and uniform.

Disadvantages:

a. No scalability between memory and CPUs.

b. Accessing correct global memory is programmer's responsibility for synchronization.

c. Due to a number of processors it is expensive.

1.5.2 Distributed Memory

Distributed system consists of multiple computers interconnected by network. Each node is an autonomous computer consisting of a processor (p), local memory (m), and sometimes attached by I/O devices.

- A distributed memory system needs a communication network to connect inter-processor memory.
- There is no concept of accessing global address space across all processors.
- Each processor has its own local memory so it works independently. If there is any change in its local memory than there is no effect on the memory of other processors. Hence, there is no concept of cache coherence.
- When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated.

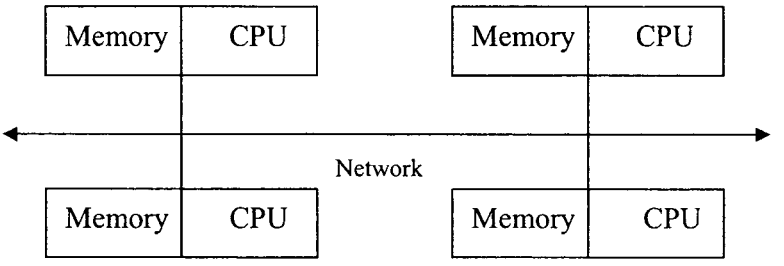


Figure 1.5: Distributed memory architecture

Advantages:

- Memory is scalable, means if the number of processors increases the size of memory increases.
- Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache coherency.
- It is cost effective.

Disadvantages:

- Programmer is responsible about the data communication between processors.
- Difficult to map the existing data structures to this memory organization.

1.6 Flynn's Taxonomy

Flynn's Taxonomy is a concept to classify parallel computers architecture. This concept of classification has been used since 1972. This classification concept is based on instruction and data and it is divided into four categories.

Single Instruction, Single Data (SISD):

- Used in serial computer
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
- Single data: only one data stream is being used as input during any one clock cycle
- Execution is deterministic
- This is the oldest and until recently, the most prevalent form of computer

Single Instruction, Multiple Data (SIMD):

- SIMD is a kind of parallel computer.
- Single instruction: All processing units execute the same instruction at any given clock cycle.
- Multiple data: Each processing unit can operate on different data element.
- SIMD type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity. sInstruction units.
- It is good for specialized problems.
- Execution is synchronous and deterministic.

Multiple Instructions, Single Data (MISD):

- Single data stream is given into multiple processing units.
- Each processing unit operates independently on the data.
- Some uses might be:
    - multiple frequency filters operating on a single signal stream
    - Multiple cryptography algorithms attempting to crack a single coded message.

Multiple Instructions, Multiple Data (MIMD):

- Currently MIMD is the most common type of parallel computer. Modern computers come into this category.
- Multiple Instruction: every processor may execute different instruction stream
- Multiple Data: every processor may work with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Examples: most current supercomputers, networked parallel computer "grids" and multi-processor SMP computers - including some types of PCs.

1.7 Why Parallel Computing needed?

Before knowing the reasons for the requirement of Parallel Computing, limitation of serial computation should be discussed:

Transmission speed: Speed of serial computer is directly dependent on the speed of data so that it can traverse through hardware. Absolute limit of speed of light is 30 cm/nsec and the transmission limit of copper wire is 9 cm/nsec. So if speed increases than it needs increase in speed of processing elements.

Limitation of miniaturization: Processor manufacturing technologies allow increase in the number of transistors to be placed on a chip. However, even with molecular or atomic-level components, a limit will be reached on how small components can be.

Economic limitations: Making single processor faster is quite expensive than using a larger number of moderately fast processors to achieve better performance in lesser expense.

To crossover these limitation we need parallel computing with more benefits.

Parallel computing saves time, solves bigger problems and facilitates concurrency. There are some more reasons:

- Parallel computing provides advantage of non-local resources – using available compute resources on a wide area network, or even the Internet when local compute resources are scarce.
- Parallel computing can use multiple cheap instead of paying for time on a supercomputer, in this sense it saves cost.
- Single computer has finite memory resources, but in parallel computing a large problem can be solved using the memories of multiple computers, in this way it solves the problem of memory constraints.

Since the power of computing need has increased much for different applications than a sequential computer is providing. Parallel computing provides cost effective solution by increasing the number of CPU's in a computer and by adding an efficient communication system with them. In this ways the workload can be shred among different distributed processors or processing elements. So ultimately we get high performance.

Main reasons which influences parallel computing:

1. Computational requirements are ever increasing both in the area of scientific and business computing. The technical computing problems, which require high speed computational power, are related to life sciences, aerospace, geographical information system, mechanical design and analysis etc.
2. Hardware improvements in pipelining, superscalar etc. are non-sharable and require sophisticated compiler technology. Developing such compiler technology is a difficult task.
3. Technology of parallel processing is developed and can be used commercially; there is already significant research and development work on development tools and environment is achieved.
4. Sequential architectures are reaching its physical limitation as they are controlled by the speed of light and thermodynamics laws, speed with which sequential CPU's can operate is reaching to its saturation point and hence an alternative way to get high computational speed is to connect multiple CPU's.
5. Vector processing works well for certain kind of problems. It is suitable for only scientific problems. It is not useful for other areas.

1.8 Concept of Parallel Computing

In serial computing using Von Neumann style machines, there is one processor which executes a series of instructions in order to produce a result, i.e., there is a logical flow of control through the program. At any one time, only one operation being carried out by the processor.

Parallel computing is concerned with producing the same results using multiple processors. The problem to be solved is divided up among a number of processors. There are a variety of ways in which the problem can be divided up among the available

processors. Dividing a problem in a sensible and efficient manner is critical to achieving good performance on a parallel machine.

1.9 Data Communication

Transmission of digital data to devices out side of the message source is called Communications. As a rule, the maximum possible transmission rate of a message is directly proportional to signal power and inversely proportional to channel noise. It is the aim of any communications system to provide the highest possible transmission rate at the lowest possible power and with the least possible noise. As the distance between the source and destination increases, accurate transmission of data becomes difficult. Communication starts though a *communications channel*, it is a pathway over which information can be send. It may be a physical wire that connects communicating devices Information sent through a communications channel has a source and a destination to which the information is delivered.

1.9.1 When communication is needed

Some problems can be decomposed and executed in parallel without sharing the data of the tasks. Example: In image processing operation where every pixel in a black and white image needs to have its color reversed. The image data can be distributed easily into multiple tasks and act independently from each other to do their work, so communication is not needed here.

Many of the parallel applications are not simple and do not require tasks to share data with each other. So where a problem accomplishes by sharing or dividing the problem with different tasks then the communication is needed.

1.9.2 Considerable factors

1. Cost of communication
    a. Inter-task communication implies overhead.
        b. Communication frequently requires some type of synchronization among tasks. This time is in the form of waiting time instead of doing work.

c. Competing communication traffic can saturate the available network bandwidth, again aggravating performance problems.

d. Machine cycles could be used for computation are instead used to transmit data.

2. Bandwidth and Latency

a. Bandwidth is the amount of data that can be communicated per unit of time

b. Latency is the time required to send a minimal message from one point to another point.

c. Sending many small messages can cause latency to dominate communication overheads. Sometime it is more efficient to package small messages into a larger message, thus increasing the effective communication bandwidth.

3. Synchronous and Asynchronous communication

a. Synchronous communication needs "handshaking".

b. Synchronous communications are known as blocking communication because until the communication completes other work must wait.

c. Asynchronous communication allows tasks to transfer data independently.

d. Asynchronous communications are known as non-blocking communication because when the communication is going on other work can take place simultaneously.

4. Communication scope

a. It is very difficult to know that which task is communicating to each other during the design phase of a parallel code.

b. Point-to-point scoping – Among two tasks if one is sender/receiver then other will be receiver/sender.

c. Collective scoping – Here data can be shared between more than two tasks, which are specified as being members in a common group, or collective.

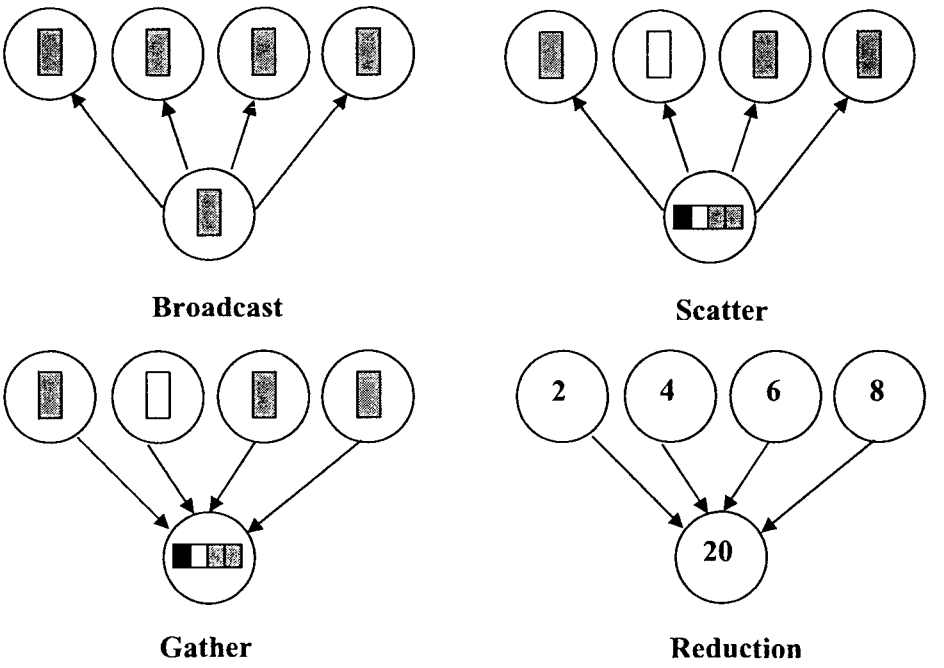


Figure 1.6: Communication Scope

1.10 Communication cost

Message Passing Cost: The time taken to communicate a message between two processing nodes in a network is the sum of the time taken to prepare a message for transmission and the time taken by the message to traverse through the network to its destination.

Commuication latency parameters:

a. Startup time: Time required to make ready to send a message, time taken to execute the routing algorithm, and time taken to establish an interface between the local node and the router. It is denoted by t_s.
b. Per-hop-time: Time taken by the header of a message to travel between two nodes connected directly. It is denoted by t_h.
c. Per-word transfer time: If a network bandwidth allows data to travel r words/sec to travel then time $1/r$ is called per-word transfer time. It is denoted by t_w, and $t_w = 1/r$.

1.10.1 Routing techniques:

a. **Store-and-forward routing:** When a message traverses through a network with multiple links, each immediate node in the path passes the message to next node after receiving and storing the message. If a message of size m travels through a network of l links. At each link the message takes t_h time to for header and $t_w m$ for the

rest the messages to traverse the link. If there are l links then total time is $(t_h + m t_w)l$. So total communication time $t_{comm} = t_s + (t_h + mt_w)l$.

For modern parallel computer t_h can be ignored then $t_{comm} = t_s + mt_w l$.

b. **Packet Routing:** In store and forward routing a message can be sent if the message sent earlier has reached to its destination but in packet routing a message is broken into some smaller and equal part before it is sent, middle node waits for only one parts of the original message to arrive before passing it on. This approach is good for better communication resource utilization. In this way packet routing reduces the communication time.

Packet routing lowers overhead form packet loss and better error correction capability. One more advantage of this concept is free to choosing different paths for packet. But the overhead involved in so that each packet must carry routing and sequencing information.

c. **Cut-through Routing:** If all the data packet is enforced to follow the same path, the overhead of transmitting routing information is eliminated with each packet like in-sequence delivery and sequencing information. If error information is added at message level then the overhead can be reduced that is incurred with error detection and correction method.

In cut-through routing a message is broken into fixed size length called flow control digits. Since the flow control digit do not contains overheads of packets so it can be much smaller packets. Cut-though routing uses less memory and memory bandwidth between intermediate nodes, and it is faster.

If a message is traversing through a network of l links and per-hop-time is t_h, then time taken by the header of the message to reach the destination node is lt_h. If the message length is m words, then entire message arriving time is $m\tau$, after the arrival of the header of the message. Therefore total communication time in cut-through-routing. If the communication time is denoted by t_c then

$$t_c = t_s + lt_h + m^{\tau}.$$

This time is less from the time taken to store and forward routing.

Chapter 2
Network Topologies

A major concern in the design of parallel systems is the set of pathways over which the processors, memories, and switches communicate with each other. These connections define the interconnection network, or topology, of the machine. Attributes of the topology determine how processors will share data and at what cost.

The diameter of a ring grows as more nodes are added, but the diameter of a fully connected network remains the same. On the other hand, a ring can expand indefinitely without changing the degree, but each time a new node is added to a fully connected network a link has to be added to each existing node. *Scalability* refers to the increase in the complexity of communication as more nodes are added.

A scalable topology that has been used in several parallel processors is the hypercube. A line connecting two nodes defines a *1-dimensional* ``cube." A square with four nodes is a *2-dimensional* cube, and a *3-dimensional* cube has eight nodes. This pattern reveals a rule for constructing an *N-dimensional* cube. Doubling the number of nodes in a hypercube increases the degree by only 1 link per node, and in this way increases the diameter by only 1 path.

Communication in a hypercube is based on the binary representation. The nodes are numbered so that two nodes are adjacent if and only if the binary representations differ by one bit. For example, nodes 0110 and 0100 are immediate neighbors but 0110 and 0101 are not. An easy way to label nodes is to assign node IDs as the cube is constructed.

Another desirable property of interconnection networks is *node symmetry*. A node symmetric network has no distinguished node, that is, the view' of the rest of the network is the same from any node. Rings, fully connected networks, and hypercubes are all node symmetric. A tree has three different types of nodes, namely a root node, interior nodes, and leaf nodes, each with a different degree. A star has a distinguished node in the center which is connected to every other node. When a topology is node asymmetric a distinguished node can become a communications bottleneck.

Three important attributes of an interconnection network are: Timing strategy, control strategy, *and* switching strategy.

Since the technologies are being advanced, now it is possible to build large scale parallel computers. One crucial step on designing a large scale parallel computer is to determine

the topology of the network because the system performance is affected by the network topology. A network is an essential component for a large-scale computer. It provides communication among the processors and the memories. There are many types of network topology like, *Hypercube, folded hypercube, Supercube folded supercube* etc.

2.1 Some important terminologies in parallel computing:

Task: A logically high level, discrete, independent section of computational work. A task is executed by a processor as a program.

Synchronization: It refers to the idea that multiple processes are to coordinate simultaneously to ensure correctness so as to commit certain sequence of action without unexpected race conditions.

Diameter: Diameter of a network is the maximum distance between any two processing nodes in the network. The distance between two processing nodes is defined as the shortest path between them. The diameter of a completely connected network is one.

Connectivity: Connectivity of a network is a measure of the multiplicity of paths between any two processing nodes. A network with high connectivity is preferable because it reduces conflict in communication. In other way it is said the minimum number of edges that must be removed from the network to break into two disconnected networks.

Cost: Cost is measured in terms of the number of communication links or the number of wires required by the network. A hypercube-connected network has $(p \log_2 p)/2$ links.

Bisection Width: Minimum number of edges deleted to cut a graph into two sub graphs is called bisection width. *Channel width* is the number of bits that can be communicated simultaneously over a link of connecting two nodes. Number of physical wires in each communication link is equal to channel width. Rate of transfer of bits through a physical wire is called channel rate. The peak rate at which data can be communicated between the ends of a communication link is called *channel bandwidth.* If *channel bandwidth* is denoted by *B*, *channel rate* is denoted by *b* and *channel width* is denoted by w then these are related as

*Channel bandwidth = channel rate * Channel width*

$$B = bw$$

This relation shows that the bisection width provides a good indicator of the maximum communication bandwidth along the bisection of a network.

Speedup: It is the ratio of time taken to execute a task sequentially and time taken to execute the same task in parallel.

Recursive Decomposition: Recursive decomposition comes into picture for inducing concurrency in problems that can be solved using the *divide and conquer* rule. In this rule, a problem is solved by first dividing into a set of independent subproblems. Each of these subproblems is solved recursively, applying the same division process into smaller subproblems following their results.

Hamming Distance: Hamming distance is the number of position differs between two binary sequences. If u and v are two binary sequences to represent two distinct nodes, u is represented by 0000 and *v* is represented by 0001 then the Hamming distance is 1, because the third bit means the least significant bits are different. In other words *HD (u, v)* is the bitwise XOR of *u* and *v*.

Performance factors in interconnection network:

Bandwidth: It is the width of communication link through the data passes. It is measured in terms of megabyte.

Hardware complexity: This refers to the implementation cost such as those for wires, switches, connectors, arbitration and interface logic.

Network latency: This refers to the worst case time delay for a unit message to be transferred through the network.

Functionality: It tells how the network supports data routing, interrupt handling, synchronization, request message combining and coherence.

Scalability: It refers how much a network is scalable with what cost to make the network bigger and easier to perform an operation.

2.2 Hypercube

Hypercube is loosely coupled parallel processors. This is the first type of architecture consists of a large number of identical processors interconnected to one another. There is no shared memory and no global synchronization. In hypercube communication is achieved by data passing and computation is performed by transferring data from one

processor to another by traveling across a sequence of nearest neighbor nodes starting with first and ending with second.

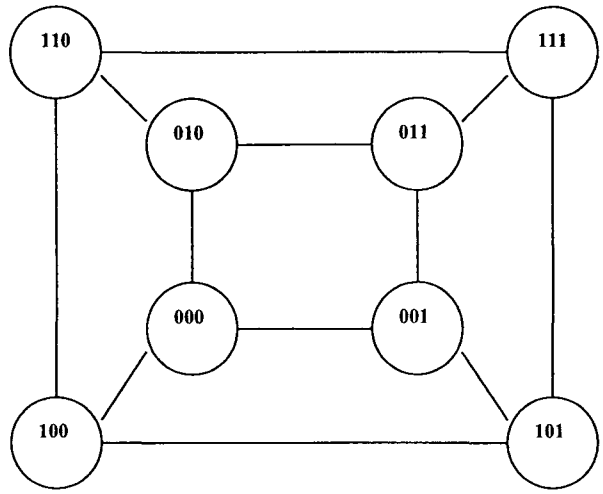


Figure 2.1
2^3 Hypercube

Synchronization is established by data, means computation at some node completes only when it's needed data are available.

The main advantage of this architecture is the simplicity of design. The nodes are identical, so it can be constructed at low cost.

Hypercube has been studied in detail as a network topology for parallel computes. The hypercube is regular, node symmetric, link symmetric and recursive. A network *W* is called regular of degree *k*-regular if all of its nodes have the same degree *k*. The degree of a node in *W* indicates the number of edges connected with it. *W* is node symmetric if for every two nodes *x* and *y* in *w*, there exist an automorphism *F* of *W* so that F maps *x* to *y* and *F(W)* = *W*. An automorphism of *W* is a one-to-one correspondence between the nodes of W which preserves all of its structure.

A *k*-dimensional hypercube consists of 2^k nodes that are labeled with 2^k Binary numbers from *0* to 2^k*-1*. Two nodes of a hypercube are adjacent if and only if their labels differ by exactly one bit or if hamming distance between two nodes is one. The node connectivity of a hypercube is *k*. That is at least *k* edges are needed to remove in order to disconnect the hypercube. The diameter of a *k-cube* is *k*.

Many important problems such as sorting prefix computation, fast fourier transformation, matrix multiplication, matrix transpose, matrix inversion, eigenvalues, connected components, all-pairs shortest path, minimum spanning tree etc. can be efficiently solved

using the hypercube. Besides many other networks such as linear arrays etc can be effectively embedded in the hypercube.

2.2.1 Properties of Hypercube

1. An n-cube graph is an undirected graph consist of $k = 2^n$ vertices labeled from 0 to 2^k-1 and such that there is an edge between two vertices if and only if the binary representations of their labels differ by one and only one bit.
2. There are n different ways of splitting an n-cube into two *(n-1)* subcubes so that their respective vertices are connected in a one-to-one way.
3. Any two adjacent nodes of an n-cube are connected in one-to-one way.
4. There is no cycle of odd length.
5. It is a connected graph of diameter *n*.

2.2.2 Hypercube scalability

Hypercube is scalable i.e. a hypercube of n node can be constructed by combining two *(n-1)* node hypercube. So we can construct a 2^4 node hypercube by combining two 2^3 node hypercube. Binary notation of all the nodes is changed by appending 0 and 1 at least significant bit (LSB) in first hypercube and second hypercube respectively, and then node with value 0 of first hypercube is connected to node value 1 in second hypercube, node value 2 in first hypercube is connected with node value 3 in second hypercube and so on.

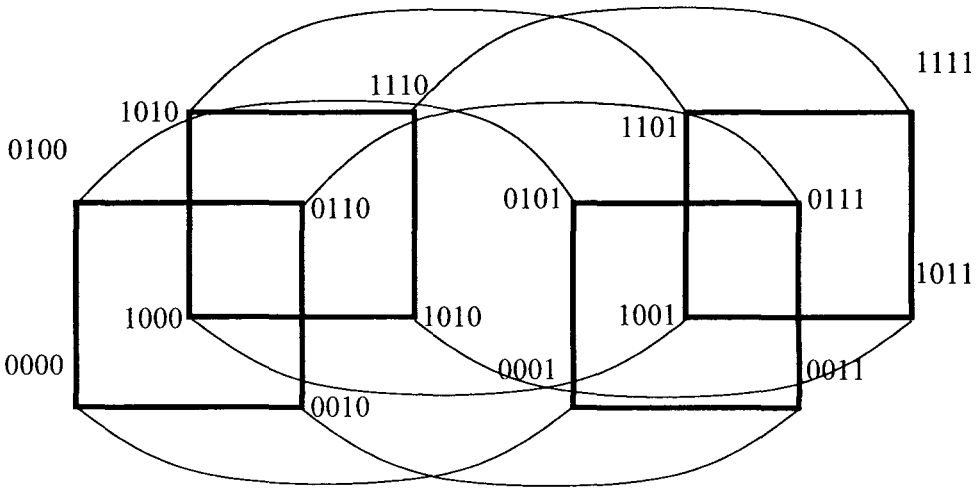


Figure 2.2
Merging of two hypercube

2.3 Supercube

Supercube is a generalized version of hypercube. After hypercube the concept of generalized hypercube came in existence. So we call it generalized hypercube also. It can be constructed for any number of nodes, there is no necessity for number of nodes to be in power of 2 as it is in hypercube.

2.3.1 Supercube construction process:

An *N-node* supercube can be constructed by an *m-dimensional* hypercube, where $2^{m-1} < N < 2^m$. For each node u, $N \leq u \leq 2^{m-1}$, merging nodes u and $u\text{-}2^{m-1}$ in m-dimensional hypercube into a single node labeled as $u\text{-}2^{m-1}$ and leaving other nodes in m-dimensional hypercube unchanged.

Corresponding Supercube of 7 nodes

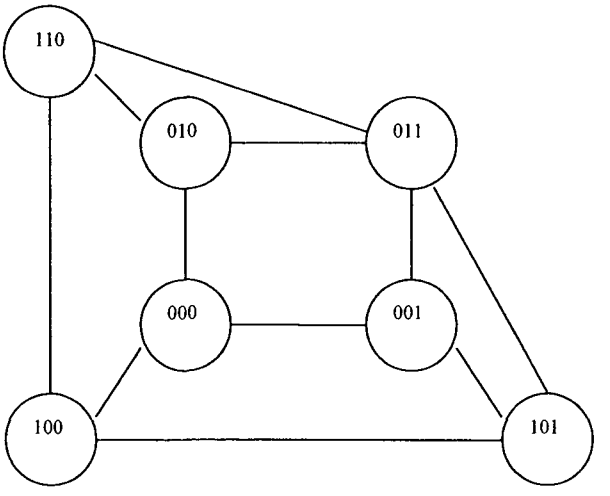


Figure 2.3
7 node Supercube

Although this network can be constructed for any number of nodes but it has two major problems:

1. When the number of node is prime it reduces to completely connected graph.
2. It needs significant change if we want to add a new node. A new modified hypercube has been proposed by Sen which does not have the drawbacks of either generalized hypercube and it maintains the same connectivity and diameter corresponding to hypercube. An *N-node* supercube is either a supergraph of *(m-1)*

dimensional hypercube when $2^{m-1} < N < 2^m$ or an m-dimensional hypercube when $N = 2^m$. In this way the constructed network is called Supercube.

2.3.2 Properties of Supercube

1. The node connectivity of an *N-node* supercube is at least $\lceil \log_2 N \rceil$.
2. The diameter of an *N-node* supercube is at most $log_2 N$. The node degree of an *N-node* supercube is between *k-1* and *2(k-1)* where $k = \lceil \log_2 N \rceil$.

2.4 Folded Hypercube

Although hypercube network topology is popular because of its low diameter and large number of node disjoint paths, it has some major drawback. The restriction on system size of hypercube leaves a large gap between two allowable sizes. Folded hypercube is one of the several generalizations of hypercube, to improve its performance. Authors make effort to reduce the diameter by adding extra connections between the nodes. These connections are called skip. A skip is established between every pair of nodes, $u_{n-1}u_{n-2}\ldots\ldots u_{n-k}u_{n-k-1}\ldots\ldots u_i\ldots\ldots u_0$ *and* $u'_{n-1}u'_{n-2}\ldots\ldots u'_{n-k}u'_{n-k-1}\ldots\ldots u'_i\ldots\ldots u'_0$ with $0 \leq k \leq n-2$ and $n = \lceil \log_2 N \rceil$. Different values of k give different enhanced hypercubes and different degree of performance improvement. Tzeng and Wei concluded that when n is even (or odd) creating a skip between a pair of nodes with $k = 0$(or *1*) can improve system performance optimally or near about that. The n-cube with $k = 0$ is called folded hypercube.

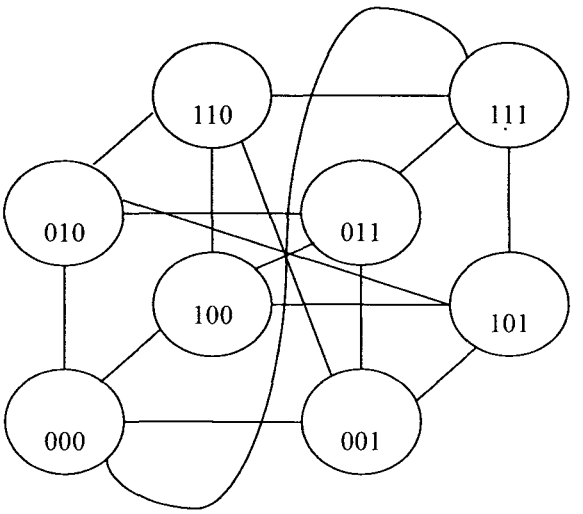


Figure 2.4
Folded Hypercube

Properties of folded hypercube:

1. Diameter of n-node folded hypercube is $\lceil n/2 \rceil$.
2. Folded hypercube is regular of degree $n+1$.
3. Node connectivity in folded hypercube is $n+1$.

2.5 Folded Supercube:

Folded Supercube is constructed form of the folded hypercube network. Like folded hypercube, some extra connections are added to reduce the diameter of a supercube.

Steps to construct an *N-node* folded supercube:

1. Make a k-dimensional hypercube, where $k - \lceil \log_2 N \rceil$.
2. Add the complementary edges.
3. Merge nodes by the technique described earlier.

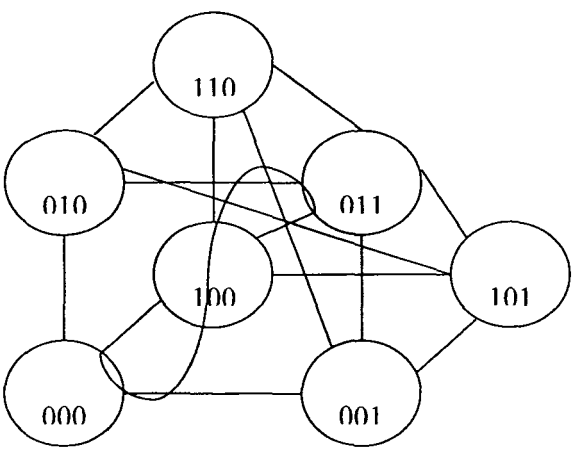


Figure 2.5
Folded Supercube

Chapter 3
SORTING sON HYPERCUBE

3.1 Introduction

As we see in academics in computer science, from graduation, computer science students are likely to have studied the basic sorting algorithms in their introductory programming class, then in their data structures class, and finally in their algorithms class. Above all still researchers are working continuously how to make it more and more efficient using various technologies. Fro this, parallel computing is playing great role to minimize the time of sorting. So we can say that sorting is very important that is used in many different fields. There are many reasons for its worth:

a. Sorting is the basic building block through which many other algorithms are built.
b. History tells that computers have spent more time in sorting than doing anything else. A quarter of all mainframe cycles are spent in sorting data.
c. Sorting is the most thoroughly studied section in computer science. Literally dozens of different algorithms are known, most of which have some advantages over all other algorithms with certain conditions.
d. Most of the interesting ideas used in the design of algorithms appear in the context of sorting, such as divide-and-conquer, data structures, and randomized algorithms.
e. Sorting improves system's ability to compare and move large amounts of data, which is the most expensive part of any scientific or commercial application.

Sorting on parallel computer was first suggested by Francis and Mathieson. The process of sorting data is an integral part of any database management system. Sorting is for creating indexes and for processing complex queries. All commands for transaction in SQL commands require an internal sort and can benefit from the use of parallel sorting.

3.2 Sorting Algorithms

Since there are various approaches for sorting for both serial and parallel machines. Sorting algorithms can be categorized in three classes: counting-based sort, fixed-topology sort, and partitioning sort.

3.2.1 Counting-based sort

Counting-based sort work by using keys as integers in the range 0... (N-1). Counting-based sorts determine the ordering of keys by counting the number of occurrences of each possible value, rather than by comparing pairs of keys.

There are two minor disadvantages: it does not perform well with large keys, since the running time is proportional to the key size, and it can not be executed in place. It has some advantages like: Easy implement, deterministic, load-balanced, stable, fast for short keys, and efficient for a wide range of problem sizes.

3.2.2 Fixed-topology sort

In fixed-topology, the interconnection network is fixed among the processors, such as hypercube, supercube etc and there is no data-dependency for communication. Bitonic sort is an example of fixed-topology sort. Bitonic sort is applicable only for the sequence of numbers in which the numbers initially increases and then decreases. This sort is quite efficient on parallel machines.

3.2.3 Partitioning sort

Partitioning sorting algorithms select a subset of the keys that partition the data and then send these partitioned elements to different processors. Using partitioning sort Wagar has given a sorting method named Hyperquicksort because it has been implemented on hypercube. In this sort input keys are partitioned parallely. This sort initially distributes the keys evenly among the processors and at each step picks a pivot element then distributes the pivot across the machine, and sends all the keys less than the pivot on one node of the hypercube and the greater keys on the other node. This process is applied recursively within each sub-cube. It is very important to pick a pivot that closely balances the two halves otherwise the load on the processors can become extremely imbalanced.

3.3 Sorting Performance

Sorting data is very expensive. That is why so many persons are still involved in developing and analyzing algorithms. For any real world application of sorting the analysis of an algorithm may be secondary consideration as compare to the constraints

involved, it is possible that for a particular problem certain algorithms may work significantly better than others.

3.4 Sorting on Hypercube

Although sorting on Hypercube is done using various different methods. Parallel Sorting on Hypercube of *4-nodes, 8-nodes, 16-nodes and 32-nodes* is implemented keeping in mind about minimum communication time and sorting process on each node of the Hypercube. Putting elements to different processors is also an important work so that it should take minimum communication time and minimum load at individual processing node. Sorting performance depends much on the distribution of items on each node. If there is an *n-node* hypercube and number of items to be distributed is *N* then distribution is easy by simply assigning *N/n* (applying division operator) items on each node so that each node get equal number of items.

There are two cases:

Case 1: When *n* is multiple of *N*

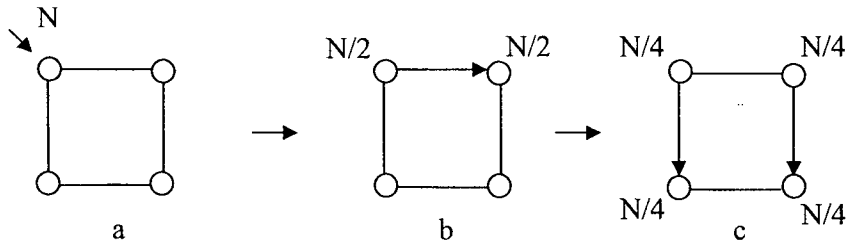


Figure 3.1 Data distribution

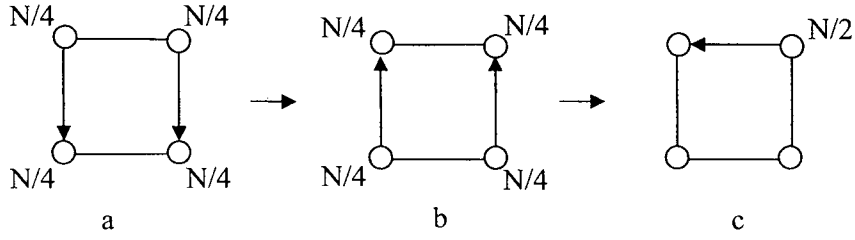


Figure 3.1 Sorting of input data

In Figure 3.1.a, the number of inputs given to a node is *N* which is a multiple of *n*, here *n* is 4. In Figure 3.1.b the input is divided into two halves and one half is transferred to its neighboring node. Again in Figure 3.1.c the inputs of the two nodes is divided and transferred to other two nodes in one communication time.

In Figure 3.2.a, all the items of each node is sorted in $N/4\log_2 N/4$ time using merge sort. In the next step all the node have inputs in the sorted form so we can merge the two nodes contents. This can be done in $N/4$ time only. Then two processor nodes become idle and the input data is distributed between two nodes, half of N on each node. Finally these two halves are merged in $N/2$ time and finally all the items are in stored form at one processor node. From Figure 3.2, it is clear that the sorting can be accomplished in two communication of time. So,

Processing Time = $N/4*\log_2 N/4+N/4+N/2$

$= N/4*\log_2 N/4+3*(N/4)$

Communication Time = 2

Total time taken = Processing Time + Communication Time

$= (N/4*\log_2 N/4 + 3*(N/4) + 2)$

General formula to calculate total time taken for 2^k –hypercube:

$= (N/p*\log_2 N/p + (2^k-1)*(N/p) + 2)$

Where p is number of processing nodes.

Case 2: When n is not multiple of N

In this case distribution of number of items is not identical at all the processor nodes. But, to minimize the load balance, the difference of number of items between any two nodes should not be much. To distribute the number of items at all the processing nodes I can follow any one of the two methods:

1. Assign $(N/p + N\%p)$ number of items to the first processing node then N/p to all other nodes. But this distribution method faces the problem of load balancing, when $N\%p$ is very close to N/p.
2. To distribute approximately equal number of items on each processing node, each node gets at the most $(N/4+1)$ items.

Following the second method

Processing Time = $(N/4+1)*\log_2(N/4+1)+ (N/4+1)+2*(N/4+1)$

$= (N/4+1)*\log_2(N/4+1)+ 3*(N/4+1)$

Communication Time = 2

Total time taken = Processing Time + Communication Time

$= [(N/4+1)* \log_2(N/4+1)+ 3*(N/4+1) + 2]$

This is the general formula to perform parallel sorting on a 4-node hypercube.

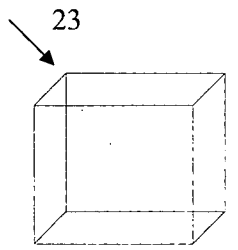
General formula to calculate total time taken for 2^k –node hypercube:

$= [(N/p+1)*\log_2 (N/p+1) + (2^k-1)*(N/p+1) + k]$

Where, p is number of processing nodes.

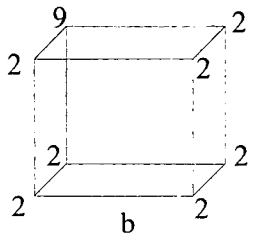
Explanation: If there is 2^3 – *node* hypercube and number of items to be sorted is 23.

If I follow the first distribution method



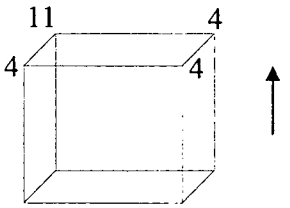
a

Input is given to a processing node



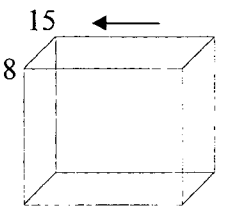
b

All the items at each node are sorted in $9\log_2 9$



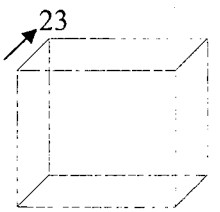
c

Four neighboring pair's contents are merged in time 9 only



d

Two neighboring pair's contents are merged in time 11 only



e

Two neighboring nodes contents are merged in time 15 only

Figure 3.2 Sorting on 2^3 – node hypercube

Processing time: $9\log_2 9 + 9 + 11 + 15$.

$= 63.529325012980811266167300991061$.

Total time taken = 66.529325012980811266167300991061.

Time taken in sequential sort: 104.041924989311296062765409615 74.
If I follow the second distribution method

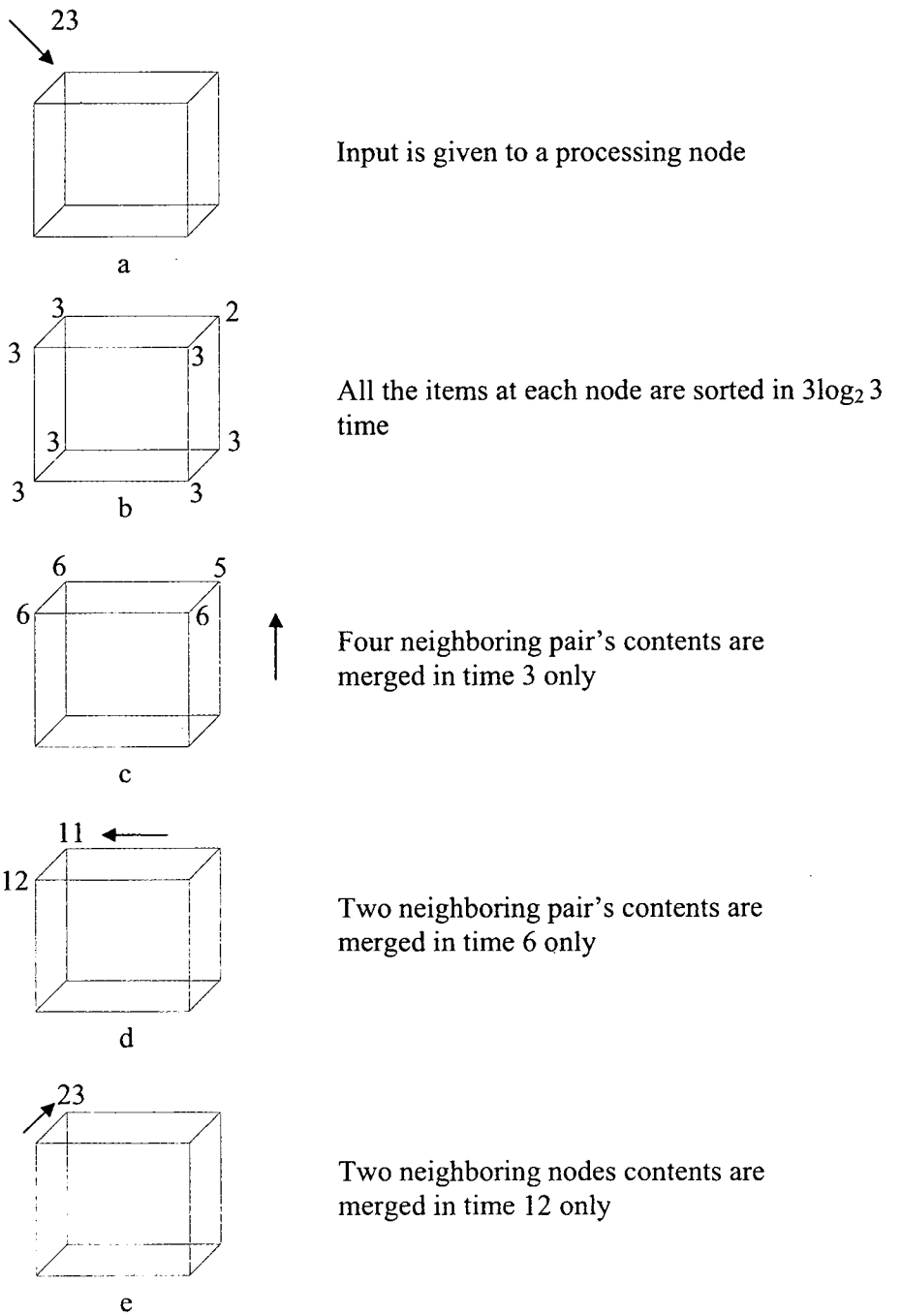


Figure 3.4 Sorting on 2^3 – node hypercube

Processing time: $3\log_2 3 + 3 + 3 + 12$.
$= 22.754887502163468544361216831843$.
Total time taken $= 25.754887502163468544361216831843$.
Time taken in sequential sort: 104.041924989311296062765409615 74.

Sorting time performance and Speedup on Hypercube (Number of nodes 4, 8, 16 and 32)

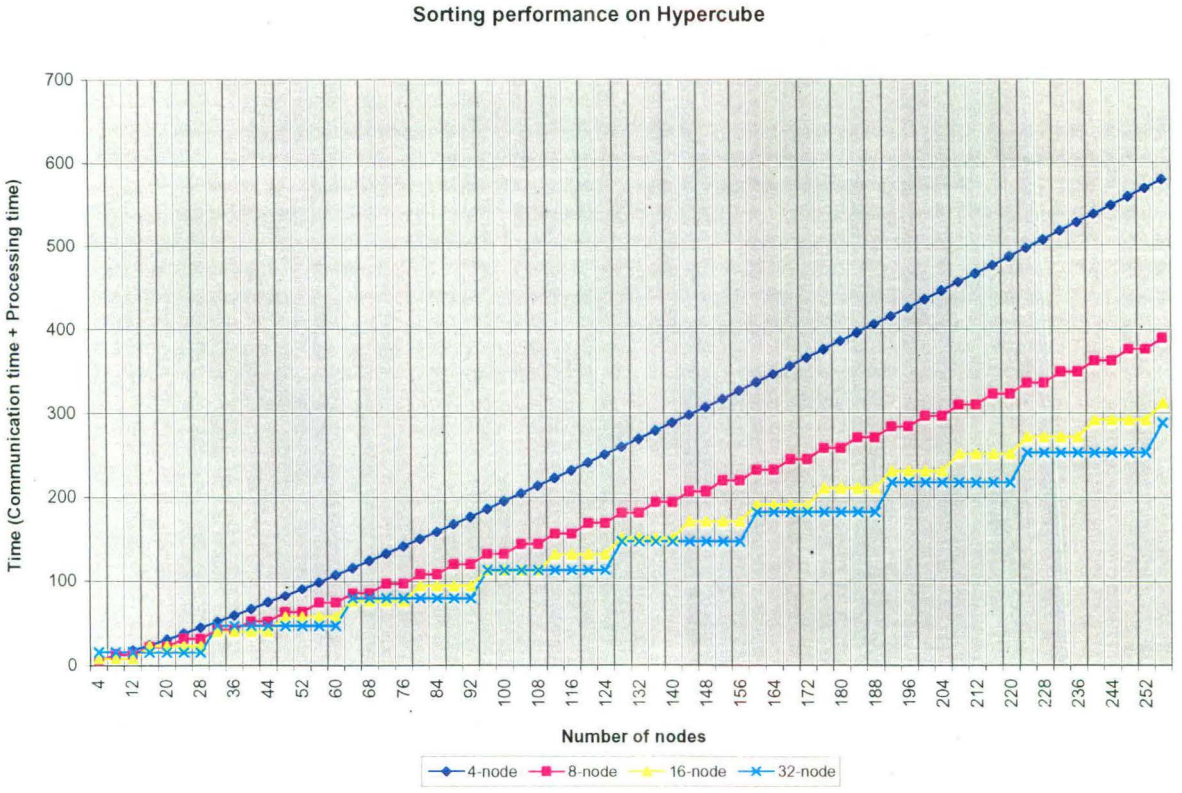
Items	P1	P4	P8	P16	P32	SP4	SP8	SP16	SP32
4	10.000	7.000	6.000	8.000	16.000	1.429	1.667	1.250	0.625
8	19.000	12.000	13.000	8.000	16.000	1.583	1.462	2.375	1.188
12	19.000	17.755	13.000	8.000	16.000	1.070	1.462	2.375	1.188
16	28.755	24.000	22.000	23.000	16.000	1.198	1.307	1.250	1.797
20	28.755	30.610	22.000	23.000	16.000	0.939	1.307	1.250	1.797
24	39.000	37.510	31.755	23.000	16.000	1.040	1.228	1.696	2.438
28	49.610	44.651	31.755	23.000	16.000	1.111	1.562	2.157	3.101
32	49.610	52.000	42.000	40.000	47.000	0.954	1.181	1.240	1.056
36	60.510	59.529	42.000	40.000	47.000	1.016	1.441	1.513	1.287
40	60.510	67.219	52.610	40.000	47.000	0.900	1.150	1.513	1.287
44	71.651	75.054	52.610	40.000	47.000	0.955	1.362	1.791	1.524
48	71.651	83.020	63.510	57.755	47.000	0.863	1.128	1.241	1.524
52	83.000	91.106	63.510	57.755	47.000	0.911	1.307	1.437	1.766
56	94.529	99.303	74.651	57.755	47.000	0.952	1.266	1.637	2.011
60	94.529	107.603	74.651	57.755	47.000	0.878	1.266	1.637	2.011
64	106.219	116.000	86.000	76.000	80.000	0.916	1.235	1.398	1.328
68	106.219	124.487	86.000	76.000	80.000	0.853	1.235	1.398	1.328
72	118.054	133.059	97.529	76.000	80.000	0.887	1.210	1.553	1.476
76	118.054	141.711	97.529	76.000	80.000	0.833	1.210	1.553	1.476
80	130.020	150.439	109.219	94.610	80.000	0.864	1.190	1.374	1.625
84	142.106	159.239	109.219	94.610	80.000	0.892	1.301	1.502	1.776
88	142.106	168.107	121.054	94.610	80.000	0.845	1.174	1.502	1.776
92	154.303	177.042	121.054	94.610	80.000	0.872	1.275	1.631	1.929
96	154.303	186.039	133.020	113.510	113.755	0.829	1.160	1.359	1.356
100	166.603	195.096	133.020	113.510	113.755	0.854	1.252	1.468	1.465
104	166.603	204.211	145.106	113.510	113.755	0.816	1.148	1.468	1.465
108	179.000	213.382	145.106	113.510	113.755	0.839	1.234	1.577	1.574
112	191.487	222.606	157.303	132.651	113.755	0.860	1.217	1.444	1.683
116	191.487	231.881	157.303	132.651	113.755	0.826	1.217	1.444	1.683
120	204.059	241.207	169.603	132.651	113.755	0.846	1.203	1.538	1.794
124	204.059	250.580	169.603	132.651	113.755	0.814	1.203	1.538	1.794
128	216.711	260.000	182.000	152.000	148.000	0.834	1.191	1.426	1.464
132	216.711	269.465	182.000	152.000	148.000	0.804	1.191	1.426	1.464
136	229.439	278.974	194.487	152.000	148.000	0.822	1.180	1.509	1.550
140	242.239	288.525	194.487	152.000	148.000	0.840	1.246	1.594	1.637
144	242.239	298.117	207.059	171.529	148.000	0.813	1.170	1.412	1.637
148	255.107	307.750	207.059	171.529	148.000	0.829	1.232	1.487	1.724
152	255.107	317.421	219.711	171.529	148.000	0.804	1.161	1.487	1.724
156	268.042	327.131	219.711	171.529	148.000	0.819	1.220	1.563	1.811
160	268.042	336.877	232.439	191.219	182.610	0.796	1.153	1.402	1.468

Table 3(a): Time performance and Speedup on Hypercube

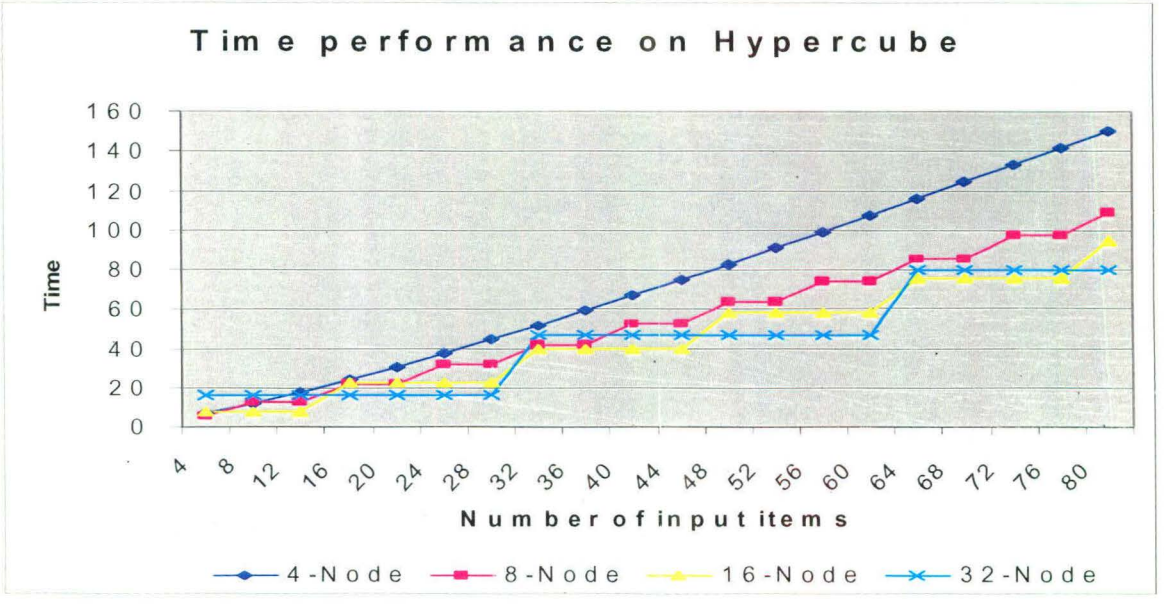
From the above table of data it is clear that the 8-NodeHypercube performs better than the 4-Node Hypercube. For the same data a 16-Node Hypercube is not performing better.

And the 32-Node Hypercube performs bad for the same data initially but later its performance improves as the number of input items increases. The reason behind that if the number of input items is less than it wastes its time in communication. From the table it is clear that the 8-Node hypercube always performs better than a 4-Node Hypercube, for the same inputs

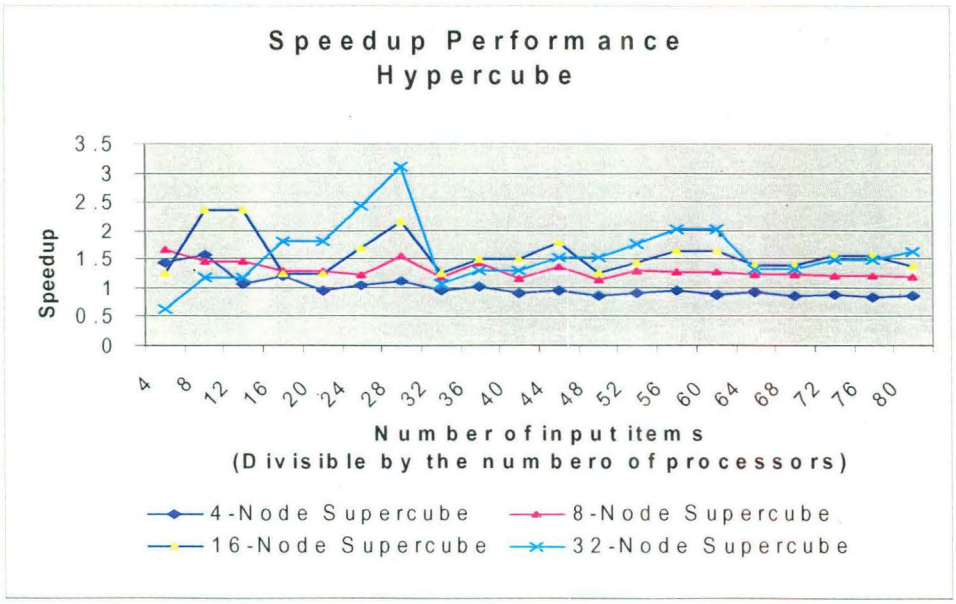
16-Node Hypercube performs poorer up to 20 inputs of items after that its performance starts improving but not much for smaller number of inputs, but if the number of input items increases its performance improves continuously.



Graph 3(a): Sorting-time performance on Hypercube for any number of items.



Graph 3(b): Sorting-time performance on Hypercube.
(Number of nodes is multiple of number of processors.)



Graph 3(c): Speedup performance on Hypercube.

Chapter 4
Sorting on Supercube

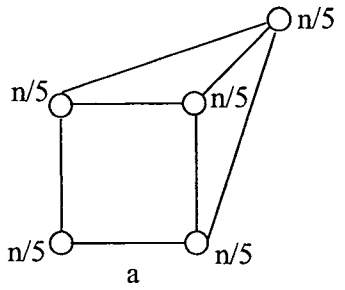
Sorting on supercube has great advantage that it gives the freedom to select any number of processing nodes. There is not a necessity to construct it with 2^k number of nodes, where k is any positive integer value.

4.1 Sorting on 5-node Supercube

It is obvious that if the number of input data is in multiple form of 5 then all the processing nodes will have equal number of input data. To perform sorting on 5-node supercube, it needs three communication time. Simultaneously we take care of that any node should not suffer from more load balance otherwise its performance degrades. 5-node supercube is constructed from an 8-node hypercube after deleting three nodes having highest leveling number.

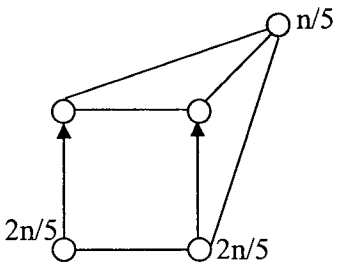
Time complexity evaluation, taking n number of input items.

Case I: If n is multiple of 5.



a

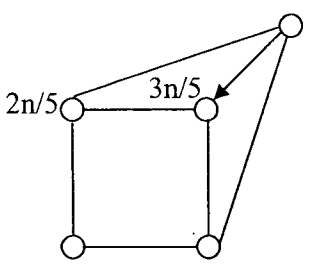
Equal number of items distributed at each processing node then sorted all in time ($n/5 \log_2 n/5$).



b

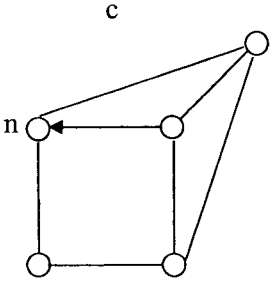
Two neighboring pairs are merged in time n/5.

Now attention should be taken here, if we merge two node having 2n/5 items then performance will degrade.



c

Here taken time is 2n/5.



d

Here taken time is 3n/5.

Figure 4.1
Sorting on a *5-node* Supercube.

Processing time: $(n/5 \log_2 n/5) + n/5 + 2n/5 + 3n/5$.

Communication time: 3

Total time: $(n/5 \log_2 n/5) + n/5 + 2n/5 + 3n/5 + 3$.

$= (n/5 \log_2 n/5) + 6n/5 + 3$.

If n is 25,

Total parallel sorting time is: 44.609640474436811739351597147447.

Total sequential sorting time: 116.096404744368117393515971474447.

Case II: If *n* is not a multiple of 5.

Worst condition may come when *n* is just one less to be a multiple of 5. In this case we assign n/5 items to each processing node and then additionally assign one more item to each of the processing node but it is not possible to all of the nodes.

Total processing time: [(n/5+1) log(n/5 + 1) + 6*(n/5+1)]

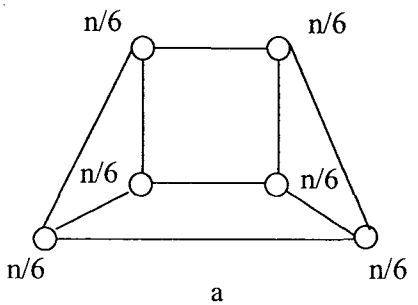
Communication time: 3

Total parallel sorting time: [(n/5+1) log(n/5 + 1) + 6*(n/5+1) + 3].

4.2 Sorting on 6-node Supercube

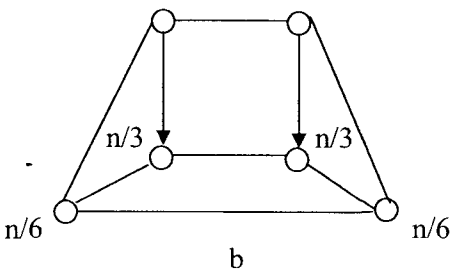
Case I: If number of input items is *n* and divisible by 6.

Sorting can be accomplished within three communication of time. Initially each of the processing nodes gets assigned n/6 items.



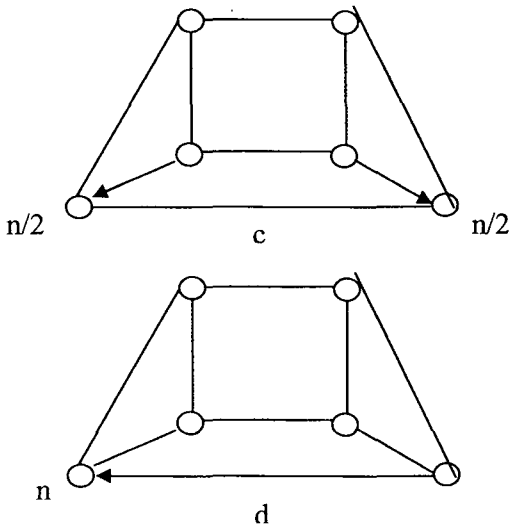
a

All the items at each processing nodes are sorted in n/6logn/6



b

Time taken to merge two pairs is n/6



Time taken to merge two pairs is n/3

Time taken to merge two pairs is n/2

Figure 4.2
Sorting on a 6-*node* Supercube

Total processing time = n/6logn/6 + n/6 + n/3 + n/2.

= n/6logn/6 + n.

Communication time = 3.

Total time taken = n/6logn/6 + n + 3.

Case II: If number of input items is *n* and is not divisible by 6.

Total processing time = (n/6+1) log (n/6+1) + 6*(n/6+1).

Communication time = 3.

Total time taken = (n/6+1) log (n/6+1) + 6*(n/6+1) +3.

4.3 Sorting on 7-node Supercube

Communication time in 7-node supercube is three. Firstly I distribute approximately equal number of items among all the processing nodes then sorting and communication starts.

Case I: If number of input items is *n* and divisible by 7.

Total processing time = n/7logn/7 + n/7 + 2n/7 + 4n/7.

= n/7logn/7 + n.

Communication time = 3.

Total time taken = n/7logn/7 + n + 3.

Case II: If number of input items is n and is not divisible by 7.

Total processing time = (n/7+1) log (n/7+1) + 7*(n/7+1).

Communication time = 3.

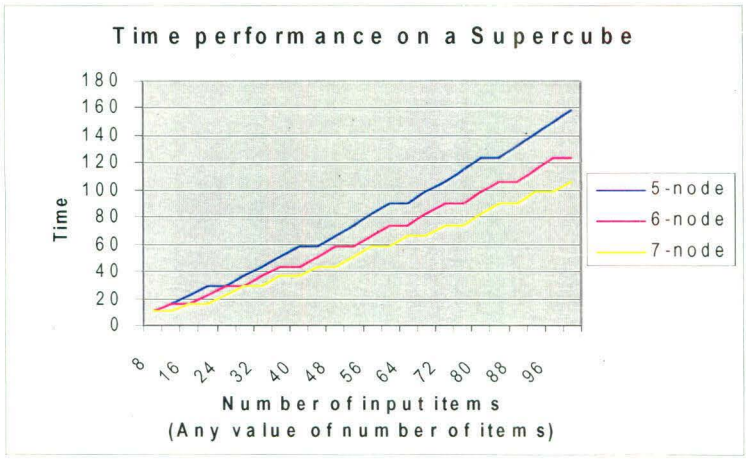
Total time taken = (n/7+1) log (n/7+1) + 7*(n/7+1) + 3.

Sorting time performance and Speedup on Supercube (Number of nodes is not multiple of 5)

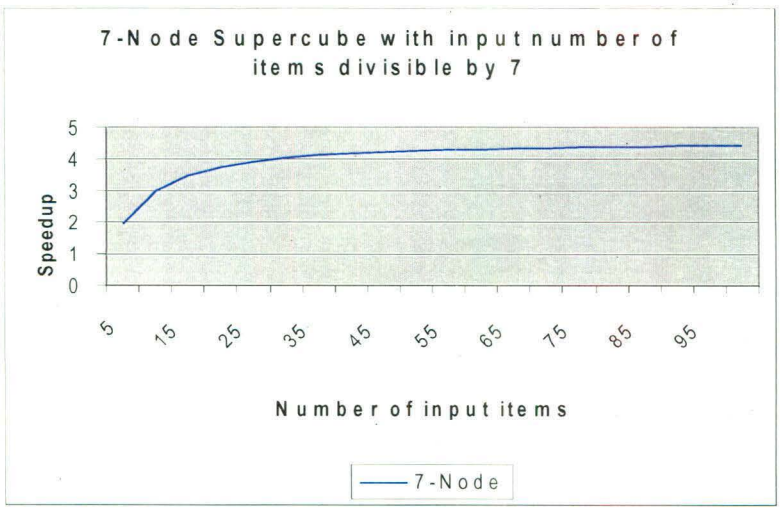
No. of inputs	1-Node (1)	5-Node (2)	6-Node (3)	7-Node (4)	Speedup (1)	Speedup (2)	Speedup (3)
8	24.000	11.000	11.000	11.000	2.182	2.182	2.182
12	43.020	16.755	16.755	11.000	2.568	2.568	3.911
16	64.000	23.000	16.755	16.755	2.783	3.820	3.820
20	86.439	29.610	23.000	16.755	2.919	3.758	5.159
24	110.039	29.610	29.610	23.000	3.716	3.716	4.784
28	134.606	36.510	29.610	29.610	3.687	4.546	4.546
32	160.000	43.651	36.510	29.610	3.665	4.382	5.404
36	186.117	51.000	43.651	36.510	3.649	4.264	5.098
40	212.877	58.529	43.651	36.510	3.637	4.877	5.831
44	240.215	58.529	51.000	43.651	4.104	4.710	5.503
48	268.078	66.219	58.529	43.651	4.048	4.580	6.141
52	296.423	74.054	58.529	51.000	4.003	5.065	5.812
56	325.212	82.020	66.219	58.529	3.965	4.911	5.556
60	354.413	90.106	74.054	58.529	3.933	4.786	6.055
64	384.000	90.106	74.054	66.219	4.262	5.185	5.799
68	413.947	98.303	82.020	66.219	4.211	5.047	6.251
72	444.235	106.603	90.106	74.054	4.167	4.930	5.999
76	474.842	115.000	90.106	74.054	4.129	5.270	6.412
80	505.754	123.487	98.303	82.020	4.096	5.145	6.166
84	536.955	123.487	106.603	90.106	4.348	5.037	5.959
88	568.430	132.059	106.603	90.106	4.304	5.332	6.308
92	600.168	140.711	115.000	98.303	4.265	5.219	6.105
96	632.156	149.439	123.487	98.303	4.230	5.119	6.431
100	664.386	158.239	123.487	106.603	4.199	5.380	6.232

Table 4 (a)
Time performance and speedup for Hypercube architecture

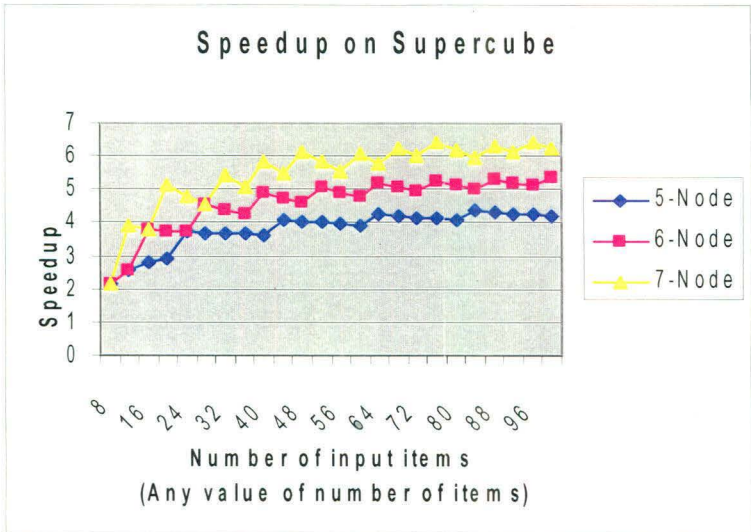
From the above table of data it is clear that the performance of a 6-Node Supercube is better than 6-Node Supercube and 7-Node Supercube performance is better than 6-Node Supercube for the same number of input items. In the beginning the performance of 7-Node Supercube is not satisfactorily good but as the number of input items increase its performance continuously improvss.



Graph 4(a): Time performance on a Supercube for any number of items.



Graph 4(b) : Speedup performance on Supercube of 7-Nodes.



Graph 4(c): Speedup performance on Supercube for any number of items.

Chapter 5

Conclusion

Sorting is an important application in scientific and commercial field. It should have some standard benchmark. To fix a benchmark, variation in size and distribution of sorted data is necessary to accurately measure how an algorithm and architecture works. Sorting is a perfect data transfer process which measures a number of important system performances including communication bandwidth.

These benchmarks can be applied in presently existing supercomputer, from single processor to many processor systems.

Since the Hypercube is restricted for the number of processing nodes in the form of 2^k, where k is any integer. Although Hypercube is easily scalable and cost effective but the restriction degrades the performance of sorting on it, if the number of processing nodes is not a multiple of 2^k, in that condition Supercube performs better.

There are many more network topologies like Folded Hypercube, Folded Supercube etc, implementation on these may give better performance but it is yet to be done.

References

[1.]L.N. Bhuyan and D. P. Agrawal, "Generalized Hypercube and Hyperbus structures for a computer network", IEEE Transactions on Computers, vol. C-33, no. 4, pp. 323-333, 1984.

[2.]Y. Saad and M. H. Schultz, "Topological properties of Hypercube", Research roport 389, Dept. Comp. Sc., Yale University, 1985.

[3.]H. P. Katseff, "Incomplete Hypercube", IEEE Transactions on Computers. 37(1998), pp. 604-607.

[4.]Sen, "Supercube: An Optimal fault tolerant network architecture", Acta inform.,26(1989), pp. 741-748.

[5.]S. M. Yuan," Topological properties of Supercube", Inform. Process letter 37(1991), pp. 241-245.

[6.]E1-Amawy and S. Latif, " Properties and Performance of Folded Hypercube", IEEE Trans parallel and distributed systems, Vol.2, No.1, Jan.1991, pp. 31-42.

[7.]N. F. Tzeng and S. Wei, "Enhanced Hypercubes", IEEE Trans. Comput., Vol.40, No.3, March 1991, pp. 284-294.

[8.]Erich Schikuta and Peter Kirkovits, "Analysis and Evaluation of Sorting onHypercube-Based System", Inst. J: Applied Computer Science, Dept. of Data Engineering, University of Vienna, Rathausstr. 19/4, A-1 01 0 Vienna, Austria.

[9.]Bulent Abali, Fusun Ozguner, Member, IEEE, and Abdulla Bataineh, Member, IEEE, "Balanced Parallel Sort on Hypercube Multiprocessors", 572 IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 4, NO. 5, MAY 1993.

[10.] Youran Lan and Magdi A. Mohamed, "Parallel Quicksort in Hypercubes", Department of Computer Science University of Missouri-Columbia.

[11.] David S. L. Wei, "Quicksort and Permutation Routing on the Hypercube and De Bruijn Networks", School of Computer Science and Engineering The University of Aizu Fukushima, 965-80 Japan.

[12.] Minsoo Jeonand Dongseung Kim, "Load-Balanced Parallel Merge Sort on Distributed MemoryParallel Computers", Dept. of Electrical Engineering, Korea University Seoul,136-701,Korea msjeon, dkim @classic.korea.ac.kr

[13.] Erich Schikuta and Peter Kirkovits, "Analysis and Evaluation of Sorting on Hypercube-Based Systems", Inst. J: Applied Computer Science, Dept. of Data Engineering, University of Vienna Rathausstr. 19/4, A-1 01 0 Vienna, Austria.

[14.] Ten H. Tzen and Lionel M. Ni, Senior Member, IEEE, "DePendence I Uniformization: A Loop Parallelization Technique", IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 4, NO. 5, MAY 1993 547.

[15.] Joseph JhJh, Senior Member, IEEE, and Kwan Woo Ryu, Associate Member, IEEE, "Optimal Algorithms on the Pipelined Hypercube and Related Networks", IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 4, NO. 5, MAY 1993. [BA,84] L.N. Bhuyan and D. P. Agrawal, "Generalized Hypercube and Hyperbus structures for a computer network", IEEE Transactions on Computers, vol. C-33, no. 4, pp. 323-333, 1984.

[16.] Kai Hwang, "Advanced Computer Architecture: Parallelism, Scalability, Programmability", McGraw Hill, 1993.

[17.] Michel J, Quinn, "Parallel Computing: Theory and Practice", McGraw Hill 1994.

[18.] Ananth Grama, Anshul Gupta, George Karpis, Vipin Kumar, "Introduction to parallel Computing", Person Education Ltd., 2003.

[19.] Narsingh Deo, "Graph Theory with applications to engineering and computer science", Prentice Hall of India pl. 2004. Thomas H. Cormen, Charles E. leiserson, Ronald L. Rivest, Clifford Stein, "Introduction to Algorithms", Prentice Hall of India pl. 2002.