PARALLEL COMPUTATION ON SPARSE NETWORKS OF PROCESSORS

GORDON BREBNER

Ph. D.
University of Edinburgh
1983

I hereby declare

- that this thesis has been composed by myself; and
- that the work described herein is my own, except where stated in the text.

Abstract

In this thesis, problems related to parallel computation on a set of processors connected together in a sparse network are considered. The two main interconnection graph families studied are the n-dimensional binary hypercubes ("cubes"), and the De Bruijn graphs of degree d (d-shuffles).

The main focus is directed towards analysis of the time and space required to route data between processors, in order to model the operation of an idealistic parallel computer in which each processor has access to a shared global memory, and so discover whether general purpose parallel computation is feasible on a fixed interconnection model. The randomised routing algorithm of Valiant, which has optimal time complexity with high probability, is used. It is shown that this may be generalised to model simultaneous reads at the expense of a space penalty, but that generalisation to modelling simultaneous writes involves a time penalty. Since analytic complexity results for the routing algorithm do not yield the tight time and space bounds that are desirable for practical purposes, its behaviour is investigated in considerable detail by simulating it on a serial computer. The basic algorithm is investigated on several other interconnection networks relevant to parallel computation. Also, various modifications to the basic algorithm are examined. The results may be summarised by the empirical relationships that, when the expected load on all routing processors is equal, (i) Expected time grows at approximately double the rate of expected route length, and (ii) Expected space grows at approximately half the rate of the base two logarithm of the number of processors (independent of interconnection pattern). These observations indicate that the routing algorithm is very efficient.

While the routing results indicate that general purpose computation is feasible using certain interconnection patterns, albeit with time complexities

increased by a logarithmic factor, it is likely that algorithms should still be optimised to take account of the pattern. A preliminary investigation of the extension of the analysis of the routing algorithms to the analysis of general algorithms is undertaken, and an analytic framework is suggested.

A technological restriction on many parallel computing devices is that they are amenable to two-dimensional rectangular realisation. An organisation where processors are arranged in a grid is bad for routing data. The necessary and sufficient areas required to lay out cubes and d-shuffles in a grid are determined, and it is proved that any such layouts require long paths in the grid, which means long inter-processor communication times. The conclusion is that general purpose computers are not viable in such technologies. It is also shown that, while algorithms for grid-style architectures may be efficiently simulated on a cube network, they cannot be efficiently simulated on a d-shuffle network.

Table of Contents

Chapter 1

Parallel Computation Schemes

1.1 Introduction

Parallelisation of computations is probably the most important issue which faces Computer Scientists at the present time and, in the future, it promises to pose an increasing challenge. From the earliest days of computing, economics have favoured an approach whereby algorithms are executed as a series of sequential steps by a processor which manipulates data held in a memory. This corresponds well with the typical observable behaviour of a human being who is tackling a non-trivial problem, and so algorithm design is frequently a formalisation of a series of human activities, albeit on a larger scale, which lead to the desired solution.

Now, advances in technology mean that the computer designer has the freedom to incorporate many processors which can operate simultaneously and the fundamental question is how this power may be effectively harnessed by algorithms. Ideally, multiplying the number of processors should lead to a proportional increase in the speed of problem solving. However, a consideration of the analogous situation in the human world where, instead of one person solving a problem, a large number of people are solving a problem, shows that it is not the case in general that the work done by one person in a year can be done by 365 people in a day. The fundamental reason for this is communication. Most tasks cannot be formulated as a set of 365 independent activities - some degree of interaction is necessary to exchange information.

In designing a parallel algorithm, the key goal is to partition a problem among parallel processors in such a way that communication can be performed simply and each processor can be used at as near maximum efficiency as possible, and so new approaches to problem solving are

needed. This thesis is concerned with communication. The viewpoint is that of the computer designer rather than that of the algorithm designer, and the aim is to investigate how communication capabilities can be efficiently and realistically provided. Thus, the emphasis is placed on communication costs rather than processing costs throughout.

In developing an understanding of serial compuatation, many different models have been proposed, such as Turing Machines [24] and Random Access Machines [1], and a property of the more fundamental results is that they hold, regardless of which reasonable model is chosen. There is considerably more scope in choosing models of parallel computation and indeed a large variety exist already, employing fairly different techniques. Research into behavioural aspects of parallel computers, such as reachability, deadlock, liveness etc., typically uses models based on finite state machines and program schemata, for example Petri nets and vector addition systems. Here, behavioural issues will be assumed to be dealt with, and more complexity-theoretic matters will be investigated. Thus, it is desirable to have a model which represents realistic parallel computers more faithfully, and one which is widely used will be employed. Cook [12] gives an excellent survey of synchronous parallel computer models. The model here consists of a collection of parallel processors, operating synchronously, which cooperate in a tightly coupled manner to solve some problem.

Clearly, the model as described leaves some degrees of freedom, which must be fixed. While the model might embrace processors which are extremely simple, for example a processing element occurring in a very large scale integrated (VLSI) circuit, it will be assumed that they do have general purpose sequential capabilities as well as some local memory. The synchronous, tightly coupled approach refers to the manner in which algorithms are implemented on this model. Each processor computes in step, and processors communicate at synchronised time intervals. Typically, each processor may be executing the same program, possibly taking account of which processor it is. The number of processors is assumed to be related to the size of the problem, for example N processors

to sort N keys or to compute an N point Fourier transform. The model is intended to represent a general purpose parallel computer. While general purpose serial computers have become accepted as the norm, most exploitations of parallelism have been for special purpose applications. This applies in particular to VLSI realisations, for example [34], but also to more conventional constructions, for example pipelined central processing units. A central question is whether realistic parallel computers can be built which can exploit the parallelism inherent in arbitrary problems.

The important feature of the model which has not yet been resolved is how processors communicate with one another. There are essentially two ways in which this can be viewed. Either a global shared memory exists and communication is done by reading from, and writing to, this memory, or the processors are connected together in a fixed network, usually assumed to be fairly sparse, and processors can only communicate with neighbours in the network. The first type of model (the idealistic model) is useful when one is trying to explore the inherent parallelism of a problem, and offers a convenient framework in which to write and analyse algorithms. Unfortunately, it is not amenable to realisation in current technologies, due to fan-in restrictions, and this is why the second type (the realistic model) is of more interest in a practical sense. The results in this thesis are primarily concerned with the efficient simulation of idealistic models by realistic models. Alternative names, suggested by Schwartz [44], for the idealistic and realistic models are paracomputers and ultracomputers respectively.

Several idealistic models of parallel computation have been proposed, for example [18, 20]. The main point on which the models differ is that of read or write conflicts, that is whether more than one processor is allowed to read from the same memory location simultaneously and whether more than one processor is allowed to write to the same memory location simultaneously (and, if so, how such write conflicts are resolved). Such restrictions affect the computational power of the model. As a simple example, consider computing the logical OR of n bits contained in memory locations $x_1, \ldots, x_n$. Then there is a simple algorithm which runs on

Goldschlager's SIMDAG [20], which allows multiple writes in one time step. Processor i ($1 \leq i \leq n$) merely reads location x_i, then writes a one in x_1 if and only if x_i is equal to one. However, if multiple writes are not allowed, Cook and Dwork [13] have shown that $\Omega(\log n)$ time is required. In Goldschlager's model, the lowest numbered processor succeeds in a write conflict. Alternatives are an arbitrary winner, or some combination of the competing data. Shiloach and Vishkin [45] allow a multiple write only if all processors wish to write the same datum.

Clearly, any of these idealistic models can simulate a realistic model, merely by allocating one memory location to each directed connection in the network. Simulation of the above varieties of idealistic models by realistic models will be considered in subsequent sections. The primary question to be resolved is how a particular memory access can be simulated efficiently. Of course, if an algorithm makes infrequent accesses to the global memory, the speed of simulation is less important.

1.2 Networks as parallel computers

The important feature of a processor network is the interconnection scheme. To describe this, it is most convenient to consider the underlying graph, which has vertices corresponding to processors and edges corresponding to inter-processor connections. Some elementary graph-theoretic terminology which will be employed throughout is collected together here.

A graph G is specified by a pair (V, E), where V is a set of vertices and E is a set of edges. If the graph is directed, E is composed of ordered pairs of edges, i.e.

$$E \subseteq \{(u,v) \mid u,v \in V\}$$

If the graph is undirected, E is a set of unordered pairs, i.e.

$$E \subseteq \{(u,v) \mid u,v \in V\}$$

An edge (u, v) in a directed graph is an out edge of u and an in edge of v. The number of out edges at u is the out degree of u, and the number of

in edges at v is the in degree of v. If u has the same in degree and out degree, it is said to have degree of the same number. In an undirected graph, the number of edges (u,v) such that $v \in V$ is the degree of vertex U. A path in G is a sequence of edges

$$(u_1, u_2), (u_2, u_3), \ldots, (u_{i-1}, u_i)$$

for some $i \geq 2$. The degree of a graph is the maximum degree over all vertices, if the degree of all vertices is defined. The diameter of a graph is equal to

$$\max (d(u,v) \mid u,v \in V)$$

where d(u,v) is the length of the shortest path between u and v. The size of a graph is the size of the vertex set, |V|.

The graphs considered here will always be directed, but some may be considered effectively undirected since, for every edge (u,v), there is a corresponding edge (v,u).

It should be noted that, when some feature of the underlying graph is of interest, the words "graph" and "vertex" will be used and, when some feature of the processor network is of interest, the words "network" and "node" will be used. "edge" is employed in both contexts, hopefully unambiguously, in preference to technologically suggestive terms such as "wire".

The organisation of the network will be such that, at each node, there is a processor and some local memory. To simulate the operation of the idealistic model, it is necessary to move data across the network from a source node, which stores some datum required from the global memory, to a destination node, which requests it. The realisation of a complete set of such requests from all processors involves a fundamental problem, namely that of efficiently routing data through a network. The term packet will be used to describe an object which is in transit between a source and destination node, and the network can be considered as a packet switching network. A packet will usually carry data, but may be used to carry control information. When routing is taking place, nodes will be termed "routing

nodes" and, when computation is taking place, nodes will be termed "processing nodes" (or just "processors"). The path followed by a packet will often be called its route.

Only routing problems will be considered here. Lev, Pippenger, and Valiant [33] show that, if the collection of packets corresponding to a simultaneous set of memory requests can be delivered in time r(n), then one computation step of a PRAC (a model without multiple reads and writes) can be simulated in O(r(n)) time. Borodin and Hopcroft [9] indicate how to achieve a similar result for more powerful models. The interesting point is what times r(n) can be achieved for routing a collection of packets. The central importance of routing networks to general purpose parallel computers can be seen in such examples as the more theoretical universal computer of Galil and Paul [19], as well as in the practical proposal of Gottlieb et al [22].

It can be seen that there is an asymmetry in simulating reads and writes. To simulate a read, it is necessary for nodes to send requests to nodes which store the required data. They then send data to the requesters. For a write, two such stages are not required since writing nodes merely have to send the data to the correct storage node. The "missing" second stage corresponds to the recipient sending an acknowledgement back to the sender. Such a feature would be desirable, for example, if an asynchronous parallel system was being developed. Here, the abstracted problem which is considered can be interpreted in terms of packets containing requests or packets containing data. In fact, from now on, little reference will be made to either the contents of packets or the motivation for routing packets.

1.3 Routing on networks

The fundamental problem which will be considered is that of realising permutations. That is, one packet starts at each node and, after routing, one packet finishes at each node. The ability of a network to permute data is crucial if it is to be successful. Permutations will also be generalised to h-relations, in which h packets start and finish at each node. Very often,

it is required to realise relations which are not total, that is all nodes are not both sources and destinations, and so partial permutations and partial h-relations are investigated. Finally, relations which alter the number of packets, that is multiple copies of the same packet are sent to several destinations or multiple packets are sent to the same destination, are required. Regardless of which type of relation is being implemented, an important characteristic of the problem is that the relation changes every time that a data transfer takes place, in general. Therefore, routing solutions designed for applications in which routes are set up and then used for multiple data transfers, for example telephone systems, are usually unsuitable because the set up time is relatively large compared with the data transfer time. An example is the Benes permutation network [6] which achieves $O(\log N)$ time for routing on a constant degree graph of size N but, even on idealistic models, the best known set up time is $O((\log N)^2)$ [33].

Even if such solutions could satisfy timing constraints, they are still unsatisfactory for another reason. This is essentially because the behaviour of each routing node is determined by knowledge of the entire permutation. Given the sparse nature of the network, it is not possible to collect information from all nodes at one node in a short time. In fact, if the network has degree d then, within $O(\log N)$ time, information can be gathered from at most $O(d \log N)$ nodes, merely because of in edge constraints. Even if some sort of merging of information packets is allowed, packets of size $\Omega(\frac{N}{d})$ would be needed to accumulate the $\log N! = O(N \log N)$ bits necessary to represent a permutation. Therefore, given that the routing strategy has only a small amount of information to work from at any node, the conclusion is that any algorithm must be highly distributed.

An example of a routing strategy suitable for sparse networks which is totally distributed, in the sense that each routing node makes decisions based only on information at itself, is obtained by considering the sorting algorithm of Batcher [5]. Parallel sorting can always be used to route permutations since, to realise a permutation π on $(1, \ldots, N)$, each packet

is labelled with $\pi(i)$ where i is its starting point, and the packets are sorted according to the keys $\pi(i)$. Given that π is bijective, if $\pi(i) = j$ then it will be ranked jth and the packet arrives at node j. It should be noted that this approach is not immediately applicable to partial permutations. However, it is possible to modify Batcher's algorithm, albeit with some loss of efficiency.

The worst case route length of Batcher's algorithm is $O((\log N)^2)$. Since the diameter of the networks on which it is implemented is typically $O(\log N)$, there is obviously scope for improvement. No totally distributed algorithm for switching networks is currently known with better worst case complexity. Two modifications to the problem are made here, namely to allow queueing as well as switching at routing nodes, and (later) to allow the routing algorithm to be non-deterministic. These enable a wider range of solutions to be explored. Having narrowed down the problem under consideration, it is possible to give a precise description of the behaviour of the routing networks to be studied, as well as an indication of the complexity measures which are of interest.

A routing network is based upon a directed graph in which all vertices have the same in degree and out degree. The size of the graph will be denoted by N, and the degree of the graph by d. There is a collection of packets in the network and, at every integral time instant, each packet is located at some node. During the interval between each time instant, one packet may be transmitted along each edge between adjacent nodes. Every node maintains d queues of packets, one for each out edge, and some queuing discipline determines which packet is sent along the edge associated with a queue at each time interval. Therefore, at each time instant, the population of a node consists of packets which were destined for the node in its processing role, and packets queued by the node in its routing role.

Each packet which is being routed must carry its destination address, in addition to its data content. Optionally, additional information may also be carried. At each node, the routing algorithm decides, using this

information, which queue (if any) an incoming packet should be placed on. This decision is independent of the state of any other nodes. When no packet is queued at any node at some time instant, all packets have arrived at their destination, and so the routing algorithm has terminated. For a given routing request, the time complexity of the algorithm is the number of time instants before termination. The space complexity is the maximum number of packets which form a node population at any time instant.

This description of the model, which is generally accepted in the literature, e.g. [9], raises some points which should be discussed further. The implication of the notion of time is that, firstly node computation is free, and secondly that transmission time for any packet along any edge is charged as one unit with all edges being usable simultaneously. The first assumption is a common one in the study of queueing networks, e.g. [27], and, while in this case it may not always be true that transmission time considerably exceeds computation time, as long as both times are related by a constant factor it amounts merely to the choice of a particular time unit. The second assumption is slightly more controversial. In some technologies it may be the case that all connections do not have a uniform propagation time; in particular, there has been much discussion recently on the correct measure to use for VLSI [7, 10]. If a technologically restrictive approach is taken, then the choice of underlying graph typically becomes very constrained, and it is questionable whether general purpose parallel computation is reasonable at all, compared with special purpose hardware. The main justification for the approach used here is that it allows the investigation of the communication capabilities of a very general class of novel architectures, all of which are technologically feasible but not necessarily in all technologies. The results are universal in a negative sense, namely that a network with bad routing time on this model can only be worse on a more specific model.

Another questionable point is whether a node can be allowed to receive [transmit] packets along all of its in edges [out edges] during one time interval. If the degree d is constant, then time is affected only by a

constant factor if this is not so. However, as Upfal points out, this is a problem for networks with higher degrees. One possible solution is to employ parallelism within nodes and, for simple queueing strategies, it is possible to achieve constant switching time as well as multiple edge driving capability. Typically, however, this might require $O(d^2)$ subprocessors within nodes.

The decision to make routing decisions purely local, without consultation even with neighbouring nodes, is one of simplification. This applies to both the routing algorithm and any analysis of it. In fact, since it is still surprisingly difficult to analyse strategies, a further restriction is applied, namely that all algorithms are non-adaptive, or oblivious. Essentially, this means that a node routes packets without taking into account any other packets that may be present at it. Therefore, if the routing algorithm is oblivious then, for each source-destination pair, there is a unique route which any packet with that source and destination must follow, assuming determinism. Not only does obliviousness aid analysis, but it also has a practical motivation since it is a requirement in the memory arbitration scheme of the New York University Ultracomputer [22].

Previously, the aim of this work has been expressed as finding "efficient" routing schemes. Now that various parameters have been defined, it is possible to quantify the notion of efficiency. The aim is to achieve resource requirements that are simultaneously logarithmic in the network size N. The primary consideration is the time complexity of routing, which will often be referred to as the completion time. Obviously, the underlying graph must have logarithmic diameter if a logarithmic completion time is to be achieved. For any packet, the difference between its routing time and its path length will be termed its delay. The delay of the whole network when implementing a relation is the difference between completion time and the maximum route length. Logarithmic bounds are also desired on space requirements, that is a node should never have more than O(log N) packets present. Also packet size (including both contents and any extra information carried) should not exceed O(log N) bits. Finally, since the network must be sparse, its degree should preferably be constant and be at

most O(log N).

Since logarithms will be occurring frequently, it is perhaps useful to summarise the notation used :

lg n is the base two logarithm of n

ln n is the base e (natural) logarithm of n

$\log_k n$ is the base k logarithm of n for some k

When the base of the logarithms is constant but not important, the notation log n is used (typically in the expression O(log n)).

1.4 Choice of graph for routing networks

Before considering the time and space requirements of routing packets through a network, it is necessary to identify graphs which satisfy the constraints on degree and diameter. This leads to an important graph-theoretic problem which has been of interest for some time, namely determining the maximum number of vertices $N(d, \delta)$ which may be in a graph with given degree d and diameter δ. A well-known upper bound on $N(d, \delta)$ for undirected graphs is due to Moore [8] and a similar argument can be used for directed graphs giving :

$$N(d, \delta) \leq \frac{d^{\delta+1}-1}{d-1} \text{ for all } d \geq 2$$

Rearranging this gives a lower bound on diameter, viz.

$$\delta \geq \log_d[N(d-1)+1] = \Omega\left(\frac{\log N}{\log d}\right)$$

Thus, given the logarithmic restrictions on degree and diameter which were mentioned earlier :

$$\Omega\left(\frac{\log N}{\log \log N}\right) \leq \delta \leq O(\log N)$$

So-called Moore graphs, which attain the Moore bound, are rather rare (one example is the family of complete graphs which form the "interconnection pattern" of the idealistic computer). Much work has been

done recently on finding families of graphs which approach the Moore bound and have small degrees and diameter, for example [49, 25]. In particular, Leland introduces interesting new families of graphs which are intended for processor networks [32].

Throughout this work, two paradigmatic graph families will be used which have attracted interest in the context of both graph theory and parallel processing. Some other examples will be introduced later. A good survey of interconnection schemes for parallel processing is given by Siegal [46]. The graphs are not chosen purely on the basis of degree and diameter. It is essential for routing that no vertex lies on a high proportion of the paths in the graph, otherwise bottlenecks will result. This rules out trees, for example. The two families will be introduced in turn, along with appropriate routing algorithms.

The first family are the n-dimensional binary hypercubes, hereafter referred to merely as "cubes". An n-dimensional cube is defined as follows :

Vertices,

$$V = \{v \mid v \in (0,1)^n\}$$

Edges,

$$E = \{(\alpha\beta\gamma, \alpha\bar{\beta}\gamma) \mid \alpha \in (0,1)^i,\ \beta \in (0,1),\ \gamma \in (0,1)^{n-i-1} \text{ for } 0 \leq i \leq n-1\}$$

The cube is graph-theoretically elegant since it is just the Cartesian product of the complete graph on two vertices, K_2, with itself n times. It has degree n and (non-optimal) diameter n. Since each edge (u,v) has a corresponding edge (v,u), it is effectively undirected. The interconnection structure reflects in a natural way the communication necessary for a class of algorithms, named the Ascend/Descend algorithms by Preparata and Vuillemin [40], whose most noted member is the fast Fourier transform. These algorithms have n stages, each of which requires communication in a different dimension of the cube. In the context of routing, a natural route for packets to follow is one which changes each bit in the binary

representation of the source node address to the correct corresponding bit in the destination address since it has length logarithmic in graph size. The cube offers an ideal structure for this, and so seems an ideal candidate for analysis. The routing algorithm executed at each node of a cube network is as follows :

```
for each node α in parallel do begin
  for each packet at α do begin
    let β be the address on the packet
    if α = β then packet is at destination else begin
      d := lowest order dimension in which α and β differ
      queue packet for out edge in dimension d
    end
  end
end
```

It is clear that the algorithm must terminate with each packet at the correct destination since, at each time step, each packet still en route must either traverse a dimension in the direction of its destination, or move forward one place in a queue.

As a notational point in connection with the cube, it should be noted that, while dimensions will be numbered $0, 1, \ldots, n-1$ (0 being low order, $n-1$ being high), in text they will be referred to as the 1st dimension, ..., nth dimension when necessary.

The second family of graphs are the de Bruijn graphs [16]. As an aid to brevity, the de Bruijn graphs with degree $d \geq 2$ will always be referred to as "d-shuffle" graphs. They are defined by :

Vertices,

$$V = \{v \mid v \in (0, 1, \ldots, d-1)^n\}$$

Edges,

$$E = \{(\alpha\gamma, \gamma\beta) \mid \alpha, \beta \in (0, \ldots, d-1),\ \gamma \in (0, \ldots, d-1)^{n-1}\}$$

The graph has degree d and, when there are d^n vertices, it has

(asymptotically optimal) diameter n. It gives the best known $N(d,\delta)$ for directed graphs. Until recently, it was also best for undirected graphs, but it has now been beaten, albeit by small constant factors. Indeed, the new technique of Jerrum and Skyum [25] is still underpinned by the d-shuffle. Therefore, d-shuffles offer hope for routing purposes since they can also have constant degree and do not have bottlenecks. In the routing context, the cube may be regarded as the "idealistic" candidate, and the d-shuffle as the "realistic" candidate. It can be noted that the 2-shuffle is very similar to the shuffle-exchange graph (considered later), which has been proposed as an architecture for both special and general purpose parallel computers [48, 22].

The routing algorithm employs similar principles to that for the cube, in that the d-ary representation of the source node is changed to that of the destination. However, since d-ary digits cannot be altered "in situ", they must be "shifted" to the one (rightmost) position where they can be changed. The algorithm will be slightly inefficient because all packets follow a route of length precisely n with a different d-ary digit changed at each step. As well as a destination address, packets carry a counter which counts down route steps from n to 0.

```
for each node in parallel do begin
  for each packet at the node do begin
    let the packet have counter c and destination σ
    if c = 0 then packet is at destination else begin
      c := c-1
      queue packet with new counter c for edge which shifts in β
          where σ = αβγ for some
          α ∈ (0,...,d-1)^(n-c-1) and γ ∈ (0,...,d-1)^c
    end
  end
end
```

At each step, every packet still en route either has its counter reduced by one and the corresponding d-ary digit altered to the correct one, or moves one place forward in a queue, so packets are routed correctly. For some

routes, this algorithm is bad (for example, 0^n to 0^n), but later on in Chapter Three it will be shown that it is not much worse than a more optimised algorithm. This one has the advantage of being easier to implement at nodes, thereby simplifying switching. A similar scheme for the cube, which has far better connectivity, leads to far greater inefficiency in routing.

Having discussed the choice of specific graphs, a negative result will now be presented which applies to all reasonable graphs.

1.5 An argument against determinism

The investigative framework has been laboriously described. Now a result of Borodin and Hopcroft [9] will be stated :

> **Theorem (1.1):** In any network with N nodes and in degree d, the time required in the worst case by any oblivious routing strategy is $\Omega(\frac{\sqrt{N}}{d^{3/2}})$.

The outline of the proof is based upon the fact that a routing request can be constructed which forces a large number of packets to pass through the same node on their routes. Since at most d packets can leave a node at each time interval, this forces a long completion time. Arguments of a similar style will be employed in the next chapter.

This easy theorem would appear to rule out further progress on the problem. One line of attack is to remove the restriction to oblivious routing, but this appears to be very hard to analyse. Instead, one other card, mentioned earlier, can be played. The notion of an algorithm will be slightly relaxed, so that it may behave non-deterministically.

Chapter 2

Randomised parallel computation schemes

2.1 Introduction

Having observed that the best known time for deterministic routing of packets on a graph with N vertices was $O((\log N)^2)$, even when the diameter was $O(\log N)$, Valiant [54] proposed a randomised parallel algorithm which ran in $O(\log N)$ time with high probability. The notion of randomised algorithms was introduced (in the context of serial computation) by Rabin [41] and Solovay and Strassen [47]. They may be viewed as being the same as normal deterministic algorithms except that, at certain points in their execution, they pick a random value according to some probability distribution. By making appropriate use of this added randomisation, useful fast algorithms with probabilistic behaviour can be obtained.

The best known class is that of "Monte Carlo" algorithms, which always produce a result fast but with a small probability of the result being incorrect. An example is Rabin's primality testing algorithm which always gives the correct answer if the number input, n say, is prime but may wrongly identify a composite n as being prime. Monier [36] has shown that the error probability is smaller than $\frac{1}{4^k}$ when the run-time of the algorithm is $O(k \log n)$. Another class which has appeared recently consists of the "Las Vegas" algorithms [3]. These run fast and usually produce a correct result but may indicate "no result found" with small probability. This is a more useful feature than the slighly dubious result obtained from a Monte Carlo algorithm. Clearly, any Las Vegas algorithm can be trivially transformed into a Monte Carlo algorithm.

The routing algorithm of Valiant displays a different property of randomisation. The algorithm always computes the correct result, but may

not always be fast. Consider the following definition :

> **Definition (2.1):** An algorithm has time complexity T(n) with confidence $1-\epsilon$ ($\epsilon>0$) if, for every input of size n, the time taken to compute the correct result exceeds T(n) with probability at most ϵ.

Note that the probability ϵ is typically either a small constant or a vanishing function of the input size n. This definition describes the tail of the run-time distribution unlike the definition of expected time complexity given by Rabin for randomised algorithms :

> **Definition (2.2):** An algorithm has expected time complexity T(n) if, for every input of size n, the expected time taken to compute the correct result, over all randomisations, is at most T(n).

The second definition illustrates a motivation for the development of randomisation. The conventional definition of expected time complexity requires assumptions to be made about the distribution of inputs in order to demonstrate that an algorithm is "fast on average". However, with the Rabin definition, only the distribution of the random numbers chosen need be considered, and this is more predictable than "real world" inputs. Rabin gives an example of an algorithm for finding the nearest pair(s) in a set of n points in k-dimensional space which has good time complexity in this sense.

The first definition, however, is more in the spirit of normal complexity theory in the sense that worst case times are of more interest than expected times. As far as is known, Valiant's routing algorithm was the first randomised algorithm (it was certainly the first parallel one) which had a provably tight time complexity bound in the sense of this definition. It has time complexity O(K log N) with confidence $1 - \frac{1}{N^K}$ for any suitably large constant K. Before looking at the algorithm, a simpler (and apparently more intuitive) algorithm employing randomisation will be considered. It is designed to be implemented on a cube graph.

2.2 Random order of edge traversal on the cube

The central problem which any routing algorithm must attempt to solve is that of collisions caused by two packets wishing to traverse the same edge at the same time interval. When packets follow paths which involve the minimum number of inter-dimensional traversals then, as has been seen earlier, it is possible to construct input permutations which inevitably lead to collisions when routing is deterministic. Here, the order of dimensions traversed in the path of each packet will be selected randomly in the hope that not only will worst case behaviour be less dependent on pathological inputs but also that frequency of collisions will be reduced as packets are distributed "more randomly" about the cube.

Therefore, the distributed algorithm is implemented as follows :

```
for each node α in parallel do begin
  for each packet at α do begin
    let β be the destination of the packet
    if α = β then packet is at destination else begin
      let D be the set of dimensions in which α and β differ
      let d be a random member of D
      queue packet for out edge in dimension d
    end
  end
end
```

The algorithm is clearly correct, since each packet is either moved one dimension closer to its destination or one place forward in a queue at each time step. At any given time interval, one might hope that, at a given node, packets will arrive randomly along in edges and then be randomly distributed amongst out edges. In this way, collisions can be minimised and a fast completion time achieved. Unfortunately, this is not the case since input permutations can still be chosen which force bad behaviour, as the following theorem, based upon one of Valiant [54], shows :

Theorem (2.3): There exists an input permutation for which the algorithm does not have expected time complexity which is

logarithmic in the network size.

Proof: Consider any edge (u,v). If the cube has n dimensions then, for any $r \leq \frac{n-1}{2}$, there are $\binom{n-1}{r}$ source nodes at distance r from u from which paths may include the edge (u,v), and $\binom{n-1}{r}$ destination nodes at distance r from v to which paths may lead after including (u,v). Now consider the algorithm applied to a partial permutation of $\binom{n-1}{r}$ packets from the source nodes to the destination nodes. Each path intersects (u,v) with probability

$$\frac{1}{r+1}\binom{2r+1}{r}^{-1}$$

so the expected number of paths intersecting (u,v) will be :

$$\frac{(n-1)!}{r!\,(n-r-1)!} \cdot \frac{r!\,r!}{(2r+1)!}$$

$$= \frac{(n-1) \ldots (n-r)}{(2r+1) \ldots (r+1)}$$

$$> \left(\frac{n-r}{2r}\right)^{r} \cdot \frac{1}{2r+1}$$

$$= \frac{5}{2n+5}\, 2^{n/5} \quad \text{if } r = \frac{n}{5}$$

$$= \Omega\left(\frac{N^{1/5}}{\log N}\right)$$

Thus, the expected time complexity for this permutation is not logarithmic in N.

Clearly then, while this randomisation may reduce bottlenecks in the network compared with the deterministic schemes, for large networks at least it is inevitable that some permutations will make heavy demands on certain edges. In order to overcome this difficulty, Valiant suggested a particularly elegant form of randomisation which ensures that the traffic is more evenly distributed about the network, hence dramatically reducing the probability of overloaded edges, for <u>all</u> input permutations. This method will be discussed in the next section.

Note that, for reasonable sized networks of, say, under 1000 nodes, no permutation has been found which causes an implementation of the

previous algorithm to perform worse than algorithms with provably logarithmic time complexity. It is therefore probably adequate for use with present-day parallel computers. However, as network size increases, the lower bound on completion time becomes more significant, and more sophisticated routing algorithms are required.

2.3 Randomised routing

In the new routing algorithm, a packet is no longer sent from its source node to its destination node by the shortest path. Instead, the route consists of two distinct phases. In the first phase, the packet travels from the source to a randomly chosen node. Then, in the second phase, it travels from the random node to the correct destination. The effect of this change is that, in each phase, the route followed by any packet is independent of the route followed by any other and moreover, the route consists of random edges independent of the input permutation. Therefore, while initially it appears that the route length is being unnecessarily doubled, these gains are sufficient to ensure that the routing time for each phase is logarithmic with overwhelming probability. This fact will be established both analytically and experimentally in later sections.

It should be noted that this routing technique is completely general, regardless of the underlying graph for a particular network. It is also valid for realising partial h-relations, not merely permutations. In the remainder of this chapter, attention will be focused on how cube and d-shuffle networks can realise partial h-relations efficiently.

In each phase, each packet carries a current destination address. In the first phase, this is chosen to be any of the N nodes with equal probability. The final destination address must also be carried. In the second phase, this is of course the same as the current destination address. No more information than this is needed to implement the routing. The manner in which the distributed routing algorithm is implemented at each node depends on which type of network is being used.

In the case of the cube, the node algorithm described in the previous

section may be used. Thus, gains from random ordering of the random route are also obtained. In fact, the overall algorithm described previously is just the new one with the counter-intuitive first phase omitted. For the more restricted d-shuffle, it is not possible to obtain a similar random ordering, and so packets will be routed just as in the scheme described in the previous chapter. Note that, if random ordering is not employed, it is possible to do without the extra (random) address carried in the first phase and use zero instead of lg N bits for the cube, and $\lceil \lg \log_d N \rceil$ instead of $\log_d N$ bits for the d-shuffle, by distributing the randomisation to nodes as follows :

```
for each node of the n-dimensional cube in parallel do begin
  for each packet do begin
    let d be the dimension which the packet arrived on
    while d < n-1 do begin
      d := d+1
      if a zero-one random choice is one then
        queue packet for dimension d and exit from loop
    end
    { if the packet is not queued in the previous loop,
      the packet has arrived at its random destination }
  end
end
```

For a d^n node d-shuffle, each packet carries a counter from n-1 down to 0 which indicates how many random digits in the d-ary representation of the destination address have still to be chosen.

```
for each node of the d-shuffle in parallel do begin
  for each packet do begin
    if counter = 0 then packet is at destination else begin
      decrement counter
      queue packet for out edge selected according to a
        random number selected from the range 0 to d-1
    end
  end
end
```

Obviously, this makes the complexity of routing operations in nodes greater. However, if packet size was particularly important, this type of randomisation might be worthwhile. The revised algorithm is still oblivious, since packets are still treated independently. It will be seen later that random ordering of cube dimension traversal is not necessary in order to achieve a fast algorithm, and so can be omitted.

There are two main approaches to the analysis of random routing. The first one to be considered will be an early proposal of Valiant [53] (this technique will also be employed in a later section). It is less powerful than the second approach, suggested recently by Aleliunas [2] and Upfal [51]. After looking at these analytic methods, a result will be quoted which shows that, for some graphs at least, the doubling of route lengths is not merely a device for obtaining proofs, but is inevitable for any oblivious routing algorithm.

2.4 Analysis 1 : Packets intersecting the path of a fixed packet

The sequence of edges forming the route of a fixed packet is considered. A collision with another packet may result if that packet uses any of these edges. By estimating the probability of this happening, and then summing over all the packets which may collide with the fixed packet, it is possible to obtain an upper bound on the probability that the fixed packet collides with some number, k say, of other packets and hence suffers a delay of k time units. If this argument is repeated with each packet being the fixed one in turn, the probability of any packet having k collisions can be obtained, giving the probability of completion time being increased by k units due to queueing.

Clearly, this method cannot be expected to yield tight bounds. All intersections of pairs of routes are counted as collisions, although in practice packets do not usually compete for an edge because they reach it at different time intervals. In effect, the proof is aiming to achieve an ideal routing whereby all paths are disjoint. In view of this, it is not surprising that it does not give useful bounds for all graphs, in particular those with constant degree. While the cube and a logarithmic degree shuffle have

sufficient edges to achieve disjoint routes in theory, constant degree shuffles certainly do not.

Before stating the theorem on completion time, two technical terms must be defined. A randomised routing scheme is <u>symmetric</u> if, independently of the permutation being realised, the expected number of distinct packets which traverse an edge is the same for every edge in the network. A scheme is <u>non-repeating</u> if whenever two packets take paths $p_1p_2 \ldots p_r$ and $q_1q_2 \ldots q_s$, in which $p_i = q_j$ and $p_k = q_l$ $(k > i)$ it is the case that $k-i = l-j$ and for all m such that $i \leq m \leq k$, $p_m = q_{m+j-i}$, that is any two paths never have more than one common sequence of edges.

It is not hard to see that the scheme for the cube which does not involve randomisation of the order of edge traversal is both symmetric and non-repeating when total h-relations are being realised. In general, it is not symmetric if partial h-relations are being realised and it is not non-repeating if edge ordering is random, and so the next theorem cannot be directly applied. However, by slight adjustments to the proof, both of these cases can be handled in the desired manner.

> **Theorem (2.4):** [Valiant] In any oblivious random routing scheme which is symmetric and non-repeating and has (i) N nodes, (ii) T = hN packets, (iii) degree d, (iv) expected route length η, and (v) maximum route length μ, the probability that some packet is delayed by at least k units during one of the phases is less than $(\frac{e\eta\mu h}{kd})^k . T$, where e is the base of natural logarithms.

For the n-dimensional cube, it is the case that $N = 2^n$, $T = h2^n$, $d = n$, $\eta = n/2$, and $\mu = n$. Therefore, the probability that a phase fails to finish within time cn is at most

$$(\frac{eh}{2(c-1)})^{(c-1)n} . h2^n$$

Now, for all $c \geq 3.5h + 1$, this expression is at most $\frac{h}{N^{c-1}}$, and thus the probability vanishes rapidly as c increases.

The standard algorithm for the d-shuffle is neither symmetric or non-repeating. By modifying it so that packets always follow the shortest path, rather than the straightforward path, the scheme can be made non-repeating. Then, by employing a modified version of the proof of the general theorem, a result of the same style can be obtained. However, in the case of a constant degree d-shuffle, which is the one of interest, a vanishing probability is not obtained. To achieve this, the tighter technique of the next section is required.

The proof of the results using this type of collision analysis involve the use of some non-trivial results from probability theory. Recently, an interesting new approach for estimating inter-packet collision probabilities has been suggested by Reisch and Schnitger [43]. They apply Kolmogorov complexity [39] in the analysis, which makes the proof easier. Without going into the details Kolmogorov complexity, the main point of the argument used is as follows : The routes involved in either phase of the scheme for the n-dimensional cube can be precisely represented as a string of $n2^n$ bits. Given the randomisation of routes, each such string is equally likely to be the description of a particular run. Now, it can be shown that any run which involves k or more collisions with a fixed route can be represented in at most

$$n2^n - \frac{k-5n}{2}$$

bits. Therefore, by an information-theoretic argument, the probability that k or more collisions occur must be at most $2^{-(k-5n)/2}$. Using a more complicated representation of runs with many collisions, this probability bound can be improved.

2.5 Analysis 2 : Packets intersecting delay sequences

In the previous analysis, a far too pessimistic view was taken of collisions. This method rectifies the problem by identifying paths in the network which cause delays, and ultimately cause the completion time to be delayed. In order to do this, it is necessary to put further conditions on the routing algorithm, but these are not unduly restrictive. The approach of Aleluinas is expressed in terms of critical path analysis, and is more

complex than that of Upfal. Here the common underlying features will be outlined together with the results obtained, which can be applied to graphs with constant degree.

To capture the notion of packets meeting at the same time interval, a rather artificial constraint is placed on the routing algorithms used by nodes. If μ is the maximum route length, then a packet can be at stage i, for $1 \leq i \leq \mu$, of its journey at any time interval. A node will only start forwarding packets at stage i when all packets which pass through it at stage i−1 have been forwarded. Obviously, in practice, a more realistic algorithm would merely give priority to packets at earlier stages of their journey, but it is easier to analyse the slower algorithm. Now the collection of <u>activities</u> $\{[u,i] \mid u \text{ is a node}, 1 \leq i \leq \mu\}$, corresponding to node u handling packets at stage i, is considered.

If (u,v) is an edge in the network, [v,i+1] cannot finish until [u,i] is complete. Also, [u,i+1] cannot finish until [u,i] is complete. Suppose that, for some node n_μ, the activity $[n_\mu, \mu]$ is one of the last activites to complete, that is it causes the completion time for routing to be delayed. Then, working backwards, a node $n_{\mu-1}$ can be found such that, either $n_{\mu-1} = n_\mu$ or the edge $(n_{\mu-1}, n_\mu)$ exists and $[n_{\mu-1}, \mu-1]$ was one of the last activities which delayed $[n_\mu, \mu]$ starting. Continuing in this way, a complete <u>delay sequence</u> $(n_1, n_2, \ldots, n_{\mu-1}, n_\mu)$ can be identified. In general, more than one delay sequence will exist for a given run and so all of the "last activities to complete" at each stage must be considered, rather than just one. Given such a delay sequence, the proofs are obtained by bounding the probability of packets reaching node n_i when they are at stage i. This allows the number of packets handled by the delay sequence, and hence the completion time, to be bounded.

It should be noted that the separation of the behaviour of the network into stages corresponding to the stages of packets on their routes is just a formal method for interpreting a recirculating network as though it was in fact an expanded network with μ stages. This distinction between recirculating and expanded is a common feature of parallel computing

architectures in general, for example FFT networks, and the point will be considered again in Chapter Four.

The following theorems are applicable to random permuting schemes employing priority queueing at nodes which have the property that, when a packet leaves a node, the out edge is chosen with equal probability. This property is clearly true for the d-shuffle scheme. However, it does not hold for the cube since a packet never leaves along the same dimension as it arrived on. The statement of both theorems has been modified slightly to fit more consistently into this chapter.

> **Theorem (2.5):** [Aleliunas] For a network with constant degree d and $N = d^n$ nodes, if $m = \frac{d}{d-1}\log_d(d-1)N$ then the probability that the completion time for a phase exceeds cm is less than $N^3 d^{-cm} e^{2m\sqrt{(cd)}}$ for any $c > 0$.
>
> **Theorem (2.6):** [Upfal] For a non-repeating scheme on a network with constant degree d, maximum route length μ, and the "balance" property that, for any [u,i] the mean number of packets handled is bounded by one, then the probability that the completion time for a phase exceeds $c'\mu$, for some c', is less than $e^{-cd/2}$ for any sufficiently large c.

The first theorem justifies the use of the d-shuffle graph analytically. No examples will be given here of schemes which are balanced in the sense of Upfal. His example employs a three-phase algorithm in which the phases are not entirely distinct. Indeed, a slightly different model to the one used here appears in his paper. Note that, when the term "balanced" is used later in this work, it refers to an intuitive ideal notion of balance rather something provable in the technical sense of Upfal. That is, packets are equiprobably distributed among nodes and packets leave nodes along equiprobably chosen out edges at each time step. More details of the style of the proofs of the above theorems will be seen in Chapter Four set in a more general context.

It can be seen that this technique is still not bounding network behaviour as tightly as might be possible because the time analysis considers the total

collection of packets at each node rather than the total collection of packets at each edge. Since the desired results can be obtained without this further complication, it is not a matter of concern. However, a further benefit of the more gross analysis is that space complexity results can be deduced immediately using the following theorem

> **Theorem (2.7):** If a random routing scheme has provable completion time T for both phases using an activity-style analysis, then it also has space complexity T for both phases.
>
> **Proof:** If a node had a population of more than T at any time, then a sequence of activities could be identified, involving the dispersal of the population at that node, which delays the completion time beyond T. The size of any node population in the first phase due to arrivals is dealt with by the simultaneous time bound requirement on both phases.

Note that the results of Aleliunas and Upfal, which assume constant degree d, are not particularly useful for obtaining asymptotic space complexity results since a scheme with completion time T clearly has space complexity bounded by d.T.

Before leaving analytic proofs of random routing schemes for realising permutations and h-relations, a further comment on the two-phase algorithm is in order. The following result proved by Valiant [55] shows that, for the d-shuffle graph, which has optimal diameter, the algorithm is essentially optimal with respect to path length, for oblivious algorithms.

> **Theorem (2.8):** [Valiant] For a d-shuffle with diameter $\log_d N$, any oblivious algorithm either takes time $\Omega(N^{\epsilon})$ for some $\epsilon > 0$ or makes packets travel at least $2\log_d N$ edges.

A more general result provides similar lower bounds on graphs which have nearly-optimal diameter. This does not apply to the n-dimensional cube which has diameter n, rather than the $\log_n 2^n = \frac{n}{\lg n}$ that might be expected. In fact, it is conjectured that a fast random routing algorithm exists which only involves packets travelling a maximum distance close to n. However, the result indicates that the introduction of the randomising

phase is not merely a technical trick.

2.6 Multiple reads and writes

So far, the relations which have been implemented were restricted, in the sense that no more than O(d log N) packets could start or finish at any one node of an N node network with degree d. This restriction is necessary so that packets can depart and arrive fast enough to enable an O(log N) overall time bound to be achieved. Even ignoring timing aspects, it is unlikely that any processor could handle more than O(d log N) packets. In this section, solutions to a pair of problems arising from this will be considered. These are multiple reads, where a packet must be sent from one source node to several destinations, and multiple writes, where packets are sent from several source nodes to a common destination prepared to accept only one packet. Because of the remarks above, it is not adequate, in general, just to realise an h-relation with h "copies" of a packet at a source, or destination, node respectively.

2.6.1 Multiple reads

Firstly, multiple reads will be considered. In order to implement these, a smaller number of packets initially have to be copied and distributed to a larger number of destinations. Since the copying cannot be done before packets start their journey, it must be done while packets are being routed. The idea is that a switching node now sends a packet along more than one out edge when appropriate, thus ensuring that a correct number of copies are ultimately delivered to their destinations. The question is whether this rather different way of loading the network (although it should be recalled that the final total number of packets is no more than for a normal permutation) has an adverse effect on completion time. The following theorem shows that it can if any significant copying is done in the first phase. However, a suitable algorithm is then demonstrated which does the copying in the second phase.

Theorem (2.9): Suppose that a random routing network has the property that, for some edge e and some constant $d > 1$, there

are at least d^k nodes such that a random route from any of them intersects e with probability at least $\frac{1}{d^{k+1}}$ for any k such that $1 \leq d^k \leq N$ where N is the graph size. Then, there is a multiple read operation which has $\Omega(n^\epsilon)$ completion time if each of N^δ initial packets is copied N^ϵ times in the first phase, for any $\epsilon \leq \delta$ such that $\epsilon+\delta \leq 1$.

Proof: Consider a set of N^δ nodes which satisfy the property in the statement of the theorem, and initially place one packet at each. The probability that a random route from any of them intersects the distinguished edge e is at least $\frac{1}{dN^\delta}$. Since the first phase destinations of N^ϵ copies of each packet must be random and independent, the expected number of packets traversing e is at least

$$N^\delta N^\epsilon \frac{1}{dN^\delta} = \Omega(N^\epsilon).$$

Thus it is not possible to do much useful copying in the first phase. It is easy to see that the theorem applies to both the cube (without random ordering of route edges) and the d-shuffle. In the first case, d = 2 and for any edge in the (k+1)th dimension, a suitable set of 2^k nodes consists of all nodes contained in the k-dimensional subcube formed by the lowest order dimensions and containing the start node of the edge. In the second case, d is the (constant) degree and for any edge, a suitable set of d^k nodes consists of all nodes which have a d-ary representation $\alpha\beta$, where $\alpha \in d^k$ and the edge starts at $\beta\gamma$ for some $\beta \in d^{n-k}$ and $\gamma \in d^k$.

Now a suitable scheme for the n-dimensional cube will be described and analysed. A similar scheme can be developed for the d-shuffle. The important change from previous algorithms is that now, instead of carrying a single destination address, packets carry a non-empty set of destination addresses. While a packet is being routed, a node may replicate it on more than one output queue with the address set being partitioned among the replicas. The algorithm will implement partial h-relations which, in this context, are defined to have at most h packets at any one node at the beginning and end of a run, that is no destination occurs in more than h

destination sets. The first phase is the same as previously but, in the second phase, a replica of each packet is sent to each node in its destination set. To implement the copying, each routing node executes the following :

```
for each node α in parallel do begin
  for each packet do begin
    let T be the set of destinations of the packet
    if α ∈ T then begin
      save packet as an arrival
      T := T \ {α}
    end
    let {T_0, . . . . , T_{n-1}} be a partition of T such that
        T_k = {τ ∈ T |
        k is the smallest dimension in which α and τ differ}
    for each non-empty T_k do begin
      queue packet with destination set T_k for dimension k
    end
  end
end
```

From this, it can be seen that packets still follow the shortest route, and a packet is not replicated until a divergence is necessary. The algorithm is both oblivious and non-repeating. It can also be proved to be fast with high probability, as follows :

Theorem (2.10): If the new algorithm is used to realise a multiple read h-relation on an n-dimensional cube, the probability that some packet is delayed in one of the phases by at least Δ time units is less than $(\frac{he n}{2\Delta})^{\Delta} . h2^n$, where e is the base of natural logarithms.

Proof: In the case of the (unchanged) first phase, the result follows from the theorem of Valiant, stated earlier. The rest of the proof refers to the second phase.

Suppose that initially there are a total of m packets with

destination set sizes $K_1, \ldots, K_m$ respectively.

Consider a fixed route B and name the edges of the graph so that $B = e_{i_1} e_{i_2} \ldots e_{i_r}$, where i_j is the number of the dimension traversed by e_{i_j}.

All packets which pass through edge $e_i = (u, v)$ must originate at a node in $L_i = \{w \mid w$ and u are in the same subcube formed by the i lowest order dimensions$\}$. $|L_i| = 2^i$, and so the probability that a packet has an appropriate starting point is $\frac{2^i}{2^n}$.

Consider a packet X. If it starts at a random node in L_i with K destination addresses, the probability that it traverses dimensions $0, \ldots, i$ to u, then goes to v, is less than or equal to the probability that at least one of K independent packets, each with one destination address, would follow a similar route passing through v, which is $K \cdot \frac{1}{2^{i+1}}$.

Hence, if P_{Xi} is the probability that a packet X intersects an edge e_i,

$$\begin{aligned}
&\sum_X P_{Xi} \\
&\leq \sum_{j=1}^{m} \frac{2^i}{2^n} \cdot K_j \cdot \frac{1}{2^{i+1}} \\
&= \frac{1}{2 \cdot 2^n} \sum_{j=1}^{m} K_j \\
&\leq \frac{h}{2} \quad \text{since there are } \leq h2^n \text{ destinations}
\end{aligned}$$

and if P_X is the probability that a packet X intersects at least one edge of B,

$$\begin{aligned}
&\sum_X P_X \\
&\leq \sum_X \sum_{i=1}^{r} P_{Xi_i} \\
&= \sum_{i=1}^{r} \sum_X P_{Xi_i}
\end{aligned}$$

$$\leq \frac{\Gamma h}{2}$$

Now consider a particular packet Y and let B be one of its routes. There are m−1 other packets which may intersect B. Since the scheme is oblivious, m−1 independent trials with probabilities summing to less than $\frac{hn}{2}$ can be considered. By a theorem of Hoeffding [23], the probability of having at least Δ successes is bounded above by $B(\Delta, m-1, \frac{hn}{2(m-1)})$ which is the probability of having at least Δ successes in m−1 independent trials each with success probability $\frac{hn}{2(m-1)}$.

Now, using an inequality due to Chernoff [11] that, for $m > Np$,

$$B(m,N,p) \leq (\frac{Np}{m})^m (\frac{N-Np}{N-m})^{N-m}$$

and observing that $(1+\frac{1}{x})^x < e$ when $x = \frac{N-m}{m-Np}$,

$$B(m,N,p) \leq (\frac{Np}{m})^m . e^{m-Np}$$

So the success probability is bounded by

$$\frac{hn}{2\Delta} e^{\Delta - hn/2} \leq (\frac{hen}{2\Delta})^{\Delta}$$

Since there are at most $h2^n$ packets to consider, the probability that one or more suffers a delay more than Δ is bounded by $h2^n$ times this number.

While the algorithm runs in logarithmic time, there is a further problem which is implicit in the previous description. This is the amount of space needed to represent the address set carried by the packet. Formerly, O(log N) bits were sufficient but now, to achieve full generality, O(N) bits are required. This is liable to become an unacceptable overhead and, unfortunately, it cannot be reduced if any node is to be able to broadcast a packet to any set of nodes it wishes. Here, a few ways are considered for reducing the space if various problem constraints are modified.

Firstly, it may be acceptable to place a bound f(N) on the size of the

address set. Given this, O(f(N) log N) bits are needed. Such a restriction may be reasonable if, for example, only more localised, rather than global, distribution of data is required. Also, there may be time constraints on a processing node which preclude the generation of a large set of destination addresses.

Secondly, a more compressed representation of addresses might be possible, if the arbitrary addressing requirement is relaxed. One simple approach is to allow address representations to be strings of symbols over an alphabet augmented by a special "wild card" symbol * which means that any alphabet symbol may replace * in a string. For example, in a five-dimensional cube, the address 1*00* could mean "all nodes with dimensions 1 and 2 zero and dimension 4 one" or ***** could mean "all nodes". More complicatedly, concise descriptions of certain interesting sets, for example all nodes with prime indices, may be possible.

The key property which any compressed representation must have is that, whenever a set is partitioned (and here the partitioning is dependent on the routing scheme), it must be possible to obtain a concise representation for each set in the partition. Clearly, examples such as the set of primes in some range are highly unlikely to have this property. However, the "wild card" representation does, with respect to the cube routing scheme just described. Occurrences of *s in the destination address can be removed when a routing node makes two copies of a packet, containing a 0 or 1 respectively in the corresponding address position, and forwards them along appropriate edges. A similar representation exists for the d-shuffle.

If all of the original destination sets do have concise representations but the above property does not hold, a further technique might be to dynamically compute the current address set of each packet at each node on its route. This requires a fast algorithm which, given the original destination set and the routing history of a packet, can determine the current address set.

Lastly, the feasibility of a more radical change will be considered.

instead of storing address sets with the packet, they would be stored in the only other possible place, namely at nodes. Because of the distributed nature of the algorithm, the nodes must be those which the relevant packet passes through. In order to store the data at the nodes, it is necessary to modify the scheme so that, instead of one node "broadcasting" a packet, it is assumed that all nodes which want a copy of the packet send a request to the source node.

Then the idea is that, when several requests for the same packet meet at a routing node, it remembers the addresses of the nodes which issue the requests and forwards a single request from itself. When the packet is delivered, it can retrace the paths followed by the requests in reverse order, recovering addresses from the "distributed stack" as it proceeds. Effectively, the whole process takes the form of a multiple write operation followed by a multiple read operation, so further consideration of its use must be postponed until multiple writes have been investigated.

However, note that packets need only carry O(log N) bits of data and, if the network has degree d and routing time O(log N) with high probability, then the extra data stored at any node will require $O(d \log^2 N)$ bits with high probability. So this method does offer a prospect of useful gains, as well as demonstrating an interesting algorithm employing a distributed stack.

2.6.2 Multiple writes

In implementing multiple writes, it is necessary to arbitrate between packets all destined for the same node. This arbitration must be performed during routing, rather than when all competing packets have arrived. As long as it is possible to define an associative arbitration operator on two packets, distributed arbitration can be implemented. All of the common mechanisms, for example "select random winner", "write logical OR of data", "write logical AND of data", are satisfactory in this respect.

The arbitration operator can be applied whenever two packets carrying the same destination address meet at a routing node, resulting in a single surviving packet. If packets are being routed to random destinations, the

number of such arbitratible collisions is necessarily small, since they should be no more probable than any other collisions. The conclusion is that arbitration must take place when packets are travelling to their final destination (and are guaranteed to collide eventually if they are going to the same node). Unfortunately, this proves to be the stumbling block in implementing multiple writes, as the following "inverted" form of theorem (2.9) indicates :

Theorem (2.11): Suppose a random routing network has the property that, for some edge e and some constant $d > 1$, there are at least d^k nodes such that a random route to any of them intersects e with probability $\geq \frac{1}{d^{k+1}}$ for any k such that $1 \leq d^k \leq N$, the graph size. Then there is a multiple write operation which has $\Omega(N^\epsilon)$ completion time if initially each of N^δ destinations has N^ϵ competitors starting at random nodes, for any $\epsilon \leq \delta$ such that $\epsilon+\delta \leq 1$

Proof: Similar to that for theorem (2.9).

This means that it is not possible to extend the random routing scheme to embrace multiple writes. Given that multiple writes extend the power of parallel computers with a globally shared memory, it is perhaps not surprising that they are harder on the model used here. Clearly, it is possible to realise a great many useful multiple write permutations fast, for example the OR of N elements, but certain pathological cases requiring, say, $\Omega(\sqrt{N})$ time prevent the scheme being truly general purpose.

Returning to multiple reads with requests, the same remark applies. If one is content with "acceptable" request patterns then, given a network which has bidirectional routes, it is possible to implement multiple reads fast without increasing packet size. Given the fact that multiple reads increase storage requirements and multiple writes increase time requirements, a routing interpretation of the hierarchy normally associated with idealistic parallel models, that is :

multiple writes

$\geq$ multiple reads

$\geq$ single read/writes

becomes possible.

Chapter 3

Experiments with parallel communication schemes

3.1 Introduction

In the previous chapter, good asymptotic bounds on the run time of parallel communication schemes were obtained. However, the analyses did not yield the tight multiplicative factors on these bounds which are necessary when considering the performance for realistic (i.e. small) numbers of processors. Furthermore, it is often necessary to place restrictions on both the underlying graph, and the routing strategy, in order to achieve the desired analytic result.

The role of this chapter is to rectify these shortcomings by investigating the behaviour of various schemes using serial simulation of the parallel routing algorithms. All of the simulations were programmed in Pascal, and run on a DEC VAX 11/780 computer. The program performed a series of experiments, each measuring interesting statistics for a fixed permutation request on a fixed graph, resulting in a set of distributions from which useful observations could be made.

The "interesting statistics" collected were chosen to reflect the load put on the network under circumstances of worst case (or perhaps, freak) conditions. Thus, it is possible to predict the resources, and indeed redundancy, required if a practical implementation was to be built. The first measure, not surprisingly since it has been of central interest, is the completion time, that is the time at which all packets have reached their destinations. It has been predicted that this time will be moderate with high, indeed asymptotically overwhelming, probability. Secondly, the maximum number of packets simultaneously congregated at one node is considered. This total is made up of packets which have arrived at their final destination and those which are queued for forwarding to neighbouring

nodes. From it, the amount of storage which must be provided with each processor for buffering packets can be estimated. Finally, the maximum queue length at any node is measured. This gives a useful indication of whether bottlenecks are occurring in the network. In the context here, a high value means that the routing algorithm is unwisely favouring a particular node, or even an edge, for many routes.

In general, when a computer program is being tested for correctness, it has a very large, possibly infinite, number of possible inputs. Therefore, exhaustive testing is not possible and, as an alternative, extensive research has been conducted into formal proof techniques over the last twenty years. A similar, but rather different, problem is associated with the parallel randomised algorithms being simulated here. It is assumed that the programming is correct, and that packets reach their proper destinations, but it is necessary to characterise various complexity measures precisely and, in general, these might vary with each different input.

Traditionally, complexity theory copes with this problem by considering either the worst case or expected value of a measure over all inputs. Two difficulties with such an approach arise here. Firstly, the inamenability of the algorithms to mathematical analysis has led to this measurement by "running the program" and it is not usually possible to identify a set of worst case inputs. Secondly, because of the randomised nature of the algorithms, the complexity measures vary over each possible input and it is necessary to consider worst case or expected values of distributions of worst case or expected values.

The second difficulty is handled by repeating an experiment many times within a particular simulation so that a close approximation to the actual distribution is obtained by considering an appropriately large sample size. In the results presented here, the distribution of a measure will generally be characterised by its mean, variance, and maximum value. The maximum value is, of course, a less robust measure than the other two in that the observed value will generally be dependent on the sample size chosen. It is included here in order to give some information about the

range of the observed distribution since space restrictions preclude the tabulation of all observations.

The first difficulty, while seemingly insurmountable without enumeration of all inputs, that is all relations, is luckily dealt with by the observation that many of the algorithms have a remarkable and unusual property :

> **Definition (3.1):** An algorithm is <u>testable</u> if, for all inputs, its behaviour, as reflected by some set of complexity measures, is the same.

In particular, randomised algorithms have the same distributions of the complexity measures for all inputs. Clearly, there is no analogous concept for program proving, except in the case where a program ignores its input.

Thus, by considering only one particular input, which can be chosen to be as trivial as possible, positive statements about program behaviour can be made. Note that this powerful technique should not be confused with the practice of running a program on a suspected "bad input", which gives no general information.

Unfortunately, not all of the algorithms described so far, or to be described, are testable. In these cases, it is sometimes possible to obtain results by identifying phases with calculable worst case inputs and combining these with other testable phases in a mixed simulation.

The form of the simulating program is, therefore, as follows :

```
for i := 1 to sample size do begin
    simulate parallel routing algorithm for identity relation
    store completion time, max. node population, max. queue size
end
analyse statistical distributions
```

The relations simulated are, in fact, usually total permutations. It is fairly obvious that :

Lemma (3.2): Both phases of the standard randomised routing algorithm for total permutations are testable.

Proof: In the first phase, one packet is sent from each node to a random destination: this is independent of the input permutation. In the second phase, one packet is sent to each node from a random source: this is similarly independent of the input, as long as each source with more than one packet transmits them in a random order.

The lemma easily generalises to the case of total h-relations for fixed h and indeed to each class of partial h-relations with fixed size initial and final packet distributions. Clearly, there is no testable algorithm which implements all h-relations, so attention is primarily restricted here to total 1-relations, that is permutations. However, the effect of increasing h is investigated for certain fixed graphs.

In the simulation, the routing of packets through the processor network is implemented slightly differently from the way described earlier. There, each packet carried a destination address and intermediate nodes deduced from this which edge to forward the packet along. Here, the source node precomputes the sequence of edges which the packet will follow. This means, in general, that forwarding nodes do significantly less computing at the expense of increased packet size and increased initial computation. In a practical system, this may be desirable.

Sometimes, precomputation of routes can lead to optimisations. For example, consider a 2-shuffle graph in which a packet has to be routed from 00101 to 10111. The normal shifting algorithm would choose the path

00101 -> 01011 -> 10110 -> 01101 -> 11011 -> 10111

whereas an optimising source node could select

00101 -> 01011 -> 10111

The d-shuffle graph also has the pleasant feature that packet size need not be increased to implement this scheme, since the destination size is never smaller than an edge sequence description. Conversely, of course, an

algorithm might be speeded up by delaying the choice of route if it was non-oblivious.

The simulation of the algorithm proceeds as a sequence of parallel computation steps until all packets have arrived. At each step, a packet is removed from each non-empty queue at each node, and transferred to either an arrivals set or a forwarding queue at the next node on its route.

In succeeding sections of this chapter, experimental results are presented and discussed. Firstly, n-dimensional cubes and d-shuffles, both analysed previously, are examined and then several other graphs which have attracted interest in connection with parallel computation are introduced. A comparison of all these graphs as suitable frameworks for parallel computation schemes follows. Then, variations of routing strategy, number of queues, queueing discipline, and pipelining of phases, are considered. These are mainly motivated by various restrictions necessary to obtain analytic proofs. All of these experiments involve implementation of permutations, then examples of more general h-relations are investigated briefly. Finally, some points of experimental method are checked, namely the suitability of the random number generator used, the correctness of the testability assumption, and the choice of sample size.

3.2 The n-dimensional cube

In this section, the realisation of permutations on cubes of size 2^2, 2^3, ..., 2^{12} is considered. The standard two-phase routing scheme is employed, with the second phase starting after the first is complete. In each phase, all packets traverse edges between source and destination in a random order of dimensions. Queues are organised on a first-in first-out basis, apart from the randomisation of queues between phases to ensure testability.

The experimental results are presented here using statistical summaries. Tabulation of individual results will only be employed when it is convenient to compare the effect of different algorithms visually.

No. dimensions	Mean	Variance	Maximum
2	1.69	0.22	2
3	2.72	0.28	4
4	3.83	0.30	6
5	4.91	0.36	7
6	6.04	0.31	8
7	7.06	0.31	9
8	8.15	0.32	10
9	9.19	0.30	11
10	10.27	0.32	12
11	11.28	0.26	13
12	12.30	0.26	14

Table 3-1: Time for 1st phase on cube

No. dimensions	Mean	Variance	Maximum
2	1.80	0.39	3
3	2.92	0.42	5
4	4.04	0.42	6
5	5.11	0.42	8
6	6.25	0.39	8
7	7.24	0.35	10
8	8.36	0.40	11
9	9.39	0.39	12
10	10.43	0.35	13
11	11.45	0.32	13
12	12.46	0.32	15

Table 3-2: Time for 2nd phase on cube

No. dimensions	Mean	Variance	Maximum
2	3.48	1.13	5
3	5.65	1.15	8
4	7.87	1.04	11
5	10.02	1.13	15
6	12.29	0.83	15
7	14.30	0.86	17
8	16.51	0.83	19
9	18.58	0.72	22
10	20.71	0.75	24
11	22.73	0.63	26
12	24.76	0.59	28

Table 3-3: Total time for both phases on cube

The completion time results provide encouraging evidence that the cube

routing algorithm is extremely efficient. They indicate that the mean completion time for each phase can be expressed in the form $(1+\epsilon)n-\alpha$, where n is the number of dimensions, ϵ is a phase-independent constant close to zero, and α is a phase-dependent constant in the range $(0, 0.5)$, larger for the first phase than the second. The mean completion time of a routing is therefore $2(1+\epsilon)n-(\alpha_1+\alpha_2)$ and, estimating the constants from the numerical results, this time is upper bounded by $2.2n$.

The relationship $(1+\epsilon)n-\alpha$ corresponds reasonably well with an intuitive view of how the cube algorithm should work. Considering the packet which has the longest delivery time, it is reasonable to assume that it is traversing all n dimensions. Then its journey time may be obtained by totalling the number of time units in which it appears at each node en route. Given a very low probability of collisions, the duration of each stay should average $1+\epsilon$ for a small ϵ. The constant α reflects experiments in which no maximum route occurs, the fact that collision probabilities reduce as the worst case packet proceeds, and the difference between phases, in that queues have size one uniformly at the start of the first phase. The first two reasons would suggest that α should in fact be a very slowly growing function of n rather than a constant but, within the constraints of experimental error and feasible sizes of n, it was not possible to verify this. Later, experimental results from other graphs will provide more intuition.

Of course, the mere fact that the mean completion time is good does not indicate that the algorithm performs well almost always. However, a consideration of the variance indicates that the distribution of completion time is sharply peaked around the mean. In both phases, the variance is smaller than 0.5, and does not appear to depend on the size of the cube used. The slightly smaller variance for the first phase reflects the slightly more predictable nature of it, given that it always starts with one packet at each node at a particular time interval.

The variance of the total run time decreases with n, and is below one for cubes with more than 32 nodes. The decrease is double the decrease in the covariance of the first and second phase run times. Since the second

phase is always run on input provided by the first phase, these two times are clearly not independent. Indeed, because the identity permutation is being implemented, the second phase routes are simply random reorderings of the first phase routes. However, as the cube size increases, the redistribution of packets during a phase becomes increasingly complex, meaning that the impact of initial coupling between phases is lessened and hence independence is more nearly achieved. In fact, for the 4096 node cube, the run time covariance is reduced to 0.0 (although this does not imply independence).

Finally, examining the extremes of the run time distribution, it can be noted that the maximum value never exceeded n+2 for the first phase or n+3 for the second phase. Apart from the two dimensional cube, which has a very restricted distribution, the maximum value never account for more than 2% (more typically < 1%) of the experiments. Therefore, regarding the maximum as a worst case complexity measure, the algorithm in practice is fast always.

From these empirical considerations, a probable upper bound on completion time, with overwhelming probability, is $U(n) = (1+\epsilon)n+c$, where ϵ is less than 0.1 and c = 3 or 4. Compare this with $U(n) = (1+K)n$, where K is "sufficiently large", for example 4.5, proved analytically for the cube.

No. dimensions	Mean	Variance	Maximum
2	2.24	0.28	4
3	2.82	0.51	6
4	3.43	0.51	7
5	4.05	0.61	7
6	4.47	0.43	6
7	4.94	0.51	8
8	5.35	0.43	8
9	5.71	0.48	9
10	6.17	0.39	8
11	6.55	0.38	9
12	6.91	0.48	10

Table 3-4: Max node populations for 1st phase on cube

No. dimensions	Mean	Variance	Maximum
2	2.19	0.26	4
3	2.70	0.51	6
4	3.26	0.49	7
5	3.85	0.55	7
6	4.23	0.37	6
7	4.67	0.46	8
8	5.10	0.39	7
9	5.45	0.40	8
10	5.88	0.42	8
11	6.23	0.33	8
12	6.56	0.38	10

Table 3–5: Max node populations for 2nd phase on cube

From the tables of maximum node populations, it can be seen that the amount of local storage needed in nodes is relatively modest. For both phases, the mean increases linearly with the number of dimensions, in two stages. Up to five dimensions, the mean is approximately $0.6n+c_1$; above five dimensions, it is approximately $0.4n+c_2$. In each stage, the first phase mean grows slightly faster than the second.

Recalling that the node population cosists of packets on the n queues for other nodes together with packets which have arrived at their destination, some observations can be made about this relationship. At first, it is tempting to argue that its linear nature is caused by the fact that the number of queues, and hence opportunities for packet storage, increases linearly. However, assuming that the routing scheme is reasonably balanced as desired, the expected traffic through one node is one packet per time interval and does not increase with n. Although the cube provides an out degree of n, the perceived advantage is to offer more opportunities for disjoint routes, rather than have all n out edges at a node in use simultaneously. Indeed, consideration of fixed degree graphs in the next section will reveal that the linear relationship still holds.

Also, the number of arriving packets does not increase with n for the second phase, where it is always precisely one per node. In the first phase, the contribution from arriving packets will be larger, but only

significant when packets are arriving while routing is still proceeding on a significant scale. This can happen for the cube routing algorithm because routes do not have uniform length, and the slightly higher mean maximum node populations for the first phase may be explained by this phenomenon.

More detailed simulations indicate that, in the first phase, the time interval at which the maximum node population occurs is fairly evenly distributed throughout the run. However, in the second phase, the maximum occurs in the initial distribution of packets in a large number of experiments. Given that both phases have some point at which packets are uniformly distributed about the network, it appears that maximum node populations will occur at the point at which packets are most randomly distributed, in the absence of imbalances leading to bottlenecks. In the first phase, the combination of routing queues and arrivals at their random destination can increase this maximum. Intuitively then, maximum node population is determined by the number of packets and number of nodes. Empirically, the hypothesis advanced now, and investigated further in later sections, is that mean maximum node population for permutations is logarithmically related to the number of packets.

Since the node population distribution between phases can be characterised precisely, it is instructive to examine this distribution in order to assess how much queueing problems affect node population. In a network with N nodes, the relevant distribution is a multinomial distribution with N equiprobable outcomes and N trials, and it is necessary to examine the distribution of the maximum number of successes atributed to one outcome. Analytically, this involves twofold problems. Firstly, estimating the distribution of the maximum of a collection of random variables is usually hard. Nair [37] tackles the problem of the extremes of a distribution (with significance testing in mind) but here, for example, such features as the mean are of interest. Secondly, estimating features of the multinomial distribution is also hard, apart from a few isolated properties such as the marginal distributions which are binomial [17]. Johnson and Young [26] investigate approximations to the multinomial distributions but, again, these are concerned with the upper tail. Combining the two problems, David and

Barton [15] do investigate approximations to the largest multinomial frequency but these are best for more extreme values of the distribution.

Therefore, continuing in the spirit of this chapter, the distribution is examined using Monte Carlo methods. This type of technique has been suggested as an aid when investigating more intractible distributions, for example by Good in [21]. The table gives a summary of the distribution of maximum successes observed during 500 tests of a multinomial distribution with N outcomes and N trials for $N = 2^2, \ldots, 2^{12}$.

N	Mean	Variance	Maximum
4	2.10	0.34	4
8	2.66	0.53	5
16	3.07	0.57	6
32	3.54	0.55	6
64	3.95	0.68	7
128	4.34	0.49	7
256	4.78	0.50	7
512	5.11	0.54	9
1024	5.56	0.51	9
2048	5.86	0.51	8
4096	6.22	0.43	9

Table 3-6: Distribution of maximum number of multinomial successes

It can be seen that the distribution of maximum node populations for the cube is close to those for the pure random distribution, but growing with the dimensionality of the cube at a slightly larger rate. This indicates that populations due to queueing are not having a particularly large effect.

The apparently different rate of growth for smaller cubes is due to the fact that an insufficient number of node populations are sampled during a run (for example, less than 100 for a four dimensional cube) to reflect the true tails of the worst case node population. The mean is therefore artificially low. Approximately, the correct mean is $0.4n+2$.

The variance of the distribution is less than 0.5, in almost all cases, and does not increase with n. Therefore, the space requirement also has a

sharp distribution. The worst maximum node population, which normally occurs in less than 5% of the experiments, is never more than the mean plus 3.5. It is less than n for all n greater than eight.

No. dimensions	Mean	Variance	Maximum
2	1.00	0.00	1
3	1.15	0.13	3
4	1.50	0.27	3
5	1.93	0.17	3
6	2.10	0.10	4
7	2.31	0.22	4
8	2.56	0.27	4
9	2.82	0.22	5
10	3.02	0.12	5
11	3.10	0.10	5
12	3.16	0.14	5

Table 3-7: Max queue length for 1st phase on cube

No. dimensions	Mean	Variance	Maximum
2	1.24	0.19	2
3	1.57	0.29	3
4	1.97	0.20	4
5	2.16	0.18	4
6	2.31	0.24	4
7	2.49	0.27	4
8	2.74	0.27	4
9	2.98	0.17	4
10	3.07	0.09	6
11	3.16	0.14	5
12	3.23	0.18	5

Table 3-8: Max queue length for 2nd phase on cube

In the previous discussion, it has been deduced that not much congestion occurs and hence queue lengths are normally at most one. The tabulated results show that even the worst case queue length during a run is normally small. For both phases, the growth of the mean is smaller than the number of dimensions, n. Because the range of mean values obtained for n between 2 and 12 is so small, it is not reasonable to place any more detailed functional interpretation on the values, which have a higher relative

error than usual.

Having observed that the maximum node population grows linearly with n, clearly the mean maximum queue length grows no faster. Indeed, since routes of packets meeting at a node are random, they will be randomly distributed among the n queues. Just as the entire network appears to have maximum node populations when packets are randomly distributed, the behaviour of a particular node can be viewed in a similar way, except that it has n trials with success probability $\frac{1}{n}$, rather than 2^n trials with success probability $\frac{1}{2^n}$. Given the previous empirical result, the expected mean of the maximum queue length would be O(log n), which is consistent with the observations.

The mean for the second phase is always higher than that for the first. It can be seen that, for each $n \geq 4$, the second phase mean for n dimensions is approximately equal to the first phase mean for n+1 dimensions. In view of the remark made when considering completion time, that the first stage of the first phase was different in the sense that packets had a non-random uniform distribution, this might suggest that the maximum queue length was, in fact, related to route length rather than number of routes. However, more detailed simulations indicate that, in most experiments, a maximum queue appears after the first step in the first phase and initially in the second phase. This indicates that path length is not relevant. If this "correlation" is indeed significant, then the increasing degree is a more likely cause.

The difference between initial distributions in the two phases does account for the (decreasing) difference between the means. The increased mean in the second phase is due to the initial queues, which are unnatural in the sense that they do not evolve during routing. As n grows, bad routing queues are more likely than bad initial queues, and bad queues are less likely because more queues are available to share the load at bad nodes.

All the values obtained for maximum queue length are contained in a small range and, for each n, are tightly grouped around the mean with

small variance. On only one occasion, a queue of length six occurred; in the same phase of the set of experiments, no other queue length exceeded four. No serious effect on completion time resulted from it.

This concludes the examination of the standard routing algorithm for the n-dimensional cube. It has been seen that the scheme is sufficiently balanced that collisions are relatively infrequent. The expected run time is close to the ideal and the number of buffers required for queueing at any node is less than the number of out edges. Of course, the major disadvantage of the cube is the fact that it has degree logarithmic in the number of nodes. In the next section, the d-shuffle, which can have constant degree but retain logarithmic diameter, will be considered experimentally.

3.3 The d-shuffle

The experiments in this section involve the realisation of permutations on d-shuffle graphs with degree $2, 3, \ldots, 10$ and up to 4096 nodes, using the standard two-phase routing scheme. In each phase, all packets traverse exactly $\log_d N$ edges, where N is the graph size, between source and destination. Queues are organised on a first-in first-out basis, with randomisation between phases.

Altogether, 39 different graphs are considered. The usual measures are tabulated, with graphs grouped by degree.

Considering the whole set of graphs, the most interesting fact to emerge is that the 3-shuffle graph can almost match the completion times of the correspondingly sized cube, and that higher degree d-shuffles all beat the cube. Since a severe restriction on connectivity has been made, this is a very encouraging result. It tends to confirm the expectation, expressed earlier, that cube edges are unused most of the time. The d-shuffle routing must be making far greater use of edges, while not significantly increasing the probability of routes colliding.

Comparing graphs with differing degrees, it can be seen that the

Degree	Diameter	Size	Mean	Variance	Maximum
2	2	4	2.45	0.25	3
	3	8	4.14	0.32	6
	4	16	5.86	0.51	9
	5	32	7.55	0.54	10
	6	64	9.41	0.59	12
	7	128	11.33	0.71	15
	8	256	13.16	0.76	16
	9	512	15.19	0.99	22
	10	1024	17.11	0.99	25
	11	2048	19.00	0.89	24
	12	4096	20.97	1.01	27
3	2	9	2.70	0.25	4
	3	27	4.39	0.32	7
	4	81	6.38	0.36	9
	5	243	8.21	0.38	11
	6	729	10.14	0.40	13
	7	2187	12.10	0.43	15
4	2	16	2.82	0.23	4
	3	64	4.62	0.34	6
	4	256	6.47	0.31	8
	5	1024	8.46	0.37	11
	6	4096	10.38	0.31	13
5	2	25	2.94	0.18	4
	3	125	4.73	0.31	7
	4	625	6.63	0.36	9
	5	3125	8.55	0.35	11
6	2	36	3.01	0.15	4
	3	216	4.86	0.28	7
	4	1296	6.76	0.33	9
7	2	49	3.03	0.15	4
	3	343	4.89	0.22	6
	4	2401	6.75	0.30	9
8	2	64	3.06	0.12	4
	3	512	4.92	0.17	6
	4	4096	6.85	0.22	8
9	2	81	3.11	0.15	6
	3	729	4.95	0.15	6
10	2	100	3.10	0.11	4
	3	1000	5.00	0.10	6

Table 3-9: Time for 1st phase on d-shuffle

Degree	Diameter	Size	Mean	Variance	Maximum
2	2	4	2.45	0.25	3
	3	8	4.23	0.40	7
	4	16	5.99	0.54	9
	5	32	7.92	0.68	13
	6	64	9.84	0.69	12
	7	128	11.83	0.90	19
	8	256	13.81	0.86	18
	9	512	15.77	0.95	20
	10	1024	17.66	0.83	21
	11	2048	19.60	0.85	24
	12	4096	21.63	0.87	26
3	2	9	2.70	0.25	4
	3	27	4.58	0.34	6
	4	81	6.56	0.42	9
	5	243	8.54	0.43	11
	6	729	10.48	0.43	14
	7	2187	12.42	0.47	15
4	2	16	2.85	0.24	4
	3	64	4.79	0.35	7
	4	256	6.77	0.33	9
	5	1024	8.66	0.35	11
	6	4096	10.64	0.35	13
5	2	25	2.95	0.19	4
	3	125	4.90	0.24	7
	4	625	6.86	0.31	9
	5	3125	8.79	0.32	11
6	2	36	2.99	0.16	5
	3	216	5.01	0.19	7
	4	1296	6.95	0.19	9
7	2	49	3.05	0.16	6
	3	343	5.05	0.15	6
	4	2401	7.01	0.20	9
8	2	64	3.04	0.10	4
	3	512	5.06	0.13	7
	4	4096	7.04	0.13	9
9	2	81	3.10	0.15	4
	3	729	5.11	0.13	7
10	2	100	3.10	0.15	4
	3	1000	5.09	0.11	6

Table 3-10: Time for 2nd phase on d-shuffle

Degree	Diameter	Size	Mean	Variance	Maximum
2	2	4	4.89	0.50	6
	3	8	8.37	0.85	13
	4	16	11.85	1.18	16
	5	32	15.48	1.32	20
	6	64	19.25	1.49	24
	7	128	23.17	1.67	31
	8	256	26.97	1.68	32
	9	512	30.97	1.88	38
	10	1024	34.77	1.79	42
	11	2048	38.60	1.70	44
	12	4096	42.61	1.82	48
3	2	9	5.40	0.50	7
	3	27	8.97	0.73	13
	4	81	12.95	0.80	16
	5	243	16.75	0.79	20
	6	729	20.63	0.78	24
	7	2187	24.52	0.86	27
4	2	16	5.67	0.47	7
	3	64	9.41	0.71	12
	4	256	13.24	0.68	17
	5	1024	17.13	0.76	20
	6	4096	21.02	0.67	23
5	2	25	5.89	0.38	8
	3	125	9.63	0.56	12
	4	625	13.49	0.71	17
	5	3125	17.34	0.67	20
6	2	36	6.00	0.31	8
	3	216	9.87	0.47	13
	4	1296	13.71	0.53	16
7	2	49	6.08	0.33	10
	3	343	9.94	0.37	12
	4	2401	13.77	0.52	16
8	2	64	6.10	0.22	8
	3	512	9.97	0.30	12
	4	4096	13.89	0.36	16
9	2	81	6.21	0.28	9
	3	729	10.06	0.27	12
10	2	100	6.20	0.25	9
	3	1000	10.09	0.20	12

Table 3-11: Total time for both phases on d-shuffle

completion time always grows at slightly less than double the rate of increase in diameter. It is pleasant that the factor of two is approximately independent of the degree, but it compares with a factor of one for the cube with corresponding diameter. This is not unexpected because, on average, the routing algorithm for the d-shuffle sends packets twice as far as that for the cube. While a worst case packet will travel the same distance on either graph, there is half as much total traffic to interfere with it. Further, "long distance" packets are sufficiently rare on the cube that the probability of them being involved in an occasional collision is very small. For the d-shuffle however, the completion time is always composed of the diameter added to the maximum number of delays incurred by any one packet.

To estimate the worst case journey time for a packet, its journey length must therefore be taken into consideration. This is trivial for the d-shuffle, and the results obtained indicate that a bad (i.e. frequently colliding) packet is delayed on average for one time interval at each node en route. Applying this observation, which holds for all degrees, to the cube, a bad packet with the mean journey length $\frac{n}{2}$ would still complete inside time n. As a working hypothesis, it is suggested that the expected completion time grows at about double the rate of increase in expected route length on a randomised routing scheme which is reasonably balanced.

A closer study of the d-shuffle results shows variations between degrees and phases for any fixed diameter. As the degree increases, the completion time increases slightly. This increase is largest by far when going from degree two to three. The cause is that the total number of packets is of course increasing with d. Hence, the number of opportunities for collisions at a particular node is increased. As the inter-packet collision probability is small given a balanced scheme, only a small contribution to completion time is expected from this fact.

The completion time for the second phase is always slightly greater than that for the first, with the difference becoming smaller with increasing degree. For sufficiently large graph sizes (more than 200 nodes) the

difference is constant as diameter increases. As usual, this can be explained by the initial distributions at the start of the second phase. With lower degree, there are fewer queues and so a bad packet suffers an additional inital queueing delay if it starts at a relatively highly populated node.

Looking at the whole distribution of completion time for d-shuffle graphs, it can be seen that those for degrees two and three are less sharply peaked than those for the cube. However, for higher degrees, the distributions are very closely packed. This is an indication that, when connectivity is severely restricted, the much-feared bottlenecks do occur, albeit rarely. Even for larger sizes of the 2-shuffle, where the maximum observed value is as much as five or six greater than the mean, the upper tail is shallow with never more than 2% of the completion times more than three greater than the mean. This is reflected in the variance, which never exceeds one in any phase. It is noticeable that the variance apparently increases with n and flattens out when n is large enough. Again, this is due to experiments on small size graphs not being large enough to reflect probability distributions of events accurately; in this case, extremes of queueing do not occur frequently enough.

The maximum value for completion time of the two phases is never more than 4n+3 and, for most graphs, is less than or equal to 4n. Considering phases independently, completion times of up to 2n+5 occurred, but in none of the experiments did two worst case phases occur in the same run. This indicates that bad packets are probably independent between phases. Confirming this, the covariance of the phase completion times is always small, and tends towards zero as the diameter increases.

So, for any d-shuffle graph with degree larger than four, an improved run time distribution is achieved over the cube. The next results will indicate whether this improvement is achieved at the expense of resources at nodes.

The maximum node population tables show that a small amount of storage

Degree	Diameter	Size	Mean	Variance	Maximum
2	2	4	2.16	0.28	4
	3	8	2.70	0.50	6
	4	16	3.27	0.41	5
	5	32	3.65	0.44	7
	6	64	4.17	0.47	7
	7	128	4.59	0.43	7
	8	256	5.02	0.53	8
	9	512	5.46	0.46	8
	10	1024	5.82	0.42	8
	11	2048	6.21	0.44	10
	12	4096	6.62	0.48	9
3	2	9	2.80	0.42	5
	3	27	3.52	0.46	7
	4	81	4.35	0.47	7
	5	243	4.93	0.50	8
	6	729	5.54	0.42	9
	7	2187	6.13	0.36	8
4	2	16	3.24	0.39	5
	3	64	4.15	0.42	7
	4	256	4.95	0.47	8
	5	1024	5.72	0.46	8
	6	4096	6.45	0.42	10
5	2	25	3.58	0.49	6
	3	125	4.43	0.45	7
	4	625	5.41	0.40	8
	5	3125	6.36	0.42	9
6	2	36	3.78	0.52	6
	3	216	4.89	0.58	8
	4	1296	5.83	0.44	8
7	2	49	4.01	0.45	7
	3	343	5.14	0.45	8
	4	2401	6.19	0.37	9
8	2	64	4.17	0.49	7
	3	512	5.42	0.38	8
	4	4096	6.42	0.36	10
9	2	81	4.36	0.49	7
	3	729	5.58	0.49	9
10	2	100	4.48	0.47	7
	3	1000	5.79	0.47	9

Table 3-12: Max node population for 1st phase on d-shuffle

Degree	Diameter	Size	Mean	Variance	Maximum
2	2	4	2.17	0.29	4
	3	8	2.71	0.51	6
	4	16	3.26	0.41	5
	5	32	3.67	0.45	7
	6	64	4.16	0.47	7
	7	128	4.61	0.51	8
	8	256	5.01	0.56	8
	9	512	5.47	0.47	8
	10	1024	5.77	0.47	8
	11	2048	6.18	0.51	10
	12	4096	6.55	0.47	10
3	2	9	2.78	0.43	5
	3	27	3.49	0.47	6
	4	81	4.31	0.47	7
	5	243	4.94	0.51	8
	6	729	5.55	0.48	9
	7	2187	6.06	0.42	8
4	2	16	3.18	0.41	5
	3	64	4.12	0.49	7
	4	256	4.92	0.51	8
	5	1024	5.67	0.47	8
	6	4096	6.40	0.39	10
5	2	25	3.53	0.49	6
	3	125	4.43	0.45	7
	4	625	5.39	0.41	8
	5	3125	6.26	0.41	9
6	2	36	3.68	0.50	6
	3	216	4.86	0.58	8
	4	1296	5.81	0.42	8
7	2	49	3.88	0.49	7
	3	343	5.11	0.47	8
	4	2401	6.11	0.44	8
8	2	64	4.10	0.49	7
	3	512	5.32	0.39	7
	4	4096	6.38	0.38	10
9	2	81	4.28	0.49	7
	3	729	5.51	0.52	9
10	2	100	4.33	0.45	7
	3	1000	5.66	0.51	9

Table 3-13: Max node population for 2nd phase on d-shuffle

is still sufficient at each node when a d-shuffle graph is used. The interesting feature is that, for a particular graph size, the maximum node population is approximately the same, independent of both degree and diameter. Indeed, the results are approximately the same as those observed for the cube. Thus, they give substantial supporting evidence to the hypothesis that maximum node population is logarithmically related to graph size for any balanced random routing scheme.

Also, it can be noted that there is very little difference between the two phases. This is consistent with the view that arrivals do not influence this measure a great deal. Since all packets travel a uniform distance, it is not expected that a cluster of arrivals in the first phase will have gathered in time to have an impact on packets in transit.

A slight decrease in mean population between d-shuffle graphs with increasing degree (in particular between the 2-shuffle and the others) and the same size can be observed. This might be attributed to either higher degree or lower diameter. Since the second phase results for the 2-shuffle and the cube are so remarkably consistent, it is reasonable to assume that the degree is not significant. Also, this indicates that the diameter, in the context of path length or completion time, is not significant. In fact, closer study of when maximum node populations occur indicate that, for higher degree d-shuffles, random distribution of packets is more frequently the cause rather than routing problems. Therefore, the gain in performance over the cube may be attributed to slightly better balance, as well as the more obvious reduction in path length. This, of course, assumes that the number of queues at a node is not a great handicap.

In all the experiments, the distribution is sharply peaked around the mean and the variances are similar. As was observed for the cube, the worst case maximum node population never exceeds the mean plus 3.5. Of course, the number of buffers required is no longer less than one per out edge, but that is because out edges (presumably more expensive) are being saved.

Degree	Diameter	Size	Mean	Variance	Maximum
2	2	4	1.45	0.25	2
	3	8	1.98	0.14	3
	4	16	2.23	0.20	4
	5	32	2.52	0.30	4
	6	64	2.97	0.28	5
	7	128	3.34	0.28	6
	8	256	3.72	0.37	6
	9	512	4.16	0.39	7
	10	1024	4.51	0.41	7
	11	2048	4.93	0.44	7
	12	4096	5.33	0.42	8
3	2	9	1.70	0.25	3
	3	27	2.21	0.18	4
	4	81	2.84	0.26	5
	5	243	3.24	0.23	6
	6	729	3.72	0.35	6
	7	2187	4.30	0.28	7
4	2	16	1.82	0.23	3
	3	64	2.42	0.26	4
	4	256	3.07	0.15	5
	5	1024	3.57	0.34	6
	6	4096	4.20	0.22	7
5	2	25	1.94	0.18	3
	3	125	2.54	0.28	4
	4	625	3.20	0.18	5
	5	3125	3.85	0.27	6
6	2	36	2.01	0.15	3
	3	216	2.72	0.29	5
	4	1296	3.25	0.20	5
7	2	49	2.03	0.15	3
	3	343	2.78	0.23	4
	4	2401	3.34	0.24	5
8	2	64	2.06	0.12	3
	3	512	2.84	0.18	4
	4	4096	3.40	0.26	5
9	2	81	2.11	0.15	5
	3	729	2.88	0.15	4
10	2	100	2.10	0.11	3
	3	1000	2.96	0.13	4

Table 3–14: Max queue size for 1st phase on d-shuffle

Degree	Diameter	Size	Mean	Variance	Maximum
2	2	4	1.45	0.25	2
	3	8	2.08	0.25	4
	4	16	2.45	0.33	4
	5	32	2.92	0.29	5
	6	64	3.33	0.37	6
	7	128	3.75	0.47	7
	8	256	4.16	0.40	7
	9	512	4.54	0.42	7
	10	1024	4.92	0.43	8
	11	2048	5.31	0.47	8
	12	4096	5.64	0.50	10
3	2	9	1.70	0.29	3
	3	27	2.34	0.28	4
	4	81	2.97	0.30	5
	5	243	3.44	0.32	5
	6	729	4.05	0.36	7
	7	2187	4.51	0.33	7
4	2	16	1.85	0.24	3
	3	64	2.49	0.31	5
	4	256	3.21	0.24	5
	5	1024	3.82	0.29	5
	6	4096	4.32	0.23	6
5	2	25	1.95	0.19	3
	3	125	2.63	0.28	4
	4	625	3.27	0.22	5
	5	3125	3.96	0.21	7
6	2	36	1.99	0.18	4
	3	216	2.72	0.29	5
	4	1296	3.36	0.27	5
7	2	49	2.05	0.16	5
	3	343	2.80	0.26	4
	4	2401	3.44	0.29	6
8	2	64	2.04	0.10	3
	3	512	2.90	0.19	5
	4	4096	3.51	0.28	5
9	2	81	2.10	0.12	3
	3	729	2.92	0.20	4
10	2	100	2.10	0.14	4
	3	1000	2.97	0.13	4

Table 3-15: Max queue length for 2nd phase on d-shuffle

As would be expected, the distributions of maximum queue length for the d-shuffle graphs differ from that for the cube. It has been shown that maximum node populations do not vary significantly with type of graph and so, when the degree and hence number of queues is reduced, queue lengths must be increased. Of course, if completion times and storage requirements are still acceptable, maximum queue lengths need not be a worry but it is interesting to examine them to see whether packets flow fairly freely in the network.

For any fixed degree, the mean maximum queue length can be seen to grow linearly with the diameter (for those d-shuffle graphs tested with a reasonable number of different diameters). However, the multiplicative factor varies with the degree. It is noticeably smaller for the 2-shuffle and appears to decrease more slowly for higher degrees. This behaviour corresponds to the intuition obtained from consideration of queue lengths in terms of individual nodes, in which the node population (of size K log d^n in the worst case) is randomly distributed among d queues. For fixed d, one would expect the queue lengths, including the worst queue length, to grow with n, which is the diameter. Since the node population grows with log d for fixed n and the number of queues grows with d then, having observed previously that a situation with O(d) trials and success probability $\frac{1}{d}$ would have O(log d) mean maximum queue length, it can be deduced that the mean here grows sublogarithmically in d. Combining these remarks, an approximate relationship for all d-shuffles would be that the mean maximum queue length is O(n log d) = O(log N), where N is the graph size. Note the difference from the cube, where the increasing degree with graph size leads to an apparent O(log log N) mean.

Queue lengths in the second phase are always worse than the second, with the difference being more acute for small degrees. As in the case of the cube, the state of the queues at the start of the second phase is responsible. While worst case node populations at this point would be the same for similarly sized graphs, the packets are distributed amongst more queues as the degree increases.

Considering the whole distribution, values extend over a very limited range, with the maximum never exceeding the mean by more than five (and that in one extreme case). In fact, the maximum is always less than lg N for all d-shuffle graphs examined. These worst cases occur for degree two; for higher degrees the distribution becomes increasingly sharp with the degree ten results showing that queueing is probably becoming negligible.

It can be noted that, as with the cube, the queue length results (indeed the whole distribution) for the 2-shuffle have the property that those for the first phase on a given graph size match those for the second phase on a graph with half the size. Having failed to establish a significant property shared by the cube and the 2-shuffle, the best explanation which can be offered here is that the relationship is coincidental, possibly due to fortuitous constant factors in logarithmic terms.

This concludes the initial examination of the d-shuffle family of graphs.

3.4 The cube-connected cycle graph

In the next few sections, some previously unconsidered graphs will be examined. All of these graphs have attracted interest because of being advantageous for the implementation of parallel algorithms, and here their amenability to routing will be considered.

The cube-connected cycle graph (CCC) is an attempt to harness the communication complexity of the cube, while having constant degree. It was first proposed by Preparata and Vuillemin [40]. Galil and Paul have proposed it as a routing network in their general purpose parallel computer [19, 35]. A CCC with size $s2^s$ is defined as follows :

Vertices.

$\{(c,d) \mid c \in \{0,1\}^s,\ 0 \leq d \leq s-1\}$

Edges, for each $c = c_{s-1} \ldots c_1 c_0$ and d.

(i) $[(c_{s-1} \ldots c_d \ldots c_0, d)\ ,\ (c_{s-1} \ldots \bar{c}_d \ldots c_0, d)]$

(ii) $[(c,d)\ ,\ (c,(d+1) \bmod s)]$

(iii) $[(c,d)\ ,\ (c,(d-1) \bmod s)]$

It is essentially an s-dimensional cube, in which every "vertex" is a cycle of size s. When s is a power of two, the CCC can simulate a (s+lg s)-dimensional cube, with each set of s vertices "sharing" s inter-dimensional edges.

As defined above, the CCC may effectively be regarded as undirected since every edge has a partner in the opposite direction. This is a quality which is shared with the cube, and it will be exploited here to shorten path lengths. In some versions of the CCC, the cycles are strictly directed, and the set of edges (iii), say, are omitted. These tend to be employed in algorithms with s stages, with data making a circular tour during the computation, each datum moving in step.

The graph can be generalised to have $h2^s$ vertices in cycles of size h embedded in an s-dimensional cube. When h=1, it is a cube and when s=0, it is a cycle. For simplicity, the only case considered here is that of h=s, although the actual simulation can handle the general case.

The routing algorithm employed is similar to that used for the cube, except that the cycles have to be taken into account. As usual, each packet is sent to a random node in the first phase and then sent to the correct node in the second phase. The route followed by a packet in each phase first takes it to its destination cycle and then takes it to the correct node in that cycle. During the first stage, all packets traverse cycles in the same direction and hence the order in which dimensions are traversed by a packet is fixed modulo s. Since the traversed dimensions are selected

randomly. It is not possible for optimisation of their ordering to yield much gain. In the second stage, when a packet must travel a random distance round a cycle, it is sent in the direction which provides a shorter path, thereby halving the expected and maximum distance.

Note that, later on, an algorithm for the cube which involves the dimensions being traversed in a defined order will be examined, and it will indicate whether this aspect of CCC routing introduces difficulties.

s	Size = $s2^s$	Mean	Variance	Maximum
2	8	3.63	0.27	4
3	24	6.11	0.22	7
4	64	9.60	0.52	12
5	160	12.44	0.60	15
6	384	16.08	0.80	20
7	896	19.20	0.82	23
8	2048	22.98	0.90	27

Table 3-16: Time for 1st phase on CCC

s	Size = $s2^s$	Mean	Variance	Maximum
2	8	3.98	0.54	7
3	24	7.49	0.63	10
4	64	11.46	0.93	15
5	160	14.68	0.90	19
6	384	18.56	1.10	24
7	896	21.83	1.12	26
8	2048	25.77	1.36	31

Table 3-17: Time for 2nd phase on CCC

s	Size = $s2^s$	Mean	Variance	Maximum
2	8	7.61	1.32	11
3	24	13.60	1.00	17
4	64	21.06	1.62	26
5	160	27.12	1.64	32
6	384	34.64	2.04	41
7	896	41.03	2.19	47
8	2048	48.75	2.27	55

Table 3-18: Total time for both phases on CCC

As one might expect, the CCC has the worst completion times seen so far. This is because the path length for each packet is relatively long, without even beginning to consider collision problems. It is composed of the number of inter-dimensional edges traversed, plus the number of cycle edges traversed in doing so, together with the length of the final journey round the cycle. In the worst case, it will be $s+s+\lceil s/2 \rceil = \lceil 5s/2 \rceil$. The expected value is given by :

Lemma (3.3): Expected path length for the CCC is

$$\frac{7s}{4} + \frac{1}{2^{s-1}} - 2 \quad \text{when s is even}$$

$$\frac{7s}{4} + \frac{1}{2^{s-1}} - \frac{1}{4s} - 2 \quad \text{when s is odd}$$

Proof: Expect packet to traverse $\frac{s}{2}$ inter-dimensional edges. The expected cycle distance travelled in doing so is equal to

$$\frac{s-1}{2^1} + \frac{s-2}{2^2} + \ldots + \frac{1}{2^{s-1}} + \frac{0}{2^s} + \frac{0}{2^s}$$

which sums to $s + \frac{1}{2^{s-1}} - 2$. Finally, the expected journey distance is :

For s even,

$$0.\frac{1}{s} + 1.\frac{2}{s} + \ldots + (\frac{s}{2}-1)\frac{2}{s} + \frac{s}{2}.\frac{1}{s} = \frac{s}{4}$$

For s odd,

$$0.\frac{1}{s} + 1.\frac{2}{s} + \ldots + \frac{s-1}{2}.\frac{2}{s} = \frac{s^2-1}{4s}$$

Looking at the results, it can be seen that completion time grows approximately linearly in s, with a greater multiplicative factor in the second phase. For the first phase, and indeed the overall run, the results support the earlier hypothesis that mean completion time grows approximately at double the expected path length. The second phase grows slightly faster but this can be explained by the initial bottlenecks created by the initial packet population queueing for the two available output edges. It has been seen that the two phases of the 2-shuffle, also with two output edges, do not differ by more than a small constant. However, for the CCC, there is

greater demand for cycle edges (around $\frac{3}{4}$ of arrivals will be queued for the forward edge during the first stage of routing) and so the network is more sensitive to long initial queues. This is the first time that a routing scheme has been used that does not have a uniform pattern of demands for out edges at each node.

The distribution of completion times is more dispersed than usual and, indeed, the variance is apparantly growing with s. This is not unexpected because of the unbalanced nature of the scheme, just noted, which is liable to make the outcomes of experiments less predictable. Occasional build-ups of packets at a node may be more noticeable when not smoothed by even distributions among queues. In this connection, it is worth noting that the second phase variance is about 50% more than that for the first.

Despite this, the maximum value for each phase is always within six of the mean and, indeed, more than 98% of the values observed are within three of the mean. The maximum total time is always within seven of the mean, with more than 98% of the values within four of the mean. Thus, there is still not a great deal of variability, especially when the increased means are considered.

s	Size = $s2^s$	Mean	Variance	Maximum
2	8	2.85	0.45	5
3	24	3.71	0.48	6
4	64	4.40	0.43	7
5	160	4.97	0.46	8
6	384	5.51	0.52	8
7	896	6.08	0.58	10
8	2048	6.60	0.49	9

Table 3-19: Max node population for 1st phase on CCC

In the first phase, the values observed for mean maximum node population have the same relation to graph size as those for the first phase of the cube. Here, as before, a maximum node population will be formed of a small number of early arrivals (due to variable path length) plus

s	Size = $s2^s$	Mean	Variance	Maximum
2	8	2.76	0.47	5
3	24	3.51	0.41	6
4	64	4.23	0.46	7
5	160	4.89	0.60	8
6	384	5.53	0.57	10
7	896	6.18	0.68	10
8	2048	6.83	0.64	11

Table 3-20: Max node population for 2nd phase on CCC

packets en route elsewhere. The interesting fact about the correspondence with the cube is that packets being routed still seem to be randomly distributed about the network. Considering a node in isolation, this suggests that arrivals and departures are balanced. In view of the asymmetric nature of arrivals and departures, already noted, the implication is that, in the vast majority of cases, queues are not forming and packets pass through a node with no delay.

The effect of asymmetric node behaviour is, however, illustrated by the second phase. The node populations do not match those for the cube, or indeed the d-shuffle, since they grow more rapidly with graph size from comparable initial values. This was to be expected if the justification of longer run time in the second phase was true. While it appears that the CCC algorithm lets packets flow fairly freely in the absence of external pressure, the initial perturbation is enough to interfere, albeit to a limited extent. The conclusion is that node behaviour is very finely tuned indeed.

While the above remarks are interesting from the point of view of studying routing algorithms, in practical terms the storage requirement at a node is not significantly more than that required on a cube. Since the first phases behave similarly, only the second phase is of concern. Here, the maximum value is at most five more than the mean, which in turn is less than one more than the cube mean. Since the variance appears stable for larger graph sizes, it reasonable to compare the CCC and cube means to estimate relative store requirements. Empirically, the results indicate that the CCC requires about 15% more than the cube.

s	Size = $s2^s$	Mean	Variance	Maximum
2	8	1.00	0.00	1
3	24	1.87	0.13	3
4	64	2.55	0.32	4
5	160	3.17	0.22	5
6	384	3.92	0.35	7
7	896	4.58	0.40	8
8	2048	5.35	0.49	9

Table 3-21: Max queue length for 1st phase on CCC

s	Size = $s2^s$	Mean	Variance	Maximum
2	8	1.84	0.31	4
3	24	2.55	0.41	5
4	64	3.54	0.47	7
5	160	4.27	0.55	8
6	384	4.93	0.55	9
7	896	5.68	0.63	10
8	2048	6.32	0.65	11

Table 3-22: Max queue length for 2nd phase on CCC

The queue size results are the worst seen so far and outpace even the 2-shuffle, which has only two queues per node. The difference between mean maximum node population and mean maximum queue length in the second phase is also the smallest seen yet. This adds further evidence for the hypothesis that, when bottlenecks do occur, they are due to packets being preferentially queued for the outgoing forward cycle edge. This biases the previously uniform distribution of packets to queues. The effect is accentuated at the start of the second phase by the (random) distribution of first phase arrivals amongst queues, which is sufficient to initiate bottlenecks. As the graph size increases, so does the worst case initial queue length and hence the second phase will worsen in relation to the first. One interesting special case is the eight node CCC graph. It is not hard to see that, for any permutation, a set of routes can be chosen such that there are no collisions. The difference between phases is thus due purely to the inpact of the initial loading of the second phase, since the first phase has no queueing delays.

It has been seen that the routing deficiencies in the CCC algorithm arise from two sources. Firstly, and by far the most important, is the relatively large expected path length. Secondly, there is the unbalanced routing of packets within nodes. The path length is primarily dictated by the diameter of the graph, and so cannot be substantially improved. Two possible modifications are (i) to optimise the round-cycle route while traversing dimensions, but this yields no improvement for worst case paths, and (ii) to miss out the random cycle trip at the end of the first phase, which should improve the first phase a little but may handicap the second phase for certain permutations. After a random dimension traversal, a packet is most likely to be in a cycle postion close to its starting position. Therefore, a permutation which sends packets to the "opposite side" of cycles will cause the second phase to have final in-cycle paths with expected length $\frac{s}{2}$ instead of $\frac{s}{4}$, which cancels out the first phase advantage.

The problem of unbalanced node behaviour can be handled by considering the CCC^+ graph, which is a variant of the CCC with unidirectional cycles, introduced by Upfal [51]. Its edges are :

For each $c = c_{s-1} \ldots c_0$ and d,

$\{[(c_{s-1} \ldots c_d \ldots c_0, d)\ ,\ (c_{s-1} \ldots \overline{c_d} \ldots c_0, (d+1) \bmod s)]\}$

and

$\{[(c_{s-1} \ldots c_d \ldots c_0, d)\ ,\ (c_{s-1} \ldots c_d \ldots c_0, (d+1) \bmod s)]\}$

The proposed routing algorithm first sends a packet on a round-cycle trip for s steps, randomly choosing whether to make an inter-dimensional move or not. Then the packet is sent to the cycle position corresponding to its target cycle position, again randomly traversing dimensions. Finally, by another round-cycle trip, the packet moves to its correct destination. Note that this algorithm differs from the normal one in having three phases.

It can be shown that node behaviour is now balanced (this time balanced in the strict sense of Upfal) as required, and this allows analytic results to be proved about the scheme. However, the expected path length is now

between 2s and 3s, depending on the permutation. Therefore, the algorithm will exhibit worse behaviour, based on the hypothesis regarding this measure dominating completion time.

This problem is inevitable in any graph where the shortest paths from a vertex to other vertices are not evenly distributed among out edges. Balance can only be achieved at the expense of path length. For the CCC, it has been seen that the imbalance does not have (but is close to having) the same impact on delays as path length does.

3.5 The shuffle-exchange graph

In the previous section, a graph related to the cube was studied. Here, a close relative of the 2-shuffle will be examined. The shuffle-exchange graph is historically important, being one of the first non-trivial architectures proposed for parallel computers. Stone [48] first demonstrated its natural suitability for problems such as the fast Fourier transform and since then, it has been studied by several researchers. It has also been proposed as a suitable basis for routing in a general purpose parallel computer [22, 44]. A shuffle-exchange graph with size 2^n is specified as follows :

Vertices,

$\{v \mid v \in \{0, 1\}^n\}$

Edges, for each $v = v_{n-1} \ldots v_0$,

(i) $(v_{n-1} \ldots v_0 \, , \, v_{n-1} \ldots \overline{v_0})$ [exchange edges]

(ii) $(v_{n-1}v_{n-2} \ldots v_0 \, , \, v_{n-2} \ldots v_0 v_{n-1})$ [shuffle edges]

It can be seen that this graph bears the same kind of relationship to the 2-shuffle as the CCC does to the CCC^+. With the 2-shuffle, exchanges and shuffles are effectively combined into one operation which always shuffles and optionally exchanges as well. It is not clear that any gains will accrue from considering the shuffle-exchange graph since it has the same degree and double the diameter of the 2-shuffle but, because of widespread

(possibly misplaced) interest in it, practical implementations of it are being considered [22] and so it is useful to know whether it performs well or badly when used for realising permutations.

The routing algorithm used is the obvious two-phase one, which is similar to that for the 2-shuffle, except that exchanges must be made explicit. When a packet with destination $\alpha_{n-1}\ldots\alpha_0$ arrives at a node $\beta_{n-1}\ldots\beta_0$, at stage i (i = n-1, n-2, ..., 1, 0) on its journey, the node executes the following algorithm :

if $\alpha_i \neq \beta_0$ **then begin**
  send to $\beta_{n-1}\ldots\bar{\beta}_0$, still at stage i
end else begin
  if i = 0 **then begin**
    packet has arrived at destination
  end else begin
    send to $\beta_{n-2}\ldots\beta_0\beta_{n-1}$, at stage i-1
  end
end

Initially, source nodes send packets along their outgoing shuffle edge, at stage n-1 of their journeys.

Note that no advantage is taken of paths which need length less than n. This will allow easy easy comparison with the d-shuffle results.

The completion times are comfortably the worst recorded so far. This is not surprising since the shuffle-exchange graph has degree two, non-optimal diameter, and the routing algorithm does not ensure balanced handling of packets within nodes. Looking at the mean completion times, it can be seen that they increase linearly with n. The constant of proportionality is slightly below three in both cases, with that for the first phase being smaller. The expected route length for a packet is $\frac{3n}{2}$, made up of n shuffle edges and $\frac{n}{2}$ exchange edges, totalling and so these results again fit the hypothesis about completion time for randomised routing schemes. Therefore, unless the graph had some magic property, nothing better would have been expected anyway.

n	Size = 2^n	Mean	Variance	Maximum
2	4	3.69	0.22	4
3	8	6.13	0.42	8
4	16	8.63	0.50	12
5	32	11.32	0.74	15
6	64	14.05	0.93	18
7	128	16.92	1.56	31
8	256	19.73	1.18	25
9	512	22.56	1.39	30
10	1024	25.58	1.56	33
11	2048	28.43	1.46	35
12	4096	31.26	1.32	37

Table 3-23: Time for 1st phase on shuffle-exchange

n	Size = 2^n	Mean	Variance	Maximum
2	4	4.50	0.63	7
3	8	7.06	0.89	10
4	16	9.79	1.03	14
5	32	12.61	1.46	20
6	64	15.48	1.78	22
7	128	18.37	1.64	25
8	256	21.37	1.59	29
9	512	24.22	1.72	32
10	1024	27.18	2.02	37
11	2048	30.18	2.15	38
12	4096	33.09	1.93	40

Table 3-24: Time for 2nd phase on shuffle-exchange

n	Size = 2^n	Mean	Variance	Maximum
2	4	8.19	1.20	11
3	8	13.19	1.65	17
4	16	18.41	1.71	24
5	32	23.93	2.43	31
6	64	29.53	2.55	37
7	128	35.30	3.33	50
8	256	41.10	3.00	51
9	512	46.77	3.03	56
10	1024	52.76	3.75	63
11	2048	58.61	3.55	66
12	4096	64.36	3.38	73

Table 3-25: Total time for both phases on shuffle-exchange

As well as the large mean values, the distribution of completion times is rather more dispersed than usual. This is a consequence of the unbalanced nature of the scheme causing bottlenecks in the network. The most striking example occurred during experiments on the 2^7 node graph. There, one experiment had a completion time for the first phase of 31 time units; the next highest was 23, and the others were under 21. This was due to a queue of size eleven (also an isolated maximum) building up. Clearly, since the completion time normally falls within the same sort of bound as that predicted from other graphs, the load is fairly evenly distributed about the network. However, whereas balanced schemes do not show much variability, one has to be very cautious when making statements about worst case behaviour of unbalanced schemes, such as this one.

Like the CCC routing algorithm, the second phase is made slower than the first by the fact that it starts with non-trivial queues. These can only serve to increase the likelihood of early bottlenecks. The variance also noticeably increases, as an indication of greater run time instability. One consolation for the fact that unpleasant worst cases do occur is that they appear to happen independently in the different phases. The difference between the worst and mean total run time is about the same as the difference for the separate phases.

n	Size = 2^n	Mean	Variance	Maximum
2	4	1.44	0.25	2
3	8	2.17	0.15	3
4	16	2.67	0.39	6
5	32	3.24	0.28	6
6	64	3.73	0.46	6
7	128	4.28	0.50	11
8	256	4.83	0.56	7
9	512	5.37	0.55	10
10	1024	5.87	0.54	9
11	2048	6.41	0.48	9
12	4096	6.96	0.61	10

Table 3-26: Max node population for 1st phase on shuffle-exchange

For the first time, the node population results vary from the previously

n	Size = 2^n	Mean	Variance	Maximum
2	4	2.20	0.33	4
3	8	2.82	0.54	5
4	16	3.39	0.46	6
5	32	3.89	0.65	8
6	64	4.48	0.63	8
7	128	4.98	0.73	8
8	256	5.53	0.67	10
9	512	5.98	0.69	9
10	1024	6.52	0.75	11
11	2048	7.04	0.71	11
12	4096	7.59	0.72	11

Table 3-27: Max node population for 2nd phase on shuffle-exchange

observed pattern. However, since the hypothesis was that balanced schemes effectively distributed packets randomly among nodes, the variation is not surprising. Given any imbalance, greater worst case populations will be expected. In both phases, the mean values grow linearly with n at a faster rate than for other schemes but, surprisingly, both phases grow at the same rate, with the second being about 0.6 more than the first, and the first phase mean is actually smaller than the normal (balanced) mean for all n up to ten. Given that a certain node population results from routing packets, it has been noted previously that early arrivals in the first phase, and initial populations in the second phase, can both cause increases logarithmic in the graph size. In the case of the cube, this meant that the first phase mean grew faster; in the case of the CCC, the second phase mean grew faster. Here, it seems that both phase-dependent features have equivalent effects, to within a constant difference, and cancel each other out. Note that the effects are not entirely independent - the arrivals at a node in the first phase form the initial population in the second phase.

A reason for the fact that the first phase mean maximum node population is smaller than that observed for comparable graphs is not obvious at first sight. Further discussion of this point will be deferred until queue sizes are considered.

For the sizes of graph involved in the experiments, the worst node populations occurring are not significantly more than those for the cube, for example. While the variance of maximum population is larger, the extreme values differ by at most one usually. However, the rogue experiment when n equalled seven indicates that, although the shuffle-exchange does not appear to make much extra demand for storage at nodes, it is prone to occasional excesses. To circumvent this, it may be sensible for a node to refuse new packets while its population is too large, hopefully without much effect on network performance.

n	Size = 2^n	Mean	Variance	Maximum
2	4	1.00	0.00	1
3	8	1.91	0.21	3
4	16	2.35	0.27	5
5	32	2.95	0.36	6
6	64	3.42	0.37	6
7	128	3.95	0.59	11
8	256	4.49	0.52	7
9	512	5.05	0.59	10
10	1024	5.53	0.52	9
11	2048	6.05	0.49	9
12	4096	6.66	0.61	10

Table 3-28: Max queue length for 1st phase on shuffle-exchange

n	Size = 2^n	Mean	Variance	Maximum
2	4	2.10	0.33	4
3	8	2.68	0.53	5
4	16	3.22	0.53	6
5	32	3.74	0.60	7
6	64	4.26	0.62	8
7	128	4.78	0.70	8
8	256	5.28	0.63	9
9	512	5.71	0.63	9
10	1024	6.28	0.80	11
11	2048	6.76	0.68	10
12	4096	7.27	0.70	11

Table 3-29: Max queue length for 2nd phase on shuffle-exchange

Relatively large queues, presumably for outgoing shuffle edges, occur. As well as being the largest queue lengths seen compared with other graphs, the mean maximum queue length is very close to the mean maximum node population. The difference is similar for both phases, which is further evidence that the "special features" of each phase, which affect node population, are balanced out. The queue lengths for the second phase are, as expected, larger than those for the first.

Having noted that the size of one queue is effectively responsible for a large node population, an explanation of why first phase maximum node populations are smaller than those on other graphs is possible. In a scheme where packets are randomly distributed among output queues at a node, a large population is formed primarily by a number of maximum length queues. Assuming that queue lengths are a property of the graph combined with the routing algorithm, if there is only one queue at a node which is likely to grow large, rather than several, the resulting node population would be expected to be smaller. Eventually, as can be seen here, the mean maximum queue length outpaces the mean maximum node population for other graphs, and this effect is not significant any longer.

The occasional queueing bottlenecks, which cause the behaviour of the shuffle-exchange graph to be more erratic than others, are perhaps best summarised by the worst queue sizes occurring over a series of experiments. These fluctuate a great deal, whereas the mean increases linearly with n. In more than 95% of the experiments, the observed maximum queue length is not more than two away from the mean. Clearly, a much larger number of experiments with this scheme would be necessary in order to get a true picture of the relevant distributions. Events such as the eleven packet queue in a 128 node graph will not occur frequently.

The two major drawbacks of the shuffle-exchange graph are, therefore, its larger diameter, which leads to longer completion times, and the unbalanced use of edges, which leads to bottlenecks. Although minor optimisations can be made to routes, the first problem cannot be handled effectively. One solution to the second problem might be to consider the

graph as being undirected with bidirectional shuffle edges. If packets are then routed using a random shifting direction, the expected shuffle edge queue lengths should be decreased. However, route lengths would not decrease. A more obvious solution is to merge the shuffle and exchange edges in such a way that imbalance is removed, and route lengths decrease. If this is done in a simple-minded manner, the resulting graph is the 2-shuffle, which has already been considered.

3.6 The two-dimensional square grid graph

The final graph to be considered differs markedly from all the others examined previously. It consists of an array of processors connected in a two-dimensional square grid. In general, it can take the form of an array of processors connected in a k-dimensional grid, for some integer $k > 0$, with each dimension of arbitrary size, but only the special case will be considered in the context of routing. The definition of the graph, with n^2 vertices, is as follows :

Vertices,

$\{(i,j) \mid 0 \leq i,j \leq n-1\}$

Edges,

(i) $\{[(i,j),(i,j-1)] \mid 0 \leq i \leq n-1,\ 0 < j \leq n-1\}$

(ii) $\{[(i,j),(i,j+1)] \mid 0 \leq i \leq n-1,\ 0 \leq j < n-1\}$

(iii) $\{[(i,j),(i-1,j)] \mid 0 < i \leq n-1,\ 0 \leq j \leq n-1\}$

(iv) $\{[(i,j),(i+1,j)] \mid 0 \leq i < n-1,\ 0 \leq j \leq n-1\}$

This organisation has been the most popular for early parallel computers consisting of distinct processing units, for example ILIAC IV [4]. It is also ideal for implementation in current two-dimensional integrated circuit technology since, as will be seen in Chapter Five, it can be laid out so that all edges have a uniform short constant length. Because of this great practical importance, it will be examined as a medium for implementing randomised routing algorithms. Some analytic results for the grid can be found in [53].

The major difference from the other graphs is that the diameter is 2(n-1), which is proportional to the square root of the graph size, compared with the normal logarithmic (and optimal for constant degree graphs) diameter. Therefore, it is clearly going to perform in a far inferior manner. Indeed, it is probably unsatisfactory as a basis for a general-purpose parallel computer and best restricted to problems of a systolic nature [30]. Nevertheless, it is interesting to see whether the empirical hypotheses already formulated still hold in such a different context.

The routing algorithm, as usual, consists of sending each packet to a random node in the first phase and then to the destination node in the second phase. Two different ways of selecting paths will be considered. In the first, a packet moves to the correct row, and then to the correct column. In the second, a packet moves between rows and columns in a random sequence.

One interesting feature of this scheme is that, whereas previously the expected path length was the same for all packets, the length for each packet differs, depending upon the starting node. Since this measure appears to determine completion time, it has to be studied. The worst case expected path length is n-1 (for a packet starting at one of the "corners"). The expected expected path length over all packets is given by:

Lemma (3.4): The expected expected path length is

$$\frac{2}{3}(n - \frac{1}{n})$$

Proof: The expected distance between a fixed node and a randomly selected one is equal to the sum of the expected row distance and the expected column distance. Since the grid is square, all nodes in row i, for each i, have the same expected row distance and it is the same as the expected column distance shared by all nodes in column i. Call this expected distance e_i. Then the expected expected path length is equal to

$$\sum_{i=0}^{n-1} \sum_{j=1}^{n-1} (e_i+e_j) \frac{1}{n^2}$$

$$= \frac{2}{n} \sum_{i=0}^{n-1} e_i$$

Now it can be seen that

$$e_i = \sum_{j=0}^{n-1} \frac{|i-j|}{n} = \frac{1}{n} \left[\sum_{j=0}^{i-1} (i-j) + \sum_{j=i}^{n-1} (j-i) \right]$$

$$= \frac{1}{n} \left[|i-(n-i)|i - \frac{i(i-1)}{2} + \left(\frac{n(n-1)}{2} - \frac{i(i-1)}{2} \right) \right]$$

$$= \frac{1}{2n} \left[2i^2 - 2(n-1)i + n(n-1) \right]$$

Hence, $\frac{2}{n} \sum_{i=0}^{n-1} e_i$ can be expressed as

$$\frac{1}{n^2} \left[\frac{n(n-1)(2n-1)}{3} - n(n-1)^2 + n^2(n-1) \right]$$

$$= \frac{1}{3n} \left[2n^2 - 3n + 1 - 3n^2 + 6n - 3 + 3n^2 - 3n \right]$$

$$= \frac{1}{3n} \left[2n^2 - 2 \right]$$

$$= \frac{2}{3} \left[n - \frac{1}{n} \right]$$

Both the worst case and the expected value will be borne in mind when considering how completion time grows, to see whether either is the dominating influence.

The range of graph sizes considered for the grid is smaller than that for other graphs because of the longer run time and hence longer simulation time. The side length n is increased from five to thirty in steps of five. Results are tabulated first for the "rows, then columns" routing, then for the "rows and columns mixed" routing. For the sake of brevity,. the two strategies will be referred to as the "strict" and the "random" strategies respectively for the rest of this section.

The mean completion times, while large, are always less than the diameter of the graph, and even the worst case completion times are not much relatively more than the diameter. Thus, it appears that no significant congestion is occurring, assuming that near-maximally sized

n	Size = n^2	Mean	Variance	Maximum
5	25	6.83	0.60	9
10	100	15.55	1.34	18
15	225	24.41	2.21	28
20	400	33.62	2.93	38
25	625	43.00	4.05	48
30	900	52.36	4.42	58

Table 3-30: Time for 1st phase on grid with strict strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	7.44	1.10	12
10	100	16.57	1.86	21
15	225	25.70	3.06	33
20	400	35.05	4.03	41
25	625	44.61	5.12	52
30	900	54.03	5.40	61

Table 3-31: Time for 2nd phase on grid with strict strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	14.26	2.72	20
10	100	32.12	5.55	39
15	225	50.11	8.97	59
20	400	68.67	12.28	78
25	625	87.61	16.24	98
30	900	106.39	17.78	118

Table 3-32: Total time for both phases on grid with strict strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	7.19	0.86	10
10	100	16.69	1.71	21
15	225	26.49	2.29	31
20	400	36.51	3.32	41
25	625	46.53	3.66	52
30	900	56.34	4.69	63

Table 3-33: Time for 1st phase on grid with random strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	7.56	1.15	12
10	100	17.17	1.62	21
15	225	27.01	2.76	35
20	400	36.94	3.93	44
25	625	46.96	3.71	53
30	900	56.76	4.24	62

Table 3-34: Time for 2nd phase on grid with random strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	14.74	3.13	21
10	100	33.87	4.94	41
15	225	53.50	7.20	61
20	400	73.45	10.80	84
25	625	93.49	9.91	101
30	900	113.10	12.33	123

Table 3-35: Total time for both phases on grid with random strategy

routes occur in each experiment. It can be seen that, for both strategies, the mean completion time grows at about twice the rate of increase in n, the rate being slightly less for the strict strategy and for the first phase of the random strategy. This is consistent with the completion time hypothesis if expected path length is taken to mean the maximum expected path length, taken over all packets, which is encouraging.

The strict strategy produces an improvement over the random one, the improvement being better for the first phase. This is because it is advantageous to have all packets moving in the same direction, that is horizontally or vertically. Then, each row (or column) acts as a pipeline with packets being "pumped through" without suffering delays. At the beginning of the first phase, this pipeline effect is achieved totally since all packets move along columns with no initial queues. As packets begin to switch to move along rows, queues can start to form and so packets are slowed down. Throughout the second phase, a similar state of affairs applies and so a lesser improvement in completion time results.

At first sight, the improvement may seem to conflict with previous remarks that routing strategies which direct a packet to a random output queue at a node are preferable. This would tend to favour the random strategy. However, there are two points to note about the strict strategy. Firstly, taken over all in edges at a node, each out edge is equally favoured, and so the output queues should be balanced. Secondly, with the vast majority of arrivals on a particular in edge being directed to a particular out edge, the lack of randomness favours free pipelining.

There is only a very small difference between the completion time for the two phases with the random strategy. This is not unexpected since routes are so long that any initial effects in the second phase are likely to be ironed out as packets progress.

The entire distributions of completion time are markedly more dispersed than for any of the other graphs. Further, they are fairly symmetric about the mean whereas, although not mentioned earlier, the others are very much skewed towards the maximum value. The variance increases with n and is larger for the strict strategy, more notably so for the second phase. Considering the variance of the completion time for both phases, it can be seen that the covariance of the phase completion times is increasing with n, and is considerably larger for the strict strategy.

Given that completion time is determined by the packet which has the worst journey time, it follows that the distributions of completion time will be different for the grid because the distribution of worst path length is far less sharp than that for logarithmic diameter graphs. As n increases, so does the range of possible worst path lengths but, whereas for other graphs the maximum of the range may occur for any packet, this is not the case for the grid, where only the four packets starting at the corners can achieve a maximum path length. The strict strategy yields a fairly uniform improvement for the first phase, and so the distributions are similar for both strategies. However, for the second phase, while the imposition of strictness improves the mean completion time, there are obviously worst cases in which initial node populations hinder queueless pipelining and so

maximum values are similar for both strategies. Hence the strict strategy has a broader second phase distribution.

The high correlation between the two phases is a feature of the identity permutation being used in experiments: packets follow the first phase path in reverse in the second phase under the strict strategy and so both completion times are essentially dependent on the choices of the intermediate random nodes for each packet. The increasing correlation with n for both strategies is again an indication of the two completion times both being determined by one packet with a long route, rather than two independent packets.

n	Size = n^2	Mean	Variance	Maximum
5	25	4.07	0.53	7
10	100	5.12	0.51	8
15	225	5.71	0.56	10
20	400	6.13	0.47	10
25	625	6.36	0.44	8
30	900	6.58	0.46	10

Table 3-36: Max node popn for 1st phase on grid with strict strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	3.76	0.48	7
10	100	4.70	0.50	8
15	225	5.19	0.42	9
20	400	5.49	0.44	10
25	625	5.68	0.46	8
30	900	5.82	0.50	8

Table 3-37: Max node popn for 2nd phase on grid with strict strategy

For both strategies, the first phase node populations are greater than the second. Since there is a large variance of path length, early arrivals will have a significant impact on the grid routing scheme, so this observation is to be expected. The difference is slightly larger for the strict strategy since the node population due to routing is smaller with respect to n, whereas the contribution from early arrivals is the same.

n	Size = n^2	Mean	Variance	Maximum
5	25	4.26	0.54	7
10	100	5.57	0.57	9
15	225	6.46	0.61	9
20	400	6.98	0.68	10
25	625	7.44	0.67	10
30	900	7.77	0.62	11

Table 3-38: Max node popn for 1st phase on grid with random strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	3.96	0.49	7
10	100	5.23	0.41	7
15	225	6.03	0.63	9
20	400	6.48	0.53	11
25	625	6.88	0.59	11
30	900	7.21	0.51	11

Table 3-39: Max node popn for 2nd phase on grid with random strategy

It is immediately obvious from the results that the strict strategy produces smaller maximum node populations on average, as a result of its freer flow. However, a more interesting distinction can be seen by considering the increase in expected worst population with graph size. For the random strategy, the increase is logarithmic in graph size, as has been observed for other graphs, whereas the increase is sublogarithmic for the strict strategy.

The result for the random strategy is encouraging in view of the hypothesis that, given random routing, the distribution of packets over the network does not cause isolated bottlenecks to occur at nodes, resulting in the logarithmic worst case node population. Here, as with the shuffle-exchange graph the constant factor multiplying the logarithmic factor is larger than the "standard" one applying to all balanced schemes. Clearly, packets are more likely to be in the central part of the grid rather than near to the edges, so the distribution is not uniform, leading to imbalance in a controlled manner.

As with completion time, the strict strategy exhibits better asymptotic node population behaviour, despite its lack of randomisation. Pipelining is operating sufficiently well that packets are more equitably distributed than is possible by random distribution, and so the worst case population is smaller.

As well as a reduced mean maximum node population, the strict strategy also produces a slightly sharper distribution. However, it can be seen, by examining the maximum values occurring, that the worst cases for the two strategies cannot be effectively differentiated, and so the peak storage requirement at nodes is similar. However, since the variance does not appear to increase with n, it is reasonable to suppose that, for larger values of n than those considered here, the two distributions will have differing extremes as the mean values diverge.

n	Size = n^2	Mean	Variance	Maximum
5	25	1.87	0.21	4
10	100	2.25	0.21	4
15	225	2.56	0.32	5
20	400	2.71	0.26	4
25	625	2.84	0.22	4
30	900	2.94	0.17	4

Table 3-40: Max queue length for 1st phase on grid with strict strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	2.72	0.47	6
10	100	3.58	0.52	6
15	225	4.09	0.53	7
20	400	4.39	0.47	7
25	625	4.63	0.51	7
30	900	4.83	0.50	7

Table 3-41: Max queue length for 2nd phase on grid with strict strategy

The most striking feature of the queue length results is the first phase of routing with the strict strategy. This highlights the gain from strictness, namely the avoidance of queues in the first phase. It can be seen that,

n	Size = n^2	Mean	Variance	Maximum
5	25	2.11	0.18	4
10	100	3.02	0.24	5
15	225	3.55	0.38	6
20	400	4.02	0.38	6
25	625	4.44	0.40	7
30	900	4.76	0.45	7

Table 3-42: Max queue length for 1st phase on grid with random strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	2.48	0.37	6
10	100	3.39	0.38	6
15	225	3.96	0.44	7
20	400	4.32	0.50	8
25	625	4.67	0.42	7
30	900	4.96	0.50	8

Table 3-43: Max queue length for 2nd phase on grid with random strategy

within the limits of experimental error, maximum queue length is not improved in the second phase. It is interesting to note that, for the strict strategy, maximum queue length grows with n in the first phase at half the rate of increase in the second phase. Intuitively, this is consistent with the fact that half of the packet movements in the first phase are expected to be queue-free. This view is rather naive, of course, because rather than obtaining a reduction to half-size of all queues at a node, one direction is reduced to zero and the other stays the same. Indeed, further experiments (not included here) indicate that, when the first phase is divided into two distinct horizontal and vertical stages, queue lengths in the second stage are not halved, but remain the same. However, it is clear that, in some more complex way, the merging of the two stages does lead to the desired improvement.

The unbalanced behaviour of nodes in the strict scheme is highlighted by the close correlation of maximum queue length and maximum node populations in the second phase. This is less noticeable when the random scheme is used. However, as already noted, the strict scheme is

deliberately unbalanced and so this observation merely confirms that it is working as expected. The distribution of maximum queue length in the second phase is similar for both strategies. Therefore, the worst bottlenecks are no different. However, the improved completion times for the strict strategy indicate that the frequency of bottlenecks is reduced.

The experimental results have pinpointed one deficiency of the grid which impacts on several performance issues. This is the lack of symmetry, in the sense that the nodes around the "sides" of the grid have smaller degree than internal nodes. The effect is to increase the diameter and expected path length, and hence completion time. Further, packets can no longer be considered as being randomly distributed about the network, since they are more likely to be at the more internal nodes.

The problem can be overcome by augmenting the graph in such a way that it becomes symmetric, while retaining the desirable property of having a planar embedding with short edge lengths (described in Chapter Five). The revised set of edges is :

(i) $\{[(i,j), (i,(j-1) \bmod n)] \mid 0 \leq i,j \leq n-1\}$
(ii) $\{[(i,j), (i,(j+1) \bmod n)] \mid 0 \leq i,j \leq n-1\}$
(iii) $\{[(i,j), ((i-1) \bmod n,j)] \mid 0 \leq i,j \leq n-1\}$
(iv) $\{[(i,j), ((i+1) \bmod n,j)] \mid 0 \leq i,j \leq n-1\}$

Clearly, the graph has been "wrapped round" in both vertical and horizontal directions, so that opposite sides are joined together. The diameter of the graph is reduced from $2(n-1)$ to $2\lceil \frac{n}{2} \rceil$, i.e. it is almost halved. For a randomised routing scheme, the expected path length is now $\lceil \frac{n}{2} \rceil$ uniformly for all packets.

Further experiments have been conducted on the revised graph and the results are now tabulated together, along with a brief discussion and comparison with those obtained for the original graph.

As predicted, these are superior in all respects to those for the unwrapped grid. For both strategies, the mean completion time is reduced

n	Size = n^2	Mean	Variance	Maximum
5	25	5.16	0.38	7
10	100	9.96	0.31	12
15	225	15.29	0.47	17
20	400	19.96	0.36	22
25	625	25.33	0.52	27
30	900	29.95	0.34	31

Table 3–44: Time for 1st phase on wrapped grid, strict strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	5.51	0.58	8
10	100	10.66	0.69	13
15	225	16.01	0.73	20
20	400	20.92	0.77	24
25	625	26.25	0.94	30
30	900	31.02	0.79	34

Table 3–45: Time for 2nd phase on wrapped grid, strict strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	10.68	1.38	14
10	100	20.63	1.17	24
15	225	31.30	1.63	36
20	400	40.88	1.39	45
25	625	51.58	1.93	57
30	900	60.97	1.37	64

Table 3–46: Total time for both phases on wrapped grid, strict strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	5.19	0.52	8
10	100	10.45	0.57	13
15	225	16.13	0.80	19
20	400	21.28	0.77	24
25	625	26.95	0.97	31
30	900	32.09	0.99	35

Table 3–47: Time for 1st phase on wrap grid, random strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	5.49	0.69	8
10	100	10.84	0.80	13
15	225	16.50	0.84	20
20	400	21.63	0.89	24
25	625	27.30	1.25	31
30	900	32.39	1.15	37

Table 3-48: Time for 2nd phase on wrap grid, random strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	10.68	1.84	15
10	100	21.29	1.73	26
15	225	32.63	2.07	37
20	400	42.91	1.84	47
25	625	54.26	2.52	59
30	900	64.48	2.40	70

Table 3-49: Total time for both phases on wrap grid, random strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	3.77	0.48	6
10	100	4.70	0.47	7
15	225	5.33	0.45	8
20	400	5.73	0.54	9
25	625	5.99	0.48	8
30	900	6.22	0.44	9

Table 3-50: Max node popn for 1st phase on wrap grid, strict strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	3.58	0.42	6
10	100	4.41	0.38	7
15	225	4.92	0.46	8
20	400	5.21	0.45	9
25	625	5.42	0.39	8
30	900	5.59	0.46	9

Table 3-51: Max node popn for 2nd phase on wrap grid, strict strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	3.81	0.47	6
10	100	4.85	0.50	8
15	225	5.49	0.47	8
20	400	5.87	0.49	9
25	625	6.16	0.41	8
30	900	6.38	0.44	9

Table 3-52: Max node popn for 1st phase on wrap grid, random strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	3.62	0.42	6
10	100	4.49	0.37	8
15	225	5.11	0.44	8
20	400	5.36	0.41	8
25	625	5.57	0.40	8
30	900	5.79	0.45	8

Table 3-53: Max node popn for 2nd phase on wrap grid, random strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	1.83	0.23	3
10	100	2.28	0.22	4
15	225	2.55	0.29	5
20	400	2.68	0.24	4
25	625	2.82	0.20	4
30	900	2.89	0.15	5

Table 3-54: Max queue len for 1st phase on wrap grid, strict strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	2.34	0.32	5
10	100	3.12	0.28	5
15	225	3.54	0.37	6
20	400	3.80	0.38	7
25	625	4.02	0.34	6
30	900	4.16	0.31	6

Table 3-55: Max queue len for 2nd phase on wrap grid, strict strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	1.91	0.20	3
10	100	2.41	0.25	4
15	225	2.94	0.16	4
20	400	3.11	0.11	4
25	625	3.27	0.22	5
30	900	3.44	0.30	5

Table 3-56: Max queue len for 1st phase on wrap grid, random strategy

n	Size = n^2	Mean	Variance	Maximum
5	25	2.20	0.18	3
10	100	2.84	0.26	5
15	225	3.23	0.21	6
20	400	3.39	0.31	6
25	625	3.62	0.33	6
30	900	3.86	0.30	5

Table 3-57: Max queue len for 2nd phase on wrap grid, random strategy

by around 45%. Although the worst expected path length has been reduced by 50%, a corresponding reduction in completion times is not possible because, in the wrapped case, one packet is expected to travel the worst case distance during every experiment, whereas this event occurred with low probability in the unwrapped case. In fact, the wrapped grid shares a property with the cube, namely that the worst case distance for any packet is always double the expected distance, and so the completion time must be slightly larger than double the expected path length.

The benficial effect of the strict strategy is preserved on the modified network, with the improvement, relative to completion time, being the same as before. For both strategies, the variance is substantially reduced, as desired. However, the random strategy now has the larger variance. Intuitively, one would expect that the random strategy would lead to more variability than the strict one. In the unwrapped case, most of the variability was due to varying worst path lengths and so the variance of the strict strategy, which has completion time more sensitive to path length, was greater.

All of the node population distributions are moved downwards. The reduction for the strict strategy is very small but, for the random strategy, it is more significant. This is a result of making the grid symmetric. While the strict strategy relies on pipelining to avoid large node populations, the random one relies on random distribution of packets throughout the network. Now that a packet is equally likely to be at any node, the mean maximum node population is reduced. In fact, the random strategy on a wrapped grid is a completely balanced random routing scheme. It is extremely interesting to consider the following table, which compares the distributions of maximum node population for the second phase over 500 experiments on a ten dimensional cube, a diameter ten 2-shuffle, and a 32x32 wrapped grid, i.e. three graphs all with size 1024. The distribution of the maximum number of successes observed from 1024 trials with 1024 equiprobable outcomes is also given.

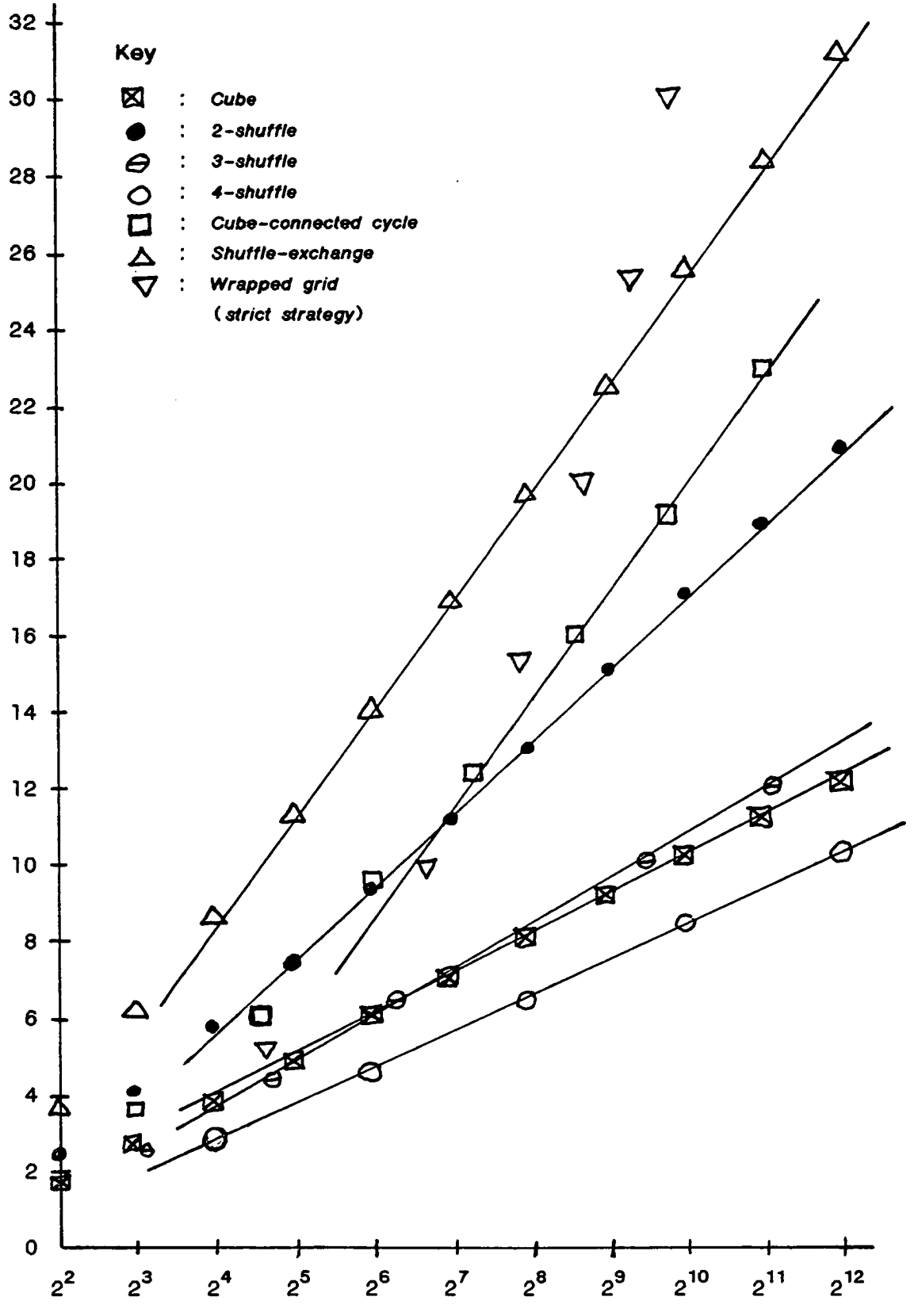
Population	4	5	6	7	8	9
Cube	-	133	299	63	5	-
2-shuffle	-	181	258	54	7	-
Grid	-	145	296	50	8	1
Multinomial	6	259	188	42	4	1

The distributions for the cube and grid are remarkably similar. The 2-shuffle one is more skewed to the left since the constant path length lessens the effect of the second phase arrival packets contributing to a maximum node population size. Given the disparities of degree, diameter, and expected path length, these results are a striking demonstration of the near independence of packet distribution and graph structure. Since all of the distributions are similarly displaced from the distribution solely due to random addressing, this is a comment about queueing behaviour.

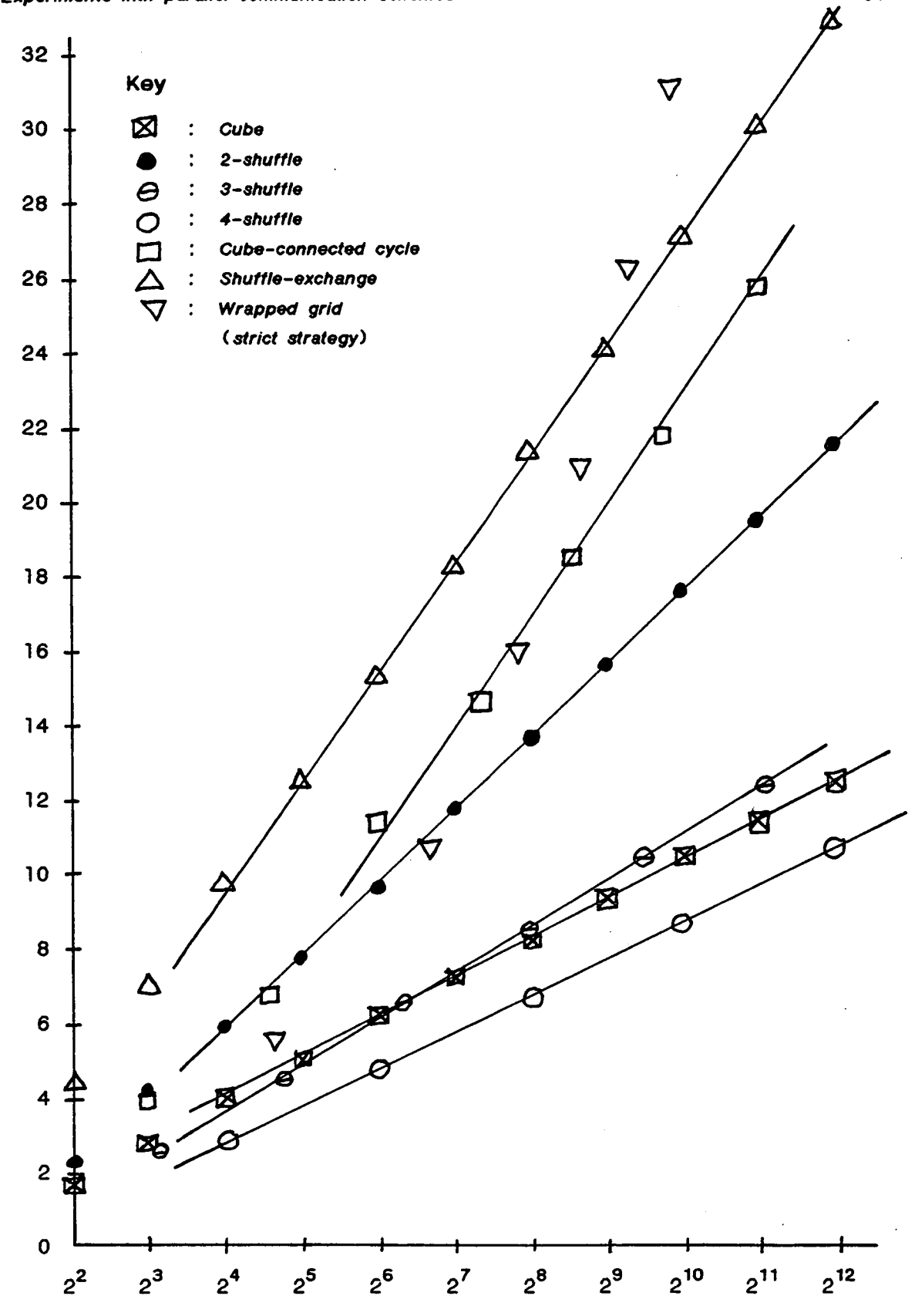
Returning to the comparison of unwrapped and wrapped grids, it can be seen that mean maximum queue sizes are reduced, with the more substantial gain being for the random strategy. Indeed, the change in first phase means for the strict strategy is imperceptible. Given that the strict strategy does not appear to benefit much from more uniform packet distribution during routing, it is reasonable to deduce that the gain in the

second phase is due to the shortening of the initial queues due to all nodes randomly distributing the first phase arrivals between two queues. The random strategy, which is better suited to a symmetric system, shows significant improvements in both phases. In fact, for the second phase, the random strategy has smaller maximum queue lengths than the strict strategy.

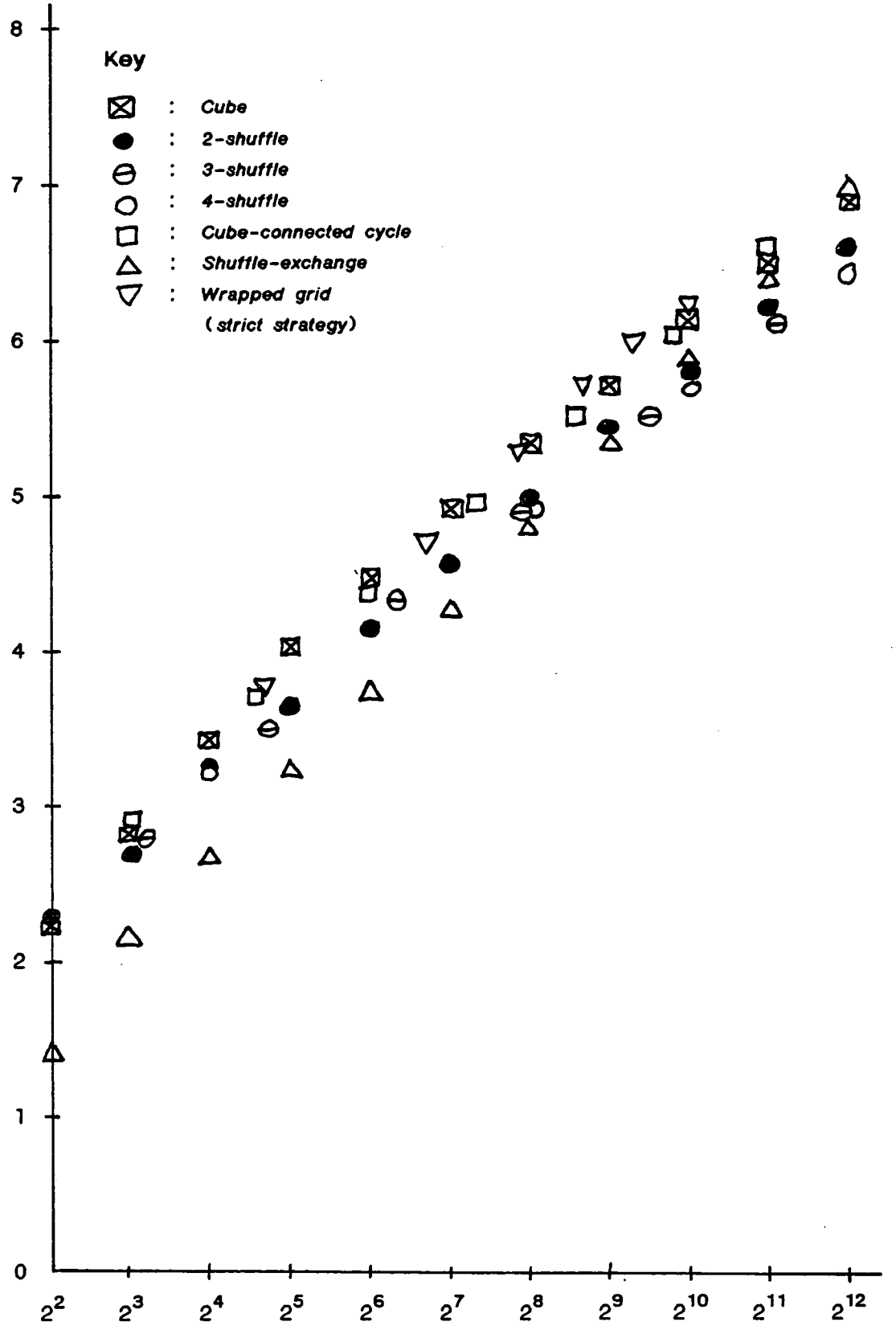
This concludes the consideration of grid graphs as random routing media, and it is the last graph to be looked at here. The next four pages contain plots which allow a pictorial comparison of the completion times and maximum node populations to be made between all of the graphs studied.



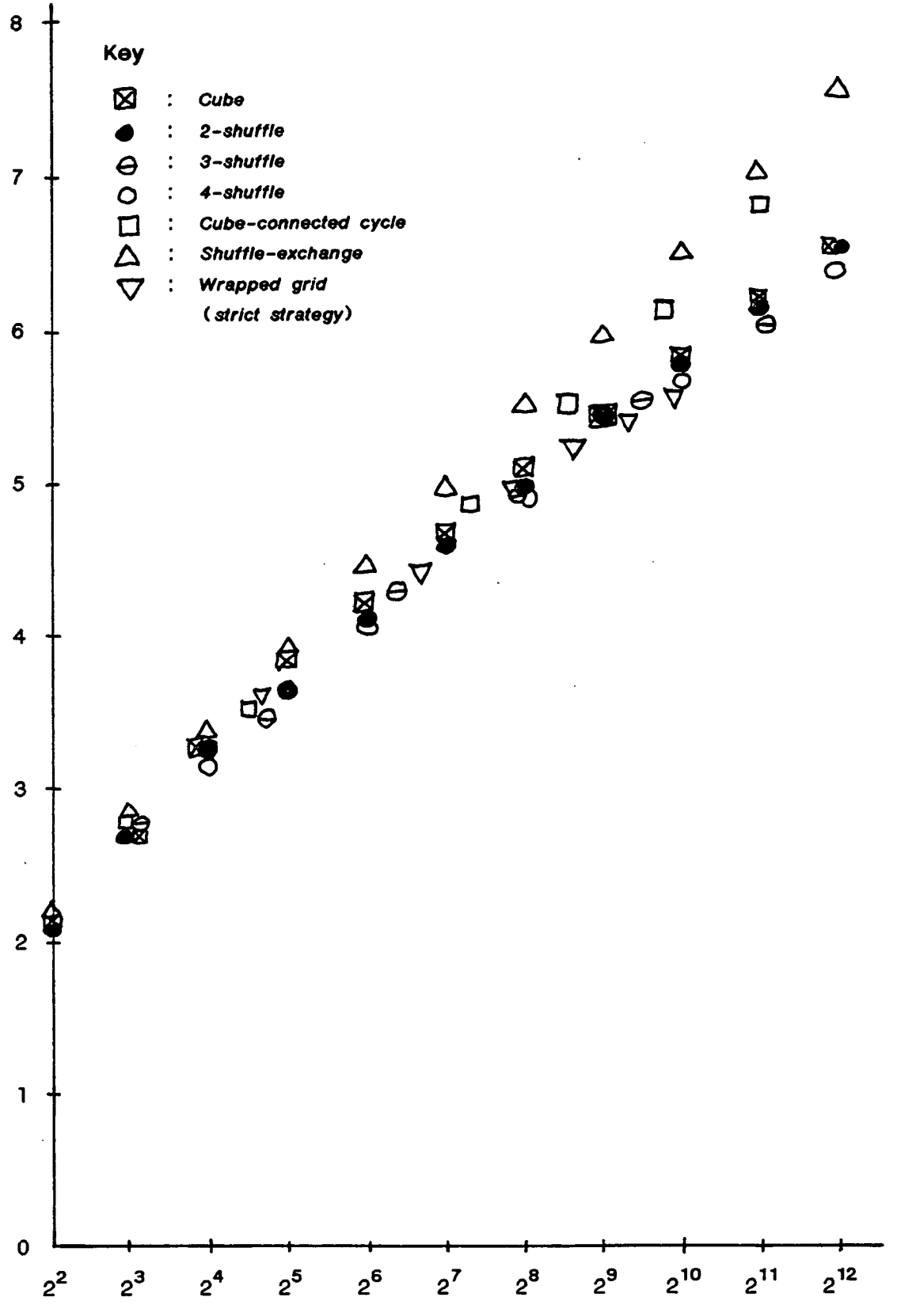
Mean completion time for first phase v. Graph size



Mean completion time for second phase v. Graph size



Mean maximum node population for first phase v. Graph size



Mean maximum node population for second phase v. Graph size

3.7 Variations of the standard routing algorithm

The routing algorithms employed in the previous experiments have, in general, been the most simple-minded possible. This has a practical motivation, in that computation within routing nodes is kept efficient. Thus, packets follow "obvious" routes; queues are first-come first-served, one per edge; and the two phases are regarded as separate entities. There are two reasons for varying this approach. Firstly, gains in performance may result. Secondly, analytic guarantees of performance sometimes rely on the imposition of special routing algorithm features. This section examines some modifications, for the cube and the d-shuffle graphs, to assess their effects experimentally. The presentation of results is far briefer than before, and typically consists of comparative distributions between standard and modified algorithms for two instances of a particular graph.

3.7.1 Variation of cube routing

In the routing algorithm for the cube, the randomly chosen dimensions to be altered are traversed in a random order. Some analytic results proved for cubes depend upon the dimensions being traversed in increasing order. This allows guarantees to be made about equal expected utilisation of all edges (the symmetry property). This "strict" variant is compared here with the original "random" variant and with a "semi-strict" scheme, in which dimensions are traversed in increasing order modulo the number of dimensions, with a random starting dimension. The cube representatives chosen, as throughout this section, are the 256 and 512 node networks.

	7	8	9	10	11	12	Mean	Variance
Random 1st phase	39	355	98	8	-	-	8.15	0.32
Semi-strict 1st phase	65	373	61	1	-	-	8.00	0.26
Strict 1st phase	72	389	38	1	-	-	7.94	0.22
Random 2nd phase	19	304	155	21	1	-	8.36	0.40
Semi-strict 2nd phase	31	357	106	6	-	-	8.17	0.29
Strict 2nd phase	3	245	225	24	3	-	8.56	0.39

Table 3-58: Comparative time for varied routing on 256 node cube

	8	9	10	11	12	Mean	Variance
Random 1st phase	30	350	113	7	-	9.19	0.30
Semi-strict 1st phase	64	364	69	3	-	9.02	0.29
Strict 1st phase	55	403	42	-	-	8.97	0.19
Random 2nd phase	15	297	170	16	2	9.39	0.39
Semi-strict 2nd phase	29	356	106	8	1	9.19	0.32
Strict 2nd phase	4	196	250	48	2	9.69	0.44

Table 3-59: Comparative time for varied routing on 512 node cube

It might be expected that the imposition of less random routing strategies would worsen the completion time. It can be seen that this is true for the second phase with the strict strategy, but a small improvement is obtained in all the other cases. Looking at the other experimental statistics, the maximum node population distributions are similar for all three strategies, but the maximum queue length distributions do provide interesting information, and so they will be reproduced here.

	2	3	4	5	6	Mean	Variance
Random 1st phase	229	264	7	-	-	2.56	0.27
Semi-strict 1st phase	342	157	1	-	-	2.32	0.22
Strict 1st phase	344	155	1	-	-	2.31	0.22
Random 2nd phase	149	330	21	-	-	2.74	0.27
Semi-strict 2nd phase	198	287	15	-	-	2.63	0.29
Strict 2nd phase	6	319	159	14	2	3.37	0.34

Table 3-60: Comparative max queue for varied routing on 256 node cube

	2	3	4	5	6	Mean	Variance
Random 1st phase	108	377	14	1	-	2.82	0.22
Semi-strict 1st phase	220	275	5	-	-	2.57	0.27
Strict 1st phase	195	295	10	-	-	2.63	0.27
Random 2nd phase	46	417	37	-	-	2.98	0.17
Semi-strict 2nd phase	102	378	19	1	-	2.84	0.22
Strict 2nd phase	-	194	266	36	4	3.70	0.40

Table 3-61: Comparative max queue for varied routing on 512 node cube

From this, it can be seen that the maximum queue lengths occurring in the second phase with the strict strategy are dramatically worse. The

explanation for this lies, as so often, in the initial queues. Instead of the first phase arrivals being randomly distributed amongst queues, there is now a preponderance of packets in the queues for edges of the lower dimensions. This has the effect of lengthening the completion time.

In the other cases, maximum queue lengths are reduced corresponding to reductions in completion time. This is because, when two packets arrive at a node at the same time interval, the probability of them both joining the same queue is reduced. A limited form of pipelining has been introduced, in that packets arriving on different edges are likely to depart on different edges since they follow separate paths in order of increasing dimension. Thus, packets progress more freely and are not subjected to the random collisions which are forced by the original strategy. It is hard to separate the completion times for the semi-strict and strict strategies within the limits of experimental error, but the marginal improvement for the strict scheme does correspond with the intuition that packets are more likely to proceed independently when their dimension traversals are more in step.

Therefore, although strict routing is used analytically to guarantee "symmetry", i.e. each edge having equal probability of usage during a run, its practical effect is that any gains result, in fact, from a lack of symmetry in the behaviour of packets at nodes. This has been a characteristic brought out by most of the experiments: consideration of node balance at each time interval appears more useful than consideration of edge balance over the whole run.

3.7.2 Optimised routing for the d-shuffle

Next, an optimised path for packets being routed in a d-shuffle graph is considered. The standard algorithm makes each packet traverse a path of length n, where n is the diameter of the graph. However, as mentioned earlier, a shorter path is possible if the source and destination node representations have a substring in common (i.e. a path from $\alpha\beta$ to $\beta\gamma$ for some α, β, γ has length $n-|\beta|$). Therefore, expected path lengths can be reduced, at the expense of introducing more variability, and so expected completion time should decrease.

Analytically, it is often necessary to modify routes in this way in order to prevent multiple intersections of paths (or packets) during a run (the non-repeating property). Since there is at most one shortest route between any two nodes, packets cannot suffer two different intersections while en route.

The representative graphs used here are a diameter six 3-shuffle and diameter four 4-shuffle, which will be the standard examples throughout this section in view of the routing utility of these degrees, together with diameter eight and nine 2-shuffle graphs, which are of interest in this particular case. Only the completion time results are given here. Other statistics are omitted, but may be referred to to explain the results.

	9	10	11	12	13	14	Mean	Variance
Normal 1st phase	54	334	99	12	1	-	10.14	0.40
Revised 1st phase	86	317	90	6	1	-	10.04	0.42
Normal 2nd phase	3	287	182	22	5	1	10.48	0.43
Revised 2nd phase	16	259	192	28	4	1	10.50	0.50

Table 3-62: Time using varied routing on 729 node 3-shuffle

	6	7	8	9	10	Mean	Variance
Normal 1st phase	281	203	16	-	-	6.47	0.31
Revised 1st phase	287	195	15	2	1	6.47	0.36
Normal 2nd phase	149	318	30	3	-	6.77	0.33
Revised 2nd phase	164	299	36	1	-	6.75	0.34

Table 3-63: Time using varied routing on 256 node 4-shuffle

	11	12	13	14	15	16	17	18	Mean	Variance
Normal 1st phase	-	103	257	102	31	7	-	-	13.16	0.76
Revised 1st phase	50	251	138	47	11	3	-	-	12.45	0.84
Normal 2nd phase	-	12	203	181	83	16	4	1	13.81	0.86
Revised 2nd phase	4	110	243	108	31	2	2	-	13.13	0.80

Table 3-64: Time using varied routing on 256 node 2-shuffle

It can be seen that the completion time is only significantly affected in the case of the 2-shuffle graph. In all cases, mean maximum node

	13	14	15	16	17	18	19	20	21	22	Mean	Variance
Normal 1st phase	3	107	243	103	33	8	1	1	0	1	15.19	1.00
Revised 1st phase	46	236	157	41	17	2	0	1	-	-	14.52	0.91
Normal 2nd phase	-	26	185	198	66	19	4	2	-	-	15.77	0.95
Revised 2nd phase	7	115	227	112	27	11	0	1	-	-	15.15	0.95

Table 3-65: Time using varied routing on 512 node 2-shuffle

populations were slightly increased, as a consequence of the reduced balance in the network.

Of course, given the small number of d-shuffle graphs considered, it would be rash to attempt to make general statements about the behaviour of completion time. It can be seen that the gains for the 2-shuffle are approximately the same in both graphs and in both phases. An explanation of this and the absence of improvement in the higher degree cases is provided by the following result concerning expected path length under the new regime :

Lemma (3.5): Let $s(i)$ be the expected number of shuffle steps to move from vertex i to a random vertex j in a d-shuffle graph with d^n vertices. Then, for all i,

$$n - \frac{d}{(d-1)^2} + \frac{1}{(d-1)d^n}\left(n + \frac{d}{d-1}\right) \leq s(i)$$

and

$$s(i) \leq n - \frac{1}{d-1} + \frac{1}{(d-1)d^n}$$

Proof: Let $\sigma(k)$ be the number of vertices which can be reached using at most k shuffle steps from some fixed start vertex. Clearly, $\sigma(n) = d^n$ and, for $0 \leq k \leq n-1$,

$$\sigma(k) \leq \sum_{l=0}^{k} d^l = \frac{d^{k+1}-1}{d-1}$$

and

$$d^k \leq \sigma(k)$$

Now,

$$s(i) = \sum_{k=1}^{n} k.\frac{\sigma(k)-\sigma(k-1)}{d^n}$$

$$= n - \frac{1}{d^n}\sum_{k=0}^{n-1} \sigma(k)$$

and so

$$n - \frac{1}{d^n}\sum_{k=0}^{n-1} \frac{d^{k+1}-1}{d-1} \leq s(i) \leq n - \frac{1}{d^n}\sum_{k=0}^{n-1} d^k$$

Summing the series gives the required result.

The bounds on s(i) are tight. The lower bound is achieved when the start vertex is $0^{n-1}1$, for example, and the upper bound is achieved when the start vertex is 0^n, for example.

The lemma shows that the worst case expected path length reduction from the previous constant value for all nodes is $\frac{1}{d-1}$ for the d-shuffle. Since path lengths are integral, there can only be a significant reduction when d equals two. This corresponds with the observation made about completion time.

The fact that the only effect of introducing the revised routing scheme is to reduce completion time because of shorter path lengths indicates that the problems regarding repeating routes in analytic proofs are not significant. Note that the rate of increase in completion time with diameter does not change significantly (looking at results for the 2-shuffle over the complete range of diameter). This is consistent with the hypothesis about completion time, given that the improvement in expected path length is only a constant.

3.7.3 Variation of queueing discipline

Having observed that the imposition of shortest path routing on a d-shuffle does not degrade performance, and may indeed improve it, another modification, applicable to all graphs, is examined. This is the choice of queueing discipline employed at nodes. Some analytic proofs do not make any constraints on queueing discipline, as long as the first item on a non-empty queue is always defined. Others, however, are more specific and demand that packets which have the furthest distance to go should be given priority. Intuitively, this should speed up the overall algorithm. Here, three queueing disciplines are compared for both the cube and the d-shuffle. They are the normal first-come first-served (FCFS), furthest-to-go first-out (FTGFO), and remove in random order (RIRO). The graphs are the 2^8 and 2^9 node cube, and the 3^6 and 4^4 node d-shuffle.

	7	8	9	10	11	Mean	Variance
FCFS 1st phase	39	355	98	8	-	8.15	0.32
FTGFO 1st phase	150	350	-	-	-	7.70	0.21
RIRO 1st phase	35	325	121	19	-	8.25	0.40
FCFS 2nd phase	19	304	155	21	1	8.36	0.40
FTGFO 2nd phase	156	344	-	-	-	7.69	0.21
RIRO 2nd phase	10	292	169	27	2	8.44	0.42

Table 3-66: Time for varied queueing discipline on 256 node cube

	8	9	10	11	12	Mean	Variance
FCFS 1st phase	30	350	113	7	-	9.19	0.30
FTGFO 1st phase	183	317	-	-	-	8.63	0.23
RIRO 1st phase	19	329	142	10	-	9.29	0.32
FCFS 2nd phase	15	297	170	16	2	9.39	0.39
FTGFO 2nd phase	177	323	-	-	-	8.65	0.23
RIRO 2nd phase	4	290	180	23	3	9.46	0.39

Table 3-67: Time for varied queueing discipline on 512 node cube

From the tables, it can be seen that FTGFO always makes an improvement to the completion time for FCFS, as expected. Indeed, for the 512 node cube, the completion times are now essentially optimal with respect to the routing strategy. In this case,

	9	10	11	12	13	14	15	16	Mean	Variance
FCFS 1st phase	54	334	99	12	1	-	-	-	10.14	0.40
FTGFO 1st phase	180	271	41	7	0	0	0	1	9.76	0.51
RIRO 1st phase	1	131	255	91	21	0	1	-	11.01	0.65
FCFS 2nd phase	3	287	182	22	5	1	-	-	10.48	0.43
FTGFO 2nd phase	208	249	41	2	-	-	-	-	9.67	0.41
RIRO 2nd phase	-	26	249	157	57	9	2	-	11.56	0.73

Table 3-68: Time for 729 node 3-shuffle

	6	7	8	9	10	Mean	Variance
FCFS 1st phase	281	203	16	-	-	6.47	0.31
FTGFO 1st phase	308	179	13	-	-	6.41	0.29
RIRO 1st phase	104	317	72	7	-	6.96	0.41
FCFS 2nd phase	149	318	30	3	-	6.77	0.33
FTGFO 2nd phase	328	163	9	-	-	6.36	0.27
RIRO 2nd phase	28	292	151	26	3	7.37	0.48

Table 3-69: Time for 256 node 4-shuffle

Prob(no packet has a length 9 route)

$$= \left(1 - \frac{1}{512}\right)^{512} \simeq 0.367$$

and

Prob(no packet has a length 8 or 9 route)

$$= \left(1 - \frac{1}{512} - \frac{9}{512}\right)^{512} \simeq 0.000$$

so, over 500 experiments, it is expected that 184 will have a maximum path of length eight and 316 will have a maximum path of length nine. This correlates remarkably with the experimental result, indicating that queueing is not affecting the completion time. In the case of the 256 node cube, where packets with relatively long path lengths are more likely, the completion time is less than optimal, so some of the length seven packets are obviously not getting a "clear run" without queueing.

For the d-shuffle, such a striking improvement is not possible because all packets have the same path length, rather than there being a few which can extract maximum benefit from FTGFO. Any gains will be achieved by balancing the rate of progress of packets through the network, and so achieving a fairly uniform arrival time for all packets, with no bad worst

cases. The 3-shuffle graph shows more evidence of improved completion time than the 4-shuffle, given the limited number of results. This would be expected since, with the higher degree, there is less likelihood of packets being delayed for several time periods. Also, in this case, the diameter, and hence path lengths, are smaller and so give less scope for variations in completion time to develop.

For all of the graphs, it can be seen that FTGFO has the effect of reducing the completion time for the second phase so that it is approximately equal to that for the first phase. Throughout, it has been assumed that the difference between phases was due to dispersing the initial node populations in the second phase. This is supported by the behaviour here because, for the cubes, the queues will be emptied in order of distance to be travelled which avoids all but very improbable initial delays occurring, and for the d-shuffle (modulo the freak first phase completion time for one of the 3-shuffle experiments), any packets detained on initial queues will catch up more fortunate packets during the run, rather than being permanently disadvantaged.

Unlike FTGFO, there is no reason to expect that RIRO would lead to improvements over FCFS, and this is borne out by the results. For the cubes, there is a slight, but not serious, deterioration in all cases. This is further evidence that queues do not play a very significant part in causing delays; if they did, RIRO should have a retarding effect when packets were overtaken in a queue by later arrivals. The d-shuffles show a greater worsening under RIRO, more so in the second phase. This is due to an inversion of the effect produced by FTGFO. When all packets are travelling the same distance, similar progress rates are desirable, but RIRO only serves to increase variation in the time which packets spend in queues. Supporting evidence for this comes from the increased variance of completion time. The implication of the behaviour of RIRO is that analytic methods which do not make assumptions about queueing discipline may lead to less tight bounds on completion time. Of course, more "outrageous" disciplines than RIRO, such as last-come first-served, should lead to further slow downs, but are not studied here.

In all of the experiments, no significant changes in mean maximum node population and mean maximum queue length were seen. This is not surprising since the revised queueing disciplines affect the order in which packets flow through the network, but do not change their global distribution among nodes in the network throughout the run.

3.7.4 Single queue shared by all out edges

Having looked at the effect of varying the queueing discipline, the effect of having only one queue for outgoing packets at each node, instead of one queue per edge, will be examined. In a practical sense, this would model a processor with a shared output device, which could only transmit one packet per time interval. More abstractly, graphs in which many resources, i.e. edges, are unused at each time interval can be examined to assess the effect of explicitly making these resources available. With this in mind, the only graph considered here is the cube. Since it has logarithmic degree rather than constant degree, it is liable to suffer most from the queue restriction. However, if the routing was ideally balanced, the connectivity of the cube should mean that, on average, only one packet leaves each node at each time interval. Certainly, it is true that $(n-1)2^n$ edges of a $n2^n$ edge cube are unused at each time interval.

	7	8	9	10	11	12	13
8 queue, 1st phase	39	355	98	8	–	–	–
1 queue, 1st phase	–	–	9	112	223	113	32
8 queue, 2nd phase	19	304	155	21	1	–	–
1 queue, 2nd phase	–	–	–	3	88	202	134

14	15	16	17	Mean	Variance
–	–	–	–	8.15	0.32
13	3	–	–	11.22	1.02
–	–	–	–	8.36	0.40
57	10	5	1	12.42	1.14

Table 3-70: Time for varied queues on 256 node cube

All of the statistics collected in this case are reproduced, in view of the fundamental change made to the structure of the algorithm, which obviously

	8	9	10	11	12	13	14	15	16
9 queue, 1st phase	30	350	113	7	-	-	-	-	-
1 queue, 1st phase	-	-	-	8	161	212	88	27	3
9 queue, 2nd phase	15	297	170	16	2	-	-	-	-
1 queue, 2nd phase	-	-	-	-	13	129	228	88	28

17	18	19	20	21	22	Mean	Variance
-	-	-	-	-	-	9.19	0.30
1	-	-	-	-	-	12.96	0.86
-	-	-	-	-	-	9.39	0.39
8	4	1	0	0	1	14.08	1.21

Table 3-71: Time for varied queues on 512 node cube

	4	5	6	7	8	9
8 queue, 1st phase	23	304	150	21	2	-
1 queue, 1st phase	3	198	211	73	12	3
8 queue, 2nd phase	64	334	91	11	-	-
1 queue, 2nd phase	-	158	224	83	32	2

10	11	12	Mean	Variance
-	-	-	5.35	0.43
-	-	-	5.80	0.68
-	-	-	5.10	0.39
-	-	1	6.00	0.85

Table 3-72: Max node population for varied queues on 256 node cube

	4	5	6	7	8	9
9 queue, 1st phase	-	203	243	50	2	2
1 queue, 1st phase	-	43	241	164	42	9
9 queue, 2nd phase	2	302	168	24	4	-
1 queue, 2nd phase	-	29	238	161	54	9

10	11	12	Mean	Variance
-	-	-	5.71	0.48
1	-	-	6.47	0.72
-	-	-	5.45	0.40
6	2	1	6.61	0.96

Table 3-73: Max node population for varied queues on 512 node cube

affects queue sizes, and may in turn affect node populations. Also, given

	2	3	4	5	6	7	8	9	10	11	Mean	Variance
8 queue, 1st phase	229	264	7	-	-	-	-	-	-	-	2.56	0.27
1 queue, 1st phase	-	-	125	274	81	16	4	-	-	-	5.00	0.61
8 queue, 2nd phase	149	330	21	-	-	-	-	-	-	-	2.74	0.27
1 queue, 2nd phase	-	-	12	205	193	67	22	0	0	1	5.77	0.82

Table 3-74: Max queue length for varied queues on 256 node cube

	2	3	4	5	6	7	8	9	10	11	12	Mean	Variance
9 queue, 1st phase	108	377	14	1	-	-	-	-	-	-	-	2.82	0.22
1 queue, 1st phase	-	-	3	214	215	55	13	-	-	-	-	5.72	0.59
9 queue, 2nd phase	46	417	37	-	-	-	-	-	-	-	-	2.98	0.17
1 queue, 2nd phase	-	-	-	60	253	140	32	11	2	1	1	6.39	0.89

Table 3-75: Max queue length for varied queues on 512 node cube

the tight coupling between queue size and node population, the effect of arrivals on maximum node population may be observed easily.

The fundamental change is, of course, in maximum queue length. It has increased, in all cases, to a level greater than that of the maximum node population for the multi-queue case. Thus, new bottlenecks are being created, in addition to packets being forced to use the same queue. On average, the "bottleneck increase" in queue size is not large but, considering the entire distribution, the variance is relatively large. The effect of this is to make the distributions of completion time and maximum node population more variable than before. As an example, in both an eight and a nine dimensional experiment, the impact of one particular bad

queue can be traced through all of the results.

The completion time, as well as being greater for the one queue scheme, grows at a faster rate with the dimensionality of the cube. This is verified by results not included here. As observed previously, for schemes in which node behaviour with respect to arrivals and departures is not symmetric, completion time is still linear in the expected path length, but the constant of proportionality is larger. The constant is still approximately the same for both phases, despite the increased difference in queue sizes, which only affect the mean completion time by a constant amount.

The maximum node populations are only increased by a relatively small amount, the difference being more prominent in the second phase. This is encouraging, on average, since it means that packets are still being fairly evenly distributed about the network. The increased variance, however, means that worse imbalance occurs in worst case runs. Atypically, maximum node populations are larger in the second phase than the first. This is because maximum queue lengths are sufficiently larger to dominate the effect of packets which have terminated at a node. Observing the difference between both maximum queue length and maximum node population in the two phases, long-awaited positive evidence in favour of the explanation that higher first phase populations are due to early arrivals is obtained. In these experiments, there is a very noticeable separation occurring.

The maximum queue length in the second phase is larger than that in the first in the one queue scheme. However, the difference decreases as cube size increases, since the effect of the initial bottlenecks at the start of the second phase becomes less pronounced as routing bottlenecks begin to dominate the queue length measurements. Note that, since packets on any queue are being sent along a random edge in a logarithmic degree graph, a large queue at a node is unlikely to re-form later at some succeeding node. The most extreme maximum queue lengths (and thus the larger variances) occur in the second phase, indicating that the worst queues do develop from the seed provided by an initial queue.

The imposition of the one queue restriction is a little unnatural with respect to the parallel model, which stresses communication complexity by regarding processing time as negligible. Upfal [51] has suggested that the operation of the scheme should be further restricted by only allowing one packet to arrive at, or one packet to depart from, any node in any time interval. Analytically, the best upper bound on completion time under the first restriction for the n-dimensional cube is $O(n^2)$, rather than $O(n)$. Thus it is of the same complexity as can be achieved using a deterministic scheme. The best upper bound under the second restriction is $O(n^3)$. Some limited experiments have been performed with a cube operating under Upfal's regime, and these indicate that an $O(n^2)$ mean completion time results, with an extremely large variance of the distribution.

3.7.5 Merging of routing phases

Finally in this section, another variation on the basic randomised routing scheme is considered. This is the merging of the two phases so that, instead of being run independently, they are overlapped. The motivation in doing this is to achieve gains in completion time by pipelining. The drawback, in this context, is that such a change makes the scheme non-testable because combined path lengths are no longer independent of the input permutation in general. For example, in the case of the n-dimensional cube, which will be examined, for the identity permutation, first and second phase path lengths are the same, the total length having mean n and worst case 2n. However, the inverting permuation, which sends packets to the node with the complemented binary address of the source, has every path of length precisely n. In fact, these two permutations have the two extreme distributions of total path length, and so are considered here to investigate the best and worst improvements possible by pipelining. It is important to remember, though, that these results do not apply to all permutations.

In the case of the d-shuffle, there is no such problem because all paths have the same length, namely the diameter. This feature means, however, that pipelining is less liable to produce significant gains since packets arrive at their first phase destinations at similar times and so the

two phases are inherently separate. No d-shuffle experiments are reported here.

Two variations are tested. In the first, full merging, a packet commences the second phase as soon as it completes the first phase and then proceeds to its final destination. The second variation, partial merging, behaves similarly except that packets still on the first phase are always given priority in queues. This is a higher level version of the furthest-to-go first-out queueing discipline seen earlier, and it occurs in a slightly modified form in the parallel communication scheme proposed by Upfal [51].

The results tabulated include the standard scheme implementing the inverting permutation, for completeness. Comparing the results for this with those for the identity permutation confirms the testability assumption for these two differing permuations.

	10	11	12	13	14	15
Identity, no merging	–	–	–	–	7	33
Identity, part merging	–	–	–	–	26	130
Identity, full merging	–	–	–	–	10	112
Inverse, no merging	–	–	–	–	2	37
Inverse, part merging	148	318	32	2	–	–
Inverse, full merging	134	335	31	–	–	–

16	17	18	19	Mean	Variance
239	149	62	10	16.51	0.83
238	101	5	–	15.86	0.69
217	137	21	3	16.11	0.79
239	180	48	4	16.51	0.62
–	–	–	–	10.78	0.33
–	–	–	–	10.79	0.29

Table 3-76: Total time for routing with merged phases on 256 node cube

The significance of the input permutation is immediately obvious from the results. The improvement obtained for the identity permutation is very little, whereas that for the inverting permutation is of the order of a 50%

	11	12	13	14	15	16	17
identity, no merging	-	-	-	-	-	3	30
identity, part merging	-	-	-	-	-	18	127
identity, full merging	-	-	-	-	-	12	88
inverse, no merging	-	-	-	-	-	-	25
inverse, part merging	35	378	82	5	-	-	-
inverse, full merging	30	391	76	3	-	-	-

18	19	20	21	22	Mean	Variance
209	200	49	8	1	18.58	0.72
236	107	11	1	-	17.94	0.72
217	143	33	7	-	18.24	0.89
219	184	61	11	-	18.63	0.71
-	-	-	-	-	12.11	0.26
-	-	-	-	-	12.10	0.23

Table 3–77: Total time for routing with merged phases on 512 node cube

reduction in completion time. These observations confirm the stature of the cube as a medium for routing with few delays per packet. When all paths have equal lengths, the mean completion time is much closer to the path length than for a comparable situation on, say, the d-shuffle. When the path length has a binomial distribution, the mean completion time is almost equal to the maximum path length.

There is no difference between the partial and full merging methods for the inverting permutation, whereas (as might be expected) there is for the identity. Assuming that packets travel at approximately similar rates, there is no advantage in giving priority to packets when all routes are the same length, since phases are irrelevant. However, when a long route in the first phase means also a long route in the second phase, packets still on the first phase should always take precedence.

The measurements of maximum node population and maximum queue length have been omitted, but the main features of each can be succinctly summarised. The distributions of maximum node population are similar for all experiments with merging. The means are close to the second phase means for separately phased routing, but the variances are smaller. This is as expected, given that the "unnatural" node populations generated at the

end of the first phase have been eliminated. The distributions of maximum queue size are again similar. However, the means are slightly larger than in the unmerged experiments, while the variances are slightly smaller. This is explained by the fact that the utilisation of the network has increased, with all packets now travelling to their final destination without artificial inter-phase waits. There are longer time periods during which the majority of packets are active, and so more opportunity for bottlenecks to develop.

In conclusion, it has been seen that meging of phases does not alter worst case complexity of randomised routing a great deal. Clearly, the partial merging technique cannot worsen completion time, and experimental results indicate that node populations are not worsened. In practical circumstances, substantial gains can be achieved for appropriate permutations. However, in a theoretical and experimental sense, it is little loss to consider separate phases since worst case permutation performance is normally of greatest interest.

3.8 Generalising permutations

All of the experimental results so far have been concerned with realising permutations. In this section, more general h-relations are considered, in which h packets start and finish at each node. It has already been observed that, for any fixed h, randomised routing algorithms are testable. Values of h from one (permutation) up to eight will be considered, firstly on the 2^7 and 2^8 vertex cubes and secondly on the 3^6 and 4^4 vertex d-shuffles. For reasons of space, the style of presentation reverts to summaries of mean, variance, and maximum only for the three usual measures.

The effect of increasing h from one is that the minimum worst case completion time for a packet must be increased from n, the dimensionality of the cube, to n + (h-1), corresponding to a packet with maximum path length being last to leave its source node. Looking at the maximum completion times recorded, they approximately follow this rule. The mean completion time also grows with h, but with a constant of proportionality

h	Mean	Variance	Maximum
1	7.11	0.33	9
2	8.02	0.37	10
3	8.80	0.48	11
4	9.61	0.54	13
5	10.42	0.74	17
6	11.19	0.78	14
7	12.05	0.85	16
8	12.76	0.97	17

Table 3–78: Time for 1st phase of h–relations on 128 node cube

h	Mean	Variance	Maximum
1	7.30	0.40	10
2	8.19	0.40	10
3	8.97	0.49	12
4	9.81	0.62	12
5	10.60	0.64	14
6	11.38	0.86	16
7	12.10	0.87	16
8	13.00	1.18	18

Table 3–79: Time for 2nd phase of h–relations on 128 node cube

h	Mean	Variance	Maximum
1	8.15	0.27	10
2	9.15	0.36	11
3	9.89	0.46	12
4	10.65	0.59	14
5	11.46	0.56	15
6	12.30	0.71	15
7	13.05	0.74	17
8	13.89	0.84	18

Table 3–80: Time for 1st phase of h–relations on 256 node cube

less than one. The encouraging feature of the results is that, in the vast majority of experiments, increased completion time is no more than the time taken to disperse h packets from a source. Therefore, the network gives the appearance of processing h times as much data with only an increase of h in time. This is classical behaviour of a pipelined system but, in this case, that impression is not true. The point is that, given the

h	Mean	Variance	Maximum
1	8.32	0.32	10
2	9.26	0.42	11
3	10.03	0.48	13
4	10.88	0.66	14
5	11.58	0.57	15
6	12.35	0.62	16
7	13.11	0.76	16
8	13.99	0.95	19

Table 3-81: Time for 2nd phase of h-relations on 256 node cube

high degree of the cube, an initial population will be quickly dispersed with high probability and it is the high redundancy of edges which allows a greatly increased throughput without greatly increased delays. Obviously, the scope for bottlenecks is greater, and this is reflected in the increasing variance, which indicates that bad routes are sufficiently rare not to occur in every experiment.

To flesh out these remarks, the statistics for node population and queue length will be examined.

	Max. population			Max. queue length		
h	Mean	Var	Max	Mean	Var	Max
1	4.90	0.53	8	2.26	0.19	3
2	7.18	0.82	11	3.10	0.18	5
3	9.02	0.94	14	3.68	0.36	6
4	10.86	1.29	16	4.34	0.39	7
5	12.55	1.61	20	4.89	0.50	10
6	14.16	1.75	20	5.48	0.51	8
7	15.66	1.72	21	6.03	0.60	10
8	17.15	1.85	23	6.49	0.69	10

Table 3-82: Max. popns and qs for 1st phase of h-rel on 128 node cube

The maximum node population is growing more rapidly than the other measures. Recalling that mean maximum node population was approximately O(log N) when permutations were realised on a N node cube and packets were randomly distributed, it appears that the mean here is O(log h log N) and so, for fixed graph size, this measure grows most

h	Max. population Mean	Var	Max	Max. queue length Mean	Var	Max
1	4.63	0.50	8	2.50	0.27	4
2	6.82	0.75	11	3.30	0.25	5
3	8.67	0.88	14	3.99	0.35	6
4	10.49	1.24	16	4.61	0.49	8
5	12.17	1.54	20	5.24	0.46	8
6	13.78	1.61	19	5.82	0.65	9
7	15.20	1.72	20	6.32	0.73	11
8	16.83	1.93	23	6.85	0.84	12

Table 3-83: Max. popns and qs for 2nd phase of h-rel on 256 node cube

h	Max. population Mean	Var	Max	Max. queue length Mean	Var	Max
1	5.41	0.49	8	2.56	0.38	4
2	7.68	0.58	11	3.30	0.26	6
3	9.72	1.09	14	3.99	0.31	6
4	11.55	1.17	17	4.55	0.36	7
5	13.32	1.44	20	5.19	0.38	7
6	14.88	1.45	21	5.72	0.47	9
7	16.59	1.67	23	6.27	0.67	10
8	18.23	1.75	24	6.82	0.63	10

Table 3-84: Max. popns and qs for 1st phase of h-rel on 512 node cube

h	Max. population Mean	Var	Max	Max. queue length Mean	Var	Max
1	5.09	0.43	8	2.72	0.26	5
2	7.28	0.58	10	3.47	0.32	5
3	9.29	1.03	14	4.21	0.31	6
4	11.11	1.15	17	4.87	0.50	7
5	12.82	1.41	20	5.49	0.48	8
6	14.36	1.28	20	5.98	0.61	10
7	16.07	1.61	20	6.53	0.60	9
8	17.63	1.57	23	7.01	0.67	11

Table 3-85: Max. popns and qs for 2nd phase of h-rel on 512 node cube

rapidly in the range of h under consideration. However, because these populations are distributed amongst log N queues, the maximum queue sizes (and hence delays) do not increase by a similar amount. Repeating the same argument would imply that mean queue size was

O(log log h log log N), but obviously the experimental results are not accurate enough to speculate about this. As evidence for the maximum node population mean, comparing the results for the eight and nine dimensional cubes shows that, for each h, the ratio of corresponding means is approximately the same.

Both the completion time and maximum queue length grow at similar rates during both phases. The maximum node population grows faster during the first phase since, as h increases, so does the number of packets which are destined for each node. The limited number of experiments performed here are not sufficient to make statements about implementing h-relations on the cube with absolute certainty. However, it does appear that, if additional storage capacity is provided at nodes of the cube, then routing time is fast, in the sense that it increases by only an additive term in h, rather than the multiplicative factor suggested by analytic proofs.

It might be expected that the degree-bounded d-shuffle will not show the same capacity for handling the increased load. This will now be investigated, again in a rather restricted manner.

h	Mean	Variance	Maximum
1	10.16	0.41	13
2	13.30	0.77	17
3	16.47	1.24	22
4	19.36	1.32	24
5	22.31	1.55	29

Table 3-86: Time for 1st phase of h-relations on 3^6 node 3-shuffle

h	Mean	Variance	Maximum
1	10.47	0.43	14
2	13.61	0.74	17
3	16.67	1.26	22
4	19.62	1.55	25
5	22.62	1.68	28

Table 3-87: Time for 2nd phase of h-relation on 3^6 node 3-shuffle

h	Mean	Variance	Maximum
1	6.54	0.37	9
2	8.37	0.49	11
3	10.16	0.74	15
4	11.79	0.94	16
5	13.44	1.16	18
6	15.04	1.36	20
7	16.74	1.55	21
8	18.24	1.44	24

Table 3-88: Time for 1st phase of h-relation on 4^4 node 4-shuffle

h	Mean	Variance	Maximum
1	6.75	0.38	9
2	8.50	0.52	12
3	10.26	0.75	13
4	11.92	0.93	15
5	13.47	1.17	18
6	15.22	1.38	21
7	16.68	1.47	22
8	18.24	1.87	24

Table 3-89: Time for 2nd phase of h-relation on 4^4 node 4-shuffle

For both graphs, the completion time again grows by a linear term in h. However, with a more restricted degree, the linear factor is somewhat larger. Compared with the cube, the variability of the distribution is larger. Since the ability of a node to disperse abnormal populations is at the root of these observations, further discussion is postponed until the other statistics have been tabulated.

	Max. population			Max. queue length		
h	Mean	Var	Max	Mean	Var	Max
1	5.61	0.52	8	3.76	0.33	7
2	8.15	0.67	12	5.55	0.51	8
3	10.33	0.90	15	7.11	0.76	12
4	12.35	1.03	17	8.55	0.73	12
5	14.23	1.14	18	9.81	0.93	13

Table 3-90: Max popns and qs for 1st phase of h-rels on 3^6 node 3-shuff

h	Max. population Mean	Var	Max	Max. queue length Mean	Var	Max
1	5.54	0.54	8	4.08	0.32	7
2	8.12	0.69	11	5.90	0.59	9
3	10.33	1.02	15	7.58	0.73	11
4	12.35	0.99	17	8.99	0.84	13
5	14.29	1.25	19	10.40	1.17	15

Table 3-91: Max popns and qs for 2nd phase of h-rels on 3^6 node 3-shuff

h	Max. population Mean	Var	Max	Max. queue length Mean	Var	Max
1	4.99	0.46	8	3.11	0.19	5
2	7.34	0.80	10	4.20	0.35	7
3	9.32	0.97	14	5.31	0.50	8
4	11.11	1.04	16	6.26	0.57	10
5	12.76	1.33	17	7.20	0.68	11
6	14.54	1.66	22	8.07	0.83	11
7	16.17	1.81	21	8.96	1.09	13
8	17.80	1.98	24	9.69	0.88	14

Table 3-92: Max popns and qs for 1st phase of h-rels on 4^4 node 4-shuff

h	Max. population Mean	Var	Max	Max. queue length Mean	Var	Max
1	4.95	0.48	7	3.19	0.21	5
2	7.25	0.79	10	4.46	0.43	8
3	9.26	1.06	14	5.60	0.57	9
4	11.09	1.06	16	6.66	0.70	10
5	12.80	1.24	17	7.53	0.76	11
6	14.51	1.80	23	8.55	0.98	13
7	16.19	1.76	21	9.33	1.05	14
8	17.80	1.97	26	10.20	1.26	14

Table 3-93: Max popns and qs for 2nd phase of h-rels on 4^4 node 4-shuff

Comparing the second phase mean maximum node population of the 256 node d-shuffle and the 256 node cube for different values of h, it can be seen that they are approximately equal. This means that the hypothesis, predicated on random distribution of packets about the network, about maximum node populations being a function of graph size rather than graph structure, still applies when there are more packets than nodes. This fair

distribution of packets among nodes is achieved despite an increasingly large difference in maximum queue length between the cube and the d-shuffle. Of course, the variance of the maximum node population distribution increases with h in both cases, but more detailed significance testing of the two distributions indicates that it is reasonable to assume that the samples come from the same distribution.

In both phases of routing on the d-shuffle networks, the completion time and maximum node population distributions are similar. However, the maximum queue length distributions differ, with larger queues occurring in the second phase. This indicates that the worst queues which form are not having a dominating effect on the performance of the system. The larger queues in the second phase do form at the most appropriate time, namely at the beginning of the phase, and so have the most opportunity to disperse without prolonging completion time.

The maximum queue sizes are now becoming large, and it must also be the case that all queues are non-trivial since the expected number of packets at a node at any time, h, exceeds the number of queues (here three or four). Therefore, the simple hypothesis relating mean completion time to expected path length must be modified. At each node on its path, a packet is expected to meet (h-1) other packets and use the same output queue as any one of them with probability $\frac{1}{d}$. So, to take account of overloading of the system by a factor of h, it is reasonable to expect an increase in completion time of O(expected path length. (h-1). $\frac{1}{d}$).

Analysing the completion time results for the cube and the d-shuffle, it can be seen that the "linear term in h" does indeed follow this pattern, with the implied constant in the O(...) notation being around $\frac{3}{2}$. In the case of the cube, the function is truly a constant multiple of h. However, for the d-shuffle with fixed d, the function takes the form O(h.diameter), which means that completion time is as bad as suggested by analytic methods. While the 4-shuffle is superior to the cube when h = 1, it would be necessary to increase the degree to get comparable performance for h > 1 on some particular graph size. The node storage requirement would be

similar for all graphs of this size.

3.9 Experimental method

When analysing the behaviour of randomised computations by collecting statistics from simulation experiments, it is vital to ensure that the sample runs reflect the underlying distributions faithfully. In this section, two important factors will be considered. Firstly, the pseudo-random number generator used is tested for approximation to a true random number generator in the context of the simulation. Secondly, the number of experiments, which determines how closely observed results match expected results, is examined.

The pseudo-random number generator used was that supplied by the Vax library software, and it is of the linear congruential type with successive values computed by the relation

$$y_{i+1} = (69069.y_i + 1) \bmod 2^{32}$$

yielding a value in the range [0,1) from the most significant 24 bits. A value in the range [0,r) was obtained by calculating $\lfloor r.y_i \rfloor$. The initial value y_0 was given by the number of centiseconds which had elapsed in the current clock hour.

Various tests of the goodness of pseudo-random number generators are well-known, such as the serial correlation test and the spectral test [29]. Here, a method particular to the problem under discussion will be used. It is based upon some experiments done by Valiant [52]. However, the statistical analysis performed here will be different, and more applicable to the probability distributions involved.

Valiant compared the above procedure with a linear congruential random number generator, designed following the rules-of-thumb recommended by Knuth which guarantee passing of several of the tests for randomness. The standard two phase algorithm was run on an eight dimensional cube 500 times, using both random number generators, to realise three differene permutations. These were the identity, the complement, and the

i → (i+117) mod 256, permutations. The ad hoc acceptance criterion was whether the six sets of results could reasonably come from a common distribution. The results for completion time in the second phase (which has the least sharp distribution of the various statistics) were as follows :

	7	8	9	10	11
Identity, standard gen.	18	329	131	20	2
Complement, standard gen.	17	314	157	12	-
(i+117), standard gen.	20	318	147	14	1
Identity, "Knuth" gen.	24	309	154	12	1
Complement, "Knuth" gen.	23	305	153	17	2
(i+117), "Knuth" gen.	24	324	129	20	3

Table 3-94: Comparison of different random number generators

Valiant used a chi-square test to obtain slightly weak confidence limits on a common underlying distribution. However, this test assumes that the underlying distribution is normal, which is not the case here since, as has already been remarked, most of the observed distributions are skewed in the upper direction. Instead, the Kolmogorov-Smirnoff test is applied, since it is distribution-free. The test statistic is obtained by comparing the samples pairwise to find the largest difference in observed cumulative frequencies, and then dividing by the root of the sample size. Here, the value

$$\frac{348-328}{\sqrt{500}} = 0.89$$

is the test statistic. Since the threshold for the 0.05 significance level is 1.22, the conclusion is that there is a common underlying distribution.

Based on this result, the random number generator is taken as being acceptable for the experiments. Note also that the testability assumption has been checked in this experimental case. The identity and complementing permutations have been compared earlier. Here, a third permutation is introduced. Testability is closely linked to the goodness of the random number generator and, based on this limited sample, is assumed to be true in general. Of course, to be completely rigorous, each experiment attempted should be repeated, as a check, with other permutations and random number generators. In practice, this was not

done on cost grounds, and verification was limited to occasional spot checking of experiments for evidence of significant variation.

Given that the sampling technique is faithful, it is necessary to choose sample sizes which are adequate to approximate the underlying distribution. Throughout, the sample size used here has been 500 experiments. This was chosen bearing in mind both the available simulation resources and the significance required from the results. Since the principal measure of interest for each distribution has been the mean, it is necessary to establish adequate confidence limits on the estimate obtained by calculating the sample mean. As the distribution is not assumed to be normal, a rather weak confidence interval can be established for an arbitrary distribution using Chebychev's inequality giving that the estimated mean lies within

$$\sqrt{\left(\frac{v}{n\alpha}\right)}$$

of the true mean with probability $1-\alpha$, where v is the sample variance and n is the sample size.

The criterion for acceptibility here is that the above quantity, when α = 0.05 (i.e. with 95% confidence), is smaller than 0.05 of the estimate. When n is equal to 500, this means that $\sqrt{v}$ must be less than the sample mean divided by four. Most of the results given are comfortably within this bound, the only exception being those for some of the very smallest graph sizes. However, since the smallest results have generally been ignored because their underlying distribution differs from that for larger graphs, this is not a problem. Their role should be seen merely as one of completeness.

Earlier published results from similar experiments [53] used a sample size of 100. This was adequate for the limited range of graphs considered but the increased sample size here allows more scope (although, of course, a fivefold increase in error limits is not obtained, merely a $\sqrt{5}$-fold improvement).

Clearly, the addition of the second decimal place is rather gratuitous. In the discussion of the results, it has been disregarded. Indeed, the precision obtainable in any given experiment has always qualified the interpretation placed on the results, even if this is not explicitly pointed out in the text. Luckily, results for the main graphs considered, the cube and the d-shuffle, have sharp distributions and more detailed study of them is facilitated by the good approximations obtainable from a sample size of 500.

Note that the maximum observed sample, as remarked at the beginning, is included for informational purposes, rather than as a "good" parameter of the distribution - it is certainly not unbiased, but is consistent. The distribution tables in the later sections are intended to give an instant picture of the effect of modified algorithms. The parameter of interest is still the mean; no attempt is made to put confidence limits on the observed frequencies as indicators of the underlying distribution.

Chapter 4

Routing in parallel algorithms

4.1 Introduction

This thesis is primarily concerned with routing of data in order to simulate arbitrary parallel memory accesses. However, in this chapter, entire computations of algorithms on networks of parallel processors are examined in order to indicate that some of the ideas used in the analysis of routing (such as priorities, merging of phases) reappear at a more global level, expressing familiar concepts in the optimisation of parallel algorithms. An attempt is made to begin the construction of a framework in which it is possible to analyse parallel algorithms for processor networks by employing similar techniques to those used for routing analyses. It is felt that one major reason why there is a dearth of non-trivial algorithms for this type of parallel computer is that analytic tools are not available and so algorithms are either very special purpose for particular network structures, or for idealistic parallel models. As before, the approach concentrates exclusively on the movement of data between processors and thus the notion of run time is based upon communication time. Of course, there is no longer a fixed set of data since, if the computation is non-trivial, some transformations are inevitable.

The behaviour of the network is expressed as a collection of activities related by a notion of causality. Essentially, an activity consists of some processor transmitting data to its neighbours at some step in the computation, in a similar style to activities in the routing analyses of Aleliunas and Upfal. In its most general sense, an activity will be defined by

Definition (4.1): An <u>activity</u> is a quadruple [p,d,a,s] where

p is a processor in the network
d is a set of data involved in a particular computation
a is an algorithm step being applied to the data set d
s is a stage in routing of data at algorithm step a

Thus, [p,d,a,s] involves processor p transmitting packets, containing data from the dth set of data operated upon by the algorithm, which are at stage s of routing to satisfy the data exchanges at step a of the algorithm.

Sometimes, the structure of the network or the algorithm might make some of the four activity parameters redundant, or make some of them interdependent. In other cases, it might be convenient to view the parameters as being tuples themselves, in order to further refine the scope of an activity. However, in general, the triple (d,a,s) reflects three important levels of a parallel computation. Assuming that p, d, a, and s range over sets P, D, A, and S respectively, a few simple observations can be made immediately.

If |P| = 1, then the algorithm must be serial and, in the absence of communication problems, is not of interest here. If |D| = 1, then the algorithm handles only one set of data at any given time period. That is, there is no pipelining of data in the parallel processing sense of optimising resource usage by processing several different inputs at the same time. If |A| = 1, then the notion of an algorithm collapses into being a single routing problem of the type considered at length previously. If |S| = 1, the routing is simplified in the sense that each processor only communicates with its neighbours and so the concept of a routing node disappears. Clearly, if any of P, D, A, or S is empty, then analysis of communication behaviour is rather trivial.

A form of interdependence among the parameters p, d, a, and s which can occur frequently independently of particular algorithms, reflects an important design feature in the construction of parallel computers. That is, whether the network should be <u>expanded</u> or <u>recirculating</u>. Fundamentally, this reflects a decision on whether the concept of stages in a computation should be directly incorporated into a network so that its structure can be

viewed as consisting of a number of processing stages.

It has already been seen that, in the context of analytic proofs of routing network behaviour, the major advance in the approach of Aleliunas and Upfal over that of Valiant was that the behaviour of a particular processor could be viewed as a number of separate activities over time, rather than as a single entity throughout the computation. This enables a more precise analysis of collisions at the processor. In order to achieve the separation of processor activities over time, it was necessary to envisage a routing algorithm in which a processor does not deal with data packets at a given routing stage until all packets at earlier stages have been dealt with.

4.2 Staged networks

A more direct approach to implementing the desirable feature of staging is to expand the network in such a way that the processors are copied with different copies handling different stages. This is a familiar feature of parallel architectures, such as the sorting network of Batcher [5]. The final stage of the network may be connected back to the first stage: in this case, the network will be described as being wrapped.

As examples of expanded networks, the shuffle family may be considered. An expanded version of a d^n vertex d-shuffle (V, E) has nd^n vertices

$$\{(v, s) \mid v \in V,\ 0 \leqslant s \leqslant n-1\}$$

and edges

$$\{[(u, s), (v, (s+1) \bmod n)] \mid (u, v) \in E,\ 0 \leqslant s \leqslant n-1\}$$

That is, each shuffle edge moves to the next stage. The conventional expanded version of the shuffle-exchange graph is similar in that shuffle edges also change stage. However, exchange edges remain within a single copy of the graph.

It is interesting to note that the cube-connected cycle graph, which was motivated as a practical compromise which can simulate the properties of the n-dimensional cube, is in fact related to these expanded shuffle

graphs. It turns out that the CCC, when either undirected, or directed with backward cycle edges only, is isomorphic to the wrapped expanded shuffle-exchange graph. Further, the variant of the CCC, the so-called CCC^+, employed by Upfal is similarly just a wrapped expanded 2-shuffle graph.

Theorem (4.2): (for undirected graphs) (i) A CCC with $n2^n$ vertices is isomorphic to a wrapped expanded n-stage shuffle-exchange graph with 2^n vertices. (ii) A CCC^+ with $n2^n$ vetices is isomorphic to a wrapped expanded n-stage 2-shuffle graph with 2^n vertices.

Proof: In both cases, the CCC (CCC^+) vertex

$$(c, d),\ 0 \leq c \leq 2^n-1;\ 0 \leq d \leq n-1$$

is mapped to the expanded shuffle-exchange (2-shuffle) vertex

$$(c>>d, (n-1)-d)$$

where the notation c>>d means the binary representation of c shifted cyclically right by d places. It can easily be seen that this map is a bijection.

Also for both graphs, the edges

$$[(c, d), (c, (d-1) \bmod n)]$$

map to

$$[(c>>d, (n-1)-d),\ (c>>((d-1) \bmod n), ((n-1)-d+1) \bmod n)]$$

For the CCC^+, the edges

$$[(c_{n-1}\ldots c_{d-1}\ldots c_0, d), (c_{n-1}\ldots \overline{c_{d-1}}\ldots c_0, (d-1) \bmod n)]$$

map to

$$[(c_{d-1}\ldots c_0 c_{n-1}\ldots c_d, (n-1)-d),\ (c_{d-2}\ldots c_0 c_{n-1}\ldots \overline{c_{d-1}}, ((n-1)-d+1) \bmod n)]$$

For the CCC, the edges

$$[(c_{n-1}\ldots c_d\ldots c_0, d), (c_{n-1}\ldots \overline{c_d}\ldots c_0, d)]$$

map to

$$[(c_{d-1}\ldots c_0 c_{n-1}\ldots c_d, (n-1)-d),$$
$$(c_{d-1}\ldots c_0 c_{n-1}\ldots \overline{c_d}, (n-1)-d)]$$

Thus, the edges are mapped bijectively from one graph to the other. The underlying idea is that the traversal of a sequence of stages in the shuffle graph corresponds to traversing a cycle in the CCC graph, but in the opposite direction. Indeed, recent analyses of algorithms for the CCC^+, for both routing [51] and sorting [42] rely on this implicit staging. The 2-shuffle interconnection pattern is the key feature of the actual algorithms.

Of course, the advantages of an expanded network are bought at a price, namely the increase in graph size. It has always been an assumption here that the number of processors and the number of data packets is related by a constant factor. If this linear relationship is to be maintained, some algorithmic modifications are necessary. Either the size of the set of data handled by the algorithm can be enlarged to match the number of processors, or multiple sets of data, each of the original size, can be handled simultaneously. Typically this distinction depends upon whether the network is wrapped or not.

At this point, it is necessary to make the notion of a "set of data" more concrete. With respect to one particular computation being performed according to some algorithm, the data set at any point in time consists of all data packets which are being sent from one processor to another (whether queued or actually in transmission). That is, the contents of the packets in a data set correspond to those items which are currently being read from, or written to, the equivalent of a global memory in some instantiation of the algorithm. Initially, the data set may be considered as the collection of input data to the computation which arrives in the network. Ultimately, the data set may be considered as the collection of output data from the computation which leaves the network. During routing steps, packets may be copied or merged, in the sense of the section on multiple

reads and writes. It will be assumed that such alteration is done in a way which preserves the property that the total number of packets in the network at any time does not exceed the number of processors by more than a constant factor.

If the size of individual sets of data is of the same order as the number of processors in an n-stage expanded network, it is necessary for each processor to be able to handle any stage of routing in the algorithm if the input data is to be treated uniformly. For example, if one input datum starts at each processor, the data packet starting at (i,j), $0 \leqslant i \leqslant 2^n-1$; $0 \leqslant j \leqslant n-1$ say, will be handled by a processor at stage $(k+j)$ mod n at the kth step of its journey. An example of such an approach which has all processors in use simultaneously in an expanded network is Upfal's routing algorithm.

The other approach, namely handling multiple sets of data simultaneously, is more familiar, since it corresponds to the normal notion of pipelined processing. In its purest form, each stage in the network handles a different data set at each time interval, and moreover always handles the same step in the route of each data set. Examples of this are the systolic arrays of Kung [30], in which the expanded networks take the form of one, or two, dimensional arrays of uniform processors. Note, however, that the uniformity of processors is not essential. In a practical context, this approach is also more attractive since it allows a far smaller proportion of processors to be capable of handling input or output from the external world.

When an algorithm for an n-stage network is less straightforward, in the sense that data packets are involved in journeys with a total length greater than n, the distinction between the above two approaches becomes blurred. Pure pipelining is only intended for situations where essentially the structure of the network reflects the expanded structure of the algorithm. It is interesting to note, however, that pipelining of data sets may occur at an intermediate stage of the algorithm, even if all input data sets arrived at the same time.

Such behaviour can be observed in the sorting algorithm of Reif and Valiant [42] where, simplifying slightly, input data starts at each of the $n2^n$ nodes of a CCC^+ network and then all packets are routed to stage 0 processors. After this, the data packets are pipelined through the network, as though a sequence of data was being input to the stage 0 processors. Thus "subsets of data" are being dynamically formed during execution of the algorithm.

When the execution of an algorithm can be divided into a series of stages, one per time interval, it can obviously be easily analysed even if the network is not expanded. In such circumstances, where all activities are guaranteed to last for at most one time interval, a non-expanded network is usually described as a recirculating network. Examples of algorithms for such networks, including the fast Fourier transform, may be found in Stone [48]. It is interesting that many of these algorithms, intended for a recirculating shuffle-exchange network, reappear in Preperata and Vuillemin [40] as algorithms for a CCC network which is, of course, just an expanded shuffle-exchange network.

Of course, since the aim here is to relax the rigid connection between algorithms and networks, it is not generally possible to obtain analyses based upon exact notions of stages. In such circumstances, it is convenient to impose restrictions on the algorithm which make its behaviour over time more predictable. However, great care must be taken to ensure that such restrictions do not adversely affect performance in order to achieve analytic ends.

4.3 Delay sequences

To analyse the time complexity of parallel algorithms, it is necessary to consider not only the time required for each processor in the network to complete all of its activities, but also the delays which an activity may suffer due to non-completion of other activities.

Suppose that α_1 and α_2 are activities.

Definition (4.3): α_1 <u>delays</u> α_2, written $\alpha_1 \rightarrow \alpha_2$, if the transmission time of some packet in α_2 is delayed until some packet has been transmitted by α_1.

Definition (4.4): A <u>delay sequence</u> of activities $\alpha_1, \alpha_2, \ldots, \alpha_k$ consists of the sequence of delays $\alpha_1 \rightarrow \alpha_2 \rightarrow \ldots \rightarrow \alpha_k$.

Using the notation $S(\alpha)$ for the starting time of activity α and $F(\alpha)$ for the finishing time of α, the time requirement to complete all of the $\alpha_1, \ldots, \alpha_k$ in a delay sequence is

$$\max_{1 \leq i \leq k} F(\alpha_i) - \min_{1 \leq i \leq k} S(\alpha_i)$$

Since it is not usually possible to estimate the time exactly, it will be upper bounded by applying the following fact :

If $\alpha_1 \rightarrow \alpha_2$ then $F(\alpha_2) - |\alpha_2| \leq F(\alpha_1)$,
where $|\alpha|$ denotes the number of data packets transmitted by activity α.

If the fact was not true then α_1 could not be responsible for delaying the transmission of any data packet by α_2, contradicting $\alpha_1 \rightarrow \alpha_2$. From it, the following can be deduced :

$$\text{For all } i,\ F(\alpha_i) \leq F(\alpha_1) + \sum_{j=2}^{i} |\alpha_j|$$

and so

$$\max_{1 \leq i \leq k} F(\alpha_i) \leq F(\alpha_1) + \sum_{j=2}^{k} |\alpha_j|$$

Now define a delay sequence to be <u>complete</u> if there is no activity α which satisfies either $\alpha \rightarrow \alpha_1$ or $\alpha_k \rightarrow \alpha$. Then the fact that α_1 is undelayed implies that $S(\alpha_1) = 0$ and $F(\alpha_1) = |\alpha_1|$. This establishes the result that :

The time requirement for the complete delay sequence

$$\alpha_1 \rightarrow \ldots \rightarrow \alpha_k$$

is upper bounded by

$$\sum_{j=1}^{k} |\alpha_j|$$

Further, since α_k delays no other activity, it can be deduced that the run time required for a computation is upper bounded by the maximum, over all complete delay sequences, of the total number of data packets transmitted by activities in the sequence.

Since the run time is the measure of interest, it will be assumed from now on that delays sequences are complete, without loss of generality. Another assumption which can be made about the activities comprising a delay sequence is that, for any $\alpha_i \rightarrow \alpha_{i+1}$, there is not an activity α such that $\alpha_i \rightarrow \alpha$ and $\alpha \rightarrow \alpha_{i+1}$. One consequence of this is the following:

Lemma (4.5): Given any delay sequence under the above assumption, it has the property that, for any $[p,d,a,s] \rightarrow [p',d',a',s']$ in the sequence, either $p = p'$ or (p,p') is an edge in the network.

Proof: Suppose that $p \neq p'$ and (p,p') is not an edge. Because of the delay property, there must be a path from p to some processor p'' and an edge (p'',p') in order for $[p,d,a,s]$ to affect $[p',d',a',s']$. Clearly there exists a'',d'',s'' such that $[p'',d'',a'',s''] \rightarrow [p',d',a',s']$, and so $[p,d,a,s] \rightarrow [p'',d'',a'',s'']$ also. This contradicts the absence of an intermediate delaying activity.

The two types of neighbouring activity in a delay sequence illustrate the two fundamental sources of delay. If both activities occur at the same processor, then the delay is a consequence of competition for an outgoing edge. If the activities occur at adjacent processors, then the delay is a consequence of the sequence of operations performed by the algorithm or by routing.

It is important to note that, unlike usual critical path techniques, there is no constraint that a processor must only handle one activity at a time. It is assumed that, in each time interval, each processor will transmit as many packets as possible. There is not even a per se assumption that any activity will be given priority over another although, in practice, this is liable to be enforced by means of some ordering on triples (d,a,s). The manner in which activities are defined precludes analyses which take account of the order in which packets are transmitted during one particular activity since such a fine resolution does not appear to be very helpful, merely more complicated.

Having established this framework of activities and delay sequences, it is possible to explain how analysis of parallel algorithms can be attempted. Of course, just as analysis of serial algorithms often involves ad hoc procedures, it is not possible to give general purpose results. Rather, the important features of the analysis will be discussed, and useful properties of certain classes of algorithms will be investigated.

4.4 Analysing parallel algorithms for networks

4.4.1 Localisation of communication

Consider the original motivation for the study of routing. This was to simulate memory accessing at each step of the algorithm. If the algorithm is viewed as a sequence of discrete steps over time then, if routes may have length up to n, a typical delay sequence may look like :

$$[p_{00},0,0,0] \rightarrow [p_{01},0,0,1] \rightarrow \ldots \rightarrow [p_{0,n-1},0,0,n-1]$$

$$\rightarrow [p_{10},0,1,0] \rightarrow [p_{11},0,1,1] \rightarrow \ldots \rightarrow [p_{1,n-1},0,1,n-1]$$

$$\rightarrow \ldots$$

$$\rightarrow [p_{s-1,0},0,s-1,0] \rightarrow [p_{s-1,1},0,s-1,1] \rightarrow \ldots$$

$$\rightarrow [p_{s-1,n-1},0,s-1,n-1]$$

where there are s steps and one data set. Since it is known that the routing delay at each step is O(n), a total time bound of O(sn) can be obtained.

Unfortunately, such a simple-minded approach cannot usually yield anything better than crude upper bounds. This is for two main reasons. Firstly, at any step of the algorithm, memory accessing may be more localised, that is the routes followed by data packets can all have length less (sometimes considerably less) than the diameter of the network. Secondly, both algorithmic steps and processing of data sets may begin before previous steps and data sets have been completed. Both of these features may lead to substantial reductions in run time and, of course, both highlight the advantages of tailoring algorithms to networks, rather than mindlessly employing a general-purpose computer. It would be desirable to imagine some future optimising compiler which could fit programs to networks, much as present-day compilers match programs to instruction set peculiarities, but here it is impossible to do anything more than try to point in the correct direction.

A simple example of the impact which locality can have on the analysis is given by the Ascend/Descent class of Preparata and Vuillemin [40]. These have the desirable feature that, at each step, data is sent only to adjacent processors and moreover that no processor receives data from more than one neighbour. From this, it follows that any activity handles one packet only and that an activity can only be delayed by an activity with a smaller algorithm step parameter. Thus, the run time is the same as the number of algorithm steps.

More generally, one class of algorithms which is amenable to parallelisation with localised communication is that of so-called "divide and conquer" algorithms. In serial form, such an algorithm when applied to N items, partitions them into some number of sets and then applies itself recursively to each of the smaller sets. To obtain a parallel implementation, the aim is to route data at each partition step to an adjacent processor in such a way that the members of each set in the

partition become closer together in the network. As an example, suppose that N items have to be transformed in some way (for example, sorted, Fourier transformed) using a d-shuffle network with $N = d^n$ processors. Then the outline method is as follows, starting with a serial algorithm :

Call Transform (S, n) where |S| = N and $n = \log_d N$.

procedure Transform (**set** S, **integer** n)
  if n = 0 **then** transform S non-recursively **else begin**
    partition S into $S_1, \ldots, s_d$, preferably of equal size
    Transform (S_1, n-1)
    . . .
    Transform (S_d, n-1)
    recombine $S_1, \ldots S_d$ to form a new S
  end
end Transform

From it, a parallel algorithm can be obtained :

for each processor p in parallel **do begin**
  for step := n **downto** 1 **do begin**
    partition arrivals into $S_1, \ldots, S_d$
    transmit S_i along out edge i for each i in parallel
  end
  apply serial non-recursive algorithm to arrivals
  for step := 1 **to** n **do begin**
    combine arrivals into $S_1, \ldots, S_d$
    transmit S_i along out edge i for each i in parallel
  end
end

The efficiency of the algorithm depends crucially upon the partitioning process and the combination process. Note that, at step i, the sets of processors

$$\{ \{\alpha\beta \mid \alpha \in \{0, \ldots, d-1\}^i\} \mid \beta \in \{0, \ldots, d-1\}^{n-i} \}$$

correspond to the d^{n-i} subproblems which the serial algorithm considers. For some divide and conquer algorithms, such as FFT, partitioning and

combination can be done using information local to each processor and this leads to a fast parallel algorithm. Others, such as Quicksort, use information which is local to each set of processors, which could lead to communication overheads. In these cases, overlapping of algorithm steps is necessary if an $O(n)$ run time is to be achieved rather than the obvious $O(n^2)$ run time. One part of the Flashsort algorithm of Reif and Valiant [42] essentially demonstrates that such an optimisation is possible for Quicksort.

One important feature which divide and conquer algorithms should have is that their partitioning is approximately balanced (see, for example, Aho, Hopcroft, and Ullman [1]). In the way that they have been presented here, this ensures that the final non-recursive stage does not have to deal with a large subproblem. However, in the parallel context, another familiar advantage is highlighted, namely that the packets leaving a node are approximately evenly distributed. This is helpful when performing a detailed analysis of such algorithms. Thus it becomes easy to see the common underlying feature of the random choice of partitioners in Quicksort and the random selection of intermediate addresses in the two-phase routing algorithm.

4.4.2 Analysis of activities

When a parallel algorithm has been developed, it is necessary first to specify the set of activities and then determine all of the ways in which one activity may delay another. Given this, the set of all possible delay sequences can be investigated. In all but the simplest algorithms, it will be impossible to ascertain all interactions between packets and activities in delay sequences and so a probabilistic analysis of collisions is necessitated. This yields results which upper bound the number of data packets handled by each delay sequence with high probability, and hence upper bound the run time with high probability.

The composition of the set of delay sequences is very dependent upon the algorithm. As has been seen, various optimisations may be necessary in order to bound the maximum length of any such sequence. Clearly, the

length of delay sequences gives a lower bound on the run time of the computation. In addition, a crude upper bound on the number of delay sequences can be obtained from maximum sequence length and the maximum number of activities which one particular activity can affect. This will be useful if the set of delay sequences cannot be precisely enumerated.

In order to simplify the analysis of dependent activities, the delay sequence might be partitioned into a collection of subsequences, so that the set of data handled by the activities in a subsequence is independent of any activities in other subsequences. For example, one elementary partition could be based upon the data set associated with each activity in the delay sequence. A further partition may be done by noting that the sets of packets handled by the same processor appearing in adjacent activities are generally non-intersecting.

Now, a technique for probabilistic analysis of the number of packets handled by a delay sequence is considered. It incorporates notions due to both Aleliunas and Upfal, and is intended to be applicable over a wide range of algorithms with good probabilistic behaviour.

Considering the data packets handled by some activity in the sequence, if there are λ such packets then, given that the routing behaviour of the algorithm is oblivious, λ independent events can be considered, the jth event being that packet j is handled by a further activity in the sequence. Suppose that, for every packet which intersects the sequence, the probability of this event at any stage in the sequence can be upper bounded by, say, r_d for packet d, with $r_d \leq \frac{1}{2}$.

Then, independently of all other packets, it is true that, for any packet d which intersects the subsequence, the number of activities which handle it after the first can be upper bounded by a random variable X with

$$\text{Prob}\,(X = t) = (1-r_d)\,r_d^{\,t}$$

i.e. a geometric random variable with parameter r_d. The problem of estimating the delay then reduces to one of summing a set of independent

geometric random variables.

To do this, a result of Chernoff [11] is useful. From it,

$$\text{Prob}\left(\sum_d \geq x\right)$$

$$\leq z^{-x} \cdot \prod_d \frac{1-r_d}{1-r_d z} \quad \text{for any } z \in \left[1, 1/\left(\max_d r_d\right)\right)$$

Since all $r_d \leq 1/2$, this can be bounded by

$$z^{-x} \cdot \prod_d \left(1-\frac{1}{2}\right) / \left(1-\frac{z}{2}\right)$$

and by considering a constant $c \geq 1$ and letting $z = \frac{2c}{c+1}$, then if there are H intersecting packets,

$$\text{Prob}(\text{no. packets handled} \geq cH)$$

$$\leq \left[\frac{2c}{c+1}\right]^{-cH} \left[\left(1-\frac{1}{2}\right) / \left(1-\frac{c}{c+1}\right)\right]^H$$

$$= \left[\frac{c+1}{2}\left(\frac{1}{2}\left(1+\frac{1}{c}\right)\right)^c\right]^H$$

$$\leq \left[\frac{(c+1)e}{2^{c+1}}\right]^H$$

By choosing c large enough, this probability can be made arbitrarily close to zero, as H increases.

Having this result, it is necessary to determine H, the number of packets which may intersect the sequence. In general, this part of the analysis relies upon packets being randomly routed through the network if a probabilistic bound is required. The most convenient condition to apply to an algorithm to achieve this aim is one due to Upfal which, adjusted to the algorithmic context here, demands that the expected number of packets handled by any activity is less than or equal to one. Without this condition, some activities are expected to handle an atypical amount of traffic and more careful analysis would be required.

Suppose that there are at most T packets in the data set, and the length

of the sequence is λ. Then there are at most T independent Bernoulli trials with the expected number of successes being at most one, at each activity in the sequence. The theorem of Hoeffding bounds the probability of $m \geq 2$ successes by $B(m, T, \frac{1}{T})$. From this, the probability of achieving m intersections with the whole sequence is less than $B(m, T\lambda, \frac{1}{T})$.

Thus, using a binomial corollary to the result of Chernoff derived, for example, in [42],

For $0 < c' \leq 1$,

Prob($\geq (c'+1)\lambda$ intersecting packets) $\leq e^{-c'^2\lambda/2}$

The previous two results can be combined to prove the following :

Theorem (4.6): For any length λ delay sequence in an oblivious algorithm which has the properties that (i) no activity is expected to handle more than one packet, and (ii) if a packet is handled by one activity, then the probability of it later being handled by another is at most $\frac{1}{2}$.

For any constant $c \geq 1$,

Prob (Time taken for sequence $\leq 2c\lambda$)

$$\geq 1 - [e^{-\lambda/2} + (\frac{(c+1)e}{2^{c+1}})^{2\lambda}]$$

As a trivial example of the application of the theorem, consider a phase of the normal random routing algorithm, in a graph with degree d and diameter δ. Then all delay sequences have length δ and there are at most $N(d+1)^{\delta}$ of them. Therefore, applying the above, the probability of the run time exceeding $2c\delta$ for $c \geq 1$ is at most

$$N(d+1)^{\delta}\,[e^{-\delta/2} + (\frac{(c+1)e}{2^{c+1}})^{2\delta}]$$

which is bounded by N^{-k} for some k.

It should be noted that care must be taken when a packet is being sent to several destinations. For, in this case, the second property required by the theorem is unlikely to hold for the packet, which has a larger than typical probability of being handled by subsequent activities in the

sequence. In the case of broadcasting using a random routing approach, then the same idea as used in Theorem (2.10) can be applied, that is the behaviour of the broadcast packet can be upper bounded by that of a collection of independent packets, one corresponding to each address on the broadcast packet. If this is done, the same bound as that just obtained for the normal algorithm follows.

Chapter 5

Routing graphs and their relationship to grids

5.1 Introduction

In this chapter, the implications of technology on the graphs which underly routing networks are considered. To be specific, the particular technology is one in which it is most advantageous for vertices to be laid out in a regular pattern in a plane, with uniform length edges connecting adjacent vertices. This is the desirable type of graph for VLSI in order to achieve efficient use of chip area and minimum on-chip signal propagation time. Three types of graph which meet these constraints are triangular, rectangular, and hexagonal, arrays. Most designs employ rectangular arrays (a notable exception being the remarkable hexagonal matrix multiplier of Kung and Leiserson [34]), and so the two-dimensional grid will be taken as a technologically sound graph with which the routing graphs can be related. It may be noted that the use of grids for parallel architectures predates VLSI, and that they have been constructed in discrete component technologies.

The relationship between graphs will be that of embedding one graph in another, that is mapping vertices to vertices and edges to disjoint paths. More formally,

> **Definition (5.1):** If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, then an <u>embedding</u> $f : G_1 \rightarrow G_2$ has the properties that
>
> (i) f is a one-one map from V_1 into V_2
>
> (ii) f is a one-one map from E_1 into a set of edge-disjoint paths in G_2 such that the image of (u, v) under f has start vertex $f(u)$ and end vertex $f(v)$

Such an embedding is sometimes referred to as an <u>edge</u> embedding.

The motivation for considering embeddings involving routing graphs and grids is twofold. Firstly, embedding routing graphs in a grid corresponds to finding a two-dimensional rectangular layout of the graph. Secondly, embedding grids in routing graphs corresponds to simulating algorithms for grid architectures, on which much time has been spent, on the routing graph. In both cases, the aim is to determine whether it is reasonable to employ routing graphs for practical purposes in place of the grid which, as has been seen earlier, is not particularly useful for routing. The two directions of embedding will be considered in turn.

5.2 Embedding routing graphs in grids

The relevance of such embeddings to VLSI is provided by the VLSI circuit model of Thompson [50]. This allows the concepts of chip area and propagation time to be precisely defined. The assumptions made in this model regarding area are generally accepted, but those for time have provoked considerable controversy. In the model, processors and wires are laid out on a grid with processors at some grid vertices and wires following grid edges. No wire may cross a processor and each wire has a processor at each end. Two wires may not occupy the same path in the grid, and wires may only cross at grid vertices. The area of a layout is the area of the smallest rectangle that contains all processors and wires. The proposed propagation time for a wire is constant, regradless of the length of the path followed by the wire. More realistically, the time should be proportional to the path length [10], or even to the square of path length [7]. Note that, in this specific context, edges in the grid will often be referred to as tracks, reflecting common usage.

Clearly, the problem of finding an area-efficient layout of a graph is that of embedding the graph in a grid with a small number of vertices since area is equal to the grid vertex count. However, because the graphs used here have degree greater than four in general, the notion of embedding must be extended and, to do this, it is necessary to represent each embedded vertex by a set of adjacent vertices in the grid.

Note that, in this section, all edges will be considered as being

undirected. Since all directed grid edges have opposite partners, area results are affected by at most a constant factor. The assumption allows the results to be presented in a manner consistent with that of other researchers.

Definition (5.2): If $G = (V, E)$ is an undirected graph with degree d, then a <u>grid embedding</u> f of G in the grid with vertices (v_{ij}) has the properties that:

(i) f maps each $v \in V$ into a set of vertices

$$\{v_{ij} \mid r \leq i \leq r+\lceil d/4 \rceil - 1,\ s \leq j \leq s+\lceil d/4 \rceil - 1\}$$

for some r, s such that $\bigsqcup_{v_1 \neq v_2 \in V} (f(v_1) \cap f(v_2)) = \phi$

(ii) f maps each $(u, v) \in E$ into a path

$$(v_{i_1 j_1}, v_{i_2 j_2}),\ (v_{i_2 j_2}, v_{i_3 j_3}),\ \ldots,\ (v_{i_{r-1} j_{r-1}}, v_{i_r j_r})$$

for some $r \geq 2$ such that

$$v_{i_1 j_1} \in f(u),$$

$$v_{i_s j_s} \notin f(w)\ \forall\ 1 < s < r \text{ and } w \in V$$

$$v_{i_r j_r} \in f(v)$$

and $\bigsqcup_{e_1 \neq e_2 \in E} (f(e_1) \cap f(e_2)) = \phi$

Intuitively, each vertex is being mapped on to a "square" in the grid with perimeter $4\lceil d/4 \rceil$. Embedded edges are not allowed to cross these squares.

The results for lower bounds on the area required depend upon a theorem of Thompson. Before stating it, two further graph-theoretic definitions are required. Let graph $G = (V, E)$. If $E' \subseteq E$ then E' is said to <u>bisect</u> G if the removal of E' induces a partition of V into V_1 and V_2, each containing half of the vertices in V.

More formally,

(i) $V_1 \cup V_2 = V$

(ii) $|V_1| \leq |V_2| \leq |V_1|+1$

(iii) Every edge in $E \backslash E'$ has both endpoints in V_1 or in V_2

The minimum bisection width of G is the size of the smallest bisecting edge set E'.

Theorem (5.3): [Thompson] A layout of any graph with minimum bisection width ω and degree at most four must occupy at least $\Omega(\omega^2)$ area.

A detailed proof of the theorem may be found in [50]. As it stands, the theorem is not directly applicable to grid embeddings. However, since the the lower bound on area is obtained by considering embedded edges only, the result applies to the area occupied by edges in a grid embedding. The area occupied by vertices (which is zero for Thompson's embeddings) may also be included, giving the following :

Theorem (5.4): A grid embedding of any graph with minimum bisection width ω, N vertices and degree d must occupy at least $\Omega(Nd^2 + \omega^2)$ area.

For the cube and d-shuffle, the main graphs of interest, this theorem gives a tight lower bound on area since layouts will be demonstrated which achieve it. However, it will also be shown that any layouts of these graphs will inevitably have long paths, which implies large propagation times if Thompson's model of time is not accepted.

Before considering these graphs, a layout for the wrapped grid graph used in Chapter Three will be given. Clearly, the original grid graph has a trivial layout. The new layout involves "folding" such a layout about a horizontal, and a vertical, axis and interleaving the overlapping vertices and edges. Figure 5-1 illustrates the process.

Theorem (5.5): A wrapped grid graph with N vertices can be embedded in an unwrapped grid graph with O(N) vertices and

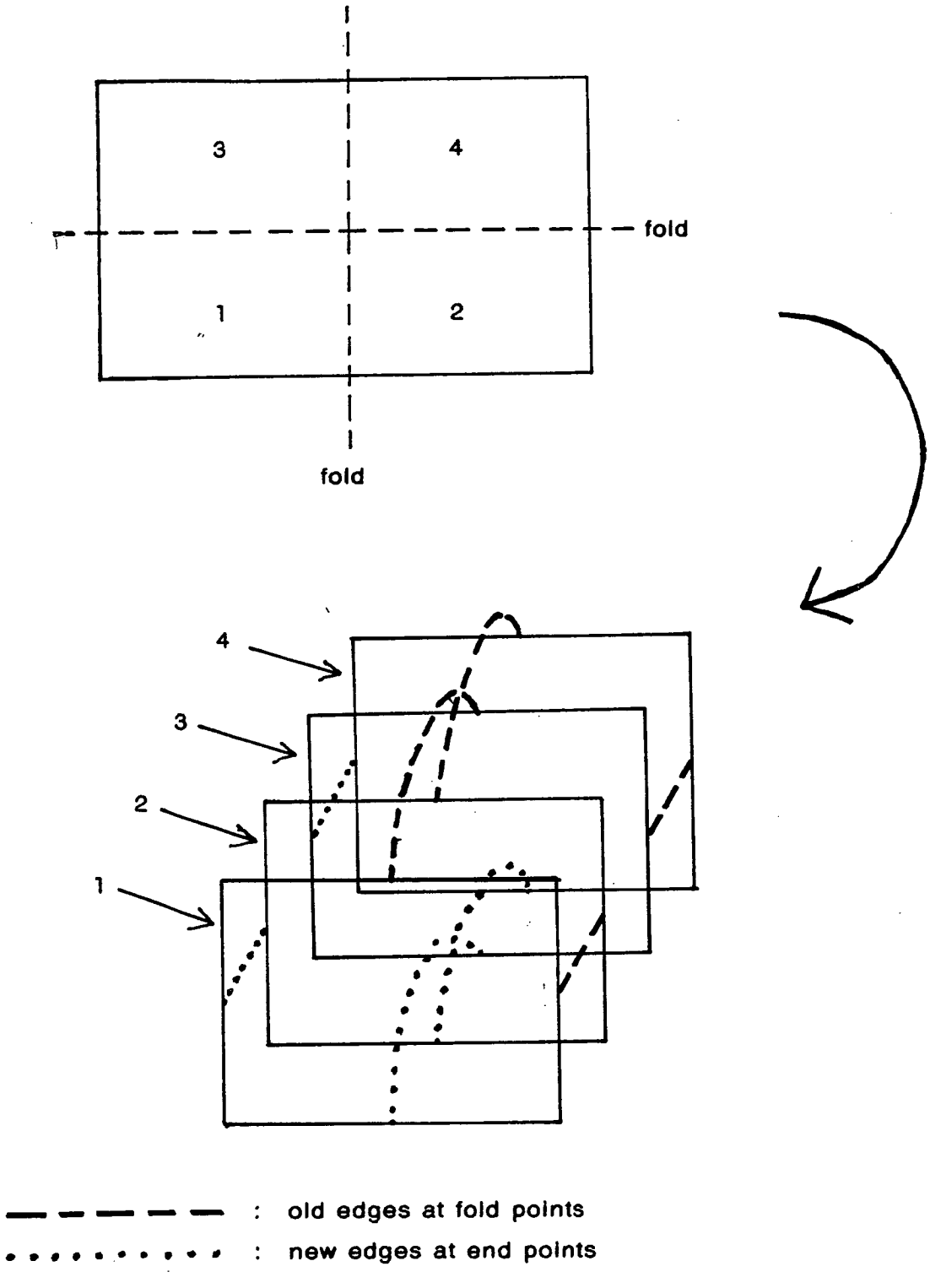


Figure 5-1: Folding a grid layout to give a wrapped grid layout

each embedded edge represented by a path with (short) constant length.

Proof: An embedding can be expressed by the following mapping (assuming, for simplicity, that the grid is of size 2n x 2n)

Vertex (i,j) mapped to :

(1) $(4i, 4j)$ for $0 \leq i,j \leq n-1$

(2) $(4(n-i-1)+1, 4j+1)$ for $n \leq i \leq 2n-1$, $0 \leq j \leq n-1$

(3) $(4i+2, 4(n-j-1)+2)$ for $0 \leq i \leq n-1$, $n \leq j \leq 2n-1$

(4) $(4(n-i-1)+3, 4(n-j-1)+3)$ for $n \leq i,j \leq 2n-1$

This increases the length of each side of the layout by a factor of two, and so the total area is quadrupled. All edges internal to any of the four blocks (1), (2), (3), or (4) of vertices are mapped to the obvious horizontal and vertical paths of length four. The other edges (i.e. those between blocks at either folds or ends) may be added using paths of length four and eight. All of these edges are formed by paths between vertices in the rows and columns of four grid edges at the extremes of the layout. To include the paths, it is necessary to add one new track at the left and right sides of the layout and two new tracks at the top and bottom of the layout. Then eight sets of paths can be defined corresponding to the two types of edge embedded at each of the sides of the layout. These are :

Left side, wrap edges between (1) and (2).
$[(0,j), (2n-1,j)]$, for $0 \leq j \leq n-1$, is embedded as
$(0,4j) \rightarrow (-1,4j) \rightarrow (-1,4j+1)$
$\rightarrow (0,4j+1) \rightarrow (1,4j+1)$

Left side, wrap edges between (3) and (4).
$[(0,j), (2n-1,j)]$, for $n \leq j \leq 2n-1$, is embedded as
$(2,4j+2) \rightarrow (1,4j+2) \rightarrow (0,4j+2)$
$\rightarrow (-1,4j+2) \rightarrow (-1,4j+3)$
$\rightarrow (0,4j+3) \rightarrow (1,4j+3)$
$\rightarrow (2,4j+3) \rightarrow (3,4j+3)$

Right side, fold edges between (1) and (2).
$[(n-1,j),(n,j)]$, for $0 \leq j \leq n-1$, is embedded as
$(4(n-1),4j) \rightarrow (4(n-1)+1,4j) \rightarrow (4(n-1)+2,4j)$
$\rightarrow (4(n-1)+3,4j) \rightarrow (4n,4j)$
$\rightarrow (4n,4j+1) \rightarrow (4(n-1)+3,4j+1)$
$\rightarrow (4(n-1)+2,4j+1) \rightarrow (4(n-1)+1,4j+1)$

Right side, fold edges between (3) and (4).
$[(n-1,j),(n,j)]$, for $n \leq j \leq 2n-1$, is embedded as
$(4(n-1)+2,4j+2) \rightarrow (4(n-1)+3,4j+2) \rightarrow (4n,4j+2)$
$\rightarrow (4n,4j+3) \rightarrow (4(n-1)+3,4j+3)$

Bottom side, wrap edges between (1) and (3).
$[(i,0),(i,2n-1)]$, for $0 \leq i \leq n-1$, is embedded as
$(4i,0) \rightarrow (4i,-1) \rightarrow (4i,-2)$
$\rightarrow (4i+1,-2) \rightarrow (4i+2,-2)$
$\rightarrow (4i+2,-1) \rightarrow (4i+2,0)$
$\rightarrow (4i+2,1) \rightarrow (4i+2,2)$

Bottom side, wrap edges between (2) and (4).
$[(i,0),(i,2n-1)]$, for $n \leq i \leq n-1$, is embedded as
$(4i+1,1) \rightarrow (4i+1,0) \rightarrow (4i+1,-1)$
$\rightarrow (4i+2,-1) \rightarrow (4i+3,-1)$
$\rightarrow (4i+3,0) \rightarrow (4i+3,1)$
$\rightarrow (4i+3,2) \rightarrow (4i+3,3)$

Top side, fold edges between (1) and (3).
$[(i,n-1),(i,n)]$, for $0 \leq i \leq n-1$, is embeded as
$(4i,4(n-1)) \rightarrow (4i,4(n-1)+1) \rightarrow (4i,4(n-1)+2)$
$\rightarrow (4i,4(n-1)+3) \rightarrow (4i,4n)$
$\rightarrow (4i+1,4n) \rightarrow (4i+2,4n)$
$\rightarrow (4i+2,4(n-1)+3) \rightarrow (4i+2,4(n-1)+2)$

Top side, fold edges between (2) and (4).

$[(i, n-1), (i, n)]$, for $n \leq i \leq 2n-1$, is embedded as

$(4i+1, 4(n-1)+1) \rightarrow (4i+1, 4(n-1)+2) \rightarrow (4i+1, 4(n-1)+3)$
$\rightarrow (4i+1, 4n) \rightarrow (4i+1, 4n+1)$
$\rightarrow (4i+2, 4n+1) \rightarrow (4i+3, 4n+1)$
$\rightarrow (4i+3, 4n) \rightarrow (4i+3, 4(n-1)+3)$

It may be verified that these paths are disjoint in order to confirm that this is a valid embedding.

A similar folding idea has been used independently by Culik and Pachl [14]. However, they restrict the folding process to one dimension, and assume that a two-layer embedding is used. This simplifies the "embedding" to the point where it is trivial.

5.2.1 Layouts for the n-dimensional cube

In view of the high connectivity of the cube, it is not surprising that it has a high minimum bisection width, as indicated by the following:

Theorem (5.6): The minimum bisection width of a cube with N vertices is $\Theta(N)$.

Proof: The minimum bisection width is at most N, since the cube can be bisected by removing the set of N edges which traverse one particular dimension.

Now consider the set of all paths between pairs of vertices in the cube which consist of a sequence of edges which traverses the differing dimensions in strictly increasing order. If the set of vertices is partitioned into two halves, there are clearly $(\frac{N}{2})^2$ such paths connecting pairs of vertices in different halves. Let $e = (u, v)$ be an edge which traverses dimension i, for some $0 \leq i \leq \lg N - 1$. Then the maximum number of paths including e can be obtained. Any path reaching it must start at a vertex which has dimensions $i, i+1, \ldots, \lg N - 1$ in common with u, and any path reaching v must finish at a vertex which has dimensions $0, 1, \ldots, i$

in common with v. Thus, there are at most

$$2^{i} \cdot 2^{\lg N - (i+1)} = \frac{N}{2}$$

paths including e.

Hence, removing any edge can break at most $\frac{N}{2}$ paths, and so at least $\frac{N}{2}$ edges must be removed to break all paths between different halves of the partition. That is, the minimum bisection width is at least $\frac{N}{2}$.

Corollary (5.7): Any grid embedding of a cube with N vertices requires $\Omega(N^2)$ area.

An optimal layout for the cube will now be constructed.

Theorem (5.8): A cube with N vertices can be laid out in $O(N^2)$ area.

Proof: The basic idea is that the vertices are arranged as a $\sqrt{N}$ by $\sqrt{N}$ square in the grid. Edges which traverse even dimensions follow horizontal paths, and edges which traverse odd dimensions follow vertical paths. The construction is described inductively by considering a layout for the N vertex cube constructed by combining two $\frac{N}{2}$ vertex cube layouts. New edges are added which run in the horizontal or vertical direction at alternate induction steps and which leave from, and arrive at, their endpoints in a north, west, south, east direction repeating every four induction steps.

As an illustration of this, consider the construction of a 16 vertex cube starting from a one vertex cube which is shown in the figure. New edges at each stage are drawn dotted.

Without loss of generality, suppose that $N = 2^{4k+1}$ for some $k \geq 0$. Then new edges are being added in a horizontal direction, to and from the north of their endpoints. By symmetry, similar arguments apply to the other three cases.

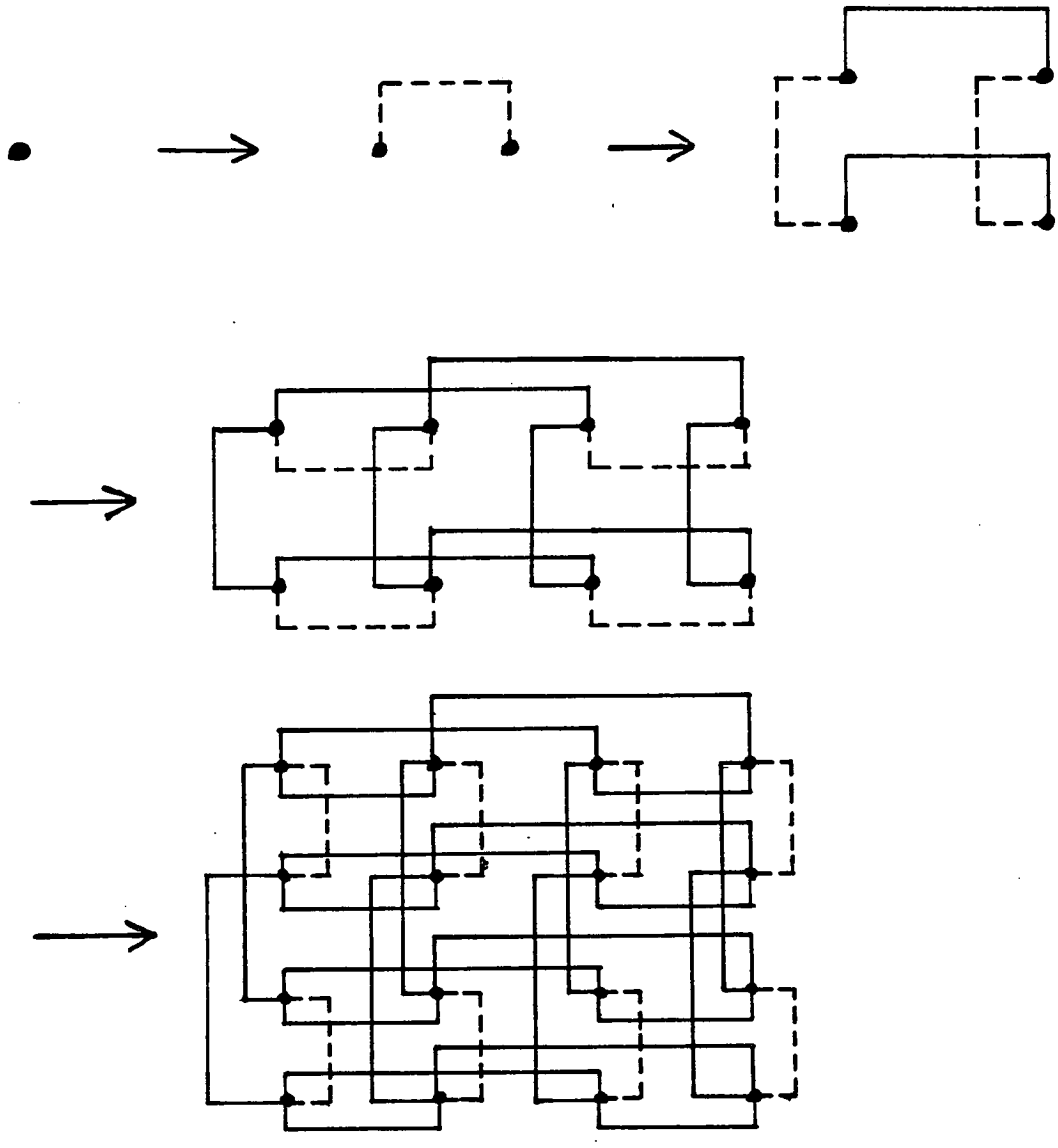


Figure 5-2: Layout of 16 vertex cube

Take two identical 2^{2k} by 2^{2k} layouts of $\frac{N}{2}$ vertex cubes. By doubling the horizontal spacing of each column of 2^{2k} vertices, form a 2^{2k+1} by 2^{2k} layout in which corresponding columns in the sublayouts are horizontally adjacent. In doing this, horizontal paths in the sublayouts will have become overlapped, although vertical paths remain disjoint. These horizontal paths correspond to cube edges traversing dimensions $0, 2, 4, \ldots, 4k-2$. To remove the overlapping, the horizontal paths in the sublayouts

must be double spaced so that they occupy alternating disjoint horizontal tracks. Finally, new edges traversing dimension 4k may be included using one extra horizontal track which contains paths connecting horizontally adjacent vertices.

A last consideration occurs only when N is of the form 2^{4k+1}. The size of each vertex increases by one unit in each direction. Since there are 2^{2k} rows of vertices and 2^{2k+1} columns of vertices, the extra area requirement is $2^{2k}2^{2k+1}(2k+1) = O(N\log N)$

If A(N) is the area required for the layout, then the above construction gives

$$A(N) \leq 2(2(A(\frac{N}{2}))) + O(N\log N)$$

and solving this recurrence gives

$$A(N) = O(N^2)$$

as desired.

Therefore, it has been established that a grid embedding of an N vertex cube has area $\Theta(N^2)$.

5.2.2 Layouts for the d-shuffle

A similar technique to that used for the cube will be used to lower bound the bisection width of the d-shuffle. Not surprisingly, the bisection width of the d-shuffle is smaller.

Theorem (5.9): The minimum bisection width of a d-shuffle with N vertices is $\Omega(\frac{dN}{\log_d N})$.

Proof: Let $n = \log_d N$ and consider the set of paths of length n which connect any pair of vertices $\alpha_{n-1}\ldots\alpha_0$ and $\beta_{n-1}\ldots\beta_0$ with the ith edge, $1 \leq i \leq n$, being

$$(\alpha_{n-i}\ldots\alpha_0\beta_{n-1}\ldots\beta_{n-i+1},\ \alpha_{n-i-1}\ldots\alpha_0\beta_{n-1}\ldots\beta_{n-i})$$

If the set of vertices is partitioned into two parts of size $\lfloor d^n/2 \rfloor$ and $\lceil d^n/2 \rceil$ then there are $\lfloor d^n/2 \rfloor . \lceil d^n/2 \rceil$ such paths between vertices in different halves.

Now, if $e = (\phi_n \ldots \phi_1, \phi_{n-1} \ldots \phi_0)$ is any edge in the graph, then e can be the ith step in the path between any pair of vertices $\chi_{n-1} \ldots \chi_{n-i+1} \phi_n \ldots \phi_i$ and $\phi_{i-1} \ldots \phi_0 \psi_{n-i-1} \ldots \psi_0$ for $1 \leq i \leq n$ and any χ_j, ψ_k. Therefore, if e is removed, the number of paths which are broken is at most $nd^{i-1}d^{n-i} = nd^{n-1}$.

Hence, to break all paths between different halves of the partition, at least

$$\frac{1}{nd^{n-1}} \cdot \lfloor \frac{d^n}{2} \rfloor . \lceil \frac{d^n}{2} \rceil = \Omega(\frac{d.d^n}{n})$$

edges must be removed.

The minimum bisection width is, in fact, $\Theta(\frac{dN}{\log_d N})$, a result which follows from the area-optimal embedding to be demonstrated.

Corollary (5.10): Any grid embedding of a d-shuffle with N vertices requires $\Omega((\frac{dN}{\log_d N})^2)$ area.

Now, a d-shuffle layout which achieves this area bound will be developed in two stages. First, a layout for the 2-shuffle graph is obtained, and then it is utilised as a basis for layouts of arbitrary degree d-shuffles.

Theorem (5.11): A 2-shuffle graph with N vertices can be laid out in $O((\frac{N}{\lg N}))$ area.

Proof: As has been remarked earlier, there is a close relationship between the 2-shuffle graph and the shuffle-exchange graph. This enables the exploitation of a non-trivial area-optimal shuffle-exchange layout developed by Kleitmann et al [28]. It is necessary to map the shuffle edges of the 2-shuffle on to the shuffle and exchange edges of the shuffle-exchange. Consider a vertex $\alpha\beta$, where $\alpha \in \{0,1\}$ and $\beta \in \{0,1\}^{n-1}$. Then edges leave $\alpha\beta$ to $\beta 0$ and $\beta 1$ and these can be simulated by paths $\alpha\beta \rightarrow \beta\alpha$

and $\alpha\beta \rightarrow \beta\alpha \rightarrow \beta\bar{\alpha}$. Thus, each shuffle edge and each exchange edge in the shuffle-exchange is used in exactly two paths.

Therefore, to obtain a 2-shuffle layout, the shuffle-exchange layout is doubled in each direction so that two copies of each embedded edge are adjacent to one another. Then the layout must be adjusted at the endpoints of the edges since there are now six edges with endpoints in the region of each vertex. As an illustration, consider a vertex $0\alpha0$ as shown in the figure.

It can be seen how these six edges must be incorporated into the new layout. The two original shuffle edges remain intact. One of the new shuffle edges is connected to the previously unused "fourth side" of the vertex. The other is connected to the original exchange edge. Finally, the new exchange edge is connected to the vertex at the point vacated by the original exchange edge. In order to achieve this rerouting of edges, it is sufficient to expand the area in the region of each embedded vertex (and hence the total area) by a small constant factor.

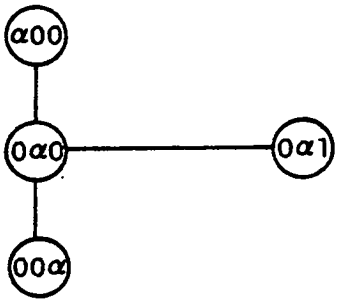
Therefore, given that the shuffle-exchange layout requires

$$O\left(\left(\frac{N}{\lg N}\right)^2\right)$$

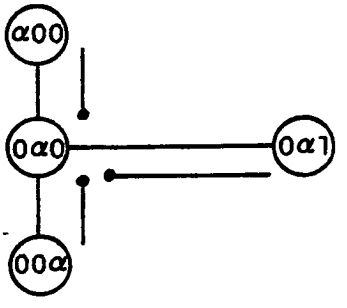
area, the same area is sufficient for the 2-shuffle. The previous result indicates that this area is also necessary.

Theorem (5.12): A d-shuffle graph with N vertices can be laid out in $O\left(\left(\frac{dN}{\log_d N}\right)\right)$ area.

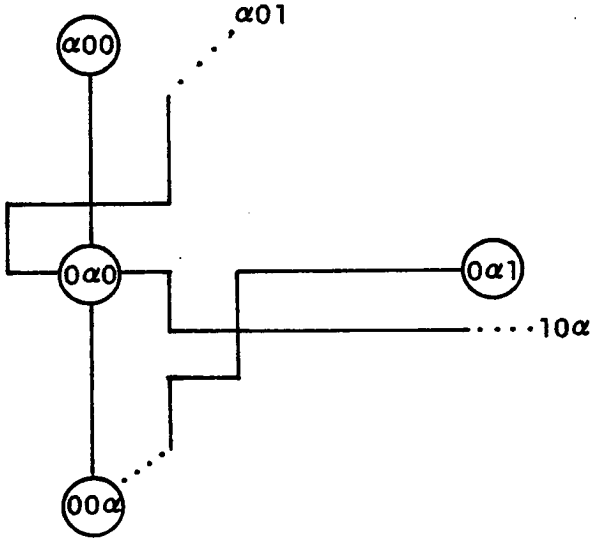
Proof: The 2-shuffle layout will be used as a basis for a general d-shuffle one. The basic idea is to view the d-ary string encoding each d-shuffle vertex in a "binary coded d-ary" representation. That is, if $N = d^n$, each vertex is represented by a string consisting of n blocks of $\lceil \lg d \rceil$ bits. Then, starting from a layout of a $2^{n\lceil \lg d \rceil}$ vertex d-shuffle, adjustments are made in



(i) shuffle-exchange



(ii) edges doubled



(iii) 0α0 adjusted

Figure 5-3: Layout of 2-shuffle vertex 0α0

order to achieve the desired goal.

First, it should be noted that the area required by each embedded vertex of the degree 2d undirected graph is $O(d^2)$, and so the original layout must be expanded appropriately. There are then $4\lceil d/2 \rceil$ interconnection points at each vertex instead of four. To embed the edges of the d-shuffle, it is necessary to realise each as a path of length precisely $\lceil \lg d \rceil$ in the 2-shuffle. This path effects a shift by one d-ary digit using $\lceil \lg d \rceil$ shifts by one binary digit. As an example, consider the edge between 615_8 and 152_8 in an 8-shuffle. The binary representation of these vertices are 110 001 101 and 001 101 010 respectively. Then the edge between 615_8 and 152_8 is embedded as the path

110 001 101 → 100 011 010
→ 000 110 101
→ 001 101 010

In order to embed all edges, it is clear that 2-shuffle edges must be replicated since they will form part of many paths. In general, an edge may appear at stage 1, 2, ..., or $\lceil \lg d \rceil$ of a path. If it appears at stage i then the path may have started at any one of 2^{i-1} vertices and also the path may finish at any one of $2^{\lceil \lg d \rceil - i}$ vertices. Thus, the edge may occur in at most

$$\lceil \lg d \rceil 2^{\lceil \lg d \rceil - 1} = O(d \lg d)$$

paths. To replicate each edge this number of times, the original layout must be expanded in area by an $O((d \lg d)^2)$ factor. Note that this area increase is simultaneous with, not in addition to, that required to accomodate the degree 2d vertices. This will be made clear by seeing how the replicated edges are incorporated.

A replicated edge need only have contact with one of its end vertices if it is forming the first stage of a path in the 2-shuffle, and with the other if it is forming the $\lceil \lg d \rceil$th stage of a path. The above remarks indicate that at most $2^{\lceil \lg d \rceil - 1}$ replicas come into

each category. All others may be connected to neighbouring edges in the path at each end. To illustrate what must be done, figure 5-4 shows vertex 0α0 in the 2-shuffle (if d is not a power of two, some vertices may be surplus to the requirements of the embedding without affecting the upper bound in area. Assume that the selected vertex is relevant). The orientation of edges is chosen for diagrammatic simplicity rather than to imply anything about the 2-shuffle layout.

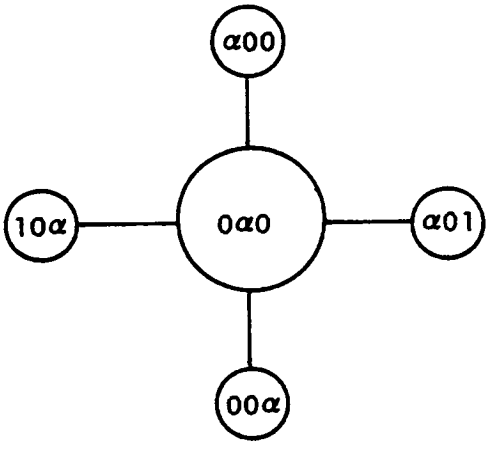
Of the (at most) $\lceil \lg d \rceil 2^{\lceil \lg d \rceil - 1}$ edges incident at each side of the vertex, at most $2^{\lceil \lg d \rceil - 1}$ are connected to the vertex. The remainder are divided between connections to two of the other three incident edges, with at most $(\lceil \lg d \rceil - 1) 2^{\lceil \lg d \rceil - 2}$ going to either. In the example, this means connections as shown in figure 5-5.

In the diagram, the ordering of incident replicated edges is particularly amenable to the required routing. In general, the routing can still be performed at the expense of a constant increase in area around the vertex. If the ordering of incident replicas is random, $O((2^{\lceil \lg d \rceil - 1})^2) = O(d^2)$ extra area is needed to align path start and finish edges correctly (if d is not a power of two, all replicas may not have the same orientation at the vertex). Finally, a doubling of the number of tracks in the horizontal and vertical direction is sufficient to connect the remaining edges together.

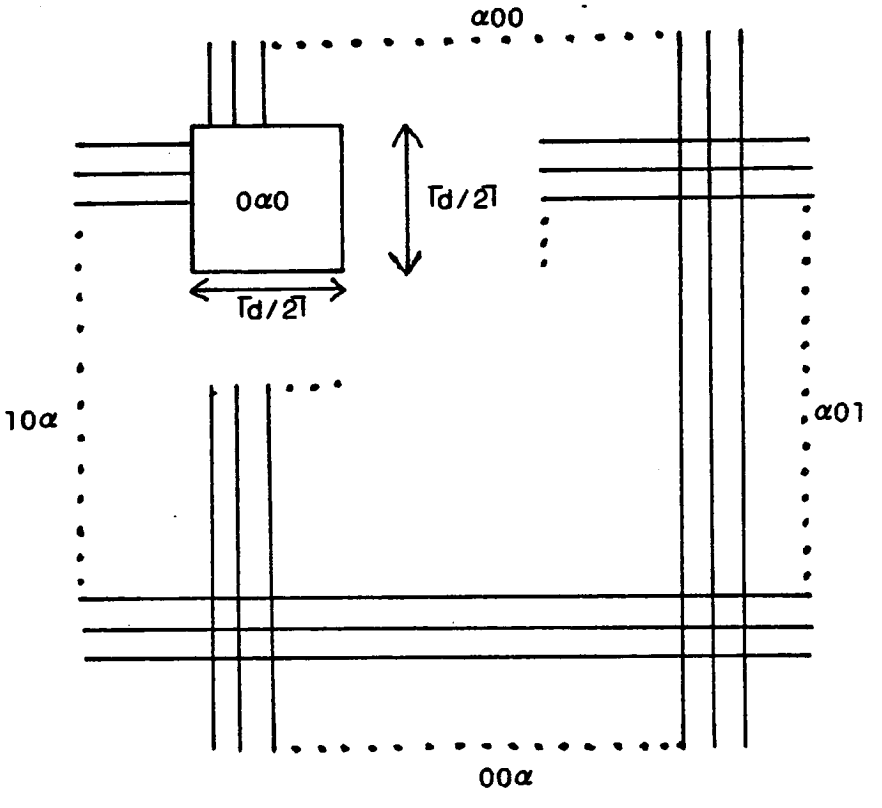
Thus, the area expansion of the 2-shuffle layout is $O((d \lg d)^2)$. Since the original area is $O((\frac{N}{\lg N})^2)$, the d-shuffle layout occupies

$$O((\frac{d . \lg d . N}{\lg N})^2) = O((\frac{dN}{\log_d N})^2) \text{ area}$$

which matches the lower bound.



Original 2-shuffle at vertex 0α0



2-shuffle at 0α0 after expansion of vertices and edges

Figure 5-4: Role of 2-shuffle vertex 0α0 in d-shuffle layout

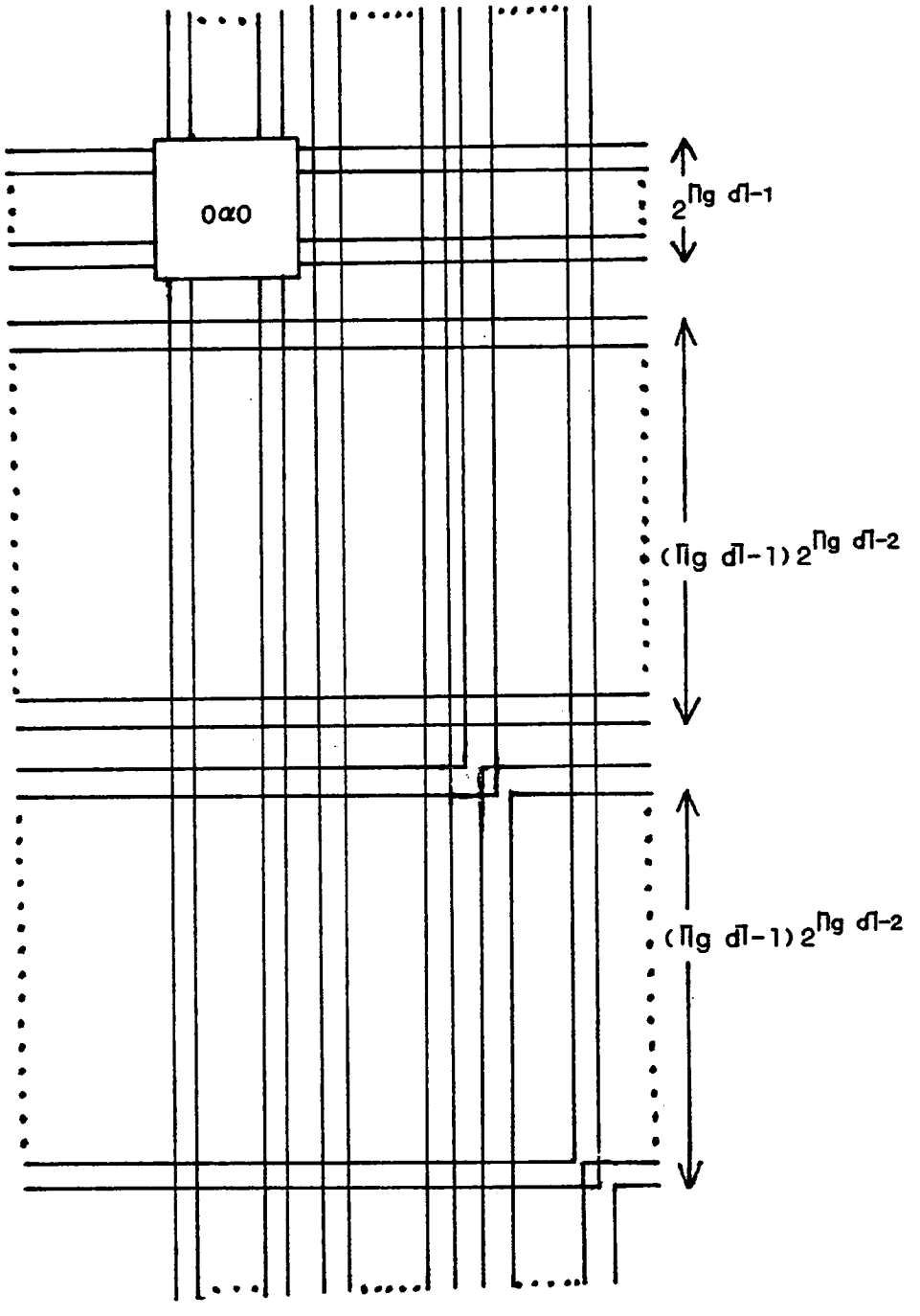


Figure 5–5: Wiring at vertex 0α0 in d-shuffle layout

5.2.3 Maximum edge lengths in layouts

As already remarked, while the embeddings demonstrated have optimal area, they require long paths in the grid. The necessity for long paths can also be shown by a simple argument.

Theorem (5.13): Any graph with diameter δ which has a grid embedding with area A must have an embedded edge with path length $\Omega(\frac{\sqrt{A}}{\delta})$ in the grid.

Proof: Since the grid has area A, it is possible to choose two points in the grid such that any path connecting the two points must have length at least $\sqrt{A}$. Further, these points may be chosen so that they lie on embedded edges of the graph. Therefore, because the graph has diameter δ, at most δ edges can separate the two points in the embedding. Thus, some edge must have length at least $\frac{\sqrt{A}}{\delta}$.

Corollary (5.14): Any layout of an N vertex cube has an edge of length $\Omega(\frac{N}{\lg N})$.

Corollary (5.15): Any layout of an N vertex d-shuffle has an edge of length $\Omega(\frac{dN}{(\log_d N)^2})$

Similar minimum maximum edge length results have also been shown for other graphs recently by Leighton [31] and Paterson, Ruzzo, and Snyder [38], for example. The layouts obtained here are not optimal in the sense of maximum edge length. Both are worse by a logarithmic factor. In the case of the cube, it is easily seen from the construction that the longest edges are those in the first and second dimensions and, since they run half the length of a side of the grid, they have length O(N).

The details of the d-shuffle are not so obvious, primarily because no information has been given about the underlying shuffle-exchange layout. The only two details of this layout which are needed are

(i) it has $O(\frac{N}{\lg N})$ side length

and

(ii) No edge runs in more than a constant number of tracks

From these, an $O(\frac{N}{\lg N})$ maximum edge length follows. The adjustments made to the shuffle-exchange edges to obtain 2-shuffle edges approximately quadruples their length (doubling of layout, then combination of pairs of edges) in the worst case. This means that the $O(\frac{N}{\lg N})$ bound still applies. When the 2-shuffle layout is expanded up to form the d-shuffle layout, edge lengths are increased firstly by a $d\lceil \lg d \rceil$ factor and secondly by a $\lceil \lg d \rceil$ factor when $\lceil \lg d \rceil$ edges are concatenated. This gives a

$$\frac{d . \lg d . N}{\log_d N}$$

bound on maximum edge length, which is $O(\lg N)$ away from the lower bound.

The nature of exact bounds for these two graphs is unresolved. The naivity of the lower bound proof suggests that a tighter result may be possible. However, techniques for proving lower bounds on problems related to grid embeddings are currently not well developed. It seems likely that the cube layout demonstrated is optimal. On the other hand, the d-shuffle layout involves certain inefficiencies, notably each embedded edge following a 2-shuffle path of length $\lg d$, and the upper bound may be reducible to $O(\frac{dN}{\log_d N})$. The reader should not seize upon the similarity of these bounds to those on minimum bisection width - consider at one extreme a tree and, at the other, the grid itself.

Regardless of whether the given lower bounds are tight, they do indicate the infeasibility of implementing the routing graphs in a two dimensional rectangular technology if transmission time is proportional to edge length. It has been seen empirically that $\Theta(\delta)$ randomised routing time can be achieved on any of the graphs with diameter δ considered. However, the effect of maximum edge lengths means that that

$$\Omega(\delta . \frac{\sqrt{A}}{\delta}) = \Omega(\sqrt{A})$$

time is required in a linear time model for routing on a graph with layout area A. This changes the perspective on what is a "good" routing graph.

Naturally, the grid (which has an ideal grid embedding) is best. The d-shuffle is superior to the cube, unless the maximum edge length bounds hold in opposite directions. This matches intuition since the cube has many more edges, which do not buy it proportional extra power in routing. The obvious conclusion from these results is that grid-style technologies are not appropriate for general purpose parallel computation, and should be reserved for more special purpose applications.

5.3 Embedding grids in routing graphs

Assuming that technologies are available in which desirable routing graphs can be constructed with good timing characteristics, it is interesting to ask whether it is possible to efficiently simulate the many algorithms for grid architectures without employing general-purpose routing at each step. In other words, can a grid graph be efficiently embedded in the routing graph?

This question can be affirmatively answered for the cube.

Theorem (5.16): An N x N grid can be embedded in a cube with $2^{2\lceil \lg N \rceil}$ vertices, which is optimal.

Proof: The proof is inductive, with the base case N = 1 being trivial. Suppose the result is true for every grid with side length at most 2^i for some $i \geq 0$.

Consider a cube with $2^{2(i+1)}$ vertices. Then it can be divided into four subcubes, each with 2^{2i} vertices, which contain embedded $2^i \times 2^i$ grids. If the subcubes are combined in pairs giving two subcubes with 2^{2i+1} vertices then edges now connect corresponding vertices along one side of the embedded $2^i \times 2^i$ grids, meaning that a $2^{i+1} \times 2^i$ (or $2^i \times 2^{i+1}$) grid is embedded). Repeating this process again means that a $2^{i+1} \times 2^{i+1}$ grid is embedded in the cube with $2^{2(i+1)}$ vertices. Clearly, any grid with side length at most 2^{i+1} must therefore be embedded also. The embedding also has the pleasant property that edges are mapped one-one into edges.

Unfortunately, but not unpredictably, the grid cannot be simulated so effectively on the d-shuffle graph. The next theorem shows that the minimum number of d-shuffle vertices required is more than a constant factor times the number of grid vertices being embedded. While such increases in graph size were regarded as acceptable when graphs were embedded in grids (which had a geometrical motivation), in this context an increase in the number of processors is necessary and this is regarded as expensive.

Theorem (5.17): Any embedding of an N x N grid in a directed d-shuffle graph requires

$$\Omega(N^2 . \frac{\log_d N}{d^2 \log \log_d N})$$

vertices.

Proof: Consider the following pairs of paths in the grid

For $0 \leq i,j \leq \frac{N}{2}-1$,

$(2i,2j) \rightarrow (2i+1,2j) \rightarrow (2i+1,2j+1)$

and

$(2i,2j) \rightarrow (2i,2i+1) \rightarrow (2i+1,2j+1)$

Fix i and j and suppose that vertices α and β in the d-shuffle graph correspond to (2i,2j) and (2i+1,2j+1) respectively. Now, there must be two paths between α and β corresponding to the above grid paths. If δ is the diameter of the d-shuffle then, if both paths are to be as short as possible they will have length less than δ and so α and β must have the form

$$\alpha = \rho\sigma\tau \text{ and } \beta = \sigma\tau x\phi = \tau y\psi \text{ where } x \neq y$$

Thus, the total path length is at least $\delta - |\tau|$.

Now, if $k = \delta - |\tau|$, there are at most kd^k strings β which have their initial substring τ repeated internally. To accomodate all such pairs of paths of the grid, $\frac{N^2}{2}$ strings β are required. The smallest total embedded path length is therefore at least

$$\sum_{k=1}^{m-1} k.d^k$$

where m is defined by

$$\sum_{k=1}^{m-1} kd^k < \frac{N^2}{4} \quad \text{but} \quad \sum_{k=1}^{m} kd^k \geq \frac{N^2}{4}$$

Hence, summing the series,

$$md^m = \Omega(N^2)$$

and so

$$m = \Omega\left(\frac{\log_d N}{\log \log_d N}\right)$$

The total number of d-shuffle edges required is

$$\Omega((m-1)^2 d^{m-1})$$

$$= \Omega\left(N^2 \frac{\log_d N}{d \log \log_d N}\right)$$

The result follows from the fact that the d-shuffle has degree d.

□

Acknowledgements

Two people have supervised this work, and both are due many thanks. Firstly, Les Valiant, who passed on his enthusiasm for Complexity and supplied many suggestions and much advice. Secondly, Mark Jerrum, who had the difficult task of taking over midway but responded to the challenge magnificently.

Many others have helped in various ways. Worthy of note are Carl Sturtivant, for many interesting discussions; George Ross, for dealing with numerous queries about the text formatter used to print this thesis; and Peter Schofield, for some conversations at tricky moments.

Bibliography

[1] Aho, A. V., Hopcroft, J. E. and Ullman, J. D. "The design and analysis of computer algorithms". Addison Wesley, 1974.

[2] Aleliunas, R. Randomised parallel communication. Proc. ACM Symposium on Principles of Distributed Computing, Ottawa, Canada, 1982, pp. 60-72.

[3] Babai, L., Grigoryev, D. Yu. and Mount, D. M. Isomorphism of graphs with bounded eigenvalue multiplicity. Proc. 14th Annual ACM Symposium on Theory of Computing, San Francisco, California, 1982, pp. 310-324.

[4] Barnes, G. H. et al. The ILIAC IV computer. IEEE Transactions on Computers 17 (1968), pp. 746-757.

[5] Batcher, K. Sorting networks and their applications. Proc. AFIPS Spring Joint Computing Conference, 1968, pp. 307-314.

[6] Benes, V. E. "Mathematical theory of connecting networks and telephone traffic". Academic Press, New York, 1965.

[7] Bilardi, G., Pracchi, M. and Preparata, F. P. A critique and an appraisal of VLSI models of computation. Proc. CMU Conference on VLSI Systems and Computations, Pittsburgh, Pennsylvania, 1981, pp. 81-88.

[8] Bollobas, B. London Math. Soc. Mongraphs. Volume 11: "Extremal graph theory". Academic Press, London, 1978.

[9] Borodin, A. and Hopcroft, J. E. Routing, merging and sorting on parallel models of computation. Proc. 14th Annual ACM Symposium on Theory of Computing, San Francisco, California, 1982, pp. 338-344.

[10] Chazelle, B. and Monier, L. A model of computation for VLSI with related complexity results. Proc. 13th Annual ACM Symposium on Theory of Computing, Milwaukee, Wisconsin, 1981, pp. 318-325.

[11] Chernoff, H. A measure of asymptotic tests of efficiency for tests of a hypothesis based on the sum of observations. Annals of

Mathematical Statistics 23 (1952), pp. 493-507.

[12] Cook, S. A. Towards a complexity theory of synchronous parallel computation. L'Enseignement mathematique 27 (1981), pp. 99-124.

[13] Cook, S. A. and Dwork, S. Bounds on the time for parallel RAMs to compute simple functions. Proc. 14th Annual ACM Symposium on Theory of Computing, San Francisco, California, 1982, pp. 338-343.

[14] Culik II, K. and Pachl, J. Folding and unrolling systolic arrays. Proc. ACM Symposium on Principles of Distributed Computing, Ottawa, Canada, 1982, pp. 254-261.

[15] David, F. N. and Barton, F. N. "Combinatorial chance". Griffin, 1962.

[16] Even, S. "Graph algorithms". Pitman, 1979.

[17] Feller, W. "An introduction to probability theory and its applications, Volume 1". Wiley, 1967 (3rd edition).

[18] Fortune, S. and Wyllie, J. Parallelism in random access machines. Proc. 10th Annual ACM Symposium on Theory of Computing, San Diego, California, 1978, pp. 114-118.

[19] Galil, Z. and Paul, W. J. An efficient general purpose parallel computer. Journal of the ACM 30 (1983), pp. 360-382.

[20] Goldschlager, L. M. A unified approach to models of synchronous parallel machines. Proc. 10th Annual ACM Symposium on Theory of Computing, San Diego, California, 1978, pp. 89-94.

[21] Good, I. J. A Monte Carlo method that will be impracticable before the year 2001. Journal of Statistical Computation and Simulation 14 (1981), pp. 67-69.

[22] Gottlieb et al. The NYU ultracomputer - designing a MIMD shared memory parallel machine. Ultracomputer Note #40, Courant Institute, New York University, 1982.

[23] Hoeffding, W. On the distribution of the number of successes in independent trials. Annals of Mathematical Statistics 27 (1956), pp. 713-721.

[24] Hopcroft, J. E. and Ullman, J. D. "Introduction to automata theory, languages, and computation". Addison-Wesley, 1979.

[25] Jerrum, M. R. and Skyum, S. Families of fixed degree graphs for processor interconnection. Internal Report CSR-121-82, Department of Computer Science, University of Edinburgh, 1982. To be published in IEEE Transactions on Computers.

[26] Johnson, N. L. and Young, D. H. Some Applications of two approximations to the multinomial distribution. Biometrika 47 (1960), pp. 463-469.

[27] Kleinrock, L. "Queueing systems, Volume 2 : Computer applications". Wiley-Interscience, New York, 1976.

[28] Kleitman, D., Leighton, F. T., Lepley, M. and Miller, G. L. New layouts for the shuffle-exchange graph. Proc. 13th Annual ACM Symposium on Theory of Computing, Milwaukee, Wisconsin, 1981, pp. 278-292.

[29] Knuth, D. E. "The art of computer programming, Volume 2 : Seminumerical algorithms". Addison-Wesley, 1969.

[30] Kung, H. T. Let's design algorithms for VLSI systems. Technical Report CMU-CS-79-151, Department of Computer Science, Carnegie-Mellon University, 1979.

[31] Leighton, F. T. New lower bound techniques for VLSI. Proc. 22nd Annual IEEE Symposium on Foundations of Computer Science, Nashville, Tennessee, 1981, pp. 1-12.

[32] Leland, W. E. "Density and reliability of interconnection topologies for multicomputers". Ph.D Thesis, University of Wisconsin - Madison, 1982.

[33] Lev, G., Pippenger, N. and Valiant, L. G. A fast parallel algorithm for routing in permutation networks. IEEE Transactions on Computers 30 (1981), pp. 93-100.

[34] Mead, C. and Conway, L. "Introduction to VLSI systems". Addison Wesley, 1980.

[35] Meyer auf der Heide, F. Infinite cube-connected cycles. Information Processing Letters 16 (1983), pp. 1-2.

[36] Monier, L. Evaluation and comparison between two efficient probabilistic primality testing algorithms. Manuscript, 1978.

[37] Nair, K. R. The distribution of the extreme deviate from the sample mean and its studentized form. Biometrika 35 (1948), pp. 118-144.

[38] Paterson, M. S., Ruzzo, W. L. and Snyder, L. Bounds on minimax edge length for complete binary trees. Proc. 13th Annual ACM Symposium on Theory of Computing, Milwaukee, Wisconsin, 1981, pp. 293-299.

[39] Paul, W. J. Kolmogorov complexity and lower bounds. Proc. 2nd International Conference on Fundamentals of Computation Theory, Berlin, 1979, pp. 325-334.

[40] Preparata, F. P. and Vuillemin, J. The cube-connected cycles : a versatile network for parallel computation. Communications of the ACM 24 (1981), pp. 300-309.

[41] Rabin, M. O. Probabilistic algorithms. In "Algorithms and complexity", J. F. Traub (Ed.), Academic Press, New York, 1976.

[42] Reif, J. H. and Valiant, L. G. A logarithmic time sort for linear size networks. Technical Report TR-13-82, Aiken Computation Laboratory, Harvard University, 1982. (Also in Proc. 15th Annual ACM Symposium on Theory of Computing, Boston, Massachusetts, 1983).

[43] Reisch, S. and Schnitger, G. Three applications of Kolmogorov-complexity. Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science, Chicago, Illinois, 1982, pp. 45-52.

[44] Schwartz, J. T. Ultracomputers. ACM Transactions on Programming Languages and Systems 2 (1980), pp. 484-521.

[45] Shiloach, Y. and Vishkin, U. Finding the maximum, merging, and sorting in a parallel computation model. Journal of Algorithms 2 (1981), pp. 88-102.

[46] Siegel, H. J. Interconnection networks for SIMD machines. Computer 12 (June 1979), pp. 57-65.

[47] Solovay, R. and Strassen, V. A fast Monte Carlo test for primality. SIAM Journal on Computing 6 (1977), pp. 84-85.

[48] Stone, H. S. Parallel processing with the perfect shuffle. IEEE Transactions on Computers 20 (1971), pp. 153-161.

[49] Storwick, R. Improved construction techniques for (d,k) graphs. IEEE Transactions on Computers 19 (1970), pp. 1214-1216.

[50] Thompson, C. D. "A complexity theory for VLSI". Ph.D Thesis, Department of Computer Science, Carnegie-Mellon University, 1980.

[51] Upfal, E. Efficient schemes for parallel communication. Proc. ACM Symposium on Principles of Distributed Computing, Ottawa, Canada, 1982, pp. 55-59.

[52] Valiant, L. G. Experiments with a parallel communication scheme. 18th Allerton Conference on Communication, Control, and Computing, University of Illinois, 1980, pp. 802-811.

[53] Valiant, L. G. and Brebner, G. J. Universal schemes for parallel communication. Proc. 13th Annual ACM Symposium on Theory of Computing, Milwaukee, Wisconsin, 1981, pp. 263-277.

[54] Valiant, L. G. A scheme for fast parallel communication. SIAM Journal on Computing 11 (1982), pp. 350-361.

[55] Valiant, L. G. Optimality of a two-phase strategy for routing in interconnection networks. Technical Report TR-15-82, Aiken Computation Laboratory, Harvard University, 1982.