1983

A Simulation Study of Multimicrocomputer Networks

Daniel A. Reed

Report Number:
83-437

A Simulation Study of Multimicrocomputer Networks

Daniel A. Reed

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD-TR-~~485~~ 437

ABSTRACT

Recent developments in integrated circuit technology have suggested a new building block for parallel processing systems: the single chip computer. This building block makes it economically feasible to interconnect large numbers of computers to form a multimicrocomputer network. Because the nodes of such a network do not share any memory, it is crucial that a interconnection network capable of efficiently supporting message passing be found. We present a model of time varying computation based on task precedence graphs that corresponds closely to the behavior of fork/join algorithms such as divide and conquer. Using this model, we investigate the behavior of five interconnection networks under varying workloads with distributed scheduling.

March 8, 1983

Introduction

Two evolutionary trends in integrated circuit fabrication suggest that the single chip computer will become a new building block for designers of parallel processing systems:

- the increasing device densities made possible by very large scale integration (VLSI) and
- the increasing discrepancy between the speed power product of signal paths on a single chip and those between chips.

By placing the memory on the same chip as the processor, memory access time is reduced and processing power is increased. The introduction of the Intel 8051 [INTE81] with processor, memory, and I/O ports all on a single chip is evidence of movement in this direction.

These new building blocks make it cost effective to consider a new parallel processing paradigm based on large networks of interconnected single chip computers. The single VLSI chip comprising each network node would contain a processor with a modicum of locally addressable memory, a communication controller capable of routing internode messages without delaying the processor, and a small number of connections to other nodes.

Among the suggested application areas for these multimicrocomputer networks are partial differential equations solvers [REED83b] and divide and conquer algorithms [ELDE79]. The cooperating tasks of a parallel algorithm for solving one of these problems would execute asynchronously on different nodes and communicate via internode message passing. Because each network node is a single VLSI chip, the number of connections to each node must be small. This limitation, together with the absence of shared memory, makes it crucial

to select an interconnection network capable of efficiently supporting message passing. In this paper we discuss computation paradigms for multimicrocomputer networks and techniques for assessing the performance of network interconnections.

Models of Computation

There are two primary views of parallel computation on a multimicrocomputer network. In the first, all parallel tasks are known *a priori* and are statically mapped onto the network nodes before the computation begins. This approach corresponds closely to the operation of partial differential equations solvers [REED83b] and finite element problem solvers [SMIT82] in which blocks of data points are placed in the nodes, and each node iteratively updates its block of points after sending messages to and receiving messages from other nodes. For this type of computation, queueing theoretic models can be used to estimate the performance of a given multimicrocomputer network executing a particular algorithm [REED83a].

In the alternate view, a parallel computation is defined by a dynamically created task precedence graph. Tasks are created and destroyed as the computation proceeds, and the mapping of tasks onto network nodes is done dynamically. This approach is well suited to parallel divide and conquer algorithms [ELDE79] and mini-max game tree searches [AKL80] where the computation state is time dependent.

Because most queueing theoretic models assume steady state behavior, they are not generally applicable to study of time dependent parallel computations. In particular, models of time dependent computation must account for

- time varying workloads,
- distribution of data to multiple tasks, and
- dynamic mapping of tasks onto network nodes using only partial knowledge of the global network state.

Because we know of no analytic technique capable of accurately representing this behavior, we have pursued simulation studies.

Subsequent sections of this paper present five multimicrocomputer interconnection networks, outline a task precedence model of time dependent computation, and discuss the results of a parametric simulation study of these interconnection networks when supporting time dependent computations.

Interconnection Networks

Because of the computational expense of simulation, we limited our study to the five interconnection networks shown in Figure 1 that earlier analysis suggested were worthy of further investigation:

- the 2-*D* spanning bus hypercube [WITT81],
- 2-*D* toroid [REED83a],
- cube-connected cycles [PREP81],
- 2-ary *N*-cube [BURT81], and
- the complete connection.

We have included the complete connection to determine the performance degradation attributable to incompletely connected networks. Before discussing the results of our simulation study, we must briefly digress to describe the geometric properties of the aforementioned networks.

Spanning Bus Hypercube

The 2-D spanning bus hypercube [WITT81] is a square mesh of width w with each node connected to buses spanning a row or column, whence the name. The maximum internode distance is two, one bus in each of the two dimensions. The interconnection easily generalizes to three or more dimensions, albeit with an increased number of connections to each node.

Toroid

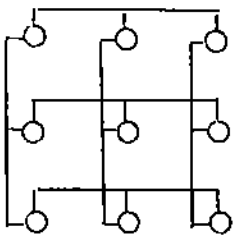
The toroid has the same topology as the spanning bus hypercube, but each bus is replaced by a bidirectional ring connecting the w nodes in each dimension. A message is routed toward its destination by selecting a dimension in which the current node and destination node address differ and moving along the ring spanning that dimension in the shorter of the clockwise or counterclockwise directions. Although the toroid also generalizes to three or more dimensions, we consider only the two dimensional case.

Cube-connected Cycles

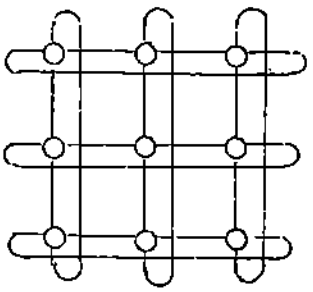
When adding nodes to the spanning bus hypercube or the toroid, the maximum internode distance is minimized by increasing the number of dimensions rather than increasing the network width w. Unfortunately, node fanout constraints generally preclude this approach because the number of connections to each node also increases as the number of dimensions increases. The cube-connected cycles network [PREP81] was designed to permit expansion by increasing the number of dimensions without violating the fanout constraint.

Each of the 2^D nodes of a D dimensional cube is replaced with a ring of D nodes. Each of the D links incident on each vertex of the cube is connected to

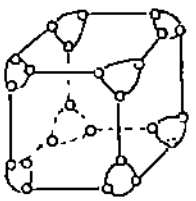
Figure I Interconnection networks



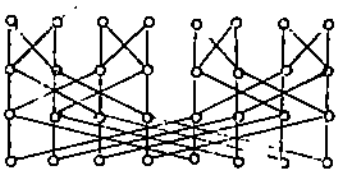
2-D Spanning Bus Hypercube ($w = 3$)



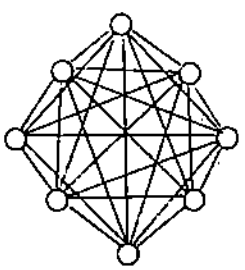
2-D Toroid ($w = 3$)



Cube-connected Cycles ($D = 3$)



2-ary N-cube ($N = 3$)



Complete Connection

a different node of the ring. Geometrically, this means that each of the now $D2^D$ nodes is connected to the two neighboring ring nodes at its vertex and one other node in the same ring position at another vertex. Because the number of connections to each node is fixed at three, expansion by increasing the number of network dimensions becomes straightforward.

2-ary N-cube

A 2-ary N-cube network [BURT81] contains $N2^N$ nodes, each of which is connected to four other nodes. The network is a variation of the indirect binary n cube originally proposed by Pease [PEAS77] with the first and last rows of switching elements being identified and with interchange boxes and links replaced by nodes and communication links.

Conceptually, the nodes are arranged on a horizontal cylinder in N rows, each of length 2^N. Thus, each node has a row and column address of the form:

$$(i, j) \qquad 0 \leq i < N \qquad 0 \leq j < 2^N \ .$$

A node in row i is connected to a node in row $(i + 1)$ *mod* N if and only if the radix two representations of their column numbers are identical except in the ith digit, with the least significant digit being considered the 0th digit.

Task Precedence Graphs

As stated earlier, our model of time varying computation is the task precedence graph. A precedence graph, illustrated in Figure II, represents a computation as a series of dependencies. The results of all computations providing input to a task, its antecedents, must become available before the task is eligible for execution. In the figure, the evaluation of antecedent tasks B, C, and D must be complete before the evaluation of task E can begin.

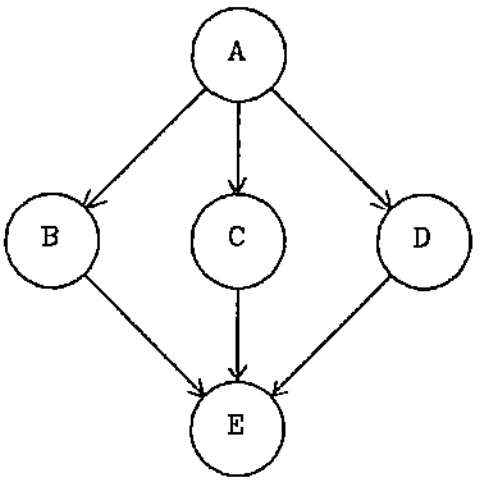


Figure II Precedence constraints for a simple graph

In each precedence graph, three types of tasks can be distinguished: fork tasks, join tasks, and regular tasks. A *fork* task has a single antecedent task and one or more consequent tasks; it represents the computation preparatory to initiation of parallel subtasks to solve a problem. A *join* task has one or more antecedent tasks and a single consequent task; it represents the combination of subproblem solutions to yield a solution to an entire problem. Finally, a *regular* task is any task that is not a fork or join task; it represents a simple computation. If we interpret the juxtaposition AB of tasks to mean "A is an antecedent of B", a task precedence graph can be formally defined by the following grammar.

<precedence graph> ::= *<regular task>* |

<fork task> *<precedence graph>*$^+$ *<join task>*

As illustrated in Figure III and summarized in Table I, the characteristics of a precedence graph are determined by several parameters. Because the number of possible graph parameterizations is so large, we have somewhat arbitrarily selected a set of values, given in Table II, to be used as a reference point in our study. By systematically varying subsets of these parameters, we obtain different performance results. By comparing these results to those obtained using the reference parameters, we can estimate the effect of the variations.

Simulation Methodology

For comparative purposes, we generated twenty five task precedence graphs using the reference parameters shown in Table II. All service times were drawn from negative exponential distributions, the number of consequents of each fork task was uniformly distributed between B_{min} and B_{max}, and all graphs were constrained to have between *Maxtasks* / 2 and *Maxtasks* tasks. Each node was assumed to possess complete knowledge of the network state and each task eligible for execution was scheduled on the idle node nearest to its location. We will return to this assumption when discussing distributed scheduling algorithms. Finally, because each network node is by assumption a single chip, it has a fixed communication bandwidth. To model this fact, we scaled the mean data communication times by the number of link connections to each node.

The average parallelism P attained when evaluating a precedence graph on a network has been taken as the measure of performance. This is

$$P = \frac{\sum_{i=1}^{Numtasks} S_i}{parallel\ execution\ time} \qquad where \quad S_i \in \{ S_P, S_R, S_J \} .$$

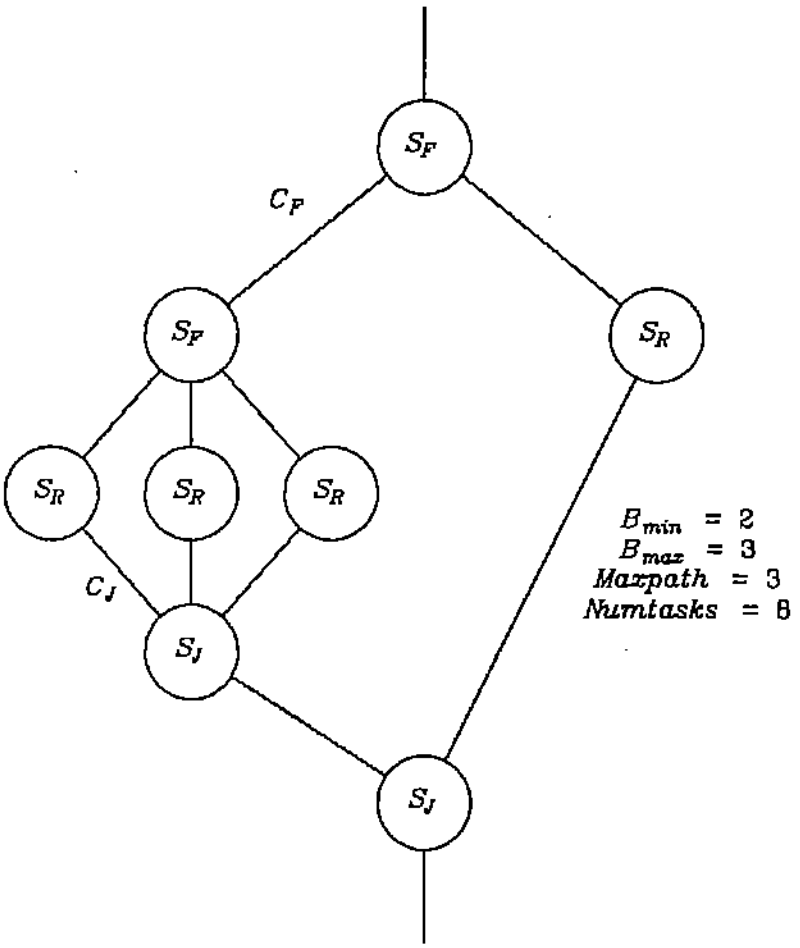


Figure III Illustration of precedence graph parameters

Table I Precedence graph parameters

Quantity	*Definition*
B_{min}	minimum number of consequents of a fork task
B_{max}	maximum number of consequents of a fork task
C_F	mean data communication time to initiate a fork or regular task
C_J	mean data communication time to initiate a join task
Maxpath	maximum length path through the graph
Numtasks	number of tasks in the graph
S_F	mean fork task service time
S_R	mean regular task service time
S_J	mean join task service time

Simulation Experiments

Using the assumptions discussed above, we explored five different variations of precedence graph parameters and network characteristics and their effect on network performance:

- precedence graph structure,
- the event horizon of a distributed task scheduler,
- the maximum task branching factor,
- the mean computation time/communication time ratio, and
- the number of network nodes.

Each of these variations is discussed below; we conclude with some general observations.

Table II Reference values for precedence graph parameters

Quantity	*Value*
B_{min}	1
B_{max}	4
C_F , C_J	1
Maxpath	60
Maxtasks	1024
S_F , S_R , S_J	10

Precedence Graph Structure

Figure IV shows the graph parallelism when each of the twenty five graphs derived from the reference graph parameters was simulated on the five networks with 64 nodes. The precedence graphs were sorted in increasing order of parallelism on the complete connection. Table III shows the average parallelism over the set of graphs using each network.

Two features of Figure IV are of particular interest. The first is the way networks other than the complete connection exhibit the same performance trends from precedence graph to graph. This suggests that something inherent to the graphs is affecting the time required for their evaluation. To determine what this might be, we examined two precedence graphs, numbers nine and eleven in the figure, that represented two extremes of behavior. Figure V shows the time varying parallelism when the two graphs were evaluated on a 2-D toroid with 64 nodes. The simulation of precedence graph nine exhibits a striking decrease in the number of parallel tasks near time 90. Because a similar simulation on the complete connection exhibits no such decrease, we can only conclude that this variation is caused by the collapse of a parallel subgraph requiring the transmission of results across several communication

Table III
Average graph parallelism for 64 node networks
using the reference precedence graph parameters

Network	*Average parallelism*	*Fraction of complete connection*
Complete Connection	22.17	1.00
Cube-connected Cycles	15.33	0.69
2-ary 4-cube	15.42	0.70
2-D Spanning Bus Hypercube	18.04	0.81
2-D Toroid	14.75	0.67

links. During the delay caused by this transmission, tasks otherwise eligible for execution were forced to wait for these results.

The time required to evaluate a single precedence graph can vary considerably even if no other computation is being performed in the network. As discussed earlier, a task eligible for execution is scheduled on the the nearest idle node. If more than one such node is available, a single node is selected at random from among them. Different sequences of choices can result in the placement of entire subgraphs in different parts of the network, causing increased communication delays. This is illustrated in Figure VI for precedence graphs nine and eleven when they were evaluated ten times on the 2-*D* toroid. The efficacy of this randomized scheduling is discussed in the next section.

Figure IV also points out the performance differential between the spanning bus hypercube and the networks using dedicated links. Although, this behavior may appear somewhat anomalous in light of the apparently greater communication bearing capacity of the dedicated link networks, this is not the case. A detailed examination of the simulation results shows that tasks are

generally scheduled for execution on nodes near their point of origin. In other words, the precedence graph evaluation exhibits considerable communication locality. For this communication pattern and the given ratio of computation time to communication time for tasks, the utilization of the communication links is low. Because of this, the buses of the spanning bus hypercube permit more rapid distribution of tasks to other nodes than the dedicated links of the other networks. For the same reason, distinct differences among the dedicated link networks are also not apparent.

Event Horizon of a Distributed Scheduler

Heretofore we have assumed that the task scheduler at each node always possesses complete knowledge of the global network state. In practice, only limited information is available, and it is often no longer completely accurate when it is received from distant nodes.

To determine a scheduler's operation in the face of partial knowledge, we postulated the existence of an *event horizon* for each network node. We assume the scheduler at each node has no knowledge of network activity at any nodes beyond its event horizon and that it must schedule all eligible tasks on nodes within its event horizon. Using the reference precedence graph parameters, Figure VII shows the average graph parallelism as a function of the distance to the event horizon from a node. Similar results are obtained when the ratio of computation times to communication times is between 1:1 and 100:1. Based on this limited evidence, it appears that state knowledge of nodes within a small distance from each source node is sufficient to achieve reasonable results. This is encouraging because it suggests that efficient distributed schedulers can be constructed for multimicrocomputer networks.

Two final observations about distributed schedulers should be made. First, this dynamic scheduling strategy does not use the precedence graph structure to aid its decisions. Although optimal distributed scheduling is known to be *NP*-complete, it should be possible to design heuristics that take advantage of some graph specific information. Van Tilborg and Wittie [VANT81] obtained some promising preliminary results for hierarchical distributed schedulers that attempt to map subgraphs onto a small group of adjacent nodes. Unfortunately, the algorithm requires the subgraphs to be known in advance, and the tasks must represent a sufficiently large amount of computation to justify the non-trivial overhead required for scheduling. Further research is needed to determine acceptable heuristics for scheduling dynamically created tasks of the size expected in mini-max game tree searches [ALK80] or distributed finite element problem solvers [SMIT82].

Second, the acquisition of state information from nodes within an event horizon is decidedly more difficult for networks connected by buses than for those using dedicated communication links. This is primarily because so many more nodes are within a small number of bus crossings from a source node. Communicating state information to other nodes on the same bus could conceivably consume a significant portion of the available communication bandwidth. Additional work is needed to determine the cost of acquiring state information.

Task Branching Factors

The average number of fork task consequents is a measure of the rapidity with which a computation subdivides into independent tasks. If this division is done too slowly, the computation will not achieve enough parallelism to

effectively use all the network nodes. Conversely, if tasks subdivide too quickly, they may be unable to diffuse through the network rapidly enough to find idle nodes. If the average number of fork task consequents is greater than the number of link connections to each node, some tasks will be *forced* to wait for access to a communication link before being scheduled on another node. Thus, finding an acceptable task branching factor is important.

Figure VIII shows average graph parallelism as a function of B_{max}, the maximum number of fork task consequents. Although the performance of the complete connection increases as B_{max} increases, no such gain is seen for partially connected networks. One should be somewhat chary about drawing general conclusions from such a paucity of data, but it appears that branching factors much greater than the connectivity of a network are not of great value.

Ratio of Computation to Communication Time

Finally, Figure IX shows the effect of increasing the ratio of computation time to communication time for tasks. As expected, the average parallelism increases as each task represents more useful computation. Similarly, Figure X illustrates network performance as a function of the number of nodes.

Summary

We have presented a model of time dependent parallel computation and studied the behavior of five multimicrocomputer interconnection networks supporting computations similar to those of the model. Among the issues considered were the relative performance of interconnection networks and the efficacy of distributed scheduling using incomplete information.

For small, dynamically created tasks, the spanning bus hypercube appears to have better performance than the dedicated link networks because it can diffuse work more rapidly. This is not always true; if message routing does not exhibit enough locality (i.e., messages must cross many links to reach their destination), the smaller communication bearing capacity of the spanning bus hypercube will be saturated, and the dedicated link networks will be preferred. Clearly, the selection of a particular network must be made with knowledge of communication patterns and task sizes required by an algorithm.

Finally, dynamic task scheduling using only locally available information seems successful for the class of algorithms represented by precedence graphs, suggesting that efficient distributed schedulers can be designed.

References

AKL80 S. G. Akl, D. T. Barnard, and R. J. Doran, "Simulation and Analysis in Deriving Time and Storage Requirements for a Parallel Alpha-beta Algorithm," *Proceedings of the 1980 International Conference on Parallel Processing*, August 1980, pp. 231-234.

BURT81 F. W. Burton and M. R. Sleep, "Executing Functional Programs on a Virtual Tree of Processors," *Proceedings of the 1981 Conference on Functional Programming Languages and Computer Architecture*, October 1981, pp. 187-194.

ELDE79 O. I. El-Dessouki and W. H. Huen, "Distributed Enumeration on Network Computers," *Proceedings of the 1979 International Conference on Parallel Processing*, August 1979, pp. 137-146.

INTE81 Intel Corporation, "MCS-51 Family of Single Chip Microcomputers Users Manual," July 1981.

PEAS77 M. C. Pease, "The Indirect Binary n-cube Microprocessor Array," *IEEE Transactions on Computers*, Vol. C-26, No. 5, May 1977, pp. 458-473.

PREP81 F. P. Preparata and J. Vuillemin, "The Cube-Connected Cycles: A Versatile Network for Parallel Computation," *Communications of the ACM*, Vol. 24, No. 5, May 1981, pp. 300-309.

REED83a D. A. Reed and H. D. Schwetman, "Cost-Performance Bounds for Multimicrocomputer Networks," *IEEE Transactions on Computers*, Vol. C-32, No. 1, January 1983, pp. 83-95.

REED83b D. A. Reed, "Performance Based Design and Analysis of Multimicrocomputer Networks," *PhD Dissertation*, Purdue University, in preparation.

SMIT82 C. U. Smith and D. D. Loendorf, "Performance Analysis of Software for an MIMD Computer," *Proceedings of the 1982 ACM Symposium on Measurement and Modeling of Computer Systems*, *Performance Evaluation Review*, Vol. 11, No. 4, pp. 151-162.

VANT81 A. M. van Tilborg and L. D. Wittie, "Wave Scheduling: Distributed Allocation of Task Forces in Network Computers," *Proceedings of the Second International Conference on Distributed Computer Systems*, Paris, 1981.

WITT81 L. D. Wittie, "Communication Structures for Large Multimicrocomputer Systems," *IEEE Transactions on Computers*, Vol. C-30, No. 4, April 1981, pp. 264-273.

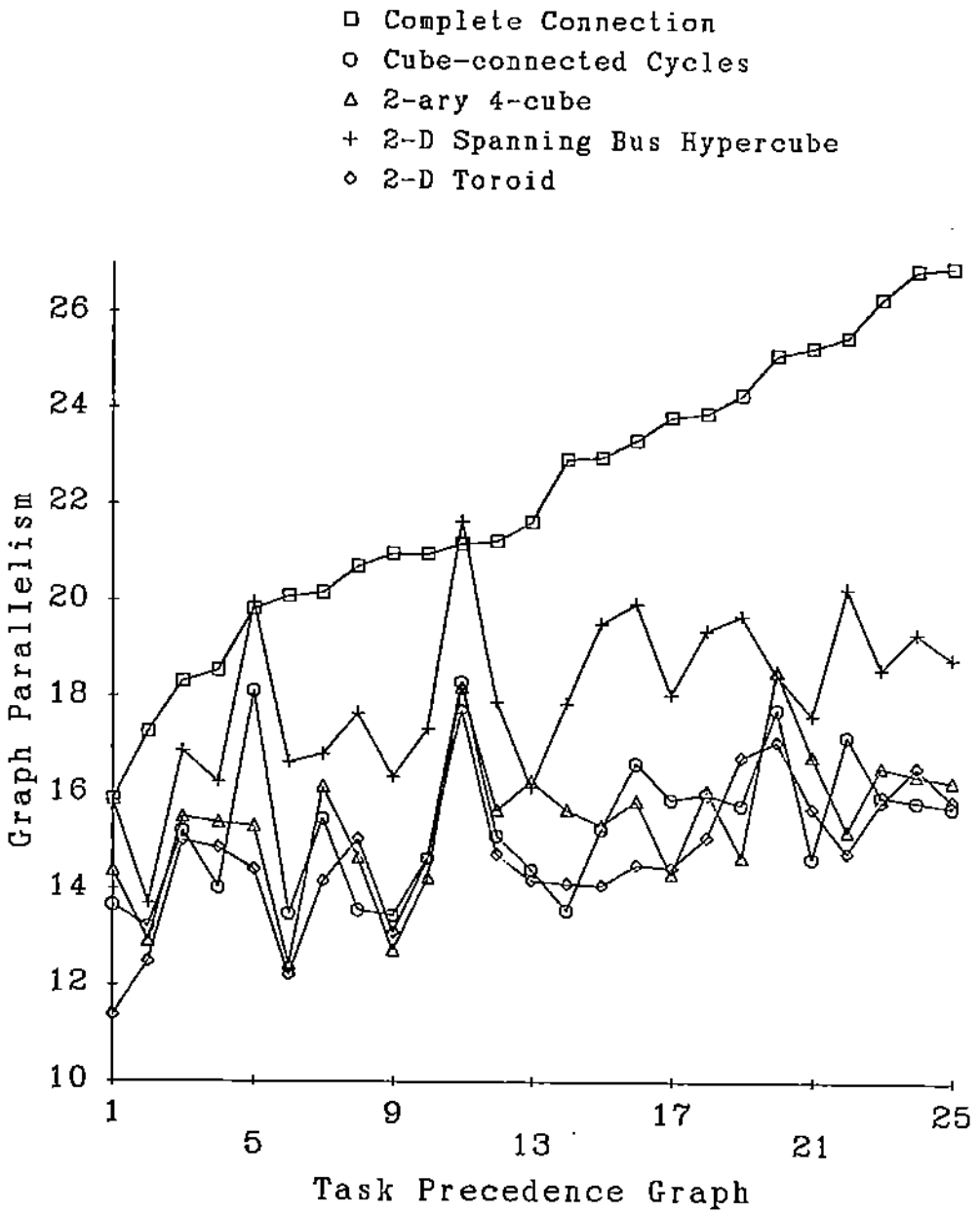


Figure IV
Graph parallelism for 64 node networks
using the reference precedence graph parameters

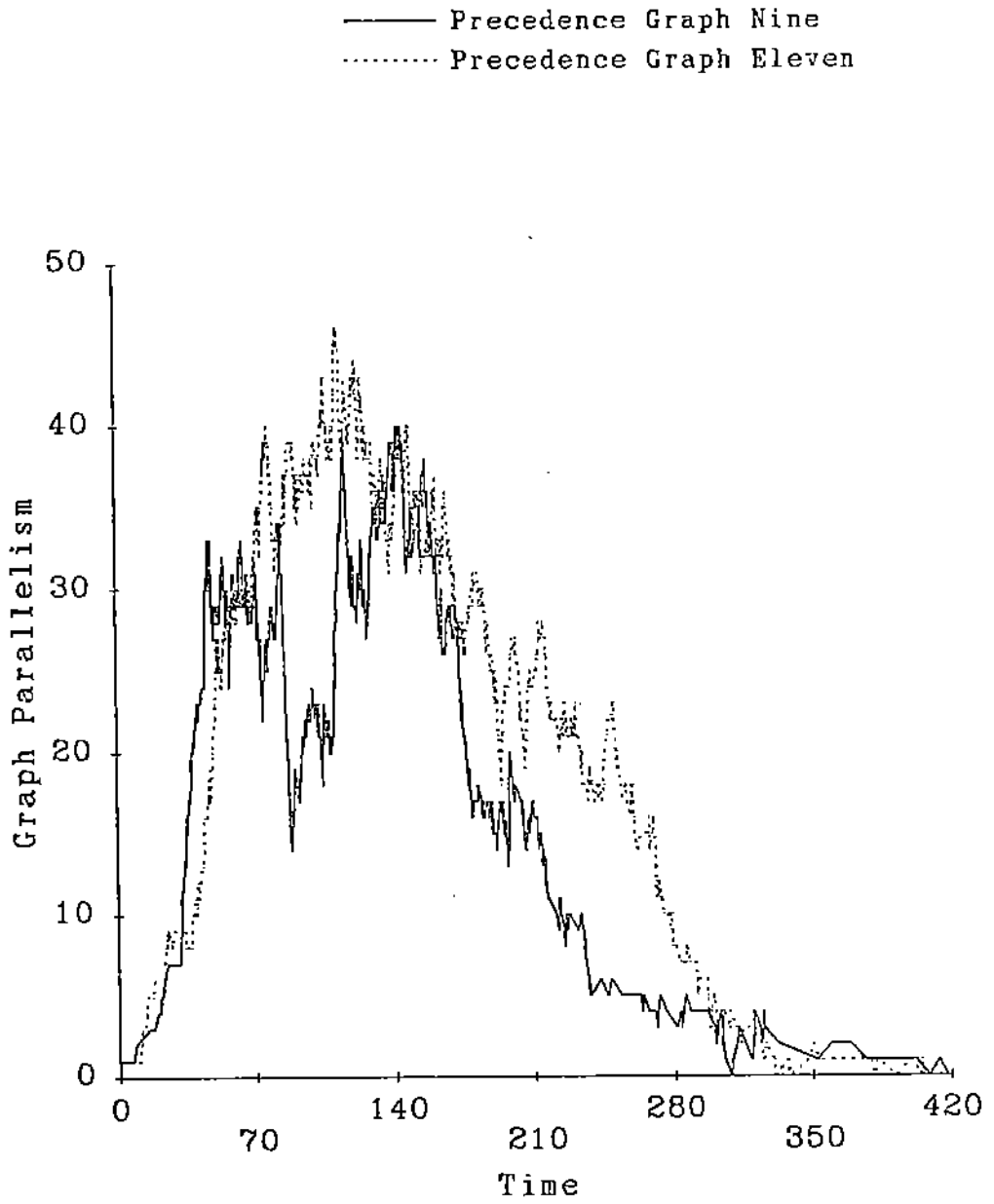


Figure V
Time varying parallelism for two precedence graphs on a 64 node toroid

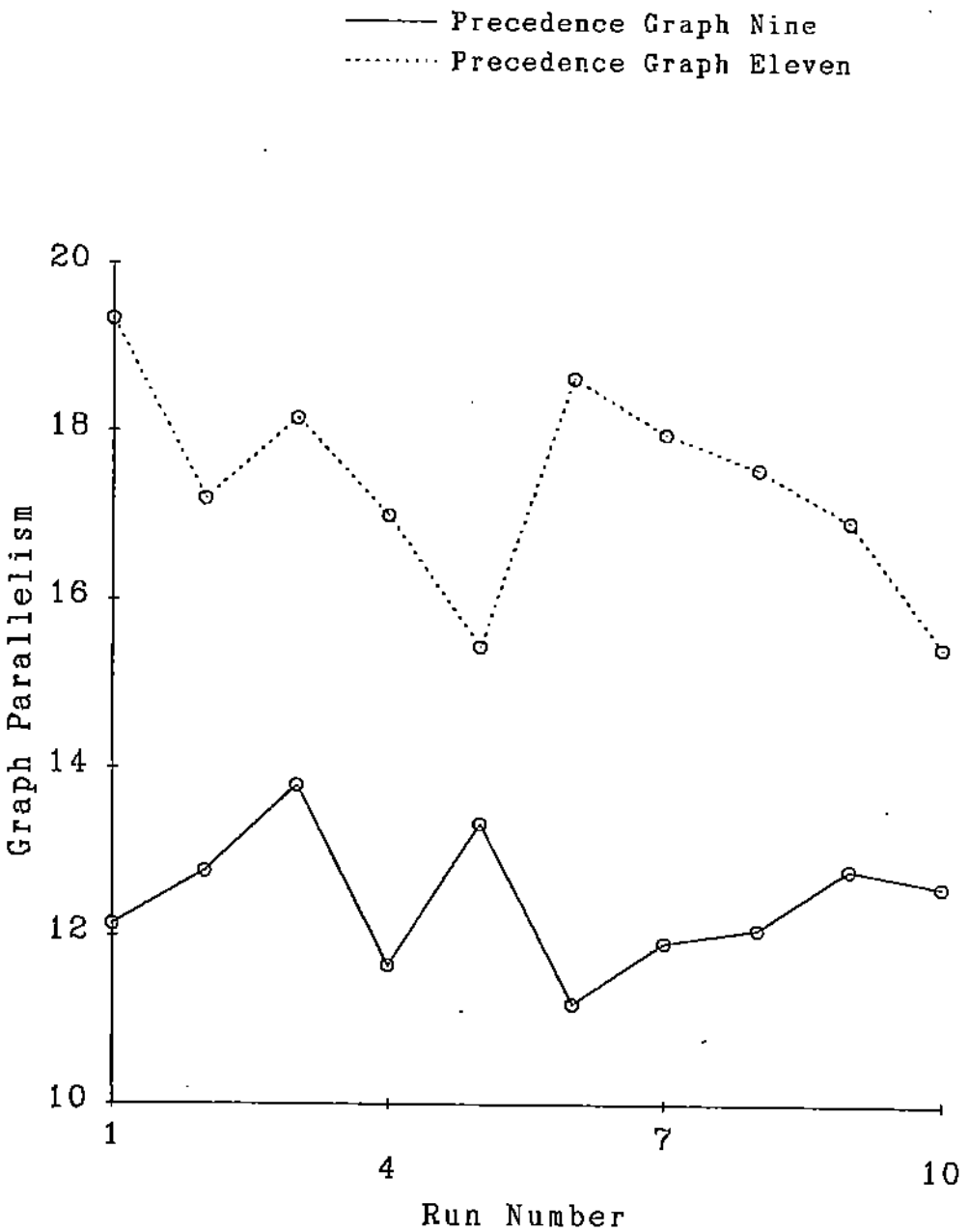


Figure VI
Variation in parallelism due to scheduling decisions

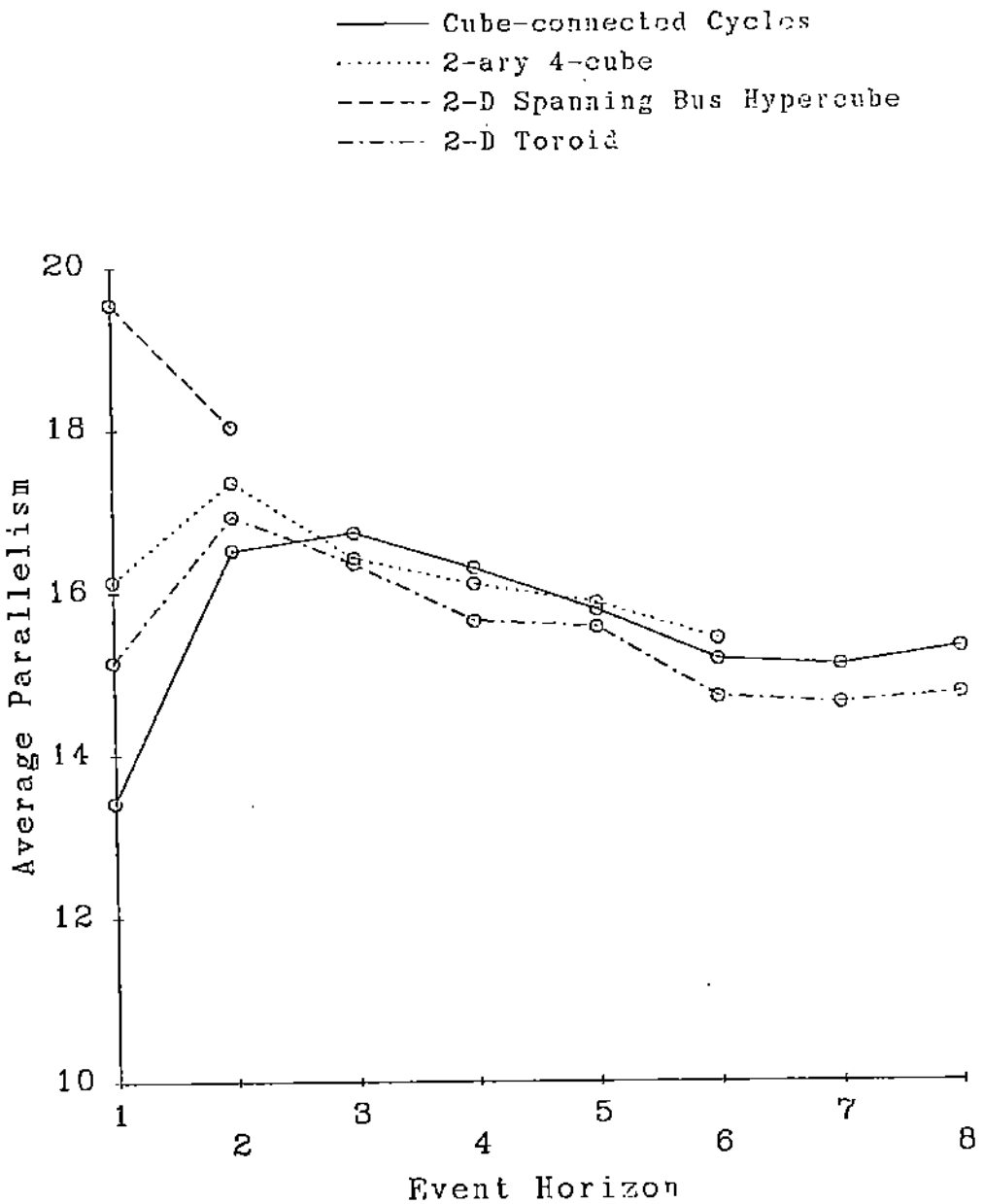


Figure VII
Average parallelism for 64 node networks
with varying amounts of scheduler information

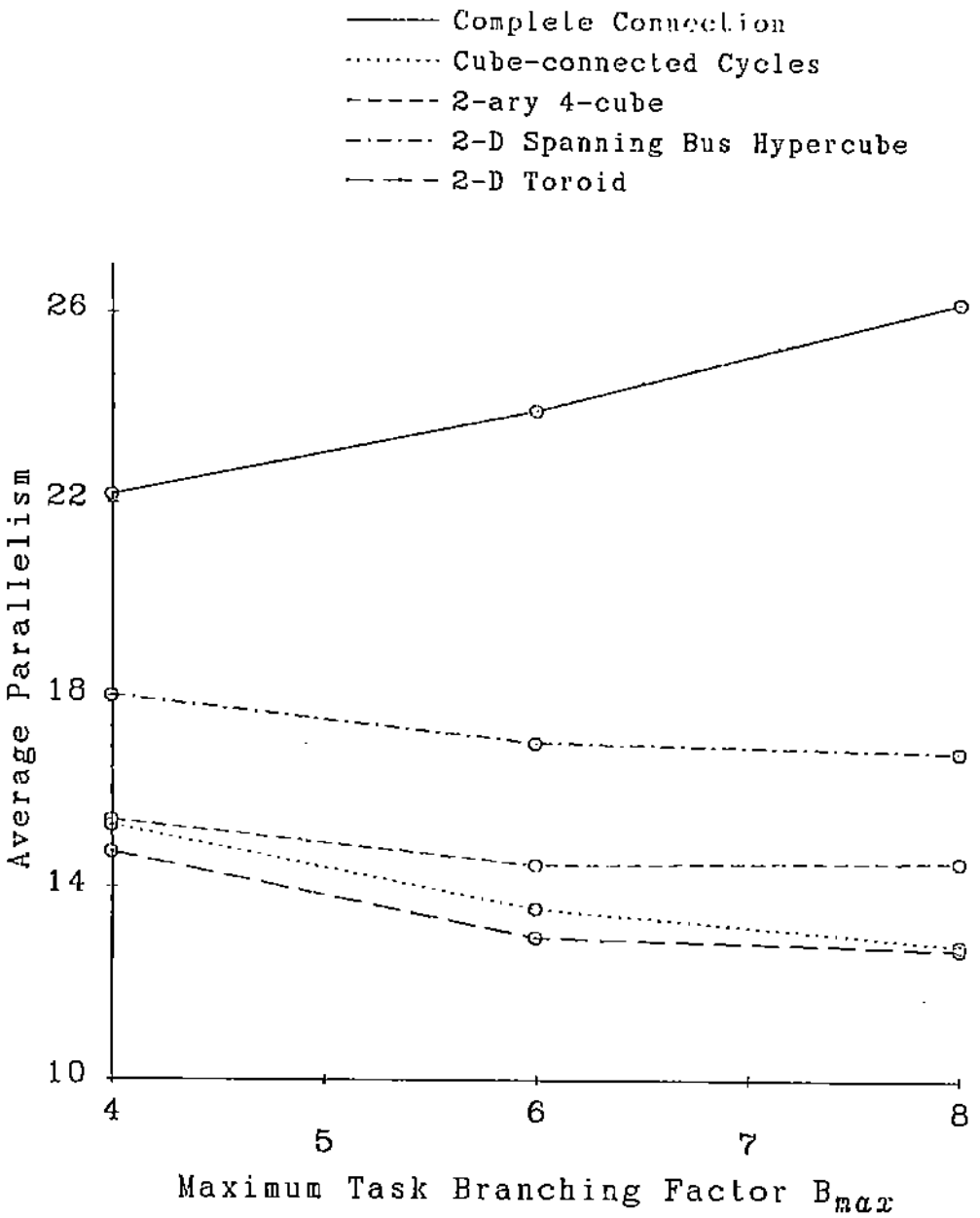


Figure VIII
Graph parallelism for 64 node networks
using varying maximum task branching factors B_{max}

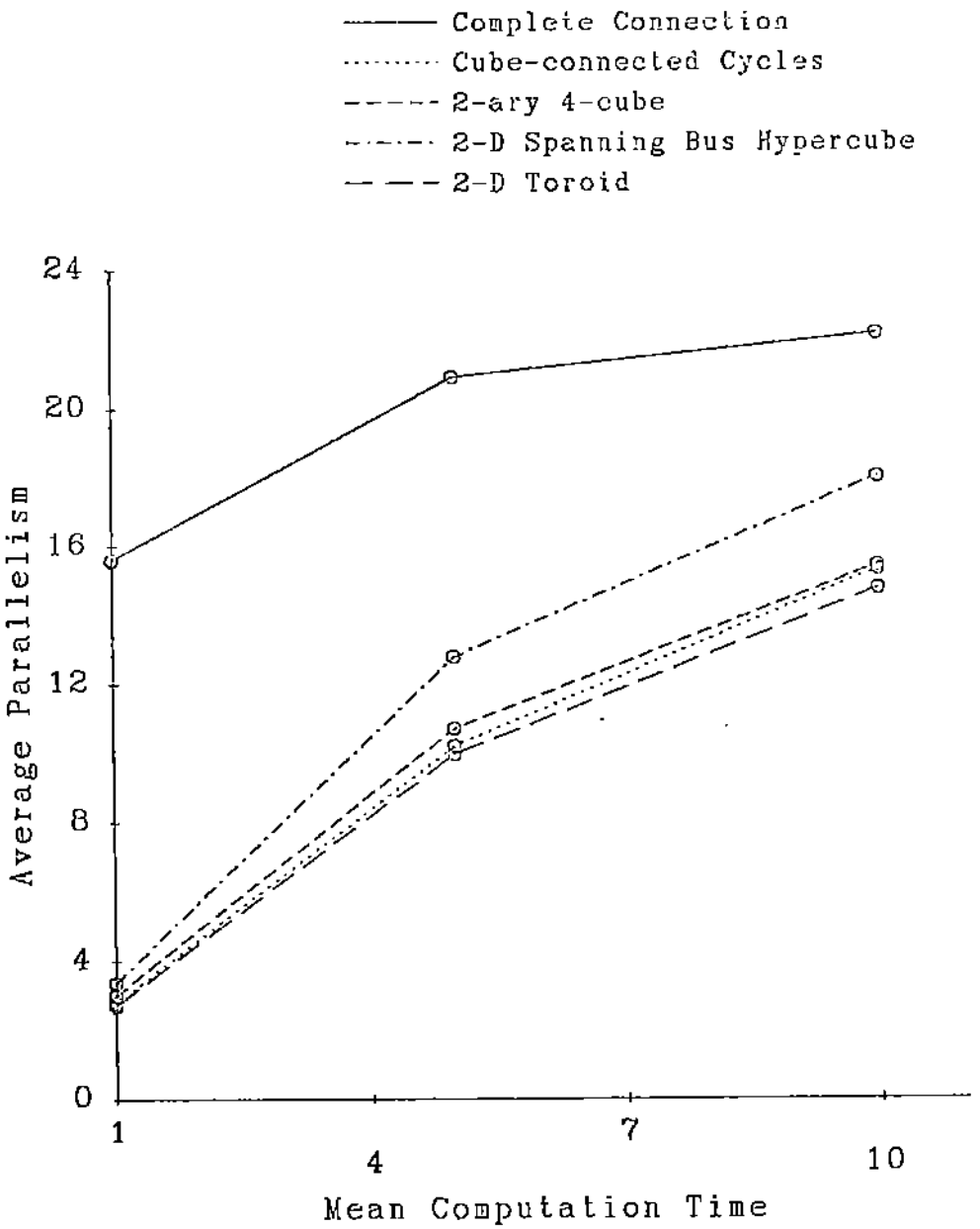


Figure IX
Graph parallelism for 64 node networks with unit communication time and varying computation time

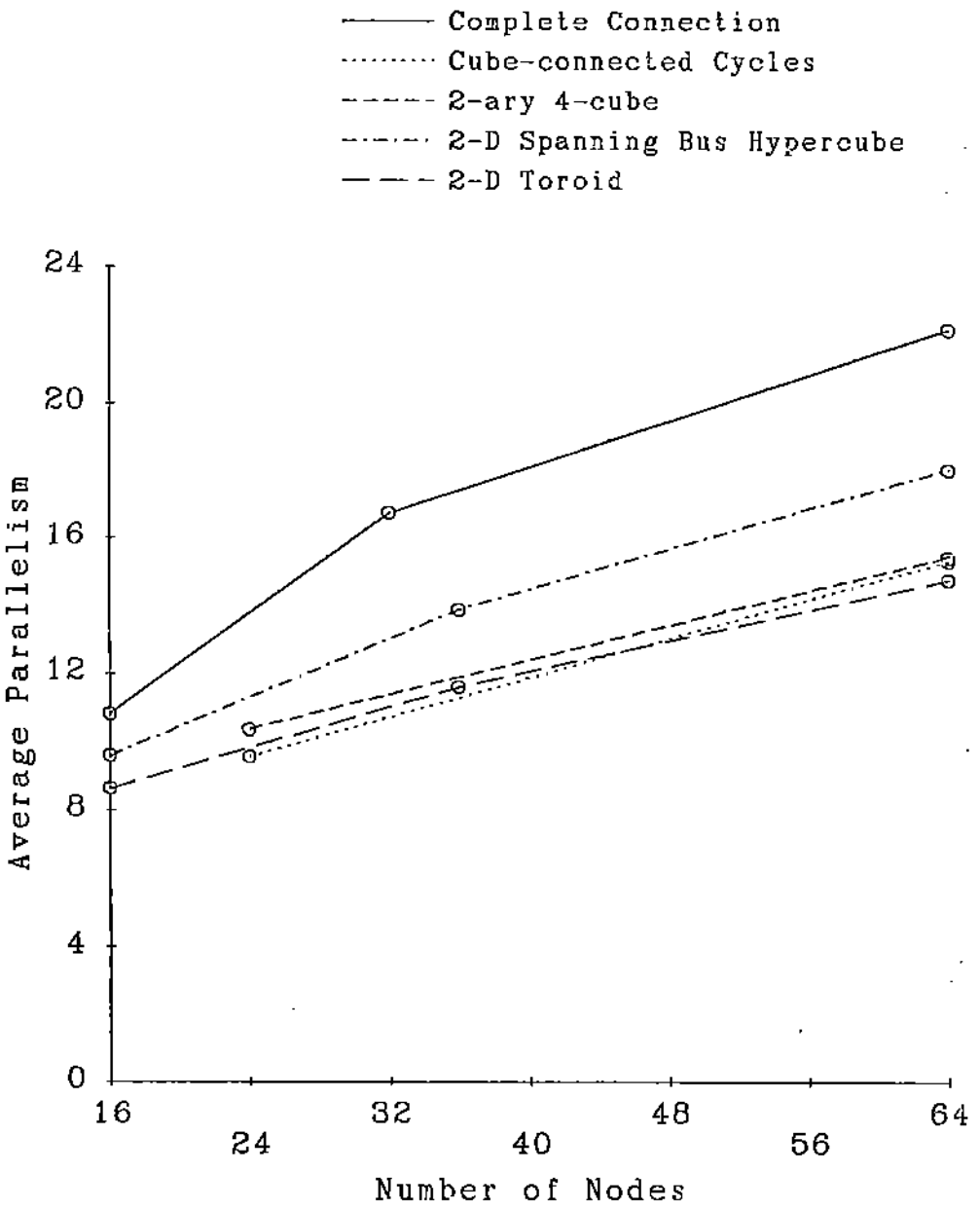


Figure X
Graph parallelism using the reference precedence graph parameters on networks with varying numbers of nodes