This is to certify that the

dissertation entitled

COMMUNICATION PERFORMANCE OF
MULTICOMPUTERS

presented by

Suresh Chittor

has been accepted towards fulfillment
of the requirements for

Ph.D degree in Computer Science

Major professor

Date May 31, 1991

COMMUNICATION PERFORMANCE OF MULTICOMPUTERS

By

Suresh Suryanarayana Chittor

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science
1991

ABSTRACT

COMMUNICATION PERFORMANCE OF MULTICOMPUTERS

By

Suresh Suryanarayana Chittor

Massively-parallel distributed-memory concurrent computers or multicomputers are beginning to dominate the supercomputing arena. The continued success of multicomputers has resulted in an ever increasing demand on their communication performance. In this thesis, we perform theoretical, experimental and simulation studies to understand and improve the communication performance of large multicomputers that use direct networks. We show that 2D/3D mesh networks provide much higher performance than popular hypercube networks, when the effect of contention is negligible. The performance improvement is due to higher bandwidth channels, and the ability of the switching technique to efficiently handle communication over long paths. However, in the case of large multicomputers that use mesh networks, contention for network channels can be significant which will lower the performance. We study the effect of contention for a given mapping of parallel tasks on a set of multicomputer nodes. A metric called path contention level is introduced as a measure of contention and quality of mapping. The results of our studies have been used to implement a tool that helps users predict the effect of contention, identify communication bottlenecks and to evaluate the adequacy of a mapping for a given application. We also show that random mapping is not advisable for large multicomputers that use mesh networks. We illustrate in several cases that careful mapping and routing can minimize contention which in turn can significantly improve communication performance. Theoretical results are supported and complemented by experimental results on a Symult 2010, and simulations that give the performance of large multicomputers not yet available. We conclude that mesh networks will replace hypercube networks as they can provide better performance under contention-free conditions, and that research interest will shift from minimizing path length to minimizing contention.

To my parents
Ramamani and Suryanarayana Rao

n

ar

u

ti

D

a

n

se

th

th

ga

S

th

ACKNOWLEDGMENTS

I owe a lot to my thesis advisor Dr. Richard Enbody for his constant encouragement and support right from the beginning. I am grateful to the many invaluable and long discussions I had with him. Those discussions helped very much in shaping up my ideas into this thesis in its present form.

Many thanks to Dr. Lionel Ni for his wise and timely suggestions that kept me on the right track during my research. I thank my other guidance committee members, Dr. Esfahanian and Dr. Drachman, for their help, encouragement and guidance. I am grateful to the chairperson, Prof. Wojcik, for all his help, and for providing the much needed financial assistance during my stay at MSU.

I thank the members of the parallel processing research group at MSU. The weekly seminars and discussions with them had been very useful, and stimulating. Special thanks to Arun, Jayashree, and Tyan-Shu for their help and encouragement. I thank the system managers for keeping the computer facilities up and running all the time.

I thank my roommates, friends and students at MSU who made my stay at Michigan State University very pleasant and unforgettable. Special thanks to Sateesh, Shashidhar, Keshavchandra, Vibhu, Mohankrishnan, Bharath, and Sanjay.

I am indebted to my wife Aruna for her support, understanding, and patience during many long days and late nights I spent on my research and thesis.

Table of Contents

E

List of Figures

List of Tables

Ch

INT

Rapid p
increase

- Fe
ac
pe

- Sy
m
ex

- Sc
co
th
bo
pe
fo

The
the tech
that nee

1.1

A concu
network
In additio

Chapter 1

INTRODUCTION

Rapid progress has been made in concurrent computing technology, and the result is increased interest in this technology. Some important reasons for this interest are

- Feasibility : Large systems are becoming viable and cost-effective due to rapid advances in VLSI technology. Available systems have demonstrated remarkable performance for various applications.

- Supercomputing : Concurrent systems that use more than one computing element are already capable of providing 1–10 Gflops peak performance, and are expected to reach a teraflop speed in the near future.

- Scalability : The performance of concurrent systems is scalable in the number of computing elements. Real systems in the supermini range have demonstrated that the capacity of the systems can be easily increased by adding processor boards. Large systems having hundreds of processors have shown that the performance can be directly proportional to the the number of processors used for various scientific applications.

The architecture of concurrent systems varies widely and is changing rapidly, as the technology is far from maturity. There are many research problems and issues that need to be investigated.

1.1 Concurrent Computers

A concurrent computer consists of a set of nodes connected by a communication network (Figure 1.1). Individual nodes may have a processor or memory or both. In addition, nodes will have communication hardware that will provide the interface to th

depe

1.

2.

3. p

l

r

a

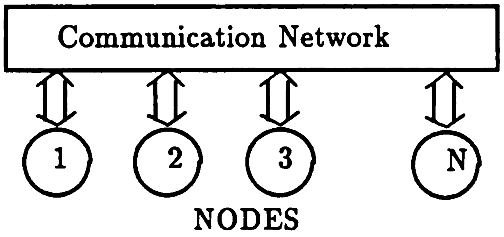


Figure 1.1: Overview of a concurrent computer

to the network. Concurrent computers can be divided into three broad categories depending on the organization of the memory and access to it by various processors.

1. Shared Memory systems : In these systems any processor can access any part of the memory with the same latency. The entire memory is global as it is physically and logically shared. Individual nodes will be either a processor or a memory, each with a communication interface to the network. The size of such systems tend to be small, at most 20-30 nodes, due to limitations of the communication network technology. Sequent's Balance and Encore's Multimax are examples of such commercially available systems.

2. Non-Uniform Memory Access (NUMA) systems : In these systems, as in shared memory systems, any processor can access any part of the memory. There is, however, a difference in access times to various parts of the memory. Each node has a processor and some memory called *local memory.* Access to the memory of another node, called *remote memory,* is slower than access to the local memory. The memory is physically distributed, but logically shared. Medium sized NUMA systems are available which have hundreds of nodes. BBN's Butterfly GP1000 [1] or its successor the TC2000 are examples of such commercially available systems.

3. Distributed memory systems or Multicomputers : In such a system, memory is physically and logically distributed. Each processor can only access its local memory. Interaction between processors is by passing messages. The systems are also known as message-passing concurrent computers or *multicomputers* [5].

Some of the largest parallel systems available belong to this category such as NCUBE/1 [48], iPSC/1 [67], Symult-2010 [74], iPSC/2 [4] etc.

Large multicomputers are gaining interest as they already have achieved some of the fastest computing rates, and seem to be the way to design a teraflop [1] computer in near future. A critical architectural component of all concurrent systems, including multicomputers, is the communication network.

1.2 Communication Networks

The network provides a means for the various nodes in a concurrent system to communicate, exchange information and coordinate their activities. The performance of the communication network is essential to the success of any concurrent computer. Extensive research has explored alternate network architectures over the last ten to fifteen years. Ideally one wants a completely-connected network where each node has a dedicated communication link to every other node. Implementation difficulties, however, prevent one from using such a network. The goal is therefore to design a network that is feasible and cost-effective, and can provide a performance comparable to that of a completely-connected network. Four types of networks have dominated current concurrent systems.

1. High-speed time-shared bus : In such a network a single bus, typically, connects all the nodes in the system [51]. As a result, communication time will be same for any pair of nodes. The main disadvantage is that the single bus becomes the bottleneck when the number of nodes or the size of the system reaches a certain threshold (20–30 for current technology). For large systems, we have to modify the network. One possibility is to use more than one bus. Researchers have proposed various ways of using more than one bus such as the hierarchically organized buses [78, 63], multi-dimensional structures as in Hyperbus [7] and so on. In spite of those modifications, contention for the bus continues to be a major problem with bus-based networks.

[1] A computing rate of 10^{12} floating point instruction per second.

2.

N no

any c

based

nodes

two n

quadr

make

3.

a com

crossb

emplo

appro

each s

netwo

some

since

Conte

signif

switch

pairs

4.

by hig

of othe

called

is used

is conn

then w

connect

network

2. Crossbar network : Here, a crossbar switch of N^2 cross points is used to connect N nodes to one another [51]. Such a network allows any node to communicate to any other node, and the time taken will be same for any pair of nodes as in bus based systems. One major advantage of the crossbar over the bus is that all the nodes can communicate simultaneously without any contention, provided that no two nodes communicate to the same node. However, the cost of a crossbar increases quadratically with the size of the system. In addition, power and pinout problems make a crossbar network unattractive for large systems.

3. Indirect or Multistage Interconnection Network (MIN) : These networks strike a compromise between the price/performance alternatives offered by the bus and the crossbar networks. An $N \times N$ MIN network connects N nodes to one another by employing multiple stages of banks of switches in the interconnection pathway. One approach possible with N as a power of 2 is to use $\lg N$ stages of $N/2$ switches, each switch being a 2×2 crossbar. Different ways of connecting successive stages of a network have been proposed, such as Omega, indirect binary n-cube and so on, with some of them shown to be equivalent. One advantage of a MIN is its expandability, since the number of stages increases only as $\lg N$. There are also disadvantages. Contention for paths such as the hot-spot problem can degrade the performance significantly. The network cost increases with the system size as the number of switches increases as $N \lg N$. Communication time remains the same for different pairs of nodes, provided there is no contention for resources within the network.

4. Direct or Point-to-point networks : In direct networks, nodes are connected by high-speed communication links. Each node is connected to only a small number of other nodes. Two nodes that are connected directly by a communication link are called *adjacent nodes*. The communication hardware in each node, called the router, is used to support communication between two non-adjacent nodes. If each node is connected to only a fixed number of other nodes irrespective of the system size, then we have N routers, one in each node, and $O(N)$ total communication links connecting the N routers. The cost of the network increases only as O(N). Direct networks are therefore much simpler to build than either MINs or crossbars. In

earlie

the co

than t

in tech

has b

Sh

syster

comp

due t

Direc

thesi

in m

direc

level

1.3

Dire

sche

dete

routi

betw

for a

the

bits.

T

a dir

the

comp

and

earlier direct networks, the communication time depended on the relative position of the communicating nodes. Communication between adjacent nodes was much faster than the communication between non-adjacent nodes. However, with recent advances in technology, the dependence of the communication time on the position of the nodes has been significantly reduced.

Shared memory systems use a bus-based network or a crossbar network. NUMA systems use MIN networks, where as direct networks seem to be the choice for multicomputers. The performance of direct networks has dramatically improved recently due to advances in technology, so they have become competitive to other networks. Direct networks can be used for both small and large concurrent systems. In this thesis, we study the performance of direct networks when used for message-passing in multicomputers. Such a performance study will be useful even if systems using direct-networks are organized as a shared-memory or a NUMA system at the user level.

1.3 Direct Network Architecture

Direct networks have four main characteristics: the switching technique, the routing scheme, the network topology and the channel bandwidth. The switching technique determines how communication takes place between two non-adjacent nodes. The routing scheme decides the sequence of channels, the path, used for communication between any two nodes. The network topology determines the set of adjacent nodes for any given node in the network. The bandwidth of a network channel depends on the width of the channel and the maximum rate at which the channels can transmit bits. The bandwidth will put an upper limit on the communication performance.

The last few years have witnessed changes in all the four main characteristics of a direct network used by a multicomputer. Current series of multicomputers such as the iPSC/2 [4, 28, 66] and the Symult-2010 [74] are called second-generation multicomputers, while earlier systems such as the Cosmic-cube [73], the NCUBE/1 [48], and the iPSC/1 [67] are called first-generation multicomputers [5]. The technology of

the

pute

feat

man

othe

and

1.3

The

are

swit

into

that

inter

repea

mess

Defi

D.

of the networks used by second-generation systems differs significantly from that of the first-generation multicomputers.

	Switching Technology	Routing Scheme	Network Topology	Channel Width
First Generation Multicomputers	Store-and-forward	Fixed	Hypercube	1-bit wide
	↓	↓	↓	↓
Second Generation Multicomputers	Circuit Switching or Wormhole routing	Fixed or Adaptive	2D Mesh or 2D Torus	8-32 bit wide

Figure 1.2: Trends in direct networks

The trends in network characteristics from first to second-generation multicomputers are shown in Figure 1.2. It is necessary to understand the changes in network features, as they have a significant influence on the network behavior and performance. As we see later, changes in one network characteristic influence changes in other network characteristics. We now discuss the four main network characteristics and trends in more detail.

1.3.1 Switching Technology

The three common switching techniques used in current multicomputer networks are shown in Figure 1.3. First generation multicomputers used store-and-forward switching which is common in wide-area and local-area networks. A message is split into packets if necessary. Each packet is then sent completely to an adjacent node that is nearer to the destination than the source. This adjacent node, called an intermediate node, relays the message to one of its adjacent nodes and the process repeats. Each transmission takes the message nearer to the destination until the message finally reaches it.

Definition 1 *Path length is the number of hops a message travels, and is denoted by D.*

F

for a

and

The

take

a an

deno

Proc

thro

signi

gene

s

i1

i2

d

T

stead

Also,

ing th

the so

Path length depends on the routing scheme which determines the channels used for a communication. First generation multicomputers used fixed routing schemes, and path length was equal to the shortest distance between the communicating nodes. The total communication time, t_{st}, is proportional to the number of hops. The time taken for one hop, t_h, is a linear function of message length L, i.e. $t_h = a + bL$. Here, a and b are system-dependent constants. The fixed overhead is denoted by a, and b denotes the effective message transmission rate. Hence, total communication time is

$$t_{st} = D * t_h = D(a + bL) = O(DL) \tag{1.1}$$

Processors at each intermediate node are involved in processing messages passing through the node. Software control at each intermediate node slows communication significantly as the number of hops, D, increases. Typical values for a and b for first generation computers are 1000 μs and 1 μs/byte.

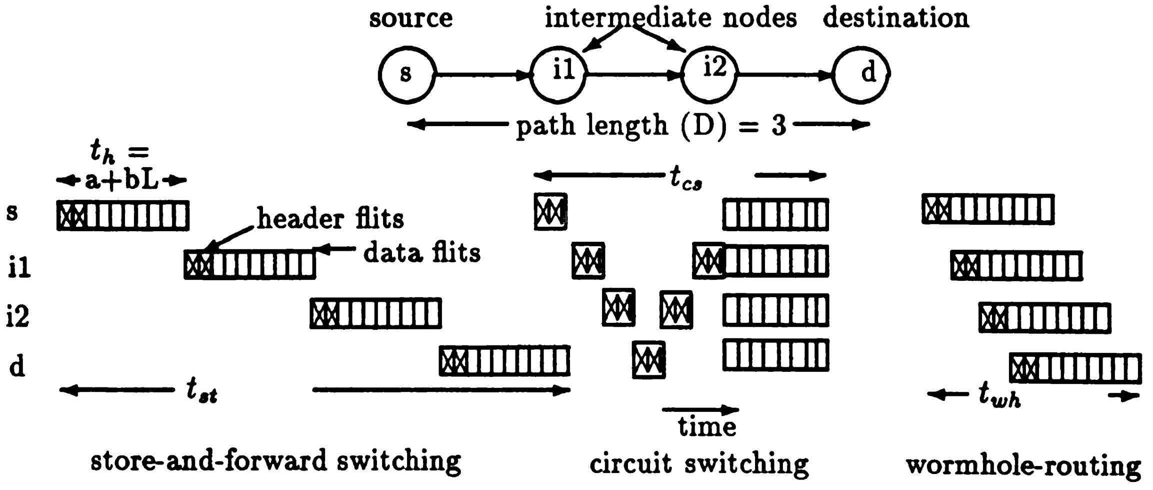


Figure 1.3: Common switching techniques

The switching technology has changed in second-generation multicomputers. Instead of store-and-forward switching, circuit switching or wormhole routing is used. Also, each node has dedicated communication hardware to process the messages passing through the node. In circuit switching, a header (one or two words) is sent from the source, and the header reserves channels on its way to the destination node thereby

establ
nodes.
the de
time f

Here,
estab
circui
0.36

W
pute
prop
head
word
wait
The
it m
swit
port
pack
nod
to n
in w
has
ther
each
per

establishing a physical communication path between the source and the destination nodes. Once the path is established, it takes the same time to send the message to the destination as it does to an adjacent node, t_h. Hence the total communication time for a circuit switched network is

$$t_{cs} = C_1 D + t_h = C_1 D + a + bL = O(D + L) \tag{1.2}$$

Here, C_1 is a system dependent constant that represents the time taken per hop to establish the path. One example of a second generation multicomputer that uses circuit switching is iPSC/2 for which C_1 is 10–30 μs, and a and b are 660 μs and 0.36 μs respectively [13].

Wormhole routing has also been used in commercial, second-generation multicomputers [32]. Wormhole-routing is a modified form of virtual cut-through switching proposed earlier for computer networks [54]. It is similar to circuit switching as the header reserves channels establishing a dedicated path for the message. However, words, called flits, of a message follow the header in a pipelined fashion instead of waiting until the entire path is established as was done in circuit-switched networks. The last message word, called the tail flit, automatically releases the channels as it moves towards the destination. This switching appears to be similar to packet switching where the packet size is reduced to one flit. There is, however, an important difference between this level of packet switching and wormhole routing. In packet switching, it is possible for several packets to accumulate in an intermediate node which requires each node to have a sufficient number of buffers for packets and to manage these buffers. In contrast, there is no need for a large number of buffers in wormhole routing. A flit is not sent to a subsequent node unless the previous flit has left that node. Hence, if a message header is blocked due to a reserved channel, then the entire flit stream following it stops progressing, with typically one flit in each intermediate node. Hence, one buffer that can hold a flit is enough per channel per node. Details of wormhole routing and actual implementations can be found in

Once
hop to
claime
respec
value
low o

N
tion t
worm
stead
first-
muni
node
reser
affec
as St

1.3.

The
that
cons
func
rout
dead

F

•

[37, 77, 32, 40, 35]. For wormhole routing, the communication time needed is

$$t_{wh} = a + C_2 D + b(L - 1) = O(D + L) \tag{1.3}$$

Once again, C_2 is a system dependent constant that represents the time taken per hop to establish the path. Current technology is quite impressive as the lowest values claimed for C_1 and b is about 20 ns for NDF [77] and 13 ns/byte for iWarp [14] respectively. However, a is still high for currently available systems as the lowest value claimed is 177 μsecs for Symult 2010 [75]. Systems are being designed to have low overheads such as the iWarp [14] and the J-machine [29, 30].

Note that all the three equations above for t_{st}, t_{cs} and t_{wh} gives the communication time under contention-free conditions. We see that both circuit switching and wormhole routing used in second-generation multicomputers take O($L + D$) time instead of the O(LD) time required by the store-and-forward switching used in the first-generation multicomputers. Since, $L \gg D$, O($L + D$) $\approx$ O(D), and hence communication time is not sensitive to D, the distance between the two communicating nodes. However, in both circuit switching and wormhole routing channels need to be reserved. The resulting contention for channels can become a serious problem and affect network performance. Hybrid switching schemes have also been suggested such as Staged Circuit Switching [3] which is similar to virtual cut-through.

1.3.2 Routing Scheme

The routing scheme decides the path taken by a message, and is another main feature that can significantly affect the network's performance. Multicomputers put a tight constraint on the time spent in deciding routing. We need to implement all routing functions in hardware to keep the processing times at an acceptable level. Hence, routing schemes should be simple. At the same time, it should ensure freedom from deadlocks and utilize the channels as best as possible.

Routing schemes can be divided into two main classes.

- Fixed routing schemes : In such schemes, the choice of a path depends only on

the

uti

- Ad

pa

Th

ch

ch

co

The

multico

the pat

path po

process

channel

their u

sufficie

Seco

which,

routing

overhea

and the

can bec

scheme

tion of

perform

We

be impl

perform

on the communicating nodes. A message between two given nodes always uses the same path. The choice of the path does not change with time or current utilization levels of various network channels.

- Adaptive routing schemes : These schemes tend to be smarter as the choice of path depends on the network state or the state of various network channels. The choice of a path taken by a message between a given pair of nodes may change with time. Adaptive routing schemes attempt to avoid already congested channels, strike a balance in the utilization of the channels and improve the communication times.

The routing scheme depends on the switching technology used. In first-generation multicomputers which used store-and-forward technology, it was essential to minimize the path lengths. Hence, the routing scheme was designed to select the shortest path possible between the communicating nodes. The routing decision was made by processors in each node by software, and hence the overheads were high. Network channels were never a bottleneck as messages were not injected fast enough to increase their utilization to high levels. Hence, fixed routing scheme were used and were sufficient.

Second-generation multicomputers use circuit switching or wormhole routing which, as explained in the previous section, are not sensitive to path lengths. Hence, routing schemes are not constrained to choose the shortest path possible. The routing overheads are significantly reduced as all routing decisions are made by the hardware, and the utilization of the channels can become quite high. Hence, network channels can become a bottleneck as individual communications reserve channels. Routing schemes may have to choose a path that reduces contention and balances the utilization of channels [10, 8]. Simple adaptive routing schemes may increase the network performance appreciably .

We see that the challenge is to design a routing scheme that is simple enough to be implemented in hardware, but at the same time good enough to keep the network performance at a high level. Both fixed and adaptive routing schemes have been proposed th

we do not

schemes to

1.3.3 N

The networ

the set of a

when each

is represen

represented

important

Degree

of a topolo

Diame

The diam

pairs of no

The ne

does not i

a low diar

(alternate

[16, 42, 50

hops slow

gies were

(Figure 1.

the vertic

$0 \leq d_i <$

$a_i = b_i$ fo

$k = 2$, and

the numb

shown to

proposed that are simple and avoid deadlock for modern networks [36, 60]. However, we do not have much information on the relative performance of various routing schemes to make a correct choice for the routing scheme.

1.3.3 Network Topology

The network topology defines the way nodes are connected; the topology determines the set of adjacent nodes for any given node. It is represented by the graph obtained when each node is defined as a vertex and each physical channel between two nodes is represented by a directed edge between corresponding vertices. The topology is represented by G(V,E), where V is the set of vertices and E is the set of edges. Two important features of a topology are

Degree : The degree of any node is the number of its adjacent nodes. The degree of a topology is the maximum degree of any node, i.e. $\max_{v \in V} d(V)$.

Diameter : The shortest distance between any two nodes u, v is denoted by $l(u, v)$. The diameter of a topology is the maximum value of the shortest distance over all pairs of nodes, i.e. diameter $= \max_{u,v \in V} l(u, v)$.

The network topology should have a small number of edges, a fixed degree (degree does not increase with the system size), regular (all vertices have the same degree), a low diameter, symmetric (network looks alike from each node), and fault-tolerant (alternate paths between vertices). Various network topologies have been suggested [16, 42, 50, 64, 71, 43, 38, 53], and compared [2, 56, 70]. Since an increased number of hops slowed communication in first-generation multicomputers, low diameter topologies were suggested and used [27, 39]. The hypercube topology was the most popular (Figure 1.4), which is a member of k-ary n-dimensional topology. In such a topology, the vertices can be numbered by a n-digit k-radix number $d = d_n d_{n-1} \ldots d_1$, where $0 \leq d_i < k$. Two vertices $a = a_n a_{n-1} \ldots a_1$ and $b = b_n b_{n-1} \ldots b_1$ are adjacent if $a_i = b_i$ for all i except $i = \bar{i}$, and $\bar{a}_i = \bar{b}_i \pm 1$. Hypercube topology is obtained when $k = 2$, and $n = \lg N$. The diameter and the degree of a hypercube are both $\lg N$, and the number of edges increases only as $O(N \lg N)$. The hypercube topology was also shown to have other topologies such as meshes and trees as subgraphs. Even though the degree and the number of edges are reasonable when compared to a completely connected network, they are still quite high for large systems having hundreds or thousands of nodes. The large number of edges results in increased wire densities and pin-outs, making network implementation difficult.

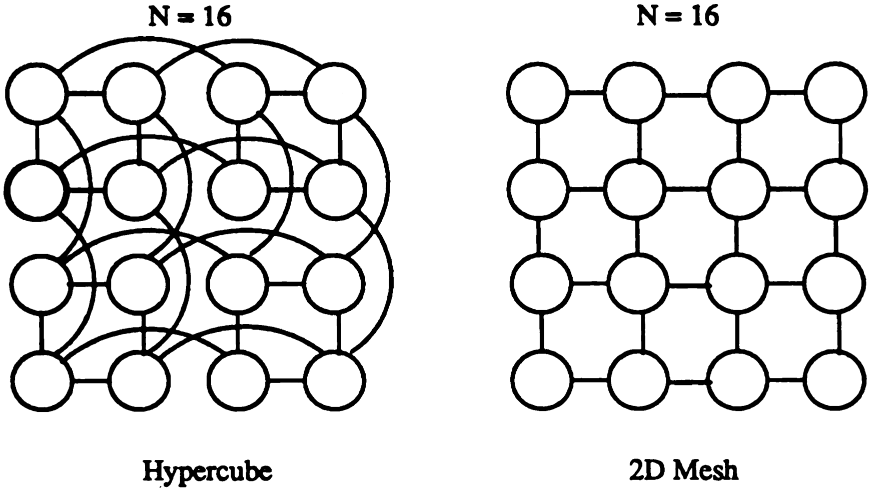


Figure 1.4: Two common network topologies

In the second-generation multicomputers, the distance between communicating nodes is no longer a serious problem, and hence a low diameter topology is no longer necessary. Simpler topologies such as the 2D mesh or torus (mesh with wrap-around edges) are worth considering. Mesh is also a member of k-ary n-dimensional topology, for which $n = 2$, and $k = \sqrt{N}$. In case of a 2D mesh, the number of edges increases only as $O(N)$ and the degree of a node is constant, i.e. independent of N. A mesh is therefore easier to implement. However, if we use circuit switching or wormhole routing and a 2D mesh or a 2D torus, then there will be increased contention for network channels as only $O(N)$ of them are available. Some important characteristics are summarized in Table 1.1. [2]

It is also possible to use switching element to increase the flexibility of a chosen topology, or even provide configurable networks i.e. networks whose topology can be changed by setting the active switching elements properly [76, 18, 41].

[2] The concept of bisection width, wire density and wire lengths are precisely defined in Section 2.2

I
I
E
V
V

1.3.4

The p
affect
i.e. nu
and th
on the
will no
individ
will ne

Th
that h
Hence
networ
level.
to incr
faster
can be
The tr
and co
to be s

	2D Mesh		Hypercube	
Degree	+	4,fixed	-	$\lg N$, variable
No. of edges	+	$2(N - \sqrt{N})$, $O(N)$	-	$(N \lg N)/2$, $O(N \lg N)$
Diameter	-	$2(\sqrt{N} - 1)$, $O(\sqrt{N})$	+	$\lg N$, $O(\lg N)$
Alternate paths	+	yes	+	yes
Bisection width	+	$\sqrt{N}$,$O(\sqrt{N})$	-	$N/2$, $O(N)$
Wire Density	+	Uniform	-	Non uniform
Wire lengths	+	Uniform and short , $O(1)$	-	Non uniform

(+(-) indicates an advantage (disadvantage))

Table 1.1: Important characteristics of mesh and hypercube topologies

1.3.4 Channel Bandwidth

The physical bandwidth of the network channels is another important feature that can affect the network performance. The bandwidth depends on the width of the channel, i.e. number of data bits that can be simultaneously transmitted across the channel, and the rate at which bits can be transmitted. The bandwidth puts an upper limit on the communication performance of a direct network. A higher channel bandwidth will not only increase the peak communication rates, but also reduce the possibility of individual channels getting saturated. High bandwidths may ensure that the channel will never become a bottleneck and avoid the need for adaptive routing schemes.

The maximum bandwidth possible depends on the network topology. A topology that has more edges is difficult to implement as it will increase the wire density. Hence, width of channels has to be reduced if we use a richer topology. Hypercube networks typically use bit-serial lines to keep the wire complexity to a manageable level. However, if a simpler topology such as the 2D mesh is used, then it is possible to increase the width of the channels to 8 or even 32 bits wide, and operate them at faster rates as the length of the channels will be small. Mesh networks can therefore can be designed to have channels with much higher bandwidths than hypercubes. The tradeoff between simpler, high-diameter networks that have wide, fast channels and complex, low-diameter networks that have slower channels is not clear, and needs to be studied.

In

netwo

increa

proce

comp

achie

does

help

maxi

1.4

The

featu

netwo

a dep

and c

A de

netwo

Ap

studie

archit

studie

will de

multic

perfor

the pe

about

the mo

Res

In addition to the changes in network switching technique, network topology and network channel bandwidth, the router or communication coprocessor is becoming increasingly complex and that in turn is minimizing the involvement of the node processor in handling communication. It is essential that all these developments are complemented by sufficient performance studies to clearly understand the issues, to achieve highest possible performance, and to ensure that the communication network does not become the bottleneck. Efficient communication performance in turn will help us to achieve efficient computation, i.e. actual computation rates close to the maximum possible rates of the system.

1.4 Network Performance Studies

The discussion in the previous section indicates that the choices of the four main features of a direct network are not independent of one another. A change in one network feature will have influence on the choice for another network feature. Such a dependence makes network design difficult. Rapid advances in network technology and changes in individual network features further complicates the design decisions. A detailed understanding of the dependencies and the effect of the choices on the network performance is essential to design a good network.

Appropriate performance studies help both the system designer and the user. The studies will help the system designer in understanding the drawbacks of the current architecture, and to come up with better network designs for future systems. The studies also help a user to get a better and deeper understanding of the factors that will degrade the communication performance when his application is run on modern multicomputers. Such an understanding will enable him to get the best possible performance on a given machine for his application. In this section, we briefly review the performance studies done so far and explain why there are still many questions about the performance for which answers are not clear. The discussion also explains the motivation for our thesis.

Researchers have attempted to compare the performance of different switching

techniq
been m
adaptiv
ing less
cently,
second-
cuit sw
effect, a
iPSC/1
problen
nodes.

Fix
works c
which a
routing
bered c
the net
The fix
increas
improv
to prov
been su
ity inst
that re

Rec
ability
is an im
compar
advanta
such as

techniques suggested for direct networks of multicomputers. Simulation studies have been made considering small hypercube systems [46, 26, 49, 15]. Results show that adaptive circuit switching and wormhole routing perform well for small systems having less than a hundred nodes. The studies do not consider larger systems. Recently, experimental results are available about the performance of iPSC/2 [13], a second-generation hypercube multicomputer that uses Direct-connect (a type of circuit switching) technology [66]. The results show that path lengths have negligible effect, and that the communication rate is about 5 times better than that of the iPSC/1. The results also show that contention for network channels can be a serious problem. The experiments were run on a small system having less than a hundred nodes.

Fixed routing schemes have been suggested that make the wormhole-routed networks deadlock-free. A general scheme has been suggested based on virtual channels which avoids deadlocks [36]. All channels are appropriately numbered and the fixed routing scheme is designed such that it always routes a message from a higher numbered channel to a lower numbered channel. Circular waits are thus avoided, making the network deadlock-free. The scheme can also be used for circuit-switched networks. The fixed routing scheme, however, fails to utilize all the channels and can result in increased contention for network channels. Virtual channels have been suggested to improve the performance under contention [34]. Virtual channels have also been used to provide adaptive routing schemes [52, 60]. A new adaptive routing scheme has been suggested for which network traffic is shown to saturate at 80% network capacity instead of 40% capacity if we use fixed routing [65]. Multicast communication that reduce network traffic has been suggested and evaluated [59, 58].

Recently, different network topologies have been compared considering their suitability to implementation [69, 62, 31, 33]. Different studies show that the wire density is an important implementation constraint. Hence, the network topologies have been compared assuming constant wire density [31, 33]. The studies clearly indicate the advantage of simpler networks such as the 2D mesh or torus over richer topologies such as the hypercube. Simpler networks can be designed to have higher bandwidth

chan
resul
comp
torus
The
sider
show
studi
of mo

HA
them
chann
perfor
hyper
systen
tions.
made
ground
be non
can deg
the non
network

Curr
large sy
an indiv
irrespect
tional to
communi
threshold
for larger

channels for a given wire density than rich networks. Higher bandwidth channels result in higher communication rates between adjacent nodes. Dally has recently compared various members of k-ary n-dimensional torus networks, which include 2D torus and hypercube, to determine the best dimension for a given network size [33]. The studies assume wormhole routing and uniformly distributed messages, and consider the effect of path length and the difference in channel bandwidths. The results show that the best dimension is 2 or 3 for systems up to ten thousand nodes. All these studies have resulted in a 2D or 3D mesh or torus topology being used for networks of modern multicomputers in place of a hypercube network used earlier.

High bandwidth channels of simpler, high-diameter topology networks enable them to provide higher performance if contention is negligible. Smaller number of channels and fixed routing, however, may result in increased contention and reduced performance. A recent study by Dally shows that 2D mesh performs better than hypercube even if we consider the effect of contention for the network channels for system sizes up to 4K nodes [33]. The theoretical study has been supported by simulations. The results hold only for uniformly distributed communication; an assumption made in the studies. Even though uniformly distributed messages give a common ground to compare different topologies, in reality it is difficult to achieve. Traffic will be non-uniform. Some nodes will suffer from contention more than others, and they can degrade the overall performance of an application. Hence, it is necessary to study the nonuniformities in network traffic and the effect of such nonuniform traffic on the network performance.

Current results also indicate that contention may become a serious problem for large systems [13, 33, 10]. Parallel algorithms are designed to be scalable. Hence, an individual node running a parallel task will have the same message injection rate irrespective of the system size. The network traffic [33], which is inversely proportional to the path length, will then increase with the system size. Results show that communication time increases exponentially once the network traffic reaches a certain threshold. Hence, we can expect a saturation effect at lower message injection rates for larger systems. Large systems are not yet available, and we do not have a clear idea of

a given

The

to syste

the last

[9, 6, 12

as com

commu

at a rea

has also

multico

is negli

claim s

sufficie

systems

architec

commu

traffic,

mappin

and he

commu

mappin

We

designe

The pe

plicatio

commu

idea of the message injection rates at which the saturation effects will be visible for a given communication pattern and system size.

The performance also depends on mapping—the way parallel tasks are assigned to system nodes. The problem of mapping has been explored by many researchers in the last decade in order to find the effect of mapping and to find the best mapping [9, 6, 12, 44, 72] Most of these studies have concentrated on minimizing path lengths as communication time increased linearly with path length. For a given mapping, communication may be appropriately routed to ensure utilization of channels remain at a reasonable level, and none of the channels are becoming a bottleneck [8, 11]. It has also been claimed that the mapping problem no longer exists in second generation multicomputers [75]. The claim is based on the observation that the path length effect is negligible, and that the peak node injection rates are limited. Even though the claim seems to hold for current, small systems, it is not clear if random mapping is sufficient for large, future systems having hundreds or thousands of nodes. Future systems can also have higher message injection rates as a result of improved node architecture and reduced fixed overheads per message. Mapping will decide the exact communication pattern for a given application and the non-uniformity in network traffic, and random mapping may result in saturation of network channels. A careful mapping and routing may balance the utilization of channels and network traffic, and hence avoid saturation of channels and the consequent exponential increase in communication times. It is necessary to study how non-uniformity depends on the mapping and how it increases with the system size.

We find that most of the performance studies done so far are from the system designer perspective, and help the system designer to make correct design choices. The performance studies do not give sufficient information for the user to run his applications well on the second-generation multicomputers, and ensure the best possible communication and overall performance.

1.5 Thesis Overview

We explained in Section 1.3 that direct communication networks used in second generation multicomputers are quite different from that used in first generation multicomputers. The discussion clearly showed that the choices of network features are dependent on one another, which makes network design difficult. In addition, changes in one network feature will influence changes in other network features. Section 1.4 explained that performance studies of modern networks are still inadequate and that many questions related to network performance remain unanswered. This thesis is an effort to improve our understanding of the performance of communication networks used in second generation multicomputers. Results from our studies will complement the results already available. In particular, we investigate the following important research issues about the network performance.

1. Performance of large, mesh-connected networks: Second generation multicomputers are using 2D/3D mesh or Torus networks instead of hypercube network. We do not have much information about the performance of mesh-connected wormhole-routed networks. The impact of the transition from hypercube networks to 2D or 3D mesh networks on the user is also not clear. Currently only small second generation multicomputers having less than a hundred node are available. Larger systems are expected in the near future.

2. Contention and Non-uniform Traffic: It is not clear how the contention problem scales up with the system size, and whether the effects of contention will be more easily visible in larger systems. We need to be able to model and predict the effect of contention on network performance. The results will be useful to a user of second-generation multicomputers in ensuring the best possible communication performance. Non-uniformity in traffic also needs to be modeled and analyzed to understand the variation in performance, and to predict the performance of the worst-case node.

3. The Mapping Problem: For second-generation multicomputers, minimizing path lengths no longer seems to be important, and random mapping seem to be good enough. Contention for network channels depends on the mapping, and hence

a mapping can affect the communication performance if the effect of contention is visible. We may need to redefine the mapping objective based on contention rather than on path length. It would be useful for a user to have a quick way of deciding if random mapping is sufficient for a problem at hand.

The three issues are related as we see later. The thesis is organized as follows. In Chapters 2 and 3, we study theoretically several issues that affect the communication performance. Theoretical studies provide a rough estimate of the performance. However, it is difficult to develop exact models to accurately predict the performance for many cases. Experimental and simulation studies are needed, which also help in validating the theoretical results. The theoretical studies are complemented by experimental studies in Chapter 4, and by simulations in Chapter 5.

In Chapter 2 we study the improvement in performance provided by simpler, high-diameter topology networks over richer, low-diameter topologies [19, 23, 17]. We do this by comparing the performance of two typical networks: hypercubes which represent rich topologies that have been quite popular and 2D mesh networks that represent a simple topology highly suitable for implementation. We take into consideration the difference in bandwidth, the effect of the switching technique, path lengths and contention for network channels. Results show that mesh networks are capable of $O(N \log N)$ improvement in performance. Path lengths are negligible as long as message lengths are much longer than path lengths. However, increased contention for network channels reduces the performance, and may take away the advantage of mesh networks.

We study the problem of contention for network channels for a given mapping in Chapter 3 [24, 20, 22]. We model the contention for network channels, and the nonuniformity in traffic in terms of *path contention level* and relate mapping to contention. We also show that the effect of contention is to put an upper bound on the rate at which messages can be injected into the network by a node—measured by *node traffic.* For a given communication pattern and mapping we show how to compute the saturation node traffic. We also analyze the case of random mapping, and show that random mapping may not be advisable for large systems having hundreds or thousands of nodes that use mesh networks. Careful mapping and routing will reduce contention and nonuniformity in traffic and increase the saturation levels for node traffic. We illustrate the advantage of a careful mapping in the case of several common communication patterns. We also develop a software module based on the theory which computes the upper bound on node traffic, and nonuniformities for a given communication pattern, and mapping.

In Chapter 4, we describe several experiments we conducted on a second-generation mesh-connected multicomputer, and analyze the results [21]. The experimental results demonstrate the performance offered by the modern mesh-connected wormhole-routed multicomputers at the user level. The results confirm that the effect of path lengths are negligible, and that the effect of contention may no longer be negligible as suggested by theoretical studies. The experimental results agree well with the theoretical results, and hence validate theoretical analysis in the earlier chapters. Analysis of the the results indicates that high, fixed overheads and smaller system sizes are currently limiting the effect of contention, and that the contention problem will be more serious in future, larger multicomputers. The studies identify other architectural drawbacks in current networks, and suggest improvements needed in future networks.

In Chapter 5, we describe a high-performance simulator that can be used to study the performance of large future multicomputers not yet available [25]. We discuss the design of the simulator to show that the design is flexible and efficient. The simulator implements all the essential features of a wormhole-routed network, but can still simulate large networks in reasonable time. Several simulation experiments were done to investigate questions about the network performance. The results of the experiments are presented. The experiments demonstrates the usefulness of the simulator. The results agree with the theoretical as well as the experimental results given in earlier chapters, thus assuring the credibility of our results. Finally we conclude our thesis and suggest some interesting research problems.

Chapter 2

COMMUNICATION PERFORMANCE : CONTENTION-FREE CASE

In this chapter, we analyze the simpler case of contention-free communication. We assume that there is no contention for network channels or any other network resources. The communication time then depends on the bandwidth of network channels, path length and the switching technique used. The channel bandwidth depends on the network topology, so communication performance depends on the network topology. We illustrate this dependency by analyzing the improvement in performance provided by 2D mesh networks relative to hypercubes. The analysis can be easily extended to evaluate the performance of other networks such as the 3D mesh.

We first introduce our notation in Section 2.1. In Section 2.2, we describe how implementation constraints can be taken into account in the analysis. The performance of mesh networks relative to hypercubes is analyzed in Section 2.3. Finally, in Section 2.4 we summarize the results of this chapter.

2.1 Terms and Notations

We already introduced the notations *L, D, N*, and τ_m in previous chapters which represent message length, path length, number of nodes in the system, and communication time per message respectively. In addition, we introduce the following terms and notations that will be used later in the analysis. We make the following assumptions about the network which hold for most implementations.

- All network channels are identical and have the same bandwidth.

- The transmissions are error-free.

Definition 2 Channel width, *denoted by* W, *is the number of bits that a physical channel can transmit simultaneously between two adjacent nodes.*

Definition 3 Channel rate, *denoted by* R, *is the peak rate in bits/sec at which bits can be transferred over the individual lines of a physical channel, and is equal to the reciprocal of the time delay to transfer one bit.*

Definition 4 Channel bandwidth, *denoted by* B, *is the peak rate in bits/sec at which data can be transmitted over a physical channel between adjacent nodes. Channel bandwidth is equal to product of channel width and channel rate, i.e.* $B = W \times R$.

Definition 5 Message aspect ratio, *denoted by* A, *is the ratio of the message length to the channel width, i.e.* $A = L/W$.

Definition 6 Header length, *denoted by* H, *is the length of a message header (generated and used by the communication network) in number of bits. For current systems,* H *is typically 1–4 bytes.*

Definition 7 Network latency, *denoted by* τ_l, *is the time from the instant the first bit of a message leaves the source PE to the instant the first bit reaches the destination PE.*

Definition 8 Network message time, *denoted by* τ_M, *is the time from the instant the first bit of a message arrives at the destination PE. to the instant the last bit of a message arrives at the destination PE.*

Definition 9 Network delay, *denoted by* τ_n, *is the total network delay for a message. We have* $\tau_n = \tau_l + \tau_M$.

Communication time per message, τ_m, will be sum of network delay and the processing delays per message at the source and the destination nodes. Since, we are not interested in studying the node architecture and its effect on communication time, we assume that communication time per message is equal to network delay, i.e. $\tau_m = \tau_n$. However, fixed overheads at the source and the destination nodes can be accounted for by adding them to the computation time between messages.

Definition 10 Effective Bandwidth, *denoted by E, is the rate at which a message can be transmitted between the source and the destination; it is equal to the ratio of the message length to the network delay, i.e. $E = L/\tau_n$.*

Any parameter with a superscript $h(m)$ indicates its value for a hypercube (2D mesh) network.

Definition 11 Channel bandwidth ratio, *denoted by R_b, is the ratio of the channel bandwidth of a mesh network to that of a hypercube network of the same size, i.e. $R_b = B^m/B^h$.*

Definition 12 Effective bandwidth ratio or Speedup, *denoted by R_e, is the ratio of the effective bandwidth for mesh network to that of a hypercube, i.e., $R_e = E^m/E^h$.*

Note that effective rates and speedup depend on the specific case under consideration. We use the speedup as the metric to compare the performance, and show how to derive the speedup for various cases in later sections.

2.2 Wire Complexity

A major implementation problem is the wires that connect various components of a system. In a large concurrent system, most of the wires will be the communication wires that connect different nodes of the system. The complexity of the connection is limited by the maximum wire density possible. The communication rates are limited by the wire lengths, and the majority of the power consumed by the system is used to drive the wires. Concurrent systems are therefore *wire-limited* [33, 31]. Any reasonable analysis of network performance should account for the wire complexity. Recently, several researchers have suggested ways to account for the wire complexity in theoretical analysis of the performance of communication networks of concurrent systems [62, 69, 31, 33].

2.2.1 Wire Density

First, we should be able to relate wire density to topology and system size, and measure wire density. The following definitions help us to quantify and define a metric for the wire density.

Definition 13 *A* linear layout *of a concurrent system is a 1-to-1 mapping of the N nodes onto the integers 0–(N-1).*

A linear layout in which node i is mapped onto integer i is called the *identity layout.*

Definition 14 *For a given linear layout and k,* cut width *is the number of edges connecting a processor numbered less than or equal to k, to a processor numbered higher than k, and is denoted by $\eta(k)$.*

Definition 15 *The* bisection width *of a topology, denoted by η_b, is the minimum value of $\eta(N/2)$ taken over all possible linear layouts of the given topology.*

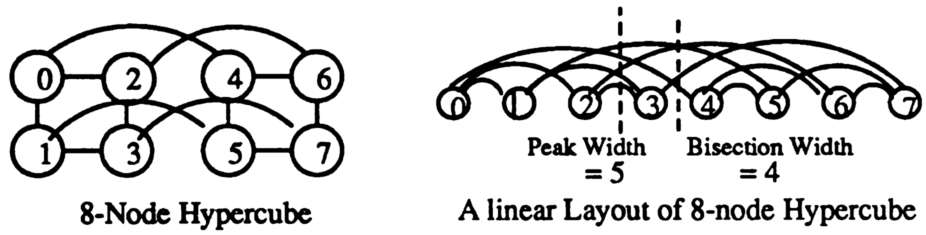
Definition 16 Bisection density *is the product of bisection width and the width of a channel i.e $\eta_b \times W$.*

Bisection density denotes the minimum number of wires needed to connect any two halves of a system. Researchers have proposed bisection density as a measure of wire density [31, 33]. Hence, different network designs can be compared by assuming equal bisection density. Here, we use a more general metric, *peak density* defined similar to bisection density.

Definition 17 *The* peak width *for a given linear layout is equal to the maximum value of cut width, i.e. $\max_{k=0}^{N-1} \eta(k)$. The peak width of a topology, denoted by η_m, is the minimum peak width over all possible linear layouts.*

Definition 18 Peak density *is the product of the peak width and the width of a channel, i.e $\eta_m \times W$.*

We denote the value of k which gives the maximum cut width by k_m. We can compare different topologies assuming same peak densities. We then find that the width of a channel will be inversely proportional to the peak width of the network topology.



8-Node Hypercube

A linear Layout of 8-node Hypercube

Figure 2.1: Linear layout and wire density : An example

Figure 2.1 shows the identity layout of an 8-node hypercube. The bisection width and the peak width are 4 and 5 respectively for the given layout. For an 8-node mesh (4×2), the values will be 2 and 3 respectively for an identity layout. For a given topology, the peak width is a function of the system size. We now derive peak width for hypercube and 2D mesh.

Let us consider the hypercube topology first. The dimension of the hypercube is denoted by n so $N = 2^n$. Let $\eta^i(k)$ denote the cut width if we consider only the i^{th} dimension edges of the hypercube so that

$$\eta(k) = \sum_{i=1}^{n} \eta^i(k) \tag{2.1}$$

It has been shown that the identity layout gives the minimum cut width for the hypercube topology [47], but the value of η_m is not derived for the identity layout. Before we can derive the value of η_m for the identity layout, we have to find the value of k_m, value of k for which η_k is maximum ($= \eta_m$).

For the identity layout of an N-node hypercube topology, observe the value of $\eta^i(k)$ as k increases. We find that $\eta^i(k)$ increases from 1 to 2^{i-1} and then decreases to 0, a

$\eta^1(k)$

He

the ra

Lemm

out th

symm

2 –

Pr

Ba

$\eta(1) =$

Hy

$\eta(2^{n-}$

In

n^{th} di

Equat

$\eta^1(k_j$

Nex

to 0, and the cycle repeats. We can state this observation precisely as

$$\eta^i(k) = \min(\bar{k}+1, 2^i - 1 - \bar{k}) \quad \text{where} \quad \bar{k} = (k \ \underline{\mathrm{mod}}\ 2^i) \quad \textit{and} \quad 0 \leq k \leq N-1 \tag{2.2}$$

Hence, $\eta^i(k) = \bar{k}+1$ over the range $0 \leq k < 2^i/2$, and $\eta^i(k) = 2^i - 1 - \bar{k}$ over the range $2^i/2 \leq k < 2^i$.

Lemma 1 *The cut width for an identity layout of hypercube is symmetric if we leave out the last value $\eta(N-1)$, i.e. $\eta(k) = \eta(N-2-k)$ for $0 \leq k \leq N-2$. The symmetric property also holds for edges of each dimension, i.e. $\eta^i(k) = \eta^i(N-2-k)$ for $1 \leq i \leq n$ and $0 \leq k \leq N-2$.*

Proof: We prove the lemma by induction on the dimension of the hypercube.

Basis : For $n = 1$ ($N = 2$), we have the cutwidth values $\eta(0) = \eta^1(0) = 1$ and $\eta(1) = \eta^1(1) = 0$ which satisfy the lemma.

Hypothesis : The lemma holds for an $(n-1)$-dimension hypercube, i.e $\eta^i(k) = \eta^i(2^{n-1} - 2 - k)$ for $0 \leq k \leq 2^{n-1} - 2$, and hence $\eta(k) = \eta(2^{n-1} - 2 - k)$.

Induction : Consider n-dimension cube, $N = 2^n$. First, consider the highest, i.e. n^{th} dimension edges. We can show that $\eta^n(k)$ satisfies the symmetric property. From Equation 2.2, we have for $0 \leq k \leq N-2$

$$\begin{aligned}
\eta^n(k) &= \min(k+1, N-1-k) \\
&\qquad \text{as} \quad \bar{k} = (k \ \underline{\mathrm{mod}}\ 2^n) = (k \ \underline{\mathrm{mod}}\ N) = k \quad \text{and} \quad 2^n = N \\
&= \min(N-1-k, k+1) \ \text{; after swapping the two terms} \\
&= \min((N-2-k)+1), N-1-(N-2-k)) \ \text{; after rewriting the terms} \\
&= \eta^n(N-2-k) \ \text{; by Equation 2.2}
\end{aligned} \tag{2.3}$$

Next, consider the lower dimension edges, i.e. $i < n$, we have from Equation 2.2

$$\begin{aligned}
\eta^i(k) &= \eta^i(2^{n-1} - 2 - k) \quad \text{from the induction hypothesis} \\
&= \eta^i(2^{n-1} + 2^{n-1} - 2 - k) \quad \text{as} \quad x \ \underline{\mathrm{mod}}\ 2^i = (x + 2^{n-1}) \ \underline{\mathrm{mod}}\ 2^i \\
&= \eta^i(2^n - 2 - k)
\end{aligned}$$

$$= \quad \eta^i(N-2-k) \qquad\qquad \forall i < n \tag{2.4}$$

From Equations 2.3 and 2.4 we have $\eta^i(k) = \eta^i(N-2-k)$ for $1 \leq i \leq n$. Combining this result with Equation 2.1, we have $\eta(k) = \eta(N-2-k)$, and induction completes the proof for the lemma. □

Due to the symmetry of the cut width values, we find that peak width should occur in both halves of the layout. Hence, we have the following corollary.

Corollary 1 *The peak width for an identity layout of the hypercube occurs in the first half of the layout, i.e.* $0 \leq k_m \leq N/2 - 1$.

The next lemma restricts the possible value of k_m further.

Lemma 2 *The cut width for an identity layout of an n-dimensional hypercube,* $\eta(k)$*, is related to the cutwidth for an identity layout of an* $(n-2)$*-dimensional hypercube,* $\eta'(k)$*, as*

$$\eta(k) \quad = \quad N/2 + \eta'(k - N/4) \geq \eta(k - N/4) \qquad \forall k \quad N/4 \leq k < N/2$$

Proof: From Equation 2.1, we have

$$\eta(k) \quad = \quad \eta^n(k) + \eta^{n-1}(k) + \sum_{i=1}^{n-2} \eta^i(k) \tag{2.5}$$

We can use Equation 2.2 to find the values of the terms on the RHS of the above equation, for the range of k values considered $(N/4 \leq k < N/2)$.

$$\begin{aligned} \eta^n(k) &= \min(k+1, 2^n - 1 - k) \quad ; \text{as} \quad \bar{k} \;=\; k \,\underline{\text{mod}}\, 2^n = k \,\underline{\text{mod}}\, N = k \\ &= k + 1 \;\; ; \quad \text{as} \quad 2^n - 1 - k \geq k + 1 \end{aligned} \tag{2.6}$$

Similarly, we can show in case of the second term of Equation 2.5 that

$$\eta^{n-1}(k) \quad = \quad \min(k+1, 2^{n-1} - 1 - k) \quad = \quad 2^{n-1} - 1 - k \tag{2.7}$$

as $2^{n-1} - 1 - k \geq k + 1$ for the range of k values considered

In case of the third term on the RHS of the Equation 2.5, we have

$$\eta^i(k) \;=\; \eta^i(k - N/4) \;\text{ as }\; k \;\underline{\text{mod}}\; 2^i = (k - N/4) \;\underline{\text{mod}}\; 2^i \qquad (2.8)$$

We can use Equations 2.6, 2.7 and 2.8 in Equation 2.5 to get the following result for $N/4 \leq k < N/2$.

$$\begin{aligned} \eta(k) &= (k+1) + (2^{n-1} - 1 - k) + \sum_{i=1}^{n-2} \eta^i(k - N/4) \\ &= 2^{n-1} + \sum_{i=1}^{n-2} \eta^i(k - N/4) \\ &= 2^{n-1} + \eta'(k - N/4) \quad ; \text{ by Equation 2.1} \qquad (2.9) \end{aligned}$$

By a similar argument, we can show that if $0 \leq k < N/4$

$$\begin{aligned} \eta(k) &= \eta^n(k) + \eta^{n-1}(k) + \sum_{i=1}^{n-2} \eta^i(k) \\ &= (k+1) + (k+1) + \eta'(k) \\ &\leq 2^{n-2} + 2^{n-2} + \eta'(k) \\ &\leq 2^{n-1} + \eta'(k) \quad 0 \leq k < N/4 \qquad (2.10) \\ \text{or } \eta(k - N/4) &\geq 2^{n-1} + \eta(k - N/4) \;\; ; \;\; N/4 \leq k < N/2 \qquad (2.11) \end{aligned}$$

Using Equation 2.11 in Equation 2.9, we get

$$\eta(k) \;=\; 2^{n-1} + \eta'(k - N/4) \geq \eta(k - N/4) \quad (N/4 \leq k < N/2) \qquad (2.12)$$

which completes the proof. □

From the lemma it follows that $\eta(k) \leq \eta(k + N/4)$ for $0 \leq k < N/4$. Hence, k_m cannot occur in the range 0–$(N/4 - 1)$. From Corollary 1, we know that k_m is in the range $0 \leq k < N/2$. Hence, we have the following corollary.

Corollary 2 *The cut width is maximum in the second quarter of the layout, i.e.* $N/4 \leq k_m < N/2$.

We find that $\eta(k)$ reaches the maximum value at a value of $(k + N/4)$, if η' reaches maximum at a value of k. From Lemma 2, the maximum values of $\eta(k)$ and $\eta'(k)$ are related as given by the first part of its equation. Hence, we have the following corollaries from Lemma 2.

Corollary 3 *The values of k_m for n and $n-2$ dimensional hypercubes (N and $N/4$ nodes) are related as $k_m(N) = N/4 + k_m(N/4)$.*

Corollary 4 *The peak width of an n-dimensional hypercube is related to the $(n-2)$-dimensional hypercube as $\eta_m(N) = N/2 + \eta_m(N/4)$.*

We are now in a position to state and prove the main theorems. The next theorem states the value of k_m.

Theorem 1 *The cut width for an identity layout of an n-dimensional binary hypercube is maximum when $k = \lfloor N/3 \rfloor$.*

Proof: The proof is by induction on the dimension of the hypercube.

Basis : For a hypercube of dimension 1 ($N = 2$), the cut width values for the identity layout are (1,0). We find that the theorem is satisfied as $k_m = 0 = \lfloor 2/3 \rfloor$.

For a hypercube of dimension 2 ($N = 4$), the cut width values for the identity layout are (2,2,2,0). The theorem is once again satisfied as $k_m = 1 = \lfloor 4/3 \rfloor$.

Hypothesis : The cut width of an identity layout of an i-dimensional binary hypercube is maximum when $k = \lfloor 2^i/3 \rfloor$ for all $i \leq n$.

Induction : Consider the n-dimensional hypercube (N = 2^n). From Corollary 3, we have

$$\begin{aligned}
k_m(N) &= N/4 + k_m(N/4) \\
&= N/4 + \lfloor \frac{N/4}{3} \rfloor && \text{; by the induction hypothesis} \\
&= \lfloor N/4 + (N/4)/3 \rfloor && \text{; since N/4 is an integer} \\
&= \lfloor N/3 \rfloor
\end{aligned}$$

and induction completes the proof. □

The next theorem gives the value of the peak width.

Theorem 2 *The peak width for an n-dimensional hypercube topology is* $\lfloor 2N/3 \rfloor$.

Proof: As mentioned earlier, it has already been proven that the identity layout gives the minimum peak width. Hence, we only have to prove the theorem for the case of the identity layout. We once again prove the theorem by induction on the dimension of the hypercube.

Basis : For a hypercube of dimension 1 ($N = 2$), the cut width values for the identity layout are (1,0). We find that the theorem is satisfied as $\eta_m = 1 = \lfloor (2 \times 2)/3 \rfloor$.

For a hypercube of dimension 2 ($N = 4$), the cut width values for the identity layout are (2,2,2,0). The theorem is once again satisfied as $\eta_m = 2 = \lfloor (2 \times 4)/3 \rfloor$.

Hypothesis : The peak width for an identity layout of an i-dimensional binary hypercube is $\eta_m = \lfloor (2 \times 2^i)/3 \rfloor$ for all $i \leq n$.

Induction : Consider the n-dimensional hypercube now. By Corollary 4, we have

$$\begin{aligned}
\eta_m(N) &= N/2 + \eta_m(N/4) \\
&= N/2 + \lfloor 2N/(3 \times 4) \rfloor \quad \text{; by the induction hypothesis} \\
&= \lfloor N/2 + N/6 \rfloor \quad \text{; as N/2 is an integer} \\
&= \lfloor 4N/6 \rfloor \;=\; \lfloor 2N/3 \rfloor
\end{aligned}$$

and the induction completes the proof. □

Now that we have proven the peak width of the hypercube topology, we consider the peak width for the other topology—2D mesh.

Theorem 3 *The peak width for an identity layout of a 2D square mesh of size N* $(\sqrt{N} \times \sqrt{N})$ *is* $\sqrt{N} + 1$.

Proof: Consider the identity layout for a 2D mesh network, in which a node (i, j), $0 \leq i, j < \sqrt{N}$ is mapped onto the integer $i\sqrt{N} + j$. Let $\eta^0(k)$ and $\eta^1(k)$ denote

the contribution to the cut width by dimension 0 (row) and dimension 1 (column) edges of the mesh so

$$\eta(k) = \eta^0(k) + \eta^1(k) \tag{2.13}$$

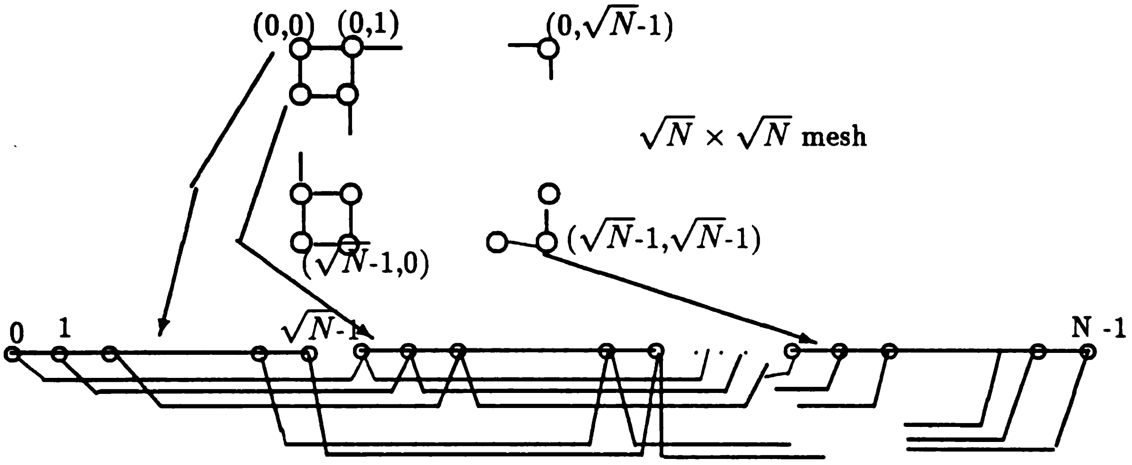


Figure 2.2: Identity layout of 2D mesh

The identity layout of 2D mesh is shown in Figure 2.2 which is basically a row-major layout. A little observation shows that

$$\begin{aligned} \eta^0(k) &= 0 && \text{if} \quad (k+1) \underline{\text{mod}} \sqrt{N} = 0, \\ &= 1 && \text{otherwise} \qquad (2.14) \\ \eta^1(k) &= k+1 && 0 \le k \le \sqrt{N}-2 \\ &= \sqrt{N} && \sqrt{N}-1 \le k \le (N-\sqrt{N}-1) \\ &= N-k-1 && (N-\sqrt{N}) \le k \le (N-1) \qquad (2.15) \end{aligned}$$

From equations 2.13, 2.14 and 2.15 we find that maximum value of cut width or peak width is $\sqrt{N}+1$ which proves the theorem. □

We assume that the identity layout gives the lowest peak width, and hence the peak width for mesh topology is $\sqrt{N}+1$ as proven in Theorem 3. Theorem 2 and 3 shows that the peak wire density of the mesh and hypercube are of $O(\sqrt{N})$ and $O(N)$

respectively. We have the following theorem if networks are limited by wire density.

Theorem 4 *Mesh networks can have $\frac{2\sqrt{N}}{3}$ wider channels than hypercubes for the same wire density.*

Proof: We can get the ratio of channel widths by equating the peak wire densities for the hypercube and the 2D mesh. By using the definition of peak density from the previous section (Definition 18) and the values of the peak width for the hypercube topology (Theorem 2) and 2D mesh (Theorem 3), we have

$$(\sqrt{N}+1)W^m = \lfloor 2N/3 \rfloor \; W^h$$

$$\frac{W^m}{W^h} \approx \frac{2N/3}{\sqrt{N}} = \frac{2\sqrt{N}}{3}$$

which completes the proof. □

2.2.2 Wire Length

Apart from wire densities, we must also account for the effect of the length of the wires on the communication rates. Three models have been suggested by researchers [33] to relate the wire lengths to the communication rates. In the *constant delay* model, we assume that the rate is independent of the the wire length. This model is appropriate if wire lengths are small. In the *linear delay* model, which is appropriate if wires are long and propagation delay is the limiting factor, rates are assumed to be inversely proportional to the wire lengths. For medium wires, the *logarithmic delay model* is used where rates are inversely proportional to the logarithm of the wire lengths. Since wire lengths are shorter in 2D meshes than in hypercubes, linear or logarithmic delay models make the communication rates in 2D meshes greater than in hypercubes. We assumed that all physical channels of a network are identical so they are designed to operate at the same rate. Hence, the longest wires (worst case) determine the communication rates. The ratio of the length of the longest wires in mesh and hypercube can be calculated from the topological properties, which will give the ratio of the communication rates for the three delay models.

Theorem 5 *The ratio of the length of the longest wire of a hypercube network, l^h, to that of a mesh network, l^m, of the same size will be at least $(\sqrt{2N}-\sqrt{2})/\lg N$, if we use a two dimensional layout.*

Proof: Without loss of generality assume that a node can be laid out in a square area of size $a \times a$ units. The best way to lay out a mesh network is $\sqrt{N}$ rows by $\sqrt{N}$ columns, so each wire will be of length a (if assumed to be equal to the distance between the centers of the layouts). On the other hand, the best lay out for hypercube is not immediately obvious. Here, we derive a lower bound on the length of the longest wire. In the case of a hypercube network, the most compact layout will again be in a square area of $\sqrt{N}a \times \sqrt{N}a$ assuming a 2D layout. This is true as we have a total layout area of Na, and a square layout minimizes the distance between diagonally opposite corner nodes. The distance between the centers of the farthest nodes (diagonally opposite corner nodes) will then be $(\sqrt{2N}-\sqrt{2})a$. The maximum path length between any two nodes in a hypercube, and hence the diagonally opposite corner nodes, can be at most $\lg N$. Hence, we get that the length of each of the wires along the path between diagonally opposite corner nodes must be at least $((\sqrt{2N}-\sqrt{2}) \times a)/\lg N$. Hence, the ratio of the lengths of the longest wires must be at least $((\sqrt{2N}-\sqrt{2}) \times a)/(\lg N \times a)$, which proves the theorem. □

The observation of the lengths of the wires in the two networks leads to the following theorem on the ratio of channel rates.

Theorem 6 *The ratio of the channel rate of a mesh network to that of the hypercube network for linear, logarithmic, and constant delay models is given by $(\sqrt{2N}-\sqrt{2})/\lg N$, $(3.89+\lg(N)-2\lg(\lg N))/2.89$, and 1 respectively.*

Proof: For the constant delay model, the ratio is 1, as channel rates do not depend on the wire length. In the case of the linear delay model, channel rates will be inversely proportional to the wire lengths. The ratio of channel rates then is the same as the ratio of the longest wires in the two systems which is derived in Theorem 5. In the case of the logarithmic delay model, the ratio R^m/R^h will be $(1 + \ln l^h/l^m)$, where $l_h(l_m)$ is the length of the longest wire in a hypercube (2D mesh) [33]. Hence

$$\begin{aligned} R^m/R^h &= (1 + \ln\left(l^h/l^m\right)) \\ &\approx 1 + \ln\left(\sqrt{2N}/\lg N\right) \quad ; \text{ from Theorem 5} \\ &= 1 + 0.5\ln(2) + 0.5\ln(N) - \ln(\lg N) \\ &= 1 + 0.5/\lg e \times (1 + \lg N + 2\lg(\lg N)) \\ &= (3.89 + \lg N + 2\lg(\lg N))/2.89 \end{aligned}$$

which completes the proof. □

2.3 Performance Comparison

We now derive expressions for the speedup provided by the mesh network over the hypercube network when both are of size N. The derivations are done at three levels as in the experiments to measure the network performance (Chapter 4). First, contention-free adjacent node communication is considered. Such a speedup is achieved for applications that *naturally* map onto the mesh network resulting in communication between adjacent nodes only. We then consider contention-free, non-adjacent node communication. Here, communicating nodes are no longer adjacent so communication performance is affected by path length D. Such speedup is achieved for applications that are not communication intensive so the effect of contention for network channels on communication performance can be neglected. Finally, we consider the speedup while allowing contention for network channels.

By the definitions of effective bandwidth and speedup from Section 2.1 (Definition 10 and 12), we have

$$R_e = \frac{E^m}{E^h} = \frac{L/\tau_n^m}{L/\tau_n^h} = \frac{\tau_n^h}{\tau_n^m} \tag{2.16}$$

We can t

single m

2.3.1

When a

$D = 1.$

have fro

We fin

is the

the de

W

band

obtai

have

Theo

hype

F

that

We can therefore derive the speedup by computing the time taken to communicate a single message of length L for the case under consideration.

2.3.1 Contention-free Adjacent-node Communication

When all communication is limited to adjacent nodes only, we have the path length $D = 1$. In addition, there is no contention for network channels. In such a case we have from Equation 2.16

$$\begin{aligned} R_e &= \frac{\tau_n^h}{\tau_n^m} = \frac{(L+H^h)/B^h}{(L+H^m)/B^m} = fR_b \approx R_b \\ &\text{since} \quad L \gg H, \quad \text{we have} \quad f = \frac{(L+H^h)}{(L+H^m)} \approx 1 \end{aligned} \qquad (2.17)$$

We find that speedup is almost equal to the channel bandwidth ratio, which in turn, is the product of the ratio of channel bandwidths and the ratio of channel rates from the definition i.e.

$$R_b = \frac{B^m}{B^h} = \left(\frac{W^m}{W^h}\right) \times \left(\frac{R^m}{R^h}\right) \qquad (2.18)$$

We can multiply the results of Theorems 4 and 6 to get the ratio of channel bandwidth for all three delay models: constant, logarithmic, and linear. The results obtained are stated in the following theorem. (In the case of the linear delay model, we have ignored the term $\sqrt{2}$ in the numerator for the ratio of channel rates (Theorem 6)).

Theorem 7 *The ratio of channel bandwidths of a 2D mesh network to that of a hypercube network for the three delay models are given in the Table 2.1.*

Delay Model	Constant	Logarithmic	Linear
R_b	$\frac{2\sqrt{N}}{3}$	$\frac{\sqrt{N}(3.89+\lg_2 N-2\lg_2(\lg_2 N))}{4.33}$	$\frac{2\sqrt{2}N}{3\lg_2 N}$

Table 2.1: Ratio of Channel Bandwidths for the three delay models

From Equation 2.17, the speedup R_e for contention-free adjacent-node communication is approximately equal to the channel bandwidth ratio given in Table 2.1.

As a result, we find that the speedup is a function of the system size. From the Table 2.1 we find that the speedup increase as $O(\sqrt{N})$, $O(\sqrt{N}\lg N)$ and $O(N/\lg N)$ for constant, logarithmic and linear delay models respectively. In Table 2.2, we give the speedup value for different system sizes. The speedup is appreciable for systems having hundreds of nodes.

N	$\frac{N}{\log_2 N}$	Speedup (R_e)		
		Linear delay	Logarithmic delay	Constant delay
64	11	10.4	8.9	5.3
128	18	17.2	13.7	7.5
256	32	30.2	21.8	10.7
512	57	53.7	34.1	15.0
1024	103	97.1	53.6	21.3
2048	187	176.3	83.3	30.1
4096	341	321.8	129.6	42.7

Table 2.2: Speedup (R_e) for different system sizes

Table 2.3 shows the channel widths and channel rates for several commercial systems. The ratio of channel widths and rates agrees with that predicted by analysis (Table 2.1). For example, if we compare iPSC/1 and MRC, both of which can have up to 128 nodes, the channel width ratio 8/1 = 8 and channel rate ratio 20/10 = 2. The channel width ratio is equal to R_e for the constant delay model. From Table 2.2 for $N = 128$, we find channel width ratio is 7.5 (value of R_e for the constant delay model) which matches the actual value 8. From Table 2.2 we can also derive the ratio of channel rates, assuming the linear delay model, by dividing the value of R_e for the linear delay model by channel width ratio. Hence, the channel rate ratio for $N = 128$ is $17.2/7.5 \approx 2$, which also matches the actual ratio. We find that the results of the analysis gives a reasonable estimate of the increased bandwidth possible by using a 2D mesh instead of a hypercube network for a given system size, when we consider the suitability of the network for implementation.

In the subsequent sections, we show that speedup will be lower than the channel bandwidth ratio due to longer path lengths and contention for network channels. In

System	Network Topology	W	R	B = W×R
Ametek(sys 14)	hypercube	1	3	3
iPSC/1	hypercube	1	10	10
Torus	2D Torus	8	8	64
MRC	2D mesh	8	20	160
iWarp	2D mesh	32	10	320
NDF	2D mesh	9	50	450

Table 2.3: Channel width, rate and bandwidth for some commercial systems

fact, the channel bandwidth ratio is an upper bound for the speedup, i.e. $R_e \leq R_b$. From now on, we will assume the constant delay model and use the corresponding value for the channel bandwidth ratio, as they have been found to be appropriate for current technology and system sizes [33].

2.3.2 Contention-free Non-adjacent-node Communication

For many applications it may not be possible to map the communicating tasks onto adjacent nodes. The path length will be more than 1 in such a case. If communication is not frequent, contention can still be ignored. We analyze the speedup for such a case in this section.

The network latency will be the time taken by the header flit to reach the destination node. For each hop, communication time for the header is H/B. For a path of length D, the network latency will be $D \times H/B$. The data flits follow the header in a pipelined fashion. Hence, once the message header arrives at the destination node, remaining data flits arrive at the rate of B. The total time for all the data flits will then be L/B. Hence, the network delay for communication over a path of length D will be

$$\tau_n = \tau_l + \tau_M = D \times H/B + L/B = (D \times H + L)/B \tag{2.19}$$

The speedup can be derived as it is the ratio of the communication times. From equations (2.16)

F

Normally, $D^m \geq$
and the value of
header length) a
the length of th
path lengths in
for 2D mesh an

$D^m_{avg} \approx$

$D^h_{avg} =$

The speedu
lengths by sub
distance from F
$D^m = D^m_{avg}$, D
by the followin

The speedu
that as long as
length does not
later clearly co
the header is lo
as L' decreases

equations (2.16) and (2.19), we have

$$R_e = \frac{\tau_n^h}{\tau_n^m} = \frac{B^m}{B^h} \times \frac{D^h H + L}{D^m H + L} = R_b \times \frac{D^h + L'}{D^m + L'} = f R_b \qquad (2.20)$$

$$\text{where} \quad L' = L/H \quad \text{and} \quad f = \frac{D^h + L'}{D^m + L'}$$

Normally, $D^m \geq D^h$. Hence, we find that speedup is a fraction of R_b ($R_e \leq R_b$), and the value of the fraction f depends on the message length (expressed in units of header length) and the path length. If the header length is one flit, then L' will be the length of the message in flits. Let D_{avg} and D_{max} be the average and maximum path lengths in a hypercube and a 2D mesh. The values of D_{avg} and D_{max} are known for 2D mesh and hypercube networks [65, 33], and are stated below.

$$\begin{array}{ll} D^m_{avg} \approx 2\sqrt{N}/3 & D^m_{max} = 2(\sqrt{N} - 1) \\ D^h_{avg} = \lg N/2 & D^h_{max} = \lg N \end{array} \qquad (2.21)$$

The speedup can be expressed as a function of system size for various message lengths by substituting the values of R_b (from Theorem 7) and the values of the distance from Equation (2.21) into Equation 2.20. The speedup for an average node ($D^m = D^m_{avg}$, $D^h = D^h_{avg}$) and a worst case node ($D^m = D^m_{max}$, $D^h = D^h_{max}$) are given by the following equations.

$$R_e = R_b \times \frac{3(\lg N + 2L')}{2(2\sqrt{N} + 3L')} \quad \text{(average case)} \qquad (2.22)$$

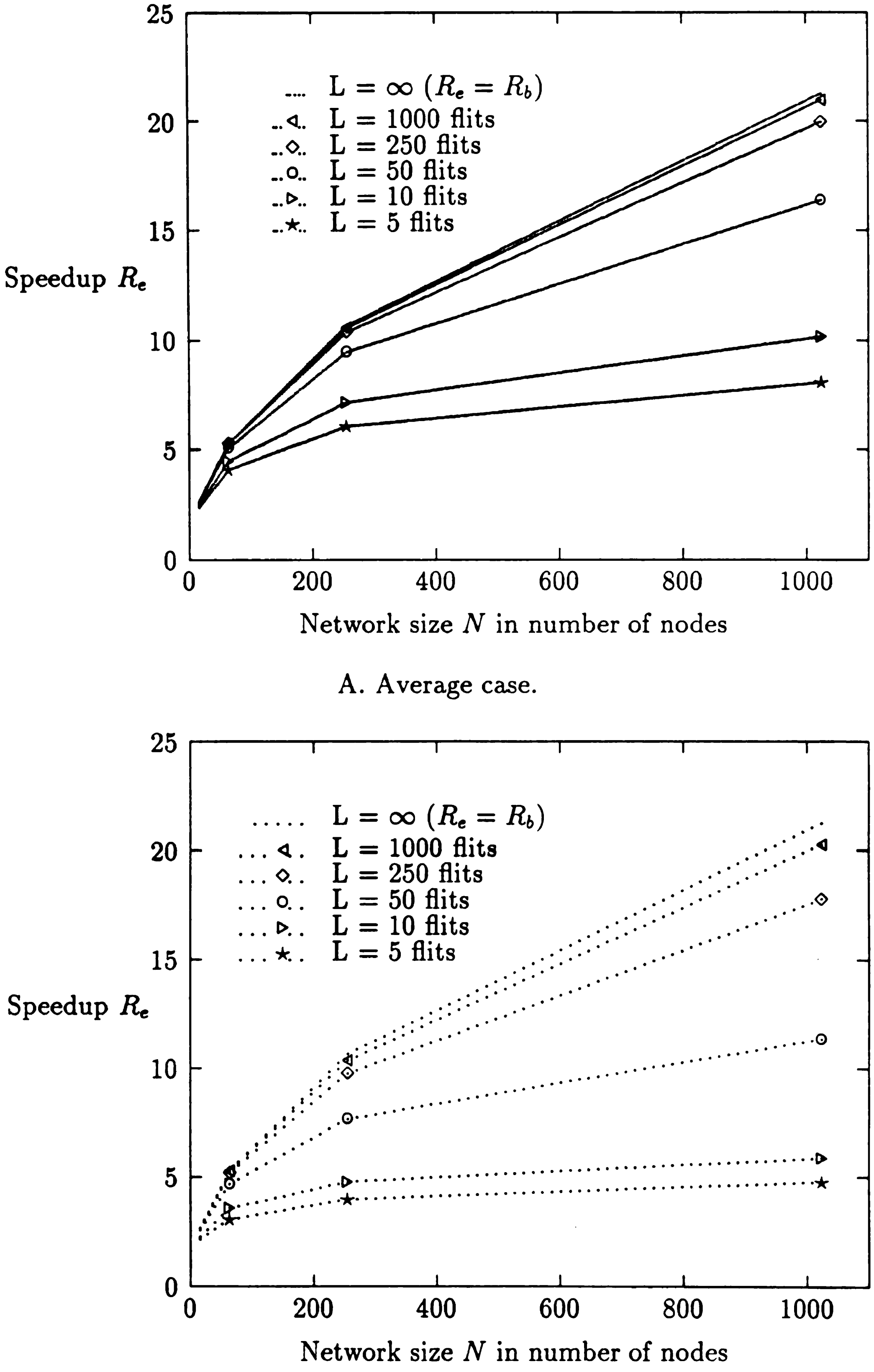
$$R_e = R_b \times \frac{(\lg N + L')}{2(\sqrt{N} - 1) + L'} \quad \text{(worst case)} \qquad (2.23)$$

The speedups are plotted for various message lengths in Figure 2.3. We observe that as long as the message length is much greater than the path length, the path length does not affect the speedup and $R_e = R_b$. Experimental results in Section 4.2.2 later clearly confirm this conclusion. We assumed the header length to be 1 flit. If the header is longer, the speedup will be further reduced for a given message length (as L' decreases for a given L).

Spec

Spe

Figure



A. Average case.

B. Worst case.

Figure 2.3: Speedup R_e for contention-free non-adjacent node communication

2.4

The re

1. M

   t

2. U

   t

   I

   c

Howev

than I

perforn

we sho

for a g

2.4 Conclusion

The results of our studies in this chapter leads to the following conclusions.

1. Mesh networks can be designed with one or two order higher bandwidth channels than hypercube networks, and hence have the potential for greater performance.

2. Under contention-free conditions meshes perform better than hypercubes due to higher bandwidth channels except when message sizes are comparable to path lengths. Applications that either map naturally onto a mesh or are not communication intensive can benefit from mesh networks.

However, increased contention in mesh networks may make meshes perform worse than hypercubes. The contention problem scales up with the system size so the performance of large networks is more sensitive to contention. In the next chapter, we show how to model the the effect of contention on the communication performance for a given mapping.

Ch

CO

PR

CO

We st

in thi

manc

conter

satura

tant.

use ra

netwo

can be

Th

includ

for la

define

we use

injecti

randor

mappi

Hence

that m

pattern

by rout

Chapter 3

COMMUNICATION PERFORMANCE UNDER CONTENTION

We study the communication performance of large multicomputers under contention in this chapter. Contention can be a significant factor that determines the performance of large mesh networks. We develop a framework to predict the effect of contention for a given application and mapping. The results help to estimate the saturation node traffic, and to find if the effect of contention is negligible or significant. We also show that we will loose the speedup provided by mesh networks if we use random mapping, and hence random mapping is not advisable in case of large networks. We concentrate on 2D mesh networks in this chapter. However, our results can be extended to networks that have 3D mesh or other topologies.

The chapter is organized as follows. In Section 3.1 we describe several key concepts including the mapping problem, notations and assumptions that lay the foundation for later sections. We model the performance for a given *path of communication* (defined later) in Section 3.2. Several paths originate from a given node. Hence, we use the performance model for a single path to compute bounds on the message injection rates for various nodes in Section 3.3. We investigate the special case of random mapping in Section 3.4. Results show that contention scales up with random mapping, and is significant for large systems (having hundreds or thousands of nodes). Hence, random mapping is not advisable for large multicomputers. Optimal mappings that minimize contention are given in Section 3.5 for several common communication patterns. Contention is reduced significantly by a careful placing of parallel tasks, and by routing various communications. We also describe steps in analyzing the effects of contention and the effect of mapping in Section 3.6. Finally, we summarize the main results of this chapter.

3.1 Modeling Communication and Contention

The effect of contention on communication time depends on the communication characteristics of an application, and the level of contention for network resources, In this section, we model both which will help us later to estimate the degradation in communication performance due to contention.

3.1.1 Communication Model

Parallel programs are usually designed assuming a specific topology such as a tree, a ring, a 3D mesh, a hypercube and so forth. Hence, the communication pattern of a parallel program can be represented by a *process graph.*

Definition 19 Process Graph, G_p, *represents a parallel program. The vertices denote the parallel tasks and the edges denote the communication between tasks.*

Definition 20 System Graph, G_m, *represents the concurrent system. The vertices represent the nodes and the edges represent the communication channels.*

We denote the set of vertices and edges of the process graph (system graph) by V(G_p) (V(G_m)) and E(G_p) (E(G_m)) respectively. In general, G_p and G_m are different. The system graph G_m is chosen based on implementation constraints, whereas G_p may be any topology that suits the problem at hand. Hence, the problem of mapping arises.

Definition 21 Mapping Scheme, μ, *decides the placement of the nodes of* G_p. *It is a function that maps the vertices of* G_p *to that of* G_m, *i.e.* $\mu : V(G_p) \rightarrow V(G_m)$.

If we assume a fixed routing scheme R, then each edge of the process graph (v_1, v_2) corresponds to a *path* between $\mu(v_1)$ and $\mu(v_2)$ defined below.

Definit

graph f

$e_1, e_2,$

and μ

We

the pro

1. F

i

2. A

3. M

4. M

c

5. T

g

C

6. V

c

These

small s

runnin

called

using c

of mes

we con

last ass

One

total n

Definition 22 A Path, $p(e)$, *is the image of an edge* $e = (v_1, v_2)$ *of the process graph for any given mapping* μ. *The path is a sequence of edges of the system graph* $(e_1, e_2, \ldots, e_D)$ *defined by the routing function for communication between nodes* $\mu(v_1)$ *and* $\mu(v_2)$, *where* D *is the path length.*

We restrict the domain of our study by making the following assumptions about the process graphs and mapping.

1. Process graphs, G_p, have an adjacency matrix that is symmetric. Hence, the indegree is equal to the outdegree for any vertex v, and is denoted by $\delta(v)$.

2. A task communicates uniformly with its neighbors in G_p.

3. Message lengths are exponentially distributed.

4. Message generation on each path of G_p is a Poisson process with an arrival rate of λ' messages/sec.

5. The mapping function is one-to-one, i.e. every vertex (node) of the system graph, G_m, is assigned at most one vertex (parallel task) of the process graph, G_p.

6. We assume that blocking communication is used, and that communication and computations do not overlap.

These assumptions imply that a node of G_m communicates uniformly with only a small set of other nodes of G_m onto which the neighbors (in G_p) of the parallel task running on that node are mapped. Two nodes that communicate during a run are called *peers*. Assumptions 3 and 4, are made so we can analyze the performance using queueing theory. Our results, however, apply quite well to other distributions of message lengths and message interarrival times. The last assumption is made so we consider the simpler case first. We can extend our results to the cases when the last assumption may not be true.

One important characteristic of a process graph is its *richness* measured by the total number of edges. Since we assumed the process graph to have a symmetric edge-ma

half the

by consi

in the pr

Here, $\delta($

2E is eq

process g

We see t

of the ric

In ad

of the pa

applicati

- t r

  me

- L i

- τ is

- τ' i

  whi

- T r

  nod

- T' r

  cont

The follow

edge-matrix, we have an even number of edges in a process graph. Let **E** be equal to half the total number of edges. The average and the maximum degree can be defined by considering all nodes of the process graph. If N_p denotes the number of vertices in the process graph, then we have

$$\delta_{avg} = \frac{1}{N_p} \sum_{v \in V(G_p)} \delta(v) \qquad \delta_{max} = \max_{v \in V(G_p)} \delta(v) \tag{3.1}$$

Here, $\delta(v)$ denotes the indegree (equal to outdegree) of a vertex of G_p. Observe that 2**E** is equal to sum of the degrees (outdegree or indegree) of all the nodes of the process graph. Hence,

$$\delta_{avg} = \frac{1}{N_p} \sum_{v \in V(G_p)} \delta(v) = \frac{2\mathbf{E}}{N_p} \tag{3.2}$$

We see that **E** and δ_{avg} are related, and either one of them can be used as a measure of the richness of a process graph.

In addition to the process graph, we need to model the communication intensity of the parallel tasks to completely define the communication characteristics of an application, and the effect of contention. We use the following notation.

- t represents the average time for local computations between two successive message injections by a node.
- L is the average length of a message.
- τ is the communication time for a message of length L.
- τ' is the value of τ if the communication is contention-free. θ is the factor by which communication time has increased due to contention, i.e. $\theta = \tau/\tau'$.
- T represents the average time between two successive message injections by a node, i.e. $T = t + \tau$.
- T' represents the average time between two successive message injections under contention-free communication conditions, i.e. $T' = t + \tau'$.

The following definitions help

Definiti

G_p in bi

channel

Definit

a vertex

a netro

Since a

Applied

from the

other no

task run

applied

Cont

message

Hence,

Definit

any edg

$\lambda_p = \frac{(L}{E}$

Definit

of G_p (c

Simi

$\lambda_1 \leq \lambda_2$

their de

Definition 23 Applied path traffic, λ_{p_a}, *is the ratio of data generated on any edge of* G_p *in bits/sec under contention-free conditions to the capacity of a network network channel,* B*, in bits/sec.*

Definition 24 Applied node traffic, λ_a, *is the ratio of the rate of data generated by a vertex of* G_p *(or* G_m*) in bits/sec under contention-free conditions to the capacity of a network channel,* B*, in bits/sec.*

Since a message of length L is generated by a node in T secs, we find that $\lambda_a = \frac{(L/T')}{B}$. Applied node traffic is the sum of the applied path traffic over all paths that originate from the given node. We assumed that a node of G_p communicates uniformly with δ other nodes; so node traffic $\lambda_a = \delta\lambda_{p_a}$, where δ is the degree (in G_p) of the parallel task running on the given node. Hence, $\lambda_{p_a} = \lambda_a/\delta = \frac{(L/T')}{\delta B}$, Applied path traffic and applied node traffic depend only on the application and the system constant B.

Contention will increase the communication time, and hence the time between messages, T. This contention will reduce the path traffic and also the node traffic. Hence, we have the following definitions for the traffic under contention.

Definition 25 Actual path traffic, λ_p, *is the ratio of the rate of data generated on any edge of* G_p *in bits/sec to the capacity of a network channel,* B*, in bits/sec, i.e.* $\lambda_p = \frac{(L/T)}{B}$.

Definition 26 Node traffic, λ_n, *is the ratio of the rate of data generated by a vertex of* G_p *(or* G_m*) in bits/sec to the capacity of a network channel,* B*, in bits/sec.*

Similar to the path traffic, we find that $\lambda_n = \frac{(L/T)}{B}$, and $\lambda_n = \delta\lambda_p$. Observe that $\lambda_n \leq \lambda_a$ and $\lambda_p \leq \lambda_{p_a}$ as $T \geq T'$. The relation between θ and λ_n can be found by their definitions.

$$\begin{aligned}\lambda_n &= \frac{(L/T)}{B} = \frac{(L/B)}{T} = \frac{(L/B)}{t+\tau} = \frac{T'\lambda_a}{t+\tau'+(\theta-1)\tau'} \\ &= \frac{\lambda_a}{1+(\theta-1)\tau'/T'} = \frac{\lambda_a}{1+(\theta-1)\lambda_a}\end{aligned} \tag{3.3}$$

We can

Since.

of λ_p t

repres

We

netwo

1.

2.

3.

4.

5

ana

3.1

We

to

We can also show a similar relation between λ_p and θ.

$$\begin{aligned} \lambda_p &= \frac{(L/\delta T)}{B} = \frac{(L/B)}{\delta T} = \frac{\delta T' \lambda_{p_a}}{\delta(t+\tau)} \\ &= \frac{\lambda_{p_a}}{1+(\theta-1)\lambda_{p_a}} \text{ by proceeding as in}\lambda_n \text{ derivation above} \qquad (3.4) \end{aligned}$$

Since, θ and λ_p are related, we can represent the effect of contention by the ratio of λ_p to λ_{p_a} or by the ratio of λ_n to λ_a, instead of θ. As we see later, this way of representing the effect of contention has some advantages.

We make the following assumptions about the system graph or the communication network. Most of these are reasonable for current networks, and simplify our analysis.

1. All network channels (edges of G_m) have the same bandwidth.

2. All transmissions are error free.

3. Wormhole-routing or Circuit-switching is used. Header length is assumed to be negligible compared to message length, and the time taken by intermediate routers to process headers is assumed to be negligible compared to the overall message transmission time.

4. A fixed deadlock-free routing scheme is used in which messages travel along the row channels first, and then along the column channels.

5. Physical channels do not support virtual channels.

The last assumption is not true for some new mulitcomputers, such as iWarp. Our analysis needs to be extended to cover those systems.

3.1.2 Contention Parameters

We define the following three parameters for a given G_p, G_m, μ and R which help us to measure the contention for network channels.

Definition 27 Channel load, *χ, for any edge e of G_m, which represents a network channel, is the number of paths that share the given edge, i.e.* $\chi(e) = |\{e' : e' \in E(G_p) \wedge e \in p(e')\}|$.

Channel load helps in characterizing the non-uniformity in traffic through various network channels, and gives an upper bound on path traffic λ_p. If the path traffic along each path sharing a channel is λ_p then $\lambda_p \leq 1/\chi$. The average and maximum channel loads, denoted by χ_{avg} and χ_{max} respectively, are defined considering the various edges of the system graph. We have

$$\chi_{avg} = \frac{1}{|E(G_m)|} \sum_{e \in E(G_m)} \chi(e) \qquad \chi_{max} = \max_{e \in E(G_m)} \chi(e) \tag{3.5}$$

Channel load alone cannot characterize the contention. We define two additional contention parameters—*logical path length* and *path contention level*—for an edge of the process graph. First, we need to define the concept of *path contention sets* which is used in the definitions.

Definition 28 *For any edge e of G_p which corresponds to a path p of G_m, the path contention set s_i is the set of paths that share with p at least one of the first i segments (which correspond to channels) of p.*

Let path $p = e_1, e_2, \ldots, e_D$. Then the corresponding path contention sets s_i, $0 \leq i \leq D$, are defined as

$$s_0 = \phi, \qquad s_i = s_{i-1} \cup \{p' | e_i \in p'\} \tag{3.6}$$

Here, ϕ denotes the null set.

Definition 29 Logical path length *(d) for any edge of G_p is the number of path segments in which at least one new path is "encountered", i.e.* $d = |\{i : 1 \leq i \leq D \wedge s_i \neq s_{i-1}\}|$.

Definition 30 Path contention level, *ν, for any edge of G_p is the total number of paths that share at least one network channel with the given path, and is equal to cardinality of the set s_D, i.e.* $\nu = |s_D|$.

Th
the n
of G_p
longe

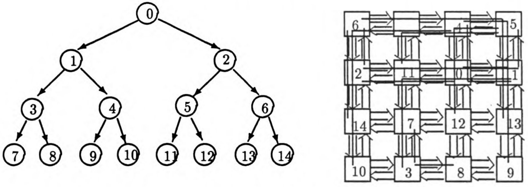
T
proc
level
case
cont

figu
mes
par
to l
of t
lab
cha
the
G_p

The logical path length is bounded by the physical path length D, and denotes the number of queues (for network channels) at which a message on a given edge of G_p may have to wait. High values of logical path length indicate a possibility of longer wait times.

The significance of path contention level is that a message on a given edge of the process graph will be blocking messages on that many paths. Hence, path contention level also puts another upper bound on the path traffic as $\lambda_p \leq 1/(\nu + 1)$. As in case of channel loads, we define average and maximum logical path lengths and path contention levels considering all the edges of the process graph.

An example is given in Figure 3.1 to illustrate the contention parameters. The figure shows a random mapping of a 15-node binary tree (G_p) onto a 16-node, 2D-mesh system graph (G_m). The paths are shown in Figure 3.1.B for the tree edges from parents to their children. We have only shown edges from parents to children in G_p to keep the example simple and the figures clear. In Figure 3.2, we present the values of the contention parameters for this example. In Figure 3.2.A, the tree edges are labeled by the triple (D, d, ν). The mesh edges has are labeled by the corresponding channel loads. The average and maximum values of the contention parameters for the example are also given. Note that for this example d and ν is same for all edges of G_p as G_p is sparsely connected and maximum channel load is 2. In general, $\nu \geq d$.



A. Process Graph (15 node binary tree) B. System Graph (16 node 2D mesh)

Figure 3.1: An example of a random mapping

I

ma

3.

An

mul

serv

trac

also

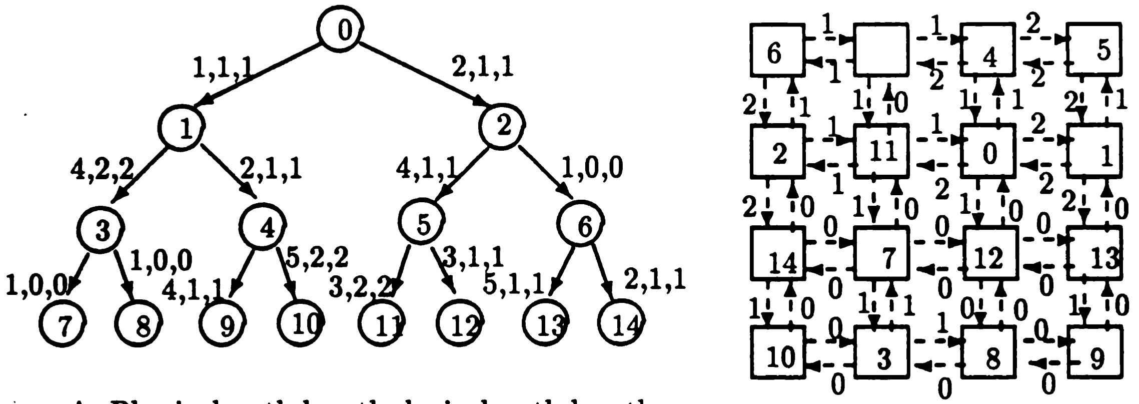
bloc

resu

mod

reaso

(



A. Physical path length, logical path length and path contention level

B. Channel loads

	Avg	Max
Physical path length (D)	2.71	5
Logical path length (d)	1.00	2
Channel load (χ)	0.79	2
Path contention levels (ν)	1.00	2

Average and Maximum values of contention parameters

Figure 3.2: Contention parameters for the mapping example

We later see that computing channel loads and path contention levels for a given mapping can be useful in several ways.

3.2 Saturation Path Traffic

An exact analysis of the contention for network channels requires treatment of a multiple server queuing system with a non-Poisson input process, and non-exponential service times, and hence is very difficult. Here, we examine a simpler, analytically tractable model used earlier to study circuit-switched networks [55]. This model can also be used for wormhole routing as its behavior is similar to circuit switching (in blocking mode) when the message lengths are much longer than path lengths. The results show the effect of contention for a single path. The results based on this simple model not only provide insight into the contention problem, but also can be used to reasonably predict the performance as we show later.

Consider a single path of communication from a source node S to a destination

node

diate

progr

n cha

the b

chan

chan

tenti

(Fig

I

over

each

node

the

stoc

the

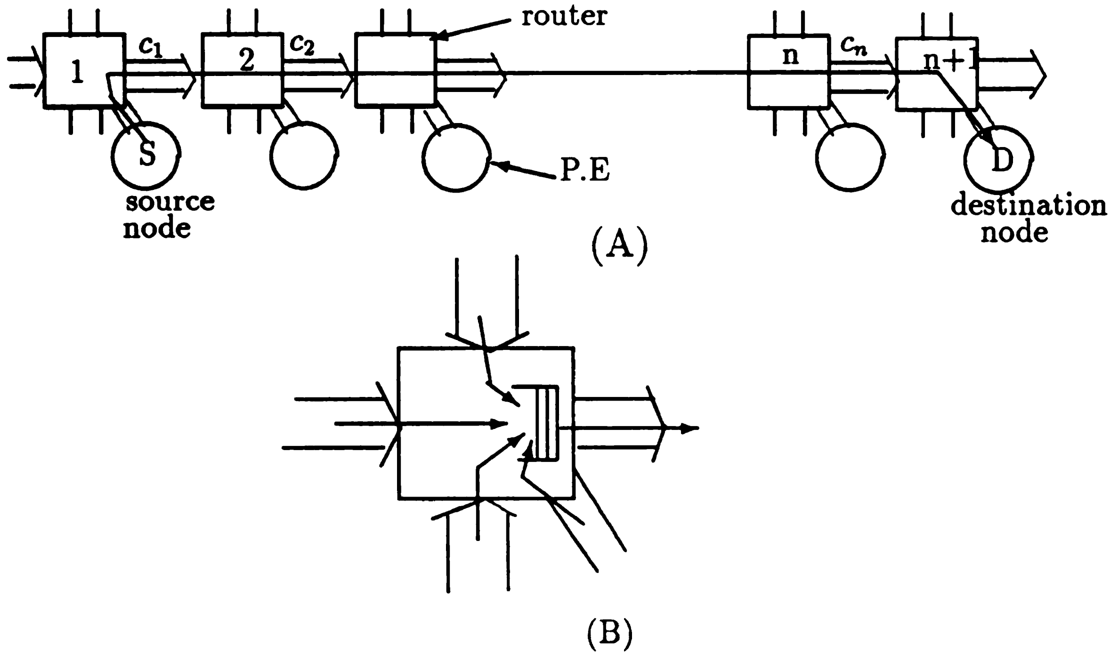


Figure 3.3: A simple contention model

node D (Figure 3.3.A). The path goes over n physical channels and $n-1$ intermediate nodes. In wormhole routing or circuit switching the header of a message will progress from the source node to the destination node reserving the channels. All the n channels must be reserved before a message can be successfully communicated. If the header encounters a busy channel, it waits in a FIFO queue associated with the channel. Assume that at any node there are k other paths over different incoming channels needing the same output channel as the path under consideration. Contention for a channel at an intermediate node can be modeled as an M/M/1 queue (Figure 3.3.B).

Let $\tilde{x}_i$ and $\tilde{s}_i$ be the random variables that represent the service time and the overall delay (sum of wait time and service time) at the i^{th} intermediate node. At each intermediate node, the service time is equal to the overall delay at the next node since processing overheads are negligible (Assumption 3), i.e. $\tilde{x}_i = \tilde{s}_{i+1}$. We use the generalized version of the independence assumption of Kleinrock to remove the stochastic dependence of the service time of one node's queueing system on that of the succeeding node [57]. The assumption is s

v

s

At eac

distrib

does

stage

assum

[37].

conte

queue

as δ_i

k hea

is est

δ_{n+1}

T

heade

W

by wh

with

δ_{n+1}

Assumption: The distribution of service time at each of the queuing systems shown in Figure 3.3.A is a negative exponential with the average values stated in Equation 3.7, and is stochastically independent of the service time of its succeeding nodes.

At each node i, the length of the message is assigned a new value from the exponential distribution with the same average value of L at each node i even though the length does not really change. We can then ignore the dependence of service time of any stage on the service times of successive stages. The results obtained based on this assumption have been found to be reasonable for networks with moderate connectivity [57]. By virtue of our earlier assumptions, and this independence assumption, the contention for a channel at each intermediate node can be modeled as an M/M/1 queue. Also, the mean values of the overall delay, $\bar{s}_i$, and service time, $\bar{x}_i$, are related as $\bar{s}_i = 1/(\mu_i - k\lambda')$, where $\mu_i = 1/\bar{x}_i$, and $k\lambda'$ represents the net arrival rate of the k headers requesting services from the same output channel. Once the entire path is established, i.e. all channels are successfully reserved, communication time will be $\bar{s}_{n+1} = L/B$. The results can be summarized in the following recurrence equation.

$$\bar{s}_i = \frac{1}{1/\bar{s}_{i+1} - k\lambda'} \quad (1 \leq i \leq n) \ ; \bar{s}_{n+1} = \frac{L}{B} \tag{3.7}$$

The recurrence equation can now be solved for the $\bar{s}_i$, the overall delays once the header reaches node i.

$$\bar{s}_i = \frac{(L/B)}{1 - (n-i+1)k\lambda' L/B} = \frac{\bar{s}_{n+1}}{1 - (n-i+1)k\lambda' L/B} \tag{3.8}$$

We are not interested in the absolute value of the overall delay, but the factor by which the overall delay increases due to contention. The delay can be normalized with respect to the communication time under contention-free conditions given by $\bar{s}_{n+1} = L/B$; this normalized time is denoted by θ_i.

$$\theta_i = \frac{\bar{s}_i}{\bar{s}_{n+1}} = \frac{1}{1 - (n-i+1)k\lambda' L/B} \quad ; \tag{3.9}$$

The

Th

λ_i (De

in the

the de

queue

assum

stamp

that

E

satur

expla

with

path

$i\nu +$

T

the

diffe

incre

that

for c

ways

for

infi

(

The normalized delay from the source (θ_1) is denoted by θ.

$$\theta = \theta_1 = \frac{1}{1 \; - \; nk\lambda' L/B} \tag{3.10}$$

The factor ($\lambda' L/B$) in the denominator of Equation 3.10 is the actual path traffic λ_p (Definition 25) since $\lambda' = 1/T$. Hence, we have $\lambda_p = \lambda' L/B$. The remaining factor in the second term of the denominator, which is nk, can be shown to be $\nu + D$ from the definition of ν and D. The system is therefore equivalent to a single M/M/1 queue with $nk = \nu + D$ paths contending for a single resource. We derived this result assuming a FIFO queue policy at each intermediate node. However, if messages are stamped with the time they entered the network and priority is given for messages that have earlier time stamps, then we find that $nk \; = \; (\nu + 1)$. Therefore, we have

$$\theta \; = \; \frac{1}{1 - (\nu + 1)\lambda_p} \; = \; \frac{1}{1 - \lambda_p/\lambda_{sat}} \qquad \textit{where} \quad \lambda_{sat} = \mathit{1}/(\nu + \mathit{1}) \tag{3.11}$$

Equation 3.11 shows that θ increases exponentially as $\lambda_p \rightarrow \lambda_{sat}$. In fact, the saturation path traffic, λ_{sat}, is an upper bound on the value of λ_p. There is an easy explanation for that fact. Observe that the path under consideration is competing with ν other paths, and that it blocks all of them when it is progressing. If all paths have the same path traffic λ_p, i.e. are busy for a λ_p fraction of the time, then $(\nu + 1)\lambda_p \leq 1$ or $\lambda_p \leq 1/(\nu + 1) = \lambda_{sat}$.

The effect of contention given by Equation 3.11 is shown in Figure 3.4. Figure 3.4.A displays the normalized communication time θ as a function of λ_p, for different values of ν. The figure illustrates the factor by which communication time increases due to contention. For example, $\lambda_p \; = \; 0.15$ and $\nu \; = \; 5$ yields a θ of 4; that is, the communication time under contention is 4 times higher than the time for contention-free communication. Figure 3.4.B shows the same result in a different way: the figure shows constant θ curves in the $\lambda_p \times \nu$ space. The curves are shown for $\theta = 2$, 5, 10 and ∞. Everything in the "forbidden region" requires "more" than infinite time, and hence the network cannot operate in this region.

Observe that the region of operation where the contention effect is negligible is restrict

such as

are lin

the eff

increa

will b

W

to exp

quadr

The

old f

Actu

find

and

Figu

W

then

wher

3.3

In la

resul

level

the n

paths

restricted to a small region in which the value of λ_p or ν is small. In current systems such as the Symult 2010, small sizes are limiting the value of ν, and high overheads are limiting the peak value of λ_p. We, therefore, always operate in the region where the effect of contention is negligible. However, in future systems both λ_p and ν will increase due to larger system sizes and lower overheads. Hence, the contention effect will be increasingly visible in future systems.

We can combine Equations 3.11 and 3.4 by eliminating θ from these two equations to express λ_p in terms of λ_{p_a} and λ_{sat}. After some simplification, we get the following quadratic equation for λ_p

$$\lambda_p^2(1 - \lambda_{p_a}) - \lambda_p(\lambda_{p_a} + \lambda_{sat}) + \lambda_{p_a}\lambda_{sat} \;=\; 0 \tag{3.12}$$

The solution for λ_p can be simplified by approximating Equation 3.11 using a threshold function.

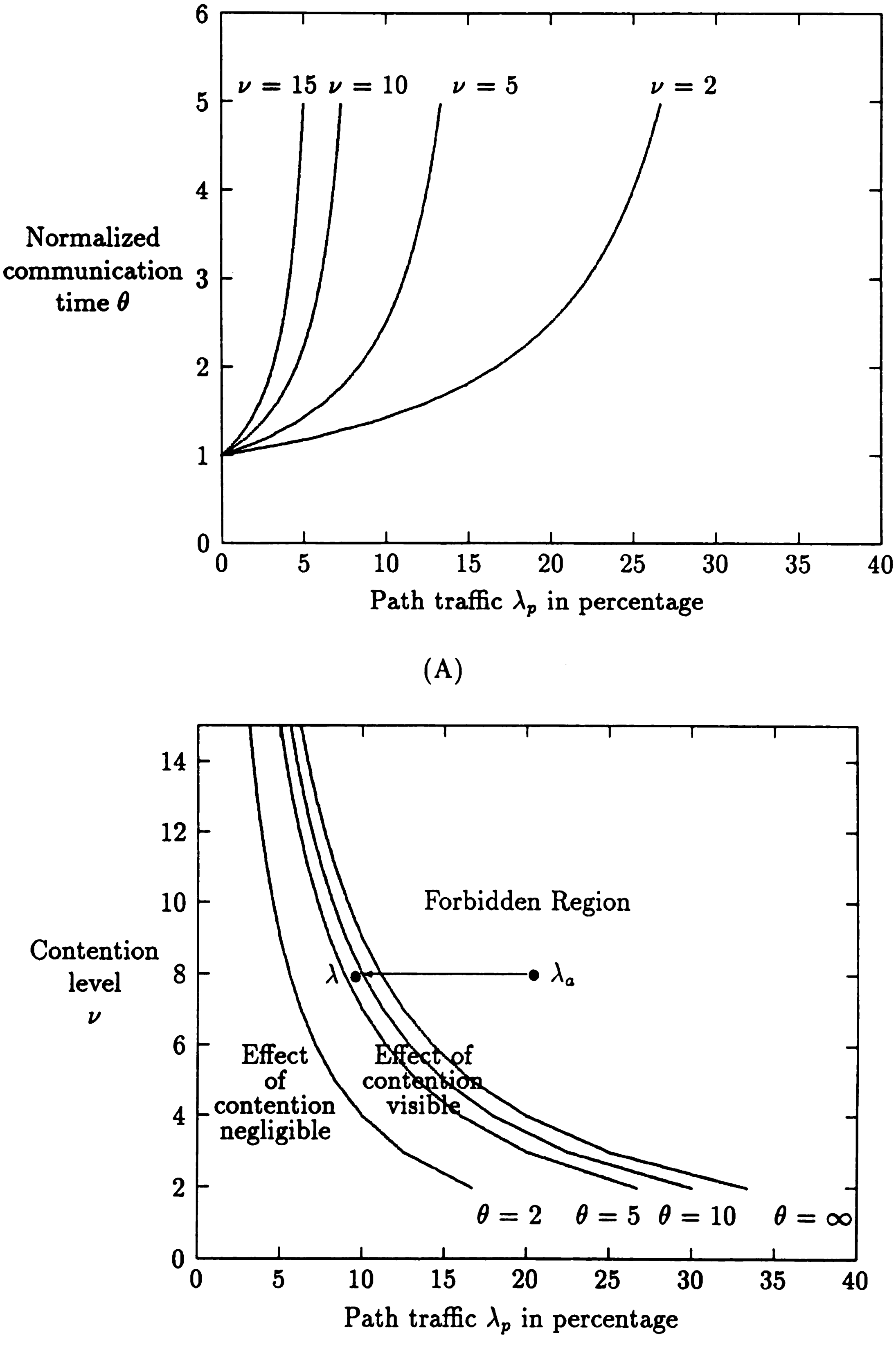
$$\theta = \begin{cases} 1 & \text{if } \lambda_p \leq \lambda_{sat} \\ > 1 & \text{if } \lambda_p = \lambda_{sat} \end{cases} \tag{3.13}$$

Actual path traffic, λ_p, will then be the solution of Equations 3.4 and 3.13. We find that $\lambda_p = \lambda_{p_a}$ if $\lambda_{p_a} < \lambda_{sat}$, and that $\lambda_p = \lambda_{sat}$ if $\lambda_{p_a} \geq \lambda_{sat}$. The exact and approximate solutions for λ_p are shown for a specific instance of $\lambda_{p_a} = 0.5$ in Figure 3.5.A.

We can repeatedly solve for λ_p for different values of λ_{p_a} for a given λ_{sat}. We can then plot λ_p as a function of λ_{p_a}. The resulting curves are shown in Figure 3.5.B when $\lambda_{sat} = 33.33\%$ ($\nu = 2$).

3.3 Saturation Node Traffic

In last section, we analyzed the effect of contention considering a single path. The results show that path traffic saturates at a level determined by the path contention level. The effect of contention on node traffic is similar—to put an upper bound on the node traffic. We can determine the bound on node traffic by considering multiple paths originating from a given node.

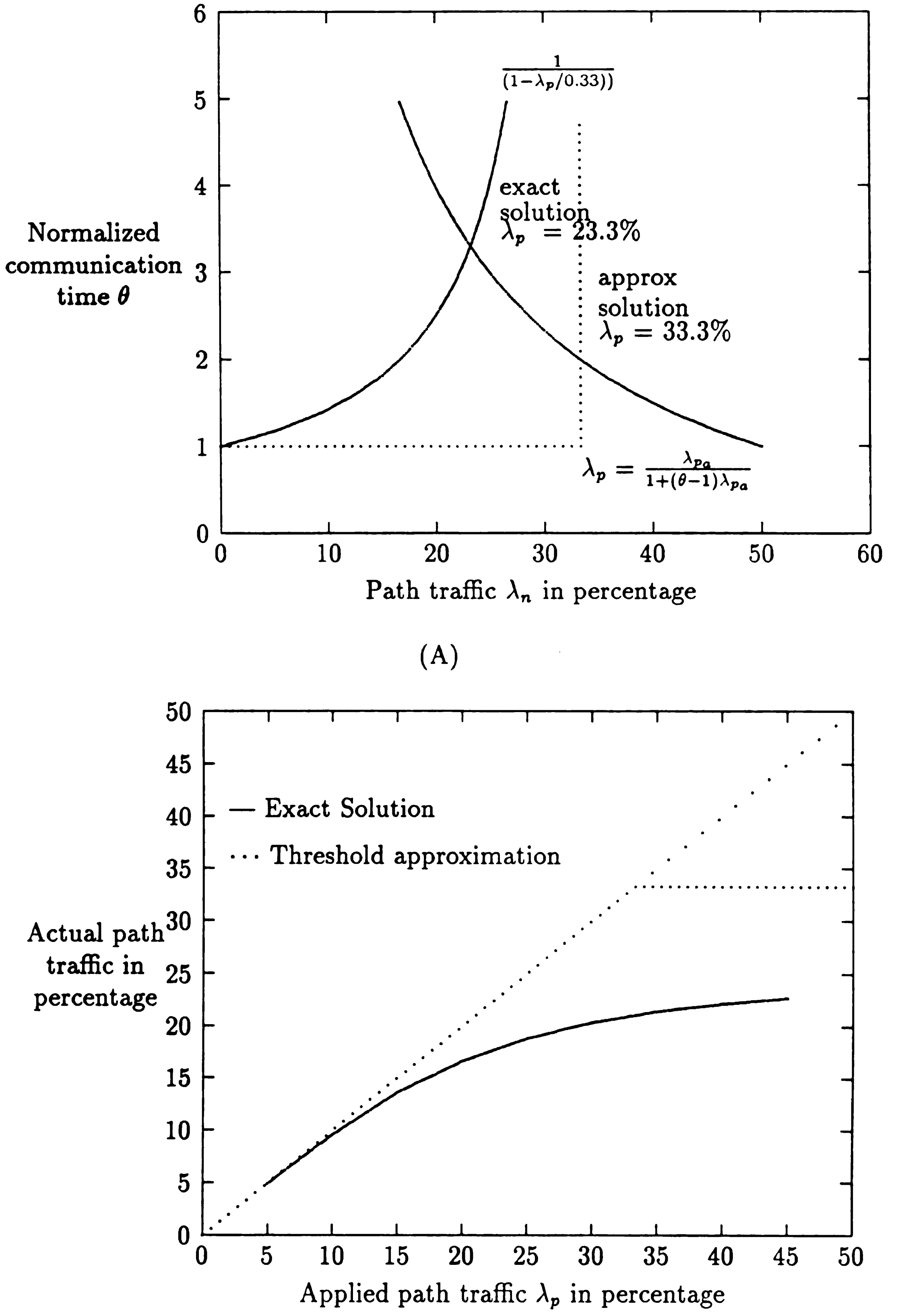


(A)

(B)

Figure 3.4: Effect of contention on communication time for a single path

co



(A)

(B)

Figure 3.5: Actual path traffic vs. Applied path traffic for a single path (ν = 2)

The f

Asserti
that nod

We
parallel
to Sect
of data
equal f
We

Lemm
$\delta_{avg}/($

Pr
these
path
(Equ

whic

W
nod
traf

Le
δ_{avg}

Le

The following assertion holds for the limited problem space we are considering.

Assertion 1 *For any node, the path traffic is same for all the paths originating from that node.*

We assumed that each node is running at most one parallel task, and that a parallel task communicates uniformly with its neighbors in the process graph (refer to Section 3.1). Hence, over a given time a node would have sent an equal amount of data to all paths originating from the node. The result is that path traffic will be equal for any node.

We consider an average node first.

Lemma 3 *The saturation node traffic for an average-case node is given by* $\lambda^{a}_{n_{sat}} = \delta_{avg}/(\nu_{avg} + 1)$.

Proof: The average number of paths that originate in a node is δ_{avg}. Each of these paths has an average path contention level of ν_{avg}. From the results for a single path in Section 3.2, the saturation path traffic for each path is $\lambda_{sat} = 1/(\nu_{avg} + 1)$ (Equation 3.11). Hence, we have

$$\lambda^{a}_{n_{sat}} = \delta_{avg}\lambda_{sat} = \delta_{avg}/(\nu_{avg} + 1)$$

which completes the proof. □

We can also use our framework to estimate the saturation levels for individual nodes and the worst node which will help in characterizing the nonuniformities in traffic We have the following result for the worst case.

Lemma 4 *The saturation node traffic for the worst-case node* $\lambda^{w}_{n_{sat}}$ *is bounded by* $\delta_{avg}/(\nu_{max} + 1)$.

Proof: Consider the path p that has the maximum path contention level (ν_{max}). Let S' be the set of paths that are in the path contention set of p. Let $\lambda_p(x)$ denote

the actu
and on

Since t
node s
Also, δ
same t
$\delta\lambda_p$ =

Th
not be
node
origin
of at

W
paths
defin

whic

W
node
only

the actual path traffic on a path x. Then, since the traffic on the different paths of S and on p are mutually exclusive, we have

$$\sum_{x\in(S'\cup p)} \lambda_p(x) \leq 1 \tag{3.14}$$

Since the worst-case node has the least node traffic, the node traffic of every other node should be greater than or equal to the node traffic of the worst node, $\lambda^w_{n_{sat}}$. Also, δ paths originate from a node, and from Assertion 1 all paths should have the same traffic. Since the sum of each node's path traffic is its node traffic, we have $\delta\lambda_p = \lambda_n \geq \lambda^w_{n_{sat}}$ or $\lambda^w_{n_{sat}}/\delta \leq \lambda_p$.

The value of δ is not constant for a given process graph as the process graph need not be regular. Let $\wp_i$ be the probability that a randomly selected path is from a node of degree i. Since $|S| = (\nu_{max}+1)$, we find that $\wp_i(\nu_{max}+1)$ paths in set S originate from a node whose degree is i, and each of these paths have a path traffic of at least $\lambda^w_{n_{sat}}/i$. Since, the range of i is 0–δ_{max}, we have

$$\sum_{i=0}^{\delta_{max}} (\wp_i(\nu_{max}+1))(\lambda^w_{n_{sat}}/i) \leq 1 \tag{3.15}$$

We can find $\wp_i$ easily. If n_i is the number of nodes whose degree is i, then $n_i i$ paths originate from nodes of degree i. The total number of paths is $2\mathbf{E}$ (from the definition of $\mathbf{E}$). Hence, $\wp_i = n_i i/2\mathbf{E}$. Using this result in Equation 3.15, we have

$$\begin{aligned}\lambda^w_{n_{sat}} &\leq \frac{1}{(\nu_{max}+1)\sum_{i=0}^{\delta_{max}}(\wp_i/i)} = \frac{1}{(\nu_{max}+1)\sum_{i=0}^{\delta_{max}}\frac{n_i i}{2\mathbf{E} i}}\\ &= \frac{2\mathbf{E}}{(\nu_{max}+1)N_p} = \frac{\delta_{avg}}{(\nu_{max}+1)} \quad \text{by Equation 3.2}\end{aligned}$$

which completes the the proof. □

We show in the next chapter by simulation that the above bound is tight, i.e. the node traffic for the worst-case node saturates at this bound if messages are given priority based on the time they entered the network. Hence, we conclude that saturation node t

We

which

Lemm

the m

Pr

given

path t

level

highe

path

path

in th

all p

Usin

for

node traffic for the worst case node is given by $\delta_{avg}/(\nu_{max}+1)$.

We can also compute an upper bound on the node traffic for any individual node which is stated as the following lemma.

Lemma 5 *For any node, the node traffic is bounded by $\delta\left(1-\frac{\nu_m}{\nu_{max}+1}\right)$, where ν_m is the maximum path contention level for any path originating from the node*

Proof: From Assertion 1, we know that all of the paths that originate in a given node should have the same path traffic in the steady state. We know that the path traffic for any path is bounded by $\lambda_{sat} = 1/(\nu+1)$, ν being the path contention level for that path (Equation 3.11). Hence, the bound is lower for paths that have a higher path contention level. Therefore, of all the paths originating from a node, the path p_m that has the maximum path contention level equal to ν_m will determine the path traffic on the remaining paths from the node. Let S be the set of paths that are in the path contention set for p_m. The sum of path traffic on p_m, and path traffic on all paths in S should be less than 1. Therefore, we have

$$\lambda_p(p_m) \;\leq\; 1-\sum_{p\in S}\lambda_p(p) \tag{3.16}$$

Using the same argument as in the proof of Lemma 4 we find that

$$\begin{aligned}\lambda_p(p_m) \;&\leq\; 1-\sum_{i=0}^{\delta_{max}}\wp_i\nu_m(\lambda^w_{n_{sat}}/i) = 1-\nu_m\lambda^w_{n_{sat}}\sum_{i=0}^{\delta_{max}}(\wp_i/i)\\ &=\; 1-\frac{\nu_m\lambda^w_{n_{sat}}}{\delta_{avg}} \leq 1-\frac{\nu_m}{(\nu_{max}+1)} \quad \text{by Lemma 4}\end{aligned}$$

Since, $\lambda_n = \delta\lambda_p$, we prove the lemma. □

Lemmas 3, 4 and 5 show that it is possible to estimate the bound on node traffic for an average node, the worst-case node or any node.

3.4 Ra

We consider
term suggest
We first show
average and
show that ou
puted values
Combining t
section, we
sufficient for
to the avera
in traffic an

Here, we
possible me
A node in r

3.4.1 P

Path length
graph is ma
imum path
corner nod
by assumin
stochastica
expected le

$$D_{m}$$

$$D_{z}$$

3.4 Random Mapping

We consider the special case of random mapping now. In random mapping, as the term suggests, tasks are mapped onto a randomly-selected system node that is free. We first show that we can use graph theory and probability theory to estimate the average and maximum values of contention parameters for a random mapping. We show that our estimates are sufficiently accurate by comparing them with the computed values from 25 different random mappings for several known process graphs. Combining the estimates for path contention levels with the results of the previous section, we can find the saturation node traffic, and decide if random mapping is sufficient for the given application. We can estimate the worst-case values in addition to the average-case values. Hence, the results also show the extent of nonuniformity in traffic and contention.

Here, we assume that process graphs are randomly mapped onto the smallest possible mesh network. Such a system graph will have N nodes where $N = \lceil\sqrt{N_p}\rceil^2$. A node in row i and column j is denoted by (i,j), $0 \leq i,j < \sqrt{N}$.

3.4.1 Path Lengths

Path length depends only on the size of the system graph onto which the process graph is mapped, and is independent of the richness of the process graph. The maximum path length (p_{max}) occurs for communication between the diagonally-opposite corner nodes of the system graph. The average path length (p_{avg}) can be computed by assuming that the lengths of the segments of a path in the two dimensions are stochastically independent [33, 65]. The average path length will then be twice the expected length in each dimension. We, therefore, have

$$D_{max} = 2(\sqrt{N}-1) \tag{3.17}$$

$$D_{avg} = 2\left(\frac{1}{N}\sum_{i=0}^{\sqrt{N}-1}\sum_{j=0}^{\sqrt{N}-1}|i-j|\right) = \frac{2\sqrt{N}}{3} - \frac{2}{3\sqrt{N}} \approx \frac{2\sqrt{N}}{3} \tag{3.18}$$

We o

time

T

path

abil

the l

riche

Figu

3.4

The

leng

equa

sam

T

node

chan

a col

used

If des

given

the n

We observe that $D_{max} \approx 3D_{avg}$, i.e. the maximum path length is approximately three times higher than the average path length.

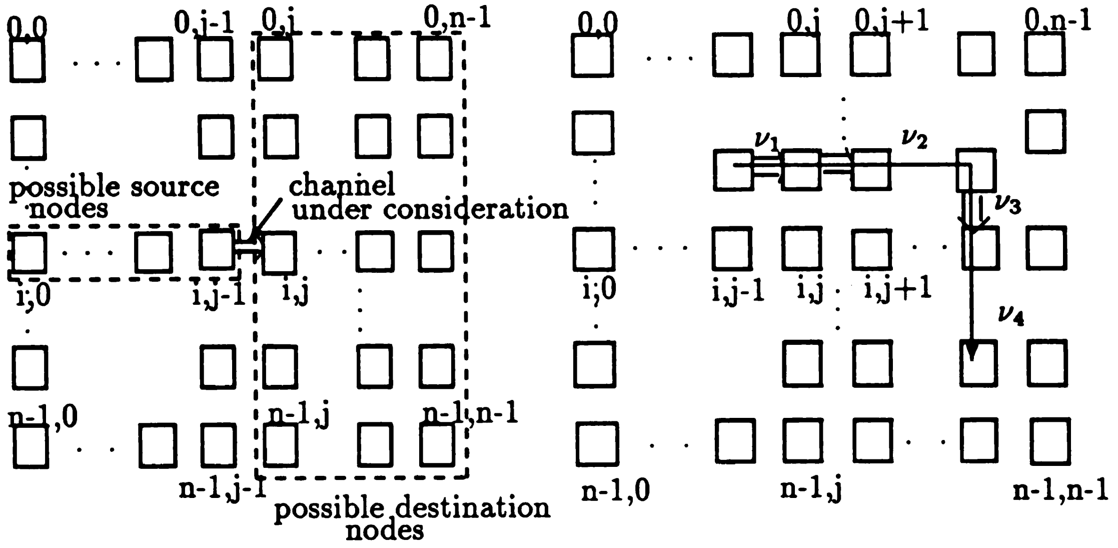
The logical path length is approximately equal to and is bounded by the physical path length. It can be seen that if the process graph is richer, there is higher probability that a new path will be "encountered" at each segment of the path. Hence, the logical path length become closer to the physical path length as a graph becomes richer. The estimates and actual values from 25 random mappings are shown in Figure 3.8.A and B. We observe that estimates match well with actual values.

3.4.2 Channel Loads

The average channel load is easy to calculate as it is related to the average path length. The sum of the channel loads over all edges of the system graph should be equal to the sum of path lengths over all edges of process graph, as both count the same quantity in two different ways. Hence, we have

$$\begin{aligned}
\chi_{avg} &= \frac{1}{|E(G_m)|}\sum_{e\in E(G_m)} \chi(e) = \frac{1}{4\sqrt{N}(\sqrt{N}\ -\ 1)}\sum_{e\in E(G_p)} D(e) \\
&= \frac{|E(G_p)|D_{avg}}{4\sqrt{N}(\sqrt{N}\ -\ 1)} \\
&\approx \frac{2\mathbf{E}(2\sqrt{N}/3)}{\sqrt{N}(4\sqrt{N}-1)} \quad usingEquation\ 3.18 \\
&\approx \frac{\mathbf{E}}{3\sqrt{N}} \qquad (3.19)
\end{aligned}$$

To find the maximum channel load, consider the load on a row channel between nodes $(i, j-1)$ and (i, j) (Figure 3.6.A). The source nodes of paths that need that channel are the nodes $(i,0)$ .. $(i, j-1)$, and the destination of these paths should be in a column greater than or equal to j (This is a consequence of the fixed routing scheme used by the network). A total of $j\delta_{avg}$ paths originate from nodes $(i,0)\ldots(i,j-1)$. If destinations are uniformly distributed, the probability, $\wp$, that the destination of a given path is in a column greater than or equal to j is $(\sqrt{N}\ -\ j/\sqrt{N})$. Therefore, the number of paths through, or channel load, on the channel under consideration,



A. Estimating Channel load B. Estimating path contention level

Figure 3.6: Estimating contention parameters

ij, is a fraction $\wp$ of the $j\delta_{avg}$ paths.

$$\chi_{ij} = \wp(j\delta_{avg}) = \frac{j(\sqrt{N}-j)\delta_{avg}}{\sqrt{N}}$$

The above expression will be maximum when $j = \sqrt{N}/2$ (which makes $\wp = 1/2$). Hence,

$$\chi_{max} = \frac{\sqrt{N}\delta_{avg}}{4} = \frac{\sqrt{N}}{4} \times \frac{2\ \mathrm{E}}{N_p} = \frac{\mathrm{E}}{2\sqrt{N}} \tag{3.20}$$

The maximum channel load derived so far is the expected value for a single row. There are $\sqrt{N}$ rows. When we consider all the rows, it is possible that for the worst-case row, all destinations are on the right half. Hence, we conclude that the maximum channel load is given by

$$\chi_{max} = \frac{\mathrm{E}}{\sqrt{N}} \tag{3.21}$$

Here, only row channels were considered. The distribution of loads on column channels can be estimated similarly, i.e. considering the possible source and destination nodes for the paths that need a given channel. We find that the distribution of loads on column channels is similar to that of row channels. Hence, we conclude that maximum channel load is given by the Equation 3.21. Note that loads increase as one moves t

The

shown

estima

estima

values

instead

times

3.4.3

The p

length

partit

of ν_1 .

on var

path l

Th

(row)

We sh

Th

remain

and de

moves towards the center of the mesh network.

The estimated and the actual values as computed from 25 random mappings are shown in Figure 3.9 for four common process graphs. The results show that our estimates for channel loads are good. The actual values are slightly less than the estimate, except for the binary tree, for maximum channel loads. We find that actual values are better approximated by

$$\chi_{max} = 0.75\mathbf{E}/\sqrt{N} \tag{3.22}$$

instead of $\mathbf{E}/\sqrt{N}$. The maximum channel load is therefore about $0.75 \times 3 \approx 2.25$ times higher than the average channel load.

3.4.3 Path Contention Levels

The path contention level is more difficult to estimate. A path on average has a length of $\sqrt{N}/3$ in each dimension as explained in Section 3.4.1. The path can be partitioned into four segments ν_i, $1 \leq i \leq 4$ (Figure 3.7). We can estimate the values of ν_1 .. ν_4 by considering the possible source and destinations of paths "encountered" on various segments of the path, and making use of the results for channel loads and path lengths. We can then add them to get ν, i.e.

$$\nu \; = \; \nu_1 \; + \; \nu_2 \; + \; \nu_3 \; + \; \nu_4 \tag{3.23}$$

The first term, ν_1, is the number of paths that use the first, higher dimension (row) channel. The average value of ν_1 will be one less than the average channel load. We showed in Section 3.4.2 that the average channel load is $\mathbf{E}/3\sqrt{N}$ so

$$\nu_1 \; = \; \chi_{avg} - 1 \; \approx \; \frac{\mathbf{E}}{3\sqrt{N}} \; - \; 1 \; \approx \; \frac{\mathbf{E}}{3\sqrt{\mathbf{N}}} \tag{3.24}$$

The next term, ν_2, in Equation 3.23 is the number of new paths encountered on the remaining channels of the higher dimension. To estimate, consider the possible sources and destinations for a new path encountered at the channel connecting node (i, j) to

(i + 1

node

$\delta_{i,j}$ p

of the

Hence

Th

the fir

chann

Th

remain

the nu

source

interme

that ro

should i

in a row

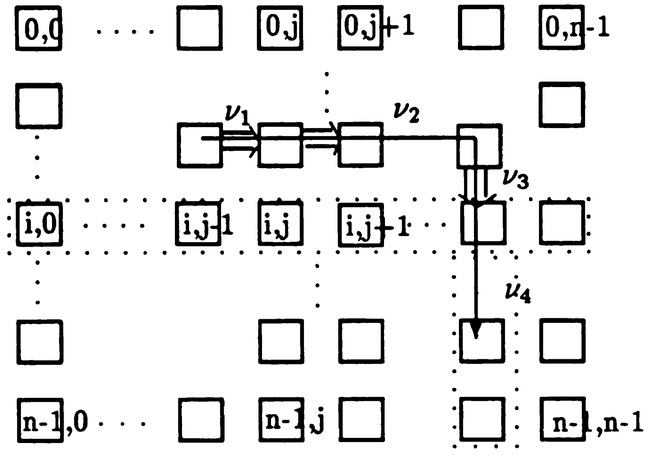


Figure 3.7: Estimating path contention level.

$(i+1, j)$. According to the routing protocol, the source must be the intermediate node (i, j), and the destination be a node in a column higher than j. By definition, δ_{avg} paths originate at each of the $\sqrt{N}/3$ intermediate nodes, and approximately half of them can be expected to go in the same direction as the path under consideration. Hence,

$$\nu_2 \approx \frac{\delta_{avg}}{2} \times \frac{\sqrt{N}}{3} = \frac{2\text{E}}{2N_p} \times \frac{\sqrt{N}}{3} \approx \frac{\text{E}}{3\sqrt{N}} \tag{3.25}$$

The third term, ν_3, in Equation 3.23 is the number of new paths encountered on the first channel of the lower dimension, which is approximately equal to the average channel load.

$$\nu_3 = \nu_1 \approx \frac{\text{E}}{3\sqrt{N}} \tag{3.26}$$

The last term, ν_4, in Equation 3.23 is the number of paths encountered on the remaining channels of the lower dimension. We can compute this value by considering the number of new paths encountered at each intermediate node (Figure 3.7). The source node of these new paths should be one of the nodes in the same row as the intermediate node under consideration. The total number of paths emanating from that row to all destinations is $\delta_{avg} \times \sqrt{N}$. The destination of the paths we encounter should be one of the nodes in the same column as the node under consideration, and in a row numbered higher than the intermediate node. The probability $\wp$ will be

$\sqrt{N/2}/N = 1/2\sqrt{N}$ on an average. Hence,

$$\nu_4 = \frac{\sqrt{N}}{3} \times \left(\wp \times \delta_{avg}\sqrt{N}\right) = \frac{\sqrt{N}}{3} \times \frac{1}{2\sqrt{N}} \times \frac{2\mathrm{E}}{N_p}\sqrt{N} = \frac{\mathrm{E}}{3\sqrt{N}} \tag{3.27}$$

Combining Equations 3.24-3.27 with Equation 3.23, we have

$$\begin{aligned}\nu_{avg} &= \nu_1 + \nu_2 + \nu_3 + \nu_4 \\ &= \frac{\mathrm{E}}{3\sqrt{N}} + \frac{\mathrm{E}}{3\sqrt{N}} + \frac{\mathrm{E}}{3\sqrt{N}} + \frac{\mathrm{E}}{3\sqrt{N}} = \frac{4\mathrm{E}}{3\sqrt{N}}\end{aligned} \tag{3.28}$$

From Equations 3.28 and 3.19 we find that $\nu_{avg}/\chi_{avg} = 4$, i.e. the average path contention level is about four times the average channel load.

The maximum path contention level is more difficult to estimate. The problem is that the values of the four components are dependent on one another. For no path can all the components reach their maximum values. There are two cases to consider. In the first case, the source node of the path is at the center of the mesh. In that case, the first channel of the path in each dimension can be shown to have the maximum load equal to $\chi_{max} = \mathrm{E}/2\sqrt{N}$. The maximum length of the path segments can only be $\sqrt{N}/2$. Hence, ν_2 and ν_4 will not have maximum possible values. Using $\sqrt{N}/2$ instead of $\sqrt{N}/3$ in Equations 3.25,3.27, we can show that ν_2 and ν_4 to be $\mathrm{E}/2\sqrt{N}$. The maximum path contention level will be

$$\nu_{max} = \frac{\mathrm{E}}{2\sqrt{N}} + \frac{\mathrm{E}}{2\sqrt{N}} + \frac{\mathrm{E}}{2\sqrt{N}} + \frac{\mathrm{E}}{2\sqrt{N}} = \frac{2\mathrm{E}}{\sqrt{N}} \tag{3.29}$$

Another possible case resulting in high path contention level is when the length of the path segments is maximum i.e. $\sqrt{N} - 1$ instead of $\sqrt{N}/3$. In that case, we find that ν_2 and ν_4 will be maximum ($\approx \mathrm{E}/\sqrt{N}$), by using the values ($\sqrt{N} - 1$) instead of $\sqrt{N}/3$ in Equations 3.25,3.27. However, ν_1 and ν_3 will be minimum equal to 0. Once again we find that $\nu_{max} \approx 2\mathrm{E}/\sqrt{N}$, and we conclude that Equation 3.29 is the estimate for maximum path contention level.

Actual values of average path contention level are slightly lower than our estimates. We find that $\nu_{avg} = \mathrm{E}/\sqrt{N}$ fits the actual values better than our estimate of

$4E/3\sqrt{}$

are no

four co

our es

$2E\sqrt{}$

Fr

mappi

-

- t

-

W

with t

The re

Lemm

rando

Pr

mappi

results

λ^2

$4\mathrm{E}/3\sqrt{N}$. The difference arises due to the fact that the values of the four components are not independent of each other, and some error is introduced when we estimate the four components independently and then add them to get ν_{avg}. We, therefore, modify our estimates, and summarize the final results in Table 3.1 where $C = \delta_{avg}\sqrt{N} = 2\mathrm{E}\sqrt{N}/N_p \approx 2\mathrm{E}/\sqrt{N}$.

Contention Parameter	Average	Maximum
Channel load (χ)	C/6	3C/8
Path contention level (ν)	C/2	C

Table 3.1: Contention parameters : summary of estimates

From the results so far, we conclude the following about the special case of random mapping.

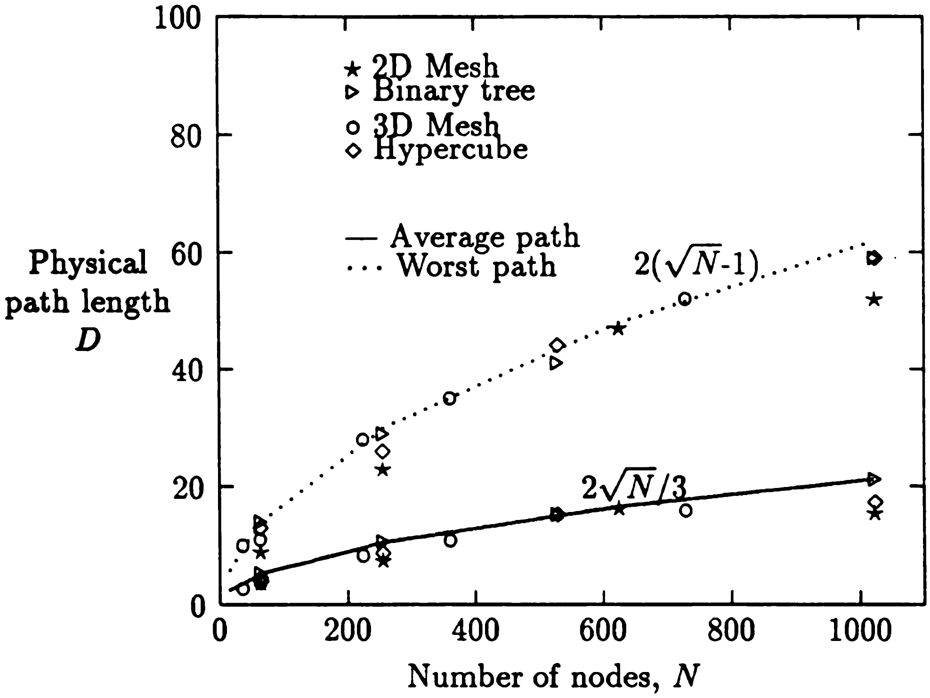
- contention increases with both system size and the degree of the process graph.
- the worst case values are 2 to 3 times higher than the average case.
- the ratio of path contention level to the channel load remains approximately equal to 3 as the system size or richness of process graph changes.

We can combine the estimates of the contention parameters for random mapping with the result of the previous chapter for saturation node traffic (Lemmas 3 and 7). The results are the following lemmas.

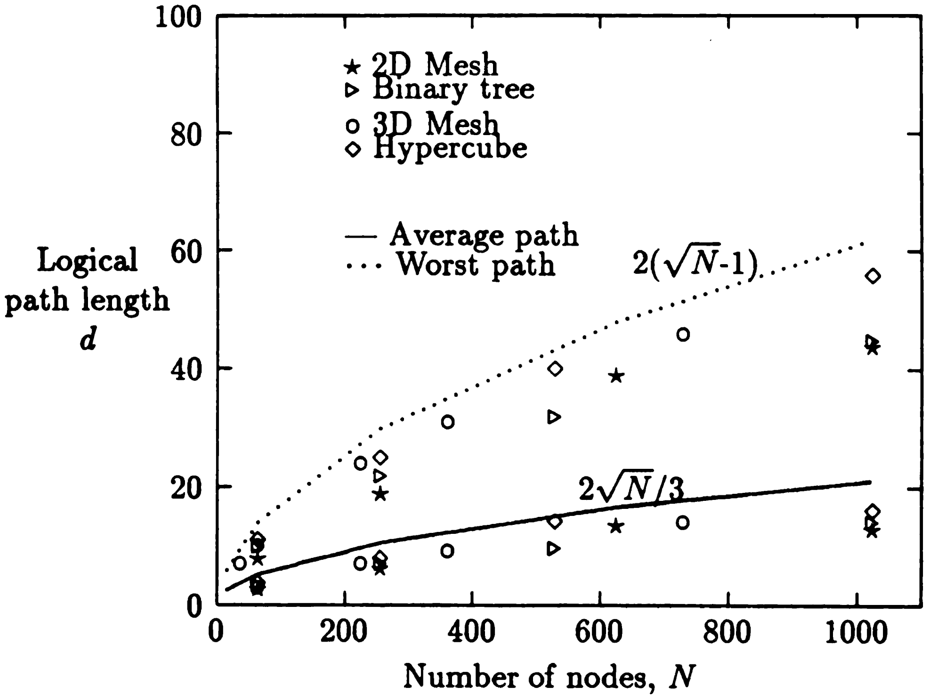
Lemma 6 *The saturation node traffic for an average node when an application is randomly mapped onto a mesh of N nodes is* $\lambda^a_{n_{sat}} = 2/\sqrt{N}$.

Proof: We have shown that the average path contention level for random mapping is $\mathrm{E}/\sqrt{N}$ (Table 3.1), and $\delta_{avg} = 2\mathrm{E}/N_p$ (Equation 3.2). Using these results in Lemma 3, we have

$$\lambda^a_{n_{sat}} = \frac{\delta_{avg}}{\nu_{avg}+1} = \frac{2\mathrm{E}/N_p}{(\mathrm{E}/\sqrt{N})+1} \approx \frac{2\sqrt{N}}{N_p} \approx \frac{2}{\sqrt{N}} \quad \text{as} \quad N_p \approx N \qquad (3.30)$$

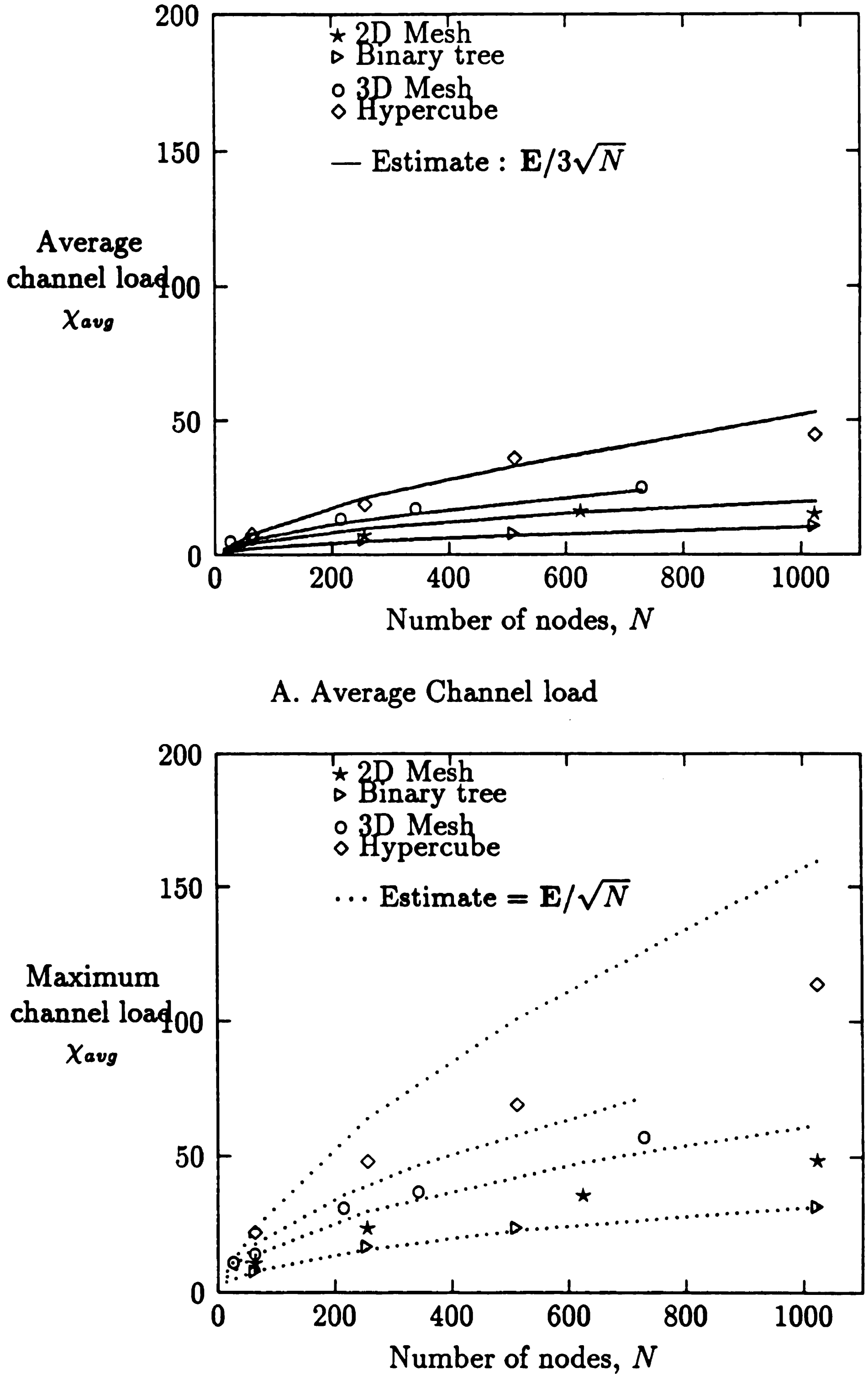


A. Physical path lengths.



B. Logical path lengths
(lines show the estimates, legends show the actual values)

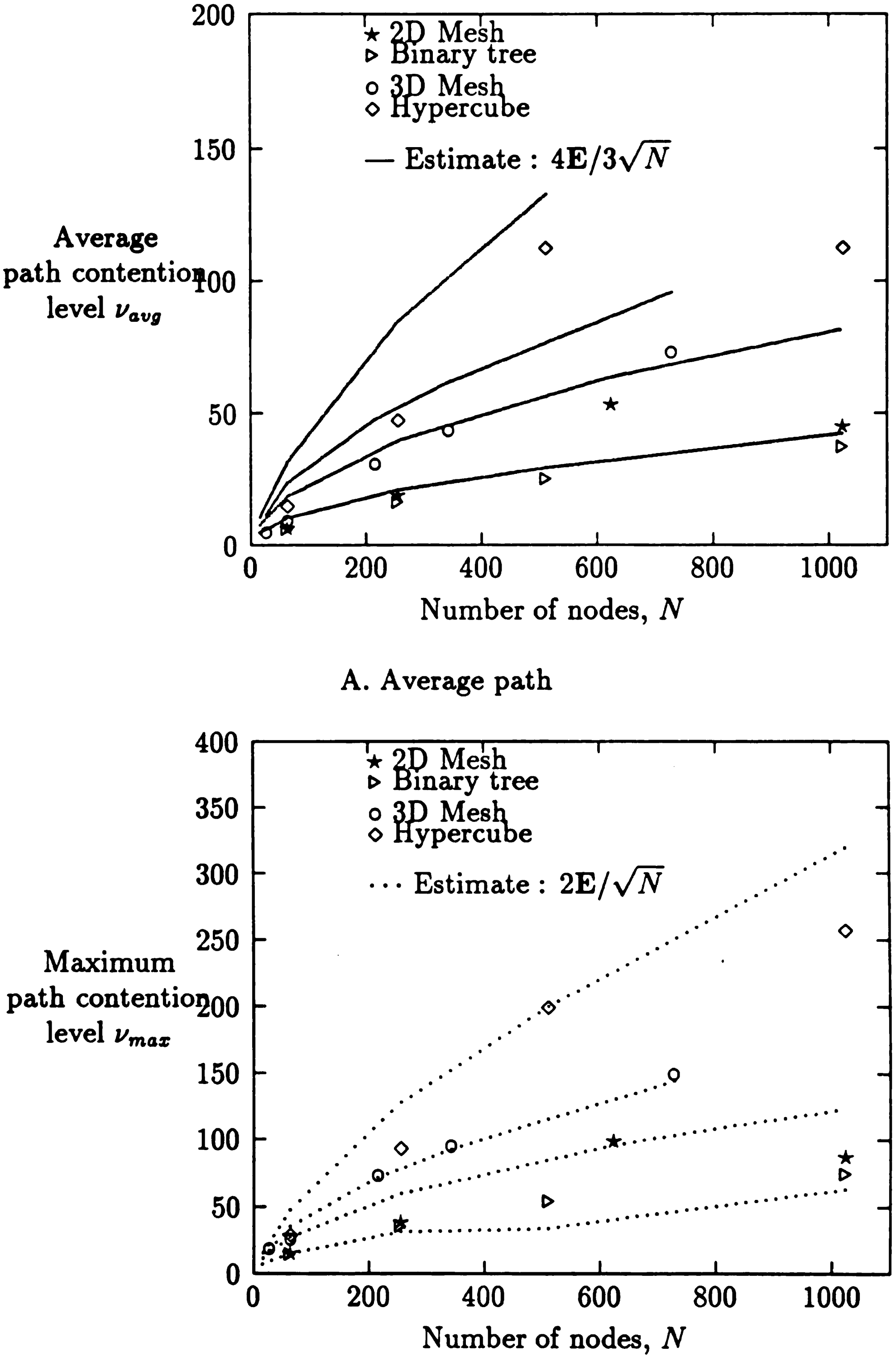
Figure 3.8: Path lengths, estimated and actual values



A. Average Channel load

B. Maximum channel load
(lines show the estimates, legends show the actual values)

Figure 3.9: Channel loads, estimated and actual values



A. Average path

B. Worst path
(lines show the estimates, legends show the actual values)

Figure 3.10: Path contention level, estimated and actual values

which

Lem

rand

P

map

resul

whic

L

tions

level

Lem

the

map

comp

in ac

Table

mapp

In

accura

which completes the proof. □

Lemma 7 *The saturation node traffic for the worst node when an application is randomly mapped onto a mesh of N nodes is bounded by* $\lambda^{w}_{n_{sat}} = 1/\sqrt{N}$.

Proof: We have shown that the maximum path contention level for random mapping is $2\mathrm{E}/\sqrt{N}$ (Equation 3.29), and $\delta_{avg} = 2\mathrm{E}/N_p$ (Equation 3.2). Using these results in Lemma 4, we have

$$\lambda^{w}_{n_{sat}} = \frac{\delta_{avg}}{\nu_{max}+1} = \frac{2\mathrm{E}/N_p}{\frac{2\mathrm{E}}{\sqrt{N}}+1} \approx \frac{\sqrt{N}}{N} \approx \frac{1}{\sqrt{N}} \quad \text{as} \quad N_p \approx N \tag{3.31}$$

which completes the proof. □

Lemmas 6 and 7 also give a rough estimate of the saturation levels when applications are randomly mapped. However, we can actually compute the path contention levels even for the case of random mapping, and use these values with the results of Lemmas 3, 4 and 5 to estimate the saturation levels more precisely. Table 3.2 gives the estimated saturation levels for four different topologies when they are randomly mapped (a specific instance). In fact, the software we have developed automatically computes the saturation node traffic values for any given process graph and mapping, in addition to contention parameters.

	2D Mesh		Binary tree		3D mesh		Hypercube	
	Avg	Wc	Avg	Wc	Avg	Wc	Avg	Wc
$N = 64$	47	23	31	14	36	19	41	22
$N = 256$	19	10	12	6	13	7	17	9
$N = 1024$	8	4	5	3	5	3	7	4

Table 3.2: Saturation node traffic for various process graphs for a specific random mapping

In the next chapter, we show by simulations that average estimates are quite accurate. The worst case (Wc) estimates also match, but simulation results seem to give about

Observ

node (Ler

expression

to any pr

is comple

which res

corollary.

Corollar

node tra

In th

node is t

be 2/16

recently

network

Definiti

to the c

total ba

and is d

We h

each ha

of node

For unif

be $2\sqrt{N}$

give about 20% higher values consistently for all process graphs.

Observe that the saturation node traffic for an average node as well as the worst node (Lemma 6 and 7) is independent of the the richness of the process graph as the expression for $\lambda_{n_{sat}}$ does not contain the term δ_{avg} or E. In general, the result applies to any process graph including a completely connected graph. If the process graph is completely connected, then a node communicates with all other nodes uniformly which results in uniformly distributed messages. Therefore, we have the following corollary.

Corollary 5 *For uniformly distributed messages in an N-node mesh, the saturation node traffic is $2/\sqrt{N}$.*

In the case of uniformly distributed messages, all nodes are identical and the worst node is the same as the average node. For a 256-node mesh, the saturation level will be $2/16 = 12.5\%$. The saturation traffic level for uniformly distributed messages has recently been derived in a different way by Dally [33], in which a different unit, called network traffic, was used to measure traffic.

Definition 31 Network traffic *per node is the ratio of the data injected by the node to the capacity of the network, and is denoted by λ_t.* Network capacity *per node is the total bandwidth out of a node divided by the average distance traveled by a message, and is denoted by C.*

We have $C = 4B/D_{avg}$ for a mesh network, as there are four outgoing channels, each having a bandwidth of B. The data injected by a node is $\lambda_n B$ by the definition of node traffic, our metric for traffic (Definition 26). Hence, we have

$$\lambda_t = \frac{\lambda_n B}{C} = \frac{\lambda_n B}{4B/D_{avg}} = \frac{D_{avg}\lambda_n}{4} \tag{3.32}$$

For uniformly distributed messages in a mesh network, we know the value of D_{avg} to be $2\sqrt{N}/3$ (Equation 3.18). Hence, we have

$$\lambda_t = \frac{2\sqrt{N}}{3} \times \frac{\lambda_n}{4} = \frac{\sqrt{N}}{6}\lambda_n \tag{3.33}$$

The e
relati
(Cor

whic
[33].

T
pred

3.5

In th
by p
grap
imp

3.5

Sev
and
N
tile
16-
rou
Obs
com
tree
corr
leve

The expression above shows that node traffic is related to network traffic. The above relationship holds even for saturation traffic values. Hence, using the value of $\lambda^a_{n_{sat}}$ (Corollary 5) we have

$$\lambda^a_{t_{sat}} = \frac{\sqrt{N}}{6}\lambda^a_{n_{sat}} = \frac{\sqrt{N}}{6} \times \frac{2}{\sqrt{N}} = \frac{1}{3} = 33.33\% \qquad (3.34)$$

which match the reported result for the uniformly distributed messages by Dally in [33].

The advantage of our framework is that it is more flexible, and can be used to predict the saturation node traffic levels for various process graphs and mappings.

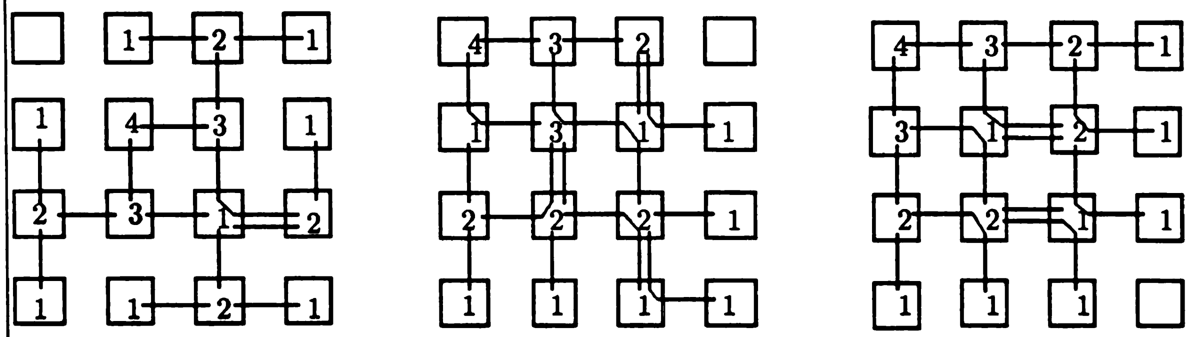
3.5 Optimal Mapping

In this section, we present mappings and routings that minimize contention (measured by path contention level and channel load) in the case of several common process graphs. Simulations later show that these mappings provide a significant performance improvement over random mapping.

3.5.1 Binary Tree

Several schemes have been suggested recently to optimally layout a binary tree. Youn and Singh [79] have suggested a tile-based scheme which uses just one extra node i.e. $N = N_p + 1$. We adapted their scheme to reduce χ_{max} to 2. Three types of basic tiles that give the mapping and routing for a 15-node complete binary tree onto a 16-node mesh are shown in Figure 3.11. The edges shown in the figure indicate the routing for communication in either direction i.e. child to parent or parent to child. Observe that for these tiles χ_{max} is 2. The scheme can be extended for larger trees by combining the basic tiles. The path corresponding to higher level edges of the larger tree uses only the channels along the perimeter of the basic tiles. Also, no two paths corresponding to higher level edges share a channel. Hence, the maximum channel level still remains 2. The scheme also reduces path contention levels significantly.

Contention is reduced even if we only use the placement suggested, and use the standard fixed routing instead of the suggested routing. In the next chapter, we show that even if we only use the placement suggested by Youn's scheme and the standard fixed routing (i.e. ignore the routing suggested by this scheme) we still get significant improvement in performance compared to random mapping. Another scheme suggested by Gordon [45] uses slightly more nodes for a given tree, but reduces χ_{max} to 1 and ν_{max} to 0, completely eliminating contention.



Tile M1 Tile M2 Tile M3

Figure 3.11: Three basic tiles in modified Youn's scheme

(The labels indicate the levels of the tree node
1 indicates the leaf level, and 4 indicates the root level)

3.5.2 Mesh-3D

The 3D mesh is another popular communication pattern used by many applications. Figure 3.12 suggests two schemes. Scheme A does not use any extra nodes, but results in high channel loads of $\chi_{max} = \sqrt[3]{N_p} + 1$. Scheme B may need more nodes, but reduces the value of χ_{max} to $\sqrt[3/2]{N_p} + 1$.

3.5.3 Hypercube

We explained and proved in Section 2.2 that an identity layout of hypercube topology gives the minimum peak width equal to $\lfloor 2N/3 \rfloor$. We can therefore use an identity m

o

w

i

r

c

p

u

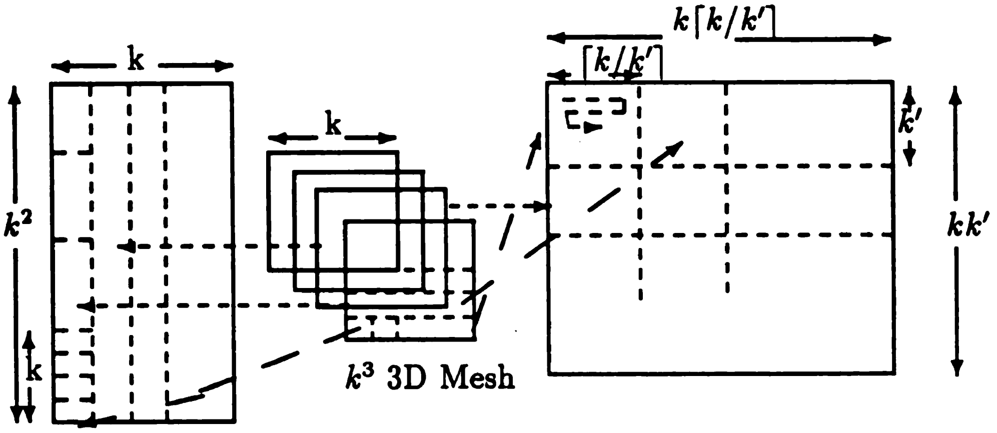
v

3

(

t

e



Scheme 1

Scheme 2

Figure 3.12: Mapping 3D Mesh on to 2D mesh network

($N_p = k^3$, $k' = \lfloor\sqrt{k}\rfloor$)

mapping to minimize contention where node (i, j) of mesh runs the $i\sqrt{N} + j$ task of the hypercube process graph. If the dimension of the hypercube is even, then we will be mapping $N_p = 2^n$ nodes of the process graph onto a $2^{n/2} \times 2^{n/2}$ mesh (then $N_p = N$ and $\sqrt{N} = 2^{n/2}$). Each row or column then will have $\sqrt{N}$ nodes. The maximum channel load for this optimal mapping will be $\lfloor 2\sqrt{N}/3 \rfloor$, from the result of Theorem 2.

Alternate schemes of mapping and routing that are possible in the case of several process graphs indicate that there is a tradeoff between the number of extra nodes used (i.e. $N - N_p$) and the extent to which the contention can be minimized (χ values or ν values).

3.6 Speedup

Contention can reduce the performance of 2D or 3D mesh networks more than a richer topology such as the hypercube. The following lemma relates λ_{sat} to a reduction in effective rates.

Lemma 8 *The effective rate under contention can be reduced by a factor of λ_{sat}, i.e. $E = \lambda_{sat} B$.*

Proof: By Definition 26 $\lambda_n = \frac{(L/B)}{T}$. or $T = \frac{L/B}{\lambda_n}$. Also, the communication time is the loop time minus the computation time, i.e. $\tau = T - t$. For communication intensive applications, $\lambda_n \to \lambda_{sat}$. Also in such applications, the term t can be ignored in comparison to τ. Hence, $T \approx \tau$. By the definition of effective rate (Definition 10), we have

$$E = \frac{L}{\tau} = \frac{L}{T} = \frac{L}{\frac{L/B}{\lambda_{sat}}} = \lambda_{sat} B \tag{3.35}$$

□

In the case of a richer topology, the effect of contention will be negligible and $\lambda_{sat} \approx 1$. Hence, the effective rate will be the same as channel bandwidth. We find that the speedup also decreases by a factor of λ_{sat} for communication intensive applications if we use random mapping.

$$\begin{aligned} R_e &= \frac{E^m}{E^h} \quad \text{(Definition 12)} \\ &= \frac{\lambda_{sat} B^m}{B^h} \quad \text{(Lemma 8)} \\ &= \lambda_{sat} R_b \quad \text{(Definition 11)} \\ &= \frac{2}{\sqrt{N}} \times \frac{2\sqrt{N}}{3} \quad \text{(Table 2.1)} \\ &\approx 1.33 \end{aligned}$$

This result shows that all the speedup we obtained by using a mesh would be canceled by increased contention if we use random mapping. Hence, mesh would perform just slightly better than hypercube for communication intensive application. In reality, the bandwidth of mesh networks may not be exactly $2\sqrt{N}/3$ times that of hypercube networks. If the bandwidth is less than this factor, the mesh will perform worse than hypercubes.

For applications that have applied node traffic comparable to $2/\sqrt{N}$ or greater, we can use careful mappings. Such careful mappings will reduce the path contention level, and

then the

that to re

effective

careful in

3.7

The theo

form of a

the satur

At prese

2D Toru

rewriting

other use

The s

for an ap

1. Fi

be done

- Pr

ma

the

mu

ces

con

of

ter

con

it w

- Ru

level, and increase λ_{sat}. If mapping and routing are sufficient to make $\lambda_{sat} > \lambda_a$, then the effect of contention will be negligible, and $R_e \approx R_b$. Hence, we conclude that to retain the advantage of the higher bandwidth of mesh networks and to get an effective communication speed equal to the channel bandwidth, we may have to be careful in mapping parallel tasks.

3.7 Contention Analyzer: A Software Tool

The theoretical framework developed in this section has been implemented in the form of a contention analyzer. It does all the computations involved, and hence gives the saturation levels for node traffics directly, for a given process graph and mapping. At present, the tool works for Linear array, 2D mesh, 3D mesh, Hypercube, and 2D Torus. The tool can be easily modified for other system graphs, as it involves rewriting one or two small functions. Random mapping is built into the tool. Any other user-specific mapping has to be defined by the user in the form of a function.

The software tool is useful in analyzing contention, and evaluating the mapping for an application. The following steps are suggested.

1. First estimate the applied node traffic for the application. This estimation can be done in several ways.

- Program analysis : If the program involves a single kernel loop, we can estimate the data generated per loop on average, say L bytes. We can also estimate the average number of instructions, say I, executed per loop. Then, from the multicomputer specification sheet we can use the MIPS rating of the node processor P to find the computation time t. Recall that communication time under contention-free conditions can be specified by a linear function $a+bL$. The value of a and b are either known from the machine specification or can be easily determined. Hence, communication time under contention-free condition can be computed if we know L. The Applied node traffic, λ_a, can also be computed as it will be $\frac{(bL)}{(I/P+a+bL)}$.

- Run the application on a small system say 9 or 16 nodes, carefully mapping the para

the

The

2. Co

using the

is not go

becomes

good eno

If $\lambda^{a}_{n,}$

affected.

nodes are

determin

nodes is r

enough.

3.8

The com

in this c

traffic, a

bound ca

that the

ping is no

neutraliz

imize pat

node traf

parallel tasks if possible. In such a case contention will be negligible. Measure the total amount of data generated by a node L', and the total run time T'. Then, $\lambda_a = bL'/T$.

2. Compute the worst-case saturation node traffic under random mapping $\lambda^{a}_{n_{sat}}$ using the tool. If λ_a is comparable to or greater than $\lambda^{w}_{n_{sat}}$, then a random mapping is not good enough. Try an improved mapping, and recompute $\lambda^{w}_{n_{sat}}$. Once $\lambda^{w}_{n_{sat}}$ becomes at least a few percent greater than λ_a, then the mapping can be considered good enough.

If $\lambda^{a}_{n_{sat}}$ is less than or equal to λ_{sat}, the performance of most of the nodes is affected. However, if $\lambda^{a}_{n_{sat}}$ is greater than λ_a but $\lambda^{w}_{n_{sat}}$ is lower than λ_a, only a few nodes are affected by contention. The bound on individual node traffic can be used to determine the nodes that will be affected by contention. If the performance of those nodes is not critical to the overall performance, the mapping may be considered good enough.

3.8 Conclusion

The communication performance under contention for network channels was modeled in this chapter. The results show that message injection rates, measured by node traffic, are bounded for a given communication pattern and mapping, and that this bound can be predicted. The special case of random mapping was analyzed to show that the performance can be poor for large multicomputers, and hence random mapping is not advisable for them. Increased contention in mesh networks will effectively neutralize their advantage—higher bandwidth channels. Careful mappings that minimize path contention levels can increase the saturation levels well above the applied node traffic of an application, and hence make the effect of contention negligible.

Chapter 4

EXPERIMENTS ON SYMULT-2010

In this chapter, we describe several experiments we conducted on a Symult 2010, a second-generation multicomputer that uses a mesh-connected wormhole-routed network for interprocessor communication. Although the Symult computer is no longer in production, there are many lessons to be learned from this machine. For a long time, the largest machine available had only 64 nodes, but it was recently increased to 192 nodes.

The experimental studies in this chapter are significant for several reasons.

- The experiments were run on a real system that uses a state-of-the-art, wormhole-routed network, so the results indicate the performance that can be expected from current technology at the user level.

- The experiments reveal the problems and bottlenecks in the current architecture, and indicate architectural areas to be examined in future designs for improved performance.

- The experimental results for communication under contention are analyzed, and compared with our theoretical results. This comparison is done by expressing the performance in terms of node traffic defined in the previous chapter.

- We discuss the current architectural trends and how they will affect our conclusions.

In particular, our results show that path lengths are no longer a problem whereas the effect of contention for network channels will become an important concern, even though the effect is negligible in small systems currently available.

The chapter is organized as follows. In Section 4.1, we briefly describe the Symult 2010's architecture and show how wormhole routing is implemented. In Section 4.2, we describe the experiments and present the results. The results give the network performance both under contention-free condition and under contention at user level. The results under contention are analyzed in detail in Section 4.3. We generalize the performance curves by expressing them in terms of the applied node traffic (defined in Chapter 2). In Section 4.4, we discuss the weakness and idiosyncrasies of Symult 2010, and current architectural trends in order to extend the results to the performance of future multicomputer. The discussion clearly shows that contention for network channels will become a serious concern in future multicomputers. Finally, we summarize our results.

4.1 Symult 2010 Architecture

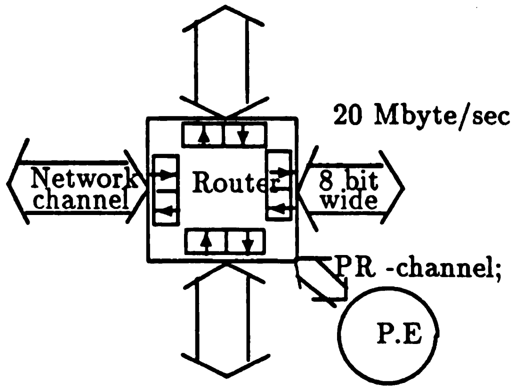


Figure 4.1: The architecture of a node of Symult 2010.

We briefly describe the architecture and salient features of a Symult 2010, as that will help in understanding our experiments and experimental results. The overall architecture of a node is shown in Figure 4.1. The two main components of a node are thè router and the processing element (or a PE). The PE includes a processor, local memory and an interface to the router. The router supports interprocessor communication through five bidirectional physical channels, one connected to each of its four neighbors in the 2D mesh topology and one connected to the local PE. We refer to a router-to-router channel as a *network channel* or *rr-channel* and a PE-to-router channel as a *pr-channel.* Each physical channel is 8 bits wide and operates at a peak rate of 20 MHz, giving a peak bandwidth of 20 Mbytes/sec. Traffic in opposite directions is supported by time-multiplexing the physical channel at a *flit* level, a flit being the data unit for communication between adjacent nodes. Hence, a physical channel can be thought of as supporting two logical, unidirectional channels in opposite directions. For each such unidirectional channel, the router has one buffer whose size is big enough to hold one flit.

The Symult 2010 uses wormhole routing which was explained in Section 1.3.1. A message consists of a set of data flits preceded by two head flits—one for each dimension. The head flit indicates the number of hops needed along each dimension to reach the destination node. As soon as a router receives a message, it examines the head flit to determine whether the message is for the local PE. If it is, the head flit and the message flits following it will be forwarded to the local PE through the interface. Otherwise, the router will determine the next channel, and forward the flit if that channel is available.

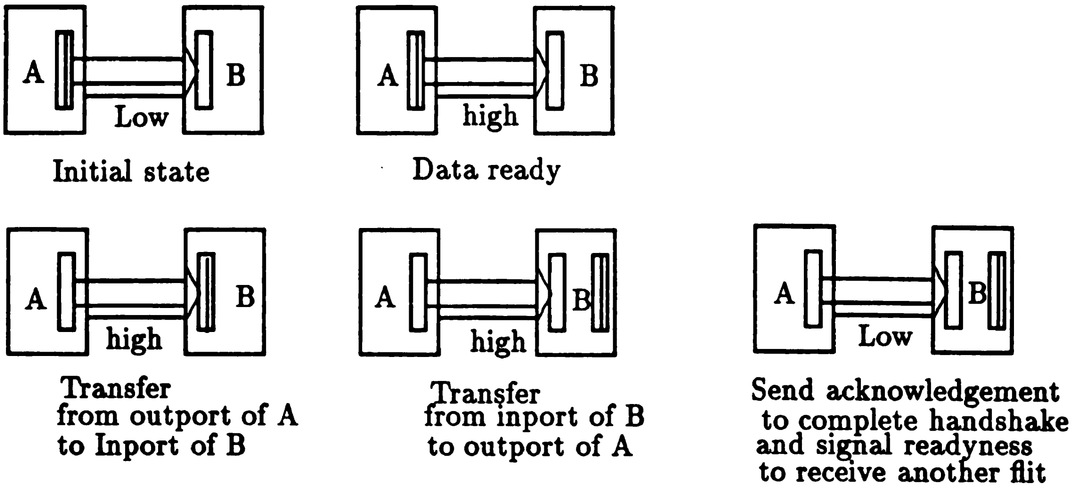


Figure 4.2: Transfer of a single flit over one hop

A single control line is sufficient for each logical channel. The line will be used both for handshaking and for flow-control. Figure 4.2 shows the basic steps involved in a single hop of a data flit from one node to its adjacent node. The control line is normally low, indicating the readiness of the receiving router. Whenever data is to be transmitted, the sending router will raise the line indicating the availability of a new data flit. The receiving router will then read the data flit on the channel into its input buffer. The router will then examine if the next output port to which data flit is to be routed is free. If it is free, the router will transfer the flit to the output port, and lower the control line to acknowledge to the sender the successful transfer of a data flit, and also to indicate its readiness to receive another data flit. Once the sending router detects the lowering of the control line, it will mark the output buffer free to be used for another transfer. A router cannot transfer the received flit to the next output port, if that port has been reserved by another message. In that case, the router will not lower the line, and hence will not complete the message transfer. Similarly, the sending node will not be available to use the output port for another transfer. Successive flits then get blocked along the partially established path, one flit per node.

Each router requires only a limited number of storage buffers, one per channel, to hold the flits of the messages passing through. All routing functions are implemented in hardware so the PE is not involved in processing messages that are merely passing through. The time taken to process a newly arrived flit and to send it to the appropriate adjacent node is less than 100 ns. In order to avoid communication deadlock, the Symult 2010 uses a fixed (static) routing scheme. In such a fixed routing scheme, flits first travel along the rows until they reach the column of the destination node and then travel along the columns to reach the destination node.

4.2 Experiments

Experiments were run on Symult 2010 systems at Caltech under the Cosmic Environment / Reactive Kernel (CE/RK) – a message-passing programming environment available on several multicomputers [75]. All test programs were written in the 'C' language and used the run time library supported by the CE/RK environment for interprocessor communication. We ran the experiments using 16 nodes (connected as 4×4 2D mesh), 64 nodes (8×8) or 144 nodes (12×12). We start this section with a list of general comments, notation and precautions that were necessary to ensure the validity of the measurements.

In our experiments, the following steps were taken to isolate the desired network performance from other system characteristics.

1. Once the programs were spawned onto the nodes, the host waited for 5 secs for spawning to complete before sending (broadcasting) the input parameters of the experiment. This delay ensured that intensive traffic due to program spawning will end before the start of the experiments. If we had not waited, this extraneous traffic would have interfered with the messages generated by the experiments and affected the measured times.

2. In each experiment our aim was to find the average time taken by a node to complete a given program segment. Such a segment, called the *base loop*, consists of both local computation and communication with other nodes. The loops of the experiments require 0.5–15 ms, but the clock resolution is only 1ms so the base loop was repeated N times. Traffic due to the broadcast of the experimental parameters interfered with the communication in the initial loops. Hence, the first n loops were ignored. The clock reading at the beginning of the $n + 1^{th}$ loop was subtracted from the clock reading at the end of the n'^{th} loop, yielding the time for $n' - n$ loops, say $T_{n'-n}$ ms. The average loop time, T, was then computed as $T = T_{n'-n}/(n'-n)$. In our experiments we found that $n' = 11{,}000$ and $n = 1000$ were sufficient. We measured the loop time on each node in milliseconds by averaging the time for 10,000 loops.

3. The objective of the experiments was to measure the performance of the communication network and not the Cosmic Environment functions that support messages. Hence, messages were created before the beginning of the first loop, and the same set of messages was used throughout the experiment. Messages were neither created nor destroyed once the first loop started. This strategy prevented any processing time by functions, such as `xalloc` or `xfree` (which manage message buffers),

from affecting the measured times.

4. Blocking receives were used in all experiments to capture the delays in the communication. The design of the base loop ensured that receiving nodes were always ready to receive by the time a message arrived at the node. This technique prevented any program-generated delays in accepting a message from appearing as communication delays.

5. The kernels were different on different machines, and were modified during the course of the experiments. Changes due to kernel performance were avoided as follows. Many experiments were performed in pairs one after the other; the first tested the inclusion of a specific factor under study, while the other tested the exclusion of the factor. We then calculated the difference in communication times. Timings measured on different days were never compared.

6. Each node computed the average loop time as described above, by using its own clock for each reading. This local timing avoided any problems due to the variance in different clock readings. In addition, each node executed the same program. In some experiments, the mean of the loop times determined by individual nodes was used as the final loop time to avoid idiosyncrasies of individual node hardware (or kernel).

We use the following notation. The system consists of N nodes connected as a $\sqrt{N} \times \sqrt{N}$ 2D mesh. Nodes are numbered 0..N-1, and rows and columns are numbered 0..$\sqrt{N}$-1. Node (i, j) refers to a node located in the i^{th} row and the j^{th} column.

Definition 32 *Two nodes which communicate with each other during an experimental run are called peer nodes.*

In most experiments a node (i, j) will communicate with only one other node, and hence has a unique peer node. A communication pattern is specified by defining the peer nodes. The positions of the peers were changed to form different communication patterns, causing changes in the demand on the network.

In all the experiments, the base loop consisted of communication with peer nodes followed by local computation. Communication is represented by two functions *tsend* and *trecv*.

- *tsend (dst,msg,L)* : Send a message of length L bytes, which is stored starting at the address *msg*, to the node *dst*. The function is non-blocking.
- *msg = trecv (src)* : Receive a message from the node *src*. The message will be put at the address *msg* when the call returns. This function is a blocking receive so that the call will not return until the message arrives.

In all experiments the length of the message is same in each base loop. The computation times of different loops are uniformly distributed over the interval $0..2t_{avg}$. We used such a distribution to reflect the fact in actual applications the computation times may vary depending on the contents of the received messages. Hence, t_{avg} is the average computation time per loop, and is the same for any node. On the other hand, the communication time per loop is different for different nodes as messages from different nodes are affected differently by contention and path length. Communication time per loop is represented by τ_i for the i^{th} node.

Since communication time differs for different nodes, the loop times differ for different nodes, and are denoted by $T_i = t_{avg} + \tau_i$ for the i^{th} node. The loop times for an average and a worst node, denoted by T_{avg} and T_{max}, are defined as

$$T_{avg} = \frac{1}{N}\sum_{0}^{N-1} T_i \qquad \text{and} \qquad T_{max} = \max_{0}^{N-1} T_i \tag{4.1}$$

The communication time per loop for an average and a worst case node can be computed by subtracting t_{avg} from the corresponding loop times.

In each experiment, nodes are partitioned into three classes: I, A and B. Nodes belonging to the the class I are idle, and do not participate in the experiment. The remaining nodes belong to either class A or B, and execute the corresponding base loop repeatedly. A node and its peer usually belong to different classes.

The experiments are explained in three sections. In Section 4.2.1, we measure the performance for contention-free communication between adjacent nodes, which is the best an application programmer can achieve. The times in this section are neither affected by longer path lengths, nor by contention for channels and data paths. In Section 4.2.2 we concentrate on the effect of path lengths. Non-adjacent nodes communicate, but contention is still carefully avoided. In Section 4.2.3, several experiments are described which involve communication under contention.

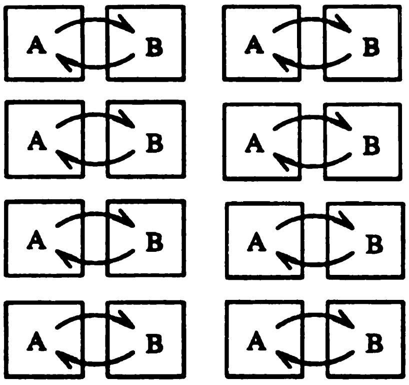
4.2.1 Contention-free Adjacent-node Communication

In the first set of experiments, communication is restricted to physically adjacent nodes. The communication pattern is defined by (Figure 4.3).

peer of node(i,j) = node$(i, j \oplus 1)$ ($\oplus$ denotes the bitwise ex-or function
)

class of node(i,j) = even(j) $\Rightarrow$ A , odd(j) $\Rightarrow$ B.

None of the nodes are idle, i.e. none belong to the class I. The N active nodes of the system are partitioned into $N/2$ pairs, and these pairs do not interfere with each others communication. In the base loop, a message of L bytes is sent from an A node to its peer, and then a message of the same size is sent back from the peer (Figure 4.3).



communication pattern

```
Class A node                 Class B node

tsend(peer,msg.l) ;     - ► msg = trecv(peer) ;
msg = trecv(peer) ;     ◄ - tsend(peer,msg.l) ;
compute(t) ;                 compute(t) ;
```

Base loop

Figure 4.3: Contention-free adjacent-node communication

Contention is carefully avoided. First notice that at any time a node will be either sending or receiving. Thus there is traffic along only one direction on the pr-hannels. The communication between the various pairs does not interfere with one another. Also, on each network channel, the traffic is only along one direction at any instant.

In addition, both the peers compute for the same amount of time after completing their communication. This strategy ensures that the receiving node will always be ready for the incoming message.

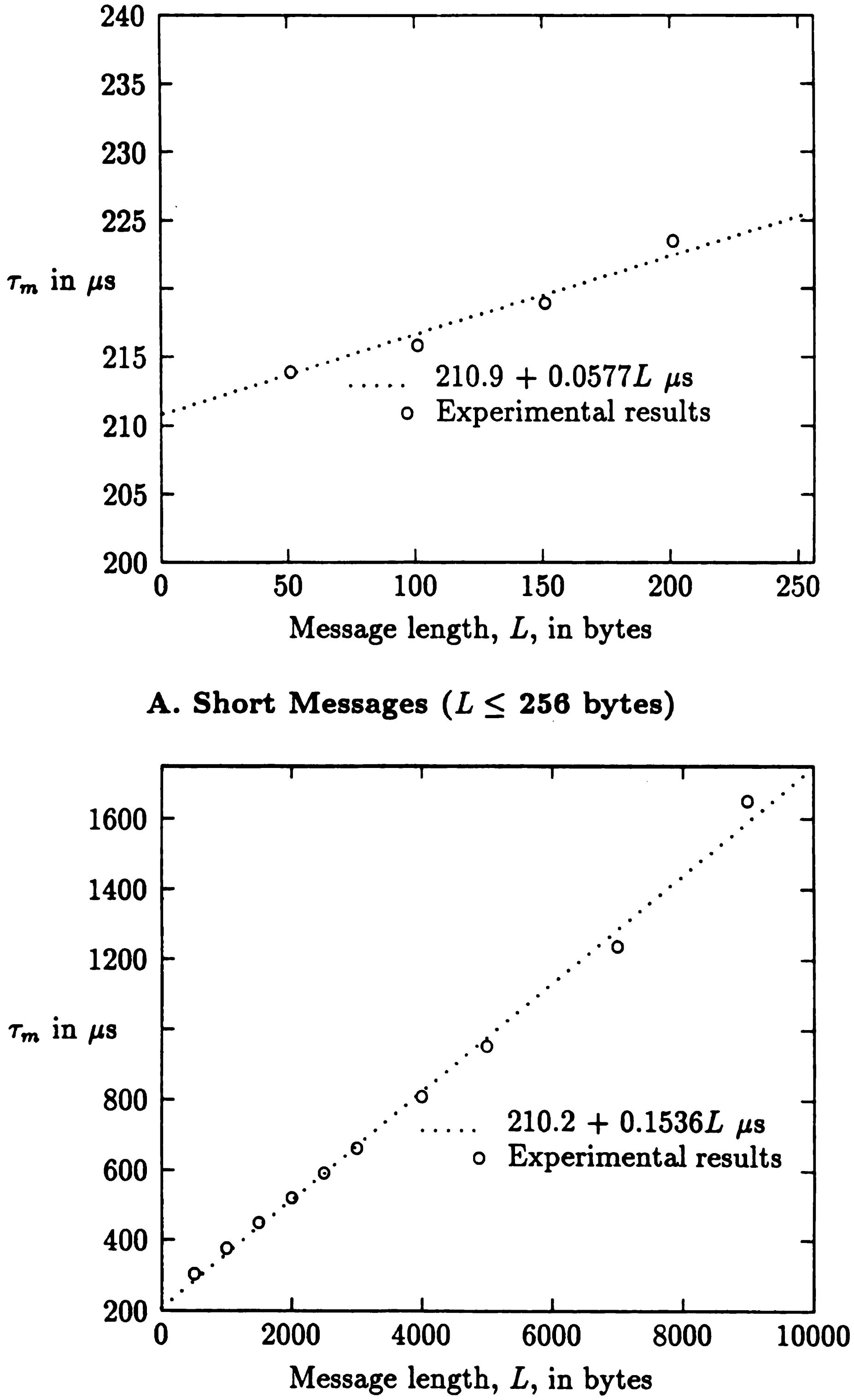
The communication time should be approximately the same for all the nodes as there is no contention and all path lengths are unity. Each loop involves communication of two messages sequentially. Hence, the communication time per message, τ_m, will be half the communication time per loop. We have

$$\tau_m = \tau/2 = (T - t_{avg})/2 \tag{4.2}$$

The experiment was repeated for different message lengths; communication time per message was plotted as a function of the message length in Figure 4.4. The plot shows the mean value of τ_m computed by 16 nodes. Even though different pairs could progress at their own rates, the communication time per message for different nodes was not significantly different. The maximum deviation of τ_m from the mean value is less than 4 μs for messages up to 8 Kb long. This small variation demonstrates that the communication performance of different nodes (hardware and the kernel) are almost identical.

These experiments show that the communication time per message can be specified as a linear function of message length L, i.e. $\alpha + \beta L$. Here, α is the fixed overhead and β is the effective bandwidth. We observe that the fixed overhead is still significant in this second generation machine—about 210 μs. Our measured value is slightly larger than the value of 177 μs reported recently [75]. The effective bandwidth is 1/0.0577 $\approx$ 17 Mbytes/sec for small ($\leq$ 256 bytes) messages which is about 87% of the channel bandwidth (20 Mbytes/sec). Larger messages are sent as packets of 256 bytes. The packetization is enforced[1] to prevent long messages from blocking the transfer of small messages. We observe in Figure 4.4.B that packetization has a noticeable impact on the effective bandwidth. The packetization overhead reduces the effective bandwidth to 1/0.1536 $\approx$ 6.5 Mbytes/sec. The effective bandwidth is again slightly lower than

[1] In fact, there is no way to avoid this packetization in the current system.



A. Short Messages ($L \leq 256$ bytes)

B. Long Messages ($L > 256$ bytes)

Figure 4.4: Communication time for contention-free adjacent-node communication

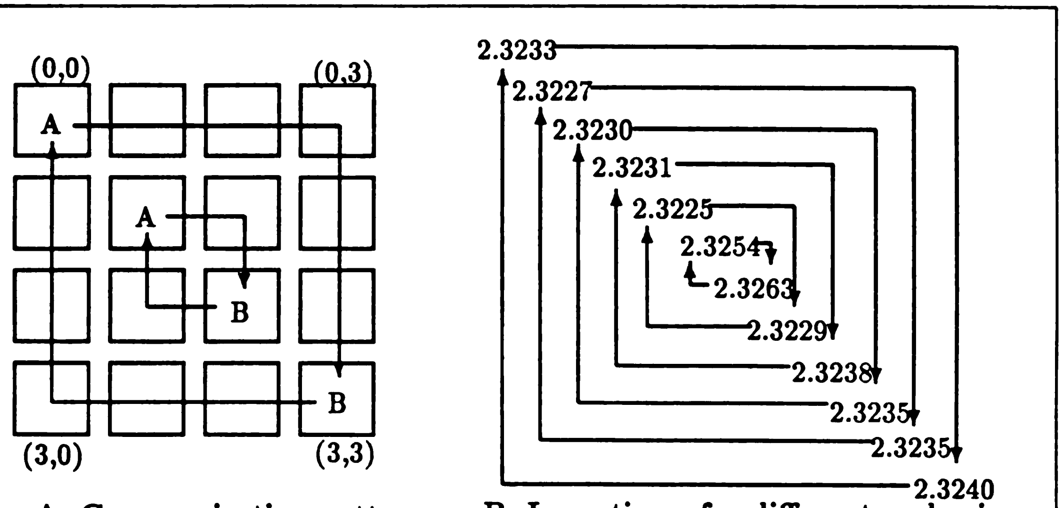
the claimed value of $1/0.11 \approx 9$ Mbytes/sec [75]. Communication of longer messages is about $17/6.5 = 2.6$ times slower than for smaller messages.

4.2.2 Contention-free Non-adjacent-node Communication

In this section, we examine the performance when communication is between nodes that are not physically adjacent. Contention for both rr-channels and pr-channels is to be avoided so we can concentrate on the effect of longer path lengths. We achieve that goal by only allowing nodes along the the leading diagonal to be active. The communication pattern is defined by (Figure 4.5)

$$\text{peer of node}(i,j) = (\sqrt{N} - i - 1, \sqrt{N} - i - 1)$$
$$\text{class of node}(i,j) = (i \neq j) \Rightarrow I, (i < \sqrt{N}/2) \Rightarrow A, (i \geq \sqrt{N}/2) \Rightarrow B$$

Thus, we have $\sqrt{N}/2$ pairs communicating—avoiding any interference between their communications. The result is no contention for network channels. The base loop is the same as shown in Figure 4.3. Hence, contention for pr-channels is avoided as before.



A. Communication pattern **B. Loop times for different nodes in ms**

Figure 4.5: Contention-free, non adjacent node communication

The path lengths for the various pairs are 2, 4, 6, ..., $2\sqrt{N}$-2. If longer path lengths increase delays, we would expect to see communication time (and hence loop time) increase as we move from node ($\sqrt{N}/2$,$\sqrt{N}/2$) to node (0,0). The average loop times for the diagonal elements of a 12×12 2D mesh are shown in Figure 4.5.B. The times are for 4 Kb messages and 450 μs average computation time per loop. The times do not show any dependency on path length. Figure 4.6 shows how time for a single message varies with path length for different message sizes. The change is less than ±4 μs. Since such variations existed for adjacent node communication, we conclude that the effect of path lengths can be ignored in current wormhole-routed networks [2].

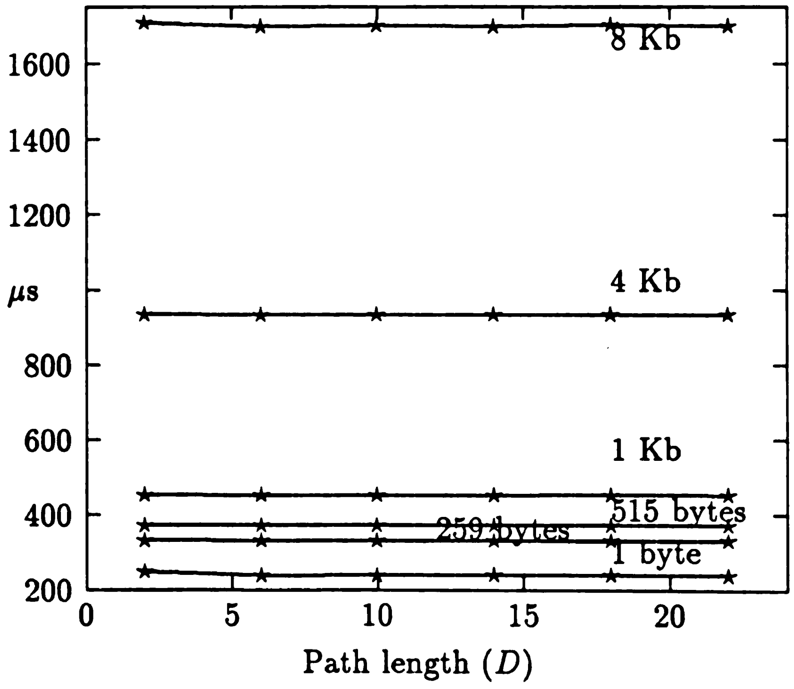


Figure 4.6: Communication time per message as a function of path length

[2] The times here are slightly different than the values given in Section 4.2.1, as the systems were different and so were the CE kernels.

4.2.3 Communication Under Contention

In this section, we describe several experiments designed to study the effect of contention on the communication time. We establish communication patterns so messages will have to compete for network channels (rr_channels) and/or PE. to router channels (pr-channels). Unlike the experiments in the previous two sections, we find that communication time increases exponentially as the computation time decreases.

Contention for Network Channels

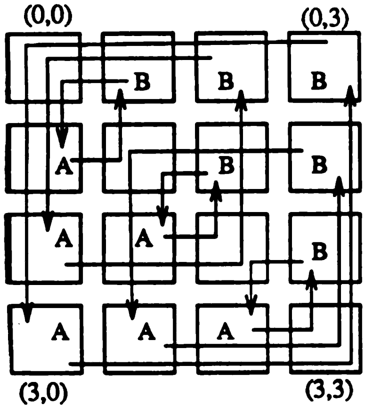
In order to study the effect of contention for network channels, we need a communication pattern in which the nodes are affected differently by contention, and in which node pairs can progress independently of each other. Under these conditions, it is possible to measure the effect of contention by computing the difference in loop times. We call the pattern used here the *matrix-transpose* pattern (Figure 4.7.A).

In this pattern, all nodes along the leading diagonal are idle, and others are active. The communication pattern is defined by

$$\text{peer of node}(i,j) = \text{node}(j,i)$$
$$\text{class of node}(i,j) = (i = j) \Rightarrow \text{I},\ (i > j) \Rightarrow \text{A}\ ,\ (i < j) \Rightarrow \text{B}$$

We have $\sqrt{N}(\sqrt{N}-1)/2$ pairs communicating. The base loop is identical to that of the earlier experiments, and there is no contention for pr-channels. The experiment is more demanding on the network because the peers are widely separated, and the messages between them will result in severe contention for network channels.

An examination of Figure 4.7.A provides an idea of the distribution of contention due to communication in the matrix-transpose pattern. To construct that contention pattern, we have utilized the fact that Symult 2010 systems use a fixed routing scheme. Messages travel along rows first, and then along the columns. The channels connected to the two corner nodes (0,0) and ($\sqrt{N}$-1,$\sqrt{N}$-1) have the maximum loads ($\sqrt{N}$ -1), whereas the channels connected to the other two corner nodes have the lowest loads (1). The loads on other channels vary between those two extremes. Logical path lengths (equal to 1/2 the physical path length) increase as we move away from the



A. Communication pattern

--, 2.085, 2.092, 2.095, 2.102, 2.109, 2.116, 2.110
2.084, --, 2.086, 2.099, 2.104, 2.116, 2.116, 2.115
2.092, 2.086, --, 2.095, 2.101, 2.106, 2.112, 2.110
2.095, 2.099, 2.094, --, 2.092, 2.099, 2.104, 2.103
2.102, 2.104, 2.101, 2.092, --, 2.091, 2.092, 2.092
2.109, 2.116, 2.105, 2.098, 2.091, --, 2.088, 2.086
2.115, 2.116, 2.112, 2.103, 2.092, 2.088, --, 2.086
2.109, 2.114, 2.110, 2.102, 2.092, 2.085, 2.086, --

B. Loop time for different nodes in ms

Figure 4.7: Communication under contention for network channels

leading diagonal. Hence, logical path length is highest for the corner nodes (0,$\sqrt{N}$ -1) and ($\sqrt{N}$ -1,1), and is equal to ($\sqrt{N}$ -2). Path contention levels are higher for upper (lower) row nodes if we consider nodes above (below) the leading diagonal, and vary from 0 to ($\sqrt{N}$ -2).

The loop times for a specific case when the message length is 4 Kb and the mean computation time is 150 μs is shown in Figure 4.7.B. Observe that nodes that are away from the leading diagonal have higher loop times. The observed distribution can be explained. Symult 2010 does not use priority queues. Hence, nodes whose paths have longer logical path length are affected more. For example, a message from node (0,$\sqrt{N}$-1) to its peer node ($\sqrt{N}$-1,0) must compete for each of the $\sqrt{N} - 1$ horizontal segments (channels) of its path. On the other hand, a node near the principal diagonal, such as node (0,1), will only have to compete for the one channel between nodes (0,0) and (0,1). Hence, communication time as well as loop time is higher for node ($\sqrt{N}$-1,0). Later we show that path contention level still determines the increase in communication time for an average and a worst case node.

Let T_{avg} and T_{max} denote the average and maximum loop time taken by any node in this experiment (refer to Equation 4.1). Let T'_{avg} and T'_{max} be the corresponding values when the peers are mapped onto adjacent nodes as in the experiment in Section

3.1 (note that $T'_{avg} \approx T'_{max}$). The increase in loop time for an average node is ($\Delta T_{avg} = T_{avg} - T'_{avg}$), and for the worst node is ($\Delta T_{max} = T_{max} - T'_{max}$). The increase in communication time, $\Delta\tau$, will be equal to the increase in loop time, ΔT, because the computation time for any loop is fixed. The increase in communication time per loop for an average and a worst-affected node are plotted for various systems sizes in Figure 4.8. Note that *decreasing* the average computation time per loop has the effect of *increasing* the communication time. Our results show that the communication time increases exponentially with the communication-to-computation time ratio. The increase is greater for larger system sizes and longer messages.

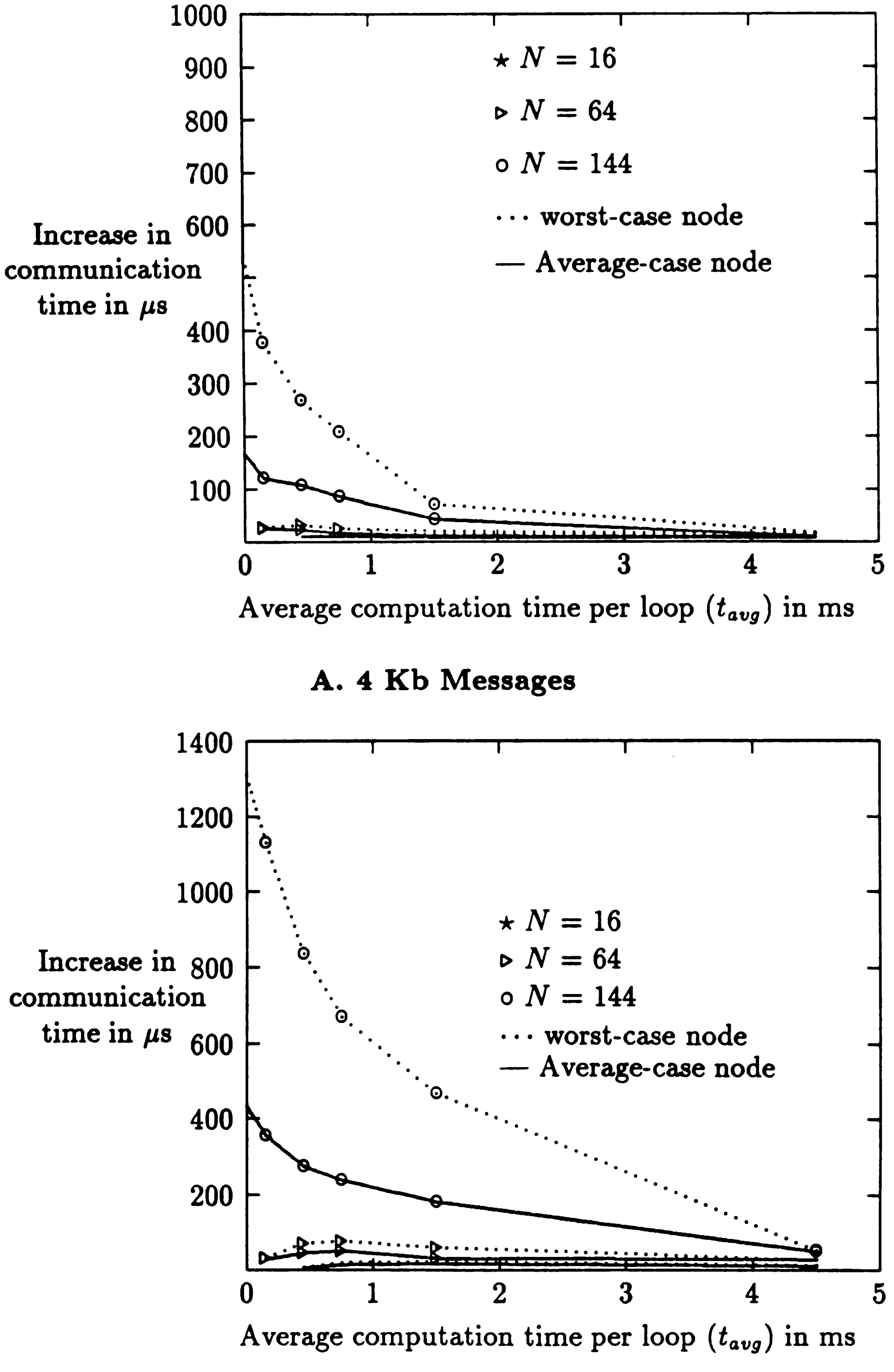
Since we showed in Section 4.2.2 that path lengths cannot account for that exponential increase in communication time, we conclude that the increase in communication time here is due to contention for network channels. The pattern in loop times (Figure 4.7.B) further confirms that the increase can only be due to contention for network channels. Although the results show that the increase in communication time is significant only for large messages and very high communication to computation ratios, analysis in Section 4.3 shows that these delays can occur for smaller messages and more reasonable communication-to-computation ratios in future, larger systems.

Contention for PR-channels

We now examine the effect of contention for the PE-to-router channel, the pr-channel. Contention for pr-channels will occur if a node receives and/or sends several messages simultaneously[3]. Such contention was carefully avoided in all our previous experiments. Contention for pr-channels is more serious in current systems than contention for network channels. For many common communication patterns, contention for pr-channels can occur and increase the communication times significantly.

The architecture of a node (shown in Figure 4.1) indicates the problem. All five channels (four rr-channels and one pr-channel) have the same bandwidth. If messages

[3] Multiple, simultaneous sends and receives will also result in contention for any node hardware involved in sending or receiving a message, including the CPU.



A. 4 Kb Messages

B. 8 Kb Messages

Figure 4.8: Increase in communication time per loop

arrive on more than one rr-channel to a given node, the messages must compete for the single pr-channel and create a bottleneck. This pr-channel bottleneck can slow down an incoming message by a maximum factor of eight. We use variants of the popular near-neighbor communication patterns here to demonstrate the effects of contention for the pr-channel. The communication patterns are shown in Figure 4.9.

In near-neighbor patterns, a node typically exchanges data with one or more neighbors in each loop. Consider the experiment in Section 4.2.1. In that experiment, each node exchanged (sent and received) a message with its peer in each loop. Contention was avoided for both rr-channels and pr-channels. Let us call that pattern the *AB-1NN* pattern[4]. The base loop is the same as given in Figure 4.3. If we now make all nodes in Figure 4.3 belong to the class A, then each node will first initiate a send and then receive the incoming message from its peer (Figure 4.9). We call this pattern the *AA-1NN* pattern. In this pattern, a node will be simultaneously sending and receiving. Traffic exists in both directions on pr-channels as well as rr-channels. Since messages are progressing concurrently, we expect the communication time per loop to be half of that for the $AB - 1NN$ pattern. However, the measured times indicate that it is almost the same as for a $AB - 1NN$ pattern (Table 4.1). We find that if traffic exists in both directions, the communication time for a message is increased by a factor of 2.

Instead of sending and receiving from one neighbor, we next consider the case where a node communicates to all its four neighbors in each loop—a more common situation. Typically, a node sends the same message to all the four neighbors in each loop. The four-neighbor pattern which avoids contention for network channels is given in Figure 4.9 (AB-4NN pattern). The base loops of A and B nodes are given in Figure 4.10. A node (i, j) belongs to class A (B) if $i + j$ is even (odd). Even though contention for network channels is avoided, four messages will be contending for the pr-channel. The communication time for the AB-4NN pattern is four times that of the AB-1NN pattern. Hence, the communication time per loop is eight times that for a single message. As in case of the 1NN pattern, we can place all nodes in class

[4] One Near Neighbor and peer nodes belong to classes A and B.

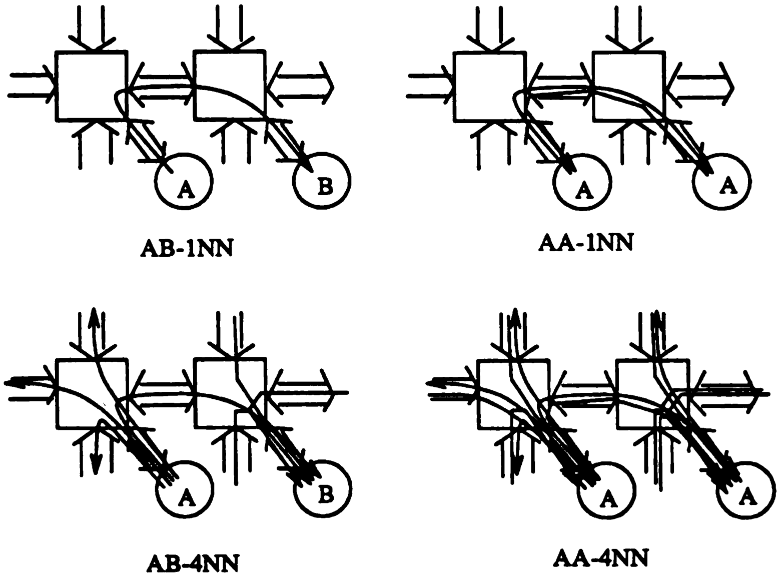


Figure 4.9: Near-neighbor communication patterns

```
            A node                                 B node
for each neighbor do                   for each neighbor do
      tsend(neighbor,msg,L) ;                msg = trecv(neighbor) ;
for each neighbor do                   for each neighbor do
      msg = trecv(neighbor) ;                tsend(neighbor,msg,L) ;
compute(t) ;                           compute(t) ;
```

Figure 4.10: Base loop for four near-neighbor (4NN) communication patterns

Msg ln (in bytes)	*AB-1NN*		*AA-1NN*		*AB-4NN*		*AA-4NN*	
	τ_1	τ_1/τ_m	τ_2	τ_2/τ_m	τ_3	τ_3/τ_m	τ_4	τ_4/τ_m
1	.412	2.00	.385	1.87	2.422	11.75	2.269	11.01
515	.687	2.00	.681	1.98	3.425	9.97	3.059	8.90
1027	.791	2.00	.785	1.98	3.782	9.56	3.683	9.31
4099	1.767	2.00	1.766	1.99	7.648	8.65	7.508	8.49
8196	3.306	2.00	3.295	1.99	12.938	7.82	12.793	7.73

Table 4.1: Communication time per loop for near-neighbor patterns in ms.

(τ_m is the time for a single message under contention-free condition)

$\mathcal{A}$ to create the AA-4NN pattern. The loop times for AA-4NN pattern are given in Table 4.1. Again, we find little difference in the loop times of AA-4NN and AB-4NN, even though nodes in AA-4NN pattern initiated all sends first before receiving any messages in a loop. As before, the problem is due to the contention for the pr-channel by the four incoming and the four outgoing messages.

The experiments in this section lead us to the conclusion that the pr-channel or node-architecture can become a serious performance bottleneck in current systems. One solution is to increase the bandwidth of the pr-channel, and design the memory with sufficient bandwidth so that multiple incoming and outgoing messages can be handled without any degradation in performance. Another solution is to improve the router to provide multicast communication support. This support will be useful as a node will often be trying to send the same message to more than one node. In that case, a single message on a pr-channel could generate multiple copies of itself on the network channels, and significantly reduce the bandwidth requirements of the pr-channel.

4.3 Analysis of Experimental Results

In Section 4.2.3, experimental results showed that contentions for network channels can increase communication times exponentially. The increase in communication time was significant (>10 μsecs) only for messages larger than 1 Kb. In this section, we analyze the results further to examine if the increase in delays can be significant in future systems even for smaller messages and higher computation times per loop. Theoretical results in Chapter 2 had indicated that normalized communication time should be a function of applied node traffic. We also verify if experimental results agree with the theory. To convert the experimental data, we need to take a look at the various components of loop time.

Consider the components of loop time for contention-free communication for the base loop shown in Figure 4.3 (refer to the experiments in Section 4.2.1). Figure 4.11 shows the components for a specific instance when the message length is 4 Kb and the average computation time per loop is 1 ms. The communication time per loop includes the time to send a 4 Kb message from an A node to its peer (A-B communication) and the time for another 4 Kb message from the peer back to node A (B-A communication). Each message was shown in Section 4.2.1 to take 210.2+0.1536×4096 = 839 μs. Of this, 210 μs is fixed overhead and 629 μs is needed to send 4096 bytes as 16 packets of 256 bytes each. Each packet therefore takes 629/16 = 39.3 μsecs. The data of individual packets are communicated over the network channels at the rate of 0.0577 μsec/byte, or 0.0577×256 = 14.8 μsecs per packet as measured for smaller messages (Figure 4.4). These times are shown as shaded regions in Figure 4.11. The remaining time, (39.3 - 14.8) = 24.5 μsecs is the overhead to process each packet.

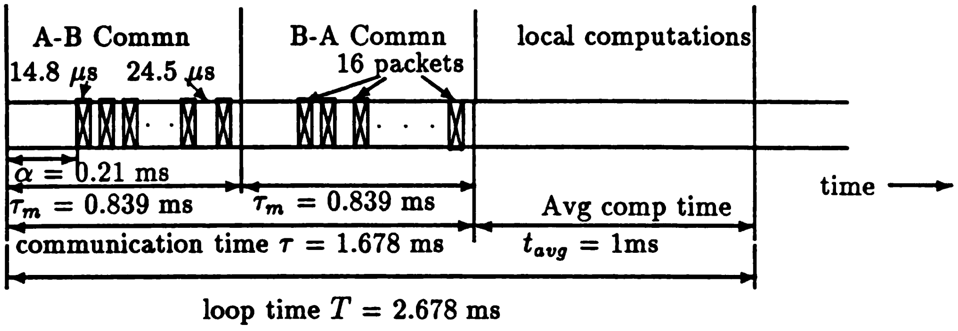


Figure 4.11: Components of loop time for 4 Kb messages

An understanding of the components of loop time indicates how to express the experimental data in terms of applied node traffic and normalized communication times. For contention-free communication, we have by Equation 4.2, $T = t_{avg} + 2\tau_m = t_{avg} + 2(a + bL)$ where a and b are system dependent constants whose values can be easily determined (Section 4.2.1). The values for Symult 2010 are shown in Figure 4.4. Hence, we find that for our base loop, applied node traffic (Definition 24) is

$$\lambda_a = (L/B)/(t_{avg} + 2(a + bL)) \tag{4.3}$$

Observe that applied node traffic depends only on the characteristics of a parallel task (L, t_{avg}) and the machine dependent constants (B, a, b). The significance of λ_a is that under contention-free conditions, the pr-channel of a node will be busy injecting data into the network for λ_a fraction of time. For the example in Figure 4.11 $\lambda_a = \frac{(4096/20\times10^6)}{2678\times10^{-6}} \times 100 \approx 7.64\%$.

We can also normalize the communication time per message under contention to the communication time under contention-free conditions less the fixed overheads, bL. The results of the experiments (Section 4.2.3) shown in Figure 4.8.A gives the increase in communication time per loop $\Delta\tau$. Since, each loop (Figure 4.3) involves the communication of two messages, the increase in communication time per message will be $\Delta\tau_m = \Delta\tau/2$. The normalized communication time therefore will be given by

$$\theta = \frac{bL + \Delta\tau_m}{bL} = \frac{bL + \Delta\tau/2}{bL} \tag{4.4}$$

The results given in Figure 4.8.A for a 144-node system are represented in terms of applied node traffic and normalized communication time in Figure 4.12. Figure 4.12 shows that for the same applied node traffic, the increase in communication time is greater for larger systems. We also observe that some nodes (such as the worst-case node) perform much worse than an average node. The behavior can be easily explained. The maximum path contention level for the matrix transpose pattern is $\nu_{max} = (\sqrt{N} - 2)$, and maximum channel load is $\chi_{max} = (\sqrt{N} - 1)$. For a 144 node system $\sqrt{N} - 2 = 10$. Hence, from Equation 3.11 we find that the saturation node traffic for the worst-case node is $\lambda^{w}_{n_{sat}} = 1/(10 + 1) = 9.1\%$. A 9.1% node traffic will saturate physical channels as $\chi_{max} = 11$. We can therefore expect a dramatic increase in communication time when applied node traffic is higher than $\lambda^{w}_{n_{sat}} = 9\%$.

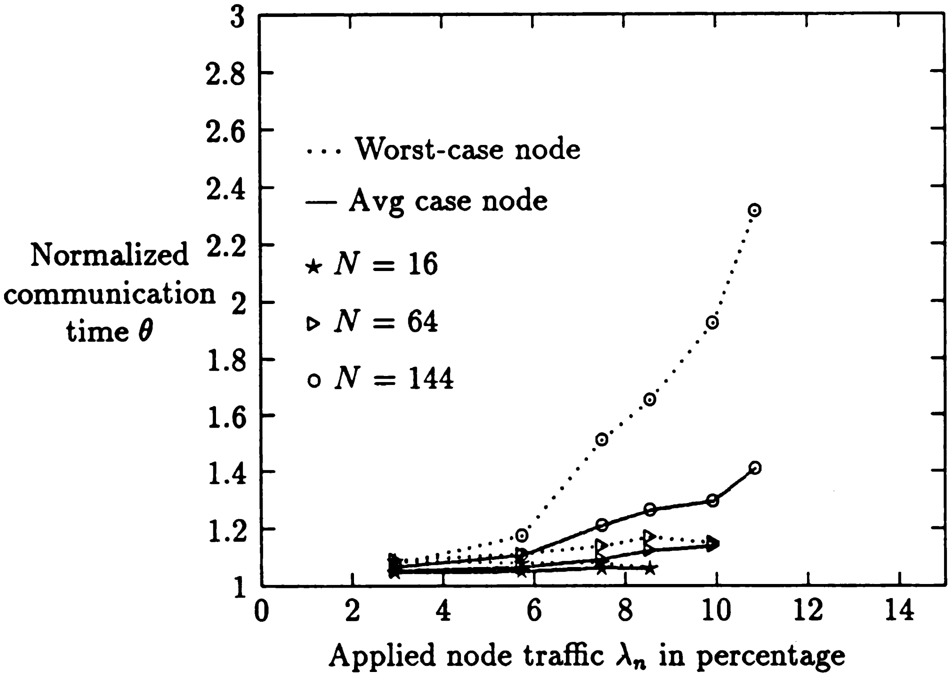


Figure 4.12: Normalized communication time per message for different system sizes. (Matrix-transpose pattern, 4 Kb messages)

Figure 4.13 gives the normalized communication time per message for three different message lengths for a 144-node system using the matrix transpose pattern. The graph clearly shows that the normalized communication time is a function of the applied node traffic and is independent of the message length. The plots in Figure 4.13 stops at $\lambda_a = 13\%$ because overheads and architectural bottlenecks in the Symult 2010 limited applied node traffic. If fixed overheads and architectural bottlenecks are removed in future systems, applied node traffic will increase for the same message length and computation time (as the values of a and b in Equation 4.3 are reduced). Such an increase in applied node traffic will seriously degrade the communication performance. Figure 4.13 also shows why we do not observe the effects of contention for less than 1 Kb messages in a Symult 2010. In that case, the overheads limit the applied node traffic to less than 6% for small messages, reducing the percentage increase in communication time. In addition, for small messages the communication time τ_m is also small. Hence, the absolute increase in communication times, $\Delta\tau_m$, will be too small to be observable. However, with reduced overheads, increased applied node traffic is possible, and we will observe the effect of contention even for small messages.

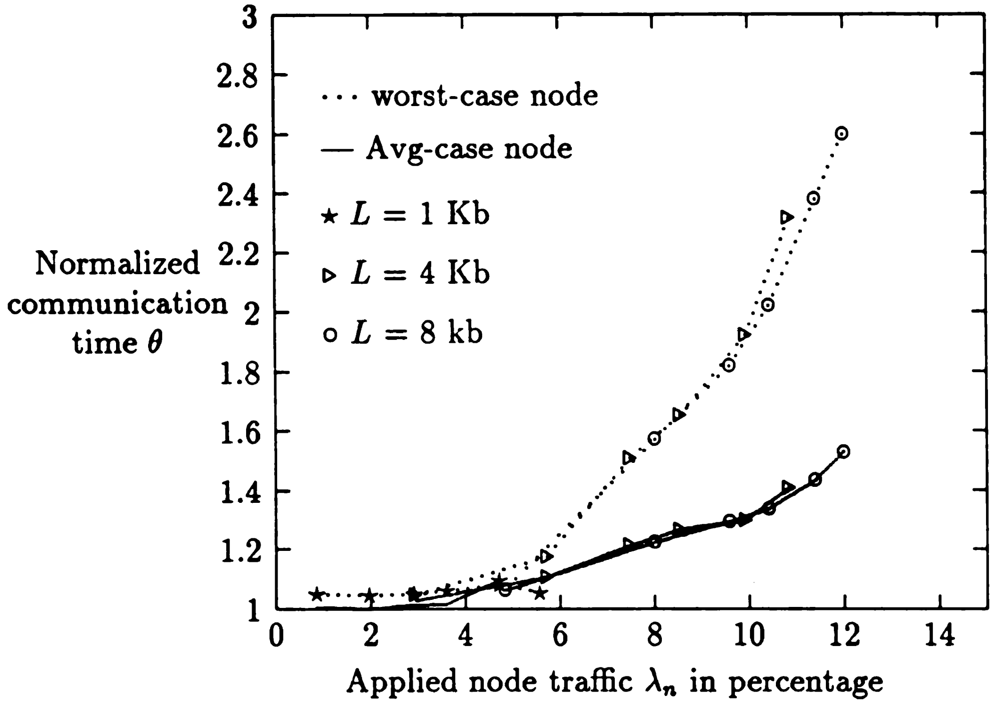


Figure 4.13: Normalized communication time per message for different message lengths.

(Matrix-transpose pattern, 144 node system)

One may argue that even such an increase in communication time may not affect the overall loop time significantly, since communication is only a small part of the loop time. We could plot the percentage increase in loop time instead of percentage increase in communication time. However, an interesting and useful way to present the effect of contention on the overall performance is to plot *actual* node traffic (Definition 26) as a function of applied node traffic (Definition 24). An increase in communication time will increase the loop time which will reduce the actual node traffic. The results given in Figure 4.13 are presented in terms of actual and applied node traffic in Figure 4.14.

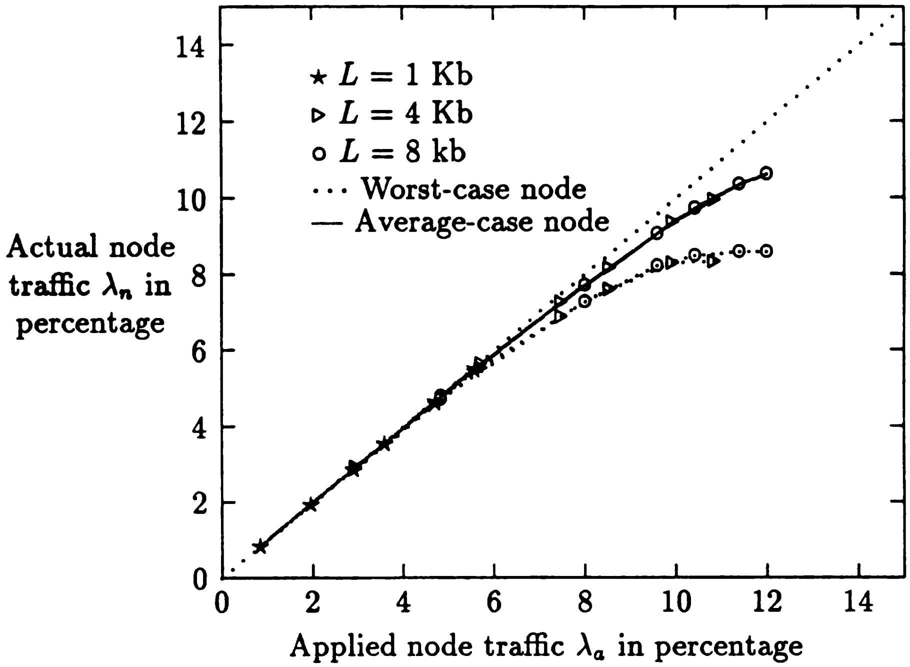


Figure 4.14: Actual node traffic for different message lengths

(Matrix-transpose pattern, 144 node system)

If the contention effect is negligible on the overall performance, the actual node traffic, λ_n, should be approximately the same as the applied node traffic, λ_a. The curves in Figure 4.14 clearly shows that for a 144-node system the actual node traffic begins to saturate at about 9% applied node traffic for the worst-case node as expected. The effect of contention on the overall performance (or loop time) then will be visible if a node attempts to generate a traffic which is more than the saturation level. Applications can never achieve a node traffic higher than the saturation level. The level at which actual node traffic saturates in general depends on the communication pattern, the mapping and the size of the system as elaborated in the previous chapter.

4.4 Discussion

In the previous section we saw that although path lengths are not a problem in wormhole-routed networks, contention can be a serious problem. Our experiments were conducted on Symult 2010 systems, which may not be a typical representative of the class of 2D mesh connected, wormhole-routed networks. In this section we discuss some of the architectural idiosyncrasies and drawbacks of the Symult 2010 system which may not be present in future systems, and extend our results to such systems.

1. Processor-to-Router channel and node architecture : In a Symult 2010, the Processor-to-Router channel (pr-channel) bandwidth is the same as that of the network channels. Hence, the pr_channel can become a bottleneck for several communication patterns, and it limits the peak rates at which a node can inject traffic into the network. It is possible to increase the bandwidth between the router and the processor by either increasing the channel width or the channel rates. Ideally four times the bandwidth of the network channels in each direction should be sufficient to eliminate this bottleneck. In addition, the node and its kernel should be designed to handle multiple incoming and/or outgoing messages. Future designs (such as iWarp [14]) are increasing the bandwidth between the router and the PE. With these improvements, however, higher applied node traffic is possible and the problem of contention for network channels can be increasingly visible.

2. System Size : At present, the largest Symult system has 192 nodes. However, system sizes are expected to be as high as 1K nodes or more in the near future. Applied node traffic is a characteristic of the parallel program, and parallel programs should be designed to be scalable. Hence, applied node traffic remains the same as system size increases. As we saw earlier, the utilization of some channels can increase to very high levels due to nonuniform traffic even though applied node traffic remains reasonable. Longer path lengths in larger systems will force a message to compete for more channels before successfully establishing a path for communication. With larger systems sizes, interference between traffic due to different users (or applications) and

between system and user traffic will also increase. Hence, the contention effect will be increasingly visible in larger systems, and contention may have to be studied seriously if inefficient performance is to be avoided.

3. Fixed software overheads : In current systems, the fixed overhead per message is about 210 μsecs which is quite high compared to network delays. It is possible to reduce this overhead further by a careful design of the node architecture, good interface between the communication and the computation processors and restricting the flexibility of the message primitives. Some of these improvements are being implemented in latest multicomputers [14, 30]. As explained in the previous section, reduced overheads can increase peak applied node traffic, and hence the effect of contention. Even with current overheads, which limit peak node traffic to about 12%, the effect of contention will be visible in large systems as saturation may occur at less than 10% node traffic.

4. Packetization overheads : In a Symult 2010, messages larger than 256 bytes are sent as 256 byte packets. Packets were introduced to prevent large messages from affecting the communication of smaller messages due to contention for channels. However, packetization introduces additional software overhead which causes larger messages to progress 2.7 times slower than smaller messages. In addition, there is no way a user can prevent this, even if he is sure that he is avoiding contention, and is not interfering with the communications of other applications. Increasing communication times by a factor of 2.7 may reduce overall performance significantly and may not be acceptable. Such packetization cannot be very effective if channels are getting saturated. Turning off packetization will increase the applied node traffic, and hence the effect of contention. However, we may achieve a significant speedup if we turn off packetization, and carefully avoid contention.

5. Multicast and Broadcast support : At present, support for multicast and broadcast communication is very inefficient. In fact, if the host needs to broadcast a message to all nodes, it will simply send a copy to each node. Only one node is connected to the host, and all the messages from host to nodes must pass through a single channel. This implementation will slow down the broadcast by a factor of N.

Similarly, if a node has to send the same message to more than one node, it has to make copies of the message and send one copy to each node. Architectural support for broadcast and multicast will be useful. However, such support will also increase network traffic for a given applied node traffic, and hence the problem of contention for network channels.

6. Wider Data paths : Future systems are being designed to have wider data paths. As the width of the channels increases, the network delays become smaller which will reduce the node traffic and utilization of channels for a given application. Hence, increased width is one main factor which may reduce the applied node traffic and hence the the problem of contention. Detailed studies are needed to determine whether increases in channel widths can compensate for all the earlier factors which increased the problem of contention.

We see that a number of trends in network architecture indicate that the problem of contention for network channels will become serious in near future. It is not clear if the higher bandwidth possible in mesh networks can compensate for the changes, and enable the mesh networks perform better than hypercube networks.

4.5 Conclusion

The results of our experiments and the analysis of these results indicate of the performance of a state-of-the-art mesh-connected wormhole-routed interprocessor communication network. The communication time per message under contention-free conditions is one of the best any implementation has achieved, and is given by the linear function $210.9 + 0.0577L$ μsecs ($210.2 + 0.1536L$ μsecs if $L > 256$) where L is the message length. This expression is accurate to within ± 4 μsecs, even for paths of length 22. Hence, with current technology we find that the effect of path length on communication performance can be ignored.

However, contention for network channels may turn out to be the new problem as predicted by theory. Contention is a more serious problem than path length as it can result in an exponential increase in the communication time once communication intensity of an application reaches a certain threshold. In the Symult 2010, the effect of contention is significant (> 10 μsecs) only for large systems (> 100 nodes) and for long (> 1Kb) messages. Large overheads are limiting the applied node traffic to less than 13%, and hence the effect of contention in small systems currently available. However, our analysis indicates that the effect of contention may be visible even at applied node traffic of less than 10% for large systems not yet available so the problem of contention can become serious in the near future.

Improvements are needed in the current architecture to reduce overheads, to remove bottlenecks in the node architecture, and to provide efficient broadcast/multicast support. Future multicomputers under development are already planning to provide some of the improvements. Although such improvements will improve the network performance for contention-free communication, those improvements will also increase the problem of contention for network channels. The network performance will become more sensitive to the contention.

Chapter 5

NETWORK SIMULATOR

5.1 Introduction

We decided to develop a high-performance simulator that can be used to study the performance of large networks. Such a simulator is needed for several reasons. Experimental studies on systems larger than 200 nodes are not possible as such large multiprocessor systems that use wormhole-routing or circuit-switching are not yet available. Such large networks and systems are, however, expected in the near future. In addition, experiments reveal only the characteristics of a specific system or network, and may fail to highlight the characteristics common to all wormhole-routed networks. Theoretical analysis in earlier chapters clearly showed that contention can be serious problem in large networks, and appropriate mapping can result in significant improvement in performance. Simulation studies can be used to confirm the simulations. If the results of the theory and simulations agree, then that would validate our simulator, too. Once the simulator is validated, it can be a useful tool to investigate various cases that are difficult to model and analyze by theory. To our knowledge, there is no published literature that describes the design of a general-purpose, efficient wormhole-routed network simulator that can simulate large networks in reasonable time. We were, therefore, motivated to design a simulator which can be used to study the performance of large, wormhole-routed networks.

We had the following goals for our simulator

- General purpose : The simulator was not intended to study any particular multiprocessor system or network. It was intended to model the fundamental characteristics of a wormhole-routed network, and to be sufficiently general so we can study the performance characteristics common to all wormhole-routed networks.

- High performance : We wanted the simulator to be fast enough so that we can use it to study large networks having hundreds or thousands of nodes.

- Flexible : It should be easy to modify the simulator to include specific characteristics of a machine so that a specific machine can be simulated more accurately if needed.

- Portable : The simulator should run on any computing resource available, and is not to be designed to run on a specific machine or require special support from the operating system.

We describe how we managed to meet our goals, and show that our simulator can be useful in understanding the behavior of modern wormhole-routed networks. In this chapter, we concentrate on the performance of 2D mesh networks as they have potential for high performance, and are also significantly affected by contention. However, the simulator can be easily modified (as explained later in the chapter) to study performance of networks having other topologies. In Section 5.2, we describe the design of the simulator kernel and explain how we could make it simulate all the essential features of a wormhole-routed network, and still keep it simple and efficient. Additional code, called the simulator shell, needs to be written to complement the kernel and complete the simulator. We describe the design of a simulator shell to study the effect of mapping of the performance on the network in Section 5.3. In Section 5.4, we describe how we validated our simulator by comparing its outputs with results from experiments on a real system and from theoretical analysis. We present the simulator results for several experiments in Section 5.6 which demonstrate the usefulness of the simulator in understanding the behavior of wormhole-routed networks. Details of the performance of the simulator on a Sun Sparc workstation are given in Section 5.5. The results show that our simulator can give useful results in reasonable time even for large systems. Finally, we conclude and mention limitations of our simulator, and possible improvements to make the simulator more powerful and flexible.

5.2 Simulator Design : Kernel

The major challenge in designing the simulator was to make it fast enough so we can simulate large networks in reasonable time and get useful results. After careful study, we chose to implement the simulator in C, a high level language, rather than a simulation language. The network model we simulate is simple, and we do not require the power of a simulation language, and we can avoid the processing overheads of a simulation language. Simulators using a language such as CSIM have been found to be too slow to simulate large wormhole-routed networks [68]. Our kernel design is similar to that of another work reported recently [61].

For the first version of our simulator, we made the following assumptions that let us concentrate on the main characteristics of a wormhole-routed network. These assumptions simplify the simulator design, and hence improve the simulator performance.

- All network channels are assumed to have the same bandwidth and all transmissions are assumed to be error-free.
- Channels between adjacent nodes are assumed to be full-duplex.
- A node-processor or memory will never be a bottleneck. We made this assumption as we were interested in studying effects of contention and non-uniform traffic, and not the limitations of a node's capability on communication performance.
- All routers are synchronized by a global clock. We assumed that the duration of a clock period is sufficient to process all the newly arrived data flits by a router.
- No support for virtual channel.
- One flit buffer per channel in each direction.

At the same time, the following essential features of a wormhole-routed network are retained, so the simulator predicts the network performance reasonably well.

- reservation of network channels and datapaths by the message header
- small headers of one or two words
- pipelined flow of message flits that follow the header
- release of reserved channels by the tail flit of the message
- limited number of buffers per router, typically one or two per channel
- blocking the the message flits if the header is blocked.

We also decided to implement the main functions in the form of a kernel, a library that supports key functions. A problem-specific shell can be then written to complete the simulator. Such a design will help us optimize the key functions, and at the same time keep the simulator quite flexible. We first describe the main datastructures, and then the algorithms for key kernel functions. Design of a simulator shell is described in the next section.

5.2.1 Kernel Data Structures

The overview of the network model simulated by the kernel is shown in Figure 5.1, and is similar to the model used in another work recently [61]. There are N nodes, each node consisting of a PE and a router. Each router, and hence the corresponding node, is connected to a limited number of other nodes by physical links. The PE's generate and absorb messages, whereas the routers communicate the messages from one node to the other. We need data structures to remember the state of the links, the routers and the PE's.

Each output port is connected to the input port of an adjacent node by a physical link. The state of a link or an input port is not remembered explicitly as they can be deduced from the state of the corresponding output port, We, therefore, can represent the status of the network and the routers by a $N \times \delta$ array of PORT structures, each structure remembering the status of an output port (Figure 5.2). Here, δ is the degree of a node, i.e. number of output or input ports in a router.

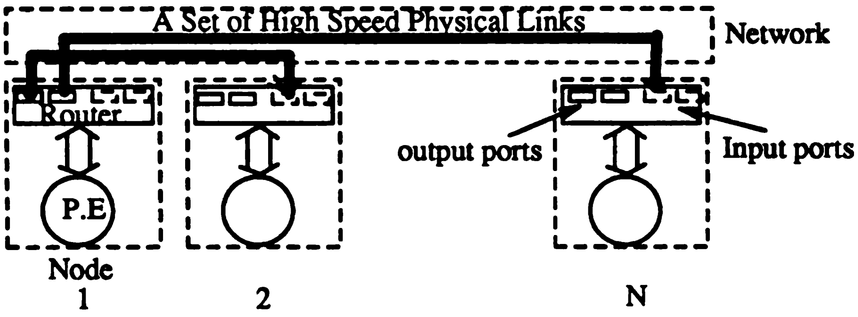
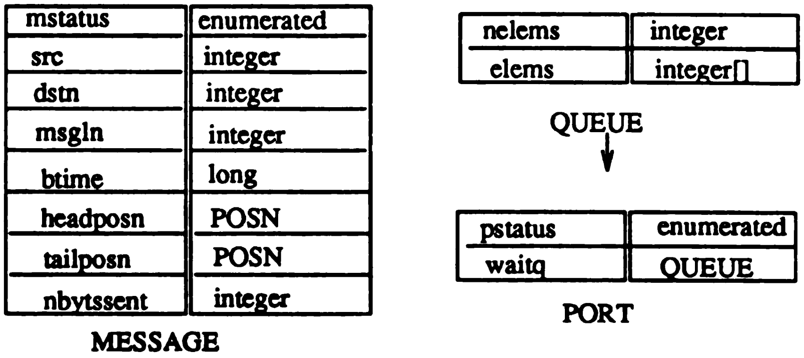


Figure 5.1: Overview of the network model



mstatus	enumerated
src	integer
dstn	integer
msgln	integer
btime	long
headposn	POSN
tailposn	POSN
nbytssent	integer

MESSAGE

nelems	integer
elems	integer[]

QUEUE

pstatus	enumerated
waitq	QUEUE

PORT

Figure 5.2: Main datastructures of the simulator

Each PORT has two fields: pstatus and waitq. The field pstatus stores the state of the output port. Its value will be IDLE, if it is not currently reserved for any message. Otherwise, its value indicates the message which has reserved it. The other field waitq is a small FIFO or priority queue. The queue is not for message flits, but for message headers that have arrived on input ports, and are waiting to reserve this port and the corresponding physical link. When the tail flit of the message currently holding the port passes through the port, it will release the port and the channel. Once the port becomes free, the first message in the waitq will get the channel. In an actual network, the elements of waitq may be input ports. In our simulator, each queue element is the index of a message, as it is more convenient to process.

A PE is treated as a source/sink for messages. Observe that every incoming message corresponds to an outgoing message from some other node. The kernel handles only the messages currently in the network and leaves other details of the PE behavior, including generation of messages in future, to the shell. Hence, the status of PEs is represented by a $N \times \delta$ array of MESSAGE structures, each MESSAGE structure remembering the status of a message currently in the network, which has the following fields:

- **mstatus**: status of the message: IDLE if there is no message at present, BUSY if the message is not blocked and is progressing, or WAIT if the message is blocked and waiting for a network channel.
- **src, dst, msgln, btime**: give the details about the message: source node, destination node, length of the message and the time at which it entered network respectively.
- **headposn, tailposn**: position of the head and tail flits respectively. If the value of the headposn (tailposn) is i, then the head (tail) flit of the i^{th} message has arrived at an input port of node i. Tailposn will be -1 if the tail flit has not left the source PE yet.
- **nbytssent**: indicates the number of message flits that have already left the source PE

Arrays of PORT and MESSAGE structures can remember the status of the entire simulator kernel. The total space needed is O(N), and hence we find that our simulator is space efficient.

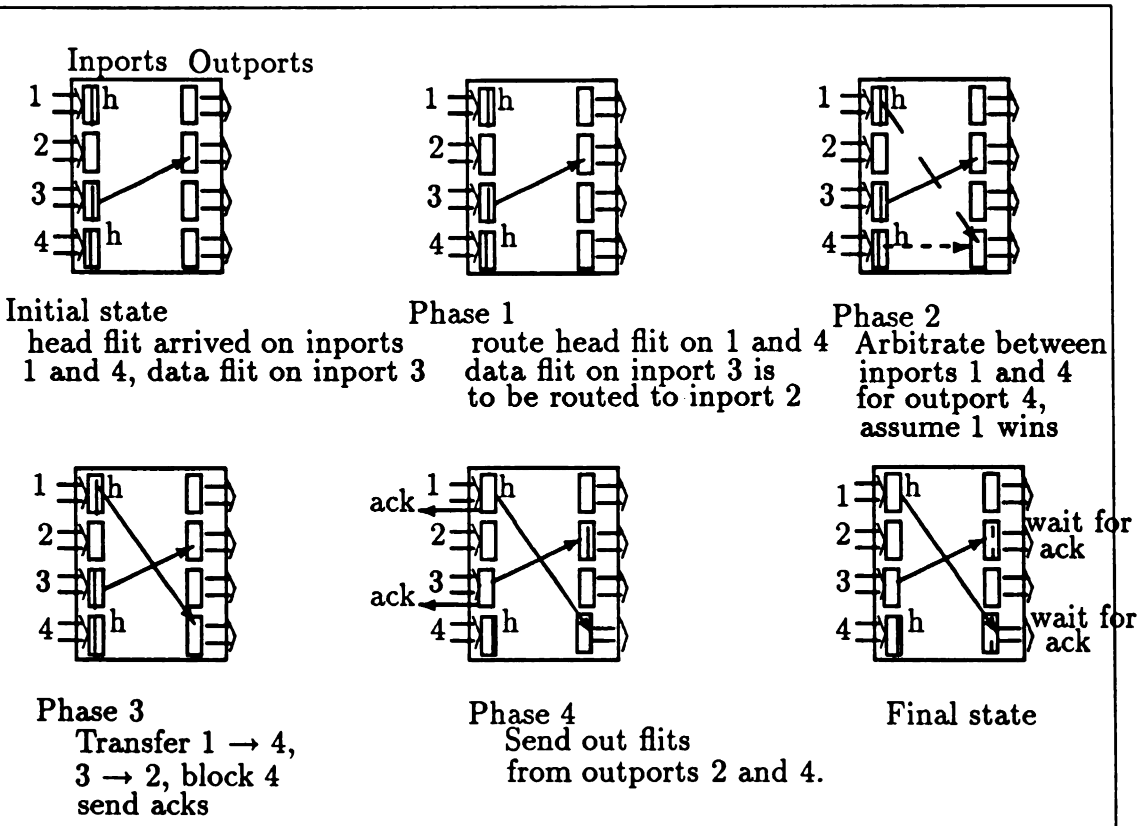
5.2.2 Simulation Cycle

The kernel functions essentially take the simulator through one simulation cycle (Figure 5.2.2). It is necessary to understand the functions carried out by a router in one simulation cycle before we can describe kernel functions. The initial state of a router before a new simulation cycle is as follows. Some of the input ports would have received message flits from adjacent nodes in the previous cycle and need to be processed. Other input ports are currently either blocked or idle. Some of the output ports would have sent flits to adjacent nodes and are waiting for acknowledgment. Other output ports are either idle or waiting for the next data flit to arrive. We decided on a four-phase simulation cycle based on related work [77, 61].

Phase 1, Routing: In the first phase, the router has to decide the output port for each newly received message flit. If the flit received is a header flit, then the router has to decode the header, and decide the output port to which the header should be sent. Otherwise, i.e. the received flit is not a header, its output port can be inferred by the path taken earlier by its header.

Phase 2, Blocking and Arbitration: In the second phase, the router has to decide if the required output port is free for each flit. For non-header flits, it just checks if the previous flit of that message has been successfully transmitted to an adjacent node so the output port is free to receive the next flit. For a header flit, it has to check if the required output port is idle. It also has to compete and win the arbitration if headers on several input ports are requesting the same output port.

Phase 3, Internal data transfer: (input port to output port) In this phase, the router will transfer all unblocked flits from their input ports to their specified output ports and send an acknowledgment to the adjacent nodes which sent these flits. All output ports for which acknowledgments are received are marked ready, and if an acknowledgment is received for a tail flit then the output port is marked IDLE.



Initial state
head flit arrived on inports 1 and 4, data flit on inport 3

Phase 1
route head flit on 1 and 4 data flit on inport 3 is to be routed to inport 2

Phase 2
Arbitrate between inports 1 and 4 for outport 4, assume 1 wins

Phase 3
Transfer 1 → 4, 3 → 2, block 4 send acks

Phase 4
Send out flits from outports 2 and 4.

Final state

Figure 5.3: A simulation cycle

Phase 4, External data transfer (output port to input port): In this last phase, all output ports that received new flits in Phase 3 will transmit them to the respective adjacent node.

At the end of the simulation cycle, we are back to the initial state except that all unblocked message flits have advanced by one node.[1]

5.2.3 Kernel Algorithms

The computational requirement will be high if we process each flit of each message. We reduced the requirement significantly, and made the simulator computationally efficient based on the following observations.

• Handling message flits: Instead of processing each message flit, we process only the head and tail flits. The movement of other data flits is implicit as they are between the head and tail flits.

• Idle routers: The fraction of routers that have an event to process in a cycle is small. We skip idle routers in a simulation cycle by scanning the list of active messages instead of scanning routers.

• Blocked messages: Any message whose header is waiting for the release of a channel is marked to be in the WAIT state, put in the appropriate `waitq` and is not processed until the required channel is free. The message will be signaled when the channel becomes free (when the tail flit passes through). Hence, in each cycle we process only the unblocked messages.

• Arbitration policy: We also need to handle efficiently the case of two or more headers which arrive in the same cycle and compete for the same output port. Each port is associated with a priority queue, and each message is associated with some priority. In Phase 1, all headers which need the channels are inserted into the queue. In the second phase, each header checks if it is at the head of the queue to determine if a channel is granted to it.

[1] Observe that it takes two cycles to complete the transfer of a message flit, and hence message flits follow one another separated by 2 nodes. However, if we have two buffers for each outport, then it is possible for the message flits to follow one another separated by just one node.

By default, the priority value of any message is the time it was injected into the network, and the the message having the lowest priority value will be the winner. Such a policy will ensure that the network tries to minimize communication time for each message when there is contention. Also, in such a policy no message can wait forever. We later demonstrate that these improvements result in a simulation rate per cycle which is independent of the system size, and is only proportional to the number of progressing messages.

The following are some important functions implemented in the kernel.

1. **route (curnode,dstnode,outport,nextnode)** The dependence of the simulator on network topology and on the routing scheme is constrained to only this function so we can easily adapt our simulator to any network topology. Given a current node number, **curnode**, (for a head or a tail flit) and destination node number, **dstnode**, the function **route** will determine the next node along the path to the destination and the output port the flit should be routed to (which is connected to an input port of **nextnode**).

The function is given for 2D mesh network below. Here, **sn** is a global variable whose value will be $\sqrt{N}$. The function is simple even for hypercube or 3D mesh networks, and can be implemented in a few statements of C language.

```
procedure route (curnode, dstn, nextnode, nextport)
   curnode, dstn : integer ; /* Inputs, curnode <> dstn */
   nextnode : integer; PORT *nextport ; /* outputs */
begin
  fn = curnode - (curnode % sn);/* First node in curnode's row */
  nextnode = curnode ;
  if(dstn < fn) { nextnode -= sn ; nextport = UP ; }
  else if(dstn >= (fn + sn)) {nextnode += sn ;  nextport = DOWN;}
  else if(dstn < curnode) { nextnode-- ;  nextport = LEFT ; }
  else if(dstn > curnode) { nextnode++ ; nextport = RIGHT ; }
  else /* dsnt == curnode */ { nextport = PE ) ;
end
```

2. **add_message (srcnode, dstnode, msgln)** Given a source node, **srcnode**, a destination node, **dstnode**, and message length, **msgln**, this function will add a message to the network in the current simulation cycle. The function will search for an available MESSAGE structure, initialize it appropriately, and add it to the active message list.

3. **advance_message**: This is a key function which will attempt to move a single, unblocked message through the network by one step. It executes phase 3 and 4 of the simulation cycle. It will advance the position of the header and/or tail flit, if that flit is in some intermediate node, and it increments the number of flits that have left the source. The function will also signal the first waiting message in the queue associated with the channel being released by the tail flit. The function is given below.

```
 procedure advance_msg(msgp)
           MESSAGE *msgp ;
 begin
   if(msgp->headposn <> msgp->tailposn) /*advance header flit*/
         route(msgp->headposn,msgp->dstn,nextnode,nextport) ;
         msgp->headposn = nextnode ;
   endfi
   if(msgp->nbytssent < msgp->msgln)
         /* Release one more flit from source */
        msgp->nbytssent += 1 ;
   else if(msgp->tailposn != msgp->dstn) /*move the tail flit*/
        route(msgp->tailposn, msgp->dstn,nextnode,nextport) ;
        if(!empty_queue(nextport->waitq))
         /* Signal the message at the head of the wait queue */
           msgp = get_queue(nextport->waitq) ;
           msgp->mstatus = BUSY ;
       endfi ;
        msgp->tailposn = nextnode ;
   else /* tail has reached destination node */
        msg_finished(msgp) ; /* inform shell */
        msgp->mstatus = IDLE ;
   endfi
end
```

Here, the functions **put_queue**, **get_queue** and **first_elem** operate on **waitq**, and put an element, get an element or examining the first entry respectively.

4. **advance (n)** : This function will make the simulator execute n simulation cycles, and is the main function supported by the kernel. The algorithm for **advance** is given below, where the global variable **simclock** is the virtual time of the simulator. Here, **user_fn** is a function implemented in the simulator shell which will simulate the behavior of the PE's at the start of a new cycle. Typically, as we see later, **user_fn** may inject several fresh messages by calling the **add_message** function.

5.3 Simulator Shell

The interface between the simulator shell and the kernel is shown in the Figure 5.4. The shell has to essentially implement three functions **main**, **msg_finished** and **user_fn**. The shell functions call the kernel functions **init_kernel**, **add_message** and **advance**. The kernel function **advance** calls the shell functions **msg_finished** and **user_fn** on specific events. The function **msg_finished** is called on the event of the tail flit reaching the destination node indicating completion of a message transfer. The function **user_fn** will be called by the **advance** function at the beginning of each simulation cycle. A proper design of **msg_finished** and **user_fn** can implement the behavior of the parallel tasks running on the PE's—the goal of the shell module.

In this section, we briefly describe a simulator shell for studying the effect of mapping on network performance. We use the example to illustrate how to write a simulator shell for a specific problem, and to show that the shell can be simple and efficient. The mapping problem is to find an optimal assignment of the tasks of a parallel application to the nodes of a parallel system to get the best communication performance.

The shell has to simulate the behavior of the individual tasks of a parallel application running on different nodes. A parallel task, typically, performs local computations for some time, and then sends a message to one of its neighbors (in the process

```
 procedure advance(n)/* Advance simulation clock by n cycles */
     n : integer ;
 begin
     n += simclock ;
     while (simclock <> n) do /* Execute a simulation cycle */
        user_fn() ; simclock++ ;
        /* Phase 1 */
        for each active, unblocked message, msgp, do
           if(msgp->headposn <> msgp->dstn)
           /* If header has not reached the destination */
          /* Attempt to get the next segment of the path */
            route(msgp->headposn,msgp->dstn,nextnode,nextport);
            put_queue(nextport->waitq,msgp) ;
           endfi
       endfor
      /* Phase 2 */
       for each active, unblocked message, msgp, do
         if(msgp->headposn <> msgp->dstn)
            route(msgp->headposn,msgp->dstn,nextnode,nextport);
            if(nextport->pstatus <> IDLE  or
                first_elem(nextport->waitq) <> msgp) /* Wait */
              msgp->mstatus = WAIT ;
            else /* Port granted, remove entry from queue */
               get_queue(nextport->waitq) ;
      /* Phase 3 and 4 */
      for each active, unblocked  message, msgp, do
           advance_msg(msgp)
    endwhile ;
end
```

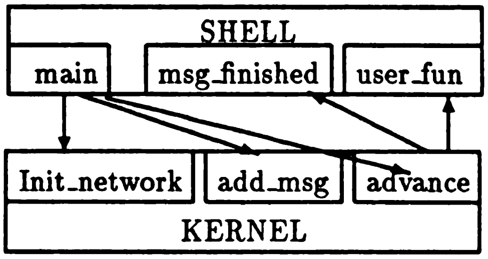


Figure 5.4: Interface between shell and kernel

graph). The sequence is then repeated[2].

In our shell, the status of a PE is maintained in a structure called MESSAGE_HISTORY, whose fields are

start_time : time at which first message was sent.

next_message : time at which next message is to be generated.

no_messages_completed : number of messages completed so far.

completion_time : time at which the n^{th} message was completed.

Various functions of the simulator shell set and use MESSAGE_HISTORY to simulate the communication behavior of a P.E. The functions implemented in shell are described now.

1. init_msgs : This function is called in the beginning of a simulation run. The function schedules a message from each P.E by setting next_message field to a value chosen randomly from the interval 0.. $2t_{avg}$. The function also initializes other members of the MESSAGE_HISTORY structure for each P.E.

2. msg_finished : This function is called by the simulation kernel whenever a message is completed. This function will increment the count of messages completed (no_messages_completed field) for the source P.E, and will schedule another message from the source after a time randomly chosen from the interval 0..$2t_{avg}$. This randomly chosen time simulates the time needed for local computation by the P.E before sending another message. The function will also record the completion time, if the message just transmitted was the n^{th} message from the source node.

3. user_fn : This function is called at the beginning of each simulation cycle, and must be efficiently implemented. In this shell, all active MESSAGE_HISTORY structures are linked by a linked-list sorted in ascending order of next_message field value. This function will examine the head of this list and add the required number of messages to the network by calling the kernel function add_message.

The functions init_msgs and msg_finished together with user_fn will simulate the message injections by a node. The destination of these messages has to be decided

[2] Note that a node will also be receiving messages from neighbors asynchronously.

based on the knowledge of the process graph and the current mapping. The knowledge is set and used by calling the following two functions.

4. **map** : maps the parallel tasks in a requested way, and remembers the mapping for later use. The function is called before a simulation run begins.

5. **find_dstn** : chooses the destination node given the source node, and is called whenever a new message is to be added to the network. The choice is made so that each parallel task communicates with all its neighbors uniformly. The function uses the knowledge of current mapping and the characteristics of the process graph.

The **main** function is shown below. It will call initializing functions including **init_msgs** and the **map**. After initialization, the **main** function enters a loop calling **advance** in each iteration to advance the simulation clock by 1 until all nodes complete n messages. The command line arguments specify parameters for a simulation run: the process graph, message length, average computation time per loop, system size, and number of messages to be completed by each node.

In our simulations, nodes continue to generate messages even if they have finished n messages. This continued generation maintains the traffic levels in the network and avoids any tail-effect. This strategy also makes a small value of n (100 in our experiments) sufficient. A small value of n in turn reduces the time taken by a simulation run.

After a simulation run is completed, the function **statistics** will scan the **MESSAGE_HISTORY** structures and compute the time taken by each node to successfully send n messages. The difference in times computed for two different mappings will then show the effect of mapping on the network performance. Note that even though parallel tasks are communicating in exactly the same way, a mapping changes the way the system nodes communicate. The change in communication pattern affects the traffic and contention, and hence the time to complete a message.

```
procedure main()
begin
  prcs_args()  ;/* Process command line arguments */
  init_kernel();/* Initialize kernel */
  map() ;      ;/* Map the process graph in a required way */
  init_shell() ;/* Initialize shell and a message on each node*/
  simclock = 0 ;/* Initialize simulation clock */
  cnt = 0      ;/* No of nodes that completed 'n' messages */
  do
    advance() ;
  while(cnt < N) ;/* Advance simulation time till all
                      nodes complete 'n' messages */
  statistics() ;/* Process results and compute statistics */
end
```

Figure 5.5: Outline of the simulation main function

5.4 Simulator Validation

One of the major challenges in developing a simulator is to validate it after it is implemented. We validate our simulator by comparing its output with the results from experiments on a real system, and by comparison with the results from theoretical analysis. We use the Matrix-transpose pattern (Section 4.2.3) for validation. Figure 5.6 shows the network performance for N=144, i.e. a 12×12 mesh. The legends in Figure 5.6 are from experiments on a Symult 2010 (Figure 4.12). We find that the simulation results match well with the experiments. The curves match even though simulator used 50 byte messages, and the experiments on the Symult used 1K, 4K or 8K bytes. The figure indicates that the node-traffic tends to saturate as predicted by theoretical results.

From theoretical analysis in the previous chapter, we also know that the saturation node traffic for a worst-case node is $\lambda^{w}_{n_{sat}} = 1/(\nu_{max}+1)$ as $\delta_{avg} = \delta_{max} = 1$ for the matrix-transpose pattern (Any node communicates with only one other node). For a 144 node ($\sqrt{N} = 12$) system, $\nu_{max} = \sqrt{N} - 2 = 10$, and hence $\lambda^{w}_{n_{sat}} = 1/11 \approx 9.1\%$.

The saturation node traffic for an average node will be $1/(1+\nu_{avg})$. To compute ν_{avg}, notice that ν value for any path originating from a node (i,j) below the principal diagonal (i.e. $i > j$) is $i - 1$, and is also equal the ν for the path originating from node $(\sqrt{N} - 1 - i, \sqrt{N} - 1 - i)$. Hence, considering the average over the paths originating from the nodes below the principal diagonal, we have

$$\begin{aligned}
\nu_{avg} &= \frac{1}{\sum_{i=1}^{\sqrt{N}-1}\sum_{j=0}^{i-1} 1} \sum_{i=1}^{\sqrt{N}-1}\sum_{j=1}^{i-1} \nu(p_{ij}) \\
&\quad \text{where } p_{ij} \text{ denotes a path originating from node}(i,j) \\
&= \frac{1}{\sqrt{N}(\sqrt{N}-1)} \sum_{i=1}^{\sqrt{N}-1}\sum_{j=0}^{i-1}(i-1) \quad \text{as} \quad \nu(p_{ij}) \;=\; i-1 \\
&= \frac{2(\sqrt{N}-2)}{3} \\
&= \frac{20}{3} \qquad (5.1)
\end{aligned}$$

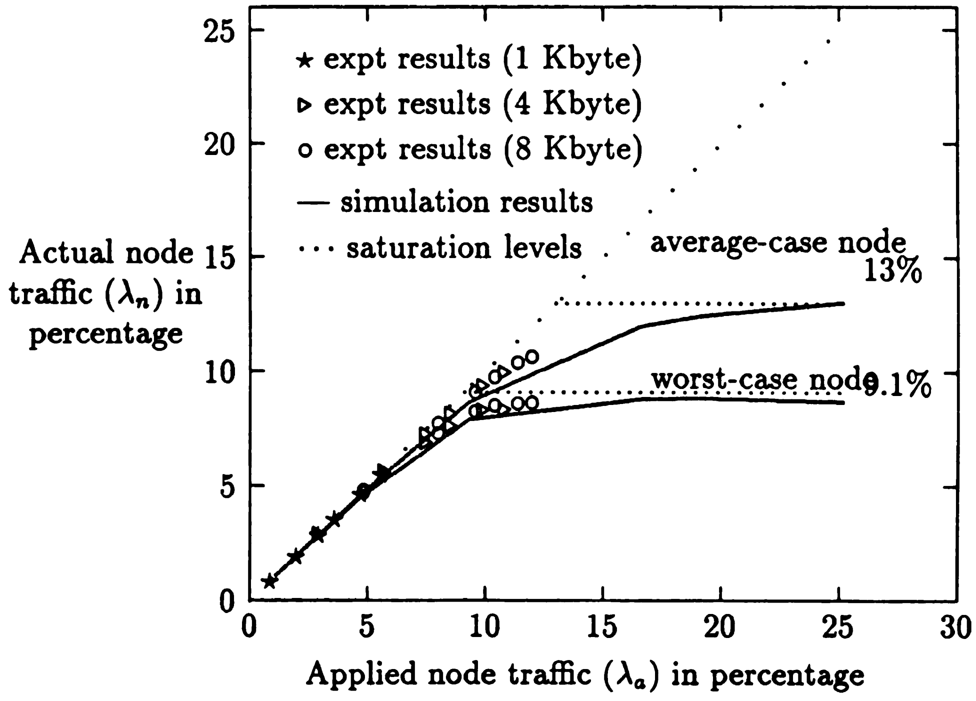


Figure 5.6: Matrix-transpose pattern : simulation results

The saturation node traffic for an average node is $1/(\nu_{avg} + 1) = 1/(20/3 + 1) =$

3/23 ≈ 0.13 or 13%. Figure 5.6 confirms that simulator results agree with the saturation node traffics predicted by theoretical results—further validating the simulator. We verify in Section 5.6 that the saturation levels indicated by the simulation results match with that predicted by theory even for other communication patterns (process graphs).

5.5 Simulator Performance

One criterion for judging the quality of a simulator is its execution performance—how long will it take to generate the desired results. The time for a simulation run depends on the message generating rate and the system size. We measured the simulation rate by running the simulator on Sun SPARC 1 workstations. Figure 5.7 shows the simulator performance as a function of applied node traffic for different system sizes. The Y-axis represents how many cycles can be simulated per CPU second, and the X-axis is the applied node traffic as defined in Section 5.4. Performance curves are presented for two different communication patterns in Figure 5.7. In one pattern, each node only communicates with its four neighbors so there is no contention for network channels (solid line). The second pattern is the uniformly distributed pattern in which each node communicates to every other node uniformly (dotted line). In this case messages need to compete for channels. In fact, simulation run times are approximately the same irrespective of the exact communication pattern used. The results show that the simulation rates are quite high. The simulation rate when there is significant contention is slightly lower due to additional work as messages are blocked and released.

We can observe from our simulator design that the time per simulation cycle depends essentially on the number of unblocked messages in the network in that cycle. Given the system size N and the node traffic λ_n, the number of unblocked messages in the system at any instant will be $N \times \lambda_n$. Hence, another useful way to display the performance of the simulator is shown in Figure 5.8. The Y-axis represents how many micro seconds of CPU time will be needed to process one, unblocked message

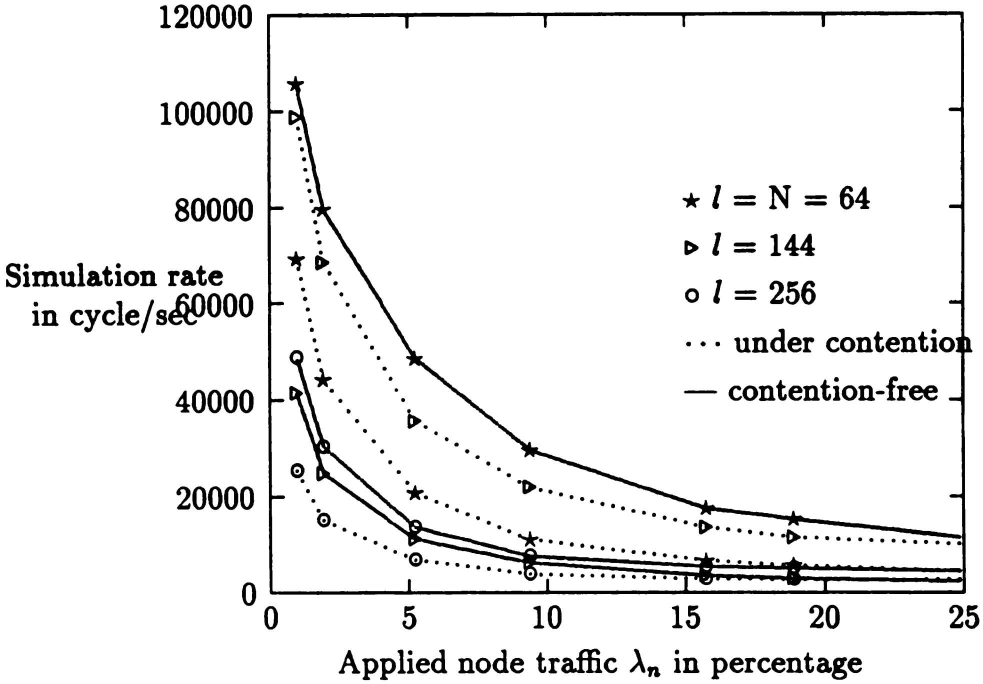


Figure 5.7: The simulator performance in cycles per CPU second

per simulation cycle. The X-axis is still the applied node traffic. We observe that the CPU time needed is approximately a constant; about 10–15 micro seconds to process one message per simulation cycle. The CPU time is independent of the system size and the communication pattern.

We can also estimate the time needed for a simulation run by using the curves in Figure 5.8. Let m be the number of messages that need to be sent by each node in a simulation run. As a first step approximation we assume $\lambda_n = \lambda_a = \frac{L}{t+L}$ (Section 3.1). So, we have $L = \frac{t\times\lambda_a}{1-\lambda_a}$. If the simulator processes events at the rate of x μsecs per message per cycle (as shown in Figure 5.8), the execution time for m simulation loops will be:

$$\begin{aligned} \text{CPUtime} &= x \times \text{number\_of\_messages\_in\_the\_system\_per\_cycle} \\ &\quad \times \text{number\_of\_cycles\_per\_loop} \times \text{number\_of\_loops} \\ &= x \times (N \times \lambda_a) \times (t + \frac{t \times \lambda_a}{1-\lambda_a}) \times m \end{aligned}$$

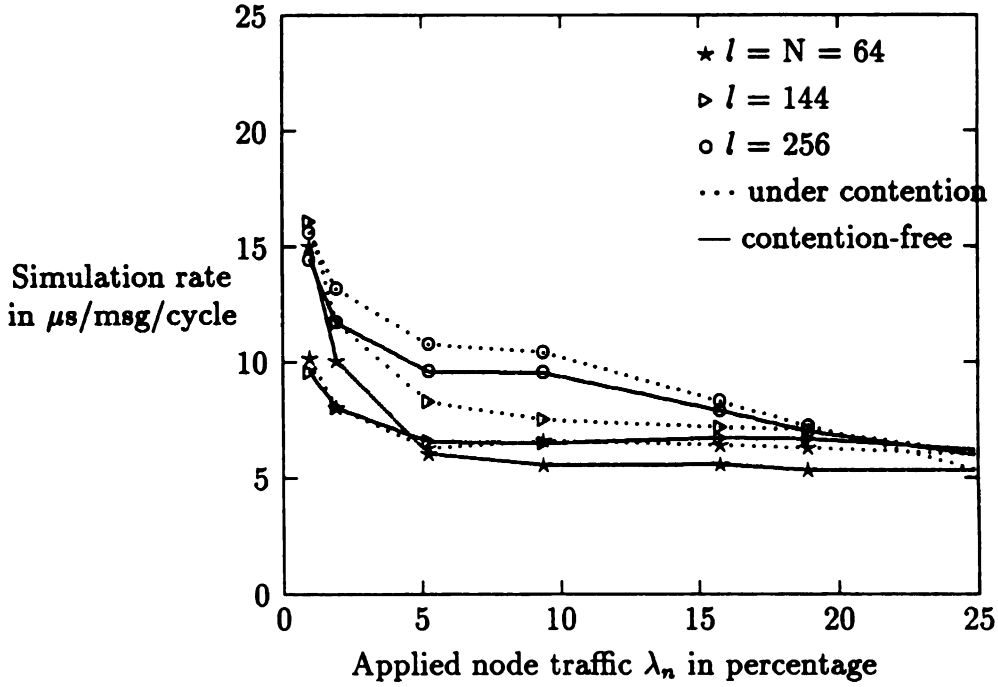


Figure 5.8: The simulator performance in micro seconds per message per cycle

For example, consider simulating a network with 256 nodes ($N = 256$), and each node completing 100 messages per simulation run ($n = 100$). Also, assume that $t = 270$ cycles, and $\lambda_n = 10\%$ ($l = 30$). From Figure 5.8 we find that the simulator is processing at a rate of $x \approx 15\mu$secs/message/cycle. Hence, we can estimate the time for the simulation run to be

$$15 \times (256 \times 0.1) \times (300 + \frac{300 \times 0.1}{1 - 0.1}) \times 100 \approx 13 \textit{ seconds}$$

which is very efficient. The simulation time is directly proportional to N if all other parameters remain the same. Hence, in the above example if instead of a 256 node network, we wanted to simulate a 1024 node system, we would need 13×4 = 52 secs to complete the simulation run. We find that our simulator performs well and can simulate even large networks in reasonable time.

5.6 Simulation Experiments

The simulator formed by combining the simulator shell with the kernel can be a very useful tool in understanding the network performance under the effects of contention for network channels. In this section, we describe some of the experiments we conducted using the simulator to better understand the network performance. The experiments show the flexibility of our simulator and the ease with which we can use the simulator to answer various questions about the network performance. In all cases, we present the performance results as a plot of actual node traffic vs. applied node traffic (refer to Section 4.3). We also indicate the saturation node traffic predicted by our theoretical results to demonstrate that our theoretical results agree with the simulator results.

5.6.1 Small vs. Large Messages

Figure 5.9 shows the results for for three different message lengths of 5, 50 and 250 flits, when the entire message is sent as a single packet. The saturation levels for 50 and 250 flits are almost identical, and match the theoretical results. We conclude that if message lengths are greater than path lengths, the saturation effect is independent of absolute message length. For small message lengths, the saturation occurs at a slightly lower value.

5.6.2 Constant vs. Probabilistic Computation Times

Another important characteristic of our experiments is the distribution for computation times of various loops. We tried two distributions. In one we had the same computation time in each loop equal to t_{avg}, and in the other, computation time was distributed uniformly over the interval $0..2t_{avg}$. The average computation time was equal to t_{avg} in both the experiments, and hence applied node traffic was same. The results are given in Figure 5.10. We observe that the performance is almost identical. The performance is slightly better for constant computation times, and the worst case node shows a sharper transition in performance when applied node traffic is near saturation level.

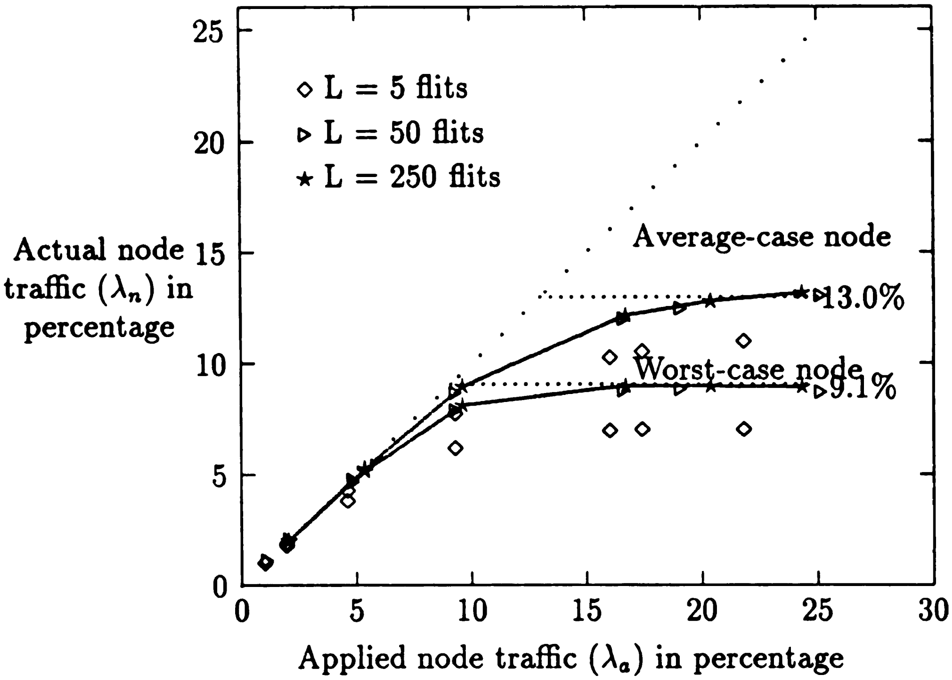


Figure 5.9: Effect of message length

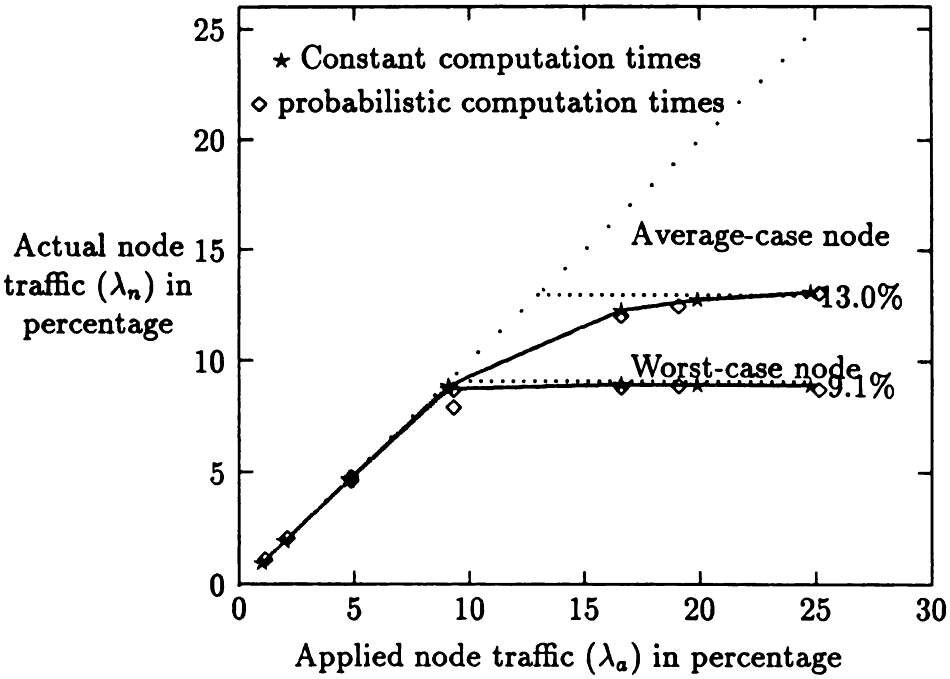


Figure 5.10: Effect of distribution on computation times

5.6.3 Packetized vs. Unpacketized Messages

The effect of packetization on performance was measured next. We sent 250-flit messages as ten packets of 25 flits each, with a inter-packet interval of 10 cycles to account for the processing overhead per packet. The results were then compared with that obtained in the first experiment when 250-flit messages were sent as a single message (Figure 5.11). We observe that the performance once again is almost identical. Experiments also revealed that neither processing overheads per packet (which determines time between packets) nor packet size has a significant effect on the saturation level. However, if the average length of the messages generated by various nodes are different, then packetization may be useful. Packetization will prevent a long message from blocking a small message for a long time, and hence the network response will be fair to different messages.

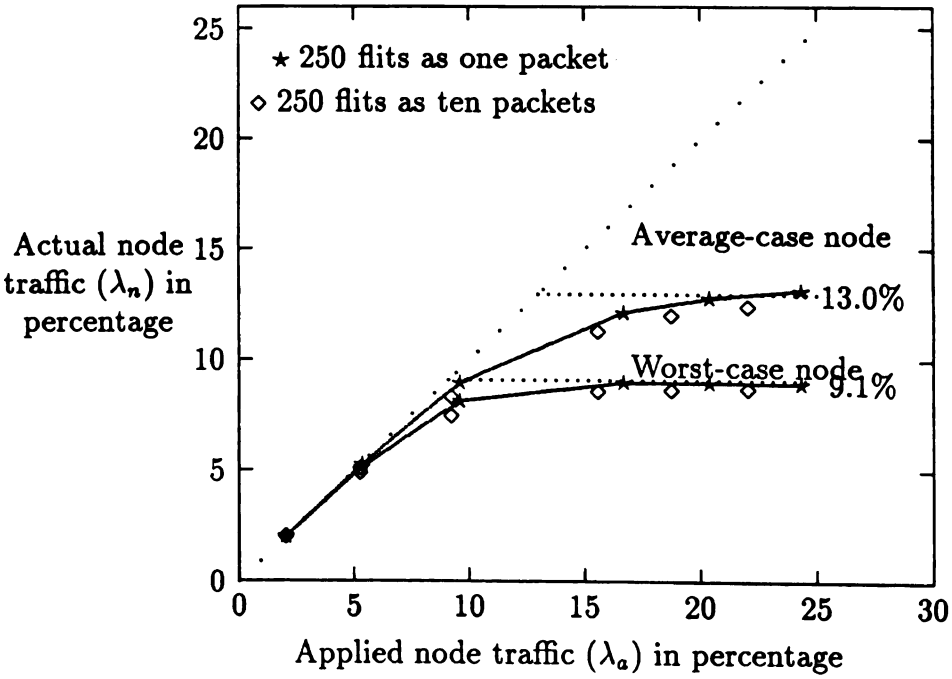


Figure 5.11: Effect of packetization

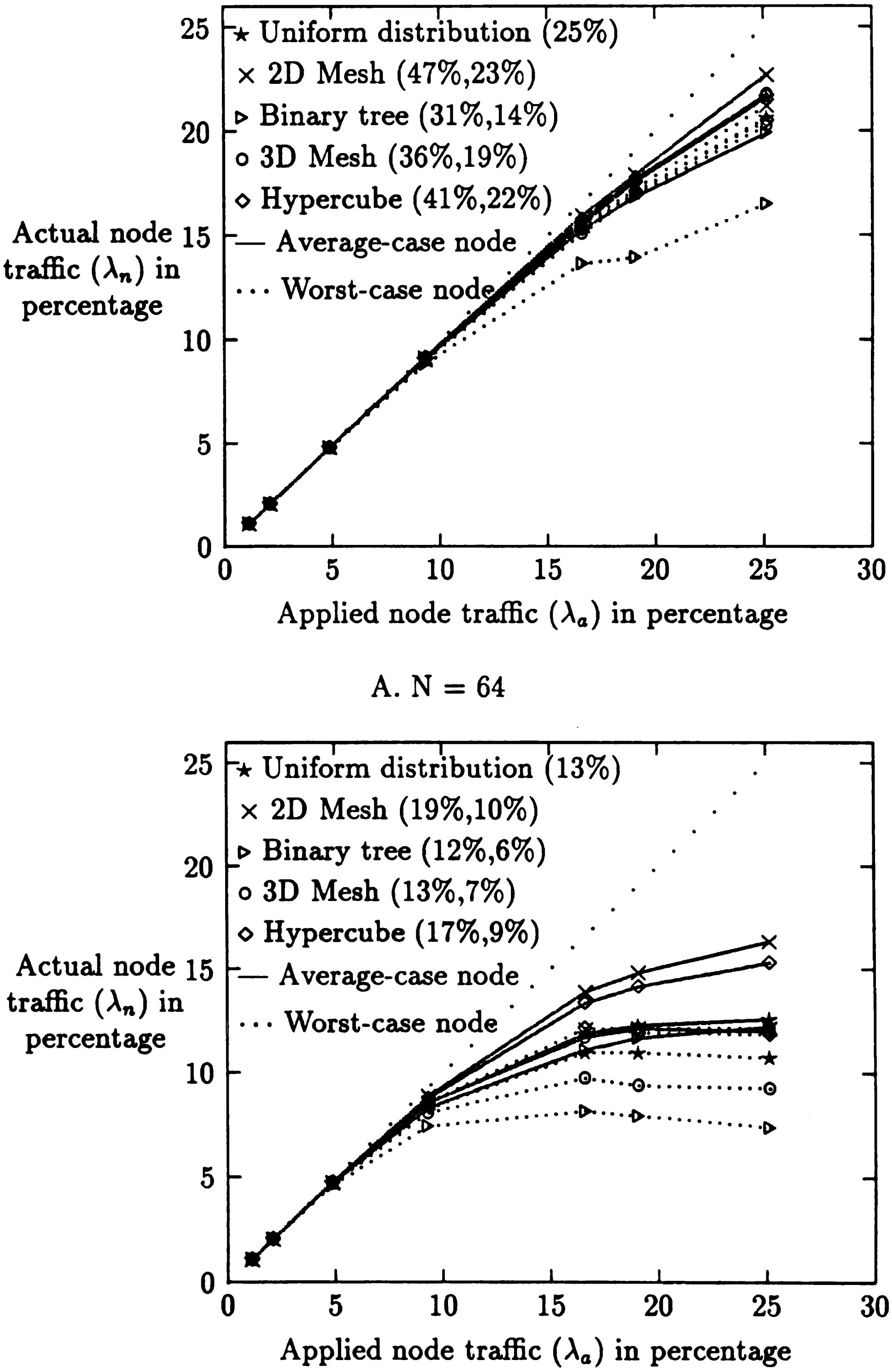
These results indicate that none of the factors—absolute message length, packetization, or distribution of computation times—affect the saturation levels. We therefore use 50 byte, single packet messages and uniformly distributed computation times in the remaining experiments.

5.6.4 Mapping

We now examine the effect of random mapping on the network performance for four common process graphs: binary tree, 2D mesh, 3D mesh and hypercube. We also present the results for a uniform distribution which acts as a reference for other results, and shows the effect of non-uniformity in traffic due to random mapping. A uniform distribution was realized by selecting the destination node randomly each time a message was generated. Figure 5.12 give the actual node traffic for different communication patterns for two system sizes of 64 nodes and 256 nodes. We observe that the actual traffic tends to saturate at different levels, depending on the communication patterns, and that the results match the saturation levels predicted by theoretical analysis in the previous two chapters.

Compare Figure 5.12.A and Figure 5.12.B. We find that saturation occurs at a lower level for larger system sizes—less than 10% for systems having 256 node or more. Hence, random mapping may not be advisable for multicomputers having hundreds of nodes if the applied node traffic of an applications is 5% or higher. As mentioned in the previous section, overheads in current multicomputers are limiting the applied node traffic to 13% or less (6% or less for small messages). With reduced overheads in future machines, higher applied node traffic are possible depending on the application. Hence, the effect of contention on performance will be increasingly visible in future multicomputers if we use random mapping. We also observe that the saturation level for the worst-case node is about 2-3 times lower than the level for an average node. The saturation levels predicted by our theoretical analysis are indicated in parenthesis in the graph's legends (average case, worst case). We find that theoretical predictions match the simulation results further validating our simulator. The actual saturation levels for the worst case are about 20% higher than the value predicted by the theoretical analysis.

One solution to the saturation problem is to use careful mappings that reduce contention. Simulations showed that for careful mappings actual node traffic was almost equal to applied node traffic for all our patterns even for a 25% applied node traffic. Hence, the mapping problem still exists, and mappings that minimize contention may

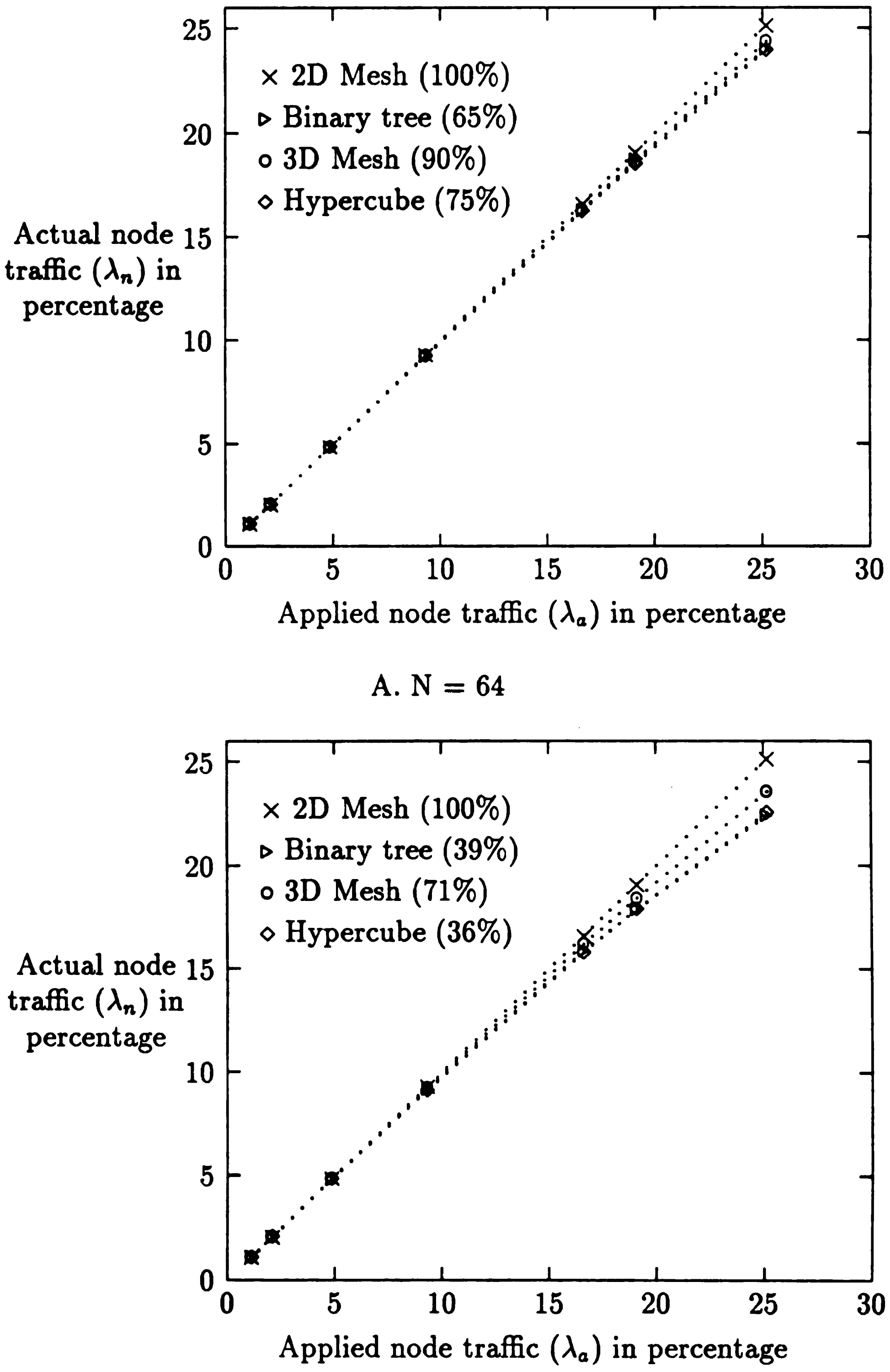


A. N = 64

B. N = 256

(The pair of values in parenthesis gives the theoretical results)

Figure 5.12: Effect of random mapping on the network performance.



A. N = 64

B. N = 256

(The plots are shown only for the worst-case node)

Figure 5.13: Effect of careful mapping on the network performance.

still make a significant difference in the performance of large multicomputers.

The results confirm our fear that the problem of contention will be increasingly visible in future, larger multicomputers. Random mapping may not be advisable and careful mapping may improve the performance significantly.

5.6.5 Arbitration Policy

The previous experiments considered a software solution—mapping. One possible hardware improvement is to find a good queueing policy when message headers on different input ports request an output port. In our simulator, we time-stamped messages when they entered the network and gave priority to messages with earlier time stamps while granting output ports. Intuitively it seems to be the best policy as the network attempts to complete older messages and minimizes the time spent waiting for messages by parallel tasks. We investigate performance of other queueing policies to determine if this is the best policy. For each policy we evaluate the performance for the four process graphs used in Experiment 2.

A simple queueing policy is FIFO with ties resolved by giving a fixed priority to input channels. Figure 5.14 shows that performance is significantly worse, and λ_n actually declines when applied node traffic is increased beyond a level. Hence, we need a better queue policy than the FIFO. FIFO can be improved by using a tie-resolution scheme that gives priority to an input port that least-recently requested the output port. However, there is little performance improvement.

An effective policy is to discourage message injections by the local node when the router senses heavy network traffic passing through the node. Different input ports are given different base priorities—the port with a lower priority is granted the output port. To prevent infinite waits, each time an input port fails to get an output port its priority is reduced by 1, and when it succeeds its priority is set to its base value. We call this queue policy *biased priority policy.* Figure 5.15 shows the performance when the bias for the input port for the channel connected to the local node was 9, and the other ports were 4. The performance is comparable to that of the time-stamped priority policy. The advantage of the biased priority policy is that it is simple and

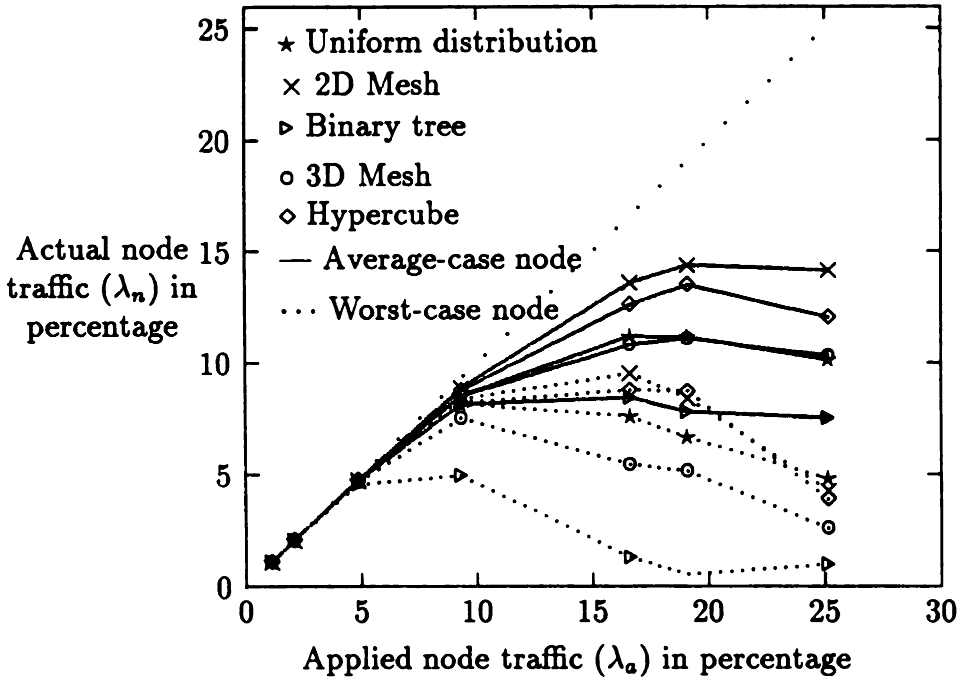


Figure 5.14: FIFO queue policy, 256 node system

does not require time stamp information in messages.

Another queue policy whose performance is comparable to the time-stamped priority is the *message-source priority*. If the message header contains the source node number, this policy can be easily implemented. Each output port remembers the source of the last message that used the port and acts as a clock hand. The message whose source node number is nearest to the clock hand gets the port next. The performance for such a policy seems to be quite similar to the time-stamped priority policy (Figure 5.16). We conclude that the time-stamped priority policy seems to be the best of all the policies considered, and that the performance of other simpler policies such as biased priority policy and message source priority policy are comparable.

The experiments and the results clearly demonstrate that our simulator can be a useful tool in understanding the performance of modern wormhole-routed networks. Our results indicate that the contention problem is significant for large networks, and that the effect depends on the applied node traffic (a measure of the communication intensity of an application) for a given communication pattern and mapping. Careful

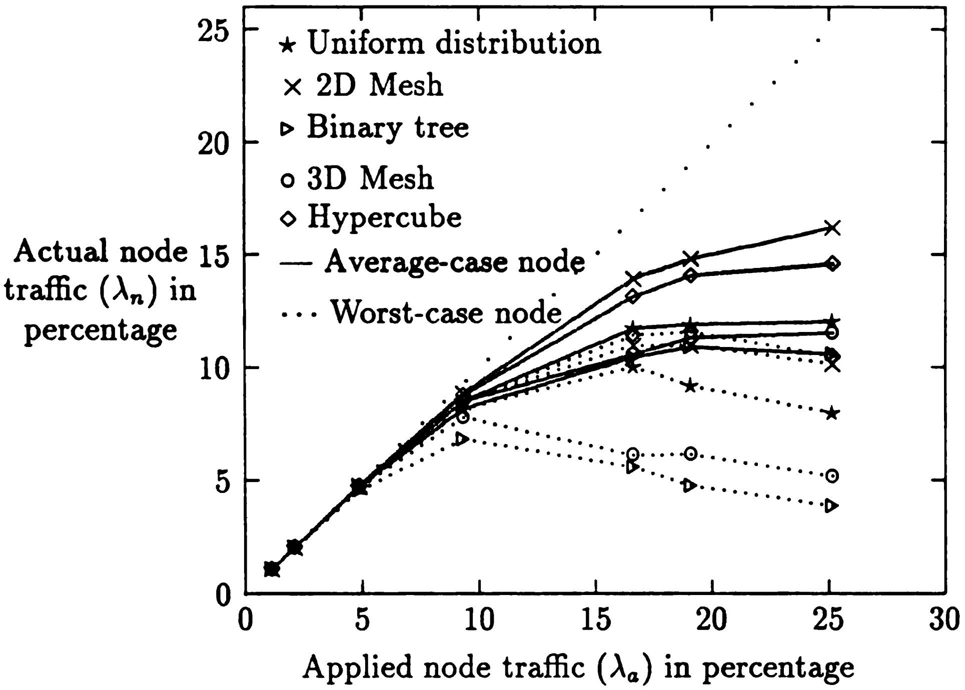


Figure 5.15: Biased priority queue policy, 256 node system

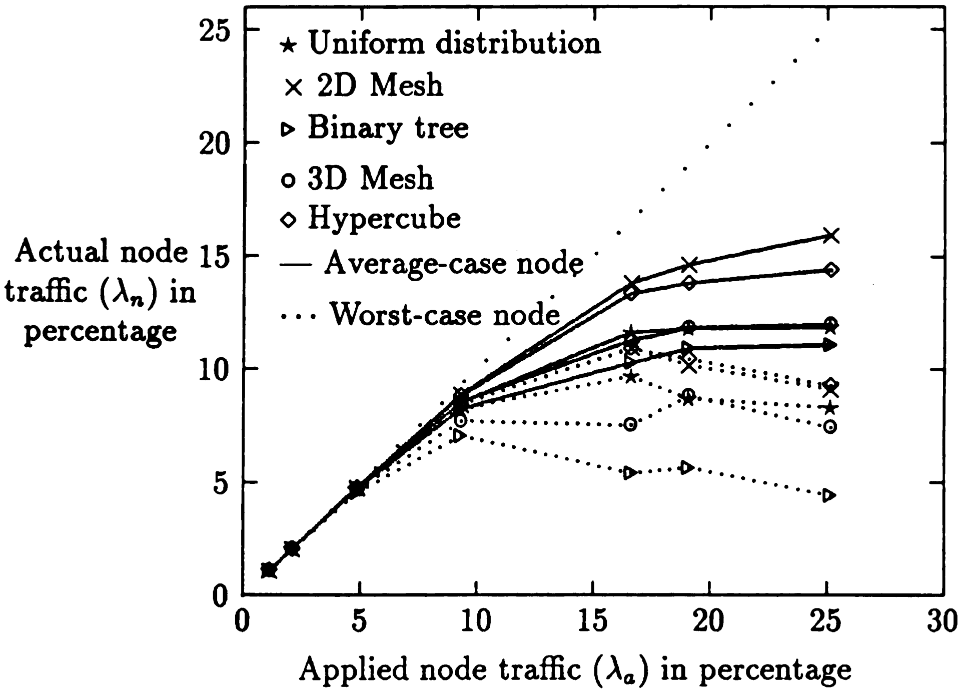


Figure 5.16: Message-source priority queue policy, 256 node system

mappings seem to be the main solution to the contention problem even though system level solutions such as appropriate queueing policies and adaptive routing can help.

5.7 Conclusion

We described a high-performance simulator for wormhole-routed interprocessor communication networks. A careful design and implementation in a high-level language resulted in a simulator that can simulate large networks in reasonable amount of time. We demonstrated that the simulator can be a useful tool in understanding the effect of contention on network performance. The simulator was validated by comparing its outputs with the results from experiments on a real machine, and the results from theory. The design clearly shows that our simulator is quite flexible, and can be easily modified to simulate any network topology, routing scheme or even circuit-switched networks.

The simulator performance can be improved further. The performance for large messages, and longer computation time per loop is not good. It is possible to improve the performance by recognizing the messages that have successfully established their paths, and avoid processing them until the tail flit leaves the source. The simulator was not designed to study the architecture of the router or node in detail. It is not useful in evaluating the bottlenecks that may be present in the router or the limitations of the node or the PE. Some recent network implementations support virtual channels that do not consume any channel bandwidth if idle. Our simulator cannot simulate such virtual channels at present.

Chapter 6

CONCLUSION AND FUTURE RESEARCH

In this thesis, we studied the performance of direct interprocessor communication networks—a critical component of message-passing distributed-memory concurrent computers. The main objective was to model and improve the performance of large direct networks that use wormhole-routing, and to determine if network performance can limit the overall performance of a multicomputer. Results showed that rapid advances in the technology of direct networks have made their performance characteristics quite different from the characteristics of the earlier networks. The results also showed that the performance of direct networks is now comparable to that of other networks. Our studies involved

- theoretical analysis to model and predict the communication performance, and to study some design tradeoffs. The analysis was in two parts. First, we studied the simpler case of contention-free communication, i.e. we assumed that there was no contention for network channels. Next, we modeled the performance under contention for a given communication pattern and mapping.

- experimental studies to evaluate the performance of a real multicomputer that uses a state-of-the-art communication network. Our experimental results provided an understanding of the performance characteristics of current networks, and helped to verify theoretical results.

- simulation studies to evaluate the performance of large future multicomputers. The simulator was validated by comparing its results with those from theoretical analysis and experiments. Simulations were also used to investigate the effect of several design factors on performance that are difficult to model theoretically.

We summarize the main contribution of our thesis now.

6.1 Summary and Major Contributions

The communication performance under contention-free conditions is determined by the bandwidth of the network channels and the switching technique. The bandwidth depends on the width of the channels and the rate of communication over these channels. Since large networks are wire-limited, the choice for network topology affects the width of network channels. Topology also affects the length of the communication wires, and the length of wires determine the maximum rate of communication. We estimated the improvement in channel bandwidths if we use 2D mesh networks instead of the popular hypercube networks. Our results showed that channel widths improve by a factor $R_w = 2\sqrt{N}/3$ and that channel rates improve by a factor, R_r, in the range $1..\sqrt{2N}/\lg N$, depending on the dependency of the channel rate on the length of a channel. Hence, overall bandwidths improve by a factor $R_b = R_w \times R_r$. Calculations showed that $R_b \approx 100$ for a 1024 node system, i.e. bandwidths of mesh networks could be two orders of magnitude higher than that of hypercube networks for a 1024 node system. Current networks use wormhole routing which is efficient for non-adjacent node communication. Considering the characteristics of wormhole-routing, we showed that higher bandwidth channels improved performance significantly even for non-adjacent node communication, in spite of longer paths in mesh networks. In short, we concluded that a mesh topology was a better choice than a hypercube because we get a significant improvement in performance under contention-free conditions.

However, in large multicomputers that use mesh networks, we showed that the effect of contention can be significant unless carefully reduced. Contention for network channels results in additional wait times which reduces effective communication rates to only a fraction of the channel bandwidths. Contention depends on mapping—the way parallel tasks are assigned to various multicomputer nodes. We developed a framework to model the effect of contention on the communication performance for a given application and mapping. For a given mapping, we introduced a new metric, ***path contention level***, to measure the contention for a specific communication path. Path contention levels for various paths together with channel loads for various network channels can characterize the nonuniformities in traffic. We used queuing theory to estimate and model the performance under contention as a function of path contention levels. Our results showed that the level of message traffic that can be injected by a node is bounded, and that this bound can be reasonably estimated if we know the path contention levels. A simple tool was developed based on our theoretical models to compute the bounds on node traffic for a given mapping.

We used the general results for performance under contention to study the special case of random mapping. The objective was to determine if random mapping was good enough, that is, to determine if the mapping problem no longer exists. We used simple probability theory to estimate the channel load and path contention levels as a function of system size. These estimates were verified by actual values computed for different popular communication patterns. We then used those estimates for path contention levels in our general results to predict the performance under random mapping. Our results showed that message injection rates, measured by node traffic, cannot exceed $2/\sqrt{N}$ for an average node. Some nodes are affected more than others, and the results also showed that bounds on node traffic for the worst case nodes can be lower than that for an average node by a factor of two to three. We concluded that careful mapping is necessary to have effective communication rates close to the channel bandwidths, and hence retain the advantage of higher bandwidth channels of mesh networks. We illustrated in case of several common communication patterns how careful mappings can reduce contention and increase bounds on node traffic. Our results clearly showed that the path contention level can be a useful metric as it can reasonably predict the communication performance under contention. Minimizing path contention levels can replace minimizing path lengths as the mapping objective.

Experiments were conducted on a Symult 2010—a real multicomputer that uses a state-of-the-art wormhole-routed mesh-connected network. The objective of the experiments was to understand the performance characteristics of current networks, and to verify theoretical results. Experimental results showed that contention-free communication can be modeled as a linear function of message length, and that it is independent of path length. The node traffic was limited to 6% if message lengths were less than 1 Kb, and less than 13% even for a 8K message. The reason was high, fixed overhead and packetization overhead. Limited injection rates and small system sizes made the effect of contention negligible, and made the performance independent of mapping for current systems. However, on the largest system which had 192 nodes was the effect of contention visible, and the performance curves matched well with theoretical results. In the future, overheads are being reduced and system sizes are increasing. We, therefore, concluded as predicted by theoretical results that the effect of contention would no longer be negligible. Mapping can make a difference in communication performance of multicomputers. We also showed that architectural bottlenecks exist inside a node in current systems.

A high-performance simulator was developed in C to study the communication performance of large multicomputers not yet available. A careful design resulted in a simulator that can reasonably model the performance of wormhole-routed or circuit-switched networks. At the same time, the simulator is fast enough to provide results for large multicomputers having hundreds of nodes in tens of seconds on SPARCstations. The simulator is also flexible as it can be easily modified to simulate any network topology, routing scheme or application characteristics under specified mappings.

The simulator was used to verify the theoretical results for large multicomputers and/or multicomputers which can have high message injection rates. Simulations show that performance is as predicted by the theory. The results also match with experimental results over the operation range of current multicomputers. Results showed that performance is a function of node traffic, and is relatively independent of message lengths, packet lengths, interpacket time and so on provided that the average length of messages generated by various nodes is approximately the same. We also used the simulator to study system-level solutions to the contention problem. One such solution is an appropriate arbitration policy to resolve contention. Current networks use FIFO policy to resolve contention. Simulations show that performance deteriorates fast in such networks once traffic reaches a certain threshold. Timestamping the messages when they entered the network, and giving priority to 'old' messages provides the best performance.

6.2 Future Research

The results of the thesis indicate that contention for network channels will be a growing concern in future, larger multicomputers as its effect on network performance will be increasingly visible. In this thesis, we did some initial work to model and improve network performance under contention. Still, there are many interesting problems that need to be studied. Some of them are

1. Automated Mapping : In this thesis, we gave careful mappings that minimized contention (measured by path contention level) only for a few common communication patterns. It is useful to have a tool that can find out such a mapping for any given communication pattern specified as a process graph. It is useful if the tool can accept process graphs with weighted edges to account for nonuniformities in message intensity along different edges. Since finding such a mapping is a hard problem, such a tool may use heuristics. A heuristic algorithm is sufficient as it is only necessary to find a mapping that results in bounds that are higher than the node traffic applied by the application, and there is no need to find the best mapping. The tool may also have to consider routing possibilities in addition to placement, as future systems may provide adaptive or user-defined routing schemes.

2. System traffic : We considered only the user-generated traffic due to one application. In a general system, the traffic from several users interfere with one another. There is also system-generated traffic which interferes with user-traffic. Some examples of system-traffic are I/O traffic, operating system messages, etc. The characteristics of such traffic may be different from that of user-generated traffic. For example, message lengths may be greater than user-generated messages. One can study the communication performance at this higher level considering the interference between

different traffic using the framework we developed. Such a study may help in making policy decisions such as the allocation of nodes to a parallel task and the position of I/O nodes.

3. Fault tolerance : In a fault-tolerant system, the routing scheme may change under faults. A different routing may in turn change contention levels, non-uniformities and communication performance. It could be interesting to study how the communication performance may change under faults, and whether we can extend our theoretical results to those cases.

4. System-level solutions : We studied one system-level solution to the contention problem—an appropriate arbitration policy. There can be other solutions such as adaptive routing, whose effectiveness is still an open question. Another solution is virtual channels. The improvements in communication performance provided by these system level solutions need to be studied. It is not clear if these solutions can allow random mapping to be sufficient. It would be interesting to see if we can model the communication performance for a mapping considering adaptive routing and virtual channels.

5. Generalized communication models : In our thesis, we made assumptions about the communication characteristics of an application to simplify the model. It would be interesting to see if we can extend our theoretical results to more general cases. For example, a task may not communicate uniformly with its peers. We can handle such a case by assigning weights to the edges of a process graph and take that into account in our analysis. In addition, the messages generated may be bursty in many applications. The message generation at a node may depend on the messages received by a node, i.e. path traffic on one path may depend on another. Multicast and broadcast communications may occur which need to be accounted for. The process graphs itself may not be static and may change during the execution. It should be possible to extend our framework to most of these cases.

To summarize, old problems such as path lengths may no longer exist due to advances in network technology. However, the communication network is a critical architectural component, and the demand on the network performance continues to grow. As a result, we face new problems, new issues, and we require new solutions to be able to continue to improve the communication performance of multicomputers.

Bibliography

Bibliography

[1] Butterfly GP1000, section 5, 1988.

[2] D. P. Agarwal, V. K. Janakiram, and G. Pathak. Evaluating the performance of multicomputer configurations. In *IEEE Computer*, pages 23–37, May 1986.

[3] M. Arango, H. Badr, and D. Gelerenter. Staged circuit switching. *IEEE Transactions on Computers*, pages 174–179, Feb. 1985.

[4] S. Arlauskas. iPSC/2 system: a second generation hypercube. In G. C. Fox, editor, *Proc. 3rd Conference on Hypercube Concurrent Computers and Applications*, pages 38–42. ACM, 1988.

[5] W. C. Athas and C. L. Seitz. Multicomputers: Message passing concurrent computers. *Computer*, pages 9–25, Aug. 1988.

[6] F. Berman and L. Snyder. On mapping parallel algorithms into parallel architectures. *Journal of Parallel and Distributed Computing*, pages 439–458, 1987.

[7] L. N. Bhuyan and D. P. Agarwal. Generalized hypercube and hyperbus structures for a computer networks. In *IEEE Transactions on Computers*, Apr. 1984.

[8] R. P. Bianchini and J. P. Shen. Interprocessor traffic scheduling algorithm for multiple-processor networks. *IEEE Transactions on Computers*, pages 396–409, Apr. 1987.

[9] S. H. Bokhari. On the mapping problem. In *IEEE Transactions On Computers*, pages 207–214, Mar. 1981.

[10] S. H. Bokhari. Communication overhead on the intel ipsc-860 hypercube. Technical report, ICASE, NASA Langley Research Center, May 1990.

[11] S. H. Bokhari. A network flow model for load balancing in circuit-switched multicomputers. Technical report, ICASE, NASA Langley Research Center, May 1990.

[12] S. W. Bollinger and S. F. Midkiff. Processor and link assignment in multicomputers using simulated annealing. In *International Conference on Parallel Processing*, pages 1–7, 1985.

[13] L. Bomans and D. Roose. Benchmarking the iPSC/2 hypercube multiprocessor. *Concurrency: Practice and Experience*, pages 3–18, Sept. 1989.

[14] S. Borkar et al. iWarp: An integrated solution to high-speed parallel computing. In *Proc. Supercomputing Conferecne*, pages 330–339, 1988.

[15] G. D. Buzzard. High performance communication for hypercube multiprocessor. *Ph.D Thesis*, University of Michigan, Ann Arbor, 1988.

[16] D. A. Carlson. Modified mesh connected parallel computer. In *IEEE Transactions On Computers*, pages 1315–1321, Oct. 1988.

[17] S. Chittor and R. Enbody. Hypercubes vs. 2d meshes. Technical Report CPS-89-17, Computer Science, Michigan State University, 1989.

[18] S. Chittor and R. Enbody. Link switching : A communication architecture for configurable parallel systems. In *proc IEEE Phoenix Conference on Computers and Communication*, Mar. 1989.

[19] S. Chittor and R. Enbody. Hypercubes vs. 2D meshes. In *Proc. SIAM Fourth Annual Conference on Parallel Processing for Scientific Computing*, pages 313–318, 1990.

[20] S. Chittor and R. Enbody. Performance degradation in large wormhole-routed interprocessor communication networks. In *Proc. International Conference on Parallel Processing*, volume 1, pages 424–428, 1990.

[21] S. Chittor and R. Enbody. Performance evaluation of mesh-connected wormhole-routed networks for interprocessor communication in multicomputers. In *Proc. Supercomputing Conference*, pages 647–656, Nov. 1990.

[22] S. Chittor and R. Enbody. Predicting the effect of mapping on the communication performance of large multicomputes. Technical Report CPS-91, Computer Science Dept, Michigan State University, 1990.

[23] S. Chittor and R. Enbody. Will 2d meshes replace hypercubes ? In *PARBASE-90*, pages 317–319, 1990.

[24] S. Chittor and R. Enbody. Predicting the effect of mapping on the communication performance of large multicomputers. In *Proc. International Conference on Parallel Processing*, (to appear).

[25] S. Chittor, R. Enbody, and T.-S. Jou. A high performance simulator for large wormhole-routed networks. In *International Journal of Computer Simulation*, 1991 (to appear).

[26] E. Chow, H. Madan, J. Peterson, D. Grunwald, and D. Reed. Hyperswitch network for the hypercube computer. In *Proc. International Conference on Parallel Processing*, pages 90–99, 1988.

[27] D. V. Chudnovsky, G. V. Chudnovsky, and M. M. Denneau. Regular graphs with small diameter as models for interconnection networks. In *International Conference on Parallel Processing*, pages 232–239, 1987.

[28] P. Close. The iPSC/2 node architecture. In *Third Conference on Hypercube Concurrent Computers and Applications*, pages 43–51, 1988.

[29] W. Dally. Network and processor architecture for message-driven computers. In R. Suaya and G. Birtwistle, editors, *VLSI and Parallel Computation*, pages 140–222. Morgan Kaufmann Publishers, Inc., 1990.

[30] W. Dally et al. Architecture of a message-driven processor. In *Proc. 14th ACM/IEEE Symposium on Computer Architecture*, pages 189–196, June 1987.

[31] W. J. Dally. Wire efficient VLSI multiprocessor communication networks. In *Proc. Stanford Conference on Advanced Research in VLSI*, pages 391–415. MIT Press, Cambridge, Mar. 1987.

[32] W. J. Dally. Fine-grain message-passing concurrent computers. In *Proc. The third Conference on Hypercube Concurrent Computers and Applications*, 1988.

[33] W. J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 39(6):775–785, June 1990.

[34] W. J. Dally. Virtual-channel flow control. In *Proc. of the 1991 International Symposium on Computer Architecture*, pages 60–68, 1990.

[35] W. J. Dally and C. L. Seitz. The torus routing chip. *Journal of Distributed Systems*, 1:187–196, 1986.

[36] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, pages 547–553, May 1987.

[37] W. J. Dally and P. Y. Song. Design of a self-timed VLSI multicomputer communication controller. In *Proc. International Conference on Computer Design*, pages 230–234, 1987.

[38] S. P. Dandamudi and D. L. Eager. Hierarchical interconnection networks for multicomputer systems. *IEEE Transactions on Computers*, pages 786–797, June 1990.

[39] K. W. Doty. New designs for dense processor interconnection networks. *IEEE Transactions on Computers*, pages 447–450, May 1984.

[40] C. Flaig. VLSI mesh routing systems. Technical Report 5241, California Institue of Technology, May 1987.

[41] J. Gecsei. Interconnection networks from three-state cells. In *IEEE Transactions on Computers*, pages 705–711, Aug. 1977.

[42] J. Ghosh and K. Hwang. Hypernet : A communication efficient architecture for constructing massively parallel computers. In *IEEE Transactions On Computers*, pages 1450–1465, Dec. 1987.

[43] J. R. Goodman and C. H. Sequin. Hypertree: A multiprocessor interconnection topology. In *IEEE Transactions on Computers*, pages 923–933, Dec. 1981.

[44] M. Goodrich, C. Koelbel, W. Robinson, and K. Showell. Prep-p: A mapping preprocessor for chip computers. In *International Conference On Parallel Processing*, 1985.

[45] D. Gordon. Efficient embeddings of binary trees in VLSI arrays. *IEEE Transactions on Computers*, c-36(9):1009–1017, Sept. 1987.

[46] D. C. Grunwald and D. A. Reed. Networks for parallel processors: Measurements and prognostications. In *Proc. The third Conference on Hypercube Concurrent Computers and Applications*, pages 610–619, 1988.

[47] S. Hart. A note on the edges of the n-cube. *Discrete Mathematics*, pages 157–163, 1976.

[48] J. P. Hayes et al. Architecture of a hypercube supercomputer. *IEEE Micro*, pages 653–660, Oct. 1986.

[49] R. E. Horan and D. A. Poplawski. Communication/computation paradigms for hypercubes. In *Hypercube Computers Conference*, pages 620–623, 1988.

[50] E. Horowitz and A. Zorot. The binary tree as an interconnection network: Applications to multiprocessor systems and vlsi. In *IEEE Transactions On Computers*, 1981.

[51] K. Hwang and W. Briggs. *Parallel Processing*. McGraw Hills, 1985.

[52] T.-S. Jou and R. Enbody. Adaptive routing on wormhole-routed two-dimensional meshes. Technical Report CPS-90-03, Computer Science, Michigan State University, 1990.

[53] A. M. Jrad and R. W. Hall. Orthogonal fast channels: An enhanced mesh architecture. In *International Conference on Parallel Processing*, pages 828–831, 1987.

[54] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication networks. *Computer Networks*, 3:267–286, 1979.

[55] P. Kermani and L. Kleinrock. A tradeoff study of switching systems in computer communication networks. *IEEE Transactions on Computers*, c-29, Dec. 1980.

[56] T. Kerola and A. Hartmann. Operational analysis on hyper-rectangulars. In *International Conference on Parallel Processing*, pages 78–82, 1988.

[57] L. Kleinrock. *Queueing Systems : Computer Applications*, volume 2. John Wiley & Sons, 1976.

[58] X. Lin, P. M. Mckinley, and L. M. Ni. Performance evaluation of multicast wormhole routing in 2d-mesh multicomputers. Technical report, Computer Science Dept, Michigan State University, 1991.

[59] X. Lin and L. M. Ni. Deadlock-free multicast wormhole routing in multicomputer networks. In *Proc. of the 1991 International Symposium on Computer Architecture*, 1991.

[60] D. H. Linder and J. C. Harden. An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes. *IEEE Transactions on Computers*, pages 2–12, 1991.

[61] J. N. Mailhot. *A Comparative Study of Routing and Flow Control Strategies in k-ary n-cube Networks.* B.S. thesis, Computer Science, Massachusetts Institute of Technology, 1989.

[62] P. Muzumdar. Evaluation of on-chip static interconnection networks. *IEEE Transactions on Computers*, pages 365–369, Mar. 1987.

[63] T. Nash et al. High performance parallel computers for science: New developments at the fermilab advanced computer program. Technical report, Fermi National Accelerator Laboratory, Illinois, 60510, Aug. 1988.

[64] D. Nath, S. Mahasweri, and P. Bhat. Efficient vlsi networks for parallel processing based on orthogonal trees. In *IEEE Transactions On Computers*, pages 569–581, June 1983.

[65] J. Ngai. *Framework for Adaptive Routing in Multicomputer Networks.* Ph.D. dissertation, Computer Science, California Institute of Technology, 1989.

[66] S. F. Nugent. The iPSC/2 direct-connect communication technology. In G. C. Fox, editor, *Proc. 3rd Conference on Hypercube Concurrent Computers and Applications*, pages 51–60. ACM, 1988.

[67] D. M. Passe and A. R. Larrabee. Intel iPSC concurrent computer. In R. G. Babb II, editor, *Programming Parallel Processors*, pages 105–124. Addison Wesley, 1988.

[68] N. M. Ram, M. Ramu, and A. Paulraj. On the performance of a backtracking variation of wormhole routing. Technical Report AR-TR-003, Centre for Development of Advanced Computing, Bangalore , India, Aug. 1990.

[69] A. G. Ranade and S. L. Johnson. The communication efficiency of meshes, boolean cubes and cube connected cycles for wafer scale integration. In *Proc. International Conference on Parallel Processing*, pages 479–482, 1987.

[70] D. A. Reed and H. D. Schwetman. Cost-performance bounds for multicomputer networks. *IEEE Transactions on Computers*, pages 83–95, Jan. 1983.

[71] D. D. Riley and R. J. Baron. Design and evaluation of a synchronous triangular interconnection scheme for interprocessor communication. In *IEEE Transactions on Computers*, pages 110–118, Feb. 1982.

[72] M. Rosing and R. P. Weaver. Mapping data to processors in distributed memory computations. In *The Fifth Distributed Memory Computing Conference*, pages 884–892, 1990.

[73] C. L. Seitz. The cosmic cube. In *Communications of ACM*, pages 22–23, Jan. 1985.

[74] C. L. Seitz et al. The architecture and programming of the Ametek series 2010 multicomputer. In *Proc. Hypercube Computers Conference*, pages 33–36, 1988.

[75] C. L. Seitz, J. Seizovic, and W.-K. Su. The C programmer's abbreviated guide to multicomputer programming. Technical Report Caltech-CS-TR-88-1, Computer Science, California Institute of Technology, Pasadena, Apr. 1989.

[76] L. Snyder. Introduction to the configurable highly parallel computer. In *IEEE Computer*, pages 47–56, Jan. 1982.

[77] P. Y. Song. Design of a network for concurrent message passing systems. Master's thesis, Electrical Engineering and Computer Science, MIT, May 1988.

[78] R. J. Swan, S. H. Fuller, and D. Siewiorek. Cm*—a modular multiprocessor. In *Proc. Nat. Comput. Conf.*, pages 39–46, 1977.

[79] H. Y. Youn and A. D. Singh. Near optimal embedding of binary tree architectures in VLSI. In *Proc. The 8th International Conference on Distributed Computing Systems*. IEEE, 1988.