Static and Dynamic Communication in Parallel Computing

by

Emmanouel A. Varvarigos

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 1992

Author ..... Signature redacted
Department of Electrical Engineering and Computer Science
August 9, 1992

Certified by ..... Signature redacted
Dimitri P. Bertsekas
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by ..... Signature redacted
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

Static and Dynamic Communication in Parallel Computing

by

Emmanouel A. Varvarigos

Submitted to the Department of Electrical Engineering and Computer Science
on August 9, 1992, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Communication efficiency seems to be the key to the broad success of massively parallel computation. We introduce several static and dynamic communication algorithms, and evaluate their performance. In particular, the first part of the thesis proposes several static communication algorithms, which execute optimally generic communication tasks in a number of regular network topologies. The second part of the thesis deals with dynamic or stochastic communication tasks. We propose and analyze routing schemes and protocols that work in stochastic environments, while satisfying at the same time some performance guarantees. We consider both one-to-one and one-to-many communication tasks. We also propose a novel switching scheme for multiprocessor communications and analyze it for the case of the hypercube. Most of the static and dynamic algorithms combine optimality properties with ease of implementation.

Thesis Supervisor: Dimitri P. Bertsekas
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I wish to thank my advisor, Professor Dimitri Bertsekas. In everybody's life there is a teacher that has a significant impact on him, and Dimitri was such a teacher ($\delta\alpha\sigma\kappa\alpha\lambda os$) for me. Equally important, Dimitri was an invaluable friend and a constant source of wise advice.

I am thankful to my thesis readers, Professors Bob Gallager, Tom Leighton, and John Tsitsiklis for their genuine interest in my work. Through our discussions I have learned a great deal. I also want to thank John for his friendship and support during my four years at MIT.

The professors, staff and students have made the Laboratory for Information and Decision Systems an excellent place to work. Special thanks go to Muriel Medard,Rajesh Pankaj, Abhay Parekh, Yannis Paschalides, Lakis Polymenakos, Jane Simmons, George Stamoulis, Bruno Suard, Emre Teletar, Paul Tseng, and Petros Voulgaris. With my officemates, Murat Azizoglu and John Young we shared a lot and we had a lot of fun: after their graduation the office has become a less pleasant place to be. Constantine Boussios, Nick Lainis, Marios Papaefthymiou, Costakis Patrikios, Gregory Poulis, and the other members of the infamous "table" made the quality of life outside MIT better than I would have imagined. I wish all of them good luck.

My fiancee Aristea was a constant reminder that there are things more important than MIT courses. My parents Andreas and Kassiani, and my sisters Dora and Rea have given me love and support that I can never repay. This thesis is devoted to them.

This research was supported in part by NSF under Grant NSF-DDM-8903385 and by the ARO under Grant DAAL03-86-K-0171.

Contents

PART II. DYNAMIC TASKS

CHAPTER ONE

Introduction

There are many reasons to seek more computational power than existing computers can give. Large computational problems (PDEs, fluid dynamics, image processing, weather prediction), simulations, optimizations, and AI applications are some of the fields that can use all the computational power they can get. At the same time it seems that the increase in computational power that can be achieved by improvements in the switching times of electronic devices alone is approaching a limit posed by physical constraints. Parallel computation promises faster computers by taking a different approach. Instead of focusing on a single processor to make it more powerful, it tries to create organizations of processors, which can efficiently cooperate on the same problem. We could say that the relation of the field of parallel computation to processors is similar to the relation of sociology to human beings.

The work that is asked from a computer is decomposed into subtasks, each of which is executed by a processor. For example, an image is decomposed into regions, with each processor responsible for the computations related to one of the regions. Then it is plausible that the task will be executed faster than if a single processor was used. Although this idea has been common ground for the past thirty years, only recently have multiprocessor computers become commercially available. The trigger for this evolution was the impressive advances in VLSI technology; extremely sophisticated parallel machines are now feasible due to the fact that very complex systems can be implemented on a single chip, which can in turn be copied at little cost.

There are parallel computers today that have up to 65,000 processors. Such numbers make people believe that Gigaflops or even Teraflops will be almost free in the near future. However, the "infinity" in total processing power does not necessarily carry to a corresponding "infinity" in real computational power. Sometimes the time required to communicate the results of a calculation is larger than the time needed to obtain them. Multiprocessor computers are communication bandwidth limited. It is quite possible that in the future the power of computers will be measured in "Gigasends" and Gigafetches" per second rather than Gigaflops.

The great promise of massively parallel computers has motivated a large amount of research that aims at reducing the communication overhead. This research can be divided into three broad categories. The first deals with the design of algorithms that exploit the parallelism inherent in many problems. The second category of research deals with choosing a topology for the multiprocessor network. The topology should either fit a specific problem at hand (special purpose architectures), or, if we are interested in a more versatile system, it should be easily reconfigured to match a class of problems. For example, an array of processors could be the best topology possible for some image processing application and totally unsuitable for a symbolic, or an AI application due to its large diameter. When choosing a topology we are interested in predicting performance measures of the system such as the diameter, the mean throughput and delay, how well it can simulate other topologies, and its performance for benchmark communication tasks (e.g., permutations, broadcasts etc.). The third major research concentration is in the area of routing and organizing the movement of data within a computer in a way that achieves high (often close to 100%) and efficient utilization of the communication resources. For example, by identifying traffic patterns (scenarios) that arise often in applications, we can create a library of efficient communication algorithms that are called as "communication primitives" when needed.

These three categories of research have had considerable interaction and influence on each other. For example, when designing parallel algorithms one should have in mind the underlying topology and the traffic patterns that can be routed efficiently. Similarly, when trying to find efficient ways to move data, one should know the topology and have some knowledge of the algorithms that will run on it. The rapid progress of parallel processing in the 1980s and 1990s owes a lot to the interaction of research in these three categories.

The focus of this thesis falls into the second and the third categories above. We look at several network topologies and we analyze their behavior under various traffic conditions. We also propose communication algorithms, rules, or protocols to organize the communication within a multiprocessor network. When doing so we are sometimes motivated by the underlying computations that give rise

to the information exchange between the processors. In other cases we adopt probabilistic models for the traffic and analyze the performance of the networks based on these models. Most of the work is related to the hypercube and the mesh networks of processors, which are currently the two most popular topologies for multiprocessor systems. In many cases the particular network is of no interest and the rules that we propose have much broader applicability. Also, the methodologies that we use can often be transferred to other networks, or other problems.

Some of the research in this thesis takes the viewpoint of a parallel computation theorist. Most of the problems addressed, however, are directly related to practical applications. We hope that sometime, the communication algorithms, rules, and protocols that we propose will be implemented in actual multiprocessor systems.

1.1. PARALLEL PROCESSING SYSTEMS

There are several issues that have to be resolved when designing a parallel machine, including the following;

- fine versus coarse granularity
- multiple versus single instruction streams
- shared memory versus message passing

We will comment on these issues to the extent that it is necessary to place our work in the general framework.

There is a tradeoff between the size and the number of processors. The fine grained, or massively parallel machines (examples being the Connection Machine model CM-2, and the DAP) have thousands or tens of thousands of small (even one-bit), and relatively inexpensive processors. In such machines a problem is decomposed into thousands of pieces each of which is executed by a small processor. Usually fine grained machines come in conjunction with a host computer, which is a conventional computer; the host computer performs the serial work that would be inefficient to assign to a small processor. The opposite and more conservative approach, which is closer to the Von Neumann principles, is taken by the coarse grained parallel systems, such as the Encore and the Alliant computers, which have some tens or hundreds of fairly powerful processors.

This thesis is implicitly inclined towards fine grained multiprocessor systems for two reasons. The first is that such systems can achieve speedups of several orders of magnitude compared to conventional computers. Coarse grained systems on the other hand achieve much smaller speedups. Such small speedups would also be welcome of course, but they would still leave many applications in need of more computational power. A second reason is that in fine grained machines, the communication part, which is the subject of this thesis, becomes more significant due to the larger size of the network and the information exchange that has to take place. In fine grained machines we expect a reference to a variable that is located at a different processor to take place every few instructions; this makes the communication requirements much stricter, and the issues addressed in this thesis more significant.

A second question when designing a parallel computer is whether the control should be local or global. There are two extreme options here, and many intermediate alternatives. A Multiple Instructions Multiple Data (MIMD) machine (examples being the Encore, Sequent, Ametek N-cube, BBN Butterfly, Intel Hypercube) is a collection of connected, but loosely coupled, autonomous computers, each capable of executing its own program. In a Single Instruction Multiple Data (SIMD) machine (Connection Machine model CM-2, DAP) all processors are controlled from a single instruction stream, which is broadcast to all the processors. Each processor has the option of executing an instruction or ignoring it, depending on its internal state. In MIMD machines synchronization is not given (although it may be built), while SIMD machines are synchronous by definition. Both modes of operation have their advantages, but since each of them can emulate the other, the philosophical issue of which to choose becomes a question of which is easier to build and operate. The results of our thesis apply to both MIMD and SIMD machines, but they are more useful for SIMD machines. The first reason is that SIMD machines usually employ fine grained parallelism, which as we explained makes the communication aspects of parallel processing more significant. A second reason is that in SIMD machines all processors execute the same instructions; this makes the resulting communication pattern more regular, easily describable, and predictable. This leaves more space for clever organizations of the data movement. Part I of our thesis which deals with creating a library of some standard static communication tasks is more directly applicable to SIMD machines.

Parallel processing systems are also distinguished with respect to whether the processors exchange information through a global shared memory, or through message passing. Shared memory machines contain a global memory that can be addressed by all processors, usually via a bus. Processors communicate by leaving messages at prespecified memory locations,

which other processors can read. Shared memory can support some tens of processors (Encore, Sequent, Alliant). The processors of message passing machines on the other hand have local memory and communicate through the exchange of messages (Illiac 4, DAP, Connection Machine models CM-2 and CM-5, Intel Hypercube, Ametek N-cube). Usually there is software that presents the programmer with the illusion of a shared memory environment. This is a dangerous illusion, however, and the programmer should not ignore the message passing nature of the machine. When a processor reads a variable from its local memory, it is like addressing a cache memory location in a conventional computer, while when it reads a variable stored in another processor, it is similar to fetching something from a disc.

This thesis focuses on message passing machines. The reasons are again that such systems can offer a higher degree of parallelism, and that their communication aspects are more important. The network topologies that we examine are prime candidates for use in message passing machines.

1.2. COMMUNICATION ASPECTS OF PARALLEL PROCESSING

N processors working on some problem do not necessarily solve it N times faster than a single processor working alone. A major cause of inefficiency is, as we explained, the communication overhead. Processors, when doing computations, often have to exchange intermediate results. In many cases, the time required for routing the messages is much more than the time spent by the processors doing computations. Thus, at least with current technology, communication seems to be the bottleneck of parallel computation. One can see that by looking at the current success stories of parallel processing, which are limited to applications that use fine grained parallelizations, or fit very well into a particular topology (e.g., image processing applications). In order to move to massively parallel computation, and be able to solve more general problems, it is important to reduce the communication delay. Even if optical fibers are used for the interconnections, the communication delay will still be present in the form of processing and propagation delay.

In order to quantify the effect of the communication overhead, the communication penalty was defined in [BeT89] as the ratio

$$CP = \frac{T_{TOTAL}}{T_{COMP}},$$

where T_{TOTAL} is the total time to solve the problem, and T_{COMP} is the time that would be required if all communications were instantaneous. The communication penalty is always larger than one. The general direction of our thesis is to make the communication penalty closer to one, by decreasing the numerator of the above ratio.

Communication Delay Components

The time required for the transmission of a packet over a link can be decomposed as follows:

- communication processing time, needed to prepare the information for transmission; this is usually considered constant for all packets.
- transmission time, which is inversely proportional to the capacity of the links (or the number of wires implementing a link) and proportional to the length of the packet
- propagation time, which is the time that elapses between the transmission of a bit and its reception at the other end of a link; this is usually considered a constant or some function of the length l of the link (proportional to $\log l$ for small l and linear in l for large l)

The communication delay for internode communication in a parallel computer includes:

- queueing delays,
- delays due to retransmissions, either because a packet was dropped or an error was detected; the latter rarely happens in multiprocessor computers, where transmission of information over a link is generally considered reliable, and
- delays due to the number of hops between the origin and the destination of a packet.

The reduction of the time required for transmission over a link is beyond the scope of this thesis; this pertains to the fields of solid state and VLSI implementation of devices. The communication delay that is due to the last three reasons can be improved by cleverly choosing the topology of a parallel computer, and the communication rules, algorithms and protocols that are built in it.

Communication Model

Throughout most of the thesis we assume that information is transmitted in the form of packets, each of which requires one unit of time to be transmitted over a link. This is usually

a realistic assumption. It also constitutes an abstraction that enables us to focus on the many remaining issues involved in the design of communication algorithms. This abstraction provides a way to measure the performance of communication algorithms, and compare it with lower and upper bounds. Some authors ([SaS89b], [JoH89]) divide the delay required for a packet transmission over a link into two parts: a start-up delay which is the same for all packets, and a part which is proportional to the length of a packet. We believe that such a distinction obscures the picture without providing any additional insight. In any case, appropriate modifications can always be made with little effort to convert the complexity results that we obtain to the other model.

We assume that packets are transmitted in a store and forward fashion. This leaves out the option of wormhole routing (called cut-through routing in data network circles), which has recently received a great deal of attention in the parallel computation literature ([KeK79], [DaS87], [Dal90b]). In wormhole routing a packet starts being transmitted before it has been fully received at a node. In this way, the packet delay is reduced due to pipelining of the packet bits over several links. Most of the static algorithms that we will propose achieve utilization close to 100% of some critical communication resource, and therefore they cannot be improved by using wormhole or some other kind of routing. Wormhole routing becomes very interesting when the load in the network is light and the messages are relatively long. In Chapter 6 we will compare wormhole routing to a number of other schemes, while elsewhere we will pause to discuss how wormhole routing can, or cannot help in a particular case.

1.3. STATIC COMMUNICATION TASKS

There are some traffic patterns that frequently arise in applications. It is desirable to devise standard communication algorithms that execute these typical patterns in the minimum number of steps. These algorithms are used as communication primitives by the programmer or the compiler, in the same way that subroutines implementing standard functions are called from a library of functions in a conventional computer. The time required to execute the prototype tasks can also be used as a performance measure in comparing multiprocessor computers ([BeT89], [Hsu90]).

The simplest of all tasks is the *one-to-one* or *one-shot* communication, where a node sends

a packet to some other node. This corresponds to a processor accessing a non-local memory location. The relevant performance criteria in this case are the diameter and the mean internode distance of the network.

A slightly more complicated task is the *single node broadcast* (SNB for brevity) where the same packet is sent (copied) from a given node to all other nodes. A single node broadcast can be accomplished by transmitting the packet along a spanning tree rooted at a given node. It corresponds to a variable being read by all processors simultaneously. A dual task is the *single node accumulation* (SNA), where a packet is sent to a given node from every other node; here, we assume that the packets can be combined on a link, with the combined transmission time being equal to the transmission time of a single packet. The operator used in combining the packets can be any associative operator, such as $+$, $\cdot$, min, max, and others. The SNA implements a concurrent write from all nodes to a single memory location. The spanning tree used for the SNB with the arcs reversed can be used for the SNA task as well; thus, the time complexity of both tasks is equal to the diameter of the network. A problem slightly more general than the SNB is the *single node multicast*, where the same packet is sent from a given node to some, but not necessarily all, other nodes ([LEN90]). A problem more complex than the previous ones is the *single node scatter* (and its dual *single node gather*), which involves sending (respectively, receiving) a separate packet to (or, from) every other node.

In all the previous tasks, there was a single node which is executing a command and is sending packets to (or receiving packets from) one or more nodes. Generalizations of the previous tasks, where the same action is taken by many or perhaps all processors, also appear very frequently in applications. For example, in SIMD machines where all processors are controlled by the same instruction stream (each processor has, of course, the option of ignoring a command) we expect the same task to be executed by all or a large subset of the nodes. Even in MIMD machines, there are many algorithms, which require the tasks described above to be executed by many or all nodes.

A generalized, multinode version of the one-to-one communication is the *permutation task*, where every processor sends a packet to some other node, with no two destinations being the same. *Partial permutations* and *h-relations* (which are the concatenation of h successive permutations) have also appeared in the literature. The permutation task is considered a benchmark task in parallel computation theory. It is also a measure of how well a given parallel architecture can simulate an "ideal parallel machine", in which all processors can concurrently write to, or read from a different processor.

A generalized version of the single node broadcast task is the *multinode broadcast* (or MNB). In the MNB each node wishes to broadcast a packet to all the other nodes. A task more general than the MNB is the *partial multinode broadcast* (or PMNB) where an (arbitrary) subset of the processors want to broadcast a packet. The PMNB task, along with being important on its own merit (see Chapter 3 for examples), is also a critical component of the dynamic broadcast algorithms that we propose in Chapter 4 (see also the next section).

A generalization of the single node scatter is the *total exchange* task, where each node sends a separate (personalized) packet to every other node. An even more general task than the total exchange is the *partial exchange* (or PE), where a (arbitrary) subset of the nodes of an N-processor system sends a separate packet to each of the $N-1$ other processors. A dual of the partial exchange task is the *partial multinode gather* (or PMNG), where each node in a subset of nodes of the network receives a separate packet from every other node of the network. Examples where the MNB, TE, PMNB, PE, and PMNG tasks arise are given in Chapter 3. The PE is the highest problem in hierarchy among the previously described problems in terms of difficulty. An algorithm for the PE can execute all the other tasks that we presented, although usually not efficiently.

A last class of communication tasks that we will examine are the *isotropic tasks*, first defined in [Var90] and [VaB90a]. These are tasks defined on regular networks (as are most of the networks used in parallel computation), which are characterized by a type of symmetry with respect to each origin. A precise definition will be given in Chapter 2 of the thesis. The total exchange task described above is a particular case of an isotropic task for hypercubes, d-dimensional meshes, folded-cubes, permutation-cubes (the latter class of networks will be defined in Chapter 7), and other networks.

All the tasks described above are *static* in the sense that there is some work to be performed once and for all. The packets that each node has to send are available at time $t = 0$, although usually some condition milder than that is required (e.g., for the isotropic tasks). All the nodes know which task they execute, and they are synchronized to start at the same time; the only objective is to finish the job as fast as possible. Except for the static tasks, where conditions are rather favorable, one can envision situations where communication requests are not deterministic, but they are generated at random instants. We call such an environment *dynamic*. The execution of asynchronous computation algorithms is one such situation, but it is reasonable to expect that in many systems a dynamic, largely unpredictable environment may be the rule and not the exception. Section 1.4 describes this dynamic setting.

1.4. DYNAMIC COMMUNICATION TASKS

In the static setup, which we described in the previous section, our focus is on designing algorithms that execute certain prototype deterministic communication tasks in the minimum number of steps. Thus the problem is equivalent to scheduling packet transmissions, given the exact communication pattern in advance. In a dynamic setup the situation is much less predictable. Communication requests of the one-to-one (one-shot) or the one-to-many (broadcast) type are generated at random instants. In such a situation, the only reasonable target can be the design of communication rules and protocols that work well in stochastic environments. The performance criteria could be the mean number of communication requests that are executed per unit of time, and the mean delay to serve a request. The communication rules and protocols can only be useful if they are on-line, distributed, and easy to implement.

The fact that the communication requests are generated in a random way does not necessarily imply that they should be executed in a greedy, unorganized way. For example, in the case where broadcast requests are generated at random instants at each node, we will find that some global information (e.g., the total number of outstanding requests) is very helpful. Such information can be gathered on line at very little cost, and can be used to organize the packet transmissions. "Reservations" and "on-line scheduling" will be central in many dynamic communication algorithms that we will propose. Other issues that are important in the dynamic setup are the switching format, the flow control, and the feedback mechanism used. In Chapter 6 we will propose a hybrid of packet and circuit switching which we call the Conflict Sense Routing protocol.

1.5. TOPOLOGIES

The choice of topology is a central issue in the design of a multiprocessor system for obvious reasons. Numerous topologies have been proposed and analyzed including the hypercube, the mesh of trees, the d-dimensional mesh (with or without wraparound), the butterfly, the tree, the ring, and the linear array (a good survey of topologies can be found in [Fen81]). The hypercube and the mesh (especially the 2-dimensional) are the most popular topologies in terms of actual

implementations. They are also the ones of greatest interest in this thesis.

The Hypercube

The hypercube graph (Fig. 1.1) comes in various sizes. In particular there is a hypercube of N nodes for every N which is a power of 2. Each node of an $N = 2^d$-node hypercube is represented by a unique d-bit binary string $w_{d-1}w_{d-2}\cdots w_0$. Two nodes are connected via a link if they differ in one bit of their representation (i.e. if $w = w_{d-1}w_{d-2}\cdots w_0$ and $w' = w'_{d-1}w'_{d-2}\cdots w'_0$ and $w_i = w'_i$, for all i except for a single j for which $w_j = 1 - w'_j$). Nodes and links of a hypercube graph correspond to processors and wires of a hypercube multiprocessor computer. We will assume, unless otherwise stated, that the links are bidirectional, and each processor can send and receive messages over all its incident links at the same time. The computing power of the hypercube is partly derived from the fact that there are d disjoint paths of length $d+1$ or less linking any pair of nodes.

The *Hamming distance* between two nodes is the number of bits in which their identities differ. The number of links on any path connecting two nodes cannot be less than the Hamming distance of the nodes. Furthermore, there is a path with a number of links which is equal to the Hamming distance, obtained, for example by switching in sequence of bits in which the bit representations of the nodes differ (equivalently, by traversing the corresponding links of the hypercube). Such a path is referred to as a *shortest path*. Given two nodes s and t, $s \oplus t$ denotes the result of their bitwise exclusive OR operation and is called the *routing tag* between the two nodes. The number of ones of the routing tag is obviously the Hamming distance between the two nodes.

The d-dimensional Mesh

The d-dimensional mesh consists of $N = p^d$ processors arranged along the points of a d-dimensional space with integer coordinates. Along the ith dimension, obtained by fixing coordinates $x_{d-1}, \ldots, x_{i+1}, x_{i-1}, \ldots, x_0$ there are p processors with identities $(x_{d-1}, \ldots, x_i, \ldots, x_0)$, $x_i = 0, 1, \ldots, p-1$. Two processors $(x_{d-1}, \ldots, x_i, \ldots, x_0)$ and $(y_{d-1}, \ldots, y_i, \ldots, y_0)$ are connected by a duplex link if and only if for some i we have $|x_i - y_i| = 1$, and $x_j = y_j$ for all $j \neq i$. In

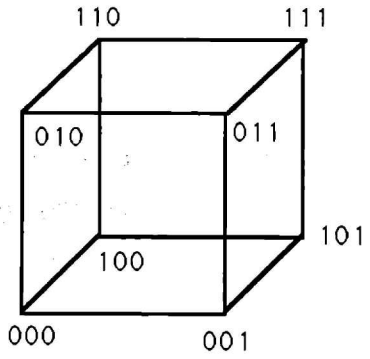

Figure 1.1: A hypercube with $N = 8$ nodes.

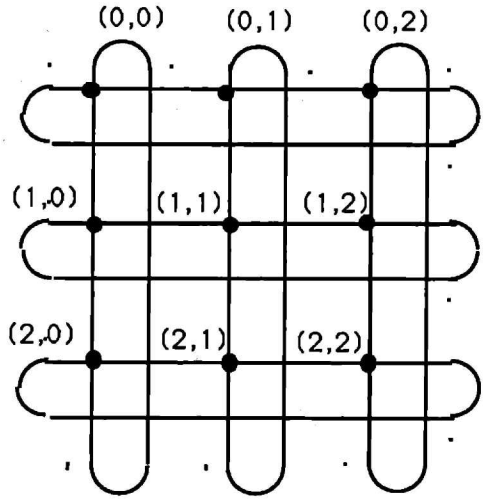

Figure 1.2: A 2-dimensional wraparound mesh with $N = 9$ nodes.

addition to these links, in the d-dimensional mesh *with wraparound* there also exist links of the type $((x_{d-1}, \ldots, x_{i+1}, 0, x_{i-1}, \ldots, x_0), (x_{d-1}, \ldots, x_{i+1}, p-1, x_{i-1}, \ldots, x_0))$. Figure 1.2 illustrates a 2-dimensional mesh with wraparound (also called *torus*).

Among the advantages of the hypercube are its logarithmic diameter, the simplicity of the routing algorithms, its versatility (many other topologies can be embedded in it; see [BeT89] and [Lei92a]), the variety of algorithms designed for it (see [Lei92a] or [RaS90] for a collection), its recursive and other mathematical properties, and its large bisection width $\left(\Theta(\frac{N}{\log N})\right)$. The bisection width is the minimum number of links that have to be removed to separate the hypercube into two parts, each with equal number of nodes. The large bisection width becomes

a disadvantage in the physical layout of the hypercube. The hypercube requires $\Theta(N^2)$ area for layout, which is more than we would desire; most of this area is wires ([Dal90a]). The logarithmic degree is also a potential problem, because of the pin limitations of a chip, and the burden that it places on the processor to handle all links at the same time. These disadvantages may become significant as we move towards larger systems: a hypercube computer having a million processors may just be impossible to build. However, for the parallel computers of today and probably those that will be built in the next decade this is not a problem. Constant degree networks, such as the *cube connected cycles*, the *De Bruijn graph*, and the *shuffle exchange*, which can simulate a hypercube with only logarithmic slow-down, have been popular in the literature, but not in practice (except for the butterfly).

Recently, there is a trend in the industry towards low dimensional meshes. Low dimensional meshes have larger diameter than the hypercube, but they fit better, and in a more natural way in the two dimensions (or perhaps three with future technology) available on a silicon wafer (see [Dal90a] and [LeL89]).

The question of which topology is better for a general purpose parallel computer has not been answered yet. In Chapter 7 we propose some networks, structurally related to the hypercube, which have interesting routing properties.

1.6. THESIS OUTLINE - CONTRIBUTIONS

The thesis is organized as follows. In Chapter 2, we describe the class of isotropic tasks for hypercubes. By introducing the notion of task matrices we transform the problem of scheduling isotropic tasks into a matrix decomposition problem. Solving the transformed problem, we obtain simple distributed rules that execute any isotropic task in strictly minimum number of steps. The rules found for the isotropic tasks give rise to a number of optimal algorithms for the total exchange task, which is a special case of an isotropic task.

Motivated by the simplicity and the nice properties of the routing algorithms for isotropic tasks, we extend the results to other, "nearly isotropic" tasks. We focus on the particular problem of storing by columns, and transposing a banded matrix in a hypercube network of processors. We propose an assignment of matrix columns to hypercube processors for which the transposition task becomes an almost isotropic task. We then present communication

algorithms to transpose a single banded matrix, or multiple banded matrices concurrently. We also prove a lower bound on the time required to transpose a banded matrix for any assignment of matrix columns to hypercube processors. The lower bound shows the optimality of our algorithm for a large range of matrix bandwidths. The fastest previous algorithm that we know of for transposing a banded matrix is the one given in [McV87], which is worse by a factor of $\log N$ compared to ours. Using the results on banded matrices we propose a way to embed arbitrary sparse graphs in a hypercube topology. We also present a way to store block diagonal matrices in a hypercube network of processors.

In Chapter 3 we first introduce the partial multinode broadcast and the partial exchange tasks. We propose several algorithms to execute these tasks in hypercubes and d-dimensional meshes. Near-optimal algorithms, or algorithms of optimal order are found for both tasks and both networks considered. The algorithms are distributed, on-line, and assume no information about the location of the active nodes. We consider two different communication models: in the first model packets can be split at the origin and be recombined at the destination, while in the second model this is not allowed. We also present results on the simulation of meshes with wraparound by meshes without wraparound, on the packing and isotone routing problems for d-dimensional meshes, on the window multinode broadcast in hypercubes, and on other routing problems. The partial multinode broadcast has been considered independently in [Sta91] for the hypercube network, where an algorithm having roughly twice the complexity of ours was proposed. The other problems with which we deal in Chapter 3 are considered for the first time.

Starting with Chapter 4 we turn our attention to dynamic communication algorithms. In Chapter 4 we consider the problem where broadcast requests are generated at each node according to a Poisson process, independently of the other nodes. We propose a simple and natural scheme to execute the broadcasts. The scheme consists of alternating reservation and broadcast intervals. In the reservation intervals some minimal global information is gathered at each node, and nodes that have a packet to broadcast reserve their participation in the next broadcast interval. At the same time the packets are moved to more favorable intermediate nodes. In the broadcast interval the packets are broadcast from the intermediate nodes to all other nodes. The scheme can be used in hypercubes, d-dimensional meshes and tori, and, in general, any network for which PMNB algorithm with certain properties can be found. Since the partial multinode broadcast algorithms of Chapter 3 are very efficient for both hypercubes and meshes, and the reservation overhead is small, the proposed scheme turns out to be very

efficient. In particular, we set two objectives for a dynamic broadcasting scheme: (1) stability for as big a load as possible, and (2) average delay which is of the order of the diameter for any fixed load in the stability region. Our scheme has an asymptotically optimal stability region, and an asymptotically optimal average delay in terms of the size of the network for any fixed load in the stability region. The only other work related to the problem of dynamic broadcasts was done by Stamoulis in [Sta91]. The two algorithms of Stamoulis that are most interesting from a theoretical point of view are the non-idling direct scheme with priority rule (or simply, direct scheme), and the indirect scheme. The stability objective described above is met by the direct scheme, but the average delay analysis of the direct scheme is approximate. The indirect scheme meets the delay objective, but its stability region is roughly 2/3 of the maximum possible. Therefore the two schemes do not provably satisfy both performance objectives. Our algorithm and analysis are of a completely different philosophy, and apply to any network for which efficient PMNB algorithms can be found. The results in [Sta91] are particular to the hypercube topology.

In Chapters 5 and 6, we shift our focus to multiple one-to-one communications in hypercubes, under a stochastic environment. The problem with which we deal in Chapter 5 is the following. Packets having a single destination are generated at each node of a hypercube in a stochastic way. This is similar to the setup of Chapter 4 with the difference that now one-to-one communication requests are generated instead of broadcasts. The packet destinations are assumed to be uniformly distributed over all nodes. We are interested in distributed routing schemes that will work well in such a stochastic environment. One-to-one routing has been analyzed extensively in the literature for a variety of topologies. Valiant [Val82], and Valiant and Brebner [VaB81] have proposed randomized routing as a way to route *uniformly* any permutation in a hypercube in logarithmic time with high probability. Their work was improved by Upfal [Upf84] who achieved similar performance by using a constant degree network, and Pippenger [Pip84] who used queues of constant size. Mitra and Cieslak [MiC87], Hajek and Cruz [HaC87], and Greenberg and Leiserson [GrL89], have extended these results to other topologies, namely the Extended Omega, and the Fat Tree network. We do not know, however, of any parallel computer that uses randomized routing (it must be hard to persuade managers to use probability theory).

In the field of deterministic routing schemes for one-to-one communications, Greenberg and Goodman [GrG86] gave an approximate numerical analysis of deflection routing in mesh networks. Greenberg and Hajek ([GrH90]) approximately analyzed (but not numerically) a de-

flection routing scheme for hypercubes. Dias and Jump [DiJ83] used approximate analysis to evaluate the performance of buffered Delta networks. Stamoulis [Sta91] analyzed a greedy routing scheme in a hypercube with infinite buffers. He analyzed a closely related Markovian network with partial servers and used the results to bound the performance of the hypercube scheme. Maxemchuk [Max89] used simulations to compare deflection routing in a shuffle exchange and in a Manhattan network. Varvarigos in [Var90] introduced a priority deflection routing scheme for hypercubes, and used a Markov chain to analyze it approximately, and to evaluate its performance numerically.

We propose two different routing schemes and evaluate their steady state throughput for various traffic loads. The schemes are simple and require simple, low-cost switches at the hypercube nodes, instead of crossbar switches. The one of the two schemes uses a priority rule to resolve conflicts over a link. The priority rule is found to increase the throughput significantly. For both routing schemes we examine the effect of the buffer size on the throughput. The results obtained are approximate, but very accurate as simulation results indicate, and they are given in particularly interesting forms: they are either in parametric form involving a single parameter, or they are obtained by a recursion of finite order equal to the dimension of the hypercube. The effect of the buffer space on the throughput of hypercubes, which we examine in this chapter, has not been adequately addressed in the literature. Also, we are not aware of any other analysis of priority routing schemes for hypercubes, with the exception of the priority deflection scheme analyzed in [Var90]. We also use simulation to evaluate the performance of two deflection schemes, called the simple and the priority deflection schemes. These schemes, which use crossbar switches at the nodes, have very satisfactory throughput; in fact, the priority deflection scheme is conjectured to have throughput asymptotically equal to the maximum possible.

In Chapter 6 we focus on an issue ignored in the previous chapters and, in fact, in most of the parallel computation literature. This has to do with the switching format used in multiprocessor communications. We propose a new switching format, which we call the *Conflict Sense Routing Protocol* (or CSR protocol). This is a hybrid of packet and circuit switching, and combines most of their individual advantages. We initially present the CSR protocol in a way that is applicable to a general topology. The CSR protocol efficiently resolves data link control issues, such as the link and buffer space allocation strategy, the feedback mechanism, and the retransmission protocol. We then present an implementation of the CSR protocol in a hypercube network. We approximately analyze the performance of the hypercube implementation assuming that

packets having a random destination are generated at random instants at each node. We find that the CSR protocol provides very satisfactory throughput, together with a number of other advantages. We conclude the chapter by making a comparison of the hypercube CSR protocol with several other switching formats and schemes.

In Chapter 7 we define some new classes of network topologies, which are structurally related to the hypercube topology. Our interest for studying these networks comes from two observations. The first is that the hypercube does not have a particularly small diameter given its degree. In a network of $N = 2^d$ nodes, each having degree d, we would hope for a diameter of the order $\Theta(d/\log d)$ instead of d. Even though we are unable to calculate the diameter of the proposed networks analytically, simulation results indicate that their mean internode distance is rather small. The second observation is that most of the properties that have made the hypercube popular are closely related to its recursive properties. From our point of view the nice routing properties of the hypercube are due to the single fact that it can be decomposed into two parts of the same kind, and each node in the one half has a neighbor in the other half. The networks proposed share with the hypercube this property, which is adequate to guarantee simple, self-routing algorithms. In addition, we show that static communication tasks, such as the total exchange, can be efficiently executed in some of the proposed networks, by using ideas similar to those used for the hypercube. We also consider a variation of the hypercube, called the *folded cube* (proposed in [AdS82]), and present the first strictly optimal MNB and TE algorithms for this topology, improving on previous results by [Ho90]. In order to support our opinion that the networks examined are viable multiprocessor topologies, we propose ways to layout these networks. We find that all the proposed networks when placed on silicon require area $O(N^2)$, which is of the same order of magnitude with the hypercube.

Networks structurally related to the hypercube have often appeared in the literature. The bridged hypercube ([ElL90]), the folded-cube ([AdS82] and [Ho90]), the incomplete hypercube ([Kat88]), the Fibonacci cube ([Hsu90]) are some examples. Our work differs from these works because it focuses on the recursive properties of the hypercube (and in fact of other parallel computing networks) that simplify routing, and ignores other, less essential we believe, properties.

CHAPTER TWO

Isotropic and Nearly Isotropic Tasks

We consider a broad class of communication tasks, which we call isotropic, in a hypercube network of processors. These tasks are characterized by a type of symmetry with respect to origin node. We show that executing such tasks in a minimum number of steps is equivalent to a matrix decomposition problem. We use this property to obtain minimum completion time algorithms for any isotropic task. We then turn our attention to communication tasks which are "nearly isotropic". In particular, we consider the problem of transposing a sparse matrix of size $N \times N$ with a diagonal band of size $2^{\beta+1}+1$, which is stored by columns (or rows) in a hypercube network of $N = 2^d$ processors. We propose an assignment of matrix columns to hypercube nodes such that the transposition task becomes a "nearly isotropic" task, that is, it looks "almost identical" to all nodes. Under this assignment, an algorithm is found to transpose the matrix in 2^β steps. We also find a lower bound on the minimum number of steps required to transpose a banded matrix, which holds for any possible assignment of matrix columns to hypercube processors. In the case that $2^{\beta+1}+1 = \Theta(N^c)$ for some constant $c \geq 0$, the completion time of our transposition algorithm is proven to be of the same order of magnitude with the lower bound. We further show that $\lfloor d/\beta \rfloor$ banded matrices, each of bandwidth $2^{\beta+1}+1$, can be stored by columns in a hypercube so that all of them can be concurrently transposed in $2^{\beta+1}$ steps. We also give a heuristic algorithm to embed arbitrary sparce graphs with small dilation in hypercubes, and we present a way to store block-diagonal matrices in hypercube networks.

2.1. INTRODUCTION

Routing algorithms have been studied by several authors under a variety of assumptions on the communication network connecting the processors of a parallel computing system. Saad and Shultz [SaS89a], [SaS89b] have introduced a number of generic communication problems that arise frequently in numerical and other methods. For example they consider the problem where each processor is required to send a separate packet to every other node; following [BeT89], we call this the *total exchange* problem. Saad and Schultz have assumed that all packets take unit time to traverse any communication link. Processors can either transmit along all their incident links simultaneously or they can transmit along a single incident link at any one time. Johnson and Ho [JoH89] have developed minimum and nearly minimum completion time algorithms for similar routing problems as those of Saad and Schultz but using a different communication model and a hypercube network. Their model quantifies the effects of setup time (or overhead) per packet, while it allows packets to have variable length, and to be split and be recombined prior to transmission on any link in order to save on setup time. In the model of [JoH89], each packet may consist of data originating at different nodes and/or destined for different nodes. The extra overhead for splitting and combining packets is considered negligible in the model of [JoH89]. Bertsekas et al [BOS91], and Bertsekas and Tsitsiklis [BeT89] have used the communication model of Saad and Shultz to derive minimum completion time algorithms for several communication problems in a hypercube. In particular, they have given an algorithm for the total exchange problem that executes in a minimum number of steps ($N/2$ for an N-processor hypercube). Several other works deal with various communication problems and network architectures related to those discussed in the present chapter; see [BhI85], [DNS81], [Ede91], [HHL88], [Ho90], [Joh87], [KVC88], [McV87], [Ozv87], [SaS88], [StW87], and [Top85].

In this chapter, we introduce a class of communication tasks, called *isotropic*, which are characterized by transmission requirements that are symmetric with respect to origin node (a precise definition will be given later). For example, the total exchange problem is an isotropic task; the communication problem "looks identical" to every node. The structure of isotropic tasks can be exploited particularly well in networks that have themselves a symmetric structure, such as a hypercube, a wraparound mesh, a folded-cube, a permutation-cube, and other networks. In this chapter, we restrict attention to the hypercube network. Algorithms for isotropic tasks in wraparound meshes can be found in [VaB90a], while relevant results for the

folded-cube and the permutation-cube are given in Chapter 7. We use the Saad and Schultz communication model, but as we will show in Section 2.4, our minimum completion time results are essentially independent of the communication model used. The idea is that to achieve minimal completion time, some critical network resource must be used 100% of the time, and this constraint is limiting for any communication model.

A central result is that executing isotropic tasks on a hypercube (and a number of other regular topologies; see comment above) is equivalent to solving a matrix decomposition problem. We use this result to characterize the class of algorithms that execute such tasks in minimum time. Within this class one can identify simple and easily implementable algorithms with further optimality properties, such as minimum or nearly minimum average packet delay, memory storage, and transient storage requirements. For algorithms which are optimal with respect to the latter performance crireria we refer the reader to [VaB90a], where algorithms simultaneously achieving various kinds of optimality are described; the only performance criterion considered in this chapter is the completion time of an algorithm. Earlier works in data communications ([BCW81] and [Wan88]) have also shown the equivalence of certain optimal time slot allocation problems and matrix decomposition problems. However, these works involve a very different context where there is one transmitter, and several receivers connected with a direct link to the transmitter; network situations are not addressed and symmetry plays no role.

Beginning with Section 2.5 we turn our attention to "nearly isotropic" tasks. The quotes are used to indicate that this is a class of tasks with no specific or rigorously defined borders. The term is used in a heuristic way to refer to communication tasks for which the tools, algorithms, and ideas developed for isotropic tasks can also be useful. Loosely speaking a "nearly isotropic" task is a task which looks almost identical to all nodes.

We focus on a particular "nearly isotropic" task, namely the transposition of a banded matrix stored by columns (or by rows) in a hypercube network. We show that for a particular assignment of matrix columns to hypercube nodes the task is close to being isotropic. We then give an algorithm to transpose a $2^{\beta+1}+1$-diagonal matrix in 2^{β} steps. This is a factor of $\log N$ improvement over the best previous banded-matrix transposition algorithm that we know of ([McV87]).

We have not proved that our banded-matrix transposition algorithm is optimal over *all* possible column-to-processor assignment for *any* matrix bandwidth. It is strictly optimal for the column to processor assignment that we propose, but it is possible that a better assignment exists. We have derived, however, a lower bound showing that when the bandwidth B is of the

order $\Theta(N^c)$ for some constant $c > 0$, the complexity of our algorithm is of the optimal order of magnitude, where optimality is considered over all possible column-to-processor assignments, and over all transposition algorithms for each particular assignment. The case $c = 1/2$ is particularly interesting in practice, since it arises, for example, in the discretization of elliptic partial differential equations. Our transposition algorithm is also of optimal order when the bandwidth B is $O(1)$. We extend our results to show that $\lfloor d/\beta \rfloor$ matrices, each of bandwidth $2^{\beta+1} + 1$, can be transposed concurrently in time $2^{\beta+1}$. When $\lfloor d/\beta \rfloor > 2$ the improvement in efficiency obtained in this way may be significant. The results on banded matrices suggest heuristic ways to embed arbitrary sparse graphs in hypercubes, with small dilation. This is a difficult problem for which very few things are known.

There are two main results in this chapter. The first is to relate the routing problem, which is a scheduling problem with a combinatorial character, with a matrix decomposition problem, which is a problem in linear algebra. Such a connection is new and quite unexpected. It provides a simpler and more powerful characterization of optimal routing algorithms for the total exchange and other related problems than in earlier works (e.g. [BOS91]). It also allows simple and elegant analyses of minimum average delay algorithms and near-optimal greedy algorithms; we will not deal with these issues in this thesis and we refer the reader to [VaB90a].

The second main result of the chapter is to introduce isotropic tasks and "nearly isotropic" tasks as practically important and analytically interesting classes of communication problems. It is clear that there is an incentive to formulate new routing problems in terms of isotropic or "nearly isotropic" tasks, whenever this is reasonable, to take advantage of the corresponding simple and elegant analysis.

The chapter is organized as follows. Sections 2.2 through 2.4 deal with the isotropic tasks, while Sections 2.5 through 2.7 deal with the banded-matrix transposition task. Section 2.2 defines the class of the isotropic tasks and introduces the key notion of the task matrix. A lower bound for the completion time of both isotropic and non-isotropic tasks is also given. Section 2.3 deals with the evolution of the task matrix when symmetric routings are used. It also transforms the problem of minimizing the task's execution time into the problem of writing the task matrix as the sum of a minimum number of permutation matrices. The solution to the matrix decomposition problem is given in Section 2.4. Section 2.5 describes the banded-matrix transposition algorithm. A lower bound on the time required to transpose a banded matrix, and the optimality of our algorithm for a broad range of matrix bandwidths are proved in Section 2.6. In Section 2.7 we give an algorithm to transpose many banded matrices simultaneously. In

Section 2.8 we describe a heuristic algorithm to embed sparse graphs in hypercubes. Finally, in Section 2.9 we present a way to embed block diagonal matrices in hypercubes.

2.2. THE TASK MATRIX

We first introduce some terminology. Given a hypercube with 2^d nodes, the *j-type* link (or j-link) of node $s = (s_{d-1} \dots s_j \dots s_0)$ is the link connecting node $(s_{d-1} \dots s_j \dots s_0)$ with node $(s_{d-1} \dots \overline{s}_j \dots s_0)$. (We denote by $\overline{x}$ the complement of the binary number x, that is, $\overline{x} = 1 - x$.) Given two nodes s and t, the node $s \oplus t$ is the node with binary representation obtained by a bitwise exclusive OR operation of the binary representations of nodes s and t.

Information is transmitted along the hypercube links in groups of bits called *packets*. In the algorithms in this section we assume that the time required to cross any link is the same for all packets, and is taken to be one unit. We assume that packets can be simultaneously transmitted along a link in both directions, and that their transmission is error free. Only one packet can travel along a link in each direction at any one time; thus, if more than one packet are available at a node and are scheduled to be transmitted on the same incident link of the node, then only one of these packets can be transmitted at the next time period, while the remaining packets must be stored at the node while waiting in queue. We assume that all incident links of a node can be used simultaneously for packet transmission and reception. Finally, we assume that each of the algorithms proposed is simultaneously initiated at all processors.

We now define the communication tasks that are the subject of this chapter.

Definition 1: A *communication task* $\mathcal{G}$ is defined as a set of triplets (s_1, s_2, k), where s_1 is a node (source), s_2 is a node (destination), and k is an integer (the number of packets whose source is s_1 and whose destination is s_2).

Definition 2: A communication task $\mathcal{G}$ is called *isotropic* if for each packet that node s_1 has to send to node s_2, there is a corresponding packet that node $s_1 \oplus x$ has to send to node $s_2 \oplus x$, where s_1, s_2, and x are arbitrary nodes. Mathematically:

$$(s_1, s_2, k) \in \mathcal{G} \quad \Rightarrow \quad \text{for all nodes } x \text{ we have } (s_1 \oplus x, s_2 \oplus x, k) \in \mathcal{G}.$$

An example of an isotropic task is the total exchange, where $\mathcal{G}$ consists of all the triplets

$(s_1, s_2, 1)$ as s_1 and s_2 range over all the pairs of distinct nodes [one packet for every origin-destination pair (s_1, s_2)].

In the algorithms that we propose, the packets carry with them a d-bit string called a *routing tag*. The routing tag of a packet is initially set at $s_1 \oplus s_2$, where s_1 is the source and s_2 is the destination of the packet. As the packet is transmitted from node to node, its routing tag changes. If at time t a packet resides at a node s_1 and has s_2 as destination, then its routing tag is $s_1 \oplus s_2$. For example, a packet which is currently at node 001010 and is destined for node 101000, has routing tag 100010.

An important data structure that will be used by our routing algorithms is that of the *task matrix* of node s at time t, which will be denoted by $T_t(s)$. The task matrix $T_t(s)$ is defined for both isotropic and non-isotropic tasks and is a binary matrix whose rows are the routing tags of all the packets that are queued at node s at time t. The routing tags appear as rows of the initial task matrices $T_0(s)$ in some arbitrarily chosen order. The rows of subsequent task matrices $T_t(s)$, $t > 0$, retain the relative order that the corresponding packets had in $T_0(s)$. When no packets are queued at node s at time t, the task matrix $T_t(s)$ is by convention defined to be a special matrix denoted Z. A task is said to be completed at time t if $T_t(s) = Z$ for all s. The smallest t for which the task is completed under a given routing algorithm is called the *completion time* of the algorithm.

A communication task can equivalently be defined in terms of its initial task matrices $T_0(s)$, $s = 0, \ldots, N-1$. The task is isotropic if and only if the task matrices $T_0(s)$ are the same for all nodes s. In what follows, whenever there is no reason to distinguish among the nodes, we simply denote the task matrix at time t with T_t. When such a notation is used, we implicitly mean that $T_t(s) = T_t$, for all s. The initial task matrix for the total exchange problem is illustrated in Fig. 2.1.

We will now derive a lower bound for the completion time of any communication task (isotropic or non-isotropic).

Theorem 1: Let $\mathcal{T}$ be the completion time of any algorithm that executes a task with initial task matrices $T_0(s)$, $s = 0, 1, \ldots, N-1$. Let also $r_i(s)$ (or $c_i(s)$) denote the sum of the elements of the i^{th} row (or column, respectively) of the task matrix $T_0(s)$. Then the following inequality holds

$$\mathcal{T} \geq \max_{i,j} \max \left(\frac{1}{N} \sum_{s=0}^{N-1} c_j(s), \max_s r_i(s) \right),$$

where the outer maximization is carried out over all rows i and columns j.

1	1	1
0	1	1
1	0	1
1	1	0
1	0	0
0	1	0
0	0	1

Figure 2.1: The task matrix for the total exchange problem has $N-1$ rows and d columns. The figure illustrates the case where d=3.

Proof: The column sum $c_j(s)$ of the j^{th} column of $T_0(s)$ is equal to the number of packets that reside at node s at time $t=0$ and have the j^{th} bit of their routing tag equal to 1. To arrive at their destination, these packets have to use a j-link at some future time. Thus, $\sum_s c_j(s)$ packets are going to use j-type links during the execution of the task. Since each node has only one j-link, there are only N links of j-type in the hypercube. Taking into account that no two packets can be transmitted on the same link in the same time slot, we conclude that

$$\mathcal{T} \geq \frac{\sum_s c_j(s)}{N}$$

for all columns j. Therefore,

$$\mathcal{T} \geq \frac{1}{N} \max_{j=0,\ldots,d-1} \left(\sum_{s=0}^{N-1} c_j(s) \right). \tag{2.1}$$

On the other hand, the packet corresponding to the i^{th} row of $T_0(s)$ is at a Hamming distance $r_i(s)$ from its destination. Thus the time $\mathcal{T}$ required to complete the task is at least $r_i(s)$ for all rows i and nodes s. This gives

$$\mathcal{T} \geq \max_{i,s} r_i(s). \tag{2.2}$$

By combining Eqs. (2.1) and (2.2), we finally obtain

$$\mathcal{T} \geq \max_{i,j} \max \left(\frac{1}{N} \sum_{s=0}^{N-1} c_j(s), \max_s r_i(s) \right),$$

where the maximization is carried out over all rows i and columns j. **Q.E.D.**

The preceding lower bound cannot always be attained by some algorithm. The following Corollary 1 specializes this lower bound for the case of isotropic tasks. As we will show later there is always an algorithm that achieves the lower bound of Corollary 1.

Definition 3: The *critical sum* h of a matrix is equal to $\max_{i,j}(r_i, c_j)$, where r_i is the sum of the entries of row i, c_j is the sum of the entries of column j, and the maximization is performed over all rows i and columns j. A row or column with sum of entries equal to h is called a *critical line*.

Corollary 1: Let an isotropic communication task have initial task matrix T_0 and h be the critical sum of T_0. Then a lower bound for the time $\mathcal{T}$ required to complete the task is h.

Proof: Using Theorem 1 and the fact that for isotropic tasks we have $T_0(s) = T_0$, $c_j(s) = c_j$, $r_j(s) = r_j$ for all nodes $s = 0, 1, \ldots, N-1$, we obtain $\mathcal{T} \geq \max_{i,j}(c_j, r_i) = h$, for any algorithm that executes the task. **Q.E.D.**

2.3. SYMMETRIC ROUTING ALGORITHMS

In this section we will be interested in isotropic tasks and a class of routing algorithms that satisfy a certain symmetry condition.

Definition 4: Given a task matrix $T_t(s)$ for each node s at time t, a *switching scheme* with respect to $T_t(s)$ is a collection of matrices $\{S_t(s) \mid s = 0, \ldots, N-1\}$ with entries 0 or 1. The matrix $S_t(s)$ has the same dimensions as $T_t(s)$, satisfies $S_t(s) \leq T_t(s)$ (i.e. if an entry of $T_t(s)$ is a zero, the corresponding entry of $S_t(s)$ must also be zero), and has at most one nonzero entry in each row or column. The switching scheme is called *symmetric* if for for every t the matrices $S_t(s)$ are independent of s, that is, if for some matrix S_t we have $S_t(s) = S_t$ for all s.

Given a time $t \geq 0$ and a task matrix $T_t(s)$ for each node s, a switching scheme $\{S_t(s) \mid s = 0, \ldots, N-1\}$ with respect to $T_t(s)$ defines the packet (if any) that will be transmitted on each link at the time slot beginning at time t. In particular, if the $(i,j)^{\text{th}}$ element of $S_t(s)$ is a one,

the packet corresponding to the i^{th} row of $T_t(s)$ will be transmitted on the j^{th} link of node s. The requirement that each column of $S_t(s)$ contains at most one nonzero entry guarantees that at most one packet is scheduled for transmission on each link.

The task matrices at a given time slot together with a corresponding switching scheme, define the task matrices for the next time slot. Given a communication task defined by the task matrices $T_0(s)$, $s = 0, \ldots, N-1$, a routing algorithm can be defined as a sequence $\{S_0(s), S_1(s), \ldots\}$, such that $S_0(s)$ is a switching scheme with respect to the task matrix $T_0(s)$, $S_1(s)$ is a switching scheme with respect to the task matrix $T_1(s)$ (which is defined by $T_0(s)$ and $S_0(s)$), and, recursively, $S_{t+1}(s)$ is a switching scheme with respect to the task matrix $T_{t+1}(s)$ (which is defined by $T_t(s)$ and $S_t(s)$).

The key fact, proved in the following theorem, is that if at some time t, the task matrices are the same for all nodes s, and a symmetric switching scheme with respect to $T_t(s)$ is used, then the next task matrices $T_{t+1}(s)$ will be the same for all nodes. As a result, for an isotropic task, one may use a routing algorithm defined by a sequence of symmetric switching schemes. Such a routing algorithm will be called *symmetric*. Its action is specified at a single node and is essentially replicated at all the other nodes; this is a very desirable property for implementation purposes.

Theorem 2: Assume that for a given routing algorithm, at some time t we have a set of nonzero task matrices $T_t(s)$, which are the same for all nodes s. Then if S_t, a symmetric switching scheme with respect to $T_t(s)$ is used by the algorithm at time t, the task matrices $T_{t+1}(s)$ will be the same for all s. In particular, we have

$$T_t(s) = T_t, \text{ for all } s \qquad \Rightarrow \qquad T_{t+1}(s) = T_{t+1}, \text{ for all } s,$$

where T_{t+1} is a task matrix consisting of the nonzero rows of the matrix $T_t - S_t$, except if $T_t = S_t$ in which case T_{t+1} is equal to the special matrix Z and the algorithm terminates.

Proof: Suppose that at time slot t, node s sends a packet with routing tag $x_{d-1} \cdots x_j \cdots x_0$ over its j-link to node $s \oplus e_j$. Then by the symmetry assumption, node $s \oplus e_j$ also sends a packet with routing tag $x_{d-1} \cdots x_j \cdots x_0$ over its j-link to node $(s \oplus e_j) \oplus e_j = s$. This packet arrives at node s with routing tag $x_{d-1} \cdots \overline{x}_j \cdots x_0$. Thus each row of the task matrix T_t, which corresponds to a packet transmitted at slot t, is replaced by a row $x_{d-1} \cdots \overline{x}_j \cdots x_0$ if $x_{d-1} \cdots \overline{x}_j \cdots x_0$ is nonzero and is discarded otherwise; see Fig. 2.2. Since the transmitted packets (if any) on the j-link correspond to the nonzero entry of the j^{th} column of the matrix S_t, we conclude that $x_j = 1$ and, therefore, $\overline{x}_j = 0$. Thus the routing tag $x_{d-1} \cdots \overline{x}_j \cdots x_0$ is

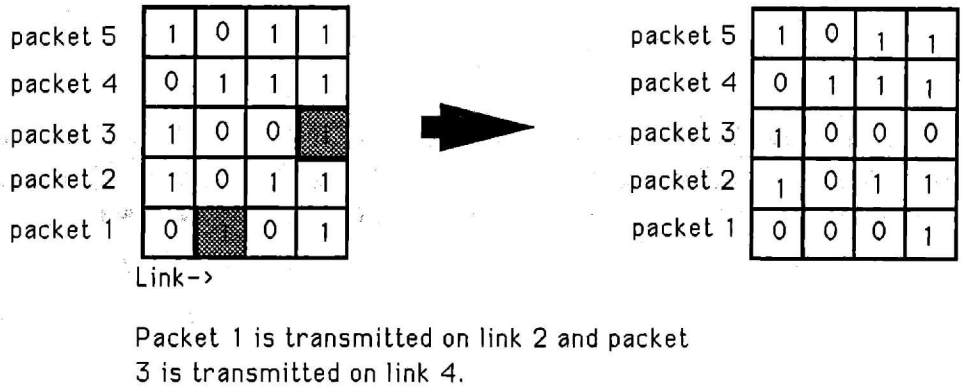

Figure 2.2: The change in the task matrix due to packet transmissions (packets 1 and 3 are transmitted on links 2 and 4, respectively).

either zero or else it is a row of the matrix $T_t - S_t$.

By symmetry, at the beginning of slot t there is a packet with routing tag $x_{d-1} \cdots x_j \cdots x_0$ at each node, and this packet will be replaced (if transmitted) by a packet with routing tag $x_{d-1} \cdots \overline{x}_j \cdots x_0$ at the end of the slot t if $x_{d-1} \cdots \overline{x}_j \cdots x_0$ is nonzero and will exit the network otherwise. Thus the task matrix will change in the same way for each node. **Q.E.D.**

From Theorem 2 we see that if the communication task is isotropic with initial task matrix T_0, we can specify a symmetric routing algorithm by a sequence of symmetric switching schemes $S_0, S_1, \ldots$ as follows:

Symmetric Routing Algorithm Specification:

The initial task matrix T_0 of the isotropic task is given. For $t = 0, 1, \ldots$, given the task matrix T_t, S_t must be a symmetric switching scheme with respect to T_t; the task matrix T_{t+1} is then specified by the nonzero rows of $T_t - S_t$, unless $T_t = S_t$ in which case the algorithm terminates.

We see therefore that a symmetric routing algorithm that terminates after $k+1$ time slots amounts to a decomposition of the initial task matrix T_0 into a sum

$$T_0 = \overline{S}_0 + \overline{S}_1 + \cdots + \overline{S}_k,$$

where each $\overline{S}_i$, $i = 0, \ldots, k$, is a binary nonzero matrix with the same dimension as T_0, and with at most one nonzero element in each column or row. The corresponding switching schemes S_i, $i = 0, \ldots, k$, consist of the nonzero rows of the matrices $\overline{S}_i$, $i = 0, \ldots, k$, respectively.

Thus, by restricting attention to symmetric routings, our original problem of finding optimal

routings for isotropic communication tasks has been reduced to the simpler problem of "clearing" the T_0 matrix (i.e. making all its entries equal to 0) in a minimum number of steps. At each step we are allowed to make 0 up to d entries, provided that these entries do not belong to the same row or column. The entries should not belong to the same row because at each step a packet cannot be transmitted on more than one link. The entries should not belong to the same column so that no two packets will use the same outgoing link. We will derive optimal algorithms within this class. These algorithms will be shown to attain the lower bound of Theorem 1, so they are guaranteed to be optimal within the class of all routing algorithms.

2.4. OPTIMAL COMPLETION TIME ALGORITHMS

We consider the problem of clearing the task matrix in the minimum number of steps. At each step we are allowed to clear at most one entry from each row or column. Our analysis will use some theorems and tools that were also used in [BCW81] and [Wan88] in a different context. We first introduce some more definitions. For any matrix, we use the term *line* to refer to a row or column of the matrix.

Definition 5: A *perfect* matrix is a square matrix with nonnegative integer entries and with the property that the sum of the entries of each line is the same for all lines.

Definition 6: A *permutation matrix* is any matrix with entries equal to 0 or 1 with the property that each line of the matrix has at most one nonzero entry.

It can be noted that the nonzero entries of a permutation matrix form an independent set of entries in the sense that no two of them belong to the same line. As a result, a set of entries of the task matrix which form a permutation submatrix can be cleared during the same step. In particular a permutation matrix S can be used as a switching scheme for any node at any time as long as the task matrix at that node and time satisfies $S \leq T$ (see Definition 4). An important result for our purposes is Hall's Theorem (see [Rys65], and [Ber91] p. 120), which states that a perfect matrix can be written as a sum of h permutation matrices, where h is the sum of the entries of its lines. The following two theorems slightly extend Hall's Theorem.

Theorem 3: Given any nonnegative integer square matrix M with critical sum h, there exists a nonnegative integer matrix E such that $M + E$ is a perfect matrix with critical sum h.

Proof: We give a constructive proof. Let r_i (c_j) be the sum of the entries of row i (column j). We augment each element M_{ij} of the matrix such that $r_i < h$ and $c_j < h$ by $\min(h - r_i, h - c_j)$ one at a time and update M after each change, thus obtaining a matrix with at least one more critical line and line sum equal to h. At the end of this process we will have added to M a nonnegative integer matrix E, thereby obtaining a matrix $M + E$ with critical sum h and such that for each pair (i, j) either row i is critical or column j is critical. For this to be true, either all rows of $M + E$ must be critical or else all columns must be critical. Assume without loss of generality that all rows are critical. Then, the sum of the elements of $M + E$ is mh, where m is the number of rows and columns, while each column sum is at most h. It follows that each column sum of $M + E$ is exactly equal to h, so each column is critical, and $M + E$ is perfect. **Q.E.D.**

Theorem 4: A nonnegative integer matrix with critical sum h can be written as the sum of h permutation matrices.

Proof: Let T be a nonnegative integer matrix with dimensions $m \times d$. We assume, without loss of generality that $m \geq d$. We can extend T to a square matrix $M = [T \mid 0]$ by adding $m - d$ zero columns. Then, by Theorem 3, M can be augmented to a perfect matrix $M + E$ with line sums equal to h. By Hall's theorem we conclude that $M + E$ can be written as a sum $\sum_{k=1}^{h} P_k$ of h permutation matrices $P_1, P_2, \ldots P_h$. Since E has nonnegative integer entries, M can also be written as a sum $\sum_{k=1}^{h} \hat{P}_k$ of square permutation matrices $\hat{P}_1, \hat{P}_2, \ldots \hat{P}_h$; each $\hat{P}_k$ is obtained by setting to zero some of the entries of P_k. Since $M = [T \mid 0]$, T can be written as a sum of h permutation matrices of dimension $m \times d$. **Q.E.D.**

The following is the main result of this section.

Theorem 5: The optimal completion time for an isotropic communication task is equal to the critical sum h of its task matrix.

Proof: From Theorem 4 we know that the initial task matrix T_0 can be written as the sum $\sum_{k=1}^{h} \overline{S}_k$ of permutation matrices $\overline{S}_1, \overline{S}_2, \ldots, \overline{S}_h$. Consider the symmetric switching scheme $\{S_k\}$, where for $k = 1, \ldots, h$, S_k is obtained from $\overline{S}_k$ by removing the zero rows. Then the task matrix at times t with $1 \leq t < h$ consists of the nonzero rows of $T_0 - \sum_{k=1}^{t} \overline{S}_k$, and at

time $t = h$ is equal to Z. Hence the communication task is completed after h steps. Since, by Theorem 1, h is also an upper bound, the corresponding symmetric routing must be optimal. **Q.E.D.**

It is easy to see that if at any step we clear one entry from each critical line of the matrix T_t matrix, we can clear the task matrix within the optimal number of steps. On the other hand we cannot clear the matrix in h steps if we are not clearing an entry from each critical line at each step. In order to see this, let h_t be the critical sum of the task matrix T_t. We observe that the critical sum of the task matrix can decrease by at most 1 at each step ($h_t \geq h_{t-1}$-1). Thus, if during slot t there is a critical line which is not served, then $h_t = h_{t-1}$ and it is not possible to clear the matrix in $h_0 = h$ steps. Thus, we conclude that a symmetric switching scheme achieves optimal completion time if and only if it adheres to the following rule:

Optimal Completion Time Rule (abbreviated OCTR):

At each step an entry is cleared from each critical line of the task matrix.

We finally note that if the initial task matrix contains a column, say the j^{th}, which is critical, then the j-type links constitute a critical resource in the sense that they must all be used 100% of the time during the execution of any optimal completion time algorithm. Under these circumstances it is impossible to reduce the optimal completion time by using an algorithm that allows packets to be split and be recombined during its course. In the less usual case where the only critical lines are rows, the optimal completion time could be reduced under a different communication model, e.g. wormhole routing [KeK79], [DaS87].

We will now use the preceding results to find optimal algorithms for two isotropic tasks.

Total Exchange

In the total exchange task (see Subsection 1.3), we have initially $N-1$ packets with different routing tags queued at each node. The tags are different because each node has to send $N-1$ distinct packets, one to each node of the hypercube. The critical sum of the initial task matrix T_0, and therefore also the optimal completion time, is $N/2$. (To see this, note that if we add the $00\ldots00$ string as an N^{th} row of T_0, half of the entries of each column will be equal to 0 and half of them will be equal to 1.) Any algorithm that works according to the OCTR is optimal

as far as completion time is concerned, and there are at least $(\frac{N}{2})!$ of them.

(K, L) Neighborhood Exchange

In this task, every node s has to send a packet to all the nodes r whose Hamming distance from s satisfies $K \leq \text{Ham}(s,r) \leq L$. For $K = 1$ and $L = d$ we get the total exchange problem but for $K \neq 1$ and/or $L \neq d$, this task apparently has not been discussed elsewhere. The initial task matrix T_0 has as rows all the d-long binary strings with i ones, where $K \leq i \leq L$. The critical sum of this matrix is

$$h = \max\left(L, \sum_{i=K}^{L} \frac{\binom{d}{i} i}{d}\right).$$

To see this, note that for each i, $K \leq i \leq L$, the task matrix has $\binom{d}{i}$ rows with i ones. Since by symmetry the d columns have equal column sums, each column sum will be equal to $\sum_{i=K}^{L} \frac{\binom{d}{i} i}{d}$. By Theorem 5, the critical sum h is the time required to execute the task.

2.5. TRANSPOSITION OF BANDED MATRICES - AN EXAMPLE OF A NEARLY ISOTROPIC TASK.

Beginning with this section we focus on the problem of transposing a $2^{\beta+1} + 1$-diagonal matrix of size $N \times N$ stored by columns (or rows) in a hypercube of $N = 2^d$ processors. We propose an assignment of the columns of the matrix to the hypercube nodes that makes the transposition a "nearly isotropic" task, and present algorithms to execute the task.

A $2^{\beta+1} + 1$-diagonal matrix, where β is a nonnegative integer, is a matrix with entries a_{ij}, $i, j = 0, 1, \ldots, N-1$, such that $a_{ij} = 0$ whenever $2^\beta < |i - j| < N - 2^\beta$ (see Fig. 2.3). We assume that the matrix is stored in a hypercube of $N = 2^d$ nodes, so that each processor stores a column of the matrix. We are free to choose the way in which the columns are assigned to the processors. We are interested in assignments and corresponding communication algorithms that make the communication time required to transpose the matrix small.

In general, the problem of transposing a matrix stored by columns in a network of processors is equivalent to the total exchange communication task. This is because each processor i has to send the entry a_{ji} to processor j, for all i and $j \neq i$, which is a total exchange. The total

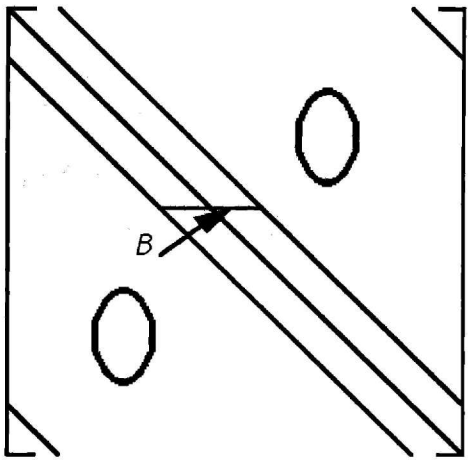

Figure 2.3: A B-diagonal matrix.

exchange task requires time $N/2$ and is the most communication intensive task among the prototype tasks described in Chapter 1. When, however, the matrix is banded the communication requirements become considerably less. Since the zero entries do not have to be communicated, the processor that stores the i^{th} column has to send a message to the processor that stores the j^{th} column only if $|i-j| \leq 2^{\beta}$ or $|i-j| \geq N - 2^{\beta}$. The sparsity pattern of banded matrices makes possible the use of transposition algorithms which are much faster than total exchange algorithms.

2.5.1. Column to Processor Assignment and Transposition Task Matrix

The communication requirements that arise during the transposition of a banded matrix depend on the way the columns are assigned to processors. If we do it in the natural way and assign column i to processor i, then the communication pattern that arises does not seem to have a structure that we can exploit to execute the communications fast. In this subsection we present an assignment that makes the transposition task "nearly isotropic". In the next subsection we will give an efficient communication algorithm to transpose the matrix.

We will use a kind of codes, called *Gray codes*, which are well known in Information Theory. A Gray code of length k is a sequence of 2^k distinct binary numbers of k bits each, with the property that successive numbers in the sequence differ in exactly one bit. Furthermore, the first and the last number in the sequence also differ in exactly one bit. A way to construct a

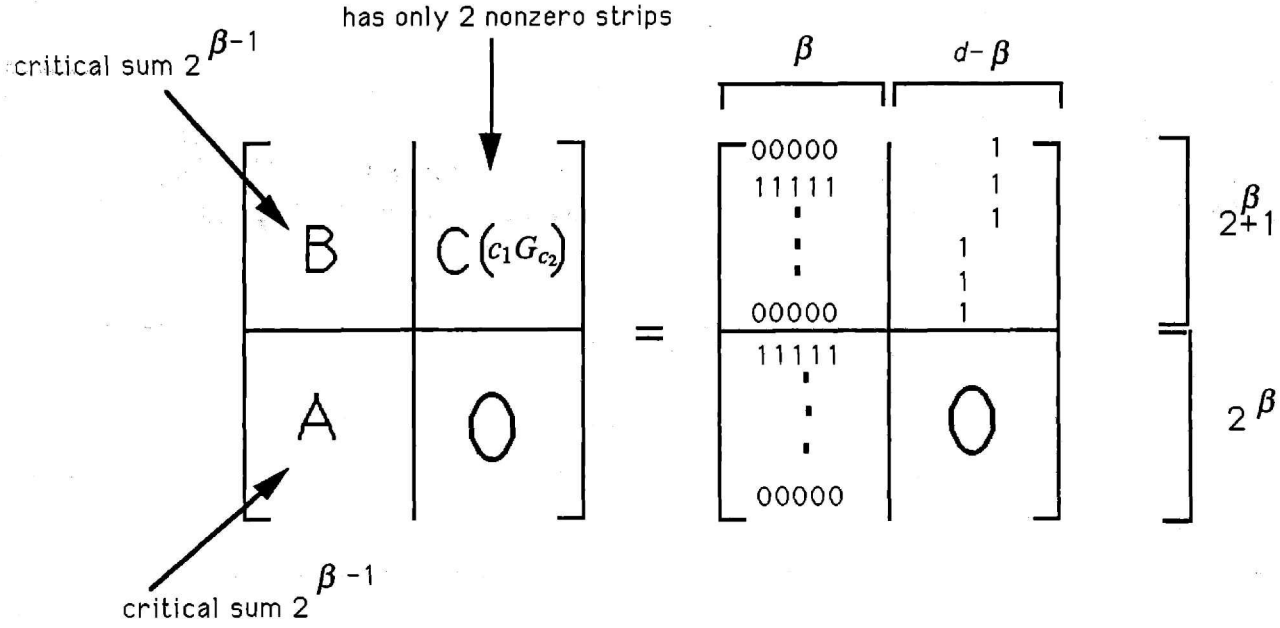

Figure 2.4: The initial task matrix $T(c_1G_{c_2})$ of node $c_1G_{c_2}$.

particular type of Gray code, called *reflected Gray code* (or RGS), is described in [BeT89].

Let $p(c)$ be the processor where column c is stored, and let c_1 (or c_2) represent the β least significant (or $d-\beta$ most significant, respectively) bits of c, that is, $c = c_2c_1$. The assignment that we propose consists of storing column c at the processor with binary representation

$$p(c) = c_1G_{c_2},$$

where G_{c_2} is the c_2^{th} number of the Gray code of length $d-\beta$. We call this assignment *Binary-Gray assignment.*

We denote by 0^β (or 1^β) the binary string of length β whose entries are all 0 (or 1, respectively). To transpose the matrix, processor $p(c)$ has to send a personalized packet to all processors $p(j)$ such that $|c-j| \leq 2^\beta$ or $|c-j| \geq N - 2^\beta$. Thus, processor $c_1G_{c_2}$ has to send a different packet to each one of the $2^{\beta+1}+1$ processors of the following set:

$$\begin{aligned}\mathcal{N}(c_1G_{c_2}) = \{&c_1G_{c_2\dot{-}1}, (c_1\ddot{+}1)G_{c_2\dot{-}1}, \ldots, 1^\beta G_{c_2\dot{-}1},\\ &0^\beta G_{c_2}, \ldots, c_1G_{c_2}, \ldots, 1^\beta G_{c_2},\\ &0^\beta G_{c_2\dot{+}1}, \ldots (c_1\ddot{-}1)G_{c_2\dot{+}1}, c_1G_{c_2\dot{+}1}\}.\end{aligned} \tag{2.3}$$

The operation $\dot{+}$ (or $\dot{-}$) refers to modulo $2^{d-\beta}$ addition (respectively, subtraction). The operation $\ddot{+}$ (or $\ddot{-}$) refers to modulo 2^β addition (respectively, subtraction).

Let $T(w)$ be the initial task matrix of node w that corresponds to the transposition task for the Binary-Gray assignment. The rows of the task matrix $T(c_1G_{c_2})$ are found by forming the bitwise exclusive OR operation between node $c_1G_{c_2}$ and the nodes in the set $\mathcal{N}(c_1G_{c_2})$ of Eq. (2.3). Ordering appropriately the routing tags, the task matrix $T(c_1G_{c_2})$ can be written in the form shown in Fig. 2.4. By convention, we let the exclusive OR operation between node $c_1G_{c_2}$ and the nodes $0^\beta G_{c_2}, \ldots, 1^\beta G_{c_2}$ of $\mathcal{N}(c_1G_{c_2})$ form the lower half part

$$\left[A(c_1G_{c_2})\ O_{2^\beta\times(d-\beta)}\right]$$

of the task matrix $T(c_1G_{c_2})$, where $O_{2^\beta\times(d-\beta)}$ represents the all zero $2^\beta \times (d-\beta)$ matrix. It can be seen that the submatrices $A(c_1G_{c_2})$ are the same for all nodes $c_1G_{c_2}$; we refer to them as submatrices A, dropping the subscript. The bitwise exclusive OR operation between $c_1G_{c_2}$ and nodes $c_1G_{c_2\dot{-}1}, (c_1\ddot{+}1)G_{c_2\dot{-}1}, \ldots, 1^\beta G_{c_2\dot{-}1}, 0^\beta G_{c_2\dot{+}1}, \ldots, (c_1\ddot{-}1)G_{c_2\dot{+}1}, c_1G_{c_2\dot{+}1}$ forms the upper half part

$$\left[B(c_1G_{c_2})\ C(c_1G_{c_2})\right]$$

of $T(c_1G_{c_2})$. Again it can be seen that the submatrix $B(c_1G_{c_2})$ is the same for all nodes $c_1G_{c_2}$; we refer to it as submatrix B omitting the subscript.

Both A and B have as rows all the binary strings of length β. However, he upper right submatrix $C(c_1G_{c_2})$ of $T(c_1G_{c_2})$ is different for each node. Since G_{c_2} differs from $G_{c_2\dot{-}1}$ and $G_{c_2\dot{+}1}$ in a single bit, submatrix $C(c_1G_{c_2})$ has only one nonzero element per row, and only two nonzero columns, called *strips*. The strips appear at different positions for each node.

Submatrices A and B will be referred to as the *isotropic part* of the transposition task matrix. If $C(c_1G_{c_2})$ were zero, the task matrix $T(c_1G_{c_2})$ would be identical for all nodes, and the Binary-Gray assignment would make the transposition task isotropic. Because of $C(c_1G_{c_2})$, however, the task matrices are not identical for all nodes; thus, our column-to-processor assignment has made the transposition task *nearly isotropic*. Nearly isotropic tasks are easy to handle, especially in view of the results of Sections 2.2-2.4.

2.5.2. The Transposition Algorithm

In this subsection we give an algorithm to transpose a banded matrix stored in a hypercube in the way described in Subsection 2.5.1. The algorithm will be strictly optimal for the particular column-to-processor assignment that we use (and for some other natural assignments). In

Section 2.6 we will show that the algorithm is of optimal order over all possible assignments for a broad range of matrix bandwidths.

If the task matrices were identical for all the nodes, a symmetric routing scheme could execute the task. The next lemma indicates a way to make the task matrices identical.

Lemma 1: If each packet that corresponds to a nonzero entry of the submatrix $C(c_1G_{c_2})$, $c_1 \in \{0,1\}^{\beta}, c_2 \in \{0,1\}^{d-\beta}$ is transmitted over its preferred link corresponding to the nonzero entry of $C(c_1G_{c_2})$ then the task matrices become identical for all nodes.

Proof: First, note that the lemma does *not* follow from any of the results for isotropic tasks of Sections 2.1 through 2.4 (for example, from Theorem 2), because neither the task matrices, nor the switching assignments corresponding to the transmissions mentioned in the lemma are the same for all nodes.

To prove the lemma note that node $c_1G_{c_2}$ receives a packet with routing tag t either from node $(c_1 \oplus t)G_{c_2+1}$ or from node $(c_1 \oplus t)G_{c_2-1}$. Thus, all the nodes receive a packet with routing tag t, for all $t \in \{0,1\}^{\beta}$, while at the same time they transmit all the packets which have different routing tags. **Q.E.D.**

The clearance of the C submatrices involves packet transmissions on links of dimensions $0, 1, \ldots, d-\beta-1$. On the other hand, clearing the isotropic part of the task matrices requires the use of dimensions $d-\beta, d-\beta+1, \ldots, d-1$ only. Thus, packet transmissions associated with entries of A can take place simultaneously with packet transmissions associated with entries of $C(c_1G_{c_2})$. The clearance of the submatrices $C(c_1G_{c_2})$ corresponds to packet transmissions on links of the form $(c_1G_{c_2}, c_1G_{c_2-1})$ and $(c_1G_{c_2-1}, c_1G_{c_2})$. For a specific c_1 and varying c_2 these links belong to the ring $c_1G_0, c_1G_1, \ldots, c_1G_{2^{d-\beta}}$. Therefore, the clearance of $C(c_1G_{c_2})$ involves communication among nearest neighbors on these rings. For $c_1 \neq \hat{c}_1$ the rings corresponding to c_1 and $\hat{c}_1$ are disjoint.

Since each processor has to send a total of $2^{\beta}+1$ packets to its neighbors on the ring (not all of them to the same one), the submatrix $C(c_1G_{c_2})$ can be cleared in 2^{β} steps. At the same time with $C(c_1G_{c_2})$, submatrix A can also start getting cleared by employing a symmetric routing scheme as described in Section 2.4. Since the critical sum of A is $2^{\beta-1}$, the new task matrix $\hat{T}(c_1G_{c_2})$ of node $c_1G_{c_2}$ after $2^{\beta-1}$ steps will be of the form illustrated in Fig. 2.5. By that step, submatrix A will have been cleared, and a new submatrix $\hat{A}$ will have been formed in its position. The rows of the lower part $\left[\hat{A}\ O\right]$ of the task matrix will be the former rows of

$[B\ C(c_1G_{c_2})]$ whose non-isotropic part was cleared. The rows of the upper part $\left[\hat{B}\ \hat{C}_{c_1G_{c_2}}\right]$ will be former rows of $[B\ C(c_1G_{c_2})]$ whose non-isotropic part has *not* been cleared yet.

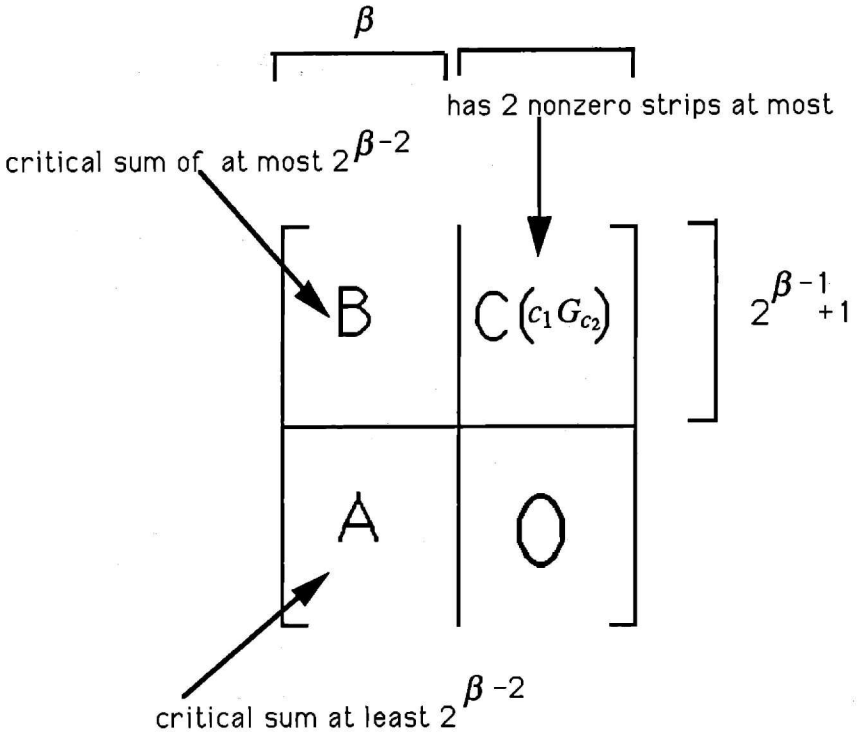

Figure 2.5: The task matrix $\hat{T}(c_1G_{c_2})$ of node $c_1G_{c_2}$ after step $2^{\beta-1}$.

When clearing $C(c_1G_{c_2})$ we insist on the following rule: the entries of $C(c_1G_{c_2})$ that are cleared at each step are those that correspond to routing tags with the largest number of ones. In this way isotropic work is created at the largest possible rate to keep the links of dimensions $d-\beta,\ldots,d-1$ busy. With this rule, the task matrix $\hat{C}(c_1G_{c_2})$, at the end of the $2^{\beta-1}$th step will have at most $2^{\beta-1}$ ones; during the same step the critical sum of $\hat{A}$ will be at least $2^{\beta-2}$, and the critical sum of $\left[\hat{B}\ \hat{A}\right]'$ will be $2^{\beta-1}$. Following this rule for a total of $2^{\beta}-1$ steps, the task matrices takes the form illustrated in Fig. 2.6. One additional step is then required to finish the task. Thus we have shown the following theorem:

Theorem 6: The time required to transpose a $2^{\beta+1}+1$-diagonal $N \times N$ matrix stored by columns (or rows) in an N-processor hypercube in the way described above, is equal to 2^{β} steps.

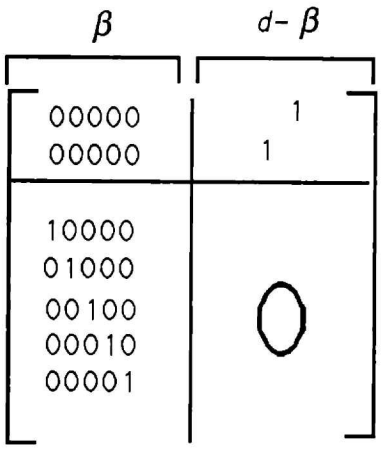

Figure 2.6: The task matrix of node $c_1 G_{c_2}$ at time $2^\beta - 1$.

In the next section we find a lower bound on the time required to transpose a banded matrix under any possible assignment of columns to hypercube processors.

2.6. A UNIVERSAL LOWER BOUND ON THE MINIMUM TIME TO TRANSPOSE A BANDED MATRIX

In the previous section we described a way to store a $2^{\beta+1}+1$- diagonal matrix by columns in a hypercube, and we gave a communication algorithm that executes the transposition task in $T = 2^\beta$ steps. Using Theorem 1, which holds for both isotropic and non-isotropic tasks, we can see that the given algorithm is optimal for the Binary-Gray assignment of columns to processors. The question that arises is whether there exists another column-to-processor assignment that results in a faster transposition algorithm.

For $\beta = 0$ (tridiagonal matrix) our algorithm requires a single step, and is optimal for any assignment of columns to processors. For $\beta = d - 1$ (full matrix) we get $T = N/2$, which is again optimal for any assignment of columns to processors. In what follows we will examine the efficiency of the proposed algorithm for values of the bandwidth that are between these two extremes. We will derive a universal lower bound on the minimum number of steps required to transpose a banded matrix, which holds for any possible assignment of matrix columns to hypercube nodes. When $B = \Theta(N^c)$ for some constant $c > 0$, the universal bound will turn out to be of the same order of magnitude as the completion time of our algorithm. The case $c = 1/2$

is particularly interesting since the discretization of elliptic partial differential equations in two dimensions by finite element or finite difference methods leads to matrices with bandwidth about $gN^{1/2}$, where g is the degree of the finite elements used for the discretization ([McV87]). Our algorithm is obviously of optimal order when $B = O(1)$, which is another practical case. We have not proved optimality of our algorithm with respect to all possible embeddings for all bandwidths B. The efficiency, however, of our algorithm for the two most extreme cases, and a broad range of intermediate cases suggests that it is a practical one.

In what follows we prove a lower bound on the completion time of a banded-matrix transposition algorithm that holds for all possible embeddings. Let $p(c)$ represent the hypercube node which stores column c. Suppose that there is a way to assign columns to nodes so that for each column c, the $B-1$ columns j that satisfy $|c-j| \leq 2^\beta$, or $|c-j| \geq N-2^\beta$ are assigned to $B-1$ hypercube nodes which are closest to node $p(c)$. Such an assignment, although not always possible, would result in a minimum number of packet transmissions, and therefore it can provide a lower bound on the minimum number of steps required to transpose a banded matrix for any possible column to processor assignment.

Let r be the distance from node $p(c)$ to the farthest of its $B-1$ closest nodes. Then

$$\sum_{i=0}^{r-1} \binom{d}{i} \leq B \leq \sum_{i=0}^{r} \binom{d}{i}.$$

This relation gives

$$\frac{B}{N} \leq \sum_{i=0}^{r} \binom{d}{i} \left(\frac{1}{2}\right)^d = \sum_{i=d-r}^{d} \binom{d}{i} \left(\frac{1}{2}\right)^d, \tag{2.4}$$

or

$$\frac{B}{N} \leq \mathcal{B}(d-r, 1/2, d),$$

where $\mathcal{B}(m,p,K)$ is the probability that we get more than m successes in K independent Bernoulli trials with p being the probability of success for each trial. The Chernof bound gives (see e.g. [VaB81]) that

$$\frac{B}{N} \leq \left(\frac{d}{2(d-r)}\right)^{d-r} \left(\frac{d}{2r}\right)^r, \quad \text{for } r < d/2,$$

or

$$B \leq \frac{d^d}{(d-r)^{d-r} r^r}, \quad \text{for } r < d/2.$$

Letting $r = \lambda d$, the previous relation is transformed to

$$B \leq \frac{1}{(1-\lambda)^{d-r}\lambda^r} = \left(\frac{1}{(1-\lambda)^{1-\lambda}\lambda^\lambda}\right)^d, \quad \text{for } \lambda < 1/2,$$

which yields

$$\frac{\log_2 B}{d} \leq H(\lambda),$$

with

$$H(\lambda) = -\lambda \log_2 \lambda - (1-\lambda)\log_2(1-\lambda)$$

being the entropy (base 2) function (see Fig. 2.7). If we restrict $H(\lambda)$ to $\lambda < 1/2$, then H^{-1} is well defined and monotonically increasing. Therefore, we can write

$$r = \lambda d \geq \min\left(dH^{-1}\left(\frac{\log_2 B}{d}\right), d/2\right).$$

In the case where $B = N^c$ for some $c > 0$, we get that

$$r \geq \min\left(dH^{-1}(c), d/2\right) = \Theta(d), \quad c > 0. \tag{2.5}$$

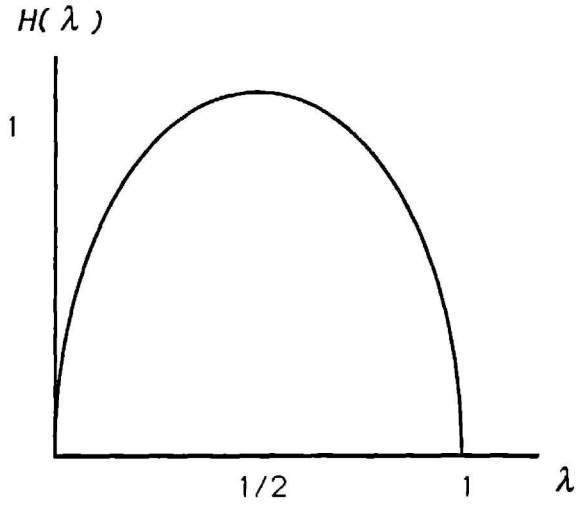

Figure 2.7: The entropy function $H(\lambda)$.

The following lemma gives a lower bound on the number of transmissions required to transpose a banded matrix.

Lemma 2: Let W be the total number of transmissions required for the packets that are sent (or received) by the node $p(c)$ in order to arrive at their destination. Then

$$W = \Omega(Br), \quad \text{for } r > 1.$$

Proof: The mean number of transmissions required by the packets sent by processor $p(c)$ satisfies

$$\begin{aligned}\frac{W}{B} &\geq \frac{\sum_{j=0}^{r-1} j \frac{d!}{j!(d-j)!}}{\sum_{j=0}^{r-1} \binom{d}{j}} \\ &= \frac{d \sum_{j=0}^{r-1} \binom{d-1}{j-1}}{\sum_{j=0}^{r-1} \binom{d}{j}} \\ &= \frac{d \sum_{j=0}^{r-2} \binom{d-1}{j}}{\sum_{j=0}^{r-1} \binom{d}{j}} \\ &= \frac{d}{2} \frac{\sum_{j=0}^{r-2} \binom{d-1}{j} \left(\frac{1}{2}\right)^{d-1}}{\sum_{j=0}^{r-1} \binom{d}{j} \left(\frac{1}{2}\right)^{d}}.\end{aligned}$$

Let $X^{(d)}$ be the sum of d independent Bernoulli random variables with mean 0.5, and let $X^{(d-1)}$ be the sum of the first $d-1$ of them. Then

$$\frac{W}{B} \geq \frac{d}{2} \frac{\Pr\left(X^{(d-1)} \leq r-2\right)}{\Pr\left(X^{(d)} \leq r-1\right)} \geq \frac{d}{2} \Pr\left(X^{(d-1)} \leq r-2 \mid X^{(d)} \leq r-1\right).$$

The conditional probability in the preceding equation is always greater than or equal to $(r-1)/d$. This is because if $X^{(d)} < r-1$ then $X^{(d-1)} \leq r-2$ always, while if $X^{(d)} = r-1$ then $X^{(d-1)} \leq r-2$ with probability $(r-1)/d$ [given that we had exactly $r-1$ successes in d independent trials, the probability that the last trial was a success is $(r-1)/d$]. Combining the previous equations we get that

$$W \geq \frac{B(r-1)}{2} = \Omega(Br), \quad \text{for } r > 1.$$

Q.E.D.

Since the packets sent by a node require a total of $\Omega(Br)$ transmissions, and each node has d links, a lower bound on the minimum time T required to transpose the banded matrix is

$$T = \Omega\left(\frac{Br}{d}\right). \tag{2.6}$$

Equation (2.6) holds when $r > 1$, or else $B > \log N$. Combining Eqs. (2.5) and (2.6) we obtain the following theorem.

Theorem 7: When $B = \Theta(N^c)$ for some $c > 0$, then

$$T = \Omega(B) = \Omega(N^c).$$

The transposition algorithm of the previous section requires $(B-1)/2 = O(B)$ steps. Theorem 7 shows that when $B = N^c$ our algorithm is of the optimal order of magnitude under

any possible assignment of columns to processors. In particular, the ratio of the lower bound to the complexity of our algorithm is roughly $\min(0.5, H^{-1}(c))$. Theorem 7, combined with the optimality of our algorithm in the cases $B = 1$ and $B = N$, is an indication of the algorithm's efficiency.

The next section deals with the storage and the concurrent transposition of several banded matrices in a hypercube. It will be shown that performing more than one banded matrix transpositions simultaneously increases the link utilization and the efficiency of our transposition algorithm when $\lfloor d/\beta \rfloor > 2$ (or equivalently, $B \leq N^{1/2}$).

2.7. SEVERAL SIMULTANEOUS BANDED MATRIX TRANSPOSITIONS

In this section we present a way to store $\lfloor d/\beta \rfloor$ matrices each of bandwidth $B = 2^{\beta+1}+1$ in a hypercube, and transpose them simultaneously in $B = 2^{\beta+1}+1$ steps. This does not contradict Theorem 7 since in the case $B = N^c$ we have $d/\beta \approx c$, and the improvement in efficiency is a constant factor. If B is of smaller order of magnitude [e.g., if B is $\Theta(1)$, or $\Theta(d)$] then $d/\beta \rightarrow \infty$ as $d \rightarrow \infty$, and the improvement is even more significant.

The main idea of the section can be summarized as follows. The tranposition algorithm of Section 2.5 uses mainly β hypercube dimensions, say dimensions $d-1, \ldots, d-\beta$. Thus, a second banded matrix can be stored in the hypercube so that its transposition uses mainly the dimensions $d-\beta-1, \ldots, d-2\beta$, and this can be extended to a total of $\lfloor d/\beta \rfloor$ matrices. Of course, in this case, it may no longer be possible to pipeline the isotropic with the non-isotropic part of the task. This results in an increase of the completion time by a factor of two, which is offset by the improvement in efficiency if $\lfloor d/\beta \rfloor > 2$.

The matrices are stored in the hypercube in the following way. Let $< \pi_0, \pi_1, \ldots, \pi_{d-1} >$ be a permutation of $\{0, 1, \ldots, d-1\}$. Given a binary number s of length d let $< \pi_0, \pi_1, \ldots, \pi_{d-1} > (s)$ be the binary number whose π_i^{th} bit is equal to the i^{th} bit of s. Let $L_j := Left_j(< 0, 1, \ldots, d-1 >)$ be the permutation obtained from $< 0, 1, \ldots, d-1 >$ by cyclically shifting it j positions to the left, and let c_1, c_2 and G_{c_1} be defined as in Section 2.5. We assign column c of the j^{th} banded matrix to the hypercube node

$$L_{j\beta}(c_1 G_{c_2}), \quad j = 1, 2, \ldots \left\lfloor \frac{d}{\beta} \right\rfloor, \quad l = 0, 1, \ldots, N-1.$$

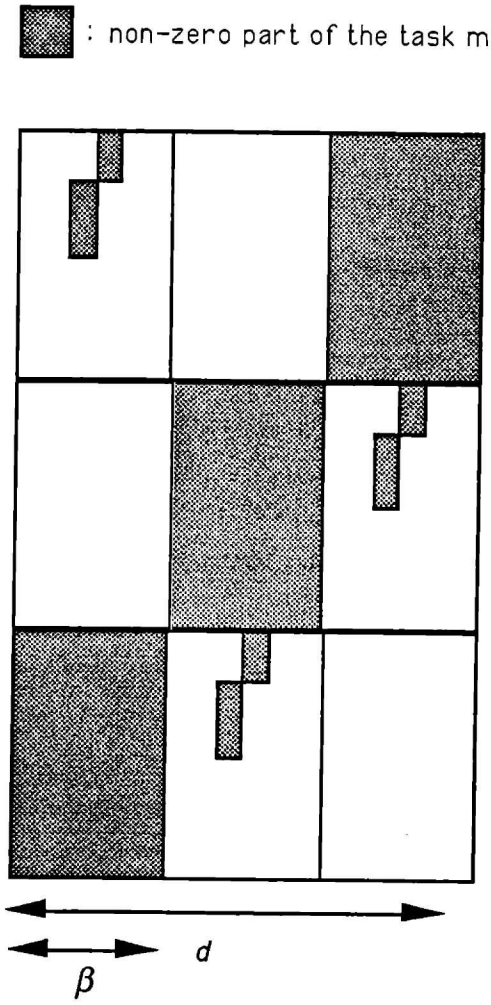

Figure 2.8: A typical task matrix at a node for the case $d/\beta = 3$.

The transposition is viewed as a single task and it is done simultaneously for all matrices. A typical initial task matrix is shown in Fig. 2.8. The transposition algorithm consists of two phases, which cannot in general overlap. In the first phase the isotropic part of the task matrices is cleared; this requires 2^β steps. This is possible because the isotropic part of the task matrix which corresponds to the j^{th} matrix requires the use of dimensions $d-(j-1)\beta-1,\ldots,d-j\beta$ of the hypercube, and does not interfere with the transposition of the other banded matrices. In the second phase the non-isotropic part is cleared, which requires at most $2^\beta+1$ additional steps. Thus, the total time T required to transpose $\lfloor d/\beta \rfloor$ banded matrices, each of bandwidth $B = 2^{\beta+1}+1$, is

$$T = 2^{\beta+1}+1 = B, \quad \text{for } \left\lfloor \frac{d}{\beta} \right\rfloor \text{ concurrent banded matrix transpositions.}$$

Remark 1: The results can be extended to the case where we have banded matrices with different bandwidths $B_k = 2^{\beta_k} + 1,\ k = 0, 1, \ldots, p$, where $\sum_{k=1}^{p} \beta_k \leq d$.

Remark 2: If we want to perform a multinode broadcast (or a multinode accumulation) in each row involving only nodes whose columns fall within the same band (a standard operation in many computation algorithms), then by using the embedding desribed earlier, the broadcasts (or accumulations) can be executed for all the $\lfloor d/\beta \rfloor$ banded matrices in time $\lceil B/\beta \rceil + 1$.

2.8. EMBEDDING SPARSE GRAPHS IN HYPERCUBES

A problem that arises in the design of algorithms for parallel computers is the mapping of a guest graph to the underlying graph of the host machine. The problem has been studied extensively in the literature for hypercube parallel computers (see [Lei92a] and references therein) with partial success except for special cases of the guest graphs (e.g., meshes, trees, and rings; see [BeT89], [JoH89]). The general case is considered a difficult problem, and there are few results available even for rather restricted classes of guest graphs. In this section we describe some preliminary ideas for the case where the guest graph is sparse. Since most large graphs are sparse, the importance of this problem is evident.

We define the *arc matrix* A corresponding to graph G as the binary matrix whose $(i,j)^{\text{th}}$ entry is equal to one if i and j are connected through an arc in G, and zero otherwise. Since the guest graph G is sparse the matrix A is also sparse. The pattern of the nonzero entries of A depend on the labelling of the nodes of G. Our proposal is to label the nodes so that A becomes a banded matrix with bandwidth as small as possible (relabelling of nodes corresponds to row and column permutations in A). A matrix $A = (a_{ij})$ is banded with bandwidth B if $a_{ij} = 0$ whenever $|i - j| > (B-1)/2$ and there is no smaller B with this property. The problem of reducing the bandwidth of a matrix is also a difficult problem, but it has been studied extensively in the literature. A variety of methods (mainly heuristics) that decrease the bandwidth (or the "average bandwidth") of sparse matrices are available ([Pis84]). Any of these algorithms can be applied to the arc matrix of G. Let $L(A)$ be the arc matrix of G after the relabelling of the nodes, and let B be its bandwidth. Then we can embed matrix $L(A)$ to the hypercube in the way given in Section 2.5. The dilation of the resulting embedding of the graph to the hypercube is equal to $\log_2 B$.

2.9. EMBEDDING BLOCK DIAGONAL MATRICES IN HYPERCUBES

In this section we present a way to store a block diagonal matrix of size $2^d \times 2^d$ in a d-dimensional hypercube H_d. We assume that the matrix has K blocks, denoted by $B_1, B_2, \ldots, B_K$ and each block B_i has size $2^{l_i} \times 2^{l_i}$.

Each block B_i is embedded in a l_i - dimensional subcube H_{l_i} of H. For the embedding to be 1-1 (one column of the matrix to one processor) the subcubes H_{l_i} must be disjoint. The problem is equivalent to partitioning a hypercube with 2^d nodes in hypercubes $H_1, \ldots, H_K$ of sizes $2^{l_1}, \ldots, 2^{l_K}$, respectively, with

$$2^d = \sum_{i=1}^{K} 2^{l_i}. \tag{2.7}$$

For any string s of length $d - l$, we let

$$(*^l s)$$

represent the l-dimensional subcube of H_d obtained by fixing the $d - l$ last bits to be equal to s. The embedding that we propose is the following. The block B_i of the matrix is mapped to subcube

$$H_{l_i} = (*^{l_i} x_i),$$

where x_i is a binary string of length $d - l_i$. The subcubes H_{l_i} are disjoint if and only if the strings x_i, $i = 1, 2, \ldots, K$ satisfy the *prefix condition*, that is, no string is a prefix of any other string. To show that we can choose binary strings x_i of length $d - l_i$, $i = 1, 2, \ldots, K$, so that the prefix condition is satisfied let $d_i = d - l_i$. Equation (2.7) gives

$$1 = \sum_{i=1}^{K} 2^{-d_i}.$$

Thus, Kraft's inequality holds. This shows that there exists a set of strings of lengths $d_i, i = 1, 2, \ldots, K$, which satisfy the prefix condition. Gallager [Gal68] (p. 48) describes a way to obtain such a set of strings.

CHAPTER THREE

PMNB and PE Algorithms for Hypercubes and Meshes

In this chapter we consider the partial multinode broadcast and the partial exchange communication tasks in hypercubes and d-dimensional meshes. The partial multinode broadcast in an N-processor network is the task in which each of $M \leq N$ arbitrary nodes broadcasts a packet to the remaining $N-1$ nodes. Correspondingly, in the partial exchange there are $M \leq N$ nodes which wish to send a separate, personalized packet to each of the other nodes. We propose algorithms for the hypercube and the d-dimensional mesh networks that execute the partial multinode broadcast and the partial exchange in near-optimal time. No assumption concerning the location of the M source nodes is made. All the communication algorithms proposed are "on line" and distributed.

3.1. INTRODUCTION

Two of the most frequent communication tasks are the *multinode broadcast* (MNB) and the *total exchange* (TE). The first task involves broadcasting a packet (the same packet) from every node to all the other nodes. It arises, for example, in iterations of the form

$$x = f(x), \tag{3.1}$$

where each processor computes an entry (or some entries) of the vector x. At the end of each iteration it is necessary that each processor broadcasts the updated value of the component that it computes to all other processors in order to be used at the next iteration.

The total exchange (see also Chapter 2) is the communication task where each node has to send a *personalized* (different) packet to each one of the other nodes. An example where the total exchange arises is the transposition of a matrix, when each processor stores, say, a column of the matrix. Then every processor i has to send the $(i,k)^{\text{th}}$ entry of the matrix to processor k, for all k, which is a total exchange.

In iterations of the kind given in Eq. (3.1) it is probable that only some of the components of the vector x change appreciably during an iteration. As these iterations approach their convergence point, fewer and fewer of the processors need to broadcast the updated values of the components of x that they compute. This gives rise to a task, where a strict (but unpredictable) subset of the processors have to broadcast a packet. We call this task a *partial multinode broadcast* (or PMNB for brevity). The PMNB task, aside from being important on its own merit, is also a critical subroutine of the dynamic broadcast schemes that we will propose in the next chapter. The PMNB task arises also in clustering algorithms (see [RaS90], Chapter 5, where the M nodes that store the coordinates of the centers of the clusters broadcast them after each iteration), and other problems. Because of its many applications we believe that the PMNB deserves a position among the prototype tasks of a communication library.

Similarly, during the transposition of a matrix that has both sparse and dense columns, it is more efficient if the nodes storing sparse columns do not participate in the TE, but send instead their packets as ordinary traffic through the 1-1 routing algorithm used by the machine. Since most large problems involve sparse matrices one can see that this situation arises frequently, giving rise to the *partial exchange task* (PE), where only M nodes send a (separate) packet to every other node. In other words, a partial exchange is the simultaneous execution of a single node scatter (see Section 1.3) by M arbitrary nodes. A task which is dual to the PE is the *partial multinode gather* (PMNG) task. In this task, M arbitrary nodes have to receive a (different) packet from every other node of the network (combining packets originated at different nodes is not allowed). Note that the PMNG task is *dual* to the PE task; if we find an algorithm to execute the PE we immediately get an algorithm of the same time complexity that executes the PMNG. In the transposition of a matrix stored by columns in a multiprocessor network, a PMNG arises when the matrix has only M dense rows. By combining a PE and a PMNG algorithm we get an algorithm that transposes a matrix which has M_1 dense columns and M_2 dense rows. The dense rows and columns can be arbitrary. This sparsity pattern arises very frequently in applications. The smaller M is, the less efficient a full MNB or TE algorithm would be and the more necessary it becomes to employ algorithms that are specially designed

for partial tasks.

The main focus of the chapter is to propose optimal and near-optimal communication algorithms for the partial multinode broadcast and the partial exchange tasks in hypercubes, and d-dimensional meshes with or without wraparound. PMNB algorithms for hypercubes have previously been studied in [Sta91]. Our hypercube PMNB algorithm has roughly half the time complexity of the one given in [Sta91]. The PMNB problem for d-dimensional meshes, and the partial exchange problem for both hypercubes and meshes, are considered for the first time here. In the course of solving the mesh PMNB problem, we formulated and solved two problems, called the mesh packing and monotone routing problems, which are of broader interest. In what follows, to avoid confusion, we call a (d-dimensional) mesh *with* wraparound a *torus* and a mesh *without* wraparound an *array*.

We will say that an algorithm is *near-optimal* if the potential loss of optimality with respect to completion time is of strictly smaller order of magnitude than the optimal completion time itself. We generally prove that an algorithm is near-optimal by showing that the leading term of its worst case time complexity (including the corresponding constant factor) is the same as the leading term of an expression which is a lower bound to the time required by any algorithm. We generally derive the optimal completion time by deriving a lower bound to the completion time of any algorithm and by constructing an algorithm that attains the lower bound; this latter algorithm is said to be *optimal*. We will say that an algorithm is of *optimal order* if its worst case time complexity is asymptotically within a constant factor of the optimal value.

One of the main contributions of the chapter is the development of near-optimal algorithms for a partial multinode broadcast in a hypercube, in a d-dimensional torus, and in a d-dimensional array. We propose algorithms for two different communication models. In the first model, packets can be split and recombined at the destination without any overhead. In the second model the splitting is not allowed, and messages are always transmitted as one packet. We also present the first partial exchange algorithm of optimal order for hypercubes, and 2-dimensional arrays.

The organization of the chapter is the following. Section 3.2 shows how a mesh without wraparound can simulate a mesh with wraparound, and presents the first (strictly) optimal multinode broadcast algorithm for 2-dimensional meshes without wraparound. We also present a theorem concerning arbitrary broadcasts in rings and linear arrays. In Section 3.3 we define and solve the packing and the monotone routing problems, in a d-dimensional mesh. In Section 3.4 we present near-optimal algorithms to execute a partial multinode broadcast in d-

dimensional meshes. In particular, in Subsection 3.4.1 we give an algorithm where packets can be split, while in Subsection 3.4.2 we give an algorithm that avoids the splitting of packets. In Section 3.5 we present an algorithm of optimal order for a partial exchange in a 2-dimensional array. Section 3.6 describes three different algorithms to execute a partial multinode broadcast in a hypercube. The first algorithm is a practical, but suboptimal one. The second and third algorithms are optimal, and each of them uses a different communication model. In Section 3.7 we present a partial exchange algorithm of optimal order for the hypercube. Section 3.8 deals with a new communication task for hypercubes, which we call *window multinode broadcast.*

3.2. SOME PRELIMINARY RESULTS

The d-dimensional mesh, denoted by M_d, consists of $N = p^d$ processors arranged along the points of a d-dimensional space that have integer coordinates numbered from 0 to $p-1$. Along the i^{th} dimension, obtained by fixing coordinates $(x_{d-1},\ldots,x_{i+1},x_{i-1},\ldots,x_0)$ there are p processors with identities $(x_{d-1},\ldots,x_i,\ldots,x_0)$, $x_i = 0,1,\ldots,p-1$. Two processors $(x_{d-1},\ldots,x_i,\ldots,x_0)$ and $(y_{d-1},\ldots,y_i,\ldots,y_0)$ are connected by a (two-directional) link if and only if for some i we have $|x_i - y_i| = 1$ and $x_j = y_j$ for all $j \neq i$. In addition to these links in the d-dimensional mesh *with wraparound* (also called a *torus*), all links of the type

$$((x_{d-1},\ldots,x_{i+1},0,x_{i-1},\ldots,x_0),\ (x_{d-1},\ldots,x_{i+1},p-1,x_{i-1},\ldots,x_0))$$

are present. The latter links do not exist in the d-dimensional mesh *without* wraparound (also called an *array*). The set of nodes of an array (or torus) whose identities differ from the identity of node $x = (x_{d-1},\ldots,x_{i+1},x_{i-1},\ldots,x_0)$ only in the i^{th} digit is called the *i-level linear array* (or *ring*, respectively) of node x, and is denoted by $(x_{d-1},\ldots,x_{i+1},*,x_{i-1},\ldots,x_0)$. The node with identity $(x_{d-1},x_{d-2},\ldots,x_0)$ is also represented by the base p number of the form $x = x_{d-1}x_{d-2}\cdots x_0$. The 0th digit is considered the least significant digit of the above representation. A link connecting two nodes which differ only in the i^{th} digit is called a *link of dimension i.*

Packets can be simultaneously transmitted along a link in both directions. Only one packet can travel along a link in each direction at any one time; thus, if more than one packet are available at a node and are scheduled to be transmitted on the same incident link of the node,

then only one of these packets can be transmitted at the next time period, while the remaining packets must be stored at the node while waiting in queue. Each node is assumed to have infinite storage space. All incident links of a node can be used simultaneously for packet transmission and reception. Each packet requires one unit of time for transmission over a link. We consider both a model where packets can be split at the origin, and be recombined at the destination, and a model where packets cannot be split; in the first model if a packet is split in d parts, each of them requires $1/d$ units of time to be transmitted over a link.

We start by describing how a torus can be simulated by an array. A linear (i.e. one-dimensional) array can simulate a ring of the same size with a slowdown factor of two. This can be done as indicated in Fig. 3.1. By using this fact, a torus of any dimension can be simulated by an array of the same size and dimension with a slowdown factor of two as shown again in Fig. 3.1.

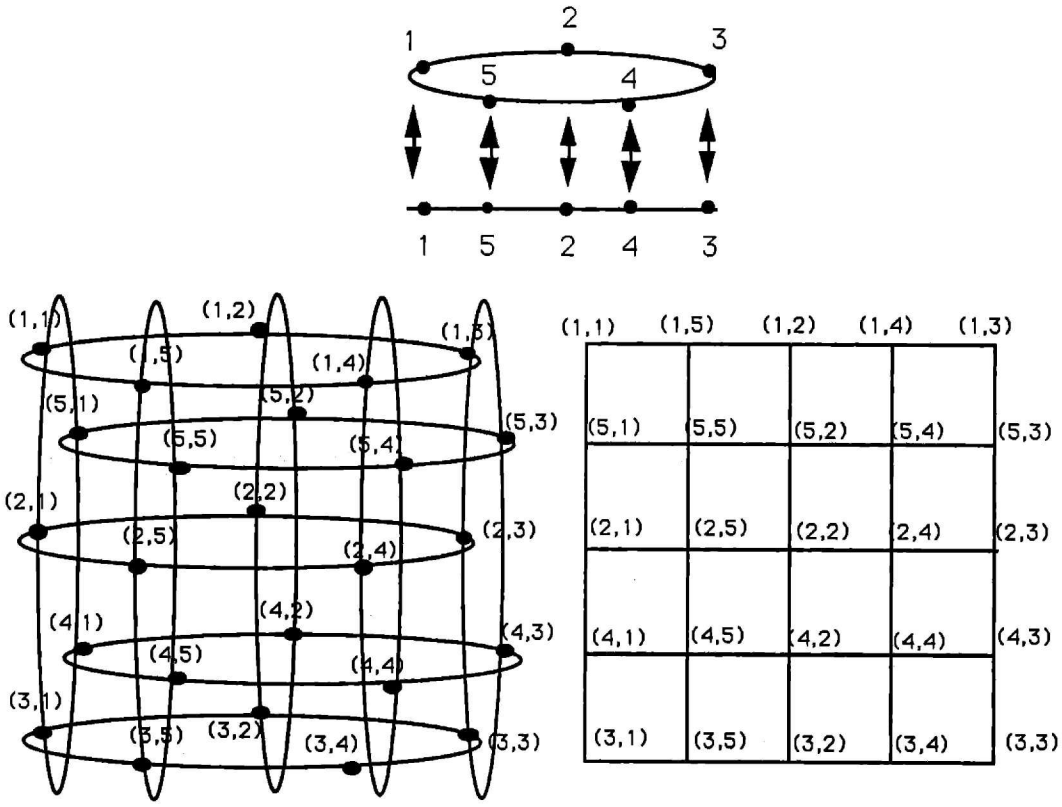

Figure 3.1: The upper part of the figure shows how a ring can be simulated by a linear array with a factor of two slowdown. This idea is easily extended to the simulation of a d-dimensional torus by a d-dimensional array, as can be seen from the lower part of the figure.

The optimal time T^t_{MNB} to execute a (full) multinode broadcast in a $p \times p$ torus was found in [BeT89] to be equal to

$$T^t_{MNB} = \frac{p^2}{4} = \frac{N}{4}$$

if p is even and

$$T^t_{MNB} = \frac{p^2 - 1}{4} = \frac{N - 1}{4}$$

if p is odd.

The following theorem gives a corresponding result for the MNB task in a 2-dimensional array.

Theorem 1: The minimum time T^a_{MNB} required to execute a (full) multinode broadcast in a 2-dimensional array is exactly twice the minimum time T^t_{MNB} required to execute a multinode broadcast in a 2-dimensional torus of the same size, that is

$$T^a_{MNB} = 2T^t_{MNB} = \left\lfloor \frac{N}{2} \right\rfloor . \tag{3.2}$$

Proof: As we indicated earlier, a mesh without wraparound can simulate with a slowdown factor of two a mesh with wraparound of the same size. Each step of a torus can be simulated in two steps by an array even if all the links of the torus are simultaneously used. This gives the inequality $T^a_{MNB} \leq 2T^t_{MNB}$. Since node $(0, 0)$ has only two neighbors and receives $N - 1$ packets we have $T^a_{MNB} \geq (N - 1)/2$. This together with the fact that T^a_{MNB} has to be integer proves that

$$T^a_{MNB} = 2T^t_{MNB} = \left\lfloor \frac{N}{2} \right\rfloor .$$

Q.E.D.

The next theorem deals with arbitrary broadcasts in rings and arrays.

Theorem 2: Consider a linear (one-dimensional) array of p nodes, where each node has a certain (not necessarily the same) number of packets to broadcast to all other nodes. Let K be the total number of packets in the array. Then the broadcasts can be completed in time less than or equal to

$$K + p - 1.$$

In a ring of the same size, the task requires half this time, provided that packets can be split into two parts without additional overhead.

Proof: Consider the following algorithm. Each node immediately transmits over its left (right) neighbor every packet that it receives from its right (left) neighbor. Whenever, a node does not receive anything from its left (right) neighbor it sends one of its own packets to the right (left). In other words, each node passes in the same direction the packets that come to it, and inserts a packet of its own whenever it sees an empty slot. Note that a packet is never delayed after it starts getting transmitted. In order to evaluate the time complexity we can focus on one direction, say the one going from left to right. Since there are K packets in the linear array, the packet can be delayed at most K times before starting transmission in this direction, and after at most $p-1$ slots it will have arrived to all the nodes in that direction. To prove the result about the ring, we can split each packet in two parts, each requiring 0.5 units of time. The ring can be viewed as two edge-disjoint unidirectional linear arrays, and by similar arguments, applied to each direction, the result follows. **Q.E.D.**

Remark: In the case where each node of the linear array has at most one packet, all the nodes can broadcast their packet in time $p-1$ (see [BeT89]). In the case of a ring the same task requires time $\lfloor p/2 \rfloor$, if the packets cannot be split, and $(p-1)/2$ if the packets can be split into two parts without overhead.

3.3. PACKING AND MONOTONE ROUTING FOR MESHES

In this section we present some new results on routing in meshes. These results will be useful in the PMNB algorithms to be given later, but they are also interesting on their own right. The problems to be addressed will be referred to as the *packing* and the *monotone routing* problems. We expect these results to be useful in a variety of algorithms, given the wide use that corresponding results for butterfly networks have had (see, e.g., [Lei92a], pp. 524-538).

Theorem 3 (Mesh Packing Routing Theorem): Let $s^{(i)}$, $i = 0, 1, \ldots, K-1$, be nodes of a d-dimensional array such that $s^{(0)} < s^{(1)} < \cdots < s^{(K-1)}$. Consider the communication task, where each node $s^{(i)}$ sends a packet to processor i. This can be done without conflicts through a greedy scheme in time $d(p-1)$. This greedy scheme uses only links of dimension j during steps $jp, jp+1, \ldots, jp+p-1$, $j = 0, \ldots, d-1$.

Proof: The greedy routing consists of d phases, each of which has duration *exactly* $p-1$ steps. During phase l, $l = 0, 1, \ldots, d-1$, the packet generated at node $s^{(i)}$ corrects its l^{th} digit to be equal to the l^{th} digit of i by crossing in the natural way the links of dimension l. Thus, at the beginning of phase l the packet is at node $s^{(i)}_{d-1}s^{(i)}_{d-2}\cdots s^{(i)}_{l}i_{l-1}i_{l-2}\cdots i_0$, and at the end of phase l the packet is at node $s^{(i)}_{d-1}s^{(i)}_{d-2}\cdots s^{(i)}_{l+1}i_l i_{l-1}\cdots i_0$, where $s^{(i)}_{d-1}s^{(i)}_{d-2}\cdots s^{(i)}_{0}$ and $i_{d-1}i_{d-2}\cdots i_0$ are the identities of $s^{(i)}$ and i, respectively.

We will prove that with this routing scheme no two packets are at any time at the same node. We will use induction on d. For $d = 1$ (linear array) this is obvious. Assume that it is also true for $d-1$-dimensional arrays. Observe that $s^{(i)} - s^{(j)} \geq i - j$. At the end of phase 0 two packets $s^{(i)}$ and $s^{(j)}$, with $s^{(i)} > s^{(j)}$, can be at the same node only if their base p representations differed only at the 0^{th} digit. In this case we have $s^{(i)} - s^{(j)} \leq p-1$, which gives $i - j \leq p-1$. Therefore i and j also differ in the 0^{th} digit. Since at the end of phase 0 the two packets will be at nodes $s^{(i)}_{d-1}s^{(i)}_{d-2}\cdots s^{(i)}_{1}i_0$ and $s^{(j)}_{d-1}s^{(j)}_{d-2}\cdots s^{(i)}_{1}j_0$ with $i_0 \neq j_0$, they cannot be at the same node. Consider now the p subarrays $S_0, S_1, \ldots, S_{p-1}$ of dimension $d-1$ defined as follows;

$$S_k = \{s \mid s_0 = k\}.$$

During phases $1, 2, \ldots, d-1$ the packets will remain at the same one of the above submeshes at which they were at the end of phase 0, because no links of dimension 0 are crossed. Focusing on one of these subarrays and forgetting about the 0^{th} digit, which is of no significance any more, we see that the routing problem within each of these arrays is a packing problem of dimension $d-1$. Using the induction hypothesis, we see that packets are not at any time at the same node during phases $1, 2, \ldots, d-1$ either. **Q.E.D.**

The next theorem treats a more general routing problem, which we call the *mesh monotone routing* problem.

Theorem 4 (Mesh Monotone Routing): Let $s^{(i)}$ and $v^{(i)}$, $i = 0, 1, \ldots, K-1$, be nodes of a d-dimensional array such that $s^{(0)} < s^{(1)} < \cdots < s^{(K-1)}$ and $v^{(0)} < v^{(1)} < \cdots < v^{(K-1)}$. Consider the communication task, where each node $s^{(i)}$ has to send a packet to processor $v^{(i)}$. This can be performed through a greedy scheme, without conflicts, in time $2d(p-1)$. The greedy scheme uses only links of dimension j during steps $jp, jp+1, \ldots, jp+p-1$, with $0 \leq j \leq d-1$, and only links of dimension $2d-j$ during steps $jp, jp+1, \ldots, jp+p-1$, with $d \leq j \leq 2d-1$.

Proof: For each i, $i = 0, 1, \ldots, K-1$, we initially send the packet of node $s^{(i)}$ to the intermediate node i. This is a packing problem and takes time $d(p-1)$ (Theorem 3). In a second

phase, called *unpacking phase*, the packet of node i is sent to node $v^{(i)}$. This is the reverse of a packing problem and can be done by crossing the dimensions in the opposite order (from higher to lower dimensions) in time $d(p-1)$ again. **Q.E.D.**

Theorems 3 and 4 assume that packets $s^{(i)}$ know their rank i. The rank can be computed in time $2(p-1)dt_p$, where t_p is the time required for a single parallel prefix step, through a parallel prefix operation as explained in various references (see e.g. [Lei92a], pp. 37-44), and described briefly in Phase 1 of the PMNB algorithm given in the next section.

3.4. PARTIAL MULTINODE BROADCAST IN D-DIMENSIONAL TORI AND ARRAYS

In this section we consider the problem where M arbitrary nodes of a d-dimensional mesh with $N = p^d$ nodes want to broadcast a packet to all the other nodes. We call these M nodes *active nodes*. Let T^t_{PMNB} be the optimal time required for the partial multinode broadcast in a d-dimensional torus, and T^a_{PMNB} be the corresponding time for a d-dimensional array. T^t_{PMNB} and T^a_{PMNB} may actually depend on the identities of the M nodes that want to broadcast. A lower bound, however, is always

$$T^t_{PMNB} \geq \frac{M-1}{2d}, \tag{3.3}$$

and

$$T^a_{PMNB} \geq \frac{M-1}{d}, \tag{3.4}$$

where d is the dimension of the mesh. To see that, note that in a d-dimensional array (or torus) node $00\cdots0$ has only d input ports (or $2d$ input ports, respectively), and has to receive at least $M-1$ packets.

One way to execute the partial multinode broadcast is to perform a full multinode broadcast (with dummy packets for the nodes that have nothing to broadcast). The optimal completion time of the MNB in a d-dimensional torus with $N = p^d$ nodes, when each packet requires one time unit (or slot) to be transmitted over a link is $\lceil \frac{N-1}{2d} \rceil$ time slots. Thus an upper bound for T^t_{PMNB} is

$$T^t_{PMNB} \leq \left\lceil \frac{N-1}{2d} \right\rceil.$$

Since a d-dimensional array can simulate a d-dimensional torus with a slowdown factor of two, an upper bound on T^a_{PMNB} is

$$T^a_{PMNB} \leq 2\left\lceil \frac{N-1}{2d} \right\rceil .$$

When $M << N$ the previous algorithms are inefficient as the gaps between the upper and the lower bounds suggest. In this section we present communication algorithms that execute the PMNB task in d-dimensional meshes with or without wraparound in near-optimal time. In Subsection 3.4.1 we present an algorithm which assumes that packets can be split at the origin, and be recombined at the destination without any overhead. This algorithm executes the PMNB task in time

$$\frac{M}{2d}\frac{N-1}{N} + 2d(p-1)t_p + 1.5(p-1), \tag{3.5}$$

for a d-dimensional torus with $N = p^d$ nodes, and in time

$$\frac{M}{d}\frac{N-1}{N} + 2d(p-1)t_p + 2(p-1), \tag{3.6}$$

for a d-dimensional array of the same size, where t_p is the time required for a single parallel prefix step. For the case where the splitting of packets is undesirable (because of the overhead introduced, and the cost of packet reassembling), we will present in Subsection 3.4.2 an algorithm that avoids the splitting of packets, and executes the PMNB task in time less than

$$\left\lceil \frac{M}{d} \right\rceil \frac{1}{p-1}\left\lceil \frac{p-1}{2} \right\rceil \frac{N-1}{N} + (p-1)d + d\left\lceil \frac{p-1}{2} \right\rceil + 4(p-1)dt_p, \tag{3.7}$$

for a d-dimensional torus and in time

$$\left\lceil \frac{M}{d} \right\rceil + 2(p-1)d - 1 + 4(p-1)dt_p, \tag{3.8}$$

for a d-dimensional array. Comparing Eqs. (3.5) and (3.7) and Eqs. (3.6) and (3.8) with the lower bounds (3.3) and (3.4), respectively, we see that the leading terms of the corresponding right hand sides have the same coefficient. So, the algorithms to be proposed are near-optimal.

3.4.1. A Near-optimal PMNB Algorithm with Splitting of Packets

The algorithm in this section assumes that packets can be split at the origin, and recombined at the destination without any overhead. Each packet requires one time slot for transmission

over a link. If a packet is split in d parts, each of these parts requires $1/d$ time units to be transmitted over a link.

Let $s_1, s_2, \ldots, s_M$, $M \leq N$, be the active nodes. The *rank* of a packet located at node s is defined as

$$r_s = \sum_{t<s} x_t - 1,$$

where x_t is equal to one if processor t has a packet to broadcast and zero otherwise.

We will first present a suboptimal partial multinode broadcast algorithm for the d-dimensional mesh, with or without wraparound. This algorithm will not make full use of the links of a mesh. We will then modify the algorithm to achieve efficient link utilization and near-optimal completion time. The suboptimal algorithm consists of three phases:

Phase 1 (Rank Computation Phase):

The rank r_s $(0 \leq r_s \leq M-1)$ of each active node s is computed. This can be done in $2(p-1)d$ steps for a d-dimensional array or a torus by performing a *parallel prefix operation* (see [Lei92a], pp. 37-44) on a tree P, called *parallel prefix tree*, embedded in the mesh. The i^{th} leaf of the tree from the left is the i^{th} node of the mesh. The operation is described in Fig. 3.2 for a linear array and a mesh with $p = 3$ and $d = 2$. Note that during each step only links of a particular dimension are used. The packets involved in a parallel prefix operation are small (one byte of information), and require only t_p time units to be transmitted over a link. Thus it is reasonable to assume that $t_p \leq 1$, where one time unit is the time required to transmit a whole packet over a link; in fact it is reasonable to expect that in many parallel machines we have $t_p << 1$. Thus Phase 1 takes $2(p-1)dt_p$ time units to be completed.

Phase 2 (Packing Phase):

The packet of node s and rank r_s is sent to processor r_s, where r_s is interpreted as a p-ary number. This is a mesh packing problem, and can be performed in $(p-1)d$ time units according to Theorem 4.

Phase 3 (Broadcast Phase):

The broadcast phase consists of d subphases $l = 1, 2 \ldots, d$. During each subphase l, every node $r = r_{d-1}r_{d-2}\cdots r_0$ broadcasts (in any order) to all the nodes in the ring or linear array (depending on whether we are considering a mesh with or without wraparound) $(r_{d-1}\cdots r_{d-l+1} * r_{d-l-1}\cdots r_0)$, the packets that were located at the node at the beginning of Phase 3 plus the

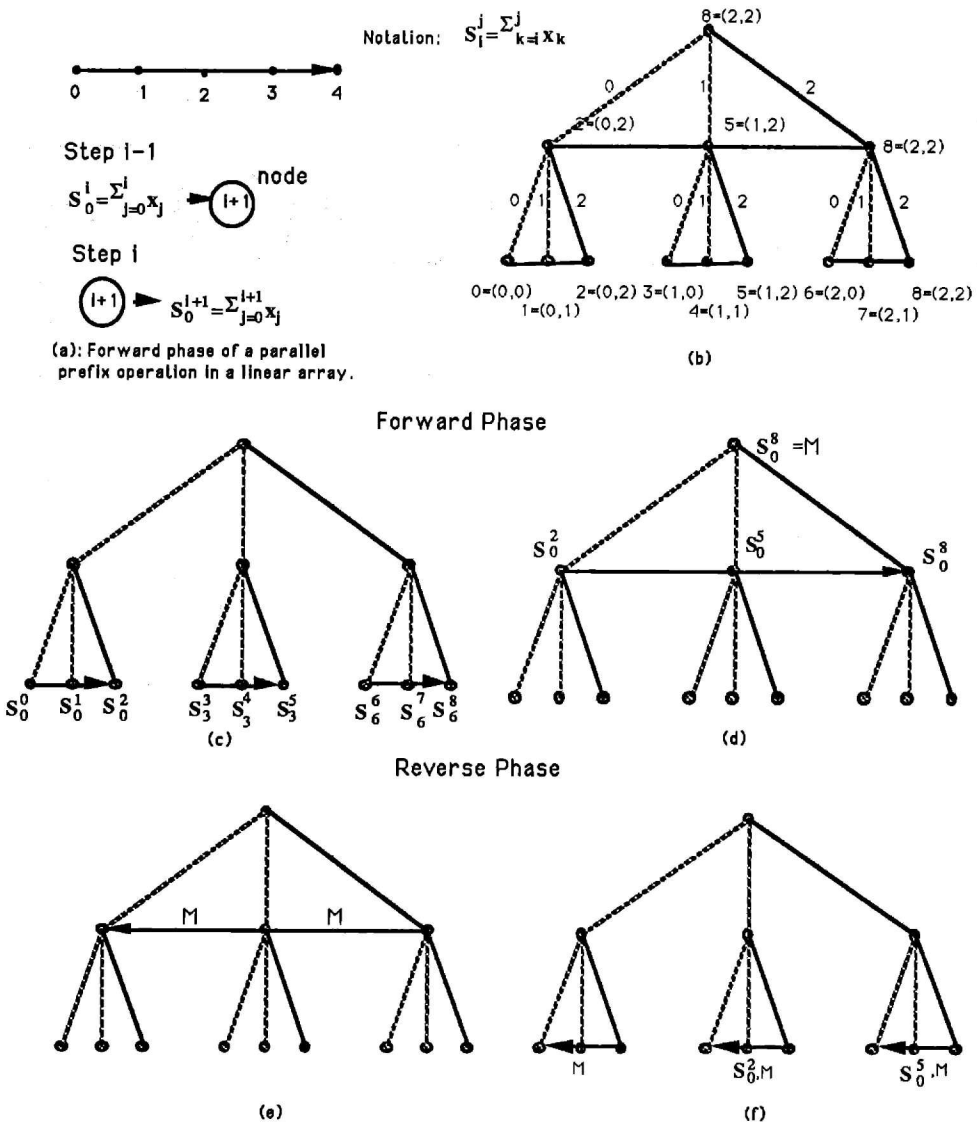

Figure 3.2: Fig. 3.2a illustrates the operation of each node during a (forward) parallel prefix operation in a linear array. The partial sums $\sum_{k=0}^{i} x_k$ are obtained at each node i in time $p-1$. Figs. 3.2b-f illustrate the parallel prefix operation in a mesh with $d = 2$ and $p = 3$. It consists of two phases (forward and reverse), each of which consists of d subphases. Each subphase is a parallel prefix operation in a linear array and requires $p-1$ steps. The total duration of the operation is $2d(p-1)$ steps. More precisely, Fig. 3.2b illustrates what we call tree representation of a mesh. An intermediate node is a root of a subtree whose leaves form a submesh of the original mesh. At the end of subphase l of the forward phase a node of level l from the bottom forms the partial sum of the values of the leaves under it. The notation S_i^j stands for $S_i^j = \sum_{k=i}^{j} x_k$. During the forward phase information moves from the bottom to the top, and from the left to the right. In the reverse phase, information moves from the top to the bottom and from the right to the left.

packets that the node has received during all the previous subphases. The broadcast algorithms

used are those described in Theorem 2.

During subphase 0 the nodes have (at most) one packet and this is the only one they broadcast. Phase 3 is easy to implement since the current subphase l is easily known.

To prove that the algorithm delivers the packets to all the nodes, it is useful to introduce some new notation. Let $\beta = \beta_{d-1}\beta_{d-2}\cdots\beta_0$ be a p-ary number of length d. We denote by $S_l(\beta) = (*^l\beta_{d-l-1}\beta_{d-l-2}\cdots\beta_0)$ the submesh of the nodes whose $d-l$ less significant digits are equal to the $d-l$ less significant digits of β.

The next theorem proves that the previous algorithm actually executes the PMNB task.

Theorem 5: For each $\beta \in \{0,1,\ldots,p-1\}^d$, at the end of subphase l of Phase 3, $l = 1,2,\ldots,d$, each node in submesh $S_l(\beta)$ has received a copy of every packet located at the beginning of Phase 3 at some node in $S_l(\beta)$, completing a PMNB within each of these submeshes.

Proof: The proof will be done by induction on l. For $l = 0$ (i.e., at the beginning of Phase 3 of the algorithm) it holds trivially since every node has its own (if any) packet. Assume it is true for some l. Every submesh $S_l(\beta)$ is composed of the p submeshes $S_{l-1}(\beta_{d-1}\cdots\beta_{d-l+1}0\beta_{d-l-1}\cdots\beta_0)$, $S_{l-1}(\beta_{d-1}\cdots\beta_{d-l+1}1\beta_{d-l-1}\cdots\beta_0)$, ..., $S_{l-1}(\beta_{d-1}\cdots\beta_{d-l+1}(p-1)\beta_{d-l-1}\cdots\beta_0)$. During subphase l every node in one of these submeshes broadcasts to all nodes in its $(d-l)$-level linear array (or ring) all the packets it has received during the previous subphases, together with its own packet. This together with the induction hypothesis proves the theorem. **Q.E.D.**

Letting $l = d$ we find that at the end of subphase d each packet has been broadcast to all the nodes, and therefore, the PMNB has been completed.

The next lemma calculates the time complexity of Phase 3.

Lemma 1: Phase 3 of the algorithm requires at most

$$\frac{N-1}{N}\frac{M}{\gamma} + \frac{(p-1)d}{\gamma}$$

time units, where $\gamma = 1$ for the d-dimensional array, and $\gamma = 2$ for the d-dimensional torus.

Proof: We denote by T_l the duration of subphase l, and we let $m = \lceil \log_p M \rceil$. At the beginning of Phase 3 only nodes $0,1,\ldots,M-1$ have a packet. From Theorem 5 we know that just before the beginning of phase l, node $s = s_{d-1}s_{d-2}\cdots s_0$ has received all the packets originally located at nodes in the submesh $(*^{l-1}s_{d-l}s_{d-l-1}\cdots s_0)$. The number of these packets

is equal to the cardinality of the set

$$\mathcal{W}_l(s) = \{w = w_{d-1}w_{d-2}\cdots w_0 \mid 0 \le w \le M-1,\ w_{d-l} = s_{d-l},\ w_{d-l-1} = s_{d-l-1}, \ldots, w_0 = s_0\}.$$

During subphase l, node s will broadcast these packets to the nodes in its $(d-l)$-level linear array or ring. Since a multinode broadcast in a linear array requires $p-1$ steps, while in a ring it requires $(p-1)/2$ steps (see the remark following Theorem 5), we have

$$T_l \le \frac{p-1}{\gamma} \max_s |\mathcal{W}_l(s)|,$$

where $\gamma = 1$ for d-dimensional arrays, $\gamma = 2$ for d-dimensional tori, and $|\cdot|$ denotes the cardinality of a set. Let $s' = s_{d-l}p^{d-l} + s_{d-l-1}p^{d-l-1} + \cdots + s_0$. The cardinality of $\mathcal{W}_l(s)$ is equal to the number of integers between 0 and $M-1-s'$, which are divisible by p^{d-l+1}. Thus

$$\max_s |\mathcal{W}_l(s)| \le \max_s \left\lceil \frac{M-1-s'}{p^{d-l+1}} \right\rceil \le \left\lceil \frac{M}{p^{d-l+1}} \right\rceil.$$

The total duration of Phase 3 satisfies

$$\begin{aligned} \text{Duration of Phase 3} &= \sum_{l=1}^{d} T_l \le \frac{p-1}{\gamma} \sum_{l=1}^{d} \left\lceil \frac{M}{p^{d-l+1}} \right\rceil \\ &\le \frac{p-1}{\gamma} \left(d + M \sum_{l=1}^{d} \frac{1}{p^l} \right) \\ &= \frac{(p-1)d}{\gamma} + \frac{M}{\gamma} \left(1 - \frac{1}{p^d} \right). \end{aligned}$$

Q.E.D.

Adding up the duration of Phases 1, 2 and 3 we obtain the following lemma:

Lemma 2: The partial multinode broadcast task can be executed in a d-dimensional torus with $N = p^d$ processors in

$$T^t_{PMNB} \le \frac{M}{2}\frac{N-1}{N} + 2d(p-1)t_p + 1.5(p-1)d$$

time units, where M is the number of active nodes. Similarly, the PMNB task can be executed in a d-dimensional array with $N = p^d$ processors in

$$T^a_{PMNB} \le M\frac{N-1}{N} + 2d(p-1)t_p + 2(p-1)d$$

time units.

The PMNB algorithm that we described so far is not of optimal order as the gap between the lower bounds of Eqs. (3.3) and (3.4), and the results of Lemma 2 indicate. In fact, they are suboptimal by a factor of roughly d. This is due to the fact that at each step only links of a particular dimension are used. In the next theorem we modify the algorithms so that all dimensions are used at the same time, and near-optimal completion time is achieved.

Theorem 6: The partial multinode broadcast task can be executed in a d-dimensional torus with $N = p^d$ processors in

$$T^t_{PMNB} \leq \frac{M}{2d}\frac{N-1}{N} + V_t \tag{3.9}$$

time units, where M is the number of active nodes, and

$$V_t = 2d(p-1)t_p + 1.5(p-1).$$

Similarly, the PMNB task can be executed in a d-dimensional array with $N = p^d$ processors in

$$T^a_{PMNB} \leq \frac{M}{d}\frac{N-1}{N} + V_a \tag{3.10}$$

time units, where

$$V_a = 2d(p-1)t_p + 2(p-1).$$

Proof: We call the PMNB algorithm analyzed in Lemmas 1 and 2 algorithm $\mathcal{A}_0$. At each step of Phases 1, 2, and 3 of $\mathcal{A}_0$, only links of a particular dimension are used. Indeed, it can be seen from Fig. 3.2 that during each step of the parallel prefix phase only links of a particular dimension are used. Similarly, in the packing phase, only links of a particular dimension are used at each step, as indicated in Theorem 4. Finally, during subphase l of the broadcast phase only links of dimension $d-l$ are used.

For any c, consider now another PMNB algorithm, referred to as algorithm $\mathcal{A}_c$. According to $\mathcal{A}_c$ a packet is transmitted over the link of dimension $(l+c)$ mod d of its current location, whenever the same packet would be transmitted under the $\mathcal{A}_0$ algorithm over the l-dimensional link of its current location. Since $\mathcal{A}_c$ is identical to $\mathcal{A}_0$ after appropriately renaming the mesh dimensions (and the nodes), and since $\mathcal{A}_0$ performs the PMNB independently of the location of the M active nodes, we conclude that $\mathcal{A}_c$ also executes the PMNB task, and requires the same amount of time as $\mathcal{A}_0$. [3]

[3] In the algorithm $\mathcal{A}_c$ the rank of an active node is defined in the following way. On the p-ary numbers of length d, we first define the *order with respect to class* c, $c \in \{0,1,\ldots,d-1\}$ (denoted by $<_c$) as

Using simultaneously all the algorithms $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_{d-1}$ we can find a new algorithm which requires the amount of time claimed in the theorem. In particular, each packet is split into d parts, called *mini packets*. Each mini packet is assigned a distinct integer c between 0 and $d-1$, called class. The mini packets of class c are routed according to algorithm $\mathcal{A}_c$. Packets of different classes use different mesh dimensions at any time. According to our communication model, a mini packet requires $1/d$ time units for transmission over a link. Therefore, the theorem follows from Lemma 2. **Q.E.D.**

The terms V_t and V_a in Eqs. (3.9) and (3.10), respectively, are growing linearly with the dimension d. In practice, however, $2d(p-1)t_p$ is small, since t_p is very small. Indeed, at each step of a parallel prefix operation only one byte has to be transmitted between neighbors. Some parallel computers, such as the Connection Machine model CM-2 of Thinking Machines Corporation, the IBM/RP-3, and the NYU Supercomputer, have very efficient implementations of the parallel prefix, otherwise called "scan" operation ([TuR88], [Ble86]). Theoretically, however, the parallel prefix operation takes time proportional to the diameter.

No upper ceilings are needed in Eqs. (3.9) and (3.10), since we allow fragmented slots. Note also that under the communication model used in this section (which allows the splitting of packets in d parts), a broadcast from a single node requires $\Theta(p)$ time units, instead of $\Theta(dp)$ which is the diameter. A near-optimal PMNB algorithm which does not use the splitting of packets is presented in the next subsection.

---

follows:

$$s <_c t \text{ iff right shift of } s \text{ by } c \text{ positions } < \text{ right shift of } t \text{ by } c \text{ positions.}$$

The *rank with respect to class c* of a packet located at node s is then defined as

$$r_s^c = \sum_{\{t: t <_c s\}} x_t \; - 1,$$

where x_t is equal to one if processor t has a packet to broadcast and zero otherwise. The parallel prefix tree P^c used in the calculation of r_s^c is the same with P, but with the digits of the nodes shifted by c positions.

3.4.2. A Near-optimal PMNB Algorithm without Splitting of Packets

In this subsection we modify the previous algorithms in order to avoid the potential drawbacks of packet splitting. This is done at the expense of a slight increase in the complexity. Messages in this section require one time slot in order to be transmitted over a link, and are always transmitted as one packet.

The algorithm consists of two parts.

Class Computation Part:

The rank r_s, $0 \leq r_s \leq M-1$, $s \in \{s_1, s_2, \ldots, s_M\}$, of each packet is computed through a parallel prefix operation. This requires $2d(p-1)t_p$ time units. The packet of node s is assigned a *class number* $c = r_s \bmod d$.

Main Part:

The packets of class c are routed according to algorithm $\mathcal{A}_c$. Recall that algorithm $\mathcal{A}_c$ consists of three phases: the rank computation, the packing, and the broadcast phase. Only packets of class c take part in the rank computation phase, or in any other phase of $\mathcal{A}_c$.

Each class has at most $\lceil M/d \rceil$ packets. Lemma 1 has been proved under the assumption that a packet can be split into two parts. When packets cannot be split, it can be shown (by a proof similar to that in Lemma 1) that algorithm $\mathcal{A}_c$ requires time less than or equal to

$$M' \frac{1}{p-1} \left\lceil \frac{p-1}{\gamma} \right\rceil \frac{N-1}{N} + d \left\lceil \frac{p-1}{\gamma} \right\rceil,$$

where M' is the number of packets that participate in $\mathcal{A}_c$, $\gamma = 1$ for d-dimensional arrays, and $\gamma = 2$ for d-dimensional tori. Substituting $\lceil M/d \rceil$ instead of M', and adding the time required for the parallel prefix operations and the packing phase, we get

$$T_{PMNB} \leq \left\lceil \frac{M}{d} \right\rceil \frac{1}{p-1} \left\lceil \frac{p-1}{\gamma} \right\rceil \frac{N-1}{N} + (p-1)d + d \left\lceil \frac{p-1}{\gamma} \right\rceil + 4(p-1)dt_p.$$

The algorithm just presented for the PMNB task gives rise to an efficient algorithm for the MNB task. Indeed, a multinode broadcast can be treated as a partial multinode broadcast with $M = N$. The class computation part and the rank computation phase are not necessary any more, since the class number and the rank of each packet are known in advance. The packing and the broadcast phases alone can execute the MNB in time less than or equal to

$$\left\lceil \frac{N}{d} \right\rceil \frac{1}{p-1} \left\lceil \frac{p-1}{\gamma} \right\rceil \frac{N-1}{N} + (p-1)d + d \left\lceil \frac{p-1}{\gamma} \right\rceil.$$

which is near-optimal for tori with p odd or arrays, and of optimal order for tori with p even. This MNB algorithm is apparently new.

3.5. PARTIAL EXCHANGE IN 2-DIMENSIONAL ARRAYS

In this section we present an algorithm to execute the partial exchange task in a $p \times p$ array (and, therefore, in a $p \times p$ torus). In particular, we initially assume that there are M nodes, called *active*, that want to send a personalized packet to each of the other nodes. The algorithm to be presented has time complexity which is of optimal order.

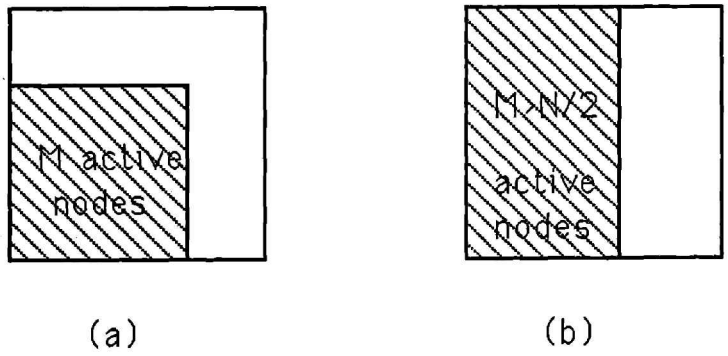

Figure 3.3: Position (a) of the active nodes corresponds to the first lower bound, while position (b) corresponds to the second lower bound.

We first present lower bounds on the minimum time required to execute the partial exchange in 2-dimensional array. Let us denote

$$m = \lfloor M^{1/2} \rfloor.$$

First, consider the case where m^2 of the M active nodes are in the $m \times m$ subarray $M_{0,0}$, where

$$M_{0,0} = \{(i,j) \mid 0 \leq i \leq m-1,\ 0 \leq j \leq m-1\}.$$

Then $m^2(N-m^2)$ packets have to cross the $2m$ links connecting $M_{0,0}$ to the rest of the array (since there are no wraparound links). This gives

$$T_{PE} \geq \frac{m(N-m^2)}{2}.$$

Thus,

$$T_{PE} \geq \frac{mN}{4}, \quad \text{for } M \leq N/2.$$

We next consider the case where $M \geq N/2$, and all the active nodes are at the left side of the mesh (see Fig. 3.3). Consider the packets that pass from left to right through the cut that

bisects the mesh. At least $N^2/4$ packets cross the p links of this cut. Thus,

$$T_{PE} \geq \frac{Np}{4} \geq \frac{mN}{4}, \quad \text{for } M \geq N/2.$$

The previous bounds show that

$$T_{PE} \geq \frac{mN}{4} = \Omega(M^{1/2}N).$$

Before describing the algorithm, we introduce some notation. The nodes of the mesh are represented as pairs (i,j) with $0 \leq i,j \leq p-1$. The *row sum* (or *column sum*) of node (i,j) is defined as the number of active nodes of row i (or column j) and is denoted by r_i (or c_j, respectively).

We now describe the algorithm. It consists of two phases:

Phase 1 (Parallel Prefix Phase):

The row sums r_i and column sums c_j are computed. All r_i's and c_j's can be found in p steps by concurrently performing a parallel prefix operation within each row and column. A row or column is a linear array, and can be viewed as a tree rooted at a median node of that linear array of depth $\lfloor p/2 \rfloor$. The parallel prefix operation is performed with value equal to one for active nodes and zero for the other nodes. Phase 1 requires pt_p time units, where $t_p \leq 1$ is the time required for a single parallel prefix step.

Phase 2 (Exchange Phase):

The set of active nodes is partitioned into the two sets R and C where

$$R = \{(i,j) \text{ active} \mid r_i \leq c_j\}, \quad C = \{(i,j) \text{ active} \mid r_i > c_j\}.$$

The nodes in R or in C send each of their packets along the unique shortest path that first crosses horizontal (respectively, vertical) links exclusively, and then crosses vertical (respectively, horizontal) links exclusively. The order of transmission of the packets at each link is arbitrary subject to two restrictions:

a) Packets originating at nodes of R (or of C) have priority on the horizontal (respectively, vertical) links over packets originating at nodes in C (respectively, R).

b) Transmission is non-wasting in the sense that no link remains idle if there is a packet waiting at the queue of the link.

We have the following lemma:

Lemma 3: The number of nodes of R that belong to the same row are at most m.

Proof: Our proof is by contradiction. Suppose that for some i, the nodes $(i, j_1), (i, j_2), \ldots, (i, j_x)$ belong to R, and $x > m$. Then by the definition of the set R, $c_{j_k} \geq r_i \geq x > m$ for all $k = 1, 2, \ldots, x$. This implies that there are $x \geq m + 1$ columns, each of which has at least x active nodes. This is a contradiction since there are only $M < (m+1)^2$ active nodes. **Q.E.D.**

We claim that Phase 2 requires at most $2m(N - p) + 2(p - 1)$ time units. To see this, we first note that the number of packets originating at nodes of R that must cross at least one horizontal link of any given row i is at most $m(N - p)$. The reason is that by Lemma 3 there are at most m active nodes from R in row i and each of these nodes has a total of $N - p$ packets to send to nodes that belong to a different column. Since each packet increases the delay of another packet by at most one unit along the horizontal path, we see that the time required for all the packets originating at nodes in R to traverse completely the horizontal portion of their path is at most $m(N - p) + (p - 1)$. Similarly, the time required for all the packets originating at nodes in C to traverse completely the vertical portion of their path is also at most $m(N - p) + (p - 1)$.

Let us make the worst-case assumption that packets originating at nodes of R (or C) are delayed after completing their horizontal (respectively, vertical) transmissions so that their transmission starts after exactly $m(N - p) + (p - 1)$ time units. We will show that at most $m(N - p) + (p - 1)$ additional time units are needed to complete Phase 2. Indeed, at the end of the first $m(N - p) + (p - 1)$ time units, each node has at most $m(p - 1)$ packets originating at nodes in R to send over the vertical links. Therefore, at most $m(p - 1)p$ such packets remain to traverse the links of its column. This requires at most an additional $m(N - p) + (p - 1)$ time units.

Adding up the times required for each phase, and taking into account that $N = p^2$ and $m = \lfloor M^{1/2} \rfloor$, we find that the time T_{PE} required for the partial exchange in a 2-dimensional array satisfies

$$T_{PE} \leq 2\lfloor M^{1/2} \rfloor (N - p) + 2(p - 1) + pt_p.$$

Comparing this inequality with the lower bound found earlier, we see that our algorithm is of optimal order (within a factor of roughly 8 of being optimal).

3.6. PARTIAL MULTINODE BROADCAST IN A HYPERCUBE

Beginning with this section we focus on a hypercube network of processors. In particular, we consider the partial multinode broadcast problem, where M arbitrary nodes of an N-processor hypercube have to broadcast a packet to all the other nodes. We call these nodes *active nodes*.

Let T_{PMNB} be the optimal time required for a partial multinode broadcast in a hypercube. T_{PMNB} may actually depend on which are the M nodes that want to broadcast. A lower bound, however, is always

$$T_{PMNB} \geq \frac{M-1}{d}, \tag{3.11}$$

where d is the dimension of the hypercube. This can be seen by arguing that each node has to receive $M-1$ or M packets, and has only d input ports. If the splitting of packets is not allowed then a slightly stronger lower bound holds:

$$T_{PMNB} \geq \max\left(d, \left\lceil \frac{M-1}{d} \right\rceil\right),$$

where by $\lceil x \rceil$ we denote the smallest integer which is greater than or equal to x. This is because when packets are not split, the diameter of the network is a lower bound on the broadcast delay.

One way to execute the partial multinode broadcast is to perform a full multinode broadcast (with dummy packets for the nodes which have nothing to broadcast). The optimal completion time of the MNB in an d-dimensional hypercube with $N = 2^d$ nodes, when each packet requires one time unit (or slot) to be transmitted over a link, was found in [BOS91] (see also [BeT89]) to be $\lceil \frac{N-1}{d} \rceil$ time slots. Thus an upper bound for T_{PMNB} is

$$T_{PMNB} \leq \left\lceil \frac{N-1}{d} \right\rceil.$$

When $M << N$ the MNB algorithm is inefficient as the gap between the preceding inequality and the lower bound of Eq. (3.11) suggests. In the three subsections of this section we will provide three communication algorithms to execute the PMNB task in a hypercube. The first algorithm, presented in Subsection 3.6.1, has time complexity

$$T^I_{PMNB} \leq \left\lceil \frac{M-1}{m} \right\rceil + 2d + 2dt_p - m,$$

in the case $M = 2^m$ for some integer m, where t_p is the time required for a single parallel prefix step ($t_p \leq 1$). If M is not a power of 2 then m should be replaced in the above expression by

$\lceil \log M \rceil$ and M by $2^{\lceil \log M \rceil}$. This algorithm is *not* of optimal order [except if $M = \Theta(N^c)$ for some positive constant c, or if $M = \Theta(\log N)$], but it is a simple and practical algorithm, as numerical examples indicate.

The second and the third algorithms, to be described in Subsections 3.6.2 and 3.6.3, respectively, execute the PMNB in near-optimal time. In particular the second algorithm, which we call *near-optimal PMNB algorithm*, executes the task in time

$$T^{II}_{PMNB} \leq \frac{N-1}{N}\frac{M}{d} + 2dt_p + 2, \tag{3.12}$$

independently of the value of M and the location of the active nodes. This is the best existing algorithm for the PMNB task, having roughly half the complexity of the PMNB algorithm given in [Sta91]. Comparing Eq. (3.12) with the lower bound (3.11) we see that the leading terms of the right hand sides have the same coefficient. The algorithm assumes that packets can be split at the origin and recombined at the destinations. For the case where this is undesirable we modify the algorithm to achieve near-optimal completion time without the need of splitting and recombining the packets. This gives rise to a third PMNB algorithm, which will be presented in Subsection 3.6.3. We call it *near-optimal PMNB without splitting of packets* and its time complexity can be bounded above by

$$T^{III}_{PMNB} \leq \left\lceil \frac{M}{d} \right\rceil + 2d + 4dt_p - 1,$$

for any M. Note that the MNB is a special case of the PMNB with $M = N$. The latter PMNB algorithm gives rise to a very efficient MNB algorithm with complexity $\lceil N/d \rceil + 2d - 1$, which does not use the splitting of packets. This MNB algorithm has not appeared in the literature before.

The benefits of using a partial multinode broadcast instead of a full multinode broadcast algorithm can be best illustrated by a numerical example.

Example:

We consider both the cases where t_p=1 and $t_p = 0$. The case $t_p = 0$ corresponds to the situation where $t_p << 1$ (a realistic assumption for computers which have an efficient implementation of the parallel prefix operation), or to the situation where the position of the active nodes is known in advance. Consider a hypercube of $N = 2^{16}$ nodes and a PMNB task in it involving $M = 2^{10}$ active nodes. A full MNB would take 4096 steps. Algorithm I requires 157 time units (or only 125 if $t_p = 0$). The near-optimal algorithm which allows splitting of packets requires 98 time units (or only 66 time units if $t_p = 0$). The near-optimal PMNB algorithm

without splitting of packets requires 160 time units (96 if $t_p = 0$). The lower bound for the PMNB in this case is 64.

3.6.1. A Suboptimal PMNB Algorithm for the Hypercube

This algorithm consists of four phases. (There can be some pipelining between the phases, but we do not try to exploit this because the gain in completion time is small).

Let $s_1, s_2, \ldots, s_M$, $M \leq N$, be the active nodes. The *rank* of a packet located at node s is defined as

$$r_s = \sum_{t<s} x_t \ - 1,$$

where x_t is equal to one if processor t has a packet to broadcast, and zero otherwise.

Phase 1 (Rank Computation Phase):

The rank r_s $(0 \leq r_s \leq M-1)$ of each active node s is computed. This can be done in $2d$ steps by performing a *parallel prefix operation* (see [Lei92a]) on a tree P, called *parallel prefix tree*, embedded in the hypercube. The i^{th} leaf of the tree from the left is the i^{th} node of the hypercube. The operation is described in [Lei92a], pp. 37-44. During each step only links of a particular dimension are used. The packets involved in a parallel prefix operation are small (one byte of information), and require only t_p time units to be transmitted over a link. Thus Phase 1 takes $2dt_p$ time units to be completed. It is reasonable to assume that $t_p \leq 1$, where one time unit is the time required to transmit a whole packet over a link; in fact it is reasonable to expect that in many parallel machines we have $t_p << 1$. Note that if the active nodes are known in advance their ranks are also known and Phase 1 can be omitted.

Phase 2 (Packing Phase):

Processor s sends its packet to processor r_s. This is known in the literature as the *packing problem* (see [Lei92a] for the case of a butterfly network), and can be done in d steps by using the following greedy algorithm; a packet during the i^{th} step of the packet phase is transmitted over an i-dimensional link if the i^{th} bit of its routing path is a one, or stays at the current node otherwise. Packets in the packing phase use disjoint paths. This can be seen by noting that when the dimensions of the hypercube are travelled in an ascending order, the hypercube resembles a butterfly, and using the well-known results for the packing problem in butterflies

(see, for example, [Lei92a], pp. 524-538).

At the end of the second phase the 2^m nodes with the smallest identities have a packet. In the last two phases, each of these packets will be broadcast to all the other processors. Note that the $M = 2^m$ nodes with the smallest identities form a subcube of dimension m, namely, the one obtained by fixing the $d - m$ most significant bits to 0.

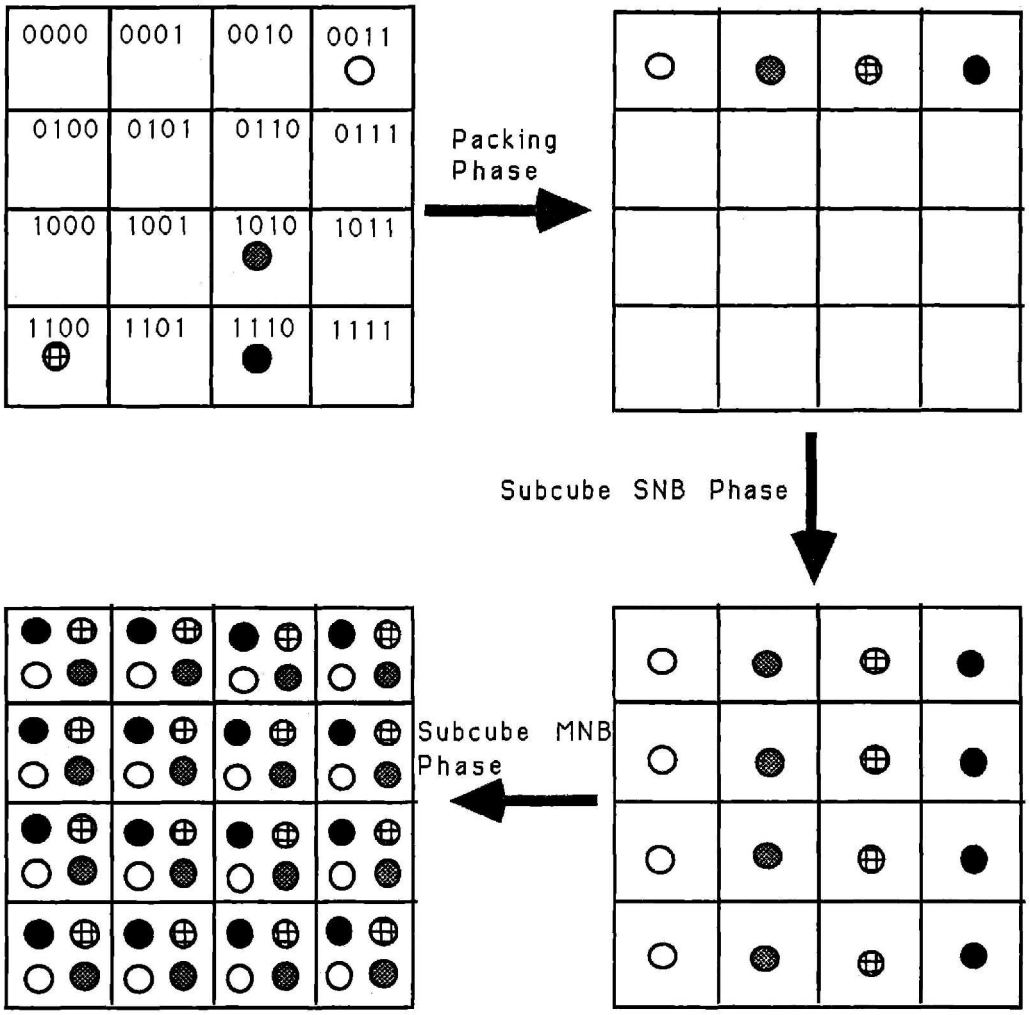

Figure 3.4: The first (suboptimal) PMNB algorith for hypercubes ($N = 16$, $M = 4$). Each column or row corresponds to a subcube.

Phase 3 (Subcube Single Node Broadcast Phase):

Processor r_s, $0 \leq r_s \leq 2^m - 1$, broadcasts its packet to the $d - m$ dimensional hypercube

$(*^{d-m}r_s)$ obtained by fixing the m less significant bits to equal the binary representation of r_s. This is a single node broadcast in a $(d-m)$-dimensional hypercube, and requires $d-m$ steps.

Phase 4 (Subcube Multinode Broadcast Phase):

At the beginning of this phase all processors wr_s, $w = 0, 1, \ldots, 2^{d-m}$, have received the packet originating at node s. During Phase 4 processor wr_s broadcasts the packet to all the nodes in the subcube $(w*^m)$. This is a full MNB in each one of these m-dimensional disjoint subcubes, and requires time $\lceil \frac{M-1}{m} \rceil$ (see [BeT89], Section 1.3).

Adding up the durations of Phases 1 through 4 we get

$$T^I_{PMNB} \leq \left\lceil \frac{M-1}{m} \right\rceil + 2d + 2dt_p - m.$$

Figure 3.4 shows how the preceding algorithm works for a 4-dimensional hypercube and $M = 4$ active nodes.

3.6.2. A Near-optimal PMNB Algorithm with Splitting of Packets

In this subsection we present a near-optimal algorithm to execute the partial multinode broadcast task in a hypercube. We will show that the time required by the algorithm satisfies

$$T^{II}_{PMNB} \leq \frac{N-1}{N}\frac{M}{d} + 2dt_p + 2,$$

where t_p is the time required for a single parallel prefix step.

The algorithm in this section assumes that packets can be split at the origin and recombined at the destination without any overhead. Each packet requires one time slot in order to be transmitted over a link. If a packet is split in d parts, each of these parts requires $1/d$ time units to be transmitted over a link. In the next subsection, we will present another near-optimal partial multinode broadcast algorithm, which does not require the splitting of packets.

We will start by presenting a suboptimal partial multinode broadcast algorithm. This algorithm will not make full use of the links of a hypercube. We will then modify the algorithm to achieve efficient link utilization and near-optimal completion time. The suboptimal algorithm consists of three phases:

Phase 1 (Rank Computation Phase):

Tha rank r_s of each active node is computed. This computation is done through a parallel prefix operation as in Phase 1 of the algorithm of the previous subsection. It requires time $2dt_p$, where t_p is the time required for a single parallel prefix step.

Phase 2 (Packing Phase):

The packet of node s and rank r_s is sent to processor r_s. This is done in d steps as described in Phase 2 of the algorithm of the previous subsection.

Phase 3 (Broadcast Phase):

The broadcast phase consists of d subphases $l = 1, 2 \ldots, d$. During subphase l, every node $r = r_{d-1}r_{d-2}\cdots r_0$ transmits to its $(d-l)$-neighbor in any order the packets that were located at the node at the beginning of Phase 3 plus the packets that the node has received during all the previous subphases.

During subphase 0 the nodes have (at most) one packet and this is the only one they broadcast. Phase 3 is easy to implement since the current subphase l is easily known.

To prove that the algorithm delivers the packets to all the nodes, it is useful to introduce some new notation. Let $\beta = \beta_{d-1}\beta_{d-2}\cdots\beta_0$ be a binary number of length d. We denote by $S_l(\beta) = (*^l\beta_{d-l-1}\beta_{d-l-2}\cdots\beta_0)$ the subcube of the nodes whose $d-l$ less significant bits are equal to the $d-l$ less significant bits of β.

The next theorem proves that the previous algorithm actually executes the PMNB task.

Theorem 7: For each $\beta \in \{0,1\}^d$, at the end of subphase l of Phase 3, $l = 1, 2, \ldots, d$, each node in subcube $S_l(\beta)$ has received a copy of every packet located at the beginning of Phase 3 at some node in $S_l(\beta)$, completing a PMNB within each of these subcubes.

Proof: The proof will be done by induction on l. For $l = 0$ (i.e., at the beginning of Phase 3 of the algorithm) it holds trivially since every node has its own (if any) packet. Assume it is true for some l. Every subcube $S_l(\beta)$ is composed of the two subcubes $S_{l-1}(\beta_{d-1}\cdots\beta_{d-l+1}0\beta_{d-l-1}\cdots\beta_0)$ and $S_{l-1}(\beta_{d-1}\cdots\beta_{d-l+1}1\beta_{d-l-1}\cdots\beta_0)$. During subphase l every node in one of these subcubes sends to its $(d-l)$-neighbor all the packets it has received during the previous subphases, together with its own packet. This combined with the induction hypothesis proves the theorem. **Q.E.D.**

Letting $l = d$ we find that at the end of subphase d each packet has been received by all the

nodes, and therefore, the PMNB has been completed.

The next lemma calculates the time complexity of Phase 3.

Lemma 4: Phase 3 of the algorithm requires at most

$$\frac{N-1}{N}M + d$$

steps.

Proof: We denote by T_l the duration of subphase l, and we let $m = \lceil \log M \rceil$. At the beginning of Phase 3 only nodes $0, 1, \ldots, M-1$ have a packet. From Theorem 7 we know that just before the beginning of phase l, node $s = s_{d-1}s_{d-2}\cdots s_0$ has received all the packets originally located at nodes in the subcube $(*^{l-1}s_{d-l}s_{d-l-1}\cdots s_0)$. The number of these packets is equal to the cardinality of the set

$$\mathcal{W}_l(s) = \{w = w_{d-1}w_{d-2}\cdots w_0 \mid 0 \le w \le M-1,\ w_{d-l} = s_{d-l},\ w_{d-l-1} = s_{d-l-1}, \ldots, w_0 = s_0\}.$$

During subphase l, node s will send these packets to its $d-l$-neighbor. Therefore, we have

$$T_l \le \max_s |\mathcal{W}_l(s)|$$

where $|\cdot|$ denotes the cardinality of a set. Let $s' = s_{d-l}2^{d-l} + s_{d-l-1}2^{d-l-1} + \cdots + s_0$. The cardinality of $\mathcal{W}_l(s)$ is equal to the number of integers between 0 and $M-1-s'$, which are divisible by 2^{d-l+1}. Thus

$$\max_s |\mathcal{W}_l(s)| \le \max_s \left\lceil \frac{M-1-s'}{2^{d-l+1}} \right\rceil \le \left\lceil \frac{M}{2^{d-l+1}} \right\rceil.$$

The total duration of Phase 3 satisfies

$$\begin{aligned} \text{Duration of Phase 3} &= \sum_{l=1}^{d} T_l \le \sum_{l=1}^{d} \left\lceil \frac{M}{2^{d-l+1}} \right\rceil \\ &\le d + M\sum_{l=1}^{d} \frac{1}{2^l} \\ &= d + M\left(1 - \frac{1}{N}\right). \end{aligned}$$

Q.E.D.

Adding up the duration of Phases 1, 2 and 3 we obtain the following lemma:

Lemma 5: The partial multinode broadcast task can be executed in a d-dimensional hypercube in

$$T_{PMNB} \leq M\frac{N-1}{N} + 2dt_p + 2d$$

time units, where M is the number of active nodes.

The PMNB algorithm that we described so far is not of optimal order as the gap between the lower bound (3.11), and the result of Lemma 5 indicates. In fact, it is suboptimal by a factor of roughly d. This is due to the fact that at each step only links of a particular dimension are used. In the next theorem we modify the algorithm so that all dimensions are used at the same time, and near-optimal completion time is achieved.

Theorem 8: The partial multinode broadcast task in a d-dimensional hypercube can be executed in

$$T_{PMNB} = \frac{M}{d}\frac{N-1}{N} + V_h \tag{3.13}$$

time units, where M is the number of active nodes, and

$$V_h = 2dt_p + 2$$

is the time required for the first two phases.

Proof: We call the PMNB algorithm analyzed in Lemmas 4 and 5 algorithm $\mathcal{A}_0$. At each step of Phases 1, 2, and 3 of $\mathcal{A}_0$, only links of a particular dimension are used. Indeed, in the parallel prefix or the packing phase only links of a particular dimension are used at each step. Similarly, during subphase l of the broadcast phase only links of dimension $d-l$ are used.

For any c, consider now another PMNB algorithm referred to as algorithm $\mathcal{A}_c$. According to $\mathcal{A}_c$ a packet is transmitted over the link of dimension $(l+c)$ mod d of its current location, whenever the same packet would be transmitted under the $\mathcal{A}_0$ algorithm over the l-dimensional link of its current location. Since $\mathcal{A}_c$ is identical to $\mathcal{A}_0$ after appropriately renaming the hypercube dimensions (and the nodes), and since $\mathcal{A}_0$ performs the PMNB independently of the location of the M active nodes, we conclude that $\mathcal{A}_c$ also executes the PMNB task, and requires the same amount of time as $\mathcal{A}_0$.

Using simultaneously all the algorithms $\mathcal{A}_0$, $\mathcal{A}_1$, ..., $\mathcal{A}_{d-1}$ we can find a new algorithm which requires the amount of time claimed in the theorem. In particular, each packet is split into d parts, called *mini packets*. Each mini packet is assigned a distinct integer c between 0 and $d-1$, called *class*. The mini packets of class c are routed according to algorithm $\mathcal{A}_c$.

Packets of different classes use different hypercube dimensions at any time. According to our communication model, a mini packet requires $1/d$ time units for transmission over a link. The duration of the packing and the broadcast phase is thus reduced by a factor of d, while the duration of the parallel prefix phase remains the same. Therefore, the theorem follows from Lemma 5. **Q.E.D.**

No upper ceilings are needed in Eq. (3.13), since we allow fragmented slots. Note also that under the communication model used in this section a single multinode broadcast requires 2 time units, the same time that would be required if cut-through routing ([KeK79]) was used. A near-optimal PMNB algorithm that does not require the splitting of packets is presented in the next subsection.

3.6.3. A Near-optimal Hypercube PMNB Algorithm without Splitting of Packets

In this subsection we modify the algorithm of the preceding section in order to avoid the potential drawbacks of packet splitting. This is done at the expense of a slight increase in the complexity. Messages in this section require one time slot in order to be transmitted over a link, and are always transmitted as one packet.

The algorithm consists of two parts.

Class Computation Part:

The rank r_s, $0 \leq r_s \leq M-1$, $s \in \{s_1, s_2, \ldots, s_M\}$, of each packet is computed through a parallel prefix operation. This requires $2dt_p$ time units. The packet of node s is assigned a *class number* $c = r_s \bmod d$.

Main Part:

The packets of class c are routed according to algorithm $\mathcal{A}_c$ described in the proof of Theorem 8. Recall that algorithm $\mathcal{A}_c$ consists of three phases: the rank computation, the packing, and the broadcast phase. Only packets of class c take part in the rank computation phase, or in any other phase of $\mathcal{A}_c$.

Each class has at most $\lceil M/d \rceil$ packets. Using Lemma 5 with $\lceil M/d \rceil$ instead of M, we get that the main part of the previous algorithm requires time less than $\lceil \frac{M}{d} \rceil + 2d - 1 + 2dt_p$ (we have taken into account that the duration of the main part, excluding the parallel prefix operation,

is an integer). Thus the total duration of the algorithm satisfies

$$T^{III}_{PMNB} \leq \left\lceil \frac{M}{d} \right\rceil + 2d + 4dt_p - 1.$$

The algorithm just presented for the PMNB task gives rise to an efficient algorithm for the MNB task. Indeed, a multinode broadcast can be treated as a partial multinode broadcast with $M = N$. The class computation part and the rank computation phase are not necessary any more, since the class number and the rank of each packet are known in advance. The packing and the broadcast phases alone can execute the MNB in time less than or equal to

$$\left\lceil \frac{N}{d} \right\rceil + 2d - 1,$$

which is optimal within $2d - 1$ time units. This MNB algorithm is apparently new.

The table of Fig. 3.5 summarizes the results on the PMNB task.

PMNB Model	Hypercube	d-dimensional array	d-dimensional torus
Splitting of packets	$\frac{N-1}{N}\frac{M}{d} + 2dt_p + 2$	$\frac{N-1}{N}\frac{M}{d} + 2(p-1) + 2d(p-1)t_p$	$\frac{N-1}{N}\frac{M}{2d} + 1.5(p-1) + 2d(p-1)t_p$
No Splitting of packets	$\lceil \frac{M}{d} \rceil + 2d + 4dt_p - 1$	$\lceil \frac{M}{d} \rceil + 3d(p-1) + 4d(p-1)t_p - 1$	$\frac{N-1}{N(p-1)}\lceil \frac{p-1}{2} \rceil \lceil \frac{M}{d} \rceil + d(p-1) + d\lceil \frac{p-1}{2} \rceil + 4d(p-1)t_p$

Figure 3.5: A table of the results for the PMNB task.

3.7. PARTIAL EXCHANGE IN A HYPERCUBE

In this section we present an algorithm of optimal order to execute a partial exchange in a hypercube with $N = 2^d$ nodes in optimal time. We assume that $M = 2^m$ nodes, called again *active nodes*, want to send a personalized packet to each one of the other nodes (therefore, each node sends a total of $N - 1$ packets). No assumption on the location of the active nodes $s_1, s_2, \ldots, s_M$ is made, and the splitting of packets is not allowed. The worst case execution time of the algorithm that we will present is

$$\frac{M}{2} + 2\left\lceil \frac{N}{d-m} \right\rceil + 2dt_p + m + 1,$$

where t_p is the time required for a single parallel prefix step. This algorithm is of the optimal order of magnitude since, as it will be shown, a lower bound on the time T_{PE} required to execute the task is

$$T_{PE} = \Omega\left(\max(\frac{M}{2}, \frac{N}{d-m}, d)\right).$$

We first prove the lower bounds. Since the diameter of the hypercube is d and the splitting of packets is not allowed we immediately get that

$$T_{PE} \geq d. \tag{3.14}$$

The minimum total number of transmissions required for the PE can be calculated to be equal to $dNM/2$. Since there are dN unidirectional links in the hypercube, we get that

$$T_{PE} \geq \frac{M}{2}. \tag{3.15}$$

To get a third lower bound consider the case where all the active nodes belong to the m-dimensional subcube $H_m(0) = (*^m 0^{d-m})$ obtained by fixing the $d-m$ least significant bits of the node adress to be equal to 0. Subcube $H_m(0)$ has M nodes and is connected to the rest of the hypercube via $(d-m)M$ links. Since the $(N-M)M$ packets that are destined for nodes outside $H_m(0)$ have to pass through these links we have that

$$T_{PE} \geq \left\lceil \frac{N-M}{d-m} \right\rceil. \tag{3.16}$$

If $M > N/2$ then Eq. (3.15) gives that $T_{PE} = \Theta(N)$. Next consider the case where $M \leq N/2$, and all the active nodes are outside $H_m(0)$. Then M^2 packets will have to enter $H_m(0)$ through the $(d-m)M$ links, which gives

$$T_{PE} \geq \left\lceil \frac{M}{d-m} \right\rceil \quad \text{for } M \leq N/2. \tag{3.17}$$

Equations (3.14)-(3.17) together with the relation $\max(x,y) \geq (x+y)/2 = \Omega(x+y)$ give;

$$T_{PE} = \Omega\left(\frac{M}{2} + \frac{N}{d-m} + d\right). \tag{3.18}$$

In the rest of the section we present a distributed, "on-line" communication algorithm which is of optimal order. The algorithm consists of four phases.

Phase 1 (Rank Computation Phase):

The rank of each active node is computed as in Phase 1 of the algorithm presented in Subsection 3.6.1. This requires at most $2dt_p$ time units, where t_p is the time of a single parallel prefix step. Let r_i be the binary representation of the rank of active node s_i.

Phase 2 (Isotone Routing Phase):

Before describing this phase we give some definitions and notations.

For any node w, we denote by $R_{d-m}(w) = (w_{d-1} \cdots w_{d-m} *^{d-m})$ the subcube of dimension $d-m$ obtained by fixing the m most significant bits of w. Restricting attention to $R_{d-m}(w)$, we define the *single node scatter tree* $T_{d-m}(w)$ rooted at w and belonging to $R_{d-m}(w)$ as in [BOS91]. We let e_i be the unit vector whose i^{th} entry is one. Then the node $w \oplus e_i$, $i \in \{0, 1, \ldots, d-m-1\}$, is the root of a subtree $T^i_{d-m}(w)$ of $T_{d-m}(w)$. Since $T_{d-m}(w)$ is a *completely balanced* tree each of the subtrees $T^i_{d-m}(w)$, $i = 0, 1, \ldots, d-m-1$, contains at most

$$S = \left\lceil \frac{N/M - 1}{d-m} \right\rceil$$

nodes. We denote by $W_k(\cdot)$ the operator which when applied to a node address yields the k least significant bits of the address.

Phase 2 of the PE algorithm goes as follows. Node s_i, $i = 1, \ldots, M$, sends a packet with destination $t \notin (*^m W_{d-m}(s_i))$ to node $r_i W_{d-m}(s_i) \oplus e_{\hat{j}}$, where $\hat{j}$ is chosen so that $r_i W_{d-m}(t) \in T_{\hat{j}}(r_i W_{d-m}(s_i))$, and a packet with destination $t \in (*^m W_{d-m}(s_i))$ to node $r_i W_{d-m}(s_i)$. Therefore, source s_i sends at most MS packets (for a total of $N - M$ packets) to each of the nodes $r_i W_{d-m}(s_i) \oplus e_{\hat{j}}$, $\hat{j} \in \{0, 1, \ldots, d-m-1\}$, and sends M packets to node $r_i W_{d-m}(s_i)$. The paths followed by these packets are the following. The dimensions are travelled in ascending order, starting with dimension 0. If a packet does not want to traverse one of the dimensions $d-m, \ldots, d-1$, then the packet spends the corresponding slot without being transmitted (idling). Packets sent from s_i to the same node (node $r_i W_{d-m}(s_i) \oplus e_{\hat{j}}$ for some $\hat{j}$, or node $r_i W_{d-m}(s_i)$) are sent one after the other (pipelined).

We claim that

Lemma 6 Packets do not collide at any slot during Phase 2.

Proof: Packets sent one after the other by node s_i to a specific node ($r_i W_{d-m}(s_i) \oplus e_{\hat{j}}$ or $r_i W_{d-m}(s_i)$) do not collide with each other, because they use different dimensions during a slot. Packets sent from node s_i do not collide with packets sent from s_k if they started transmission at different times, because they are tranversing different hypercube dimensions at each slot.

Packet sent from source s_i do not collide with packets that started transmission from source node s_k at the same time, because of the properties of isotone routing. In order to see this, assume that the packet from node s_i (or node s_k) is sent to node $r_iW_{d-m}(s_i) \oplus e_{j_1}$ (or node $r_kW_{d-m}(s_k) \oplus e_{j_2}$), respectively). Assume also, without loss of generality, that $s_i > s_k$. Then $r_i > r_k$, and, therefore, $r_iW_{d-m}(s_i) \oplus e_{j_1} > r_kW_{d-m}(s_k) \oplus e_{j_2}$. These two packets cross links of the same dimension during the same slot, in the order from lower to higher dimensions. Thus, these packets would collide only if they would collide in a corresponding butterfly with the usual greedy routing. Since the relative order of the sources of these packets is the same with the relative order of their destinations, these packets do not collide in a butterfly; thus they do not collide in the hypercube either. The last thing to prove is that a packet $\mathcal{P}_1$ sent from s_i to $r_iW_{d-m}(s_i) \oplus e_{j_1}$ does not collide with a packet $\mathcal{P}_2$ sent from s_i to $r_iW_{d-m}(s_i) \oplus e_{j_2}$ or with a packet $\mathcal{P}_3$ sent from s_i to $r_iW_{d-m}(s_i)$. This is indeed the case since the first dimension that $\mathcal{P}_1$ will cross is dimension j_1 (and it will never cross dimension j_2), the first dimension that $\mathcal{P}_2$ will cross is j_2 (and it will never cross dimension j_1), and $\mathcal{P}_3$ will not cross (in Phase 2) any of these dimensions; since packets $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$ have the same origin, they will not collide because after their first transmission they will be travelling in different subcubes. **Q.E.D.**

The duration of Phase 2 is equal to

$$MS + m + 1.$$

In particular, MS is the time when the last packet starts transmission, and $m+1$ is the length of the path that the packet has to travel (plus the idle slots).

At the end of phase 2, all the packets originated at node s_i are in hypercube $R_{d-m}(r_i0^{d-m})$, where $0 \leq r_i < 2^m$. Note that these hypercubes are disjoint. Furthermore, a packet whose final destination is node t is located at the neighbor $r_iW_{d-m}(s_i) \oplus e_j$ of node $r_iW_{d-m}(s_i)$, where j is such that $r_iW_{d-m}(t) \in T_j(r_iW_{d-m}(s_i))$. Each of these neighbors has at most SM packets. Also, the packets with destinations in subcube $(*^mW_{d-m}(s_i))$ are located at node $r_iW_{d-m}(s_i)$.

Phase 3 (Subcube Scattering Phase):

In this phase M consecutive single node scatters are performed in each one of the $d-m$ dimensional disjoint subcubes $R_{d-m}(r_i0^{d-m})$, $r_i = 0, 1, \ldots, M-1$. A packet with final destination t is sent to node $r_iW_{d-m}(t)$; thus, node $r_iW_{d-m}(t)$ receives M packets (those originated at i and destined for the nodes in $H_m(t)$). Such a packet will be located at the beginning of Phase 3 at the root of the subtree $T_j(r_iW_{d-m}(s_i))$ where node $r_iW_{d-m}(t)$ also belongs.

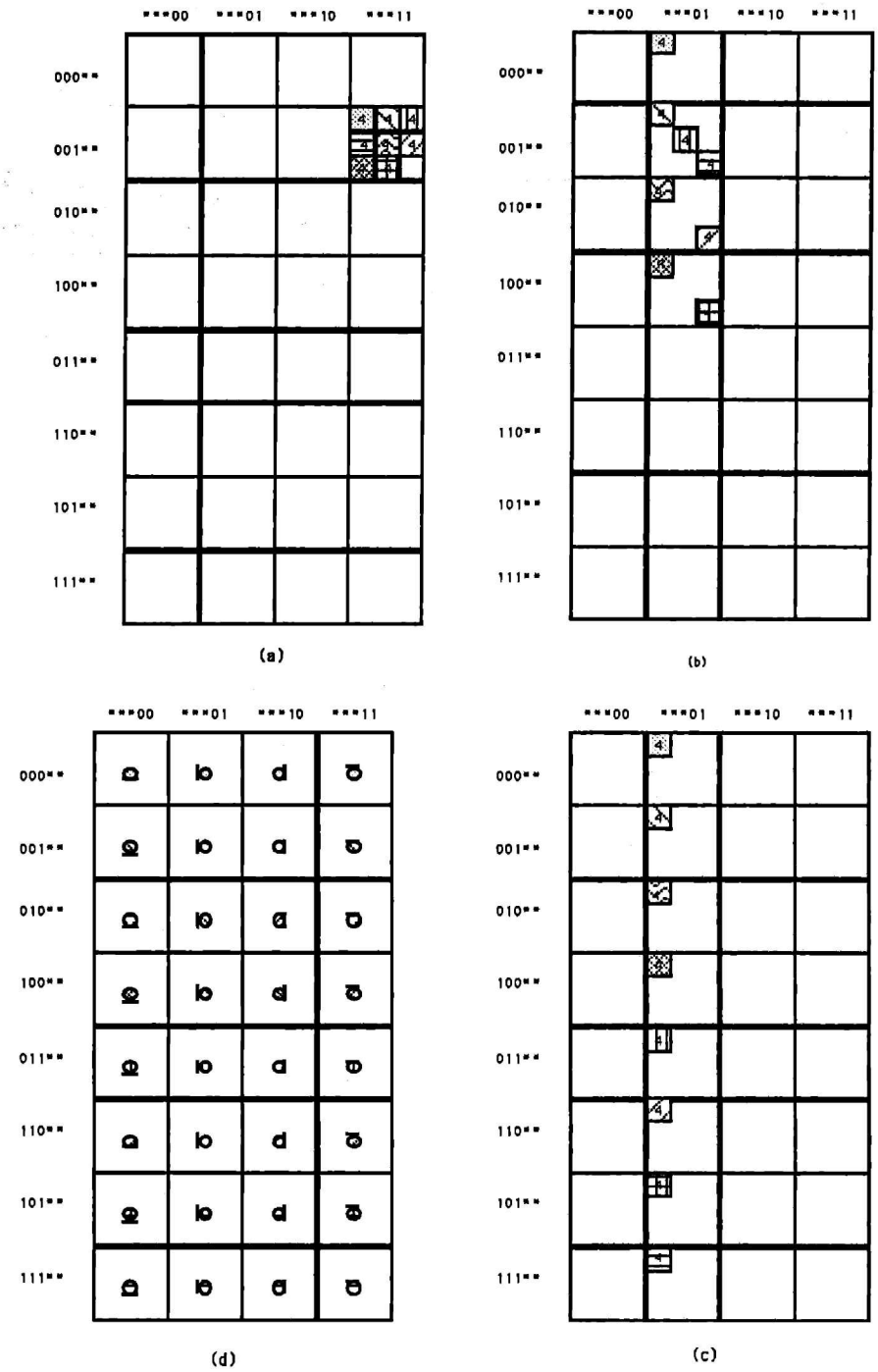

Figure 3.6: The hypercube PE algorithm for $N = 32$. The parallel prefix phase is not indicated. Each square in figures (a), (b), and (c) represents four packets (whose destination belongs to the same horizontal subcube). Only the packets with origin at the node with rank one are indicated. In going from (b) to (c) the single node scatter tree rooted at node 00001 (with subtrees rooted at nodes 00101, 01001, and 10001) has been used.

Node $r_i W_{d-m}(s_i) \oplus e_j$, $j \in \{0, 1, \ldots, d-m-1\}$ sends the packets downstream on its subtree $T_j(r_i W_{d-m}(s_i))$, starting with packets going to nodes which are furthest away. The time required for Phase 3 can then be found to be

$$M \left\lceil \frac{2N}{M(d-m)(d-m-1)} \right\rceil,$$

if $m < d-1$. In the case that $m = d-1$, Phase 3 is not necessary.

Phase 4 (Subcube Total Exchange Phase):

At the beginning of this phase node $r_i W_{d-m}(t)$ of subcube $H_m(t)$ has the packets originated at s_i whose destination belongs to $H_m(t)$. Thus a total exchange within each of these m-dimensional subcubes completes the task and requires time

$$\frac{M}{2}.$$

Adding up the times required for each phase we find that the time T_{PE} required to execute the partial exchange for $m < d-1$ satisfies

$$T_{PE} \leq \frac{M}{2} + M \left\lceil \frac{N/M - 1}{d-m} \right\rceil + M \left\lceil \frac{N}{M(d-m)\lfloor (d-m-1)/2 \rfloor} \right\rceil + 2dt_p + m + 1,$$

if $m < d-2$,

$$T_{PE} \leq 0.75N + 2dt_p + d,$$

if $m = d-1$, and

$$T_{PE} \leq \frac{7N}{8} + 2dt_p + d - 1,$$

if $m = d-2$. Of course, in the cases $m = d-1$ or $m = d$ it is preferable to use a total exchange algorithm, which requires only $N/2$ steps. The preceding equations give that for any $M = 2^m \leq N/2$ we have $T_{PE} = \Theta(\frac{N}{d-m})$. Comparing this relation with the upper bound found earlier we see that the proposed algorithm is of the optimal order of magnitude within rather small multiplicative factors.

3.8. WINDOW MULTINODE BROADCAST IN A HYPERCUBE

In this section we consider a *window multinode broadcast* (or WMNB) in a hypercube, defined as follows. We are given an m-dimensional subcube $\mathcal{S}$ of a d-dimensional hypercube $\mathcal{H}$. We

want to perform a multinode broadcast involving the nodes of $\mathcal{S}$, and we are allowed to use all links of $\mathcal{H}$. In other words, we want each node of $\mathcal{S}$ to broadcast a packet to every other node in $\mathcal{S}$, but packets can also get outside $\mathcal{S}$.

The window multinode broadcast task has not been considered previously in the literature, despite the fact that it arises often in applications. For example, when multitasking is used in a hypercube parallel computer, each task (or user) is allocated a subcube of the hypercube. Then the WMNB task arises in all the situations where a MNB arises. A WMNB arises also when an array is embedded in a hypercube so that each row of the array is embedded in a subcube of the hypercube (see [BeT89]). Then a MNB within a row or column of the array is a WMNB.

In what folows we give an algorithm that executes the WMNB in near-optimal time. Without loss of generality we assume that $\mathcal{S}$ is the subcube of $\mathcal{H}$ obtained by fixing to zero the $d-m$ most significant bits of the node addresses, that is, $\mathcal{S} = (0^{d-m}*^m)$. A way to execute the WMNB is to perform a MNB within $\mathcal{S}$, using only links in $\mathcal{S}$. This requires $(M-1)/m$ units of time (if we allow the splitting of packets), compared to $(M-1)/d$, which is the obvious lower bound. The question that arises is if we can do better by using the links in $\mathcal{H}-\mathcal{S}$. The algorithm that we will give executes the WMNB in $(M+1)/d$ units of time, which is only $2/d$ time units more than the lower bound. The algorithm consists of three phases and assumes that packets can be split and recombined without any overhead.

Phase 1: Every packet is split into d parts, called *mini-packets*, each of which is assigned a distinct class number c between 0 and $d-1$. A mini-packet requires $1/d$ time units for transmission over a link. In phase 1 of the algorithm each mini-packet with class $c \in \{m, m+1, \ldots, d-1\}$ is transmitted over the c-dimensional link of its source node. This phase requires $1/d$ time units.

Phase 2: This phase consists of m subphases. During subphase l, $l = 0, 1, \ldots, m-1$, each node transmits over its $l+c$ mod m link, if $c \in \{0, 1, \ldots, m-1\}$, or over its l link, if $c \in \{m, m+1, \ldots, d-1\}$, all the packets that it received during the previous subphases. Phase 2 requires

$$\frac{1}{d}(1 + 2 + \cdots + 2^{m-1}) = \frac{M-1}{d}$$

time units.

Phase 3: In this phase, every node outside $\mathcal{S}$ which has received a packet of class c transmits

it over its c-dimensional link. This would normally require $\frac{M-1}{d}$ time units, but since it can be pipelined with Phase 2 (every packet is sent over the c-dimensional link as soon as it is received), it requires only $1/d$ additional time units.

The WMNB algorithm requires $\frac{M+1}{d}$ time-steps, which is near-optimal. The previous WMNB algorithm can be modified using the ideas of Subsection 3.4.2 or 3.6.3 to avoid the splitting of packets, at a slight increase in the time complexity. We finish by noting that it may be interesting to consider a *Window Total Exchange* task, where a total exchange within $\mathcal{S}$ is performed and links outside $\mathcal{S}$ can be used. Algorithms for the WTE problem appear to be more complicated than WMNB algorithms.

CHAPTER FOUR

Dynamic Broadcasting Algorithms for Hypercubes and Meshes

So far we have dealt with static routing problems. Beginning with this chapter we turn our attention to routing algorithms that work in a dynamic environment. In the present chapter we consider the problem where broadcast requests are generated at random time instants at each node. In particular, in our model packets arrive at each node of a network according to a Poisson process, and each packet has to be broadcast to all the other nodes. Based on the partial multinode broadcast algorithms found in the previous chapter, we propose an on-line decentralized routing scheme to execute the broadcasts in this dynamic environment. We focus on the hypercube and the d-dimensional mesh (with or without wraparound) networks of processors; however, the results that we obtain for the dynamic broadcasting scheme apply to any topology, regular or not, for which partial multinode broadcast algorithms with certain properties can be found. We find an upper bound on the average delay required to serve a broadcast request, and we evaluate its stability region. We then apply the results to the case of a hypercube and a d-dimensional mesh network of processors. The stability region of the corresponding dynamic scheme tends to the maximum possible as the number of nodes of the hypercube or mesh tends to infinity. Furthermore, for any fixed load in the stability region, the average delay for both networks is of the order of the diameter of the network.

4.1. INTRODUCTION

In the previous chapter we considered *static* broadcasting tasks, that is we assumed that at time $t = 0$ some fixed, but unspecified nodes have to broadcast a packet once and for

all, and we proposed optimal algorithms to execute this task. Static broadcasting tasks in multiprocessor networks have been studied extensively in the literature ([BeT89], [BOS91], [Ho90], [JoH89], [LEN90], [VaB90b], [Sta91]). In this chapter we will consider the *dynamic version of the broadcasting problem.* We assume that broadcast requests are generated at each node according to a Poisson process with rate λ, independently of the other nodes. We propose routing algorithms that work under such a dynamic environment, and we evaluate their performance. The Poisson assumption is made only because the mathematics of the analysis require it and is inessential for the schemes that we present. We are interested in two performance criteria. The first is the average delay, that is, the average time between the arrival of a packet at a node, and the completion of its broadcast. The second criterion is the stability region of the scheme, that is, the maximum load that it can sustain with the average delay being finite. We set two objectives for a dynamic broadcasting scheme: stability for as big a load as possible, and average delay which is of the order of the diameter for any fixed load in the stability region.

The dynamic broadcasting problem is important for a variety of reasons. First, consider the case where iteration (3.1) takes place asynchronously. Each processor i computes at its own speed without waiting for the others, and broadcasts the updated value of x_i whenever it is available. Asynchronous parallel computation naturally results in a dynamic communication environment like the one we are considering. Asynchronous computation algorithms are increasing in importance (see e.g. [BeT89]) as a way to circumvent the synchronization penalty. The latter is a major cause of inefficiency in parallel computers, especially when the processors are not equally powerful, or when the load distribution is not balanced. In such a case, static algorithms, for example the MNB, become inefficient, since a fast processor would have to wait for all the other processors before starting a MNB. A second reason that makes algorithms for static tasks difficult to use is that the task must be detected and identified by the compiler, or the corresponding communication subroutine must be called explicitly by the programmer. It is plausible that the programmer and the compiler may fail to identify such communication tasks. Even more importantly, broadcasts may be generated in *run-time*, during the execution of a program. This poses a problem because in order to use precomputed static communication algorithms we must know the communication pattern in advance. Multitasking and time-sharing make the communications even less predictable, and the use of static communication algorithms more difficult. The preceding reasons motivate dynamic broadcasting schemes that will run continuously, and execute on-line the broadcast requests.

The only previous work on dynamic broadcasting we know of is that of Stamoulis [Sta91] for the hypercube network. There are two algorithms of Stamoulis that are most interesting from a theoretical point of view: the direct scheme, and the indirect scheme given in Chapters 6 and 7 of [Sta91], respectively. In the direct scheme, d spanning trees, where d is the dimension of the hypercube, are defined for each node, having the node as a root. A packet that arrives at a node selects at random one of the d trees of the node, and is broadcast on it. The direct scheme meets the stability objective described above, but its average delay analysis is approximate. In the indirect scheme, d spanning trees are defined in the hypercube. A packet that arrives at some node selects at random one of these trees. It is then sent to the root of that tree, and from there it is broadcast to all the other nodes using links of the tree. The indirect scheme meets the delay objective, but its stability region is not the maximum possible. Therefore, the two hypercube schemes of [Sta91] do not provably satisfy both performance objectives. Both the direct and the indirect schemes require less syncronization than our scheme. Also, if the splitting of packets is not allowed and the traffic is very light, the direct scheme has provably optimum delay.

Our dynamic broadcasting scheme has a fundamentally different philosophy: it relies heavily on finding efficient PMNB algorithms of the kind given in Chapter 3, that are used as a subroutine of the dynamic scheme. Furthermore, our scheme is very general: it applies to any network for which efficient PMNB algorithms can be found. For the hypercube and the d-dimensional mesh networks, our scheme has a stability region that tends to the maximum possible as the number of nodes tends to infinity, while its average delay for any fixed load in the stability region is of the order of the diameter of the network. Thus, our scheme compares favorably with Stamoulis' hypercube algorithms none of which meets optimally the stability and the delay objective.

Our dynamic broadcasting scheme and the corresponding performance analysis apply to any network for which we can find communication algorithms that execute the PMNB communication task in time

$$XM + V,$$

where M is the number of nodes that have a packet to broadcast, called *active nodes*, and X, V are scalars that are independent of M (they may depend on the size of the network). For networks where such PMNB algorithms exist, we can easily devise corresponding dynamic broadcasting schemes that satisfy some average packet delay and stability guarantees. The dynamic broadcasting scheme consists, merely, of executing successive PMNB algorithms, each

starting after the previous one has finished. Our scheme is modelled after reservation and polling schemes for multiaccess communication. The network is viewed as a channel, and the nodes as users of the channel. For analytical purposes, the first V time units of the PMNB algorithm are considered as a reservation interval, where some organizational work is performed, and the following MX time units as a data interval, where users with reservations transmit a packet.

Our dynamic broadcasting scheme requires the existence of a partial multinode broadcast algorithm with certain properties. In Chapter 3, we have already presented such partial multinode broadcast algorithms for the hypercube and the d-dimensional mesh networks. The PMNB algorithms of Subsections 3.6.2 or 3.6.3 for hypercubes, and of Subsections 3.4.1 and 3.4.2 for d-dimensional meshes, have the required properties, and the corresponding dynamic schemes meet our stability and average delay objectives.

The structure of the chapter is the following. In Section 4.2 we describe the dynamic broadcasting scheme in a given network, assuming that a PMNB algorithm with certain properties is available for that network. We also state the dynamic broadcasting theorem, which is the main result of the paper. In Section 4.3 we evaluate the performance of our dynamic broadcasting scheme. In particular, in Subsection 4.3.1 we describe an auxiliary queueing system, which we will use to prove the dynamic broadcasting theorem. In Subsection 4.3.2 we prove the dynamic broadcasting theorem, which gives an estimate on the average packet delay of the dynamic broadcasting scheme. Section 4.4 applies the dynamic broadcasting theorem to the case of the hypercube, while Section 4.5 applies it to the case of the d-dimensional mesh.

4.2. DYNAMIC BROADCASTING SCHEMES

In this section we will describe the dynamic broadcasting scheme for a general network. We will assume that an algorithm that executes the PMNB task in that network is given, and that it requires $XM + V$ time units, where M is the number of active nodes, that is, the nodes that have a packet to broadcast, and X, V are scalars independent of M. We also assume that during the PMNB algorithm each node learns the number of active nodes M.

Our scheme is merely a repetition of successive partial multinode broadcast algorithms, each starting when the previous one has finished (see Fig. 4.1). The time axis is, therefore, divided

into PMNB intervals. Within each PMNB interval, a PMNB is executed, involving exactly one packet from each of the M nodes that are active at the start of the interval. Each PMNB interval is divided into two parts. The first part is called *reservation interval*. Its duration can be upper bounded by a known constant V that depends only on the size of the network, and is independent of the number of active nodes M. During the reservation interval each active node s can be viewed as making a reservation for the broadcast interval. In the PMNB algorithms for the hypercube and the d-dimensional mesh of Chapter 3 this is done simply by setting $x_s = 1$ in the rank computation phase. Usually, in the reservation interval some global information is gathered at the nodes (e.g., the total number of active nodes M, and other information), and some additional organizational work is performed. For example, in the PMNB algorithms described in Subsections 3.6.2 and 3.6.3 for the hypercube, and in Subsections 3.4.1 and 3.4.2 for the d-dimensional mesh, the packets move during the reservation interval to more favorable intermediate locations. The second part of a PMNB interval is called *broadcast interval*. Its duration is equal to XM, and is therefore known once M is known. The broadcast interval is empty if there are no packets to broadcast ($M = 0$). Thus, even though the duration of each partial multinode broadcast is random (because packet arrivals are random), it is known to all the nodes of the network, because each node learns during the broadcast interval the number M of active nodes and, from there, the duration of the following broadcast interval. Therefore, if the nodes initiate the dynamic broadcast scheme at the same time, no further synchronization is needed. Even if the clocks of the processors are not accurate, a node can count how many packets it has received and detect local termination of the current PMNB period when their number equals M. The forward subphase of the parallel prefix phase serves as a termination detection mechanism (see [BeT89], pp. 571-579), and the reverse subphase as a way to announce the termination to the nodes. However, if the transmission time is different for each link, perfect global synchronization cannot be achieved, and one should expect some degradation in the performance (mainly in the duration of the monotone routing phase, since the broadcast phase is rather tolerant to synchronization errors). For the PMNB algorithms proposed in Chapter 3, in the broadcast interval the packets are broadcast from the intermediate locations to all other nodes. The details of the broadcast interval are, however, irrelevant, and all that matters for our purposes, is that the duration of the broadcast interval is less than or equal to MX.

For the optimal PMNB algorithm for $N = 2^d$-processor hypercubes described in Subsection 3.6.2, the reservation interval is taken to consist of the parallel prefix and the isotone routing

phases, and requires $V_h = 2dt_p + 2$ time units. Similarly, in the optimal PMNB algorithm for $N = p^d$-processor meshes of Subsection 3.4.1 the reservation interval consists of the first two phases, and requires at most $V_m = 2d(p-1)t_p + (1+1/\gamma)(p-1)$ time units for a mesh, where $\gamma = 2$ for a mesh with wraparound, and $\gamma = 1$ for a mesh without wraparound. It is important that V_h, and V_m are constant, in other words independent of the number of active nodes.

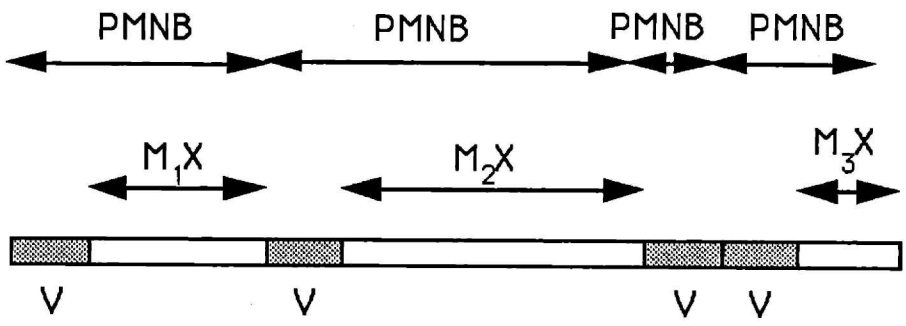

Figure 4.1: The dynamic broadcasting scheme. Each PMNB interval consists of two intervals: a reservation interval (marked by gray) of duration V, and a broadcast interval of duration MX, where M is the number of active nodes at the start of the PMNB interval.

It is important for the performance of the dynamic scheme that the duration of the PMNB algorithms is *linear* in the number of active nodes M, with the constant of proportionality being the smallest possible. In the hypercube case the constant of proportionality was equal to

$$\frac{1}{d}\frac{N-1}{N},$$

while for the d dimensional meshes it was

$$\frac{1}{\gamma d}\frac{N-1}{N},$$

which are both the smallest possible.

The main theorem that we prove in the paper is the following.

Dynamic Broadcasting Theorem: Assume that for a given N-processor network there exists an algorithm that executes the PMNB communication task in time

$$XM + V,$$

where M is the number of nodes that have a packet to broadcast and X, V are scalars that are independent of M (they may depend on the size of the network). Assume that during the

PMNB algorithm each node learns the value of M. Then the dynamic broadcasting scheme that uses this PMNB algorithm as described above has the following performance characteristics. If the packets to be broadcast arrive at each node of the network according to a Poisson process with rate λ, independently of the other nodes, the average packet delay T satisfies

$$T = W + X + aNX \leq W + X + \min\left(\frac{N-1}{2}X, \rho W\right), \tag{4.1}$$

where

$$W = \frac{\rho X}{2(1-\rho-\lambda V)} + \frac{(1-\rho)V}{2(1-\rho-\lambda V)} + \frac{(1-\rho a-\lambda V)V}{1-\rho-\lambda V},$$

$$\rho = \lambda N X,$$

and a is a scalar satisfying

$$\frac{\overline{M} + (\hat{M}-1)(2\overline{M}-\hat{M})}{2N\overline{M}} - \frac{1}{2N} \leq a \leq \frac{1}{2} - \frac{1}{2N},$$

where

$$\overline{M} = \frac{\lambda N V}{1-\rho},$$

and $\hat{M}$ is the smallest integer which is strictly larger than $\overline{M}$.

Note that the dynamic broadcasting scheme is stable for $\rho \leq 1-\lambda V$, or by using the relation $\lambda = \rho/(NX)$,

$$\rho \leq \frac{1}{1+V/(NX)}.$$

The dynamic broadcasting theorem is proved in the following section.

4.3. ANALYSIS OF THE DYNAMIC BROADCASTING SCHEME

In this section we will prove the dynamic broadcasting theorem. We first describe an auxiliary queueing system that will be used in the main proof given in Subsection 4.3.2.

4.3.1. Limited Service Gated Reservation System with Shared Reservation and Data intervals

In this subsection we describe an auxiliary queueing system, called *limited service gated reservation system with shared reservation and data intervals*, which is a reservation system for multiaccess communication. We are interested in the average delay required to serve a packet in this system. This average delay will be used in the next subsection to evaluate the average packet delay of the dynamic broadcasting scheme. The analysis of the auxiliary queueing system is based on unpublished research by D. P. Bertsekas and R. G. Gallager. We will give this analysis in detail, since it is important for our purposes and it does not appear anywhere else.

The auxiliary queueing system is defined as follows. Consider N traffic streams, each corresponding to a different user, which share a common channel. The channel is used both for packet transmissions and reservations. In particular, the time axis is divided into data intervals, where actual data is transmitted, and reservation intervals used for scheduling future data. Each user has a separate queue, and the queues are served in cyclical order. Users make reservations during the same reservation interval, and transmit at most one packet each in the subsequent data interval. A packet can be transmitted in a data interval only if it arrived before the beginning of the previous reservation interval. For this system the following theorem holds.

Theorem 1: Let the arrival processes of packets at the users of the system be independent Poisson processes, each with rate λ. Let also $\overline{X}$, $\overline{X^2}$ be the first and second moments of the packet transmission times and $\overline{V}$, $\overline{V^2}$ be the first and second moments of the duration of a reservation interval. Then the mean waiting time in queue for this system is

$$W = \frac{\lambda N \overline{X^2}}{2(1-\rho-\lambda\overline{V})} + \frac{(1-\rho)\overline{V^2}}{2(1-\rho-\lambda\overline{V})\overline{V}} + \frac{(1-\rho a-\lambda\overline{V})\overline{V}}{1-\rho-\lambda\overline{V}}, \tag{4.2}$$

where $\rho = \lambda N \overline{X}$ is the utilization factor, and a satisfies

$$\frac{\overline{k}+(\hat{k}-1)(2\overline{k}-\hat{k})}{2N\overline{k}} - \frac{1}{2N} \leq a \leq \frac{1}{2} - \frac{1}{2N},$$

where

$$\overline{k} = \frac{\lambda N \overline{V}}{1-\rho}$$

is the average number of packets per data interval, and $\hat{k}$ is the smallest integer which is strictly larger than $\overline{k}$.

Proof: Consider the i^{th} packet arrival into the system and suppose that the user associated with packet i is user j. This packet must wait in queue for the residual time R_i until the end of the current packet transmission or reservation interval. It must also wait for the transmission of the N_i packets that must be transmitted before packet i. Finally the packet must wait for the duration of reservation intervals. Thus, the expected waiting time of packet i is

$$E(W_i) = E(R_i) + E(N_i)\overline{X} + E(Y_i),$$

where Y_i is the duration of all the whole reservation intervals during which packet i must wait before being transmitted. The expected waiting time is therefore

$$W = \lim_{i\to\infty} E(R_i) + E(N_i)\overline{X} + E(Y_i). \tag{4.3}$$

From Little's law we get

$$E(N_i) = \lambda N W. \tag{4.4}$$

Let Q_i be the number of packets in the queue of user j found by packet i upon arrival, and m_i be the number (0 or 1) of packets of user j that will start transmission between the time of arrival of packet i and the end of the frame at which packet i arrives. Then

$$E(Y_i) = (1 + E(Q_i) - E(m_i))\,\overline{V}. \tag{4.5}$$

From Little' law we have

$$Q = \lim_{i\to\infty} E(Q_i) = \lambda W. \tag{4.6}$$

The mean residual time $R = \lim_{i\to\infty} R_i$ can be calculated as in [BeG87] to be

$$R = \frac{\lambda N \overline{X^2}}{2} + \frac{(1-\rho)\overline{V^2}}{2\overline{V}}. \tag{4.7}$$

Combining Eqs. (4.3)-(4.7) we get

$$W = \lambda N W \overline{X} + \left(1 + \lambda W - \lim_{i\to\infty} E(m_i)\right)\overline{V} + \frac{\lambda N \overline{X^2}}{2} + \frac{(1-\rho)\overline{V^2}}{2\overline{V}}. \tag{4.8}$$

To find W, it remains to calculate $\lim_{i\to\infty} E(m_i)$.

There are two possibilities regarding the time of arrival of packet i.

a) packet i arrives during a reservation interval. This event, call it A, has steady state probability $1-\rho$:

$$P(A) = 1 - \rho.$$

Since the ratio of the average data interval length to the average reservation interval length is $\rho/(1-\rho)$ we see that the average steady state length of data interval is $\rho\overline{V}/(1-\rho)$. Therefore, the average steady state number of packets per user in a data interval is

$$\frac{\rho\overline{V}}{(1-\rho)N\overline{X}} = \frac{\lambda\overline{V}}{1-\rho}.$$

This also equals the steady state value of $E(m_i \mid A)$ in view of the symmetry with respect to the users:

$$\lim_{i\to\infty} E(m_i \mid A) = \frac{\lambda\overline{V}}{1-\rho}.$$

b) Packet i arrives during a data interval. This event, call it B, has steady state probability ρ:

$$P(B) = \rho.$$

Denote

$$a = \lim_{i\to\infty} E(m_i \mid B),$$

$$a_k = \lim_{i\to\infty} E(m_i \mid B, \text{ the data interval of arrival of packet } i \text{ contains } k \text{ packets}).$$

Assuming $k > 0$ packets are contained in the data interval of arrival, there is equal probability $1/k$ of arrival during the transmission of any of these packets. Therefore

$$a_k = \sum_{n=1}^{k} \frac{1}{k}\frac{k-n}{N} = \frac{k-1}{2N}.$$

Let $P(k)$ be the unconditional steady-state probability that a data interval contains k packets, and $E(k)$ and $E(k^2)$ be the corresponding first two moments. Then we have by Bayes' rule

$$\lim_{i\to\infty} P(\text{The data interval of arrival of packet } i \text{ contains } k \text{ packets}) = \frac{kP(k)}{E(k)}.$$

Combining the preceding equations we have

$$a = \sum_{k=1}^{N} \frac{kP(k)}{E(k)} a_k = \sum_{k=1}^{N} \frac{P(k)k(k-1)}{2E(k)N} = \frac{E(k^2)}{2NE(k)} - \frac{1}{2N}.$$

We have already shown as part of the analysis of case a) above that

$$E(k) = \frac{\lambda N\overline{V}}{1-\rho} \tag{4.9}$$

so there remains to estimate $E(k^2)$. We have

$$E(k^2) = \sum_{k=1}^{N} k^2 P(k).$$

If we maximize the quantity above over the distribution $P(k)$, $k = 0,1,\ldots,N$, subject to the constraints $\sum_{k=0}^{N} P(k) = 1$, $\sum_{k=0}^{N} kP(k) = E(k)$, and $P(k) \geq 0$ (a linear programming problem) we find that the maximum is obtained for $P(N) = E(k)/N$, $P(0) = 1 - E(k)/N$, and $P(k) = 0$, $k = 1,2,\ldots,N-1$. Therefore

$$E(k^2) \leq NE(k).$$

Similarly if we minimize $E(k^2)$ subject to the same constraints we find that the minimum is obtained for $P(\hat{k}-1) = \hat{k} - E(k)$, $P(\hat{k}) = 1 - (\hat{k} - E(k))$ and $P(k) = 0$ for $k \neq \hat{k}-1, \hat{k}$, where $\hat{k}$ is the integer for which $\hat{k} - 1 \leq E(k) < \hat{k}$. Therefore

$$E(k^2) \geq (\hat{k}-1)^2(\hat{k} - E(k)) + (\hat{k})^2\left(1 - (\hat{k} - E(k))\right).$$

After some calculations this relation can also be written

$$E(k^2) \geq E(k) + (\hat{k}-1)(2E(k) - \hat{k}) \quad \text{for} \quad E(k) \in [\hat{k}-1, \hat{k}),\ \hat{k} = 1,2,\ldots,N.$$

Note that the lower bound above is a piecewise linear function of $E(k)$, and equals $(E(k))^2$ at the breakpoints $\hat{k} = 1,2,\ldots,N$. Summarizing the bounds we have

$$\frac{E(k) + (\hat{k}-1)(2E(k) - \hat{k})}{2NE(k)} - \frac{1}{2N} \leq a \leq \frac{1}{2} - \frac{1}{2N}, \tag{4.10}$$

where $\hat{k}$ is the positive integer for which

$$\hat{k} - 1 \leq E(k) < \hat{k}.$$

Note that as $E(k)$ approaches its maximum value N (i.e., the system is heavily loaded), the upper and lower bounds coincide. By combining the results for cases a) and b) above we have

$$\lim_{i\to\infty} E(m_i) = P(A) \lim_{i\to\infty} E(m_i \mid A) + P(B) \lim_{i\to\infty} E(m_i \mid B) = (1-\rho)\frac{\lambda\overline{V}}{1-\rho} + a\rho,$$

or finally

$$\lim_{i\to\infty} E(m_i) = \lambda\overline{V} + a\rho$$

where a satisfies Eq. (4.10). Using Eq. (4.8) and the expressions derived we obtain Eq. (4.2) where $E(K)$ is given by Eq. (4.9), and $\hat{k}$ satisfies $\hat{k} - 1 \leq E(k) < \hat{k}$. **Q.E.D.**

Note that the formula for the mean waiting time becomes exact in the limit both as $\rho \to 0$ (light load), and as $\rho \to 1 - \lambda\overline{V}$ (heavy load), in which case $E(k) \to N$ and $a \to 1/2 - 1/(2N)$. The formula is also exact if $N = 1$ ([BeG87]).

4.3.2. Main Proof

We now complete the proof of the dynamic broadcasting theorem.

Proof of the Dynamic Broadcasting Theorem: In a PMNB period each node that has a packet to broadcast at the start of the period participates with exactly one packet. Let M be the number of such nodes at the start of a period. Each of these nodes can be viewed as making a reservation during the reservation interval. The duration of the subsequent broadcast interval is at most MX time units.

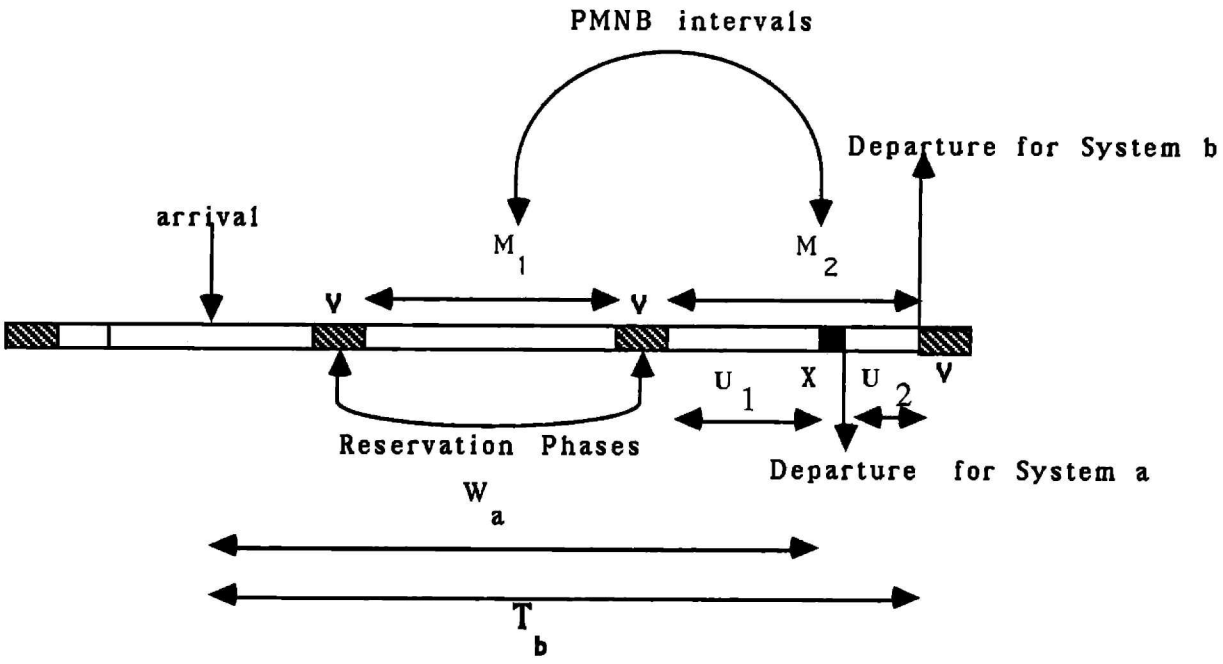

Figure 4.2: Reservation and broadcast (or data) intervals for the network broadcasting scheme, and the auxiliary queueing system.

The dynamic broadcasting scheme will be called system "b" (for "broadcast"). We also consider the limited service gated reservation system with shared reservation and data intervals presented in the previous subsection. This system will be called system "a" (for "auxiliary"). Let the reservation interval of system "a" be constant and equal to V, and the service time of a packet be constant again and equal to X.

Consider the following analogy between systems "a" and "b". Let a data interval of system "a" correspond to a broadcast interval of system "b", a user of system "a" correspond to a node

of system "b", and a packet arrival of system "a" correspond to a broadcast request arrival of system "b". Note the similarities between the two systems. During a data interval of system "a" (or broadcast interval of system "b") at most one packet (or broadcast request, respectively) from each user (or node, respectively) can be served. It is easy to see that the probability distributions of the length of the reservation intervals, the data (or broadcast) intervals, and the number of users (or nodes) served in a data interval are identical for both systems. In particular, the length of a reservation interval of both systems is equal to V by construction. The length of a broadcast interval of system "b" is equal to MX, where M is the number of active nodes. Similarly, the length of a data interval of system "a" is equal to MX, where M is the number of non-empty queues. The only difference between the two systems is that in system "b" a broadcast request completes service at the end of a PMNB period, while in system "a" packets complete service at times jX, $j = 1, 2, \ldots, M$, from the beginning of the data interval.

The waiting time W_a in queue for a packet of the auxiliary system is given from Eq. (4.10) of Subsection 4.3.1 with $\overline{X}, \overline{X^2}, \overline{V}, \overline{V^2}$ replaced by X, X^2, V, V^2, respectively. The average delay (queueing plus service time) for the auxiliary system "a" is

$$T_a = W_a + X.$$

Let U_1 be the average time between the beginning of a data interval of system "a", and the time that a packet served in this data interval starts transmission (see Fig. 4.2). Similarly, let U_2 be the average time between the end of the transmission of a packet of system "a" and the end of the data interval in which it is served. Then it can be proved by using arguments similar to those used in the proof of Theorem 1 that

$$U_1 = U_2 = aNX \leq \frac{N-1}{2}X.$$

It can also be seen that

$$U_1 = U_2 \leq E(N_i)X = \lambda N W_a X = \rho W_a, \tag{4.11}$$

where Eq. (4.3) was used. The average packet delay T_b of the broacasting scheme is

$$T_b = T_a + U_2 = W_a + X + aNX \leq W_a + X + \min\left(\frac{N-1}{2}X, \rho W_a\right). \tag{4.12}$$

This completes the proof. **Q.E.D.**

Note that for light load ($\rho \approx 0$) the dynamic broadcasting theorem gives

$$T_b \leq 1.5V + X, \qquad \rho \approx 0.$$

Until now we did not have to assume any particular topology for the multiprocessor network. The dynamic broadcasting scheme and the dynamic broadcasting theorem apply to any network for which a PMNB algorithm with certain properties exists. In Chapter 3 we have presented such algorithms for a hypercube and a d-dimensional mesh network of processors. We believe that such algorithms exist for many other regular topologies (e.g., folded-cubes, and meshes of trees). In the next two sections we will apply the dynamic broadcasting theorem in the hypercube and the mesh network of processors.

4.4. PERFORMANCE OF THE DYNAMIC BROADCASTING SCHEME FOR HYPERCUBES

In this section we evaluate the average packet delay of the hypercube dynamic broadcasting scheme. It is easy to see that the PMNB algorithms of Subsections 3.6.2 and 3.6.3 both satisfy the conditions of the dynamic broadcasting theorem. We assume that one of these two algoritms is used as a component of the dynamic scheme, and we apply the dynamic broadcasting theorem.

The reservation interval of the hypercube PMNB algorithm of Subsection 3.6.2 consists of the rank computation and the packing phases, and has duration $V = 2dt_p + 2$. If the near-optimal PMNB algorithm of Subsection 3.6.3 that does not allow the splitting of packets is used then the reservation interval consists of the class computation part, and the rank computation and packing phases of the main part, and has duration $2d + 4dt_p$. The analysis to be given can be carried out for both cases. In what follows we will give the results for the case where the PMNB algorithm of Subsection 3.6.2 is used; if the algorithm of Subsection 3.6.3 is used, V should be replaced by $2d + 4dt_p$ throughout this section. During the reservation interval every node s that has a packet to broadcast makes a reservation for the broadcast interval by setting $x_s = 1$ in the parallel prefix phase of Subsection 3.6.2 (and in the class computation phase if the algorithm of Subsection 3.6.3 is used). In addition to that, during the packing phase the packets move to more favorable intermediate destinations. Following each reservation interval, there is a broadcast interval, which is the last phase of the algorithms of Subsection 3.6.2 (or

Subsection 3.6.3), and has duration

$$\frac{M}{d}\frac{N-1}{N}$$

time units. Each node can participate with at most one packet per PMNB period, and this packet must have arrived prior to the beginning of that PMNB period.

We define the *hypercube utilization factor* as

$$\rho = \frac{\lambda(N-1)}{d}. \tag{4.13}$$

For a given load, ρ is equal to the ratio of the average total number of transmissions per unit of time necessary to execute the broadcasts (each broadcast requires $N-1$ transmissions), over the total number of links of the hypercube. To find a necessary condition for stability for any broadcasting scheme, note that λN broadcast requests are generated on the average per unit of time in the hypercube, each requiring at least $N-1$ transmissions to be completed. Since there are dN links, a necessary condition for stability is

$$\lambda N(N-1) \leq dN,$$

or else

$$\rho < 1.$$

As we will see, our scheme does not guarantee stability for any $\rho < 1$, but it does for ρ very close to 1.

The PMNB algorithm of Subsection 3.6.2 satisfies the conditions required by the dynamic broadcasting theorem to apply. Using the dynamic broadcasting theorem, and substituting X by $(N-1)/(dN)$, we see that the average delay T_h of the hypercube dynamic broadcasting scheme using the PMNB algorithm of Subsection 3.6.2 satisfies

$$T_h = W + \frac{a(N-1)}{d} + \frac{N-1}{N}\frac{1}{d} \leq W + \frac{1}{d} + \min\left(\rho W, \frac{N-1}{2d}\right), \tag{4.14}$$

with

$$W = \frac{\rho}{2d(1-\rho-\lambda V)}\frac{N-1}{N} + \frac{(1-\rho)V}{2(1-\rho-\lambda V)} + \frac{(1-\rho a-\lambda V)V}{1-\rho-\lambda V} =$$
$$= \frac{N-1}{N}\frac{\rho}{2d(1-\rho-\lambda V)} + \frac{V(1.5-0.5\rho-\rho a-\lambda V)}{1-\rho-\lambda V}, \tag{4.15}$$

where

$$V = 2dt_p + 2,$$

ρ is given by Eq. (4.13), and

$$\frac{\overline{M} + (\hat{M} - 1)(2\overline{M} - \hat{M})}{2N\overline{M}} - \frac{1}{2N} \leq a \leq \frac{1}{2} - \frac{1}{2N},$$

$$\overline{M} = \frac{\rho V d}{1 - \rho},$$

where $\hat{M}$ is the smallest integer which is larger than $\overline{M}$. ($\overline{M}$ is the average number of broadcast requests served in a PMNB period).

For any fixed load which satisfies $1 - \rho - \lambda V > 0$, we can see from Eqs. (4.14) and (4.15) that the average packet delay is $\Theta(V) = \Theta(d \cdot t_p)$. For an almost empty hypercube, $\rho \approx 0$, Eqs. (4.14) and (4.15) give

$$T_h \leq 1.5V + \frac{1}{d} = 3dt_p + 3 + \frac{1}{d}, \qquad \rho \approx 0.$$

For light load it may be preferable to use for each node a spanning tree rooted at that node (see [Sta91]). For $\rho \approx 0$, conflicts will be rare and the delay will be equal to d units of time (if packets are not split), or equal to 2 units of time (if packets can be split). The increase of the delay for light load is a generic drawback of reservation schemes.

The following theorem gives the order of magnitude of the average delay of the scheme at heavy load.

Theorem 2: In the limit, as $1 - \lambda V - \rho \to 0$, we have

$$T_h = O\left(\frac{V}{1 - \rho - \lambda V}\right).$$

Proof: Let $\rho = 1 - \lambda V - \epsilon$ with $\epsilon \to 0$. Then, as shown in Subsection 4.3.1, $a \to 0.5 - 0.5/N$. Let $a = 0.5 - 0.5/N - \delta$ with $\delta \to 0$. Then from Eq. (4.14)

$$T_h \leq \frac{\rho}{2d\epsilon} + \frac{V(0.5 + \epsilon + 0.5\rho/N + \rho\delta)}{\epsilon} + \frac{1}{d} + \min\left(\frac{V(0.5 + \epsilon + 0.5\rho/N + \rho\delta)}{\epsilon}, \frac{N-1}{2d}\right)$$
$$= O\left(\frac{V}{\epsilon} + \frac{\rho}{d\epsilon}\right),$$

where the asymptotics in the previous expression are taken with respect to ϵ. **Q.E.D.**

As proved in [Sta91], the average delay of any dynamic broadcasting scheme grows at least as $\Omega\left((1-\rho)^{-1}\right)$. Since the term $\rho dV/(N-1)$ goes to 0 (very fast) as $N \to \infty$, our dynamic broadcasting scheme has good behavior for heavy load (ρ close to 1) when the number of nodes of the hypercube is large.

The hypercube broadcasting scheme is stable for

$$\rho \leq \frac{1}{1+(2dt_p+2)d/(N-1)},$$

which is very close to the maximum ρ that can be accomodated by *any* scheme (which is the unit), and tends to it as $N \to \infty$. The reason we get this remarkable result is the efficiency of the PMNB algorithm of Subsection 3.6.2, and the small overhead introduced by the reservation intervals (parallel prefix and packing phases). A similar result would be true if the algorithm of Subsection 3.6.3 were used; the corresponding stability region is given by

$$\rho \leq \frac{1}{1+(2d+2dt_p)d/N}.$$

4.5. PERFORMANCE OF THE DYNAMIC BROADCASTING SCHEME FOR D-DIMENSIONAL MESHES

In this section we evaluate the performance of the dynamic broadcasting scheme for d-dimensional meshes (with or without wraparound) with $N = p^d$ nodes. The analysis will be similar to that of the hypercube case.

For d-dimensional meshes we have found algorithms that satisfy the conditions of the dynamic broadcasting theorem. The analysis to be given applies to both the case where the PMNB algorithm of Subsection 3.4.1 (where packets are split), and to the case where the algorithm of Subsection 3.4.2 (where packets are split) is used. In what follows we will assume that the former algorithm is used.

The duration of the PMNB algorithm for a d-dimensional meshes with $N = p^d$ processors was found in Subsection 3.4.1 to be equal to

$$T_{PMNB} = XM + V,$$

where

$$X = \frac{1}{\gamma d}\frac{N-1}{N}, \tag{4.16}$$

$$V = 2d(p-1)t_p + \left(1+\frac{1}{\gamma}\right)(p-1), \tag{4.17}$$

and $\gamma = 1$ for the d-dimensional array and $\gamma = 2$ for the d-dimensional torus.

The average packet delay is bounded as in Eq. (4.1), where X and V are given by Eqs. (4.16) and (4.17).

The scalar

$$\rho = \lambda N X = \frac{\lambda(N-1)}{\gamma d} \tag{4.18}$$

is called the *mesh utilization factor*, for reasons that will become evident soon. To find necessary conditions for stability for any broadcasting scheme, consider a d-dimensional array or torus, the outgoing links of node $(00\cdots0)$ and the traffic that passes through them. There are $2d$ such links for the torus, and d for the array. Thus, for stability we must have

$$\lambda(N-1) \leq \gamma d, \tag{4.19}$$

or

$$\rho \leq 1, \tag{4.20}$$

no matter what broadcasting scheme we use. For the torus ($\gamma = 2$) and a given load, ρ is equal to the ratio of the average number of transmissions per unit of time necessary to execute the broadcasts (each broadcast requires $N-1$ transmissions), over the total number of links of the network. For the array ($\gamma = 1$), which is not a symmetric network, ρ equals the average fraction of time during which the links of node $(00\cdots0)$ have to be used under any broadcasting scheme.

Our algorithm is guaranteed to be stable for $\rho < 1 - \lambda V$. Using Eqs. (4.18) and (4.17) we find

$$\rho < 1 - \rho\frac{\gamma d V}{N-1}$$

or

$$\rho < \frac{1}{1 + \frac{d\gamma\left(2d(p-1)t_p + (1+1/\gamma)(p-1)\right)}{p^d - 1}}. \tag{4.21}$$

The right hand side of the preceding inequality is very close to the maximum possible load that a d-dimensional array or torus could sustain. Indeed, as the number of nodes p^d tends to infinity, d^2p/p^d tends to zero. Thus, the right hand side of Eq. (4.21) tends to one, which, in view of Eq. (4.20), is the maximum utilization that can be accomodated by the network.

For any fixed ρ in the stability region, Eq. (4.1) gives

$$T = O(V) = O(pdt_p + p),$$

where we have used Eq. (4.17). Since the diameter of a d-dimensional mesh is $\Theta(pd)$, the previous relation gives $T = O(pdt_p + p) = O(t_p \cdot diameter + p)$ for fixed ρ in the stability region. In particular, for light load ($\lambda \approx 0,\ \rho \approx 0$) we get from Eq. (4.1) that

$$T \leq 1.5V + X, \quad (\rho \approx 0).$$

CHAPTER FIVE

Performance of Hypercube Routing Schemes With or Without Buffering

In this chapter two different hypercube routing schemes, called the *simple* and the *priority* scheme, are analyzed. We consider both the unbuffered and the buffered version of each scheme, and evaluate their throughput for random multiple node to node communications. The results obtained are approximate, but very accurate, as simulation results indicate, and they are given in particularly interesting forms. We find that little buffer space (between one and three packets per link) is necessary and adequate to achieve throughput close to the case where infinite buffer space is available. We also consider two deflection routing schemes, called the *simple* and the *priority deflection* schemes. We evaluate their throughput using simulations, and compare it to that of the priority scheme. We also present some results on the probability distribution of the distance of packets from their destination in shortest path routing algorithms in regular networks.

5.1. INTRODUCTION

The traffic environment under which the routing schemes are analyzed is stochastic. In particular, we assume that packets having a single destination are generated at each node of a hypercube according to some probabilistic rule. The destinations of the new packets are uniformly distributed over all the hypercube nodes. Packets have equal length and require one unit of time to be transmitted over a link. We are interested in the case where the packets are generated over an infinite time horizon, and we want to evaluate the average throughput of the

network when it reaches steady state.

One-to-one routing has been extensively analyzed in the literature for a variety of network topologies. A line of research that has been pursued is related to the so-called *randomized algorithms*. In randomized algorithms a packet is sent to some random intermediate node before being routed to its destination. Randomized routing algorithms were first proposed by Valiant [Val82], and Valiant and Brebner [VaB81] for the hypercube network, they were extended to networks of constant degree by Upfal [Upf84], and they were improved to achieve constant queue size per node by Pippenger [Pip84]. Later, Mitra and Cieslack [MiC87], and Hajek and Cruz [HaC87] proved similar results for the extended Omega network, and Greenberg and Leiserson [GrL89] for the fat tree network. The results on randomized routing are usually of the following nature: it is proved that for sufficiently large size of the network, a permutation task can be executed in time less than some multiple of the network diameter with high probability. A disadvantage of this line of research is that the results obtained are of an asymptotic nature, and apply only to permutation tasks (or partial permutations, and concatenation of permutations). Another disadvantage is that the communication task is proved to be executed within the specified time with "high probability" instead of "always".

A second line of research, which is closer to our work in this chapter, is the evaluation of the throughput of a network in steady state. A routing scheme, called deflection routing, has been analyzed by Greenberg and Goodman [GrG86] for mesh networks (numerically), Greenberg and Hajek [GrH90] for hypercubes (analytically), Maxemchuk [Max87], [Max89] for the Manhattan network and the Shuffle Exchange (numerically and through simulations), and Varvarigos [Var90] for hypercubes (for a priority deflection scheme, through the formulation of a Markov chain). The analysis in these works was approximate (also, usually numerical), or it involved simulations. Since deflection routing is a way to circumvent the need for large buffers, buffer size played a minor role in these analyses. Dias and Jump [DiJ81] analyzed approximately the performance of buffered Delta networks. Stamoulis [Sta91] analyzed a greedy scheme in hypercubes and butterflies for infinite buffer space per node. He analyzed a closely related Markovian network with partial servers, and used the results obtained to bound the performance of the hypercube scheme, without using approximations.

The results obtained in this chapter for the simple and the priority hypercube schemes are given in particularly interesting forms. The throughput of the simple scheme (with or without buffers) is given in parametric form as a function of the generation rate of new packets. The parametric solution involves a single parameter, and is very close to a closed form solution.

The throughput of the priority routing scheme is obtained from a (backward) recursion of only d steps, where d is the hypercube dimension. This is apparently the first such analysis of a priority scheme in hypercubes, with the exception of the analysis of a deflection priority scheme in [Var90]. Our results are approximate but as simulations indicate, very close to being accurate. We will devote a large portion of the chapter in examining the quality of the approximations.

In our schemes, packets follow shortest paths to their destinations. However, they may remain at some node without being transmitted even if there is a free link which they want to use (this phenomenon is called *idling*). The switch used at the nodes is simple, fast, and inexpensive, and uses $\Theta(d)$ wires as opposed to the $\Theta(d^2)$ wires of a crossbar switch. Packets can be dropped due to the unavailability of buffer space. In the simple scheme, packets competing for the same buffer space have equal priority, except for the new packets, which have the lowest priority. The priority scheme differs from the simple scheme in the way that contention over buffer space is resolved. In the priority scheme, packets which have been in the network longer have priority. The priority scheme has significantly larger throughput than the simple scheme, especially when the buffer space is small and the load is heavy.

Limited buffer space is a serious consideration when designing parallel computers. One expects massively parallel machines to have very little buffer space per node. This is because parallel and serial supercomputers must be built from the same number of components (the restrictions posed by the VLSI technology are the same for the two types of computers), which in parallel computers have to be divided among more processors. If we are using packet switching for the interprocessor communications then the only way to avoid packets being dropped is deflection routing (another way is the CSR protocol that we will propose in Chapter 6). The Connection Machine II, which with its 65,000 bit processors is the biggest parallel computer currently available, uses deflection routing and has space only for the packets being transmitted. In Section 5.6 we consider two deflection schemes, the simple and the priority deflection scheme, and evaluate their throughput by using simulations. The deflection schemes outperform the unbuffered priority scheme for hypercubes of large dimension. For uniform traffic, the throughput of the priority deflection scheme seems to tend to the maximum possible when the dimension of the hypercube increases. Deflection routing has, however, several disadvantages (see Section 6.5), and it requires the use of crossbar switches at the nodes.

In packet switching schemes, where the dropping of packets cannot be avoided, some feedback and retransmission protocol has to be superimposed on the routing scheme, at least to handle

the packets which are crucial for the execution of an algorithm. The retransmission strategy is not considered in this chapter, where we are only interested in the throughput and the probability that a packet is dropped, for a given buffer space, as a function of the offered traffic. The actual throughput may be smaller than what we will find, since we are neglecting the effect of retransmissions. The results of Subsections 5.3.2 and 5.4.2 will indicate that one or two buffers per link achieve throughput close to that of the infinite buffer case, even for high load. Increasing the buffer space further increases the throughput only marginally, and may be characterized as an ineffective use of the hardware resources.

The chapter is organized as follows. In Section 5.2 we describe the schemes, and the node model. In Sections 5.3 and 5.4 we evaluate the throughput of the simple and the priority scheme, respectively. The first subsection in each of these sections deals with the unbuffered case, while the second deals with the buffered case. In Section 5.3.3 we look at the asymptotic behavior of the throughput of the hypercube as the number of nodes increases, and the buffer space per link is fixed. In Section 5.5 we comment on the quality of the approximations used, and present simulation results. In Section 5.6 we define the simple and the priority deflection schemes, present simulation results on their throughput, and compare it with the throughput of the priority scheme of Section 5.4. In Section 5.7 we present an histogram of the packets with respect to their distance to the destination in any shortest path routing scheme. Section 5.7 can be read independently of the other sections because its analysis is of a more general nature.

5.2. DESCRIPTION OF THE SCHEMES

In this section we introduce the node model, and describe the simple and the priority routing schemes.

Each node has a queue for each of its outgoing links. The queue of node s that corresponds to link i is called the *i-th link queue*, and is denoted by $Q_i(s)$. A link queue is composed of two *buffers*, which can hold $k+1$ packets each. The first buffer is called the *forward buffer* and is denoted by $Q_i^1(s)$. The forward buffer can be used only by packets whose routing tags have the i^{th} bit equal to one, that is packets that have to cross the i^{th} dimension. The second buffer, denoted by $Q_i^0(s)$, is called *internal buffer*, and can be used only by packets that do not have

to cross the i^{th} dimension. The case $k = 0$ corresponds to the situation where a packet received during a slot is transnsmitted at the next slot (or it is dropped), and is called the *unbuffered* case.

We denote by $\oplus$ the bitwise exclusive OR operation between binary numbers. Then the neighbor of node s along the i^{th} dimension has binary representation $s \oplus e_i$. The queues of the nodes are linked in the following way: the internal buffer $\mathcal{Q}_i^0(s)$ is connected (internally) to queue $\mathcal{Q}_{(i-1) \bmod d}(s)$ of the same node, while the forward buffer $\mathcal{Q}_i^1(s)$ is connected to queue $\mathcal{Q}_{(i-1) \bmod d}(s \oplus e_i)$ of the neighbor node $s \oplus e_i$ (see Fig. 5.1). This router restricts the class of switching assignments that can be made. It is, however, much simpler, faster, and less expensive than a crossbar switch (see also relevant comments in Section 6.5).

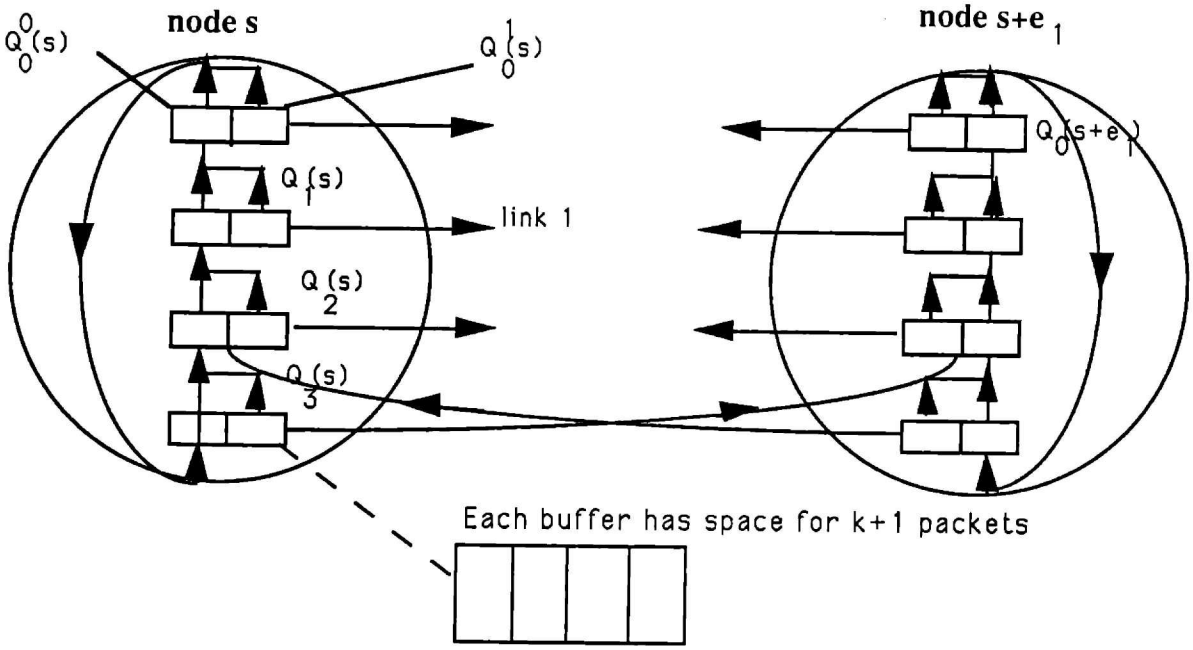

Figure 5.1: A node of the hypercube. For a comparison with other node (router) models, see Fig. 6.7 of Section 6.5.

In both the simple and the priority scheme, the packets traverse the hypercube dimensions in descending (modulo d) order, starting with a randomly chosen dimension. In particular, consider a packet that arrives at queue $\mathcal{Q}_i(s)$ (either from buffer $\mathcal{Q}^0_{(i-1) \bmod d}(s)$ of the same node, or from buffer $\mathcal{Q}^1_{(i-1) \bmod d}(s \oplus e_{i-1})$ of a neighbor node, or a new packet). Then the i^{th} bit of its routing tag is checked. Depending on whether this bit is a one or a zero, the packet claims buffer $\mathcal{Q}_i^1(s)$ in order to be transmitted during the next slot to queue $\mathcal{Q}_{(i-1) \bmod d}(s \oplus e_i)$

of the neighbor node $s \oplus e_i$, or it claims buffer $Q_i^0(s)$ in order to be internally passed to the next queue $Q_{(i-1) \bmod d}(s)$ of the same node. In both cases collisions can arise because more than one continuing and new packets may claim the same buffer of a link. When collisions occur packets are dropped if there is inadequate space at the buffer.

In the simple scheme conflicts over buffer space are resolved at random, with the exception of the newly generated packets which have the lowest priority. In the priority scheme the packets that have been in the system longer have priority when they compete for buffer space, and the packets that are dropped are those that have travelled less. If two packets have travelled an equal number of links then one of them is dropped with equal probability. Continuing packets have again absolute priority over new packets.

In the unbuffered case packets are removed from the network exactly d time units (slots) after entering the network. Since the bits of the routing tag of a packet are cyclically made equal to zero, packets are delivered to their destination with delay d, unless dropped on the way. A packet may arrive at its destination earlier, but it is not removed before the d^{th} slot; during the last slots it may travel from link-queue to link-queue of the same node until the time of its removal. However, in the buffered case, packets may wait in some buffers, thereby potentially increasing their total travel to more than d time units.

We denote by p_0 the probability that a new packet is generated during a slot and claims a buffer $Q_i^j(s)$, $j \in \{0,1\}$, $i \in \{1,2,\ldots,d\}$. At most one new packet can claim a link buffer during a slot. New packets denied acceptance to the network, or packets that are dropped are not retransmitted (*memoryless property*). One can visualize this model for the arrival process of new packets in the following way. Whenever a link is empty, a packet that wishes to use that link is requested, and with probability p_0 such a packet exists. If, for example $p_0 = 1/2$, and a link is empty 2/3 of the time, then the source inserts a packet 1/3 of the time. We will refer to p_0 as the *probability of access*. Other models for the arrival process can be incorporated in our model, as long as the new arrivals at each link are independent from the arrivals at other links, and p_0 can be calculated.

Note that for both schemes under investigation, the internal and the external links are mathematically equivalent. In order to see this, consider a packet located at the i^{th} link queue of a node s which does not have to cross dimension i, and is therefore passed to the $i-1$ mod d queue of the same node. This event happens with probability equal to that of the event where a packet has to cross dimension i and is sent to the $i-1$ mod d queue of node $s \oplus e_i$.

5.3. THE SIMPLE SCHEME

In this section we evaluate the throughput of the simple scheme. The unbuffered case is analyzed in Subsection 5.3.1, while the buffered case is analyzed in Subsection 5.3.2. As already mentioned, in the simple scheme all packets have equal priority, except for the new packets which have the lowest priority. The analysis to be given is approximate, but very close to being accurate as Section 5.5 will indicate. We will find a parametric expression that gives the throughput as a function of the probability of access p_0. The parametric solution involves a single parameter, and is almost as elegant as a closed form solution. We consider this important, since most of the results found in the literature for the steady state throughput of various networks and routing schemes were given in a less direct way (typically they were numerical or simulation results).

A packet will be referred to as a packet of *type* i when it has been transmitted (on forward or internal links) i times, including the current transmission. A packet received at a node with type different than d is a *continuing* packet. This definition does not include packets that have been received during previous slots, which are called *buffered* packets. Packets generated at a node are called *new* packets.

5.3.1 Analysis of the Simple Scheme Without Buffers

In this subsection we consider the unbuffered case where each link buffer can hold at most one packet, the one being transmitted (internally or to a neighbor node).

We denote by p_i, $i = 1, 2, \ldots, d$, the probability that an i-type packet is transmitted on a particular link at time t. We also denote by e the probability that no packet is transmitted over a link at time t. In steady state the p_i's and e are independent of t, and by symmetry they are independent of the link. Clearly, we have

$$e + \sum_{i=1}^{d} p_i = 1. \tag{5.1}$$

Consider a particular link l (for example, the one connecting $Q_i^1(s)$ with $Q_{(i-1) \bmod d}(s + e_i)$). Call l_1 and l_2 the internal and the forward link, respectively, that lead to l (see Fig. 5.2). We make the following approximating assumption.

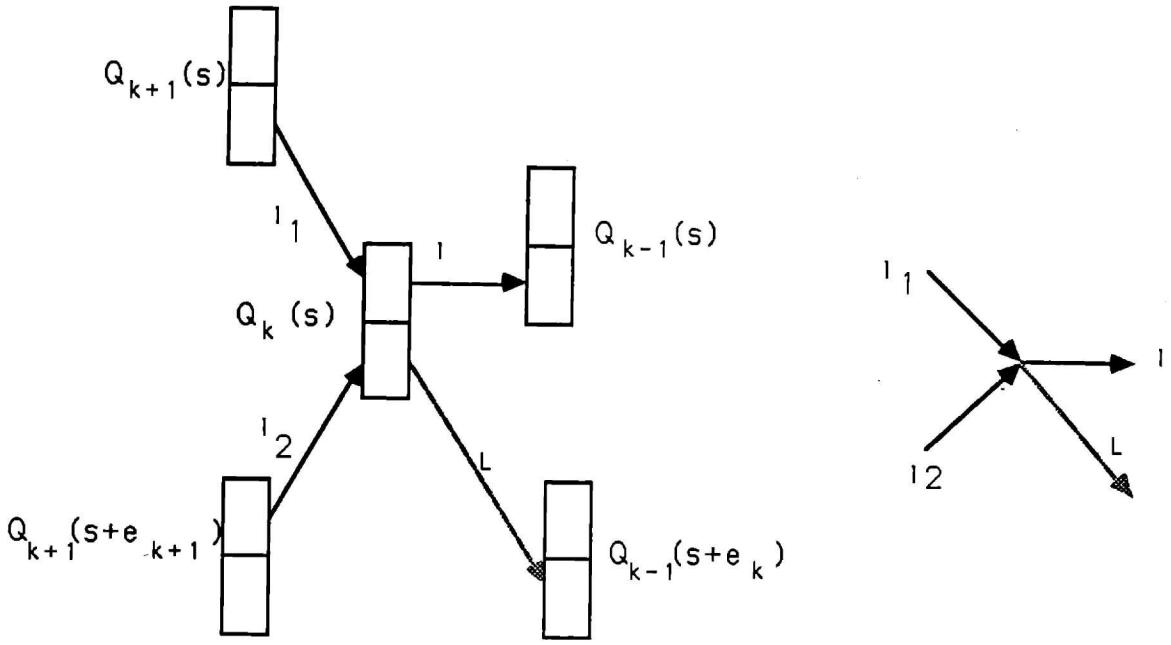

Figure 5.2: A link l, and the two links leading to it.

Approximating Assumption A.5.3.1: Events in l_1 are independent of events in l_2 during the same slot.

In our schemes, a packet transmitted on l_1 and a packet transmitted on l_2 have used in the past links belonging to different subcubes of the hypercube. However, events in l_1 are not independent of events in l_2, as will be explained in Section 5.5. Nonetheless, we believe that the approximating assumption A.5.3.1 is a very good approximation, as will be explained in Section 5.5 and supported by simulation results.

Let E_j, $j = 1, 2$, be the event that a packet $\mathcal{P}$ of type $i-1$ arrives on link l_j, requests link l, and gets it. Then, for $i = 2, 3, \ldots, d$, the probability p_i that a packet of type i is transmitted on link l at some slot is

$$p_i = \Pr(E_1) + \Pr(E_2) = 2\Pr(E_1).$$

The probability that a packet $\mathcal{P}$ of type $i-1$ arrives on l_1 is equal to p_{i-1}, and the probability that this packet will request link l is $1/2$. Thus,

$$p_i = p_{i-1}\Pr(\mathcal{P} \text{ not dropped}). \qquad i = 2, 3, \ldots, d.$$

Under the approximating assumption A.5.3.1 the probability that a packet of type different

than d arrives on link l_2 and claims link l is equal to

$$\frac{\sum_{j=1}^{d-1} p_j}{2}.$$

Since collisions are resolved at random, such a packet will cause packet $\mathcal{P}$ to be dropped with probability 1/2. Thus

$$p_i = p_{i-1}\left(1 - \frac{\sum_{j=1}^{d-1} p_j}{4}\right), \qquad i = 2, 3, \ldots, d. \tag{5.2}$$

Note that p_d does not appear in the summation above because packets of type d are removed and do not claim a link.

The probability p_1 that a packet of type 1 is transmitted on link l is the product of two probabilities:

(1) The probability that no packet arrives on l_1 or l_2 requesting link l; this happens with probability

$$\left(1 - \frac{\sum_{j=1}^{d-1} p_j}{2}\right)^2.$$

(2) The probability that a new packet is generated at l; this happens with probability p_0.

Therefore, p_1 is given by

$$p_1 = p_0\left(1 - \frac{\sum_{j=1}^{d-1} p_j}{2}\right)^2. \tag{5.3}$$

Similarly, the probability e that a link is empty is equal to

$$e = (1 - p_0)\left(1 - \frac{\sum_{j=1}^{d-1} p_j}{2}\right)^2. \tag{5.4}$$

Define

$$\theta = p_d + e = 1 - \sum_{j=1}^{d-1} p_j, \tag{5.5}$$

where θ will be treated as a free parameter. Then Eqs. (5.2)-(5.5) give

$$p_d = \frac{p_0}{4^d}(1+\theta)^2(3+\theta)^{d-1} \tag{5.6}$$

and

$$e = \frac{1}{4}(1 - p_0)(1+\theta)^2, \tag{5.7}$$

Adding Eqs. (5.6) and (5.7), and taking into account that $\theta = p_d + e$ we get

$$\theta = p_0 \frac{1}{4^d}(1+\theta)^2(3+\theta)^{d-1} + \frac{1}{4}(1-p_0)(1+\theta)^2,$$

which gives

$$p_0 = \frac{4\theta - (1+\theta)^2}{\frac{1}{4^{d-1}}(1+\theta)^2(3+\theta)^{d-1} - (1+\theta)^2}. \tag{5.8}$$

Equations (5.6) and (5.8) give the relationship between p_d and p_0 in parametric form.

Since there are $2d$ (internal or forward) links per node, the throughput R per node is

$$R = 2dp_d.$$

An upper bound on R is two packets per node; this can be proved by using arguments similar to those of Subsection 5.6.2.

Note that the feasible values of the parameter θ range continuously from 1 (when $p_0 = 0$, which gives $e = 1$, $p_d = 0$) to a small number (when $p_0 = 1$, which gives $e = 0$ and p_d less than $1/d$). By giving values to θ we can find the corresponding values of p_0 and R. The fact that the range of θ is not the entire interval [0,1] does not create any problem, since the values of θ which are not feasible give $p_0 > 1$.

Figure 5.3 illustrates the results obtained for $d = 11$, by giving several values to θ and finding the corresponding values of p_0, and R. To evaluate the results it is useful to have in mind typical values of the traffic load that appears in real systems. Measurements reported in [HsB90] for numerical and simulation algorithms have given that in almost all cases links were idle for more than 95% of the time. In our model such loads correspond to values of p_0 much less than 0.05[1].

5.3.2. Analysis of the Simple Scheme With Buffers

In this subsection we analyze the buffered version of the simple scheme. In particular, we assume that each link buffer can hold up to k packets, in addition to the one being transmitted.

[1] This value is probably an overestimation. If the measurements reported in [HsB90] corresponded to uniform traffic, then the corresponding p_0 would be less than $0.05/d$; however, the communication patterns in the measurements of [HsB90] may have had locality properties. Note also that even with infinite buffers the case $2dp_0 > 2$ would correspond to an unstable system, since 2 is the maximum throughput that can be sustained.

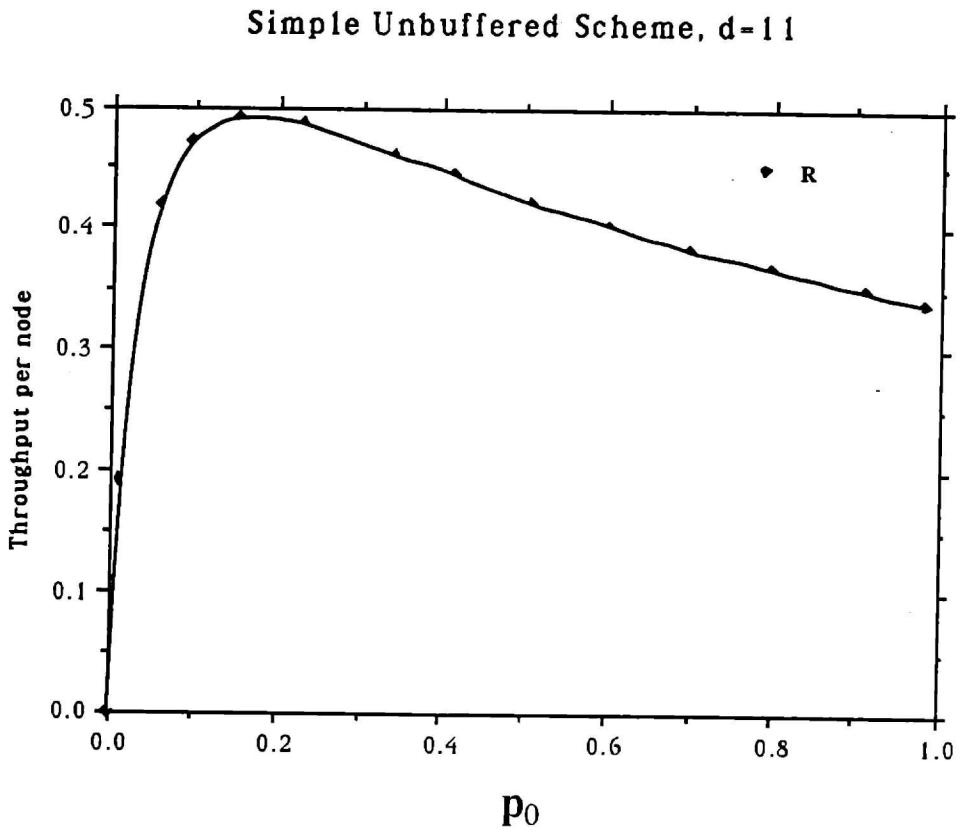

Figure 5.3: Simple scheme without buffers (d=11).

The scheme is the same with that analyzed in Subsection 5.3.1 with the difference that when packets collide one of them is transmitted and the other is stored, if there is enough space in the buffer, or dropped, otherwise. Therefore, the analysis of Subsection 5.3.1 corresponds to the special case $k = 0$ of this subsection. Continuing packets have priority over buffered packets or new packets when claiming a link. New packets are admitted in the network only if the buffer where they enter is completely empty. A new packet is available at a link buffer with probability p_0 during a slot.

We will find a relationship between the throughput p_d per link and the probability of new arrivals p_0. The relationship will be given in parametric form, involving a single parameter. Note that corresponding results in the literature are typically obtained through the use of numerical methods and/or simulations ([DiJ81], [GrG86], [Dal90a], [Max89], [Var90], [Bra91]), or they are of an asymptotic nature in the number of processors or in the buffer size ([GrH90], [Sta91]).

In order to analyze the buffered version of the simple scheme, we need in addition to the approximating assumption A.5.3.1, the following approximating assumption.

Approximating Assumption A.5.3.2: The arrivals of packets at a buffer during a slot are independent of the arrivals at the buffer during previous slots.

Independence approximating assumptions are found in most throughput analyses of buffered routing schemes of direct or indirect multiprocessor systems.

We denote by b_i, $i = 0, 1, \ldots, k$, the probability that there are i packets at a buffer at the beginning of a slot. The remainder of the notation used in this subsection is the same with that used in Subsection 5.3.1. Since there are two links (one forward and one internal) leading to a buffer, at most two continuing packets may arrive at a buffer during a slot. Thus, for $i \neq 0, k$ we have

$$b_i = b_i \Pr(\text{one arrival only}) + b_{i-1} \Pr(\text{two arrivals}) + b_{i+1} \Pr(\text{no arrivals}). \tag{5.9}$$

In getting Eq. (5.9) we have used the approximating assumption A.5.3.2 in the following way: we have assumed that the events of one, two, or no arrivals at a buffer are independent of the arrival process at previous slots, and therefore of the number of packets found in the buffer. Calculating the probability of one, two or no arrivals of continuing packets at a link buffer, and substituting in the preceding equation we obtain

$$b_i = 2b_i \left(1 - \frac{\sum_{j=1}^{d-1} p_j}{2}\right) \frac{\sum_{j=1}^{d-1} p_j}{2} + b_{i-1} \left(\frac{\sum_{j=1}^{d-1} p_j}{2}\right)^2 + b_{i+1} \left(1 - \frac{\sum_{j=1}^{d-1} p_j}{2}\right)^2, \quad i = 1, \ldots, k-1. \tag{5.10}$$

For b_0 and b_k the equations are slightly different:

$$b_0 = 2b_0 \left(1 - \frac{\sum_{j=1}^{d-1} p_j}{2}\right) \frac{\sum_{j=1}^{d-1} p_j}{2} + (b_0 + b_1) \left(1 - \frac{\sum_{j=1}^{d-1} p_j}{2}\right)^2, \tag{5.11}$$

and

$$b_k = 2b_k \left(1 - \frac{\sum_{j=1}^{d-1} p_j}{2}\right) \frac{\sum_{j=1}^{d-1} p_j}{2} + (b_k + b_{k-1}) \left(\frac{\sum_{j=1}^{d-1} p_j}{2}\right)^2. \tag{5.12}$$

Letting

$$\theta = p_d + \epsilon = 1 - \sum_{i=1}^{d-1} p_i,$$

Eqs. (5.10)-(5.12) can be rewritten as

$$b_i = 2\frac{1+\theta}{2}\frac{1-\theta}{2} b_i + b_{i-1} \left(\frac{1-\theta}{2}\right)^2 + b_{i+1} \left(\frac{1+\theta}{2}\right)^2, \quad i = 1, 2, \ldots, k-1, \tag{5.13}$$

$$b_0 = 2b_0\frac{1+\theta}{2}\frac{1-\theta}{2} + (b_0 + b_1)\left(\frac{1+\theta}{2}\right)^2, \tag{5.14}$$

and

$$b_k = 2b_k\frac{1+\theta}{2}\frac{1-\theta}{2} + (b_k + b_{k-1})\left(\frac{1-\theta}{2}\right)^2. \tag{5.15}$$

The quadratic equation corresponding to the second order recursion (5.13) is

$$\rho^2\left(\frac{1+\theta}{2}\right)^2 + \rho\left(\frac{1-\theta^2}{2} - 1\right) + \left(\frac{1-\theta}{2}\right)^2 = 0,$$

which has roots

$$\rho_1 = 1$$

and

$$\rho_2 = \left(\frac{1-\theta}{1+\theta}\right)^2.$$

Thus the solutions to the recursion are of the form

$$b_i = \alpha + \beta\left(\frac{1-\theta}{1+\theta}\right)^{2i}, \qquad i = 0, 1, \ldots k-1,$$

for some constants α and β. Taking into account Eqs. (5.14) and (5.15) we obtain after some algebraic manipulation that

$$b_i = b_0\left(\frac{1-\theta}{1+\theta}\right)^{2i}, \qquad i = 0, 1, \ldots, k. \tag{5.16}$$

To verify this equation, use it to express b_i in terms of b_0 in Eqs. (5.13)-(5.15), and see that these equations hold identically for all θ. Since

$$\sum_{i=0}^{k} b_i = 1$$

we finally get that

$$b_0 = \frac{1 - \left(\frac{1-\theta}{1+\theta}\right)^2}{1 - \left(\frac{1-\theta}{1+\theta}\right)^{2k+2}}, \tag{5.17}$$

for $\theta \in (0, 1)$.

For $k = 0$ (unbuffered case) we have $b_0 = 1$. For infinite buffer space and $\theta \neq 1$ we have

$$b_0 = 1 - \left(\frac{1-\theta}{1+\theta}\right)^2, \quad \text{for } k = \infty. \tag{5.18}$$

The probability that a buffered packet is of type $i-1$, for $2 \leq i \leq d$, is proportional to p_{i-1}, because all the packets have the same priority during collisions. Therefore,

$$\Pr(\text{buffered packet is of type } i-1) = \frac{p_{i-1}}{\sum_{i=1}^{d-1} p_i}.$$

A packet of type $i = 2, 3, \ldots, d$ transmitted over a link is either a continuing packet or, if such a packet does not exist, it is a packet that was buffered. The probability that a continuing packet is transmitted over a link is given by Eq. (5.2). The probability that a buffered packet is transmitted as an i-type packet is the product of three probabilities: (i) the probability that no continuing packet requested the link, (ii) the probability that the buffer was non-empty, and (iii) the probability that the packet at the head of the buffer was of type $i-1$. Thus,

$$\begin{aligned} p_i &= p_{i-1}\left(1 - \frac{\sum_{j=1}^{d-1} p_j}{4}\right) + \left(1 - \frac{\sum_{j=1}^{d-1} p_j}{2}\right)^2 (1-b_0)\frac{p_{i-1}}{\sum_{i=1}^{d-1} p_i} \\ &= p_{i-1}\left(\frac{\theta+3}{4}\right) + \left(\frac{\theta+1}{2}\right)^2 (1-b_0)\frac{p_{i-1}}{1-\theta}, \qquad i > 1 \end{aligned} \tag{5.19}$$

where the first term accounts for packets which are received and transmitted at the immediately following slot, and the second term accounts for packets which were buffered. Since new packets are accepted in the network only when there are no continuing or buffered packets, we have

$$p_1 = p_0 b_0 \left(1 - \frac{\sum_{j=1}^{d-1} p_j}{2}\right)^2 = p_0 b_0 \left(\frac{1+\theta}{2}\right)^2. \tag{5.20}$$

A link is empty if there are no continuing, buffered or new packets that want to use it. Thus,

$$e = (1-p_0) b_0 \left(1 - \frac{\sum_{j=1}^{d-1} p_j}{2}\right)^2 = (1-p_0) b_0 \left(\frac{1+\theta}{2}\right)^2.$$

Equations (5.19) and (5.20) give

$$p_d = \frac{p_0 b_0 (\theta+1)^2}{4^d}\left(3 + \theta + (1-b_0)\frac{(1+\theta)^2}{1-\theta}\right)^{d-1}. \tag{5.21}$$

Adding the last two equations, substituting $\theta = p_d + e$ and solving for p_0 we obtain

$$p_0 = \frac{b_0(1+\theta)^2 - 4\theta}{b_0(1+\theta)^2 - \frac{b_0(\theta+1)^2}{4^{d-1}}\left(3 + \theta + (1-b_0)\frac{(1+\theta)^2}{1-\theta}\right)^{d-1}}, \tag{5.22}$$

where b_0 is given by Eq. (5.17). Equations (5.21) and (5.22) give the relationship between p_d and p_0 in parametric form with parameter θ. The throughput per node is again

$$R = 2dp_d.$$

In the case of infinite buffer space, b_0 is given by Eq. (5.18), and Eq. (5.21) is simplified to

$$p_d = p_0\theta, \quad \text{for } k = \infty.$$

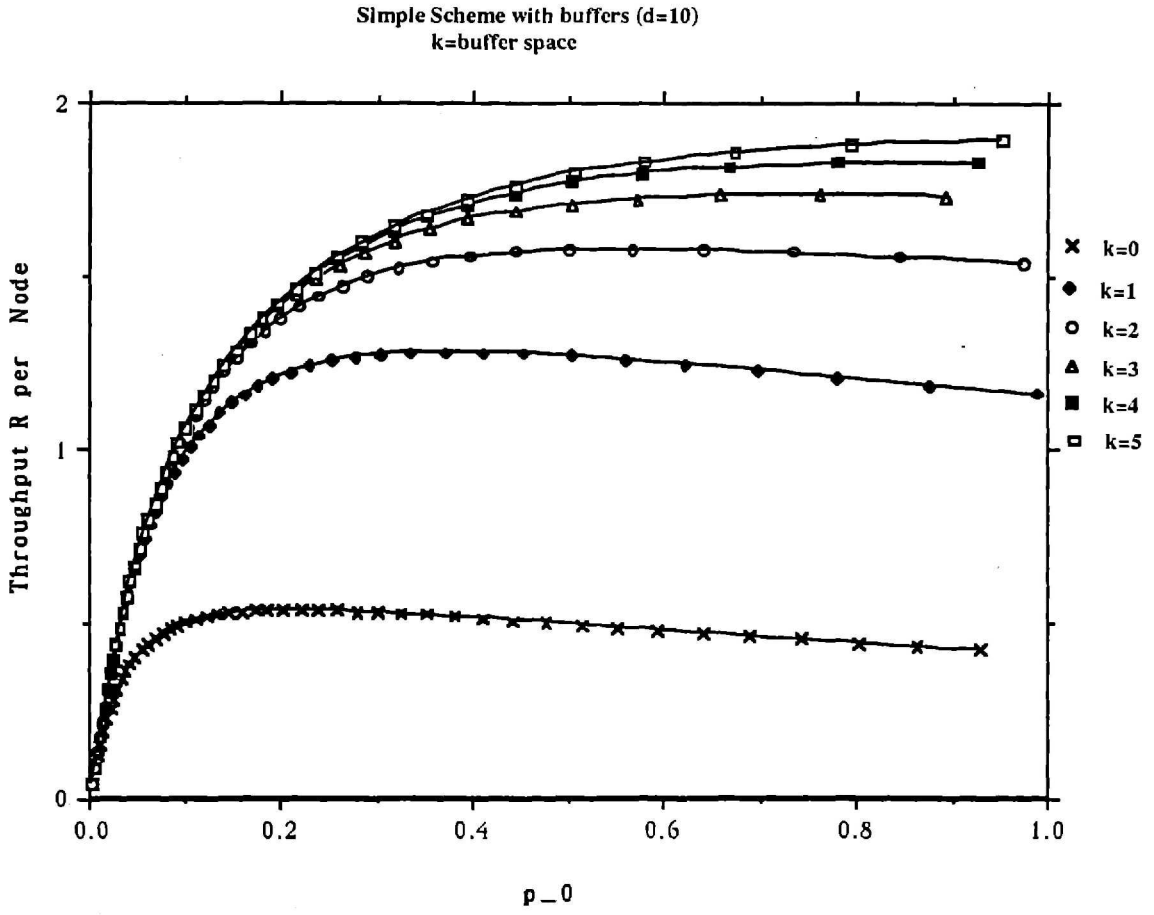

Figure 5.4: Throughput per node of the simple scheme for various buffer sizes.

As $k \rightarrow \infty$, Eq. (5.22) takes the indeterminate form 0/0. By using L' Hospital's rule we get after some calculations that

$$p_0 = \frac{1-\theta}{\theta(d-1)}, \text{ for } k = \infty.$$

Combining the last two equations and using the fact $R = 2dp_d$ we obtain

$$R = \frac{2dp_0}{1+p_0(d-1)}, \quad \text{for } k = \infty.$$

For $p_0 = 1$ and infinite buffer space, the throughput R is equal to two packets per node as expected.

In Fig. 5.4 we have plotted the throughput R per node as a function of p_0 for several buffer sizes k. It can be seen from this figure that a buffer space of two or three packets per link is adequate to achieve throughput close to the maximum.

5.3.3. Asymptotic Behavior of the Thoughput

In this subsection we will examine the asymptotic behavior of the throughput of the hypercube for a particular value of the buffer size k, as the number of nodes increases.

Combining Eqs. (5.20), (5.21), and (5.17), we obtain after some calculations that

$$\frac{p_d}{p_1} = \left(\frac{3 + \theta + \frac{(1+\theta)^2}{1-\theta} \cdot \frac{\left(\frac{1-\theta}{1+\theta}\right)^2 - \left(\frac{1-\theta}{1+\theta}\right)^{2k+2}}{1-\left(\frac{1-\theta}{1+\theta}\right)^{2k+2}}}{4} \right)^{d-1} . \tag{5.23}$$

Define

$$y = \frac{1-\theta}{1+\theta}. \tag{5.24}$$

Then Eq. (5.23) can be rewritten after some calculations as

$$\frac{p_d}{p_1} = \left(1 - \frac{y^{2k+1} - y^{2k+3}}{2(1+y)(1-y^{2k+2})}\right)^{d-1} . \tag{5.25}$$

Since the free parameter θ ranges continuously from some small number to one, y can take all the values from zero to some number close to one. We choose

$$y = d^{-\frac{1}{2k+1}}. \tag{5.26}$$

This value of y corresponds to a fixed value of p_0; therefore, the corresponding throughput is a lower bound on the maximum possible throughput. As $d \to \infty$, the right hand side of Eq. (5.23) tends to a positive constant $c = e^{-2}$, that is

$$\lim_{d\to\infty} \frac{p_d}{p_1} = c > 0, \qquad \text{for } y \text{ chosen according to Eq. (5.26).} \tag{5.27}$$

For y given by Eq. (5.26) we get $1 - \theta = \sum_{i=1}^{d-1} p_i = \Theta\left(d^{-\frac{1}{2k+1}}\right)$. Since

$$\frac{p_d}{p_1} \le \frac{p_i}{p_1} \le 1, \qquad \text{for } i = 1, 2, \ldots, d-1, \tag{5.28}$$

we get in view of Eq. (5.27) that for y given by Eq. (5.26) we have $p_i = \Theta(p_d)$, for $i = 1, 2, \ldots, d$, and

$$p_d = \Theta\left(\frac{1}{d^{1+\frac{1}{2k+1}}}\right).$$

Therefore the maximum total throughput H of the hypercube is

$$H = 2dNp_d = \Omega\left(\frac{N}{(\log N)^{\frac{1}{2k+1}}}\right). \tag{5.29}$$

In the case that $y = o(d^{-\frac{1}{2k+1}})$ we can similarly find that $p_d = o\left(d^{-1-\frac{1}{2k+1}}\right)$, and $H = o\left(N/(\log N)^{\frac{1}{2k+1}}\right)$. In the case that $y = \tilde{\Omega}(d^{-\frac{1}{2k+1}})$, where $\tilde{\Omega}$ stands for strictly larger order of magnitude, Eq. (5.25) gives $p_d/p_1 \to 0$ as $d \to \infty$. These observations together with Eq. (5.29) give

$$H = \Theta\left(\frac{N}{(\log N)^{\frac{1}{2k+1}}}\right). \tag{5.30}$$

For $y = o(d^{-\frac{1}{2k+1}})$ (or else, $p_0 = o\left(1/d^{1+\frac{1}{2k+1}}\right)$, i.e. small load), we expect almost all of the packets to succesfully reach their destination, while for $y = \Theta(d^{-\frac{1}{2k+1}})$ (or else, $p_0 = o\left(1/d^{1+\frac{1}{2k+1}}\right)$) we expect a constant fraction of the packets to reach their destination. For $y = \tilde{\Omega}(d^{-\frac{1}{2k+1}})$ (or else, $p_0 = \tilde{\Omega}\left(1/d^{1+\frac{1}{2k+1}}\right)$, almost all of the packets are dropped (for large d).

The dependence of the total throughput of the hypercube on the buffer size k per link is a very interesting one. Having buffer space for $k = 1$ packet (in addition to the one being transmitted) increases the throughput significantly. Increasing k further gives diminishing returns. For $k = \infty$, Eq. (5.27) gives $H = \Theta(N)$ as expected (in fact then we have $H = 2N$). For the unbuffered case we have $H = \Theta(N/\log N)$.

It is useful to compare Eq. (5.30) with the throughput of another routing scheme, which we will call *greedy idling scheme*, in a *Q-dilated hypercube*. A Q-dilated hypercube is a hypercube whose links have capacity Q, that is, each link can be used for up to Q packets. In the greedy idling scheme, packets traverse the hypercube dimensions in descending order, starting *always* with dimension d. This scheme has been analyzed by Koch in [Koc88] and [Koc89] (the case where the capacity is equal to one was previously analyzed in [Pat81] and [KrS83]). The maximum total throughput of the greedy idling scheme in a Q-dilated hypercube is given by

$$H = \Theta\left(\frac{N}{(\log N)^{\frac{1}{Q}}}\right).$$

From the preceding equation and Eq. (5.30), it seems (modulo our approximating assumptions, since Koch's result is rigorously obtained) that increasing the buffer space by a constant factor

impoves the throughput significantly more than increasing the capacity of the links by the same factor. This means that less packets are dropped when we have k buffer spaces per link than when we have k wires per link. This holds on the average, since we can find both scenaria where buffer space helps most, and scenaria where capacity helps most ([Lei92b]).

5.4. THE PRIORITY SCHEME

In this section we will evaluate the throughput of the priority scheme. Recall that in the priority scheme the packets that have been in the system longer have priority when they compete for a forward or an internal link. If two packets have travelled the same distance then one of them is transmitted with equal probability. New packets are admitted at a link buffer only when no continuing packet claims this buffer. We analyze the unbuffered priority scheme in Subsection 5.4.1, and the buffered priority scheme in Subsection 5.4.2.

5.4.1. Analysis of the Priority Scheme Without Buffers

In this subsection we analyze the unbuffered case. Since packets which have travelled more have priority when claiming the same link, the packets of type i are not affected by the existence of packets of type $1, 2, \ldots, i-1$ or new packets. Let p_i and ϵ be as defined in Subsection 5.3.1. Consider a link l, and the two links l_1 and l_2 leading to it. Making again the approximating assumption A.5.3.1 and reasoning as in Subsection 5.3.1 we find that

$$p_i = p_{i-1} \Pr(\text{packet of type } i-1 \text{ not dropped}). \tag{5.31}$$

A packet $\mathcal{P}$ of type $i-1$ that arrives on link l_1 is dropped if and only if one of the following two events happens:

Event 1: There was a packet of type $i, i+1, \ldots, d-1$ transmitted on link l_2 during the previous slot and this packet had the corresponding bit of the routing tag such that l was chosen. This can happen with probability

$$\frac{1}{2} \sum_{j=i}^{d-1} p_j. \tag{5.32}$$

or

Event 2: A packet of type $i-1$ was transmitted on link l_2 during the previous slot, the i^{th} bit of its routing tag was such that l was chosen, and that packet was chosen (with probability 0.5) instead of $\mathcal{P}$. The probability of this event is

$$\frac{p_{i-1}}{4}. \tag{5.33}$$

Since Events 1 and 2 are mutually exclusive, Eqs. (5.31)-(5.33) give

$$p_i = p_{i-1}\left(1 - \frac{1}{2}\sum_{j=i}^{d-1} p_j - \frac{p_{i-1}}{4}\right), \qquad i = 2, 3, \ldots, d. \tag{5.34}$$

For p_1 we have a slightly different equation:

$$p_1 = p_0\left(1 - \frac{\sum_{j=1}^{d-1} p_j}{2}\right)^2, \tag{5.35}$$

where p_0 is the probability that no new packet is available and

$$\left(1 - \frac{\sum_{j=1}^{d-1} p_j}{2}\right)^2$$

is the probability that no continuing packet that wants to use link l arrives on l_1 or l_2. The probability that a link is empty can be seen to be

$$e = (1 - p_0)\left(1 - \frac{\sum_{j=1}^{d-1} p_j}{2}\right)^2. \tag{5.36}$$

For a particulal value of p_d we can use Eqs. (5.34)-(5.36) to find the corresponding unique values of $p_{d-1}, \ldots, p_0$. This can be done by viewing Eq. (5.34) as a backward instead of a forward recursion, solving with respect to p_{i-1}, and keeping the solution that gives a legitimate probability distribution (the other solution of the quadratic equation (5.34) gives $p_{i-1} > 1$) we obtain

$$p_{i-1} = 2 - \sum_{j=i}^{d-1} p_j - \sqrt{\left(2 - \sum_{j=1}^{d-1} p_j\right)^2 - 4p_i}, \quad i = 0, \ldots, d-2. \tag{5.37}$$

For p_0 we have from Eq. (5.35) that

$$p_0 = \frac{p_1}{\left(1 - \frac{\sum_{j=1}^{d-1} p_j}{2}\right)^2}. \tag{5.38}$$

Giving a value to p_d we can obtain the corresponding value of p_0 by solving the backward recursion. The remaining performance parameters of interest (for example, the probability e

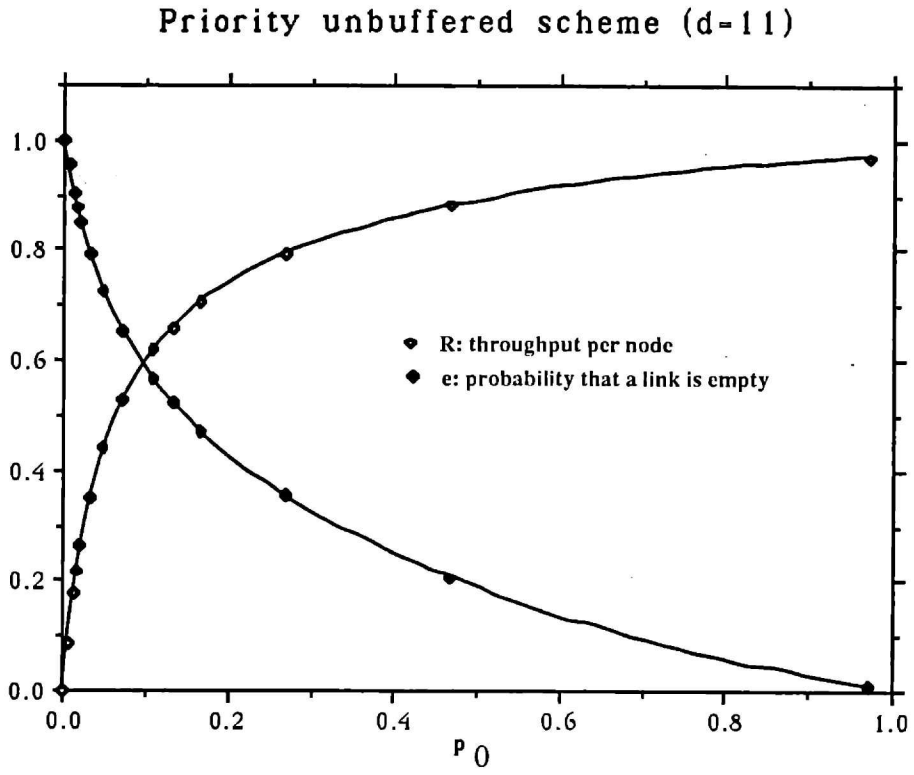

Figure 5.5: Priority scheme without buffers (d=11).

that a link is empty, and the mean throughput per node $R = 2dp_d$) can then be computed easily. Figure 5.5 illustrates the results obtained for $d = 11$.

Let $p_d = F(p_0)$, where the function F is not known in closed form. We already presented a simple recursion to compute $p_0 = F^{-1}(p_d)$ for each p_d. It can be proved by induction that F^{-1}, and therefore F, is monotonically increasing. This shows that F is 1-1 (this is also evident from the fact that Eq. (5.34) has a unique solution in the interval [0,1]), and the maximum of p_d and R occurs for $p_0 = 1$. This is a desirable characteristic of the priority scheme. It indicates that if we superimpose on it a retransmission scheme, then the system will behave well when congestion arises. Recall that for the simple scheme the relationship between p_d and p_0 is not 1-1 (at least for the unbuffered case), and the maximum throughput is attained for p_0 much less than one.

5.4.2. Analysis of the Priority Scheme with Buffers

In this subsection we evaluate the throughput of the priority scheme when there is buffer space at each link. In particular, we assume that each buffer can hold up to k packets in addition to the one being transmitted. When two packets arrive at a node and request the same link, one of them is transmitted over the link, and the other is either stored (if there is enough space in the buffer), or dropped. The packet which is transmitted is the one that has crossed more forward and internal links; if the two packets have travelled the same distance one of them is selected at random. The analysis of Subsection 5.4.1 corresponds to the case $k = 0$ of this subsection. Continuing packets have priority over buffered packets or new packets when claiming a link. New packets are admitted in the network only if the buffer where they enter is completely empty. The buffers are FIFO, and packets in the buffer that have higher priority do *not* overtake packets of lower priority that are in front of them. If we were using a priority discipline within the buffer then we could probably obtain higher throughput, but the system would be more difficult to analyze.

In order to analyze the buffered priority scheme we make again the approximating assumptions A.5.3.1 and A.5.3.2.

Following the notation of Subsection 5.3.2, we denote by b_i, $i = 0, 1, \ldots, k$, the probability that there are i buffered packets just before the beginning of a slot, and we define the parameter $\theta = p_d + e$. The probability b_i is given again by Eqs. (5.16) and (5.17), which we repeat here for completeness:

$$b_i = b_0 \left(\frac{1-\theta}{1+\theta}\right)^{2i}, i = 0, 1, \ldots, k, \tag{5.39}$$

and

$$b_0 = \frac{1 - \left(\frac{1-\theta}{1+\theta}\right)^2}{1 - \left(\frac{1-\theta}{1+\theta}\right)^{2k+2}}, \tag{5.40}$$

with $\theta \in (0, 1)$.

The probability that a buffered packet is of type $i-1$, for $2 \leq i \leq d$, is proportional to

$$p_{i-1}\left(\frac{p_{i-1}}{2} + \sum_{j=i}^{d-1} p_j\right),$$

since only conflicts with packets of types $i-1, i, \ldots, d$ can put such a packet in the buffer. Therefore, a buffered packet is of type $i-1$ with probability

$$\frac{p_{i-1}\left(\frac{p_{i-1}}{2} + \sum_{j=i}^{d-1} p_j\right)}{\sum_{l=1}^{d-1} p_l \left(\frac{p_l}{2} + \sum_{j=l+1}^{d-1} p_j\right)} = \frac{p_{i-1}\left(\frac{p_{i-1}}{2} + \sum_{j=i}^{d-1} p_j\right)}{0.5 \cdot \left(\sum_{l=1}^{d-1} p_l\right)^2} = \frac{p_{i-1}\left(\frac{p_{i-1}}{2} + \sum_{j=i}^{d-1} p_j\right)}{0.5 \cdot (1-\theta)^2}.$$

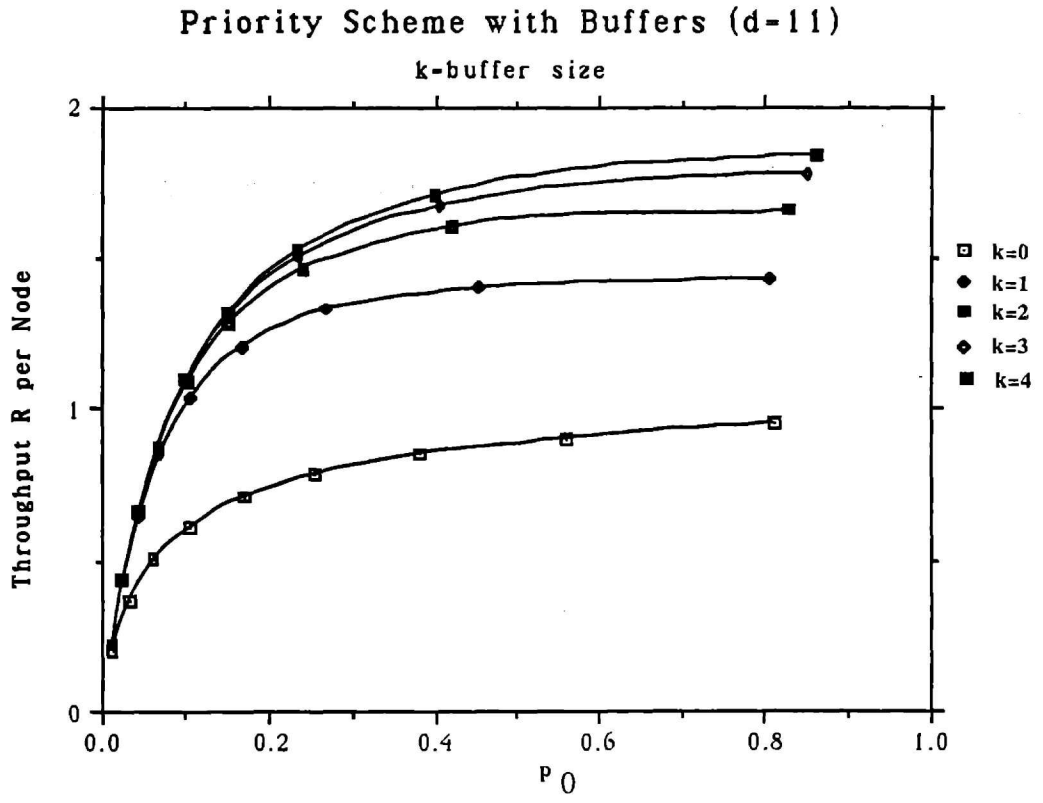

Figure 5.6: The throughput of the priority scheme for various buffer sizes.

The probability that a packet of type $i > 1$ is transmitted over a link is

$$p_i = p_{i-1}\left(1 - \frac{\sum_{j=i}^{d-1} p_j}{2} - \frac{p_{i-1}}{4}\right) + \left(1 - \frac{\sum_{j=1}^{d-1} p_j}{2}\right)^2 (1-b_0)\frac{p_{i-1}\left(\frac{p_{i-1}}{2} + \sum_{j=i}^{d-1} p_j\right)}{0.5\cdot(1-\theta)^2}$$

$$= p_{i-1}\left(1 - \frac{\sum_{j=i}^{d-1} p_j}{2} - \frac{p_{i-1}}{4}\right) + \frac{(1+\theta)^2}{2(1-\theta)^2}(1-b_0)p_{i-1}\left(\frac{p_{i-1}}{2} + \sum_{j=i}^{d-1} p_j\right), \qquad i > 1 \tag{5.41}$$

where the first term is the same with the right side of Eq. (5.34) and accounts for packets that were received during the preceding slot, and the second term accounts for packets that were buffered. Since new packets are accepted in the network only when there are no continuing or buffered packets, we get

$$p_1 = p_0 b_0 \left(1 - \frac{\sum_{j=1}^{d-1} p_j}{2}\right)^2 = p_0 b_0 \left(\frac{1+\theta}{2}\right)^2. \tag{5.42}$$

The probability that a link is empty is equal to

$$e = (1-p_0)b_0 \left(1 - \frac{\sum_{j=1}^{d-1} p_j}{2}\right)^2 = (1-p_0)b_0 \left(\frac{1+\theta}{2}\right)^2.$$

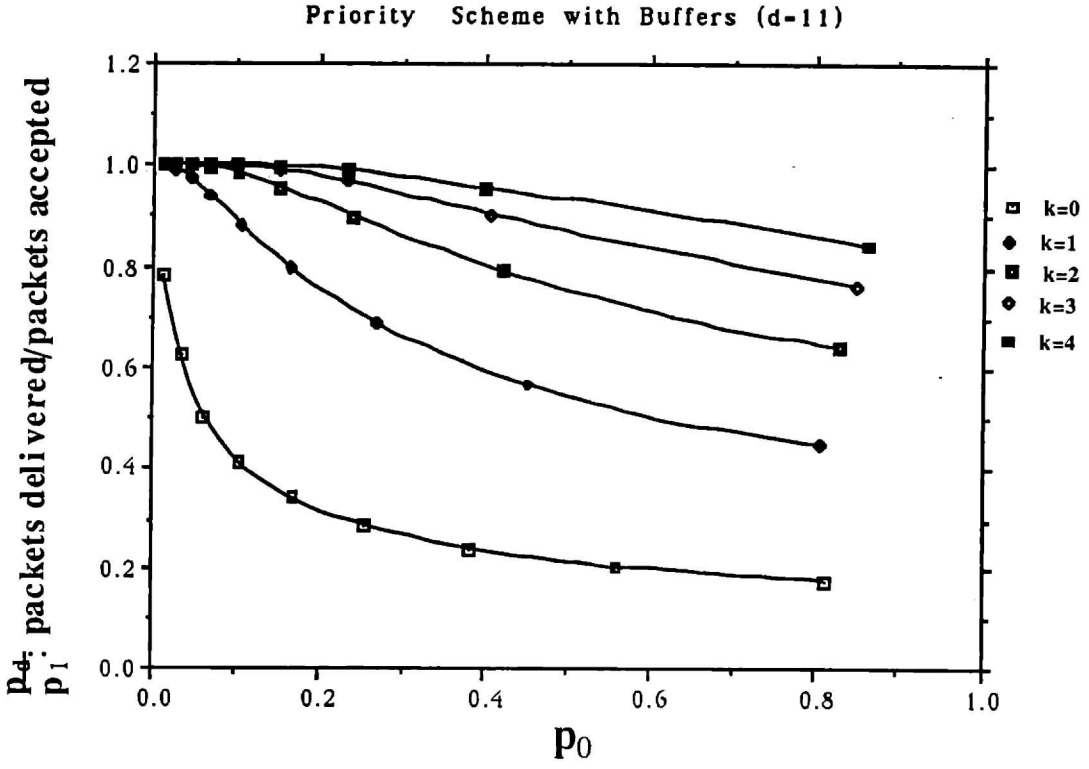

Figure 5.7: Packets delivered to their destination as a fraction of those accepted in the network for the priority scheme and various buffer sizes.

We used a Gauss-Seidel type of algorithm to find numerically p_i's that satisfy Eqs. (5.41) and (5.42). We did not prove the uniqueness of these solutions, but the results obtained for different initial conditions were the same. The results obtained are shown in Figs. 5.6 and 5.7. In Fig. 5.6 we have plottted the throughput $R = 2dp_d$ per node as a function of p_0 for several buffer sizes k. Figure 5.7 illustrates the ratio p_d/p_1, that is, the fraction of admitted packets that arrive at their destination, for several values of the buffer size k. As is evident from these figures, a buffer of size one or two is adequate in practice to achieve good throughput, and low probability of a packet being dropped. For $p_0 < 0.05$, which is the load region where we expect systems to operate (see footnote in Subsection 5.3.2), having no buffers does not degrade the performance significantly.

Comparing Fig. 5.4 with Fig. 5.6 we see that the priority rule increases the throughput significantly. The priority rule is designed to decrease the waste of resources caused from packets being transmitted many times and then being dropped. Note that our priority scheme (especially the unbuffered version) is so simple that it can be implemented entirely in hardware.

A similar priority rule was examined in [Var90], and was found to improve the throughput of deflection routing as well. The results on the priority deflection scheme will be outlined in Section 5.6.

5.5. QUALITY OF THE APPROXIMATIONS, AND SIMULATION RESULTS

The unbuffered simple scheme is similar to the greedy routing scheme of a *folded butterfly*. A folded butterfly is a butterfly where the nodes of the last stage are the same with the nodes of the first stage (see Fig. 5.8). Each link of the folded butterfly has a buffer for exactly one packet, the one being transmitted. All the nodes of the folded butterfly are seen as potential sources. A new packet is available at a link buffer (including the links of the intermediate stages) with probability p_0. An internal link in our schemes corresponds to a *straight link* of the folded butterfly, that is, a link that connects a node of some stage with the node that has the same binary representation of the next stage. Similarly, a forward link in our schemes corresponds to a *cross link* of the folded butterfly, that is, a link that connects a node of a stage with the node whose binary representation differs in one bit of the next stage. The unbuffered priority scheme also corresponds in a natural way to a priority greedy scheme in a folded butterfly.

Consider a link l, and the two links l_1 and l_2 that lead to it. We want to investigate the quality of the approximation A.5.3.1 used in the analyses of the unbuffered simple and priority schemes. In particular, we are interested in the following questions:

1. Are the packet arrivals during a *particular* slot T on link l_1 independent of the packet arrivals on link l_2 during the same slot?

2. If they are dependent, where does the dependence come from, and how strong is it?

Lemma 1: Events on links l_1 and l_2 at time T, are dependent only through events that happened at time $T - d$ and before.

Proof: By the symmetry of the system we can assume without loss of generality that the links l_1 and l_2 are of dimension d.

The sequence of links traversed by a packet, together with the corresponding times will be referred to as the *time-path* of the packet. Let $P_1 = \{(x_1, t_1), (x_2, t_1 + 1), \ldots, (l_1, T)\}$ be the time-path of a packet that passes from l_1 at time T. We use the same symbol P_1 to denote the

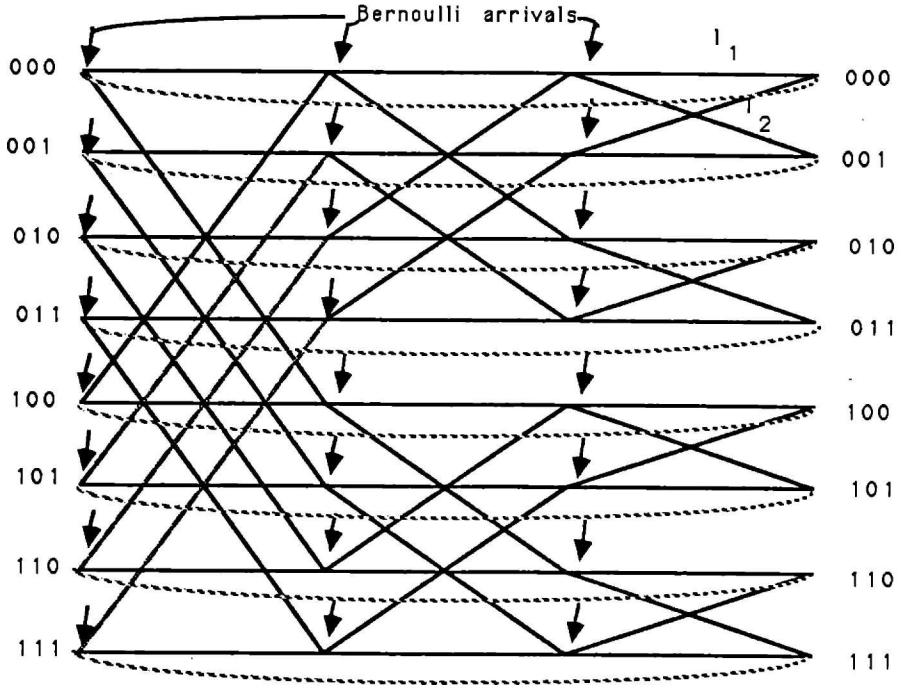

Figure 5.8: A folded butterfly.

event of a packet following that path. Let $P_2 = \{(y_1, t_2), (y_2, t_2 + 1), \ldots, (l_2, T)\}$ be a time-path leading to link l_2 at time T. We are interested in the dependence between events P_1 and P_2.

Consider a packet p that entered the folded butterfly at time t_0 at dimension i_0. During slot $t \geq t_0$ the packet traverses a link of dimension i, where $i + t$ mod d$=i_0 + t_0$ mod d. This is because packets travel dimensions in descending order and there is no buffering. Therefore, the sum $i + t$ mod d of the time slot and the dimension traversed at that slot is a constant of the packet (or of the corresponding time-path), and is called *class* of the packet. We denote the class of packet p by $c(p) = i_0 + t_0$ mod d.

We will denote the dependence between two events A and B by $A \sim B$. It is easy to prove that $\sim$ is an equivalence relation. Two time-paths intersect (or equivalently, two packets collide) only if they pass through the same link at some time. Only time-paths of the same class may intersect, and only packets of the same class may collide. Dependencies are created and spread only through the intersection of time-paths. For example, if time-path A intersects time-path B, and B intersects time path C, then events A and C may be dependent. Events corresponding to time-paths of different classes are independent.

Let H_0 (or H_1) be the sub-butterfly that consists of the nodes whose least significant bit is

equal to zero (or one, respectively). A time-path belongs to H_0 (or H_1) if all the links that it traverses belong to H_0 (or H_1, respectively). The time-paths P_1 and P_2 that lead to links l_1 and l_2 at time T satisfy

$$P_1 \in H_0 \tag{5.43}$$

and

$$P_2 \in H_1. \tag{5.44}$$

Events P_1 and P_2 can be dependent only in the following two cases:

Case A: The time-paths P_1 and P_2 intersect before time T .

Case B: There is an integer k and time-paths $X_1, X_2, \ldots, X_k$ such that

$$P_1 \text{ intersects } X_1 \text{ at time } T_1 < T$$

$$X_1 \text{ intersects } X_2 \text{ at time } T_2 < T$$

$$\vdots$$

$$X_{k-1} \text{ intersects } X_k \text{ at time } T_k < T$$

$$X_k \text{ intersects } P_2 \text{ at time } T_{k+1} < T.$$

Case A cannot happen because of Eqs. (5.43)-(5.44), and the fact that H_0 and H_1 are disjoint. In view of Eqs. (5.43) and (5.44), case B can happen, only if there is an $i \in \{1, 2, \ldots, k\}$ such that time-path X_i crosses a link of dimension d (passing from H_0 to H_1 or vice versa) at some time prior to T. But

$$c(P_1) = c(X_1) = \cdots = c(X_k) = c(P_2) = d + T \bmod \mathrm{d},$$

because any two intersecting time-paths have the same class. Thus $c(X_i) = d+T \bmod \mathrm{d}$, which means that packet X_i crosses dimension d at or prior to time $T - d$. Therefore, X_i intersects with either X_{i+1} or X_{i-1} prior to time $T - d$. This proves that the dependence between events on links l_1 and l_2 is based on an event (collision or non-collision) that has happened before time $T - d$. **Q.E.D.**

Lemma 1 says that the approximating assumption A.5.3.1 is weaker than the following assumption: "Events that happen at time T are independent from events that happen at time $T - d$". The arguments of Lemma 1 are independent of the way conflicts are resolved, and

therefore hold for both the simple and the priority unbuffered scheme. Lemma 1 suggests that the dependence between an event on link l_1 and an event on link l_2 at a given time is weak, and the approximating assumption A.5.3.1 is very accurate.

Another way to see that the previous dependence is very weak is the following. Given that the straight link l_1 has a packet of a particular type, the a posteriori probability that a cross link of dimension d has a packet is the same for all cross links. Let

$$\Delta p(l) = \Pr(l \text{ has a packet} \mid l_1 \text{ has a packet of type } i) - \Pr(l \text{ has a packet})$$

be the difference between the a priori and the a posteriori probabilities. We would like $\Delta p(l_2)$ to be small in order for approximation A.5.3.1 to be accurate. However, we can see that

$$\Delta p(l) = \Delta p(l_2) = \Delta p \text{ for all cross links } l \text{ of dimension } d.$$

The mean total flow through dimension d conditioned on the presence of a packet on link l_1 differs by $N \cdot \Delta p$ units from its a priori value. It is reasonable to expect that the knowledge that link l_1 has a packet will not change the mean flow through dimension d significantly because the latter is a parameter of a global nature. Thus, Δp must be small.

We simulated the unbuffered simple scheme for various network sizes, and several values of p_0. The difference between the analytical and the simulation results has been consistently negligible for all network sizes and all p_0's (see Table 5.1 for $d = 8$). This is a further indication that the parametric equations obtained in Subsection 5.3.1 are very accurate.

p_0	Throughput/node (analytical)	Throughput/node (simulations)
0.9983	0.6325	0.6331
0.9288	0.6401	0.6401
0.8045	0.6539	0.6540
0.6972	0.6657	0.6650
0.6042	0.6754	0.6744
0.5234	0.6827	0.6824
0.4871	0.6853	0.6843
0.3642	0.6888	0.6883
0.3142	0.6859	0.6852
0.2915	0.6831	0.6828
0.2145	0.6628	0.6621

0.1982	0.6552	0.6557
0.1094	0.5712	0.5721
0.0030	0.0448	0.0446

Table 5.1: Simulation and analytical results for the unbuffered simple scheme for $d = 8$.

We have also performed simulations for the buffered simple scheme. The results obtained from the simulations were within 3% from the analytical results. Table 5.2 illustrates the results obtained for $d = 7$ and $k = 1$.

p_0	**Throughput/node (analytical)**	**Throughput/node (simulations)**
0.931384	1.493738	1.451239
0.566517	1.477039	1.433139
0.302901	1.345433	1.354165
0.199937	1.189335	1.162777
0.169829	1.116160	1.092926
0.144199	1.038224	1.020776
0.103110	0.871355	0.861196
0.086444	0.783858	0.777389
0.052758	0.557855	0.554911

Table 5.2: Simulation and analytical results for the buffered simple scheme for $d = 7$ and $k = 1$.

5.6. COMPARISON WITH DEFLECTION ROUTING

In this section we present some results on deflection routing from [Var90], and compare them with corresponding results on the priority scheme of Section 5.4. In Subsection 5.6.1 we describe the *simple* and the *priority* deflection schemes, and the stochastic model under which they are examined. In Subsection 5.6.2 we present the results obtained.

5.6.1. The Deflection Schemes, and the Stochastic Model

Each node has a queue which can hold up to d packets. During each slot the nodes transmit all the packets of the queue, either by transmitting them on their *preferred links*, that is, links that take the packets closer to their destination, or by simply transmitting them on an available link. We assume that new packets are always available, and for every packet that exits the network at some node a new packet enters the network at the same node. At every slot, exactly d packets are received by each node. Some of these packets exit the system because they have arrived at their destination, and are replaced by an equal number of new packets. Under this model the hypercube is a closed network since each node always has exactly d packets. The destinations of the new packets are uniformly distributed over all nodes, except for their origin.

Before describing the priority deflection scheme, we give some definitions.

A *partial* switching assignment is a 1-1 match between packets and preferred links, where each packet (or link) is matched to at most one link (or packet, respectively). A *full* switching assignment is a match between all the d packets residing at the node and the d outgoing links of the node. A partial switching assignment is *wasting* if there exists a packet that has not been assigned to a preferred link, although one of its preferred links is free. By transmitting the packet on this link the number of packets that are sent towards their destinations is increased by one and the assignment remains feasible. In a *non-wasting* switching assignment such a situation is not allowed. In Fig. 5.10, both a and b are non-wasting switching assignments, while c is not.

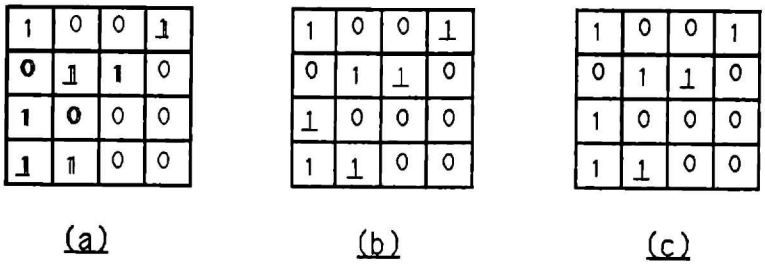

Figure 5.10: Cases a and b correspond to non-wasting assignments, while c is a wasting assignment.

There are many ways to obtain a non-wasting assignment. A simple procedure is the following. At each slot, an order of the d packets (called *processing order*) is found in some way

at each node. The packets are then picked in that order, and each of them is assigned to one of its preferred links, provided that this link has not been assigned to any of the previously considered packets. If more than one unassigned preferred links exist, one of them is chosen at random.

A deflection scheme consists of two phases. During the first phase, called *non-wasting phase*, a non-wasting partial switching assignment is found. This assignment matches some of the packets with an equal number of links. We consider two deflection schemes corresponding to two different processing orders. In the *simple deflection scheme* the processing order is random. In the *priority deflection scheme* the processing order is found as follows. The packets are partitioned in priority classes, so that the i^{th} priority class consists of the packets that are currently located at a distance i from their destination. The order of the packets within the same class is random; however, packets that are closer to their destination precede packets that are far from their destination.

In general, the partial assignment found in the non-wasting phase will cover only z of the d packets with $z \leq d$. In the second phase, called *deflection phase* the partial assignment is extended to a full assignment. This extension is achieved by arbitrarily mapping the remaining $d-z$ packets to the $d-z$ unreserved outgoing links. The $d-z$ packets that are not transmitted over preferred links, increase their distance to the destination. We will refer to such events as *packet deflections*. Every time a packet is deflected, the number of links it has to traverse increases by two.

The rationale behind the priority deflection scheme is the following. In the random processing order the packets that are at a distance one from their destination have a higher probability of being deflected than packets at distance $i > 1$ from their destination. To see that consider a packet which is one hop away from its destination. Such a packet has only one preferred link, and the probability that this link will have already been assigned when the packet is processed is large. In contrast, a packet at distance $i > 1$ from its destination has i preferred links, and will probably not be hurt if some of its preferred links have been assigned to other packets. A packet at distance d from its destination is never deflected, and it is logical to assign it to a link only after all the packets at distance $i < d$ from their destination have been processed.

5.6.2. Steady State Throughput of the Deflection Schemes

The steady state throughput of the simple and the priority deflection scheme has been calculated in [Var90] through approximate numerical analysis, and simulations. In this subsection we present the simulation results. Before doing so we give an upper bound on the throughput.

Let λ be the mean total throughput of the hypercube. Since the number of packets in the hypercube is constant and equal to Nd, Little's theorem gives

$$Nd = \lambda T,$$

where T is the mean delay of a packet from the time it is accepted in the network until the time it arrives at its destination. For uniformly distributed packet destinations we have

$$T \geq \frac{d}{2}\frac{N}{N-1}.$$

Combining the last two equations we get

$$\lambda \leq 2(N-1). \tag{5.45}$$

Figure 5.11 illustrates the simulation results obtained for the simple and the priority deflection schemes, together with the analytical results obtained for the priority scheme (with buffer size $k = 0$ and $k = 1$) of Section 5.4. Note that as the dimension of the hypercube increases the throughput of the deflection schemes increases. However, for small dimensions the priority scheme with $k = 1$ outperforms deflection routing. Thus, the priority scheme may be preferable for small hypercube dimensions. If we take into account that the switch used at each node by the priority scheme is simpler, faster, and less expensive than the crossbar switch used by the deflection schemes, then the priority scheme may be preferable for large hypercube dimensions as well.

The average delay of of the deflection schemes satisfies

$$T = \frac{d}{2}\frac{N}{N-1} + 2E(\text{Number of Deflections}), \tag{5.46}$$

where E(Number of Deflections) is the average number of deflections suffered by a packet. The first term of the right hand side of the preceding equation comes from the fact that the mean delay of a packet when it is not deflected is equal to $\frac{d}{2}\frac{N}{N-1}$. For the second term note that every time a packet is deflected, its delay increases by two steps. Figure 5.12 illustrates the

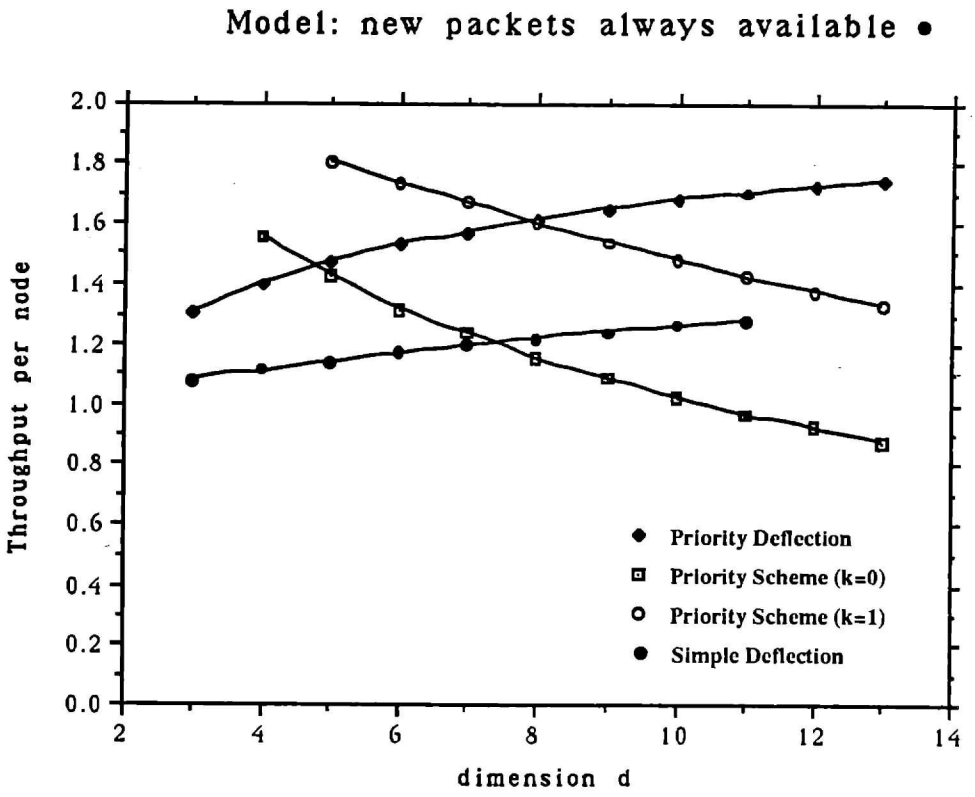

Figure 5.11: Throughput per node of (1) the simple deflection scheme, (2) the priority deflection scheme, (3) the unbuffered priority (idling) scheme, and (4) the buffered (k=1) priority (idling) scheme.

average number of deflections suffered by a packet in the priority deflection scheme, for various dimensions d of the hypercube (the average number of deflections is obtained by Eq. (5.46) and Little's theorem by using the simulation results for the throughput λ).

An interesting observation concerning Fig. 5.12 is the following. As d increases, the average delay suffered by a packet also increases, but the E(Number of Deflections) seems to remain almost constant (between 0.42 and 0.48 for $d = 3, 4, \ldots, 13$). If the average number of deflections is actually bounded above by a constant for every d, then the average delay T of the priority deflection scheme will be $T = \frac{d}{2}\frac{N}{N-1} + O(1)$. If in addition, the higher moments of the average number of deflections are also $O(1)$, then the throughput of the priority deflection scheme will tend to the upper bound of two packets per node, which is the maximum possible for uniformly distributed destinations. We could not prove this by a rigorous analysis, so we leave it as a conjecture.

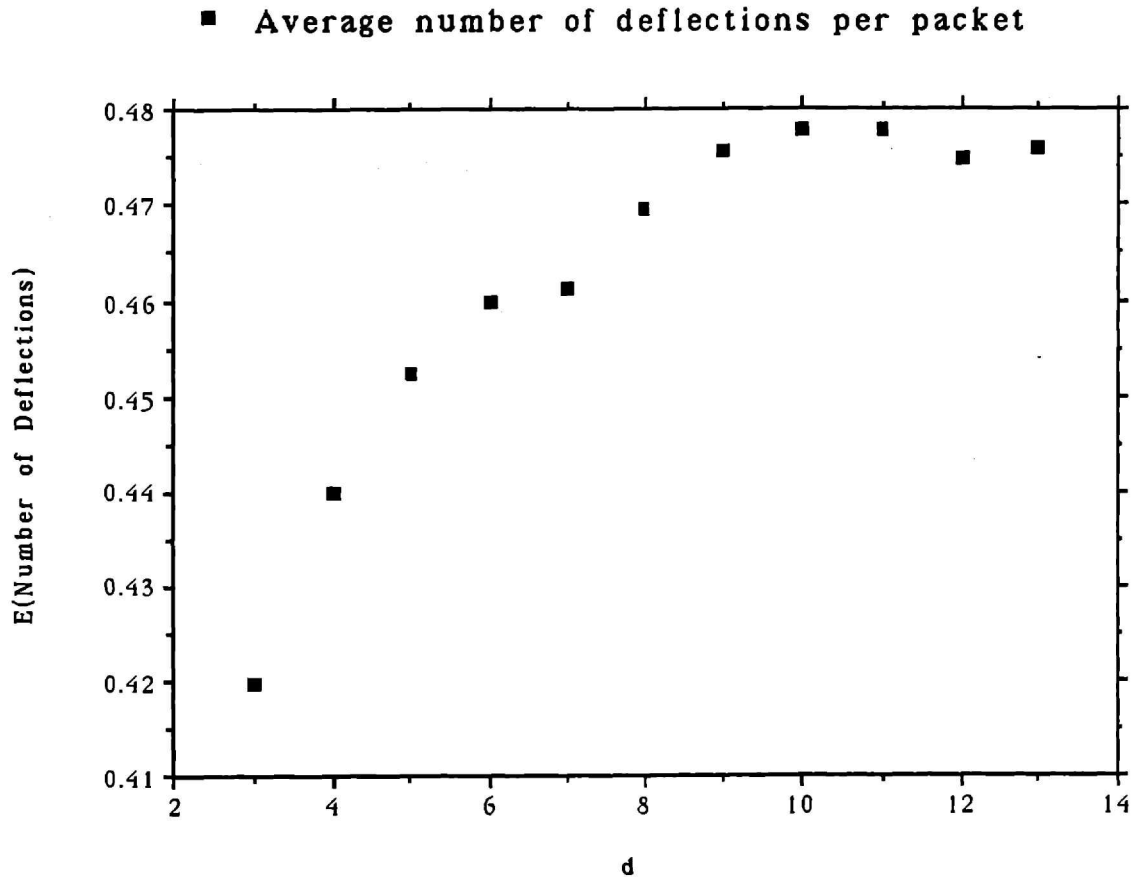

Figure 5.12: Average number of deflections per packet for the priority deflection scheme.

5.7. DISTRIBUTION OF PACKET DISTANCES TO DESTINATION IN SHORTEST PATH ROUTING SCHEMES

This section is independent from the previous sections, but includes some potentially interesting results. We consider a shortest path routing algorithm in a regular topology where packets are not dropped. We are interested in the probability π_i that a packet randomly picked from the head of some queue at some random instant, is at distance i ($1 \leq i \leq d$) from its destination.

Let m_i be the probability that the destination of a new packet is at distance i from its origin. For example, for a d-dimensional hypercube, and uniform distribution of the destinations we

have

$$m_i = \frac{\binom{d}{i}}{N-1}.$$

The analysis, however, can be carried out for any regular topology, any distribution m_i of the destinations and any shortest path routing scheme, under assumptions of "symmetry" guaranteeing that all nodes and links are "equivalent" in a statistical sense.

We pick at random a packet located at the head of a queue. Then the probability that the packet was originated at distance i from its destination is proportional to i (such a packet passes i times from the head of a queue until its delivery; since the instant is random the probability that the packet is picked is proportional to i), and is also proportional to m_i. Thus

$$\text{Pr(picked packet originated at distance i from source)} = \frac{im_i}{\sum_{j=1}^{\text{diameter}} jm_j},$$

where a normalizing factor has been included. Note that

$$\overline{m} = \sum_{j=1}^{\text{diameter}} jm_j$$

is the mean number of hops that a packet has to travel. For uniformly distributed destinations $\overline{m}$ is the mean internode distance of the network. The chosen packet is at distance $i \leq j$ from its destination with probability $1/j$, for any $i \in \{1, 2, \ldots, j\}$. Thus, the probability that a packet randomly picked from the head of a queue is currently at distance i from its destination is

$$\pi_i = \sum_{j=i}^{\text{diameter}} \frac{m_j}{\overline{m}}. \tag{5.47}$$

A first observation is that this probability is not altered if the packet is picked from any position of the queue (as long as a FIFO queuing discipline is used or, more generally, a discipline which does not distinguish the packets according to their distance from the destination). A second observation is that π_i is decreasing with i. For $i = 1$ we get

$$\pi_1 = \frac{1}{\overline{m}}.$$

This indicates that in any shortest path routing scheme in which packets are not dropped, the throughput R per node is

$$R = (1-e)\pi_1 d = \frac{(1-e)d}{\overline{m}},$$

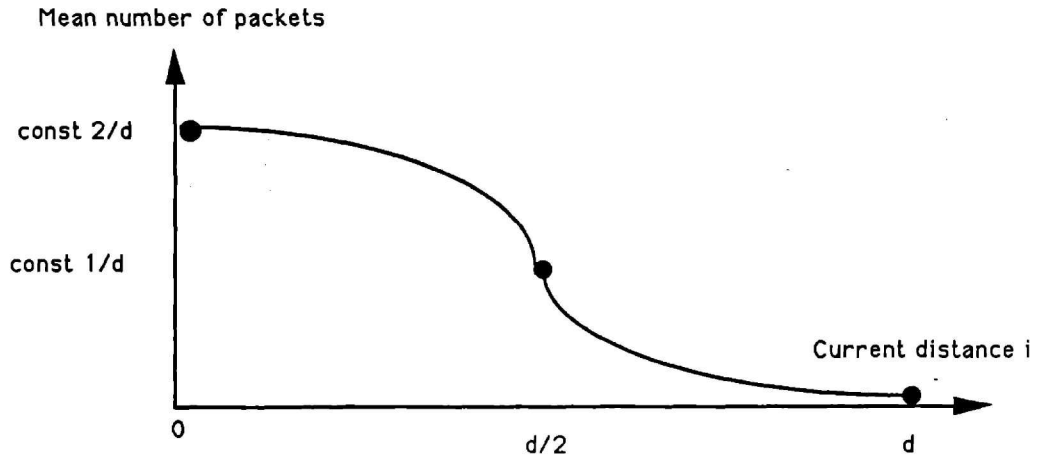

Figure 5.13: Histogram of the number of packets in a hypercube as a function of their current distance from the destination, for uniform distribution of the destinations.

where e is the probability that a link is empty, and d is the degree of the network. We can always achieve throughput of $d/\overline{m}$, provided that we use shortest path routing, have infinite buffers and new packets are always available.

For the hypercube network and uniform distribution of the destinations, Eq. (5.47) gives

$$\pi_i = \frac{2}{d} \sum_{j=i}^{d} \binom{d}{j} \left(\frac{1}{2}\right)^{j} \left(\frac{1}{2}\right)^{d-j}.$$

For d large we can use the Gaussian approximation of the binomial distribution (Central Limit Theorem) and get

$$\pi_i \approx \frac{2}{d} Q\left(\frac{i - d/2}{\sqrt{d}/2}\right),$$

where

$$Q(x) = \int_x^\infty \frac{e^{-y^2/2}}{\sqrt{2\pi}} dy.$$

The larger d is the better the approximation is. A histogram of the number of packets in the hypercube as a function of their current distance from their destination is of the form indicated in Fig. 5.13.

CHAPTER SIX

A Conflict Sense Routing Protocol and its Performance for Hypercubes

In this chapter, we propose a new switching format for multiprocessor networks, which we call Conflict Sense Routing Protocol. This switching format is a hybrid of packet and circuit switching, and combines advantages of both. We initially present the protocol in a way applicable to a general topology. We then present an implementation of this protocol for a hypercube computer and a particular routing algorithm. We also analyze the steady state throughput of the hypercube implementation for random node to node communications.

6.1. INTRODUCTION

There are two general switching formats, *circuit switching* and *packet switching*, that are used in network communications. Circuit switching combines many well-known advantages, but is seriously inefficient. The inefficiency is related to the allocation of a link to a message for more time than required. Packet switching on the other hand is efficient in terms of link utilization since a link is used whenever there is a packet that wants to cross it, but has a number of drawbacks. The queueing delays in packet switching are unpredictable and difficult to control. Flow control mechanisms are needed to slow down transmissions when congestion arises. The delayed feedback, the buffer space needed, and the possibility of dropped packets are other considerations not in favour of packet switching. In multiprocessor systems with

thousands of processors the buffer space per node is going to be small making the dropping of packets a potential problem, as discussed in the preceding chapter.

A solution that has been proposed is the deflection routing scheme examined in Subsection 5.6 (see also [GrG86], [GrH90], [Var90], [HaC90], [Haj91], and [Bra91]). With deflection routing packets are misrouted instead of dropped. This works well for several networks (for example, hypercubes, Manhattan street networks), but not for all (for example, its throughput for the shuffle exchange network is low; see [Max89]), and has the disadvantage that packets do not always follow shortest paths to their destination. Networks not having enough path redundancy will most probably be unsuitable for deflection routing. *Cut-through routing* ([KeK79]), and its variation *wormhole routing* ([DaS87]) have also been proposed for multiprocessor systems, but many theoretical problems are still unresolved. The possibility of deadlock cannot be ruled out for both deflection and wormhole routing, unless special precautions are taken. In practice, most data networks and multiprocessor systems currently use packet or circuit switching. However, for many applications, it is unclear which one of them is preferable, since each has relative advantages at exactly the same areas where the other has disadvantages.

In this chapter we introduce a new switching format, which we call *Conflict Sense Routing Protocol* (or CSR protocol). The CSR protocol is a hybrid of circuit and packet switching. According to it, a packet can enter the network only after having reserved its route (links and buffer space). This resembles circuit switching. A packet, however, reserves a resource only for the slot (or slots) during which the resource will be used. This resembles packet switching since the links and the buffer space are used on a demand basis.

The CSR protocol is more efficient than circuit switching, because in circuit switching the entire path of a packet is reserved as the packet is travelling on any one link of the path, and additional overhead is needed to "tear down" a circuit after all packet transmissions of the circuit have been completed. A further advantage of the CSR protocol over packet switching is that it avoids the waste of resources due to the dropping of packets which have been transmitted for several hops. Still another advantage is that it provides a "built-in" flow control mechanism. Flow control in a multiprocessor computer cannot be the same with the one of a general data networks. One reason is the limited buffer space per node, which will cause packets to be dropped. The flow control mechanisms designed for data networks, where the nodes are bigger and the buffering is cheap, take buffer space for granted (for example, the *go back n protocol*; see [BeG87]). Such protocols will not work well for multiprocessor computers with little buffer space per node. On the other hand, it is inefficient to use the local memory of a processor to buffer

packets, since then we are penalized with large start-up and software delays per transmission, CPU interruptions, etc.. Another drawback of traditional data link control protocols when applied to multiprocessor systems is their reliance on acknowledgements. In parallel computers it is typically impossible to piggyback acknowledgements on the opposite direction traffic (in a network of thousands of processors a particular pair of processors rarely communicates), while the use of separate acknowledgement packets increases the network load significantly. To make things worse, acknowledgements may themselves be dropped increasing the delay and complicating the implementation.

The feedback delay which is proportional to the diameter of the network is another drawback of traditional flow control algorithms, when applied to multiprocessor computers. A transmission window of small size is inefficient when the roundtrip delay is large relative to the transmission time of a packet. On the other hand a window of large size requires too much buffer space for the storage of unacknowledged packets, which is a scarce commodity in parallel computers, and an estimate of the roundtrip delay, which is not always easy.

In contrast to packet switching, the CSR protocol provides almost instantaneous feedback. In fact, a packet knows before entering the network if it will be dropped, in which case it does not enter the network (at the present slot). Note that the protocol guarantees that packets are not dropped even when the buffer space is minimal. The virtually instantaneous feedback makes the use of a size one window efficient, and the storage of packets not yet acknowledged minimal. Furthermore, the processor gets to know very quickly if the packet will eventually arrive at its destination, which is important for "send and wait" type of commands.

In this chapter we initially present the CSR protocol in its generality. The description that we give is independent of the network topology, the routing algorithm used, and the buffer space available. We then specialize the CSR protocol to the case of a hypercube network of processors with buffer space only for the packet being transmitted. We focus on a particular routing algorithm, where packets traverse the hypercube dimensions in descending order. The throughput under various traffic loads is evaluated and found very satisfactory, although the routing algorithm assumed is not the best possible (it involves idling). The results are, we believe, indicative of the improvements that can be obtained by superimposing the CSR protocol on other routing algorithms and networks. For the routing algorithm and buffer that we assume, the protocol guarantees that every packet that enters the network arrives at its destination after d slots, where d is the diameter of the hypercube.

The organization of the chapter is the following. In Section 6.2 we describe the CSR protocol

in its generality. In Section 6.3 we describe an implementation for a particular routing scheme in a hypercube network of processors. In Section 6.4 we evaluate the throughput of the hypercube implementation of the CSR protocol. In Section 6.5 we compare the CSR hypercube implementation to other switching formats and routing schemes, we discuss implementation issues, and we conclude the chapter.

6.2. DESCRIPTION OF THE CSR PROTOCOL

In this section we present the CSR protocol for a general topology, and describe the data structures that are necessary for its implementation. A pair (s, l) will represent the l^{th} link of processor s, and d will represent the diameter of the network. We assume the existence of a routing algorithm which, for each source s and destination v, finds a path leading from s to v, and provides a way to resolve conflicts among packets requiring the same link. The only property that the routing algorithm has to satisfy is that the links traversed by a packet $\mathcal{P}$, and the time instants these links are used do not depend on packets that entered the network after $\mathcal{P}$. We call such a routing algorithm *future oblivious*. The routing algorithm can be deterministic, probabilistic, distributed, and even adaptive, provided that it is future oblivious. For example, an algorithm that gives priority to a given packet over packets that entered the network earlier is *not* future oblivious, while one that gives priority to a packet over packets that entered later (and perhaps for packets that entered at the same time, it gives priority to those that are closer to their destination) is future oblivious.

Each link (s, l) of the network is assumed to have a *link buffer* and an *entry buffer*, denoted by $\mathcal{Q}_l^s$ and $\mathcal{E}_l^s$ respectively. Entry buffers form the interface of a processor with its router and can store only new packets. A packet *enters* (or *is accepted to*) the network when it moves from an entry buffer to the corresponding link buffer. A link buffer can be used only by packets already accepted to the network, and can hold up to K_l^s packets, in addition to the packet being transmitted. We assume that all packets require one time unit for transmission over a link, and a single buffer space for storage.

Following the terminology of [Dal90b] we define a *flit* (from "flow control unit") as the smallest number of bits which can contain routing information, or else the minimum number of bits which can be accepted or rejected by a link buffer. A typical size of a flit is 64 bits. Flits

are used in parallel computers that support wormhole routing (for example the J-machine, see [Dal90b]), and are similar to the set-up messages used in circuit switching. We assume that a node has a way to distinguish control flits from actual data.

A packet stored at an entry buffer sends a flit before entering the network in order to reserve the resources that it will need. This flit reserves a resource (link or buffer) only for the slots during which it will need it. A flit generated at an entry buffer follows the same path that the packet would follow, that is the path provided by the underlying routing algorithm of the network. Flits are treated one at a time by a link, and conflicts concerning the order in which they are considered are resolved in a manner determined by the routing algorithm, for example at random.

Let β be an upper bound on the time required for a flit to travel a distance of $2d$ links, where d is the diameter of the network. The unit of time is taken as the time required for a packet transmission over a link. We can take

$$\beta = 2kd\left(\frac{F}{W} + \gamma\right),$$

where F is the length of a flit in bits, W is the bandwidth of a link measured in bits per unit of time, γ is an upper bound on the propagation and processing delay of flits, and k is the buffer size for packets per link. This is because if a flit is not blocked, at most $k-1$ flits are transmitted ahead of it on a link during a control interval.

The time axis is divided into alternating *control intervals* of length β units of time, where flits are routed and reservations are made, and *transmission intervals* of length equal to one unit of time where packet transmissions actually take place. A control interval is divided into a *forward* and a *backward* phase, each of length $\beta/2$. During the forward phase flits travel from their source to their destination, reserving links and buffer space. Flits that fail to reserve a link are blocked on the spot. After $\beta/2$ time units all flits have either arrived at their destination or have been blocked. In the backward phase flits travel in the opposite direction, carrying feedback information to the source. A way to ensure that flits will not collide on links in the backward phase, is to transmit a flit on a link at time $(2kd - i)F/W$ in the backward phase, if it was transmitted on the same link (in the opposite direction) at time iF/W in the forward phase. Therefore, the feedback is 100% reliable. For a general network, and a general routing algorithm storage space for kd flits per link is required. For the hypercube CSR implementation that we will give in Section 6.3, we will see that storage space for just one flit per link is adequate. Flits that have been blocked carry negative acknowledgements

(or NACKs for brevity), while flits that have made all the necessary reservations carry positive acknowledgements (or ACKs). A NACK prevents the packet from entering the network during the transmission interval of the current slot. This saves bandwidth since such a packet would be dropped, if transmitted, at exactly the same link where the flit was blocked. This is the reason we call the protocol *conflict sense* routing protocol: it senses a conflict before it actually happens. The control interval serves as a "microscopic", inexpensive rehearsal of what would happen if the packet was transmitted. In this way, after a feedback delay of β time units, each entry buffer knows whether the packet (if any) that it holds can be transmitted without being dropped, or not.

The way the reservations are made and the data structures required for this purpose are described next. For every link queue Q_l^s there is a list $\mathcal{L}_l^s$, called *reservation list*, whose elements represent future transmission intervals. The first element represents the next transmission interval. At the end of a transmission interval, the first element of the list is deleted. The element of $\mathcal{L}_l^s$ which corresponds to the t^{th} transmission interval (transmission intervals are counted with respect to the present control interval) is denoted by $\mathcal{L}_l^s[t]$, and is composed of two fields, denoted by $\mathcal{L}_l^s[t]\_link$ and $\mathcal{L}_l^s[t]\_buffer$. In case there is no buffer space at the links except for the packets currently under transmission, the two fields collapse into one. The field $\mathcal{L}_l^s[t]\_link$ is equal to one if the link (s,l) has already been reserved for the t^{th} transmission interval, and zero otherwise. The field $\mathcal{L}_l^s[t]\_buffer$ takes integer values between zero and the buffer size K_l^s. It is equal to the number of buffer spaces of Q_l^s already reserved for the t^{th} transmission interval.

Each flit f carries with it a *counter*, denoted by c_f. The counter of a flit generated at an entry buffer is originally set to one. In the forward phase of a control interval the flit travels on the path provided by the routing algorithm from the source to the destination. Let $\mathcal{L}_l^s$ be the reservation list of link (s,l), respectively, at the time when f is considered by link (s,l), where (s,l) is a link on the path of f. We define T as the minimum integer that satisfies

$$c_f \leq T,$$

$$\mathcal{L}_l^s[T]\_link = 0,$$

and

$$\mathcal{L}_l^s[t]\_buffer < K_l^s \quad \text{for all } t \in \{c_f, c_f+1, \ldots, T-1\}.$$

If such a T exists, the link (s,l) is reserved for the T^{th} transmission interval by f. A buffer space at Q_l^s is also reserved for the intervals c_f up to $T-1$. At the same time the reservation

list of (s, l) and the counter of f are updated according to

$$c_f := T + 1,$$

$$\mathcal{L}_l^s[t]\_link := 1,$$

and

$$\mathcal{L}_l^s[t]\_buffer := \mathcal{L}_l^s[t]\_buffer + 1 \quad \forall t \in \{c_f, c_f + 1, \ldots, T - 1\}.$$

If a T that satisfies the previous relations does not exist, the flit is blocked, and the reservation fails. During the backward phase of the control interval such a flit returns to its source entry buffer by using the reverse path, carrying a negative acknowledgement (NACK) and freeing the links and buffer space it has reserved in the forward phase. A packet which receives a NACK does not enter the network at the next transmission interval and will retry to make the necessary reservations at some subsequent control interval. If on the other hand a flit manages to reach its destination reserving all the necessary resources, then at the backward phase it returns to its source entry buffer as a positive feedback. The corresponding packet will enter the network at the immediately following transmission interval, and will arrive at its destination after several transmission intervals by using the links and buffer space already reserved for it. If a packet that receives a NACK always retries at the next control interval, then the protocol preserves the order of the packets.

A last issue that has to be dealt with is the method of recording which packet reserved a link for a particular slot. One way is to store that information at the intermediate nodes, by having a third field at $\mathcal{L}_l^s[t]$ which will record the sequence number of the packet that reserved (s, l) for the t^{th} slot. A different approach is to have the bookkeeping information attached to the packet. This is done by having the flit record the sequence of values $c_f^{(i)}$ that it takes after each hop i (or, even better, the differences $c_f^{(i)} - c_f^{(i-1)}$). In the case where there is buffer space only for the packet being transmitted the book-keeping information is not needed. Note that it is not necessary to know which packet reserved which particular buffer space, since buffer spaces can be organized as a pool.

For light load and large β, the CSR protocol has larger delay than packet or circuit switching. For a packet that has to travel k hops the delay is equal to k units of time with packet switching, $k + \beta$ with circuit switching, and $k(1 + \beta)$ with the CSR protocol. For heavy load or small β, the CSR protocol is expected to have smaller delay than circuit or packet switching, because it uses links more efficiently (which means higher throughput and smaller input queueing delay).

6.3. A HYPERCUBE CSR PROTOCOL

In this section we will describe a hypercube implementation of the CSR protocol. We assume a particular routing algorithm, which is future oblivious, and superimpose on it the CSR protocol. The routing algorithm is simple, similar to the algorithms examined in Chapter 5, and assumes simple inexpensive switches for the nodes. This makes the throughput of the CSR implementation directly comparable to the throughput of the packet switching schemes of Sections 5.3 and 5.4. We start by describing the model assumed for a hypercube node, and the routing algorithm used.

6.3.1 The Hypercube Node Model and Routing Algorithm

Each link of a node has an entry buffer, which can hold one new packet. The entry buffer of link i of node s is denoted by $\mathcal{E}_i(s)$. The entry buffer is ready to accept a new packet only if the previous packet has reserved the links it will need, and positive feedback has been received. A packet that receives a NACK retries to make the reservations at the next control interval. An entry buffer holding a packet for which no positive feedback has been received is said to be *backlogged.* New packets arrive at the entry buffer of a link. New packets arriving at backlogged entry buffers are discarded.

Each node s has d queues, each of them associated with a link of the node. The link queue of link i of node s is denoted by $\mathcal{Q}_i(s)$. A link queue is composed of two *buffers* which can hold only one packet each. The first buffer is called *forward buffer*, denoted by $\mathcal{Q}_i^1(s)$, and is used only by packets which want to cross the i^{th} dimension. The second buffer, denoted by $\mathcal{Q}_i^0(s)$, is called *internal buffer*, and is is used only by packets which do not want to cross the i^{th} dimension. The queues of the nodes are linked in the following way; the internal buffer $\mathcal{Q}_i^0(s)$ is connected to link queue $\mathcal{Q}_{(i-1)\bmod d}(s)$ of the same node and the forward buffer $\mathcal{Q}_i^1(s)$ is connected to queue $\mathcal{Q}_{(i-1)\bmod d}(s \oplus e_i)$ of the neighbor node $s \oplus e_i$ (see Fig. 6.1). This router organization results in node switches which are simpler, faster, and less expensive than cross-bar switches (see the comments on the node model in Section 6.5).

The routing algorithm is the following. A new packet generated at a node selects a link, say link l, with equal probability independently of its destination and competes for one of the

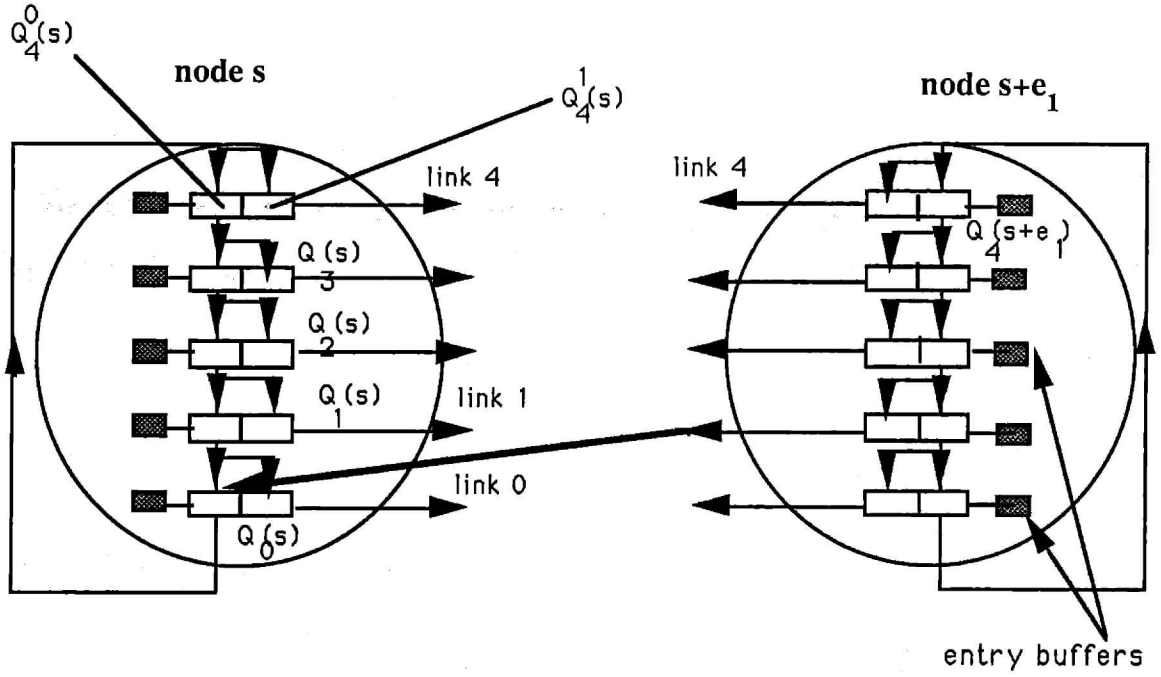

Figure 6.1: A node of the hypercube.

two buffers of the l^{th} link queue, $\mathcal{Q}_l^1(s)$ or $\mathcal{Q}_l^0(s)$, depending on whether it wants to use the l^{th} dimension or not. The packet traverses the dimensions in descending (modulo d) order, starting from the random dimension l. In particular, consider a packet which arrives at queue $\mathcal{Q}_i(s)$ of node s, either from buffer $\mathcal{Q}^0_{(i-1)\mathrm{mod}d}(s)$ of s, or from buffer $\mathcal{Q}^1_{(i-1)\mathrm{mod}d}(s \oplus e_{i-1})$ of the neighbor node $s \oplus e_{i-1}$, or from the entry buffer $\mathcal{E}_i(s)$. The i^{th} bit of its routing tag is checked, and depending on whether it is equal to one or zero, the packet claims buffer $\mathcal{Q}_i^1(s)$ in order to be transmitted to queue $\mathcal{Q}_{(i-1)\mathrm{mod}d}(s \oplus e_i)$ of the neighbor node $s \oplus e_i$, or it claims buffer $\mathcal{Q}_i^0(s)$ in order to be internally passed to the next link queue $\mathcal{Q}_{(i-1)\mathrm{mod}d}(s)$ of the same node. If two packets require the same link and there is not enough buffer space at the link, one of them is dropped. We will only analyze the case where each link buffer has space for only one packet, the one being transmitted. Note that when the CSR protocol is superimposed on the routing algorithm packets are dropped due to collisions: the flit of one of the packets that collide returns to its source carrying a NACK, preventing the packet from entering the network.

The proposed routing scheme is of the *idling* type, because of the internal transmissions that it employs. Under this routing scheme internal and forward links are mathematically equivalent. In the case of no buffering, packets are removed exactly d slots after entering the

network. If a packet arrives at its destination node earlier, it travels within the node until the time of its removal comes.

6.3.2 Superimposition of the CSR Protocol on the Hypercube Routing Algorithm

In this subsection we describe how the CSR protocol is superimposed on the hypercube routing scheme of the previous subsection. The unit of time is taken as the time required for the transmission of a packet over a link. Let β be the time (in units of time) required by a flit to travel a distance of $2d$ links. Then

$$\beta = 2d\left(\frac{F}{W} + \gamma\right),$$

where F is the length of a flit in bits, γ is the processing and propagation delay for flits, and W is the bandwidth of a link measured in bits per unit of time. The time axis is divided into slots, each of which has duration $1+\beta$ units of time. During the first β time units, the flits are transmitted to make reservations for the new packets that want to enter the network. During the remainder of the slot the old packets and the new packets that have made the necessary reservations are transmitted one hop.

An entry buffer holding a packet that wants to enter the network (a new packet or one that is being retransmitted) sends a flit to the packet's destination, containing the packet's routing tag. A flit originated at link l is transmitted during the i^{th} step over the $l - i$ mod d dimensional forward (or internal) link of a node, if the i^{th} bit of its routing tag is a one (or a zero, respectively), provided that this link has not been reserved by another packet. At the same time the flit makes a reservation of that link for the i^{th} slot (only). When two flits try to reserve a link at the same time and for the same transmission interval, one of them is selected at random to make the reservation, and the other is blocked. Flits that find a link already reserved are also blocked.

When all flits have been blocked or have arrived at their destination, which happens after at most $\beta/2$ time units, a backward phase begins. In the backward phase each flit which was blocked follows the reverse route to its origin carrying a negative acknowledgement (NACK), and freeing the links that it had reserved. The NACK will prevent the corresponding packet from entering the network during the transmission interval of the current slot. Flits which reserve all the links to their destination, return in the backward phase to their origin following

the reverse path than the one followed in the forward phase, and carrying a positive feedback (ACK). A flit is transmitted over a link at step $2d-i$ in the backward phase if it was transmitted on the same link at the opposite direction during the i^{th} step of the forward phase. In this way there are no conflicts between flits in the backward phase. After a packet enters the network, it follows its path knowing that it will not collide with any other packet; the only information it needs is its routing tag. If an entry buffer receives a negative feedback, it tries to make the necessary reservations at one of the next control intervals. If we require only ACKs to return to their origins (in this case after a constant delay of β time units, a NACK is assumed), then even with storage for just one flit per link, ACKs never get lost. This is because the paths of the flits that made it to their destination have the same length.

The preceding hypercube implementation indicates some of the typical advantages of a CSR protocol. First, packets that are going to be dropped are not allowed to enter the network. This does not allow congestion to feed on itself. Second, the feedback is obtained as soon as possible. This makes the use of a window of size one possible and efficient at the same time, and reduces the required buffer space for new packets to just one packet. All packets accepted to the entry buffer (router) arrive at their destination with constant delay. Whenever a packet is successfully transmitted, the corresponding entry buffer enables the processor to insert a new packet. Resources are reserved for as long as they are needed and yet all the advantages of circuit switching are maintained. The hypercube CSR protocol example indicates that by using capabilities available in multiprocessor systems, namely the possibility to efficiently route flits through the knowledge of the topology, the flow control mechanism becomes easy and efficient.

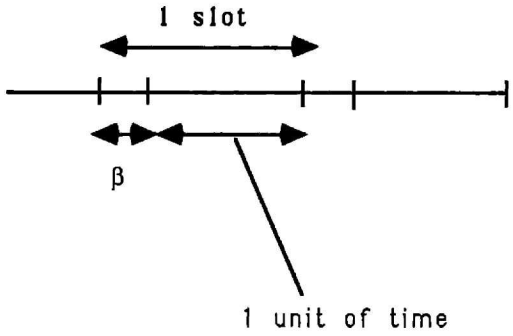

Figure 6.2: The time axis divided into slots.

The control flits can also be transmitted "off" channel. In a VLSI implementation of parallel

computers, there are usually many wires for each link, and the bandwidth of the link is proportional to the number of these wires. In such systems several bits are transmitted in parallel over a link during a clock cycle. In an implementation of the CSR protocol, one would probably choose to dedicate one of these wires to the control flits in order to simplify the design. This corresponds to a kind of FDMA multiplexing as opposed to a TDMA multiplexing of control information and data.

6.4. PERFORMANCE ANALYSIS OF THE CSR PROTOCOL FOR HYPERCUBES

In this section we present an approximate analysis of the throughput of the hypercube CSR protocol described in the previous section. We assume that packets having a single destination are generated at each node, and the destinations of the packets are uniformly distributed over all the hypercube nodes. Packets are being generated over an infinite time horizon, and require one unit of time for transmission over a link. We are interested in the average throughput when the network reaches steady state. We will limit our attention to the case where the link buffers have space only for the packet being transmitted.

Assuming that both the control flits and the data packets are using the same channel, a slot is defined to be equal to $1 + \beta$ units of time. The probability that the entry buffer of a link tries to insert a new packet during a slot is called *attempt rate* and will be denoted by p_0. The attempt traffic is the result of the merging of newly generated packets and retransmissions. Let m be the ratio of the number of backlogged entry buffers to the total number of entry buffers. Let also q_a be the probability of a new packet arrival at an entry buffer of a link, and q_r be the probability with which a blocked packet retries to enter the network (by making the necessary reservations) during a control interval. Then the attempt rate is

$$p_0(m) = (1 - m)q_a + mq_r, \tag{6.1}$$

If retransmissions are sufficiently randomized, it is plausible to approximate the process of attempted reservations from an entry buffer, by an independent Bernoulli process with parameter $p_0(m)$, where m is the fraction of backlogged entry buffers in the system. If retransmissions are not attempted from the same entry buffer, but from another available entry buffer of the same node, then more randomization is added, and the Bernoulli approximation is expected

to be more accurate even for $q_r = 1$. This approximating assumption is reminiscent of the approximating assumption used in the analysis of various multiaccess systems (for example the Aloha protocol, see [BeG87]), where the aggregate traffic of new arrivals and retransmissions is modelled as a Poisson process.

The system that we analyze has some similarities with a multiaccess system (for example, an Aloha system). Conflicts over links or buffer space correspond to collisions in a multiaccess system. An important difference is that packets in the CSR protocol collide when they request the same link for the same slot, while in Aloha whenever two nodes transmit simultaneously there is always a conflict. Another difference is that in our system whenever a conflict appears one of the packets is granted the link (or buffer), while in Aloha whenever a collision happens all transmissions are destroyed. The feedback in our system requires β time units, while in multiaccess systems it is usually assumed instantaneous. The CSR protocol also has similarities with the Carrier Sense Multiaccess protocol, since they both "sense the channel" before transmitting, in order to avoid collisions.

It is possible that a flit reserves a link l during some control interval and frees it later in the same control interval due to its failure to reserve the remaining of its path. We will refer to such a reservation as a *ghost reservation*, as opposed to a *confirmed reservation* where the flit after reserving link l, it also reserves the rest of its path. Let $p(t-i+1, t:l)$, $i = 1, 2, \ldots, d$, be the probability that a particular link l is reserved (by a confirmed or a ghost reservation) on the $t-i+1^{\text{th}}$ control interval for the t^{th}transmission interval. Assuming that the system eventually reaches steady state, the following limit exists and is independent of l:

$$p_i = \lim_{t\to\infty} p(t-i+1, t:l), \quad i = 1, 2, \ldots, d.$$

Note that we have $p_{i+1} \leq p_i$, for all $i \in \{0, 1, 2, \ldots, d-1\}$. Since $p(t-i+1, t:l)$ is independent of l for any t (and not only in steady state), we will sometimes omit the l.

Consider two flits f_1 and f_2 corresponding to packets $\mathcal{P}_1$ and $\mathcal{P}_2$, which try to make the necessary reservations during control intervals t_1 and t_2, starting with dimensions d_1 and d_2, respectively. Packets $\mathcal{P}_1$ and $\mathcal{P}_2$ may request link l for the same slot t ($t \geq \max(t_1, t_2)$) only if l is on their path, all links needed prior to slot t have been reserved, and $t_1 + d_1 \bmod d = t_2 + d_2 \bmod d$. If $t_1 < t_2$ then $\mathcal{P}_1$ is not affected by the presence of $\mathcal{P}_2$, since its attempt to make the reservations is made at a control interval prior to $\mathcal{P}_2$'s arrival. In this case, f_2 can reserve link l only if f_1 fails to reserve all of its its path. If $t_1 = t_2$ (and $d_1 = d_2$) then f_1 and f_2 will claim the same link provided that it is on their path and they have reserved all other

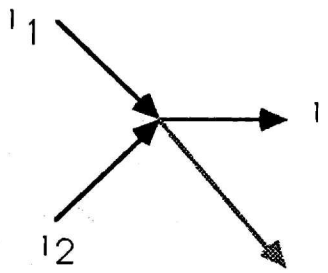

Figure 6.3: The two links l_1 and l_2 leading to l.

links they need prior to l. If the link is free, then it is allocated to one of them arbitrarily.

We want to calculate the steady-state probability that link l is reserved (either by a ghost reservation or a reservation which is confirmed) during the control interval $t-i+1$ for the transmission interval t, for $i > 1$. Let l_1 and l_2 be the internal and forward links, respectively, that lead to l. Link l may be reserved either by a flit f_1 coming on l_1, or by a flit f_2 coming on l_2. Link l can be reserved during the $t-i+1^{\text{th}}$ control interval for transmission interval t by a flit coming on l_1 only if:

- a reservation was made for l_1 for the $t-1$ transmission interval during control interval $t-i+1$; this happens with probability $p(t-i+1, t-1 : l_1)$,
- link l is on the flit's path (given that l_1 is on the flit's path); this happens with probability $1/2$,
- no confirmed reservation has been made for l during a previous control interval, and no reservation (confirmed or ghost) has been made by a flit coming on l_2 during the same control interval for transmission interval t.

Thus,

$$p(t-i+1, t : l) = 2\frac{p(t-i+1, t-1 : l_1)}{2}\left(1 - \Pr(A \mid B)\right), \quad i = 2, 3, \ldots, d, \tag{6.2}$$

where

A is the event that a confirmed reservation has been made for link l for transmission interval t during a previous control interval, or a reservation has been made during the current control interval for transmission interval t by a flit coming on l_2, and

B is the event that f_1 reserved link l_1 for the transmission interval $t-1$ during control interval $t-i+1$.

The factor 2 in Eq. (6.2) accounts for the fact that l can be reserved either by a flit coming on

l_1, or by a flit coming on l_2. Given that event B occured, we know that no confirmed reservation has been made for l for the transmission interval t by a flit coming from l_1. Therefore, the probability of the event A is equal to the (conditional on B) probability that some flit f_2 reserved l_2 for the transmission interval $t-1$ during control interval $t-j+1$ with $j=i+1,\ldots,d$, it chose link l, and its reservation was finally confirmed. Ignoring the conditional on B, this probability can be approximated by

$$\frac{1}{2}\sum_{j=i+1}^{d} p(t-j+1,t-1:l_2)\frac{p(t-j+1,t-j+d)}{p(t-j+1,t)}. \tag{6.3}$$

The ratio

$$p(t-j+1,t-j+d)/p(t-j+1,t)$$

in Eq. (6.3) is the probability that the reservation of l by f_2 was finally confirmed.

The probability that a flit f_2 claims link l during the *same* control interval $t-i+1$ with f_1, and for the same transmission interval t, and it is granted the link can also be approximated, ignoring the conditional on B, by

$$\frac{1}{4}p(t-i+1,t-1:l_2). \tag{6.4}$$

The factor 1/4 is the probability that f_2 requests link l (given that it reserved l_2), and it is selected instead of f_1. Combining Eqs. (6.3) and (6.4) we get

$$\Pr(A \mid B)=\frac{1}{2}\sum_{j=i+1}^{d} p(t-j+1,t-1:l_2)\frac{p(t-j+1,t-j+d)}{p(t-j+1,t)}+\frac{1}{4}p(t-i+1,t-1:l_2).$$

The preceding equation, together with Eq. (6.2) gives

$$p(t-i+1,t:l)=p(t-i+1,t-1:l_1)\cdot$$
$$\left(1-\frac{1}{2}\sum_{j=i+1}^{d} p(t-j+1,t-1:l_2)\frac{p(t-j+1,t-j+d)}{p(t-j+1,t)}-\frac{p(t-i+1,t-1:l_2)}{4}\right),\ i=2,3,\ldots,d. \tag{6.5}$$

Taking the limit $t\to\infty$ and using the symmetry with respect to the links we obtain from Eq. (6.5) that

$$p_i=p_{i-1}\left(1-\frac{1}{2}\sum_{j=i}^{d-1} p_j\frac{p_d}{p_{j+1}}-\frac{p_{i-1}}{4}\right)\quad \text{for } i=2,3,\ldots,d, \tag{6.6}$$

which yields

$$p_{i-1}=2-p_d\sum_{j=i}^{d-1}\frac{p_j}{p_{j+1}}-\sqrt{\left(2-p_d\sum_{j=i}^{d-1}\frac{p_j}{p_{j+1}}\right)^2-4p_i}\quad \text{for } i=2,3,\ldots,d. \tag{6.7}$$

Regarding the reservations made during control interval t for the (following) transmission interval t (i.e., the case $i = 1$), we have

$$p_1 = p_0 (1 - (d-1)p_d), \tag{6.8}$$

where $(d-1)p_d$ is the probability that a confirmed reservation exists for a link at the beginning of the control interval (the probability that a confirmed reservation has been made on the link for the immediately next transmission interval by a packet of type i, $i = 2, \ldots d$, is equal to p_d), and p_0 is the probability that a new packet is available.

For a particular value of p_d, we can use Eq. (6.7) to find p_i in terms of $p_{i+1}, \ldots, p_d$ for each i, and then Eq. (6.8) to find the corresponding p_0. Repeating this for various values of p_d we obtain a curve that gives p_d as a function of the attempt rate p_0. It is possible to prove inductively that p_d is a monotonically increasing (and 1-1) function of p_0.

Figure 6.4 illustrates the results obtained for $d = 11$. The horizontal axis corresponds to both the fraction m of backlogged entry queues and the attempt rate p_0, which are related by the linear equation (6.1). The vertical axis corresponds to the throughput per node (curve), which is

$$R = 2dp_d,$$

and the arrival rate of new packets per node (straight lines). The throughput and the arrival rate are measured in packets per $1 + \beta$ units of time. For each value of the probability of a new arrival q_a, the maximum throughput is obtained for retransmission probability $q_r = 1$ (unlike Aloha). From Fig. 6.4 we see that there is a single stable point in a CSR system, which corresponds to quite high throughput. The two straight lines which we have plotted correspond to $q_a = 0.05$ and $q_a = 1/d$. Both values of q_a represent unusually high loads. In particular, the case $q_a = 1/d$ corresponds to two new arrivals per node per slot. Two packets per node is the maximum throughput that a hypercube network can sustain for uniformly distributed destinations.

We have performed simulations in order to assess the accuracy of the analysis. The simulation results obtained for $d = 7$, together with the corresponding analytical results, are shown in Table 6.1. The relative difference between the simulation and the analytical results is less than 2%.

p_0	Throughput/node (analytical)	Throughput/node (simulations)
0.011666	0.140000	0.142795
0.027465	0.280000	0.283746

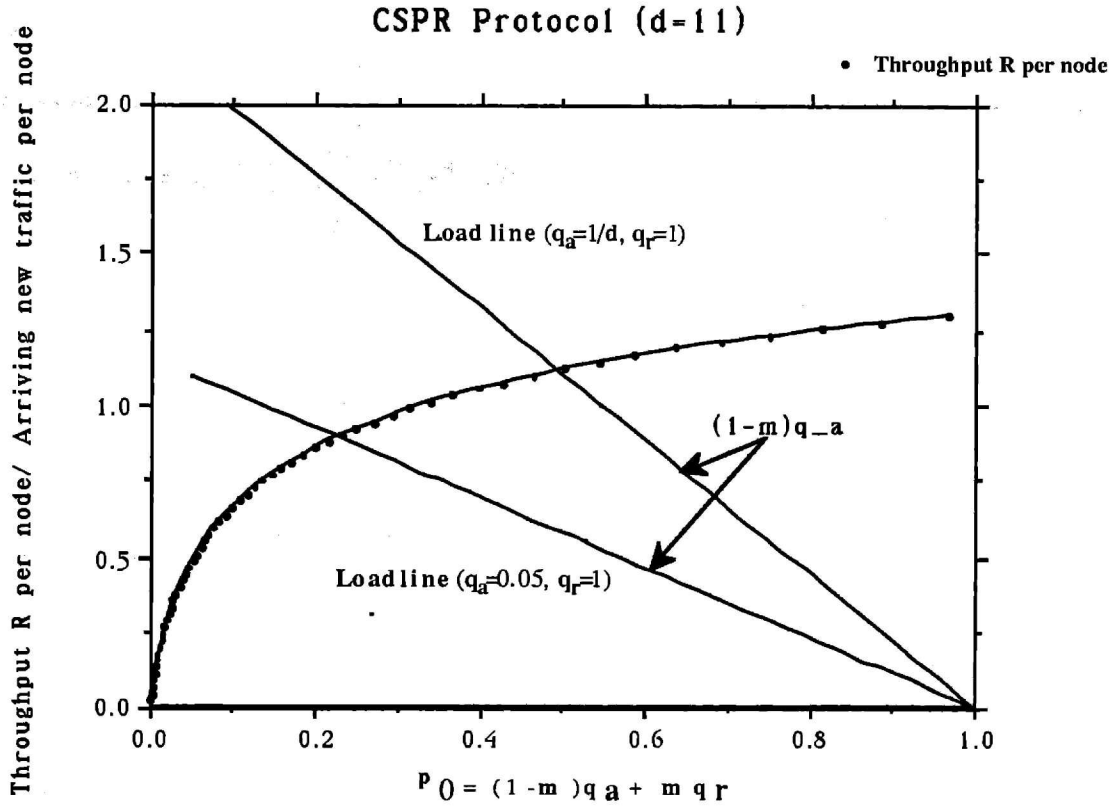

Figure 6.4: Throughput and stable point of the hypercube implementation of the CSR protocol. The probability with which a new packet is available at the entry buffer during a slot is q_a, and the probability of a retrial is q_r. The ratio of backlogged entry queues to the total number of entry queues is denoted by m. The throughput per link cannot be greater than $\min(2, 2dq_a)$ (=1.1 for $q_a = 0.05$).

0.048996	0.420000	0.418328
0.078620	0.560000	0.558200
0.119931	0.700000	0.693059
0.178584	0.840000	0.831379
0.263852	0.980000	0.965929
0.391796	1.120000	1.104581
0.592309	1.260000	1.242851
0.927213	1.400000	1.388006
1.000000	1.422100	1.409178

Table 6.1: Simulation and analytical results for the hypercube implementation of the CSR

protocol for $d = 7$.

6.5. COMPARISON WITH OTHER SWITCHING FORMATS AND ROUTING SCHEMES

The CSR protocol can be applied to various topologies and routing algorithms as a way to perform scheduling and resource management in a synchronous multiprocessor computer. In this section we will compare the hypercube CSR implementation of Section 6.3 to some other switching formats and schemes. The results concerning the switching formats and schemes to be considered are not always directly comparable. Therefore, the comparison is not intended to be a rigorous one, but we believe it will give insight into the relative advantages and disadvantages of each scheme.

We will refer to the *saturation point* of a routing scheme as the ratio of the maximum throughput of the scheme for uniformly distributed traffic over the maximum possible throughput that the network can sustain. In other words, the saturation point is the maximum fraction of the capacity of the network that performs useful work, where the maximum is taken over all possible loads. A link is not doing useful work when (1) it is idle, (2) the packet transmitted on it will be eventually dropped, (3) the packet transmitted on it is being deflected, or it had previously been deflected on a link of the same dimension. Figure 6.5 indicates the saturation point for the hypercube implementation of the CSR protocol (Sections 6.3 and 6.4), the simple and the priority deflection schemes (Section 5.5), the unbuffered simple scheme (Section 5.3), the unbuffered ($k = 0$) and the buffered ($k = 1$) priority scheme (Section 5.4), and the wormhole routing scheme proposed in [KeK79] and [Dal87], and analyzed in [Dal90a].

In order to interpret Fig. 6.5 the following comments are necessary:

CSR Protocol: The saturation point for each hypercube size is obtained from the approximate analysis of Section 6.4 (which is within 2% from simulation results), and correspond to the unbuffered implementation of Section 6.3. The routing algorithm on which the CSR protocol is superimposed assumes a very simple switch at the nodes. Other implementations would probably give a higher saturation point, but they would require more expensive nodes (see the comments below on the node cost).

Deflection Routing: The results of Fig. 6.5 on the simple and the priority deflection scheme

have been obtained through simulations. For the evaluation of the saturation point of both deflection schemes we have taken the probability of access p_0 to be equal to one, that is, we have assumed that packets are always available and enter the hypercube whenever there is an available empty slot. This does not necessarily result in the maximum possible throughput, but the difference is of the order of 1-2%, which is within the statistical error of our simulations, and is in any case negligible.

Simple and Priority Scheme: The results of Fig. 6.5 for the simple and the priority scheme have been obtained from the approximate analysis of Sections 5.3 and 5.4, respectively. The results for the unbuffered version ($k = 0$) of these schemes are very close to simulation results (see section 5.6). For the buffered ($k = 1$) priority scheme, the relative difference between the analytical and the simulation results is larger, but less than 3%. The buffered priority scheme is not directly comparable to the other schemes, because it assumes one buffer space per link in addition to the packet being transmitted; it is included in Fig. 6.5 to show the way buffer space improves performance.

Wormhole Routing: The curve shown in Fig. 6.5 refers to the blocking version of wormhole routing (see [Dal90a] and [Dal90b]). According to this version, when a conflict over a link occurs one of the packets is blocked (instead of being dropped), and when the link is freed it proceeds from the point where it was blocked. The saturation curve of the (blocking) wormhole scheme is obtained by using the analysis of [Dal90a]. This analysis was carried out for k-ary d-cubes, and uses rather crude approximations. The results obtained are within 10% from simulation results, as stated in [Dal90a], although we believe that for hypercube networks the difference may be somewhat larger. The analysis of [Dal90a] uses a continuous (Poisson) model for the arrivals.

The remainder of the section is devoted in examining advantages and disadvantages of each scheme when applied to the hypercube network, and to other topologies.

Saturation Throughput, and Congestion

Ignoring the buffered priority scheme (because it assumes more buffer space than the other schemes), the schemes that are the most interesting in terms of saturation throughput are the two deflection schemes (especially the priority deflection scheme), and the CSR scheme. One reason the CSR scheme does not achieve 100% utilization of the capacity of the network is

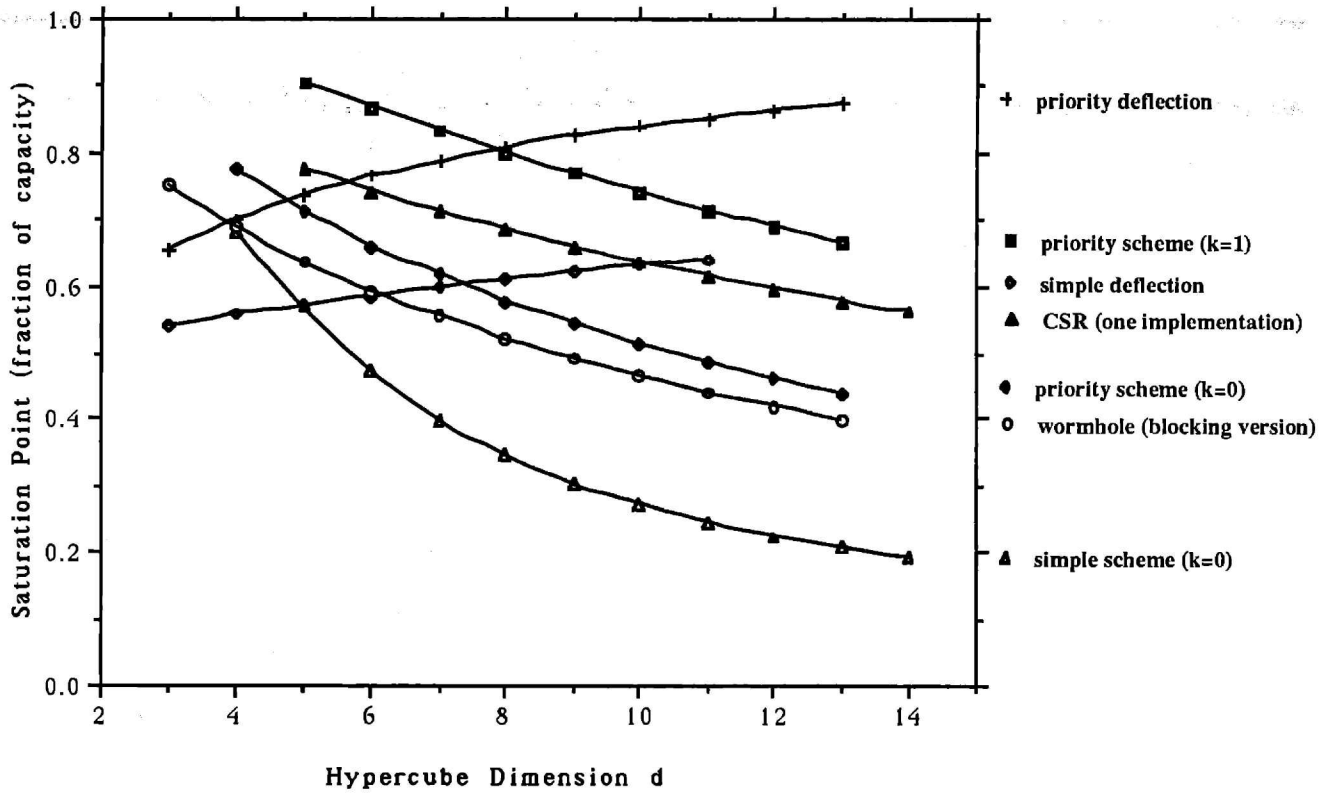

Figure 6.5: The saturation point as a function of the dimension of the hypercube for: (1) the hypercube implementation (Section 6.3) of the CSR protocol for $\beta = 0$ (for other values of β the saturation point should be divided by $1 + \beta$), (2) the priority deflection scheme, (3) the simple deflection scheme, (4) the unbuffered ($k = 0$) priority scheme, (5) the buffered ($k = 1$) priority scheme, (6) the unbuffered ($k = 0$) simple scheme, and (7) the wormhole routing scheme (the curve is subject to significant error).

source blocking: if a packet does not make the necessary reservations then the corresponding entry buffer is backlogged, and the packets behind it do not have access to the network. A second reason is related to the "segmentation" of the available link capacity: some links may be free, but if put together they may not form a whole path (d links form a path only when their link dimensions appear in descending order). This segmentation is not inherent in the CSR protocol, and is mainly due to the simple node switches assumed, which are not cross-bar switches, and do not permit arbitrary switching assignments. The saturation throughput of the CSR scheme has to be divided by $1 + \beta$, since each slot is equal to $1 + \beta$ units of

time. For example, if $\beta = 0.5$ then the curve that corresponds to the CSR protocol has to be multiplied by 2/3. It is, however, important that the CSR protocol does not require additional acknowledgement packets, while the simple, the priority, and the wormhole schemes do require. The two deflection schemes also require the use of acknowledgements for reasons to be explained later, but to a lesser extent. Therefore, for all the schemes there is some overhead not taken into account in Fig. 6.5.

A disadvantage of the unbuffered simple scheme is that, after some point, increasing the offered load decreases the throughput (see Fig. 5.3 of Chapter 5). This makes necessary the existence of a mechanism for controlling the transmission rate of the nodes (probably using the acknowledgements to obtain feedback information about the congestion, and to control the system). This is less of a problem for the priority scheme (buffered or unbuffered), the buffered simple scheme, the simple deflection scheme, and the blocking wormhole scheme (for the version of wormhole routing where packets are dropped the problem is also significant). In the latter schemes the throughput at the saturation point is somewhat smaller than the throughput when the attempted traffic is the maximum possible, but this difference is small (less than 5% for hypercube dimension less than 13). The simulations of the priority deflection scheme, and the approximate analysis of the CSR scheme have indicated that their throughput increases monotonically when the offered load increases.

The results of Fig. 6.5 assume uniform traffic. If the traffic is not uniform then congestion may become a serious problem for deflection routing, as results in [Max90] indicate. Congestion feeds on itself since it forces packets to take longer paths, increasing the utilization, and making other packets to take even longer paths. If the topology is not regular, congestion may become an even more serious problem. In regular topologies which have a severe penalty for deflections (for example, the shuffle exchange network; see [Max89], and [Max90]) deflection routing can be very inefficient in terms of throughput. The CSR protocol behaves better in congestion, is apparently least affected by the choice of the topology.

Packet Delay

Ignoring input queueing delay (in other words, if the load is very light) wormhole routing seems to have the smallest delay of the routing schemes examined. The reason is the pipelining that it achieves between the transmission delay of a packet, and the delay due to the several hops that

the packet has to travel. If, however, the input queueing delay is significant (in other words, if the load is not light) then the two deflection schemes and the CSR scheme (for small β) have the smallest delay. The CSR protocol, deflection routing (ignoring the live-lock problem decribed below), and the blocking version of wormhole routing (ignoring the deadlock problem) have the additional advantage that when a packet enters the network, it is guaranteed to arrive at its destination. This is equivalent to a fast feedback, which results in smaller delay under heavy load (when packet transmissions fail the cost of each retransmission is proportional to the feedback delay, since a node waits for that amount of time before retransmitting).

Node Cost

Deflection routing requires a $d \times d$ cross-bar switch at each node of a hypercube, which has cost $\Theta(d^2)$. The simple and the priority schemes, as well as the hypercube implementation of Section 6.3 of the CSR protocol require a much simpler switch. This switch, which we call *descending-dimensions switch*, is illustrated in Fig. 6.6. The cost of the descending-dimensions switch is only $\Theta(d)$. A cross-bar router is larger and slower, and results in a slower network (the processing time at a node and the clock cycle is larger). The switching assignments possible with the descending-dimensions switch are of course more restricted, and suffer from internal message collisions (the collisions on the internal links of the node model of Fig. 6.1). This results in a degradation in performance, which in the case of the CSR protocol was not severe. Since the descending-dimensions switch uses simple 2:2 switch/merge switches, it can be made to operate very quickly, which may offset the degradation in the performance due to the restrictions in the routing algorithm (see [Dal91]). If the CSR protocol were used with a cross-bar switch, it would probably outperform deflection routing (if we ignore the $1+\beta$ factor); however, we believe that the improvement would not be worth the additional cost. One of the advantages of the CSR protocol is that it performs well even with very simple switches.

Live-lock, and Deadlock

The live-lock problem is unique to deflection routing (see [Max90]). It occurs when packets are transmitted continuously without any chance of reaching their destination. This problem cannot be removed by an end-to-end control, since such packets do not reach their destination.

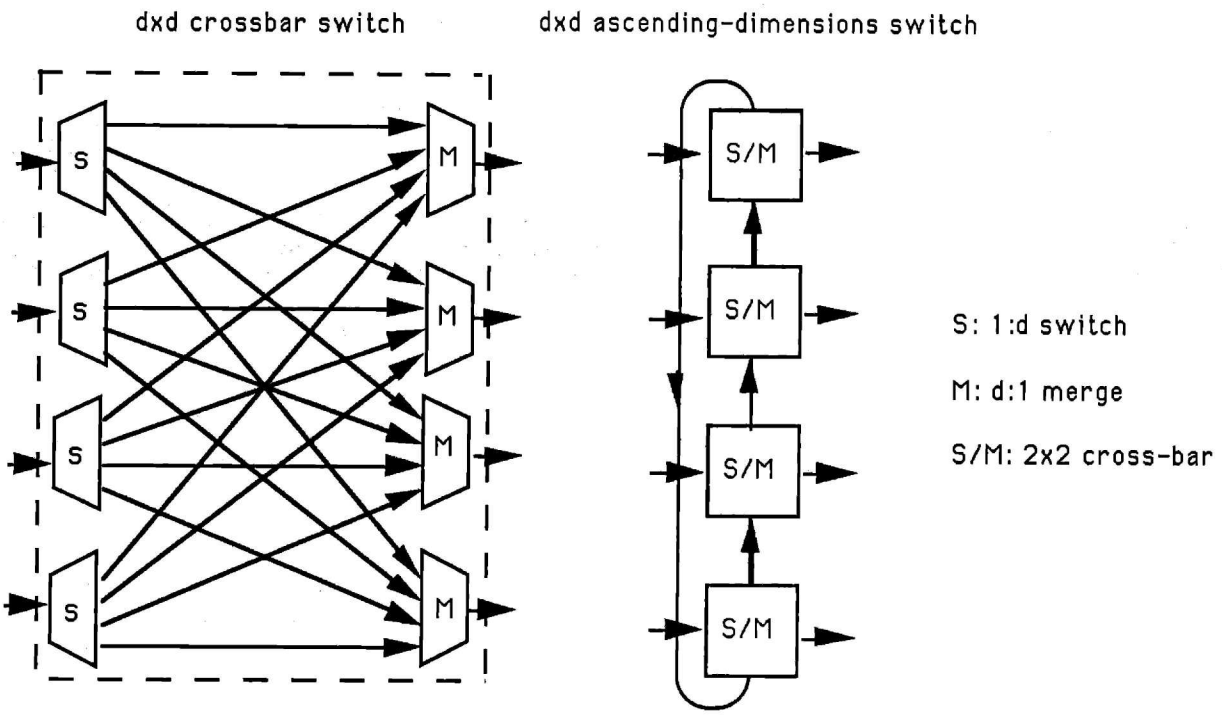

Figure 6.6: A $d \times d$ cross-bar switch, a $d \times d$ descending-dimensions switch, and the modules out of which they are composed.

If routing decisions are made deterministically then scenaria can be found where a live-lock persists forever. The problem could be overcome by having each packet count the number of hops it traverses, and be dropped when this number exceeds some limit. This is, however, undesirable, because it increases the processing time at the nodes, complicates the implementation, and makes the use of acknowledgements necessary. Of cource, if decisions are randomized then live-locks are eventually broken. However, we believe that it would be costly to randomize every switching decision, and therefore this should not be considered as a solution.

Wormhole routing is subject to a similar problem, called *deadlock* (see [Dal90b]). A deadlock arises whenever there is a cycle of packets requiring a resource that another packet in the cycle is employing. Deadlocks can happen quite frequently in wormhole routing when the buffer space is small. The solution that has been proposed in [DaS87] involves virtual channels and complicates the implementation.

Fairness

All the schemes examined, with the exception of the CSR protocol, can cause the system to operate unfairly. The first source that has access to the empty slots takes all the slots that it requires, while the source that follows takes what is left over. Figure 6.7 illustrates how a source can be locked out with the priority deflection scheme (even with randomized decisions). The CSR protocol is fair because if two nodes try to insert packets during a control interval, and are claiming the same link for the same slot, then the conflict is resolved randomly.

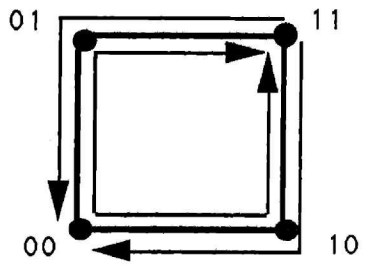

Figure 6.7: Node 00 continuously sends two packets per slot to node 11, and node 11 sends two packets per slot to node 00. Then if the priority deflection scheme is used, both nodes 01 and 10 cannot insert any packets. The situation where only some (not all) of the links of a node are blocked arises even more frequently.

Packet Resequencing

The CSR protocol can easily guarantee that packets arrive at their destination in sequence. On the other hand, the need for resequencing packets is inherent in deflection routing, and cannot be avoided. An implication of that is that resequencing buffers may overflow, dropping packets, and making the use of acknowledgements necessary. To avoid acknowledgements (which might impose a big burden on the network) it may be better to deflect packets that arrive at a destination node whose resequencing buffer is full. If the size of the resequencing buffers is small, congestion may increase significantly, because with deflection routing packets will often arrive out of sequence.

Processing at the Nodes

The simple and the priority shortest path schemes are the easiest to implement. The hardware required for these schemes is very simple (see Fig. 1 for the node model). Deflection routing requires more processing at each node, especially if we want to address the live-lock and the fairness problems. Also, a cross-bar switch is slower than an descending-dimensions switch. The priority deflection scheme requires slightly more processing time at the nodes than the simple deflection scheme. The CSR protocol can be implemented easily in a synchronous system. In the unbuffered case, the state of each link can be described by a binary number of length d (at any time reservations may exist for the next d slots at most), which should not be a problem.

Synchronization

The CSR scheme, the priority scheme, and the two deflection schemes are best suited for synchronous systems. The simple scheme and the wormhole routing scheme can also be implemented in an asynchronous system. Synchronous systems have a number of advantages which have resulted in an almost universal use (see [Dal90b]). Asynchronous systems are potentially faster (the slower component does not have to dominate the speed of the system), and avoid the problem of the distribution of the clock to all the chips with as small a clock-skew as possible, at the expense of a much more complicated implementation.

CHAPTER SEVEN

Routing Properties and Algorithms for some Hypercube Related Networks

In this chapter we examine some classes of networks, which are structurally related to the hypercube, and present routing algorithms to execute certain communication tasks in them. The network classes discussed are the pseudo-cube, the enhanced-cube, the permutation-cube, and the folded-cube. These networks, with the exception of the folded-cube, are first introduced here. The proposed networks have recursive properties and self-routing algorithms similar to those of the hypercube. Simulation results indicate that the mean internode distance for these networks is considerably less than that of a regular hypercube with the same number of nodes. For the permutation-cube we prove that any "reasonable" deterministic algorithm executes the total exchange task in less than N steps. For the folded-cube network we find the first strictly optimal multinode broadcast and total exchange communication algorithms. Finally we show that the layout of the proposed networks is simple, and requires area of the same order of magnitude with the regular hypercube.

7.1. NETWORK DEFINITIONS

The d-dimensional hypercube has $N = 2^d$ nodes, degree and diameter equal to d, and mean internode distance equal to $d/2$. Despite its nice recursive properties and the easy self-routing algorithms, which have made hypercubes popular, its diameter and mean internode distance are not particularly good. In a network of degree d, one could hope for a diameter and mean internode distance of the order of $\Theta(\log_d N)$ as pointed out in [Upf84]. In this

chapter we examine several networks of logarithmic degree, which are structurally related to the hypercube graph. These networks have recursive properties similar to those of a hypercube, but smaller diameter and mean internode distance. The interconnections of these networks are made according to easily describable rules, which is important for the feasibility of their construction. Simple self-routing algorithms can be employed in these networks, but better (although more complicated) routing algorithms also exist. The rules that define the networks leave many degrees of freedom; therefore, for each size, the proposed networks are actually classes of networks rather than individual networks. We will use the same name to refer to a network, and to the corresponding class of networks.

Each of the networks proposed has $N = 2^d$ nodes, where d is the *dimension* of the network. A node is represented by a distinct integer between 0 and $N-1$, or, equivalently, by the binary representation of this integer. A link connecting node s to node t is represented by $(s, s \oplus t)$, where $\oplus$ denotes the bitwise exclusive OR binary operation. When no confusion about the origin of a link can arise we denote the link between s and t by the binary string $l = s \oplus t$. We also denote by $L(s)$ the set of links of node s, and by e_i the binary number whose i^{th} bit is equal to 1 and the other bits are equal to 0. The first class of networks that we introduce is the following:

Definition 1: The *d-dimensional pseudocube* $\mathcal{S}_d$ is a class of networks, each member of which has $N = 2^d$ nodes. The set of links leaving each node consists of d links of the form $0^{d-i-1}1*^i$, $i = 0, 1, \ldots, d-1$, where each $*$ is either a 0 or a 1, and x^k, $(x \in \{0, 1, *\}, k \in \{0, 1, \ldots, d-1\})$, is the concatenation of k x's.

The link $0^{d-i-1}1*^i$ is called a *link of dimension i*. A d-dimensional pseudocube can be divided into two sub-pseudocubes. Each node in one of the sub-pseudocubes has a (unidirectional) link leading to a node in the other; this does not necessarily happen in a 1-1 way. The $*$'s can be different with each other not only when they appear in links of the same node, but also for links of different nodes. Thus, for $d = 3$, if $(s, 101)$ is an outgoing link of node s, it is not necessary that $(t, 101)$ is an outgoing link of node $t \neq s$. If this were the case then the network would be isomorphic to a hypercube, since instead of using the e_i's as a base of the space $\{0, 1\}^d$, another linearly independent base would be used. The outdegree of each node of the d-dimensional pseudocube is d, but the indegree is not the same for all nodes. If the $*$'s are chosen randomly, independently of each other and equal to 0 or 1 with probability 1/2, then the outdegree of each node is a random variable with mean d. The class $\mathcal{S}_d$ can also be viewed as a random

graph, where the $*$'s are chosen at random. The pseudocubes are not necessarily symmetric (although this can be enforced if desired), in the sense that when there is a link from node s to node t, there is not necessarily a link from t to s. It is easy to see that the pseudocube is a connected graph with diameter at most d.

The second class of networks that we will examine is the class of enhanced cubes, defined as follows:

Definition 2: The *d-dimensional enhanced-cube* $\mathcal{E}_d$ is a class of networks each member of which has 2^d nodes. The set of links leaving each node consists of $d+1$ links: d links are of the form e_i, $i = 0, 1, \ldots, d-1$, and one link is of the form $*^d$.

The enhanced-cube is a hypercube with one additional outgoing link per node leading to a random node. The outdegree of each node is equal to $d+1$. If the $*$'s are chosen equal to 0 or 1 with probability $1/2$, the indegree of a node is a random variable with mean $d+1$.

Definition 3: The *d-dimensional permutation-cube* $\mathcal{P}_d$ is the subclass of d-dimensional pseudocubes with the property that each node has exactly one incoming link per dimension.

A permutation-cube of dimension d can be divided into two sub-permutation-cubes, each of dimension $d-1$. The sub-permutation-cubes are not necessarily identical. For every node in one of the sub-permutation-cubes, there is a unidirectional link leading to a node in the other sub-permutation-cube, with no two of them leading to the same node.

Often it is desirable for a multiprocessor network to be symmetric, so that locality of communication can be achieved easier when assigning tasks to processors. For each of the classes of networks described above, we can define its symmetric subclass to consist of the networks with the property that whenever there is a link from node s to node t, there is also a link from node t to node s.

In all the previous networks, a self-routing algorithm similar to that of the hypercube exists. The algorithm for the enhanced-cube is obvious since it contains the hypercube as a subgraph. A self-routing algorithm for the pseudocube is obtained as follows. Consider a packet currently located at node s with final destination t. Let i be the most significant bit in which s and t differ. Then the next transmission of the packet is over the i-link of node s. Therefore, after the i^{th} step, $i = 1, 2, \ldots, d$, the binary representations of the current location and the final destination of a packet agree in at least the i most significan bits. This routing algorithm is based on the fact that a pseudocube of dimension of the packets m is divided into two (not

identical in general) sub-pseudocubes of dimension $m-1$, with each node in one of them having a link leading to the other. Thus, the routing of a packet to its destination is done in a *divide and conquer* way. Based on the same idea, one can devise many other networks which have such self-routing algorithms (e.g., a *k-dimensional pseudomesh* of p^k processors could be defined as having the property that it can be divided in p equal sub-pseudomesh, with each node in a sub-pseudomesh connected to a node in each one of two neighbor pseudomeshes). Note that in the self-routing algorithm, packets do not necessarily follow shortest paths to their destinations; however, they are guaranteed to arrive at their destination in less than d steps.

The previous three classes of networks are introduced for the first time here. The folded hypercube defined next was first introduced in [AdS82].

Definition 4: The *folded d-dimensional cube* has 2^d nodes. The $d+1$ links leaving each node are of the form e_i, $i = 0, 1, \ldots, d-1$, and 1^d.

A folded hypercube is a hypercube with additional links connecting each node s to its bitwise complement $\overline{s}$. The diameter of the folded hypercube is $\lceil d/2 \rceil$.

Note that the folded hypercube is a particular case of an enhanced-cube. Also the class of permutation-cubes is a subclass of the pseudocubes, and the hypercube itself is a special case of a permutation-cube.

In the remainder of the chapter we examine routing properties of the previously defined networks, and present algorithms to execute standard communication tasks in them. In particular, Section 7.2 examines the mean internode distance and other routing properties of the pseudocube and the enhanced-cube. Section 7.2 deals with routing properties of permutation-cubes, and proposes algorithms for the total exchange task. In Section 7.3 we propose the first optimal algorithms to execute a multinode broadcast and a total exchange in a folded-cube. Section 7.3 focuses on layout algorithms for the networks proposed.

7.2. MEAN INTERNODE DISTANCE AND OTHER PROPERTIES OF PSEUDOCUBES AND ENHANCED-CUBES

In this section we present simulation results for the mean internode distance of the pseudocube and the enhanced-cube classes of networks. We find that the mean internode distance is significantly less than $d/2$, the mean internode distance of a hypercube. We then calculate

what we call the mean "greedy routing distance" in the pseudocube for a simple routing scheme, which does not always use shortest paths. We also show that for any set of M permutations with $M < \frac{1}{2^d}\left(\frac{2k}{e}\right)^{kd}$, it is possible to find an enhanced-cube that routes (uniformly) all these permutations in $\Theta(\log N)$ time, using a greedy routing scheme.

We calculated by simulation the mean internode distance of the classes of pseudocube and enhanced-cube networks. In particular, we generated randomly some thousands of these networks, with all networks being equally probable, and calculated the mean internode distance for each of them. Fig. 7.1 illustrates the mean internode distance of the class $\mathcal{S}_d$ of pseudocube networks for dimensions up to $d = 20$. The mean is taken over all pairs of nodes and over all network members of $\mathcal{S}_d$. There are pseudocubes, which have mean internode distance less than what is indicated in Fig. 7.1, because there are also pseudocubes which have internode distance greater than the mean (an example being the regular hypercube). For the network sizes that we simulated the mean internode distance in $\mathcal{S}_d$ was close to $d/4 + 0.9$. This is significantly less the mean internode distance $d/2$ of the hypercube, and is attained at no additional hardware complexity compared to the hypercube (see Section 7.5 for the layouts of the networks proposed).

In Fig. 7.2 we indicate the mean internode distance for the enhanced-cube class of networks. The improvement over the hypercube is again significant. Note that the mean internode distance of $\mathcal{S}_d$ is significantly smaller than that of $\mathcal{E}_d$, although the networks in $\mathcal{E}_d$ have outdegree $d + 1$ as opposed to d for the networks in $\mathcal{S}_d$.

The mean internode distance shown in Fig. 7.2 is obtained when packets are routed through shortest paths in an enhanced-cube. Evaluating and storing the shortest paths for every pair of nodes may be undesirable if the nodes do not communicate often, or if they exchange small messages. A greedy routing scheme with small storage and processing overhead is the following. A node sends the packet to the one of its neighbors (including the one connected via the "extra" link), which is at smaller Hamming distance (not necessarily smaller real distance) from its destination. The mean *greedy routing* distance obtained in this way (the mean is taken over all networks and all pairs of nodes) is indicated in Fig. 7.3 for various network sizes. The results in this figure were obtained by simulating a Markov chain formulated for this routing scheme.

An important advantage of the enhanced-cubes can be seen by applying results of [Val82] and [VaB81]. First, any particular permutation can be executed in time $\Theta(\log_2 N)$ for the majority of the enhanced-cubes. This can be done by first sending the packet over the extra

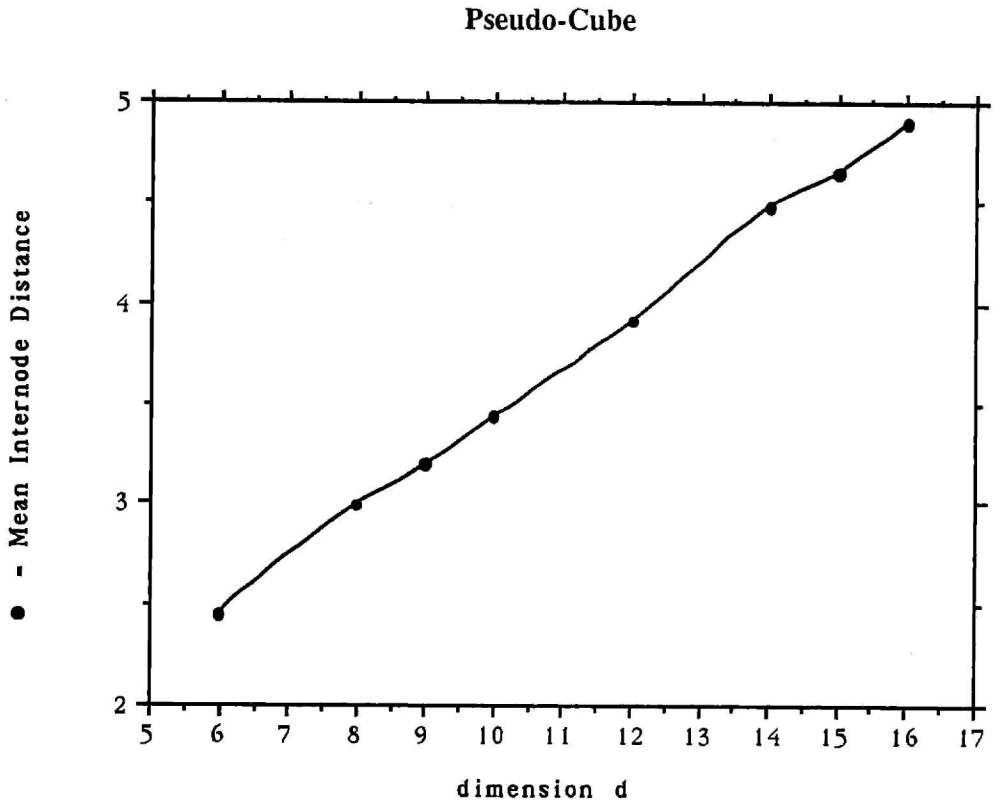

Figure 7.1: Mean internode distance of S_d as a function of d.

link (view it as a random link), and then routing the packet to its destination in the greedy way without using any extra links. This routing scheme consists of only one phase (instead of two in Valiant's work), and thus the constant in the Θ expression is half of that of [Val82]. A second observation is that worst case permutations for the regular hypercube are not worst case permutations for the big majority of the enhanced-cubes. This is important because the worst case permutations of the hypercube are very regular (because of the regularity of the hypercube), and arise often in applications. An example is the bit reversal permutation, where node $s_{d-1} \cdots s_1 s_0$ sends a packet to node $s_0 s_1 \cdots s_{d-1}$. This permutation requires $\Theta(\sqrt{N})$ steps in a hypercube with the greedy routing, while it is performed in $\Theta(\log N)$ steps for most enhanced cubes. The class of enhanced-cubes is "uniformly good" for all permutations (Valiant [Val82] used the relevant notion of "testability"), while the hypercube often performs badly for well structured permutations. A third, and more interesting, observation is given by the next

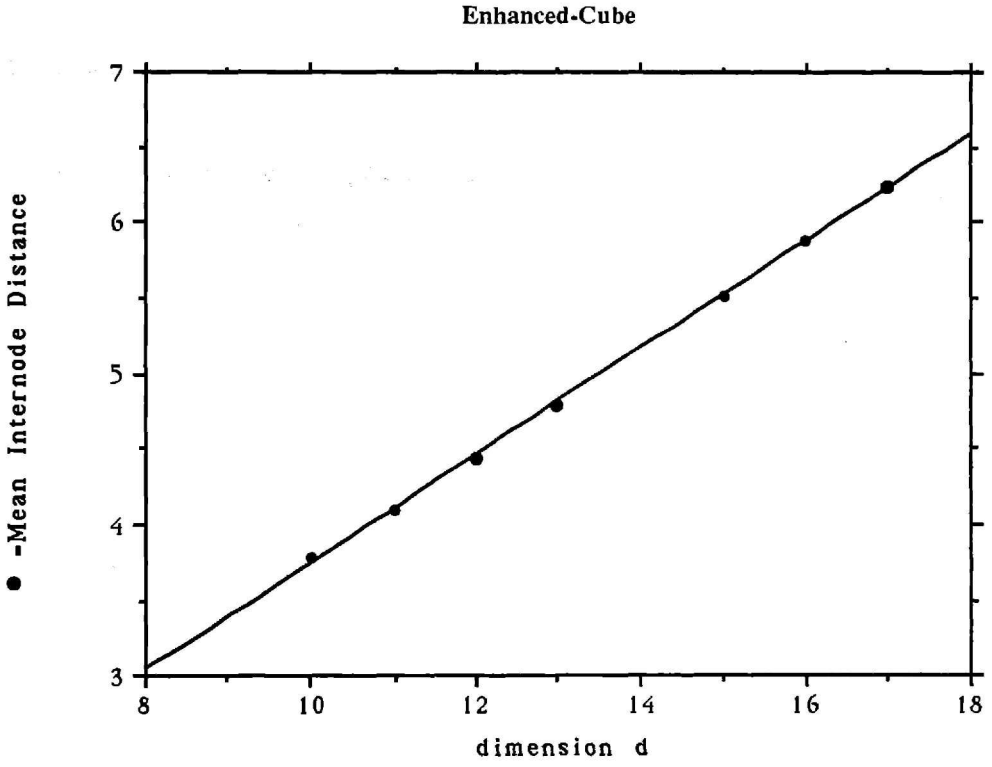

Figure 7.2: Mean internode distance of $\mathcal{E}_d$ as a function of d.

corollary to Valiant's results;

Corollary 1: Given any constant $K > 2.5$, and any set of M permutations, with

$$M < \frac{1}{2^d}\left(\frac{2K}{e}\right)^{Kd}, \tag{7.1}$$

there exists a network belonging to the class of enhanced-cubes, which executes in a greedy way *all* these permutations in less than $\frac{K+1}{2}d + 1$ steps.

The greedy routing scheme mentioned in the corollary is the one where a packet is sent at the first slot over the extra link of its origin, and then crosses the hypercube dimensions in some (particular) order. Thus, if the designer of the cube is given M special permutations [with M satisfying Eq. (7.1)], which have to be implemented efficiently, he can find an enhanced-cube that *always* (and not with high probability) executes any of these special permutations in $\Theta(d)$ steps using a greedy algorithm. Since M is growing exponentially with d, and even more

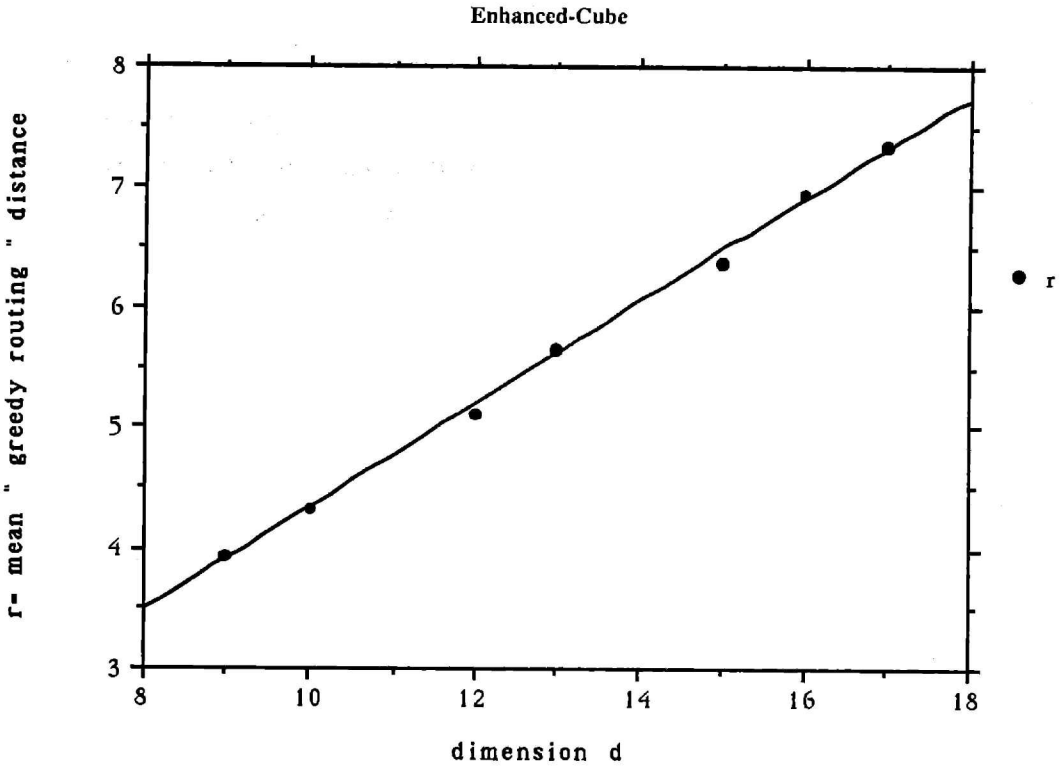

Figure 7.3: Mean "greedy routing" distance of the randomly enhanced cubes as a function of its dimension d.

rapidly with K, the previous corollary leaves space for practical applications. A way to design a "good" network is to generate randomly several enhanced-cubes, and test them for the M special permutations in which we are interested. Then chances are that after a few trials we will find a network that performs these permutations in time $\Theta(\log N)$. The testing is used just in case we are unlucky in the generation of the graph. Researchers have been trying to find which networks execute well which permutations. The class of enhanced-cubes has "good" networks, no matter which M permutations we consider. The practical issue of actually laying on silicon such a network is discussed at the last section, where it is found that enhanced-cubes require layout area $\Theta(N^2)$, which is of the same order of magnitude with that of a regular hypercube.

An interesting open problem is the following. Does there exist a network belonging to the class of enhanced-cubes that can route *all* permutations in time $\Theta(\log N)$ by employing a greedy routing scheme? We want the algorithm to be "on-line", deterministic, and adaptive (a non-

adaptive, or *oblivious*, routing scheme would not do in view of the results in [BoH85]). If we use only the regular hypercube links at most $p(N!)$ permutations require time $\Omega(\log N)$ to be executed, where p is decreasing exponentially in d. We call the set of such "bad" permutations S_Ω. If we first use the extra link, again at least $(1-p)(N!)$ permutations are routed in time $O(\log N)$. Call the set of these "good" permutations S_O. If there exists an enhanced-cube such that S_Ω is a subset of S_O, then this network can execute all permutations in $O(\log N)$ time. Intuition suggests that similar arguments must apply to pseudocube and permutation-cube networks.

7.3. COMMUNICATION ALGORITHMS FOR PERMUTATION-CUBES

In this section we consider the permutation-cube class of networks. Since the permutation-cubes form a subclass of the pseudocubes the self-routing algorithms described earlier apply to them as well. We used simulations to calculate the mean internode distance of a symmetric permutation-cube (see Fig. 7.4). The results indicate that the permutation cubes have on the average significantly less mean internode distance than regular hypercubes of the same size (there is a 40% improvement for dimension $d = 12$). In what follows, we will show that generic communication tasks, like the ones described in Section 1.3, can also be executed efficiently in permutation-cubes. We will present a class of simple-minded algorithms, called "reasonable algorithms", which perform the total exchange task in less than N steps. The same algorithm can be used to execute the total exchange for the whole class of permutation-cube networks, which is remarkable. We recall here that the optimal algorithm for the TE task in a regular hypercube requires $N/2$ steps (see Section 2.4), and achieves 100% utilization of the links.

To find a TE algorithm, we restrict our attention to paths in which the dimensions are traversed in descending order, that is, links in dimension i with $i > j$ are traversed before links of dimensions j. In particular, a packet currently residing at node s and having node t as its destination is sent over the i-link, where i is the most significant non-zero bit in the binary number $s \oplus t$. We call such paths *normal paths*. A normal path is not necessarily a shortest path; however, it is always of length less than or equal to d, and, if we assume that nodes s and t are randomly chosen with equal probability over all nodes, its mean length is $d/2$. We define the *normal relative address* between a source node s and a destination node t as the

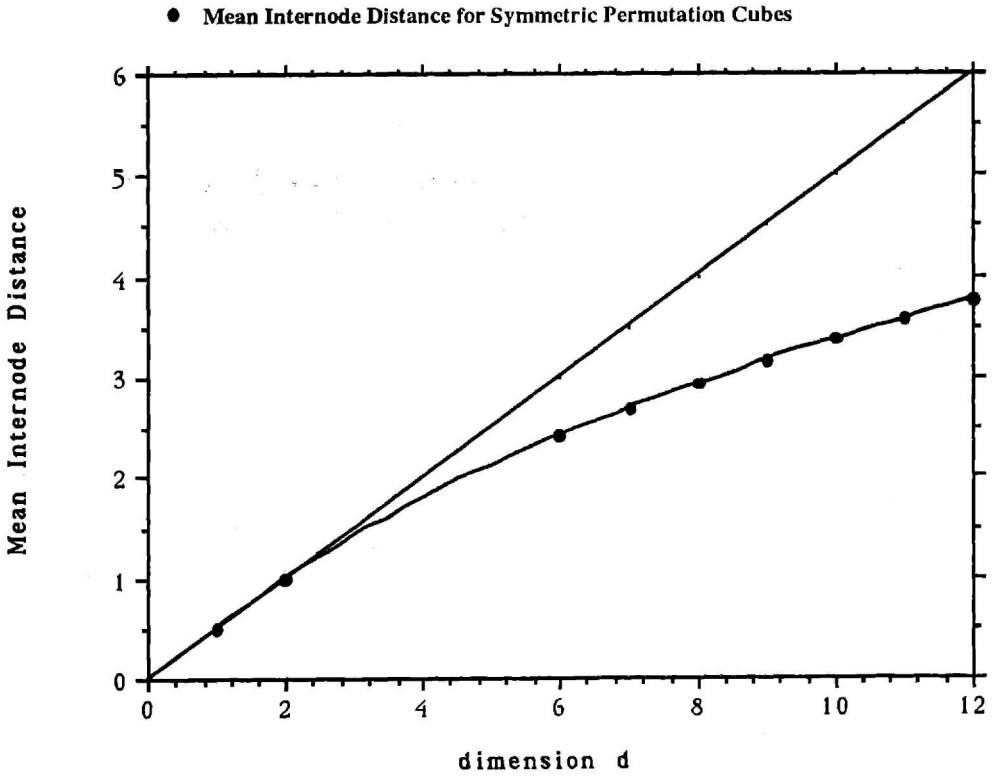

Figure 7.4: Mean internode distance of symmetric permutation cubes as a function of their dimension d. The straight line corresponds to the mean internode distance of the hypercubes.

binary number of length d whose i^{th} bit is a one if an i-link is going to be used by the normal path from s to t, and a zero otherwise. The relative address from node s to node t is not in general equal to $s \oplus t$, as is the case for the regular hypercube. In fact, it is not even equal to the relative address from node t to node s.

Suppose now that we are given a communication task where each node s has to send packets to all nodes in a set $\mathcal{S}$. We define the *task matrix* $T(s)$ of that task at node s as the binary matrix that has as rows the normal relative addresses of node s with respect to all nodes in $\mathcal{S}$. Then we have the following lemma.

Lemma 1: For the TE task the task matrices are the same for all nodes, and contain all the binary numbers between 0 and $N-1$.

Proof: Each normal relative address corresponds to a path, and paths corresponding to two

different relative addresses lead to different nodes (if i is the first bit in which the relative addresses differ then the two nodes will always differ in this bit). Thus, there is a 1-1 correspondence between the binary numbers of length d and the relative addresses of the permutation-cube nodes with respect to a particular node. **Q.E.D.**

Using the ideas developed in Chapter 2 we can show that the execution of the TE task is equivalent to clearing the task matrix according to the following rules:

- at most one entry from each row or column may be cleared at each step, and
- entries within a row are cleared from higher dimensions to lower dimensions; in other words, an entry of the task matrix can be cleared only after all the entries of the same row that correspond to higher dimensions have been cleared first.

The second rule is posed so that the paths followed are normal, in which case the relative address of some node s with respect to node t corresponds to an actual path from s to t. We are currently examining if the task matrix for the TE can be cleared in time $N/2$ (for all the examples we have tried this was possible, but we have not found any systematic way yet). We can prove, however, the following result.

Lemma 2: Any "reasonable" routing scheme can perform the total exchange (or equivalently clear the task matrix entries) in time at most N. By the term "reasonable" we mean that if there is a packet at node s and the most significant nonzero bit of its relative address is the i^{th}, then the i-link of node s will not stay idle during the next slot.

Note: The routing does not have to be symmetric (see Subsection 2.3) for the result to hold.

Proof: After time $N/2$ at least the bits of the task matrices that correspond to dimension d will have been cleared (some more transmissions may have also taken place). If only this (minimum) amount of work is done the rest of the task is equivalent to two concurrent TEs in different subcubes, each of dimension $d-1$. Thus

$$T_{TE}(N) \leq T_{TE}(N/2) + N/2 \leq N.$$

(The preceding proof can be made more rigorous. If we impose a FCFS discipline then rigor is easy to achieve, but no such queueing discipline is actually needed for the result to hold).

7.4. OPTIMAL COMMUNICATION ALGORITHMS FOR FOLDED-CUBES

In this section we present communication algorithms to execute a multinode broadcast and a total exchange in an d-dimensional folded-cube. The algorithms to be presented have strictly optimal time complexity. Ho [Ho90] proposed an optimal algorithm for the MNB, and a sub-optimal algorithm for the TE in the folded-cube. The improvement in the time complexity achieved by our TE algorithm is very significant. Also, we believe that the MNB which we give is simpler than the one in [Ho90].

In what follows we will refer to a link connecting node s to node $s \oplus e_i$, $i = 0, 1, \ldots, d-1$ as an i-link. We will also refer to the link connecting node s to its bitwise complement $\overline{s}$ as a c-link. Note that any d of the binary strings $e_0, e_1, \ldots e_{d-1}, 1^d$ are linearly independent and form a base of the space $\{0,1\}^d$. Therefore, a packet can be routed from any source to any destination by using only d of the $d+1$ dimensions of the folded-cube.

The algorithm for the MNB is based on the ideas developed in Chapter 3 , and will only be outlined here. We define the algorithms $\mathcal{A}_j$ in the way we defined the corresponding algorithms for hypercubes in Chapter 3 (Theorem 6), with the difference that c-links can be used. Each of these algorithms uses only d out of the $d+1$ available dimensions. The order in which the dimensions are traversed may be $< 0, 1, \ldots, d-1 >$, $< 1, 2, \ldots, d-1, c >$, $< 2, 3, \ldots, d-2, c, 0 >$, ..., or $< c, 0, 1, \ldots, d-2 >$. By splitting the packets into $d+1$ parts and performing $d+1$ disjoint MNBs we find that the time required to execute a MNB in a folded cube is

$$\frac{N-1}{d+1}.$$

This is optimal, since each node has $d+1$ links and has to receive $N-1$ packets.

For the TE communication task, Ho proved the following lower bound for the time T_{TE} required to execute the task:

$$T_{TE} \geq 2^{d-1} - \frac{1}{2}\binom{d}{\lceil d/2 \rceil}.$$

He also gave an algorithm with time complexity

$$\frac{dN}{2(d+1)}.$$

His algorithm requires each packet to be split in $d+1$ parts. The preceding upper and lower bounds are of the same order of magnitude. However, the gap between the two is quite important in practice. For example, in the case $d = 16$ the lower bound is 26333 time steps, while

algorithm in [Ho90] executes the task in 30840.47059 steps. This is only slightly better than the 32768 steps required for a TE in a regular 16-dimensional hypercube, which indicates that the algorithm does not take full advantage of the extra c-links. In the remaining section we give an algorithm with time complexity exactly equal to the lower bound. An additional advantage of our TE algorithm is that it does not require the splitting of packets as Ho's algorithm does.

Assume first that d is even. According to the algorithm we propose, a packet originated at node s and destined for a node t, which is at Hamming distance less or equal to $d/2$ from s, does not use any c-links and its routing tag is equal to $0(s \oplus t)$. A packet originated at node s and destined for a node t, which is at Hamming distance greater than or equal to $d/2+1$ from s, will use a c-link and its routing tag is equal to $1(\overline{s \oplus t})$, where the bar represents the bitwise complement. We define task matrices to have $d+1$ columns (one for each link) and $N-1$ rows (one for each packet). The task matrices are the same for all nodes. The critical sum of the task matrix is equal to

$$\max(T_c, T_e),$$

where

$$T_c = \sum_{i=d/2+1}^{d} \binom{d}{i} = \frac{N}{2} - \frac{1}{2}\binom{d}{d/2}$$

is the column sum of the task matrix that corresponds to the c-link, and

$$T_e = \frac{1}{d}\sum_{i=0}^{d/2} \binom{d}{i} i + \frac{1}{d}\sum_{i=0}^{d/2-1} \binom{d}{i} i = \sum_{i=1}^{d/2} \binom{d-1}{i-1} + \sum_{i=1}^{d/2-1} \binom{d-1}{i-1} =$$

$$= \sum_{j=0}^{d/2-1} \binom{d-1}{j} + \sum_{j=0}^{d/2-2} \binom{d-1}{j} = \frac{N}{4} + \frac{N}{4} - \binom{d-1}{d/2-1} = 2^{d-1} - \frac{1}{2}\binom{d}{\lceil d/2 \rceil},$$

is the column sum that corresponds to a regular link (that is, a link which is also present in the regular hypercube). Using the results in Chapter 2 we find that for d even we have

$$T_{TE} = T_c = T_e = 2^{d-1} - \frac{1}{2}\binom{d}{\lceil d/2 \rceil},$$

which equals the lower bound.

For d odd, now, a packet originated at node s and destined for a node t at Hamming distance less or equal to $\lceil d/2 \rceil - 1$ from s will not use any c-links, and has routing tag equal to $s \oplus t$. A packet originated at node s and destined for a node t at Hamming distance greater than or equal to $\lceil d/2 \rceil + 1$ from s will use a c-link and has routing tag equal to $1(\overline{s \oplus t})$. The packets that are destined for nodes at Hamming distance $\lceil d/2 \rceil$ from s are split into two sets of equal

cardinality, in the same way for all origins s; packets in the one set will use c-links and packets in the other set will not. The task matrices will again be the same for all nodes and have critical sum equal to

$$\max(T_c, T_e),$$

where

$$T_c = \sum_{i=\lceil d/2\rceil+1}^{d} \binom{d}{i} + \frac{1}{2}\binom{d}{\lceil d/2\rceil} = \frac{N}{2} - \frac{1}{2}\binom{d}{\lceil d/2\rceil}$$

is the column sum corresponding to the c-link, and

$$T_e = \frac{2}{d}\sum_{i=0}^{\lceil d/2\rceil-1}\binom{d}{i}i + \frac{1}{2d}\binom{d}{\lceil d/2\rceil} = 2\sum_{i=1}^{\lceil d/2\rceil-1}\binom{d-1}{i-1} + \frac{1}{2d}\binom{d}{\lceil d/2\rceil} =$$

$$= 2\sum_{j=1}^{\frac{d-1}{2}-1}\binom{d-1}{j} + \frac{1}{2d}\binom{d}{\lceil d/2\rceil} = \frac{N}{2} - \binom{d-1}{\lceil d/2\rceil - 1} + \frac{1}{2d}\binom{d}{\lceil d/2\rceil} = 2^{d-1} - \frac{1}{2}\binom{d}{\lceil d/2\rceil},$$

is the column sum that corresponds to a regular link. Thus for d odd, the algorithm executes the total exchange in time

$$T_{TE} = T_c = T_e = 2^{d-1} - \frac{1}{2}\binom{d}{\lceil d/2\rceil},$$

which again equals the lower bound. Note that in the algorithm we propose packets follow shortest paths to their destination and 100% utilization of all the links is achieved.

Based on the analysis in [VaB90a] one can also derive algorithms that minimize both the average delay and the completion time of the task, and define "reasonable routing schemes" that execute the TE in time less or equal to

$$2^{d-1} - \frac{1}{2}\binom{d}{\lceil d/2\rceil} + d - 1.$$

This is very close to the optimal complexity.

7.5. LAYOUT ALGORITHMS FOR THE NETWORKS PROPOSED

In this section we will present layout algorithms for the networks considered in Chapter 7. All these networks can be laid out as squares of area $\Theta(N^2)$, that is their layout area is of the same order of magnitude with that of the hypercube.

Folded-Cubes

We first lay out a hypercube as a square of area $\Theta(N^2)$ (see [Ull84], p. 129). It remains to introduce links between each node and its complement. We create channels for each of these pairs as shown in [Ull84] (p. 97). We have to create at most $4N$ (unidirectional) horizontal channels and $4N$ vertical channels. Thus the folded hypercube can be laid out as a square of area at most $\Theta(N^2) + 16N^2 = \Theta(N^2)$.

Permutation-Cubes

They can be laid out recursively, in exactly the same way with the hypercubes, since they have the same node degree and the same separators (see [Ull84]). Thus the area required is $\Theta(N^2)$.

Pseudocubes

The main problem here is the unbounded node degree, and especially the indegree of each node, which can be as large as N (although the mean over all nodes and over all pseudocubes of the indegree is equal to d). The Thompson grid layout model which we are using in this section does not allow degree more than 4 (the hypercube as well as any other unbounded degree network also need some modification of the Thompson model). We follow the approach typically taken in such cases. We lay out each node as a tree with number of nodes equal to its degree. Thus area $O(maxdegree)$ is required for each node, where $maxdegree$ is the maximum indegree of a node of the pseudocube. We now prove the folowing proposition.

Proposition 1: Every d-dimensional pseudocube can be laid out as a square of side at most $L(N, maxdegree) = 8N + \Theta(\sqrt{maxdegree})\sqrt{N}$, where $maxdegree$ is the maximum indegree and $N = 2^d$ is the number of nodes.

Note: Since $maxdegree < N$ we get that the side of the square is $\Theta(N)$ and the area $\Theta(N^2)$.

Proof: The layout algorithm will be of the kind typically used for networks having suitable separators (see [Ull84]). We assume that N is a power of 4. This is not a real restriction and in any case we can intoduce dummy nodes without affecting the bisection width of the network (which is what counts as we will see) with the area being affected by a factor of at most 4. We

replace each node by a binary tree of $d + maxdegree = \Theta(maxdegree)$ nodes (each leaf is a port of the node). Then each node can be laid out as a square of side $\Theta(\sqrt{maxdegree})$.

For $N = 1$ the proposition holds. Assume now that we can lay out a pseudocube with $N/4$ nodes and degree d for each node in a square of side $L(N/4, maxdegree)$ (we keep the degree equal to *maxdegree* throughout the induction so that we do not have to add more ports later). We place on the plane four such squares as shown in Fig. 7.5. Each of them represents the layout of one of the subcubes $H_{00} = 00*$, $H_{01} = 01*$, $H_{10} = 10*$, $H_{11} = 11*$. In order to obtain a layout for the d-dimensional pseudocube we just have to add the (unidirectional) links connecting these subcubes. Let $\Gamma(H_i, H_j)$ be the number of links connecting some node in subcube H_i to some node in H_j. Then one can check that

$$\Gamma(H_{00}, H_{01}) + \Gamma(H_{00}, H_{11}) = \Gamma(H_{10}, H_{00}) + \Gamma(H_{10}, H_{01}) =$$

$$= \Gamma(H_{11}, H_{00}) + \Gamma(H_{11}, H_{01}) = \Gamma(H_{01}, H_{10}) + \Gamma(H_{01}, H_{11}) = N/4,$$

and

$$\Gamma(H_{00}, H_{01}) = \Gamma(H_{01}, H_{00}) = \Gamma(H_{10}, H_{11}) = \Gamma(H_{11}, H_{10}) = N/4.$$

Thus, a total of $\sum_{i \neq j} \Gamma(H_i, H_j) = 2N$ unidirectional links have to be created, which requires the creation of at most $4N$ horizontal and $4N$ vertical channels (the ports at the nodes are already available). Therefore,

$$L(N, d) = 2L(N/4, maxdegree) + 4N,$$

which together with $L(1, maxdegree) = \Theta(\sqrt{maxdegree})$ proves that the d-dimensional pseudocube can be layed out as a square of side

$$L(N, d) = 8N + \Theta(\sqrt{d})\sqrt{N} = \Theta(N).$$

Q.E.D.

Enhanced-Cubes

Again the layout area required is $\Theta(N^2)$. One first lays out a hypercube with each node having degree *maxdegree* (instead of d), where *maxdegree* is the maximum degree of the enhanced-cube. This is done in the way described for the case of a pseudocube. Then adding the extra links increases the area by at most $\Theta(N^2)$ as we explained for the case of the folded-cube.

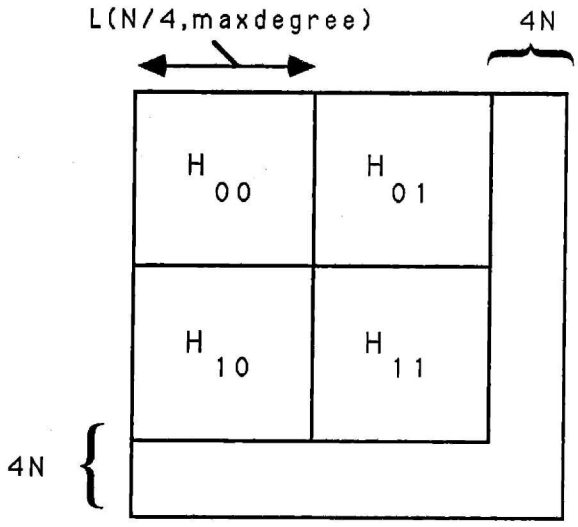

Figure 7.5: Recursive layout of a pseudocube.

7.6. CONCLUSIONS

The networks examined have routing properties similar, and sometimes superior to those of the regular hypercube. For example, the multinode broadcast and total exchange algorithms can be executed faster in a folded-cube than in a hypercube. The mean internode distance for all the networks examined was considerably less than $d/2$. The networks proposed (with the exception of the folded-cube) have the additional advantage that they tend to treat in a uniform way all permutation tasks. They have recursive structure similar to the one of the hypercube and require the same layout area. Easy recursive algorithms to layout these networks have been proposed. The previous reasons indicate, we believe, that the proposed networks are not only interesting from a theoretical point of view, but they are also viable VLSI architectures.

CHAPTER EIGHT

Conclusions - Directions for Future Research

The thesis has dealt with a number of problems concerning communication aspects of parallel computation. In this chapter we outline our contribution and discuss some problems that we consider interesting for future research.

8.1. IMPORTANCE OF THE PROBLEMS

Massively parallel computers is essentially a different way to organize and access the memory of a computer. The critical resources in a multiprocessor computer are the communication bandwidth and the memory capacity. The importance of the communication bandwidth and the memory capacity is underscored by the fact that they occupy more than 90% of the area of a parallel computer (see [Dal90b], [Joh90]). Since all the components of a parallel computer (memory, processors, routers) are made with the same technology, the cost is proportional to the area. Therefore, the processors are relatively cheap, and there is no great penalty for processors being idle. The true waste occurs when the communication bandwidth and the memory are not efficiently used.

The idea behind parallel processing is to reduce the Von Neumann bottleneck that exists whenever there is a single processor that has to fetch data from a memory in a sequential way. By putting processors near the memory we are able to access it in parallel. Our ability to access non-local memory locations fast is determined by the speed of the communication network. Thus, the communication efficiency is the key to the broad success of massively parallel computation. In order to use massively parallel computation for a broad range of applications, communication problems like the ones considered in this thesis have to be addressed.

8.2. CONTRIBUTIONS

In this section we outline our contribution, and discuss a number of open problems and directions to pursue, which are related to the content of our thesis. We also discuss some potential research directions of a more general nature, which are relatively independent of the work presented in this thesis, but concern ideas which arose during the course of the work.

In Chapter 2 we introduced the class of isotropic tasks. The theory developed there deals in a unified way with this new and broad class of communication tasks, which can be defined in a number of regular topologies (in addition to the hypercubes and the folded-cubes, considered in Chapters 2 and 7, respectively, the results have been extended to d-dimensional meshes with wraparound; see [VaB90a]). For all isotropic tasks and a variety of regular topologies, this theory gives simple, optimal routing algorithms. The rules found for the isotropic tasks give rise to a number of optimal algorithms for the total exchange task, which is a special case of an isotropic task. We also considered the problem of storing by columns, and transposing a banded matrix in a hypercube. We proposed an assignment of matrix columns to hypercube processors, and a transposition algorithm for a single banded matrix, or multiple banded matrices. The algorithm is faster by a logarithmic factor than the fastest previous algorithm. We also proposed a way to embed arbitrary sparse graphs, and presented a way to store block diagonal matrices in hypercubes.

Although we proved that the banded transposition algorithm is of optimal order when the bandwidth B is $\Theta(N^c)$ for some constant $c \in [0, 1]$, we have not proved optimality for any value of B. We also believe that it would be interesting to identify other "almost isotropic" tasks so that we can take advantage of the simple and elegant analysis of the isotropic tasks. The tools

developed for isotropic tasks can be used for hypercubes, d-dimensional meshes, Manhattan networks, and folded-cubes, and possibly other regular topologies. It would be interesting to define the most general class of networks where the tools for isotropic tasks can be used.

In Chapter 3 we gave several algorithms to execute the partial multinode broadcast and the partial exchange tasks in hypercubes and d-dimensional meshes. Near-optimal algorithms, or algorithms of optimal order were found for both tasks and both topologies, under two different communication models. We also presented results on the simulation of meshes with wraparound by meshes without wraparound, on the packing and isotone routing problems for d-dimensional meshes, on the window multinode broadcast in hypercubes, and on other routing problems. The partial multinode broadcast for hypercubes improves by a factor of roughly two previous results. The other problems considered in Chapter 3 were introduced for the first time.

Since the PMNB task arises often in applications, and it is a critical component of the dynamic broadcasting scheme of Chapter 4, it would be interesting to consider it for other topologies of interest. We believe that efficient PMNB algorithms can be found for folded-cubes, and meshes of trees by using some of the ideas of the corresponding algorithms for hypercubes and meshes (what is important for the PMNB algorithms are the recursive properties of a network). Also, the Window Total Exchange problem, defined but not solved in Section 3.8 deserves some attention.

In Chapter 4 we consider the problem where broadcast requests are generated at each node of a network stochastically. We proposed a simple scheme to execute the broadcasts, based on reservation ideas. The scheme can be used in hypercubes, d-dimensional meshes and tori, and, in general, in any network for which PMNB algorithms with certain properties can be found. For the hypercube and the d-dimensional array or torus our scheme had asymptotically optimal stability region, and asymptotically optimal average delay in terms of the size of the network for any fixed load in the stability region.

A conclusion of our analysis is that static routing algorithms when properly adapted can yield efficient algorithms for dynamic routing problems. This suggests that there is a connection between static and dynamic problems. Algorithms developed for static tasks can form a basis for effective routing algorithms in dynamic, stochastic environments. It might be fruitful to examine what the existence of efficient solutions to a static problem means for the corresponding dynamic problem.

In Chapter 5 we dealt with the problem where packets having a single destination are

generated at each node of a hypercube in a stochastic way. We proposed two different routing schemes, and evaluated their steady state throughput. The schemes are easy to implement, and require simple, low-cost switches at the hypercube nodes. The priority rule used in one of the schemes increases the throughput significantly and can be easily implemented in a parallel computer. We also examined the effect of the buffer space on the throughput. The results on the throughput were approximate, but very accurate as simulation results indicate, and they were given in interesting forms. We also analyzed through simulations the performance of the simple and the priority deflection schemes. These schemes, which use crossbar switches at the nodes, were found to have very satisfactory throughput. The priority deflection scheme is conjectured to have throughput asymptotically equal to the maximum possible. A challenging problem is to analyze without approximations the simple and the priority scheme, and the two deflection schemes. The fact that no progress has been made over the years in analyzing exactly the simple deflection scheme shows that these problems are not easy.

In Chapter 6 we proposed a new switching format, which combines most of the individual advantages of packet and circuit switching. The CSR protocol solves very efficiently data link control issues, such as the link and buffer space allocation strategy, the feedback mechanism, the retransmission protocol, and others. We also presented a hypercube implementation of the CSR and analyzed it approximately. The CSR protocol provides satisfactory throughput, when the corresponding parameter β is small, together with a number of other advantages.

An interesting question is to analyze a hypercube implementation of the CSR protocol where the hypercube nodes are crossbar switches, allowing more freedom in the switching assignments permitted. We expect the throughput of such a hypercube CSR implementation to be better than the throughput of any other hypercube scheme (for small enough β). Another open question is the analysis of the hypercube implementation of Sections 6.3-6.4 without the use of approximations. It would also be interesting (and perhaps easier) to analyze the CSR protocol for a 2-dimensional wraparound mesh or a Manhattan network where greedy shortest path routing algorithms are used.

In Chapter 7 we defined several new classes of network topologies Simulation results indicated that the mean internode distance of these networks was very satisfactory. The networks proposed have nice recursive properties, and self-routing algorithms. We proved that the total exchange task can be executed efficiently by greedy algorithms in some of the proposed networks. We also presented the first strictly optimal MNB and TE algorithms for folded cubes. We find that all the proposed networks when placed on silicon require area $O(N^2)$, which is of

the same order of magnitude with the hypercube.

A main direction for future work is the implementation of some of the proposed static and dynamic communication algorithms in actual parallel machines. The efficiency of many of these algorithms together with their simplicity makes them, we believe, potentially interesting from a practical point of view. Furthermore, the CSR protocol seems an interesting alternative to packet or circuit switching for muliprocessor networks; it remains to be seen how efficiently it can be implemented in practice.

As we move towards fine-grained parallelization, communication problems will increase in importance. Existing parallel machines have communication and synchronization overhead in excess of 500 instruction times ([Dal90b]). Truly fine-grained parallelization will require this overhead to drop to 10-20 instruction times. We hope that the results described in this thesis will help in this transition.

REFERENCES

[AdS82] Adams, G. B., and Siegel, H. G., "The Extra Stage Cube: a Fault-Tolerant Interconnection Network for Supersystems," IEEE Trans. Computers, 31(5), pp. 443-454, May 1982.

[BCW81] Bongiovanni, G., Coppersmith, D., and Wong, C. W., "An Optimum Time Slot Assignment Algorithm for an SS/TDMA system with Variable Number of Transponders", IEEE Trans. Commun., Vol. COM-29, pp. 721-726, 1981.

[Bra91] Brassil, J. T., *Deflection Routing in Certain Regular Networks*, Ph.D. Thesis, UCSD, 1991.

[Ber91] Bertsekas, D. P., *Linear Network Optimization: Algorithms and Codes*, M.I.T. Press, Cambridge, MA., 1991.

[BeT89] Bertsekas, D. P., and Tsitsiklis, J. N., *Parallel and Distributed Computation: Numerical Methods*, Prentice-Hall, Englewood Cliffs, N.J., 1989.

[BeG87] Bertsekas, D. P., and Gallager R., *Data Networks*, Prentice-Hall, 1987.

[BeG91] Bertsekas, D. P., and Gallager R., personal communication.

[BOS91] Bertsekas, D. P., Ozveren, C., Stamoulis, G. D., Tseng, P., and Tsitsiklis, J. N., "Optimal Communication Algorithms for Hypercubes," J. Parallel Distrib. Comput., Vol. 11, pp. 263-275, 1991.

[BhI85] Bhatt, S. N., and Ipsen, I. C. F., "How to Embed Trees in Hypercubes," Yale University, Dept. of Computer Science, Research Report YALEU/DCS/RR-443, 1985.

[Ble86] Blelloch, G. ,E., "Scans as Primitive Prallel Operations," Proc. Int'l Conf. Parallel Processing, pp. 355-362, August 1986.

[BoH85] Borodin, A., and Hopcroft J. E., "Routing Merging and Sorting on Parallel Models of Computation," J. Comput. Syst. Sci., 30:130-145, 1985.

[ChL89] Choudhury A., and Li, V. O. K., "Performance Analysis of Deflection Routing in the Manhattan Street and Minimum-Distance Networks," preprint.

[ChS87] Chen, M.-S., and Shin, K. G., "Processor Allocation in an N-Cube Multiprocessor Using Gray Codes," IEEE Trans. Computers, Vol. C-36, No. 12, December 1987.

[ChS89] Chen, M.-S., and Shin, K. G., "Task Migration in Hypercube Multiprocessors," Proc. 16th Annual Int'l Symp. Computer Architecture, May 1989, pp. 105-111.

[ChT90] Chuang, P. J., and Tzeng, N.-F., " Dynamic Processor Allocation in Hypercube Com-

puters", IEEE 1990.

[Dal87] Dally, W. J., "Wire Efficient VLSI Multiprocessor Communication Networks," Proc. Stanford Conference on Advanced Research in VLSI, Paul Losleben, Ed., MIT Press, Cambridge, MA, March 1987, pp. 391-415.

[Dal90a] Dally, W. J., "Performance Analysis of k-ary n-cube Interconnection Networks," IEEE Trans. Computers, 39(6), 1990.

[Dal90b] Dally, W. J., "Network and Processor Architecture for Message-Driven Computers," in R. Suaya, and G. Birtwistle (Eds.), *VLSI and Parallel Computation*, Morgan Kaufmann Publishers, San Mateo, CA, pp. 140-222, 1990.

[DaS87] Dally, W. J., and Seitz, C. L., "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," IEEE Trans. Computers, Vol. C-36, pp. 547-553, 1987.

[DNS81] Dekel, E., Nassimi, D., and Sahni, S., "Parallel Matrix and Graph Algorithms," SIAM J. Comput., Vol. 10, pp. 657-673, 1981.

[DiJ81] Dias, D. M., and Jump, J. R., "Analysis and Simulation of Buffered Delta Networks," IEEE Trans. Computers, Vol. C-30, pp. 273-282, August 1981.

[Dud78] Dudley, R. M., "Central Limit Theorems for Empirical Measures," Ann. Probability, Vol.6, No.6, pp. 899-929, 1978.

[Ede91] Edelman, A., "Optimal Matrix Transposition and Bit Reversal on Hypercubes: All-to-All Personaliced Communication," J. Parallel Distrib. Comput., Vol. 11, pp. 328-331, 1991.

[ElL90] El-Amany, A., and Latifi, S., "Bridged Hypercube Networks," J. Parallel Distrib. Comput. 10, pp. 90-95, 1990.

[Fen81] Feng, T. Y., "A Survey of Interconnection Networks", IEEE Computer, pp. 12-27, December 1981.

[Fra90] Fraigniaud, P., "Complexity Analysis of Broadcasting in Hypercubes with Restricted Communication Capabilities," Report 90-16, ENS Lyon, France, May 1990.

[Gal68] Gallager, R. G., *Information Theory and Reliable Communication*, 1968.

[GrG86] Greenberg, A. G., and Goodman, J., "Sharp Approximate Models of Adaptive Routing in Mesh Networks," in J. W. Cohen, O. J. Boxma and H. C. Tijms (Eds.), *Teletraffic Analysis and Computer Performance Evaluation*, pp. 255-270, Elsevier, Amsterdam, 1986, revised 1988.

[GrH90] Greenberg, A. G., and Hajek, B., "Deflection Routing in Hypercube Networks," to appear IEEE Trans. Communications, June 1989 (revised December 1990).

[GrL89] Greenberg, R. I., and Leiserson, C. E.,"Randomized Routing on Fat Trees," Advances in Computing Research, Vol. 5, pp. 345-374, 1989.

[HaC87] Hajek, B., and Cruz, R. L., "Delay and Routing in Interconnection Networks," in A.R. Odoni, L. Bianco, and G. Szago (Eds.), *Flow Control of Congested Networks*, Springer-Verlag.

[HaC90] Hajek, B., and Cruz, R. L., "On the Average Delay for Routing Subject to Independent Deflections," submitted to IEEE Trans. Information Theory, June 1990.

[Haj91] Hajek, B., "Bounds on Evacuation Time for Deflection Routing," Distrib. Comput., 5:1-6, 1991.

[HHL88] Hedetniemi, S. M., Hedetniemi, S. T., and Liestman, A. L., "A Survey of Gossiping and Broadcasting in Communication Networks," Networks, Vol. 18, pp. 319-349, 1988.

[Hil85] Hillis, W. D., *The Connection Machine*, Cambridge, MA: The MIT Press, 1985.

[Ho90] Ho, C. T., "Full Bandwidth Communications on Folded Hypercubes," Research Report RJ 7434 (69605), IBM Almaden Research Center, April 1990.

[HsB90] Hsu, J., and Banerjee, P., "Performance Measurements and Trace-Driven Simulation of Parallel CAD and Numeric Applications on Hypercube Multicomputers," in Proc. 17th Intl. Sypm. Computer Architecture, Seattle, WA, May 1990.

[Hsu90] Hsu W. J., "Fibonacci Cubes-A New Computer Architecture for Parallel Processing," preprint, October 1990.

[Joh87] Johnsson, S. L., "Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures," J. Parallel and Distr. Comput., Vol. 4, pp. 133-172, 1987.

[JoH89] Johnsson, S. L., and Ho, C. T., "Optimum Broadcasting and Personalized Communication in Hypercubes," IEEE Trans. Computers, Vol. C-38, pp. 1249-1268, 1989.

[Joh90] Johnsson, S. L., "Communication in Network Architectures," in R. Suaya, and G. Birtwistle (Eds.), *VLSI and Parallel Computation*, Morgan Kaufmann Publishers, San Mateo, CA, pp. 223-389, 1990.

[Kat88] Katseff, H. P. , "Incomplete Hypercubes," IEEE Trans. Computers, Vol. 37, No. 5, pp. 604-607, May 1988.

[KeK79] Kermani, P., and Kleinrock, L., "Virtual Cut-Through: A New Computer Communicating Switching Technique," Comput. Networks, Vol. 3, pp. 267-286, 1979.

[Koc88] Koch, R., "Increasing the Size of the Network by a Constant Factor Can Increase Performance by More than a Constant Factor," in 29th Annual Symposium on Foundations of

Computer Science, IEEE, pp. 221-230, October 1988.

[Koc89] Koch, R., *An Analysis of the Performance of Interconnection Networks for Multiprocessor Systems*, Ph.D. Thesis, MIT, May 1989.

[KrS83] Kruskal, C., and Snir, M., "The Performance of Multistage Interconnection Networks for Multiprocessors," *IEEE Trans. on Computers*, C-32(12), pp. 1091-1098, December 1983.

[KVC88] Krumme, D. W, Venkataraman, K. N., and Cybenko, G., "The Token Exchange Problem," Tufts University, Technical Report 88-2, 1988.

[Lei83] Leighton, F. T., *Complexity Issues in VLSI*, M.I.T. Press, Cambridge MA., 1983.

[Lei85] Leiserson, C. E., "Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing," IEEE Trans. Computers, Vol. C-34, October 1985.

[Lei92a] Leighton, F. T., *Introduction to Parallel Algorithms and Architectures: Arrays - Trees - Hypercubes*, Morgan Kaufmann, San Mateo, CA, 1992.

[Lei92b] Leighton, F. T., personal communication, 1992.

[LeL89] Leighton, F. T., and Leiserson, C. E., Class notes for 6.845, 1989.

[LEN90] Lan, Y., Esfahanian, A.-H., and Ni, L., "Multicast in Hypercube Multiprocessors," J. Parallel Distrib. Comput., Vol. 8, pp. 30-41, 1990.

[Max87] Maxemchuk, N. F., "Routing in the Manhattan Street Network," IEEE Trans. Commun., COM-35(5), pp. 503-512, May 1987.

[Max89] Maxemchuk, N. F., "Comparison of Deflection and Store-and-Forward Techniques in the Manhattan Street and Shuffle-Exchange Networks," in INFOCOM '89, Vol. 3, pp. 800-809, April 1989.

[Max90] Maxemchuk, N. F., "Problems Arising from Deflection Routing: Live-lock, Lock-out, Congestion and Message Reassembly," Proceedings of NATO Workshop on Architecture and High Performance Issues of High Capacity Local and Metropolitan Area Networks, France, June 1990.

[McV87] McBryan, O. A., and Van de Velde, E. F., "Hypercube Algorithms and their Implementations," SIAM J. Sci. Stat. Comput., Vol. 8, pp. 227-287, 1987.

[MiC87] Mitra, D., and Cieslak, R. A., "Randomized Parallel Communications on an Extension of the Omega Network," J. ACM, Vol. 34, pp. 802-824, October, 1987.

[Ozv87] Ozveren, C., "Communication Aspects of Parallel Processing," Laboratory for Information and Decision Systems Report LIDS-P-1721, M.I.T., Cambridge, MA, 1987.

[Pat81] Patel, J. H., "Performance of Processor-Memory Interconnection for Multiprocessors," IEEE Trans. Comput., Vol. C-30, pp. 545-556, April 1981.

[Pis84] Pissanetzky, S., *Sparse Matrix Technology*, pp. 106-109, Academic Press, Orlando, Florida, 1984.

[Pip84] Pippenger, P., "Parallel Communication with Limited Buffers," Proc. of the 25th Annual IEEE Symposium on Foundations of Computer Science, pp. 127-136, 1984.

[RaS90] Ranka S., and Sahni S., *Hypercube Algorithms with Applications to Image Processing and Pattern Recognition*, Springer-Verlag, New York, 1990.

[Rys65] Ryser, H. J., *Combinatorial Mathematics*, The Mathematical Association of America, Rahway, N.J., 1965.

[SaS88] Saad Y., and Schultz, M. H., "Topological Properties of Hypercubes," IEEE Trans. Computers, Vol. 37, pp. 867-872, July 1988.

[SaS89] Saad, Y., and Schultz, M. H., "Data Communication in Hypercubes," J. Parallel and Distr. Comput., Vol. 6, pp. 115-135, 1989.

[SaS89] Saad Y., and Schultz, M. H., "Data Communication in Parallel Architectures," Parallel Computing, Vol. 11, pp. 131-150, 1989.

[Sta91] Stamoulis, G., *Routing and Performance Evaluation in Interconnection Networks*, Ph.D. Thesis, MIT, Report LIDS-TH-2035, May 1991.

[StT90] Stamoulis G., and Tsitsiklis J. N., "Efficient Routing Schemes for Multiple Broadcasts in Hypercubes," Laboratory for Information and Decision Systems, Report LIDS-P-1948, February 1990.

[StW87] Stout, Q. F., and Wagar, B., "Passing Messages in Link-Bound Hypercubes," in Proc. 1986 Hypercube Conference. SIAM, Philadelphia, pp. 251-257, 1987.

[TBT88] Tseng, P., Bertsekas, D. P., and Tsitsiklis, J. N., "Partially Asynchronous Parallel Algorithms for Network Flow and Other Problems".

[Top85] Topkis, D. M., "Concurrent Broadcast for Information Dissemination," IEEE Trans. Software Engineering, Vol. 13, pp. 207-231, 1983.

[TuR88] Tucker, L. W., and Robertson, G. G., "Architectures and Applications of the Connection Machine," IEEE Computer, pp. 26-38, August 1988.

[Ull84] Ullman, J. D., *Computational Aspects of VLSI*, Computer Sciences Press, 1984.

[Upf84] Upfal, E., " Efficient Schemes for Parallel Communication," J. ACM, Vol. 31, pp.

507-517, 1984.

[VaB81] Valiant, L. G., and Brebner, G. J., "Universal Schemes for Parallel Communication," in Proc. of the 13th Annual symposium on Theory of Computing, pp. 263-277, 1981.

[VaB90a] Varvarigos, E. A., and Bertsekas, D. P., "Communication Algorithms for Isotropic Tasks in Hypercubes and Wraparound Meshes," Laboratory for Information and Decision Systems Report LIDS-P-1972, M.I.T., Cambridge, MA, March 1990, to appear in Parallel Computing.

[VaB90b] Varvarigos, E. A., and Bertsekas, D. P., "Multinode Broadcast in Hypercubes and Rings with Randomly Distributed Length of Packets," Laboratory for Information and Decision Systems Report LIDS-P-2006, M.I.T., Cambridge, MA, November 1990, to appear in IEEE Trans. on Paral. and Distrib. Systems.

[Val82] Valiant L. G., " A Scheme for Fast Parallel Communication," SIAM J. Comput., Vol. 11, pp. 350-361, 1982.

[Var90] Varvarigos, E. A. ,*Optimal Communication Algorithms for Multiprocessor Computers*, MS. Thesis, Report CICS-TH-192, Center of Intelligent Control Systems, MIT, 1990.

[Wan88] Wang, E. Y., *Traffic Control in a Multichannel Optical Fiber Communication Network*, MS. Thesis, Lab. for Information and Decision Systems Report LIDS-P-1784, MIT, 1988.