NASA-CR-172,457

NASA Contractor Report 172457

ICASE REPORT NO. 84-35

ICASE

PARALLEL, ITERATIVE SOLUTION OF SPARSE LINEAR SYSTEMS: MODELS AND ARCHITECTURES

Daniel A. Reed
Merrell L. Patrick

Contract Nos. NAS1-17070, NAS1-17130
August 1984

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING
NASA Langley Research Center, Hampton, Virginia 23665

Operated by the Universities Space Research Association

National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665

Parallel, Iterative Solution of Sparse Linear Systems: Models and Architectures

Daniel A. Reed[†]

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Merrell L. Patrick[†]

Department of Computer Science
Duke University
Durham, North Carolina 27706

ABSTRACT

Solving large, sparse, linear systems of equations is a fundamental problems in large scale scientific and engineering computation. A model of a general class of asynchronous, iterative solution methods for linear systems is developed. In the model, the system is solved by creating several cooperating tasks that each compute a portion of the solution vector. A data transfer model predicting both the probability that data must be transferred between two tasks and the amount of data to be transferred is presented. This model is used to derive an execution time model for predicting parallel execution time and an optimal number of tasks given the dimension and sparsity of the coefficient matrix and the costs of computation, synchronization, and communication.

The suitability of different parallel architectures for solving randomly sparse linear systems is discussed. Based on the complexity of task scheduling, one parallel architecture, based on a broadcast bus, is presented and analyzed.

[†]The research reported here was supported in part by the National Aeronautics and Space Administration under NASA Contracts No. NAS1-17070 and No. NAS1-17130 and was performed while the authors were visitors at ICASE, NASA Langley Research Center, Hampton, VA 23665.

Introduction

Solving large, sparse systems of linear algebraic equations is a fundamental problem in large scale scientific and engineering computation. Such systems arise during the analysis of both electrical power grids and simulation of large scale integrated circuits. Frequently, the non-zero elements of the coefficient matrices are structurally symmetric but otherwise appear in irregular patterns [5, 6]. In this paper, we focus on parallel, iterative methods for solving such systems.

We consider iterative methods because they avoid the overhead associated with matrix reordering schemes. These reordering schemes reduce the *fill-in* that occurs when direct elimination methods are used. Sparse systems can be solved by iterative methods, however, without the coefficient matrix changing its form as the computation proceeds. We tacitly assume that, for any given sparse system, iteration matrices exist such that the iterations converge to the solution at a reasonable rate. Otherwise, our methods are not applicable, and the system must be solved using a direct method.

Prior Work

In earlier work [7], we developed a model for a general class of asynchronous, iterative solution methods for linear systems. In this model, a linear system is solved by creating several cooperating tasks that each compute a portion of the solution vector. The model was analyzed to determine the expected intertask data transfer and task computational complexity as functions of the number of tasks. The recommended number of tasks was then computed as a function of the sparsity of the linear system and its dimension. The results presented in [7] are summarized in the next few sections.

In related work, Amano, Yoshida, and Aiso [4] proposed a parallel architecture, the *S*parse *M*atrix *S*olving *M*achine $(SM)^2$, for iterative solution of sparse linear systems with irregular non-zero structure. Calahan [5] has investigated approaches for solving similar

systems on a pipelined architecture, the CRAY-1, using direct elimination methods.

Adams [1] has studied parallel implementations of iterative methods for the linear systems arising from finite element analysis, particularly on the *F*inite *E*lement *M*achine *FEM*, under development at the NASA Langley Research Center. However, the linear systems of her study have symmetric coefficient matrices with non-zero elements appearing in regular patterns.

Overview

In the remainder of the paper, we precisely define both the problem and the class of iterative methods used to solve it. We discuss different ways of partitioning the problem for parallel solution and the costs associated with the partitioning. We then define a data transfer model predicting both the probability that data must be transferred between two partitions and the amount of data to be transferred. Using this data transfer model, we derive a probabilistic technique for predicting the execution time and optimal number of tasks given the dimension and sparsity of the coefficient matrix and the costs of computation, synchronization, and communication.

Finally, the suitability of different parallel architectures for solving randomly sparse linear systems is discussed. Based on the complexity of task scheduling, we present and analyze one parallel architecture based on a broadcast bus.

Problem Definition

Consider a linear system of equations of the form

$$Kx = f \tag{1}$$

where K is a large $N \times N$ sparse matrix and x and f are vectors of length N. Such systems are frequently rewritten in the form

$$x = Ax + c$$

and solved using the iteration formula

$$x^{(i+1)} = Ax^{(i)} + c \qquad (2)$$

where x and c are N-vectors and A is another sparse $N \times N$ matrix. Although A is a function of K, and c is a function of K and the f vector, there are many ways to choose A and c such that (2) describes a convergent iterative scheme for (1). We only assume that they are chosen such that the sequence of iterates $<x^{(i)}>$ converges to the solution. Henceforth, we consider only the parallel implementation of the computation defined by (2).

Parallel Solution Technique

One parallel computation schema for (2) is illustrated by the diagram in Figure I. The matrix A and the vectors c and $x^{(i+1)}$ (denoted by XN), with the rows possibly rearranged, are partitioned into sets of rows. A basic iteration step of the computation is then partitioned into the set of computations defined symbolically by

$$\text{XSET[I]} = \text{ASET[I]} * \text{XO} + \text{CSET[I]} \quad \text{I} = 1,...,\text{M}.$$

After each basic iteration, a norm of the vector XN - XO must be checked for convergence. If convergence has occurred or the maximum allowable number of iterations has been exceeded, the iteration halts; otherwise XO is replaced by XN, and the iteration step is repeated.

This computation schema can be realized as follows. In the main program, the data objects and their types are declared. In addition, worker tasks, called X_TASKs, and their controlling task, the C_TASK, are defined. The body of the main program reads the input data, initiates the control task, which in turn initiates the worker tasks,

and prints the solution vector after the control task has terminated.

X_TASK[I] computes the components of the vector XN corresponding to XSET[I]. To accomplish this, X_TASK[I] needs the non-zero elements of A corresponding to ASET[I], the elements of c corresponding to CSET[I], and a portion of the vector XO, specifically those elements whose subscripts are the same as the column subscripts of the non-zero elements of A in ASET[I]. Initially, this information is sent to each of the X_TASKs by the C_TASK. After each iteration, if convergence has not occurred, the vector XO is replaced with the vector XN. Because of this replacement, each X_TASK must send those components of XN that it computes to the other X_TASKs that need them. Each X_TASK then determines if the XN components just computed have converged and notifies the C_TASK accordingly by sending a boolean flag. When an X_TASK is notified by the C_TASK that all X_TASKs report convergence of their components, it sends the current values of its XN components to the C_TASK and terminates.

The role of the C_TASK is now clear. After initializing the X_TASKs and sending them the initial data they need to iterate, it receives local convergence information from each X_TASK, determines if global convergence has occurred, and notifies the X_TASKs accordingly. If global convergence has occurred, the C_TASK then receives the components of XN and terminates.

Problem Partitioning

Clearly, the rows of the iteration matrix A must be assigned to X_TASKs. There are an enormous number of possibilities, ranging from equal size partitions of contiguous rows to arbitrary row permutations. Indeed, there are

$$\frac{N!}{\left(\frac{N}{M}\right)!}$$

distinct ways of assigning the N rows of A to M *equal size* tasks.

Optimally, a row assignment algorithm should use knowledge of the matrix structure to minimize the expected (parallel) iteration time. Unfortunately, the computational cost of finding an optimal row assignment is both expensive relative to the cost of solving the system *and* dependent on the underlying parallel architecture. Hence, we initially consider only partitions composed of contiguous matrix rows. Although significant latitude in row assignments is sacrificed, differing size partitions are possible. Moreover, this restriction greatly simplifies later analysis, and, as we shall see, is not an unreasonable constraint for the class of bus-connected architectures. Indeed, if the matrix has truly random structure, equal size, contiguous partitions will place approximately equal numbers of non-zero elements in each partition.

An Analytical Model of the Computation

Objectives

Because the intent of parallel computation is a reduction of the expected execution time, we must consider the *performance* of the parallel, sparse, linear systems iteration algorithm just described. Unlike sequential algorithms, the performance of a parallel algorithm depends not only on the number of arithmetic operations but also on the amount and frequency of inter-task data transfer. Consequently, we derive formulae describing

- the amount of data transfer among X_TASKs needed for each iteration,
- the computational complexity of each X_TASK, and
- the time to synchronize the X_TASKs.

Based on these formulae, we create a model for predicting performance as a function of both the number and size of matrix partitions and the matrix sparsity.

Notation and Assumptions

Unless otherwise specified, we assume the elements of the matrix A are randomly non-zero with probability P (i.e., $p\ (a_{ij} \neq 0) = P$), as defined below. In our model of matrix sparsity, the probability function P is determined by imposing two very weak conditions on A. First, we require each row of A to contain at least Z non-zero elements, each randomly distributed throughout the row. Second, each row element not known to be one of the Z non-zero values is itself assumed to be non-zero with probability q.

Given the two conditions above, the value of P can be derived using a straightforward application of conditional probabilities. We define two events:

E: a_{ij} is one of the Z non-zero elements in row i

F: a_{ij} is a non-zero element with probability q but *not* one of the Z non-zero elements

Then, with N being the number of rows in A,

$$\begin{aligned} P(N, Z, q) &= p(a_{ij} \neq 0) \\ &= p(E) + p(F) - p(E \text{ and } F) \\ &= \frac{Z}{N} + q - q\left(\frac{Z}{N}\right) \\ &= \frac{Z}{N}(1 - q) + q. \end{aligned}$$

Finally, we require Z to be greater than zero (i.e., there is at least one non-zero element in each row).

Throughout our discussion, M denotes the number of partitions of A (i.e., the number of X_TASKs), and b_j and e_j respectively denote the indices of the beginning and ending rows of partition j. This notation, and that introduced throughout the remainder of our matrix analysis, is summarized in Table I.

Data Transfer for Sparse Matrices

Given a sparse matrix A whose elements are randomly non-zero with probability $P(N, Z, q)$ and two partitions j and k, we wish to determine the data transfer from partition k to partition j needed to perform one iteration. For pedagogical purposes, we consider three cases of increasing generality.

Case I: j and k are single row partitions

Partition j requires the single value x_{b_k} if and only if $a_{b_j b_k} \neq 0$. Since this occurs with probability $P(N, Z, q)$, the expected data transfer from k to j is simply $P(N, Z, q)$.

Case II: j is a multiple row partition; k is not

Clearly, partition j does not need x_{b_k} if and only if $a_{ib_k} = 0$ for all i in the range $b_j \leq i \leq e_j$. By assumption, each matrix element is randomly non-zero. Hence, the probability that at least one element of the column b_k in partition j is non-zero is

$$1 - \Big[1 - P(N, Z, q)\Big]^{e_j - b_j - 1}$$

Because partition k contains only one row, the expected data transfer from k to j is the same.

Case III: both j and k are multiple row partitions

This case is illustrated in Figure II. An immediate generalization of the previous case, partition j does not need any x_l if and only if $a_{il} \neq 0$ for i in the range

$b_j \leq i \leq e_j$ and all l in the range $b_k \leq l \leq e_k$. Consequently, the expected data transfer from partition k to partition j is just $(e_k - b_k + 1)$ times that of case II, namely,

$$Tr(k, j) = (e_k - b_k + 1)\left[1 - \left[1 - P(N, Z, q)\right]^{e_j - b_j + 1}\right]. \tag{3}$$

Finally, the probability, $P_{Tr}(k, j)$, that partition j needs at least one element from partition k is just the probability that the submatrix delimited by rows b_j and e_j and columns b_k and e_k, matrix $\hat{A}$ in Figure II, is not identically zero. This probability is just

$$P_{Tr}(k, j) = 1 - \left[1 - P(N, Z, q)\right]^{(e_j - b_j + 1)(e_k - b_k + 1)} \tag{4}$$

Although general, (3) and (4) provide little insight or intuition about data transfer as a function of either $P(N, Z, q)$ or M. If the partition size is constant, simpler expressions can be obtained. Hence, we fix $(e_j - b_j + 1)$, the partition size, at a constant $S = N/M$ for all partitions. Then replacing $P(N, Z, q)$ by its definition, we obtain

$$Tr(k, j) = \frac{N}{M}\left[1 - \left[1 - q\right]^{\frac{N}{M}}\left[1 - \frac{Z}{N}\right]^{\frac{N}{M}}\right] \tag{5}$$

and

$$P_{Tr}(k, j) = 1 - \left[\left[1 - q\right]^{\left(\frac{N}{M}\right)^2}\left[1 - \frac{Z}{N}\right]^{\left(\frac{N}{M}\right)^2}\right]. \tag{6}$$

Parallel Computational Complexity

As noted earlier, the performance of a parallel algorithm depends on both the intertask data transfer *and* the amount of computation performed by each task. Having considered the former, we turn our attention to the latter.

Each of the parallel X_TASKs is itself just a sequential code whose two primary constituents, inner product and convergence test, were described earlier. Consequently, we can apply standard techniques [3] to determine the complexity of each X_TASK. The results of this analysis are shown in Table II.

We assume that all indexing and arithmetic operations require the same amount of time C_p. Combining the results for the inner product and convergence test, the computational complexity of an arbitrary X_TASK is

$$(e_j - b_j + 1)C_p(6NP(N, Z, q) + 16) + 5C_p \tag{7}$$

for the randomly sparse matrix. The C_TASK must also check for global convergence after each iteration. This consists of ANDing the M local convergence flags received from the X_TASKs and requires

$$(3M + 1)C_p \tag{8}$$

operations.

Model Description

Having just determined the expected amount of data transfer among X_TASKs (partitions), and their computational complexity, we can now define an execution time model of the parallel, sparse matrix algorithm. This model can then be used to predict the execution time of one iteration.

Let $t_j(comp)$ denote the computational complexity of X_TASK j, $t_j(comm)$ denote the time required for task j to send and receive all data needed for the next iteration, and $t(sync)$ be the time required for the C_TASK to receive and test all local synchronization flags. Then the total execution time for one sparse matrix iteration is

$$t(sync) + \max_{1 \le j \le M} \left\{ t_j(comp) + t_j(comm) \right\}. \tag{9}$$

Clearly, the time required to transmit or receive a datum is some function of the number of partitions (X_TASKs) concurrently operating (e.g., if only two X_TASKs were operating in parallel, they should be able to exchange data more quickly than if fifty additional X_TASKs were also operating). Hence, we make both the time needed to transmit a boolean, $C_b(M)$, and the time to transmit an x value, $C_t(M)$, functions of M.

We now consider each component of the execution time. Given that $C_b(M)$ denotes the time needed to transmit a single boolean value, then $t(sync)$ is given by

$$\begin{array}{ccccc} \textit{RECEIVE FLAGS} & & \textit{TEST FLAGS} & & \textit{SEND FLAGS} \\ MC_b(M) & + & (3M + 1)C_p & + & MC_b(M). \end{array}$$

Of course, $t_j(comp)$ is given by (7). The communication component, $t_j(comm)$ is, however, somewhat more complicated. In addition to including the interpartition data transfer, it should also include startup costs for data transmission. That is, two partitions exchanging ten data values should require less time than four partitions exchanging five data values. This intent is reflected by the formula

$$t_j(comm) = \textit{send to other partitions} \tag{10}$$

$$+ \quad \textit{receive from other partitions}$$

$$= \sum_{\substack{k=1 \\ k \neq j}}^{M} \left[C_s P_{Tr}(k, j) + C_t(M) Tr(k, j) \right]$$

$$+ \sum_{\substack{k=1 \\ k \neq j}}^{M} C_t(M) Tr(k, j)$$

where C_s is the startup cost for initiating a data transfer.

Given these formulae, consider the matrix case for which we derived closed forms for $P_{Tr}(k, j)$ and $Tr(k, j)$, the randomly sparse matrix with fixed partition size.

Substituting values from (5) and (6) in (10) for $P_{Tr}(k, j)$ and $Tr(k, j)$ gives

$$t_j(comm) =$$

$$C_s(M - 1)\left[1 - \left[(1 - q)\left[1 - \frac{Z}{N}\right]\right]^{\left[\frac{N}{M}\right]^2}\right] +$$

$$\frac{2C_t(M)N(M - 1)}{M}\left[1 - \left[(1 - q)\left[1 - \frac{Z}{N}\right]\right]^{\frac{N}{M}}\right].$$

Conclusions Based on the Model

As we have seen, the total execution time for one sparse matrix iteration is given by (9). For equal sized partitions, (9) simplifies to

$$t(sync) + t_j(comp) + t_j(comm). \tag{11}$$

There are two primary means of implementing communication in a parallel system, shared memory and communication networks. In both cases, the delays incurred for data transfer increase as the number of parallel tasks increase. (Shared memory suffers from memory access conflicts, and communication networks, being necessarily incomplete connections, require additional routing of data.) Hence, it seems appropriate to make the synchronization and data transmission costs functions of the number of partitions M (i.e., the number of parallel X_TASKs). We used the functions

$$f(M) = \begin{cases} 1 \\ \log_2(M) \\ \sqrt{M} \\ M \end{cases}$$

in the communication component of (11) to reflect the possible range of communication costs one might encounter.

Using (11) and the communication cost function, $f(M)$, we then plotted total execution time as a function of matrix sparsity, $P(N, Z, q)$, computation time, C_p, communication time, C_t and C_s, and synchronization cost, C_b, for the random sparsity case. These plots, shown in Figures III-V, are discussed in detail below. In all cases, the smallest number of partitions chosen was $M = 5$.

Figure III

This figure shows iteration time as a function of the number of matrix partitions (X_TASKs) for varying communication costs. Each matrix row contains 14 non-zero elements, a typical number for a matrix arising from a finite element method.

As can be seen, there exists an optimal level of parallelism in each case. Not surprisingly, the optimum level of parallelism declines as the communication costs increase. Even constant cost communication cannot support as many parallel tasks as there are matrix rows. The reason is quite simple, as the number of partitions grows, synchronization costs become prohibitive.

Figure IV

This figure shows the effect of matrix sparsity on iteration time for communication costs proportional to $\sqrt{M}$; the lowest curve corresponds to greatest sparsity. As expected, increasing the number of non-zero elements results in increased iteration time. In addition, the optimum level of parallelism increases as the number of non-zero elements increases.

Figure V

Finally, this figure shows iteration time for varying matrix sizes, again with communication costs proportional to $\sqrt{M}$.

Results of the model clearly show that the execution time of the solution methods can be reduced by partitioning the computation into parallel subtasks. However, the optimum number of partitions is very dependent on synchronization and communication costs.

Allocating Tasks to Processors

Although the model described above can provide some insights into the appropriate balance of parallel computation and communication, it assumes both a random assignment of tasks to processors and assignment of contiguous rows to tasks. Two important questions remain.

- What factors must a good scheduling algorithm include?
- What are the constraints on the computational complexity of such an algorithm?

Each of these questions have significant ramifications for possible parallel architectures.

The Scheduling Problem

As a first approximation, the X_TASKs of the iterative algorithm should be assigned to processors to minimize interprocessor communication. At each iteration, X_TASK[I] needs a group of XN values computed by the other X_TASKs. As we have seen, the particular XN components needed are determined by the unique column subscripts of the non-zero elements of the matrix A held by X_TASK[I]. Hence, an ideal row assignment would minimize over all tasks the sum of the number of unique column subscripts. Not surprisingly, this function is minimized when all rows are placed in a single task. Obviously, the number of tasks used must also be selected. Note, however, that the number of tasks *cannot* be selected *a priori*. There exists an optimal row assignment for a given number of tasks, but there is also an optimal number of tasks.

In practical terms, the existence of p parallel processors should not mandate use of them all; communication delays may encourage use of fewer processors. Finally, architectural considerations intrude. Of the possible partitionings of an N row matrix, some are infeasible either because

- the number of partitions exceeds the number of processors, or
- a partition size exceeds a processor memory size.

Thus, the scheduling problem becomes one of finding an optimal number of partitions (tasks) and an optimal row assignment across the partitions that minimizes iteration time while satisfying all architectural constraints. Using the notation of Table III, we can formally express the scheduling problem as follows.

minimize $IterTime\ (M, P, M_{size}, ComType)$

subject to:

$$1 \leq M \leq p$$

$$P \in \mathbf{Pset}(p)$$

$$Mem_i \leq M_{size} \qquad \forall\, Mem_i \in P$$

$$ComType \in \{\text{ shared memory, bus, } \cdots \}$$

Generally, $IterTime\ (M, P, M_{size}, ComType)$ is not a simple analytic function; it must be determined either by simulation or computation. Because there are also a combinatorial number of row assignments, it is not surprising that optimal scheduling is prohibitively expensive. In the general case, it is surely NP-complete. We find ourselves in the rather unattractive position of needing to solve an NP-complete problem as a *preprocessing* step to a problem whose sequential complexity is polynomial. Since optimal scheduling is infeasible, we must consider the viability of scheduling heuristics.

Scheduling Heuristics

If a parallel iterative method is to be effective for solution of a single linear system, the prescheduling *and* the parallel iteration must be faster than the sequential method. For a sparse matrix with $P(N, Z, q)N^2$ non-zero elements, a sequential iterative method requires $O(iP(N, Z, q)N^2)$ time, where i is the number of iterations required. If M processors were dedicated to a parallel iterative method, an M-fold speedup could at best be achieved, resulting in a parallel computational complexity of $O(iP(N, Z, q)\frac{N^2}{M})$. Thus, a scheduling heuristic should require no more than $O(iP(N, Z, q)N^2)$ time and, desirably, less than $O(iP(N, Z, q)\frac{N^2}{M})$ time.

In practical terms, these results mean a scheduling heuristic can examine each non-zero matrix element at *most* a constant number of times before making a decision. Moreover, if more than a constant performance improvement is desired, the entire matrix cannot be examined at all. This suggests that finding a good scheduling heuristic is unlikely. Two natural alternatives then present themselves.

- Do not attempt to schedule.
- Select a parallel architecture where scheduling is not needed.

The first approach was adopted in our earlier analysis. If the coefficient matrix is truly randomly sparse, then the computation and communication requirements of $\frac{N}{M}$ equal size partitions should be roughly equal. The second alternative promises greater performance and is the subject of the next section.

Possible Parallel Architectures

As we have seen, an architecture capable of obviating or greatly reducing the scheduling burden is needed. Although a globally shared memory meets this need, memory conflicts limit performance; moreover, such a scheme has limited extensibility. The other obvious alternative is a network whose processors appear approximately equidistant from one another. Although the equidistance criterion excludes common networks such as meshes, the global bus and networks capable of emulating a complete connection remain.

The complete connection is both expensive and impractical for large networks, but recent preliminary work by Van Rosendale and Mehrotra (private communication) suggests that multistage networks, such as the omega network [8], may well approach the complete connection in performance for iterative methods on sparse linear systems.

Iteration on a Bus Architecture

For moderate parallelism the global bus architecture provides a simple alternative to interconnection networks; see Figure VI. Although using a single bus may seem unduly conservative, given other proposed parallel architectures, the recent emergence of local area networks (e.g., ethernet) has made distributed computation *using a global bus* widely available. Thus, our discussion and analysis of bus architectures can be applied to either tightly coupled parallel systems or loosely coupled local networks.

In the bus approach, the matrix partition in each processor *broadcasts* components of XN across the bus(es) if at least one other partition needs the values. The number of broadcasts can be reduced by judicious row assignment, but even in the worst case only one copy of the current XN vector need be broadcast each iteration. Because the total amount of computation at each iteration is proportional to N^2, substantial computational parallelism can be realized before the broadcast bus becomes the performance

limiting factor (assuming the processors and the broadcast bus differ in speed by a factor that is small relative to N).

As an added benefit, convergence flags can be transmitted with the XN values allowing each X_TASK to test for global convergence. This eliminates the need for global synchronization and a convergence test by the C_TASK. Because each communication processor (see Figure VI) must watch the bus for XN components, these convergence flag broadcasts are essentially free. Finally, good analytic performance models for bus architectures can be developed, providing a means to determine an appropriate number of processors to use. We explore this last issue in the next section.

In summary, each processor on the global bus repeatedly

- computes new values for its local XN values,
- broadcasts its convergence flag and appropriate XN values,
- receives data and convergence flags, and
- computes global convergence

until all tasks have converged.

Performance Models of Bus Architectures

Consider the simplest possible bus architecture, a *single* broadcast bus. The performance of such a system is important, both because it can help determine the number of processors needed and because it can validate performance projections. Below, we present two performance models, one an optimistic performance bound and the other pessimistic.

Optimistic Performance Model

As derived earlier, the expected data transfer from one partition (task) to another for randomly sparse matrices is given by (3) and, for equal size partitions, by (5). For a bus, however, the expected amount of data *broadcast* by a task is desired. To derive this quantity, first consider the probability that a single component of an XN iterate is needed by *any* other task. This is just the probability that at least one row in some other task's partition contains a non-zero element in the column corresponding to the component of the XN iterate. If each processor j contains a partition of size $(e_j - b_j + 1)$, the probability of a broadcast by processor j is

$$P_{Br}(j) = 1 - \left[1 - P(N, Z, q)\right]^{N - (e_j - b_j + 1)}$$

and the expected size of a broadcast is

$$Br(j) = (e_j - b_j + 1)\, P_{Br}(j).$$

For simplicity's sake, we henceforth consider only partitions of equal size $\frac{N}{M}$. The processor subscript j can then be dropped, and the above formulae simplify to

$$P_{Br} = 1 - \left[1 - P(N, Z, q)\right]^{\frac{N(M-1)}{M}}$$

and

$$Br = \left(\frac{N}{M}\right) P_{Br}$$

respectively.

Because a broadcast bus has somewhat different properties than the point-to-point networks considered earlier, our execution time model is slightly different. First, we define the communication component of iteration time for one X_TASK, $t(comm)$, as

BROADCAST DATA		BROADCAST CONVERGENCE FLAG
$t(send)$		$t(send\ convergence)$
$\left[C_s P_{Br} + C_t Br \right]$	+	C_b

(Recall that convergence flags are broadcast.) Notice that it does not include a delay for receiving data from the bus. As Figure VI shows, each node is assumed to contain a broadcast processor capable of sending and receiving broadcast messages without delaying the computation processor.

The computation component of iteration time for one X_TASK, $t(comp)$, is given by

UPDATE XN		GLOBAL CONVERGENCE TEST
$\frac{NC_p}{M}\left[6NP(N, Z, q) + 16\right] + 5C_p$	+	$(3M + 1)C_p$

where the components are obtained from (7) and (8), respectively.

Given the communication and computation components of iteration delay, we can now develop a performance model. This simple machine repairman queueing network model [9], shown in Figure VII, can provide simple estimates of the time needed for a single matrix iteration. In the model of Figure VII, a task computes new XN components, broadcasts them across the bus, and *immediately* begins a new iteration at a processor. For standard iteration schemes, each task must wait until it receives new XN components broadcast by other tasks. This delay until all values are received can significantly increase the iteration time. Hence, the iteration times predicted by this model are optimistic; we present a pessimistic model in the next section. Note, however, that this model quite accurately predicts the iteration time of iteation schemes that do not wait for updated values before beginning the next iteration (i.e., *chaotic* iteration schemes).

A cursory examination shows that upper and lower bounds on the iteration time of the machine repairman model can be easily derived. In the best case, a task computes, attempts to use the broadcast bus before computing again, and finds the bus idle. If I_{lower}^{opt} denotes this lower bound on the optimistic iteration time, then

$$I_{lower}^{opt} = t(comp) + t(comm).$$

Conversely, a task may, in the worst case, compute, attempt to broadcast, and find all other $M - 1$ tasks also trying to broadcast. This upper bound on the optimistic iteration time is then

$$I_{upper}^{opt} = t(comp) + M\ t(comm).$$

If $t(comp)$ and $t(comm)$ are assumed to be the means of negative exponential distributions, standard queueing theoretic techniques [9] can be applied to the machine repairman model to determine the mean iteration time. Using standard formulae, the mean iteration time is

$$I_{exact}^{opt} = \frac{M\ t(comp)}{\sum_{i=0}^{M-1}(M - i)P_i}$$

where

$$P_i = \frac{\left[\frac{t(comm)}{t(comp)}\right]^i \frac{M!}{(M - i)!}}{\sum_{k=0}^{M}\left[\frac{t(comm)}{t(comp)}\right]^k \frac{M!}{(M - k)!}},$$

and P_i is the probability that i processors are waiting to use the broadcast bus.

Pessimistic Performance Model

Developing a more accurate performance model requires inclusion of the synchronization delays to receive broadcast values. The simplest such approach again requires the assumption of negative exponential distributions for $t(comp)$ and $t(comm)$ as well as the notion of blocking. This model, shown in Figure VIII, allows tasks to begin computing again only after *all* tasks have broadcast their values. Because not all tasks need data from all other tasks, this model gives pessimistic estimates of iteration time.

The queueing network of Figure VIII can be in several possible states. Each such state is defined by the number of tasks computing and the number blocked after using the broadcast bus. Transitions among the states are caused by tasks completing an iteration's computation or a broadcast. The rate these transactions occur is determined by the expected delays at the processors and the bus. If these states are represented by the tuple (i, j), where i is the number of tasks computing, and j is the number of tasks blocked, then Figure IX is the state transition diagram of the network.

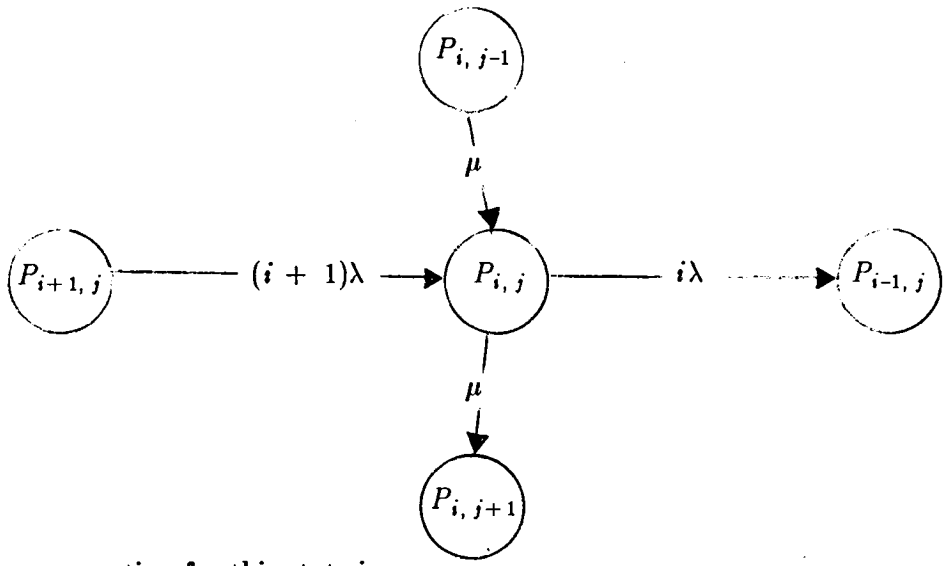
When this model is in equilibrium, the system will occupy each state with a particular probability, and the rate of transition into a given state must be the same as the transition rate from the state. These steady state probabilities are obtained by solving a linear system of state space balance equations obtained from the state diagram. For state $(M, 0)$, all tasks computing, the balance equation is

$$\mu P_{0,\,M-1} = M\lambda P_{M,\,0}$$

or

$$P_{M,\,0} = \frac{\mu P_{0,\,M-1}}{M\lambda}. \tag{12}$$

Now consider an arbitrary state (i, j):



The balance equation for this state is

$$(i + 1)\lambda P_{i+1,\,j} + \mu P_{i,\,j-1} = (i\lambda + \mu)P_{i,\,j},$$

and the state probability is

$$P_{i,\,j} = \frac{(i + 1)\lambda P_{i+1,\,j} + \mu P_{i,\,j-1}}{i\lambda + \mu}.$$

Notice that each state probability can be expressed using those states either above or to the left in the transition diagram. Hence, all state probabilities can (recursively) be expressed using $P_{M,\,0}$. As (12) shows, $P_{M,\,0}$ can in turn be expressed using $P_{0,\,M-1}$. Thus, knowing $P_{0,\,M-1}$ would permit easy calculation of all other state probabilities. But,

$$\sum P_{i,\,j} = 1$$

must hold since the $P_{i,\,j}$ are probabilities. So, $P_{0,\,M-1}$ is just the reciprocal of the sum of the $P_{i,\,j}$ coefficients. Solving for the state probabilities is then straightforward.

- Sum all probability coefficients and invert the sum; this is $P_{0,\,M-1}$.

- Solve for the other state probabilities by multiplying their coefficients by $P_{0,\,M-1}$.

Now $P_{0,\,M-1}$ is the state corresponding to all tasks but one blocked waiting for the final task to complete its broadcast. The network *must* pass through this state each iteration. Consequently, the mean iteration rate is

$$\mu P_{0,\,M-1} = \frac{P_{0,\,M-1}}{t(comm)},$$

and the mean time to perform one matrix iteration is

$$I^{pes} = \frac{1}{\mu P_{0,\,M-1}} = \frac{t(comm)}{P_{0,\,M-1}}$$

for the pessimistic performance model.

Comparison of Performance Models

Figure X shows the optimistic and pessimistic iteration time models for one set of matrix parameters. (I^{opt}_{lower}, the lower bound on optimistic iteration time, is omitted because it was indistinguishable from I^{opt}_{exact}.) As the number of X_TASKs increases, the delays caused by the presence of the blocking queue become pronounced. Because testing for global convergence is essentially a global synchronization, it is not surprising that the presence of additional tasks results in longer iteration times.

As noted earlier, the optimistic iteration time model assumes each X_TASK needs *no XN* components from other X_TASKs. Similarly, the pessimistic model assumes each X_TASK needs *XN* components from *all* other X_TASKs. Clearly, the actual iteration time must lie between these two bounds.

The precise form of the actual iteration time curve is a function of matrix sparsity and structure. For example, a tridiagonal matrix divided into contiguous partitions of size greater than three would result in X_TASKs needing *XN* components from only two

other X_TASKs. Thus, one would expect the actual iteration time to be near that predicted by the optimistic model. Similarly, a set of X_TASKs, each containing a dense matrix row, would need XN components from all other X_TASKs, and one would expect the pessimistic model to be more accurate.

In general, the selection of either the optimistic or pessimistic model depends on the matrix structure and partitioning. Although the optimistic models have the advantage of efficient evaluation, the pessimistic model should be used in any conservative designs.

Parametric Performance Study

Having derived iteration time models, we undertook a parametric study of performance. Selected results of this study are discussed below. Although we present results only for the pessimistic model, similar results hold for the optimistic case.

Matrix Size

Figure XI shows iteration time as a function of the number of matrix partitions (X_TASKs). In each case, an optimal level of parallelism, balancing computation requirements against saturation of the broadcast bus, exists. As the number of partitions grows, communication and synchronization delays increase whereas task computation decreases. Similar behavior is seen in Figure V for our analytic iteration time model, suggesting both models capture the salient aspects of delays.

Communication Costs

Figure XII shows the effects on iteration time of varying bus speeds. The lowest curves correspond to bus speeds near those of the processors, such as one might find in a tightly coupled parallel system. Interestingly, the curves for $C_t = 1.0$ and 0.1 are nearly identical, suggesting that, for sufficiently fast buses, computation and synchronization

delays dominate.

The upper curves in Figure XII reflect speed differentials more appropriate for local area networks. For large ratios of communication time to computation time, the single bus is an obvious performance bottleneck, hence the nearly horizontal performance curves for large numbers of X_TASKs. In these cases, a modest number of processors suffices to realize most of the potential performance gains.

It is interesting to compare Figure XII, the pessimistic iteration time model for the bus, with an equivalent performance prediction obtained from iteration time model initially derived. In the first model, this is the $f(M) = M$ curve in Figure III. The pessimistic bus model predicts a horizontal performance asymptote while the initial model predicts increasing iteration time. The reason for this difference is quite simple. Recall that the initial model included a controlling C_TASK that tested for global convergence. In contrast, the bus model includes distributed determination of convergence *in parallel* with computation. This overlap becomes increasingly important as the number of tasks grows. Indeed, in the non-overlapped initial model, this synchronization is the dominant delay for large numbers of tasks.

Matrix Sparsity

Finally, Figure XIII shows the effect on iteration time of matrix sparsity; the lowest curve corresponds to greatest sparsity. As with Figure IV, the optimum level of parallelism increases with the number of non-zero elements.

Summary

We have derived the expected inter-task data transfer and defined an execution time model for predicting the parallel iteration time during solution of random, sparse linear systems of algebraic equations. Although this performance model clearly shows

that solution time can be reduced by partitioning the computation into parallel subtasks, the optimum number of partitions is very dependent on synchronization and communication costs.

We also considered the mapping of partitions onto the processors of a parallel architecture. We observed that optimal scheduling was NP-complete and, because linear systems can be solved sequentially in polynomial time, infeasible. Moreover, we showed that scheduling heuristics would likely not be cost effective.

Finally, we considered one parallel architecture where scheduling is not needed, the broadcast bus. Although scheduling can be avoided on other architectures, notably those based on multistage networks, the broadcast bus is an alternative for modest numbers of processors. Moreover, local area networks have made bus systems (e.g., ethernet) widely available.

Optimistic and pessimistic performance models of the broadcast bus architecture for solving linear systems were derived and analyzed. We concluded with an investigation of the performance of the bus architecture as a function of matrix sparsity and the costs of communication, synchronization, and computation. This investigation showed that a broadcast bus architecture *can* effectively reduce the expected computation time for solving sparse linear systems.

Acknowledgments

We are particularly indebted to Loyce Adams, Piyush Mehrotra, Terry Pratt, John van Rosendale and Robert Voigt, our colleagues in the XFEM Research group at ICASE, NASA Langley Research Center, for many helpful discussions.

References

[1] L. Adams, "Iterative Algorithms for Large Sparse Linear Systems on Parallel Computers," *NASA CR No. 166027,* NASA Langley Research Center, Hampton, Virginia, November 1982 (also published as a Ph.D. dissertation, University of Virginia).

[2] L. Adams and R. Voigt, "Design, Development, and Use of the Finite Element Machine," *NASA CR No. 172250,* also ICASE Report 83-56, NASA Langley Research Center, Hampton, Virginia, October 1983.

[3] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

[4] H. Amano, T. Yoshida, and H. Aiso, "$(SM)^2$: Sparse Matrix Solving Machine," *The 10th Annual International Symposium on Computer Architecture, ACM Sigarch Newsletter,* Vol. 11, No. 3, June 1983, pp. 213-220.

[5] D. A. Calahan, "Randomly Sparse Equation Solution by Loopless Code Generation on the CRAY-1," *Technical Report 162,* Systems Engineering Laboratory, University of Michigan, May 1982.

[6] J. G. Lewis and W. G. Poole, Jr., "Ordering Algorithms Applied to Sparse Matrices in Electrical Power Problems," *Electrical Power Problems: The Mathematical Challenge,* A. M. Erisman, K. W. Neeves, and M. H. Dwarakanath (eds.), SIAM Publications, Philadelphia, 1980.

[7] D. A. Reed and M. L. Patrick, "A Model of Asynchronous Iterative Algorithms for Solving Large, Sparse, Linear Systems," *Proceedings of the 1984 International Conference on Parallel Processing,* Bellaire, Michigan, pp. 402-409.

[8] A. Gottlieb *et al,* "The NYU Ultra Computer - Designing an MIMD Shared Memory Parallel Computer," *IEEE Transactions on Computers,* Vol. C-32, pp. 175-189, February 1983.

[9] L. Kleinrock, *Queueing Systems,* Vol. I, Wiley-Interscience, New York, 1975.

Table I Matrix Analysis Notation

Quantity	*Definition*
A	arbitrary $N \times N$ sparse matrix
b_j	initial row of partition j
c	arbitrary constant N-vector
$C_b(M)$	transmission time for a boolean as a function of the number of partitions
C_p	computation time for arithmetic operations
C_s	startup time for data transmission
$C_t(M)$	transmission time for one datum as a function of the number of partitions
e_j	final row of partition j
M	number of matrix partitions
N	matrix dimension
$P(N, Z, q)$	probability that a matrix element is non-zero
$P_{Tr}(k, j)$	probability that partition k transfers data to partition j
q	probability that a matrix element is non-zero given that it is not one of the Z known non-zero elements
S	fixed partition width
$t_j(comm)$	communication time for one iteration at partition j
$t_j(comp)$	computation time for one iteration at partition j
$Tr(k, j)$	expected data transfer from partition k to j
x	N-vector of unknowns
Z	number of known non-zero elements in a row

Table II Computational Complexity of X_TASKs

Loop Cost	*Statement Cost*	*Statement*
(1)	$2C_p$	FOR I := B [J] to E [J] DO
		BEGIN
	C_p	SUM := 0;
(2)	$2C_p$	FOR K := L [J] to U [J] DO
	$6C_p$	SUM := SUM +
		ANZ [K] * X0 [COLSUB [K]]†
	$4C_p$	XN [I] := SUM + C [I]
		END;
	C_p	CONVERGED := TRUE;
(3)	$2C_p$	FOR I := B [J] to E [J] DO
		BEGIN
	$5C_p$	IF ABS (XN [I] - X0 [I]) > EPS THEN
	C_p	CONVERGED := FALSE
	$3C_p$	X0 [I] := XN [I]
		END;

(1): $(e_j - b_j + 1)C_p\left[6NP(N, Z, q) + 7\right] + 2C_p$

(2): $6C_pNP(N, Z, q) + 2C_p$

(3): $(e_j - b_j + 1)9C_p + 3C_p$

†ANZ and COLSUB are vectors of the non-zero elements of A and the corresponding column subscripts, respectively. L [J] and U [J] denote the beginning and ending indices of components of these vectors belonging to partition J.

Table III Scheduling and Performance Notation

Quantity	*Definition*
Br	amount of data broadcast by a task
$ComType$	type of interprocessor communication mechanism
I^{opt}_{lower}	lower bound on optimistic prediction of iteration time
I^{opt}_{upper}	upper bound on optimistic prediction of iteration time
I^{pes}	pessimistic prediction of iteration time
Mem_i	memory required by the *ith* partition of assignment P
M_{size}	processor memory size
p	maximum number of processors on bus
P	a possible assignment of rows to partitions
Pset(p)	set of all possible row assignments to no more than p processors
P_i	probability that i tasks are waiting to use the bus
$P_{i,j}$	probability of being in state (i, j)
P_{Br}	probability of broadcast by a task
λ	reciprocal of mean task computation time, $t(comp)$
μ	reciprocal of mean task communication time, $t(comm)$

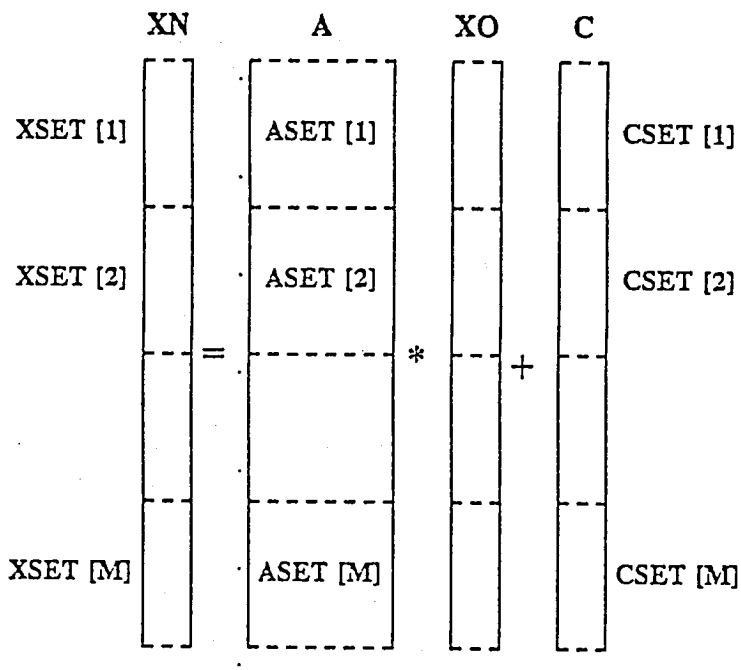


Figure I Partitioning of a linear system

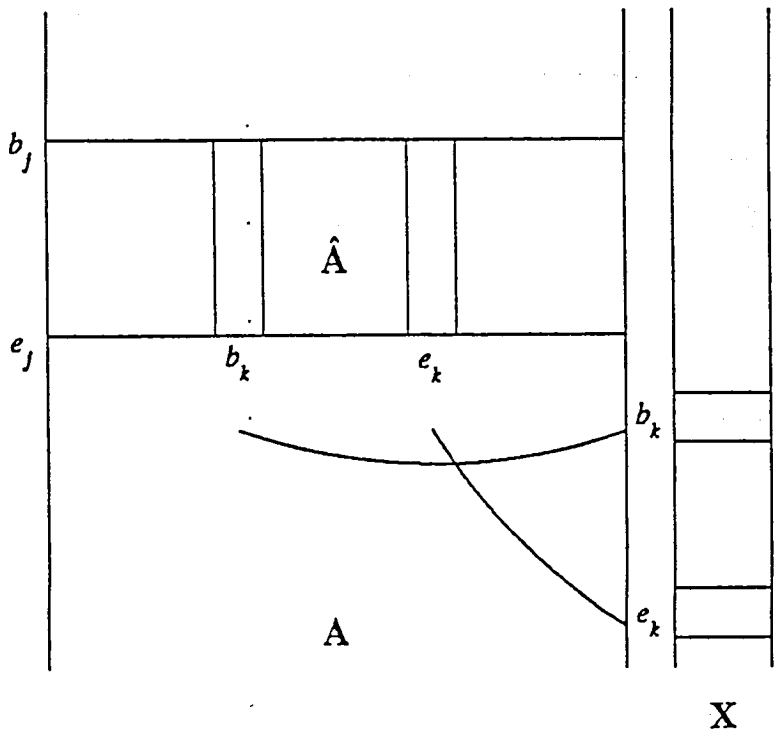


Figure II
Data transfer between multiple row partitions

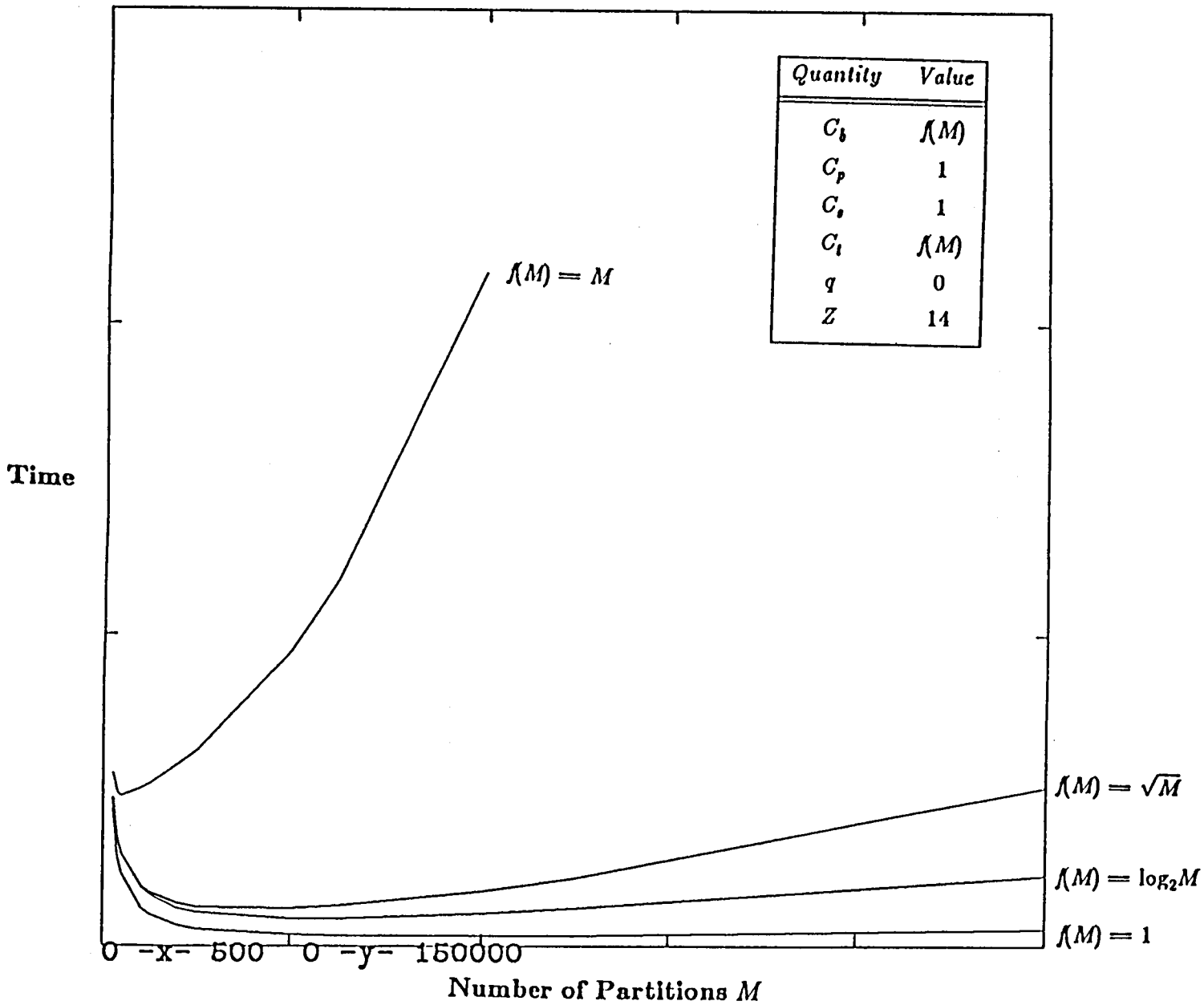


Figure III Execution time for $N = 1000$

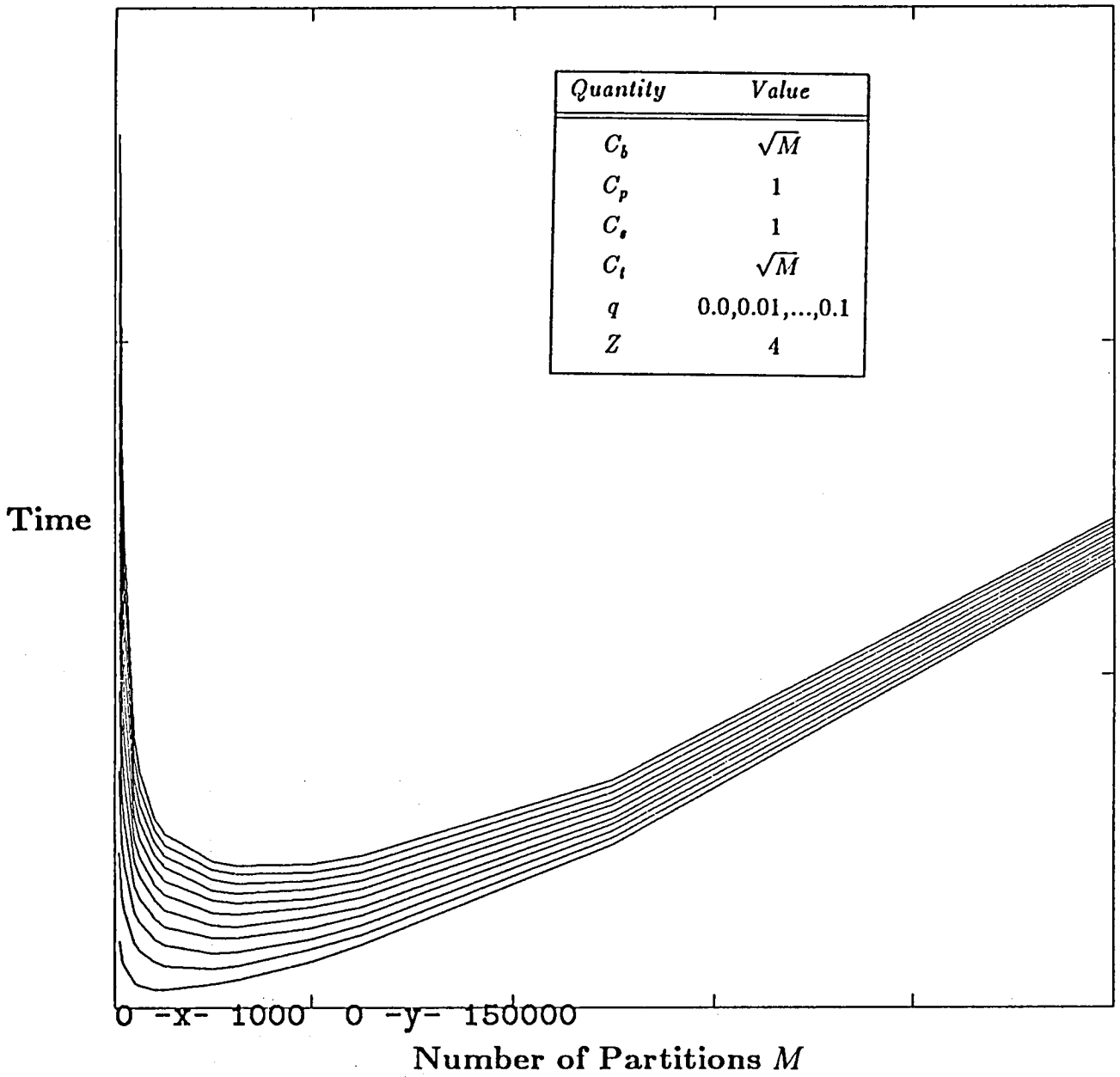


Figure IV Execution time for $N = 1000$

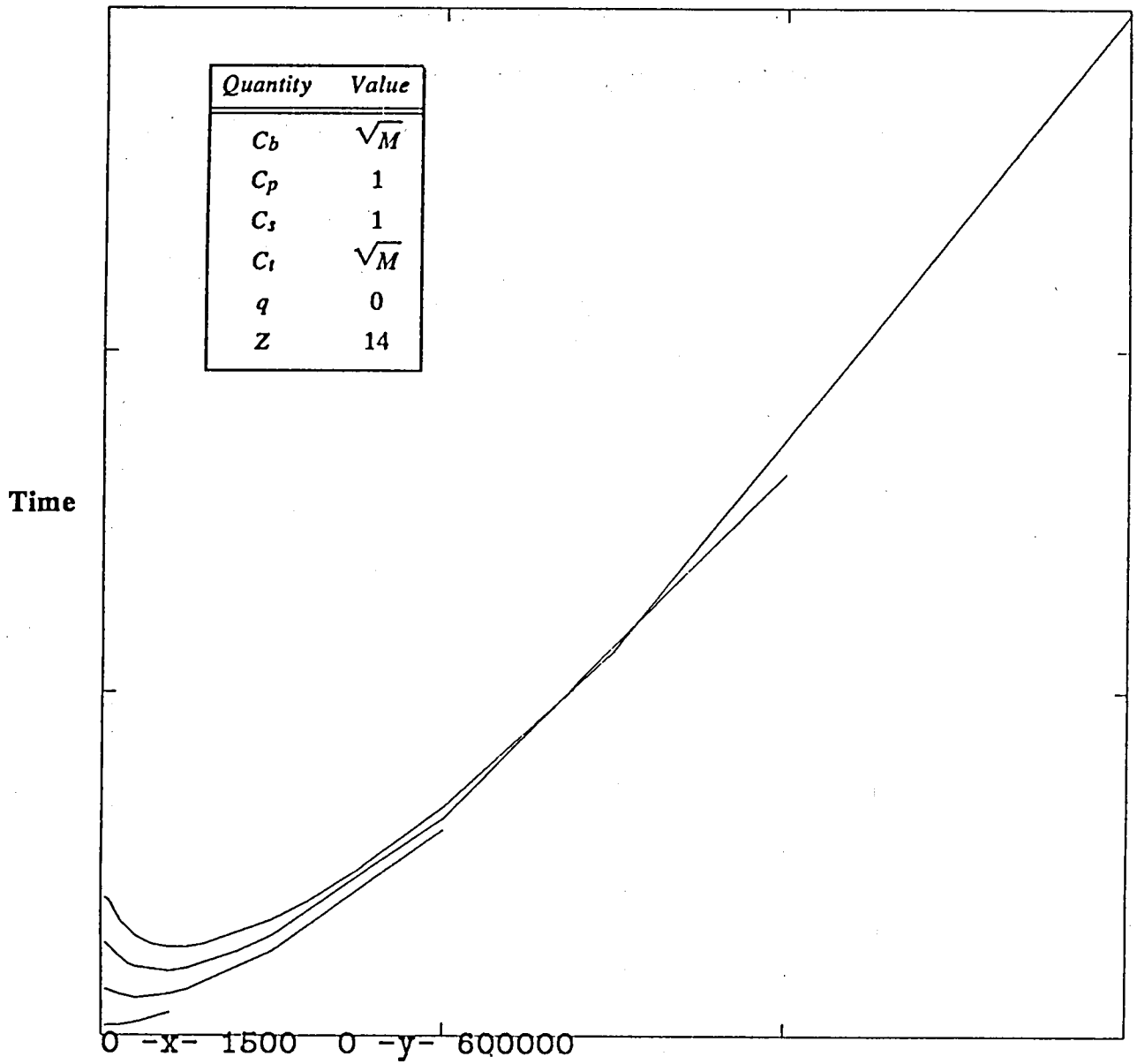
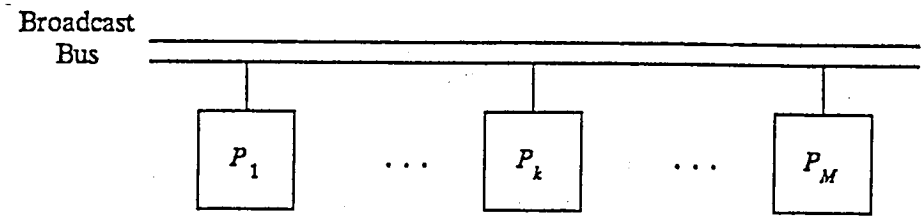


Figure V Execution time for $N = 100, 500, 1000, 1500$



Broadcast Bus Network

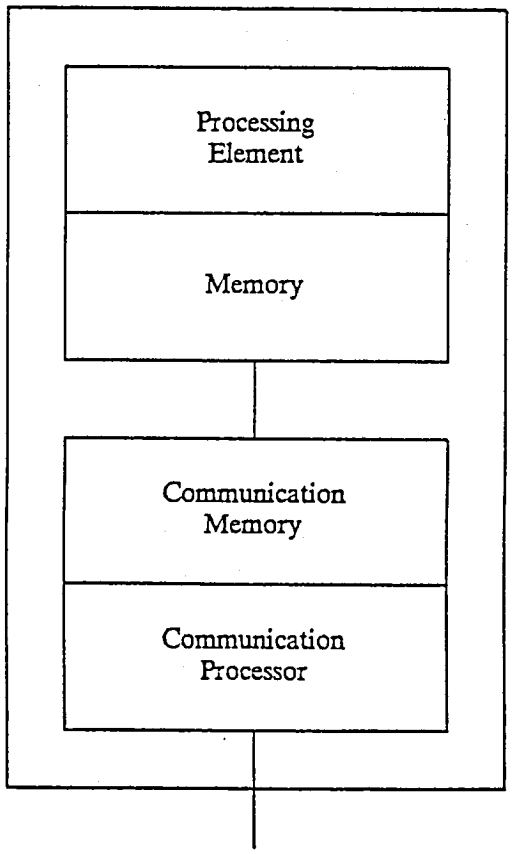


Figure VI
Network and Processor for Broadcast Bus

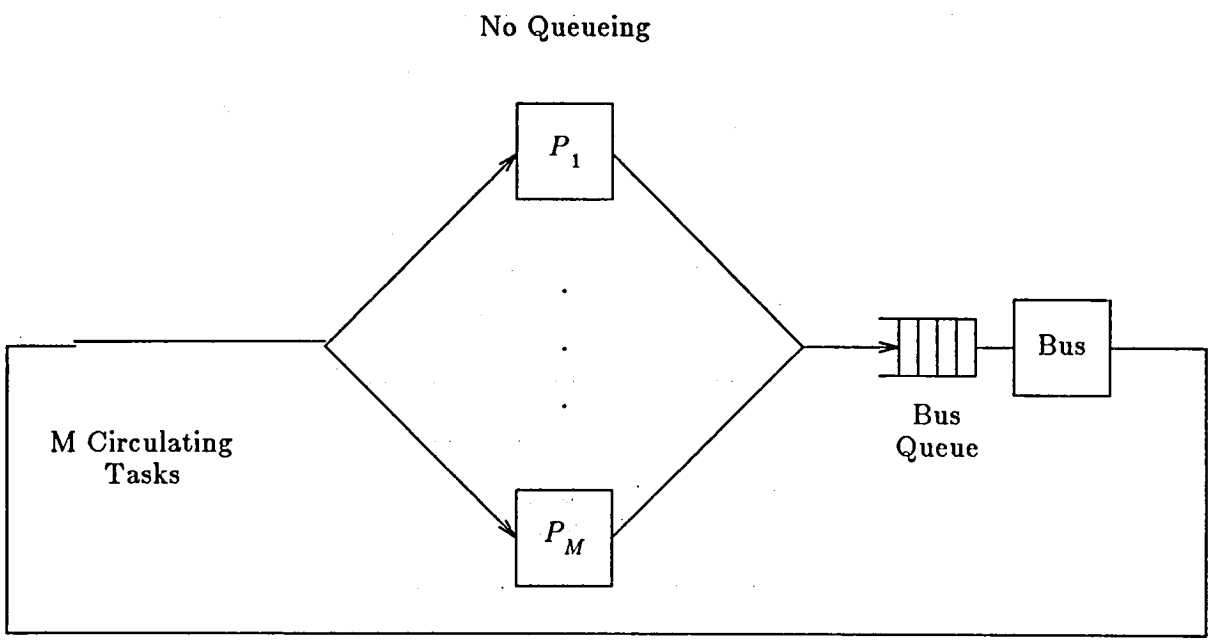


Figure VII Optimistic Queueing Model of Matrix Iteration

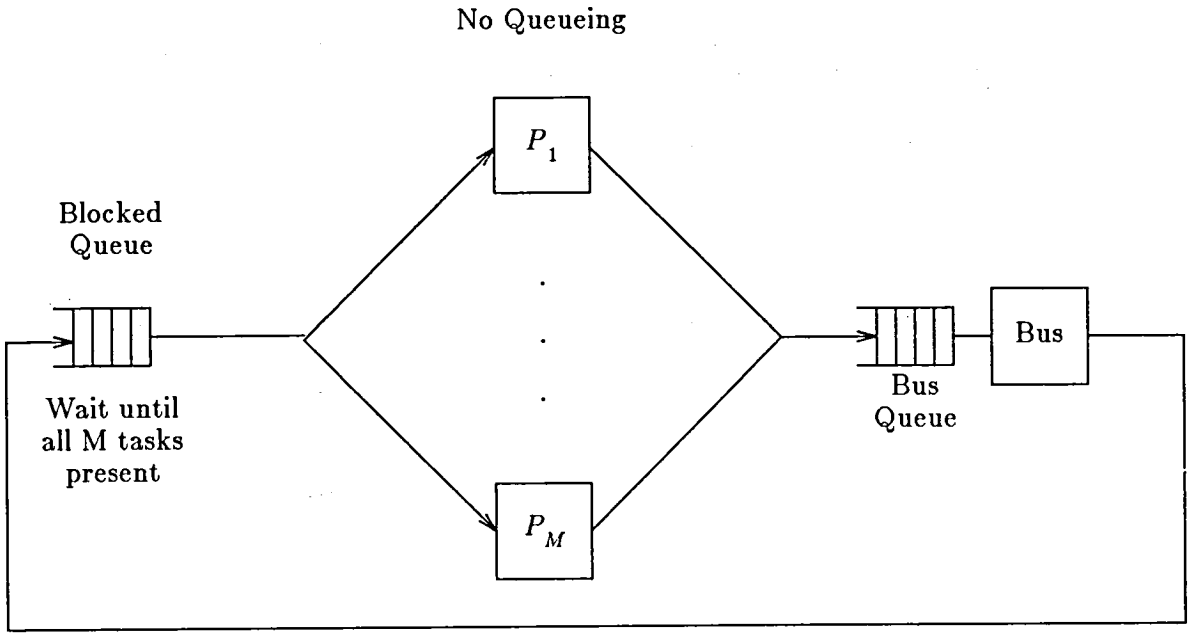


Figure VIII Pessimistic Queueing Model of Matrix Iteration

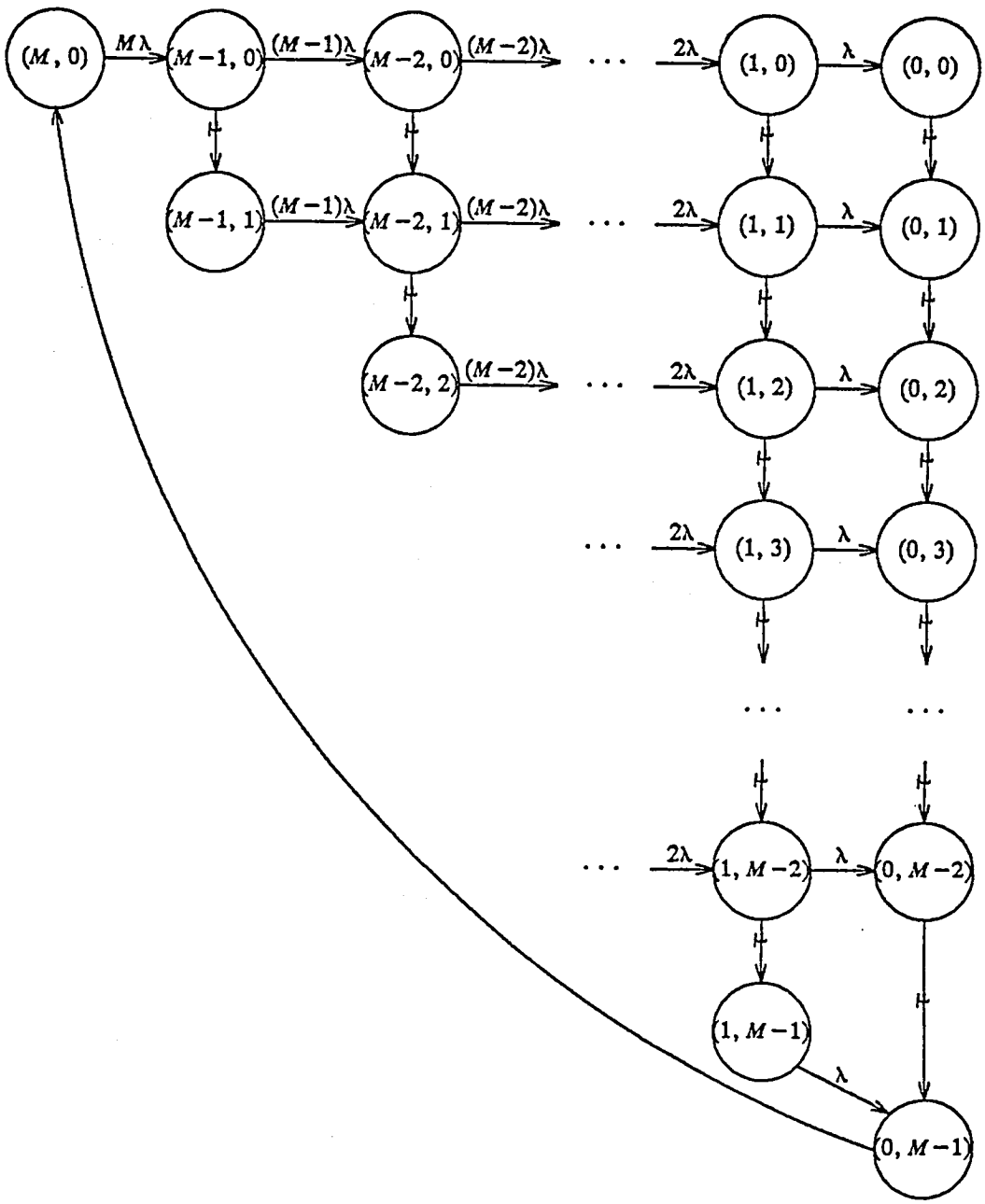


Figure IX
State Space Diagram for Pessimistic Queueing Model

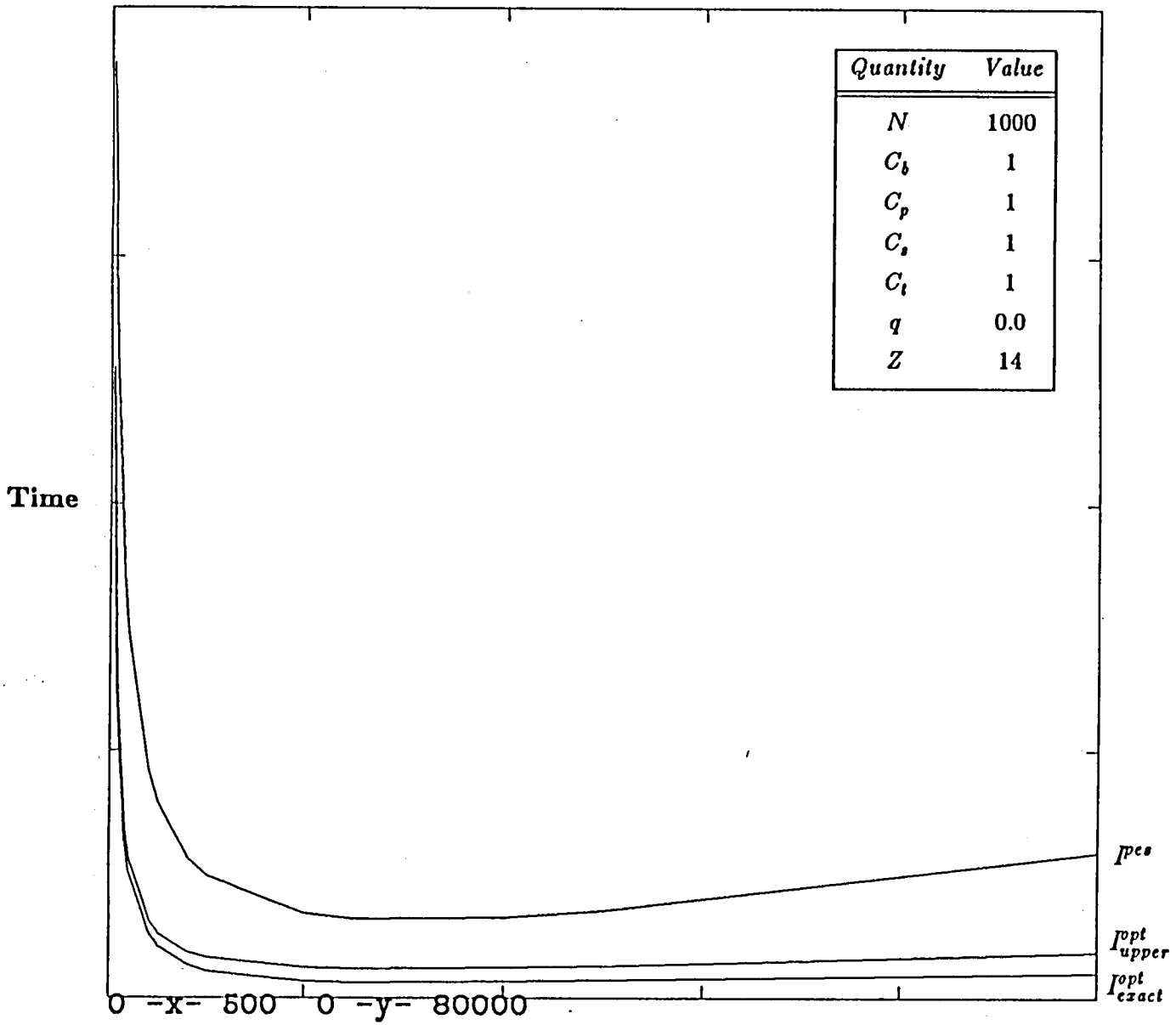


Figure X Comparison of Iteration Time Models

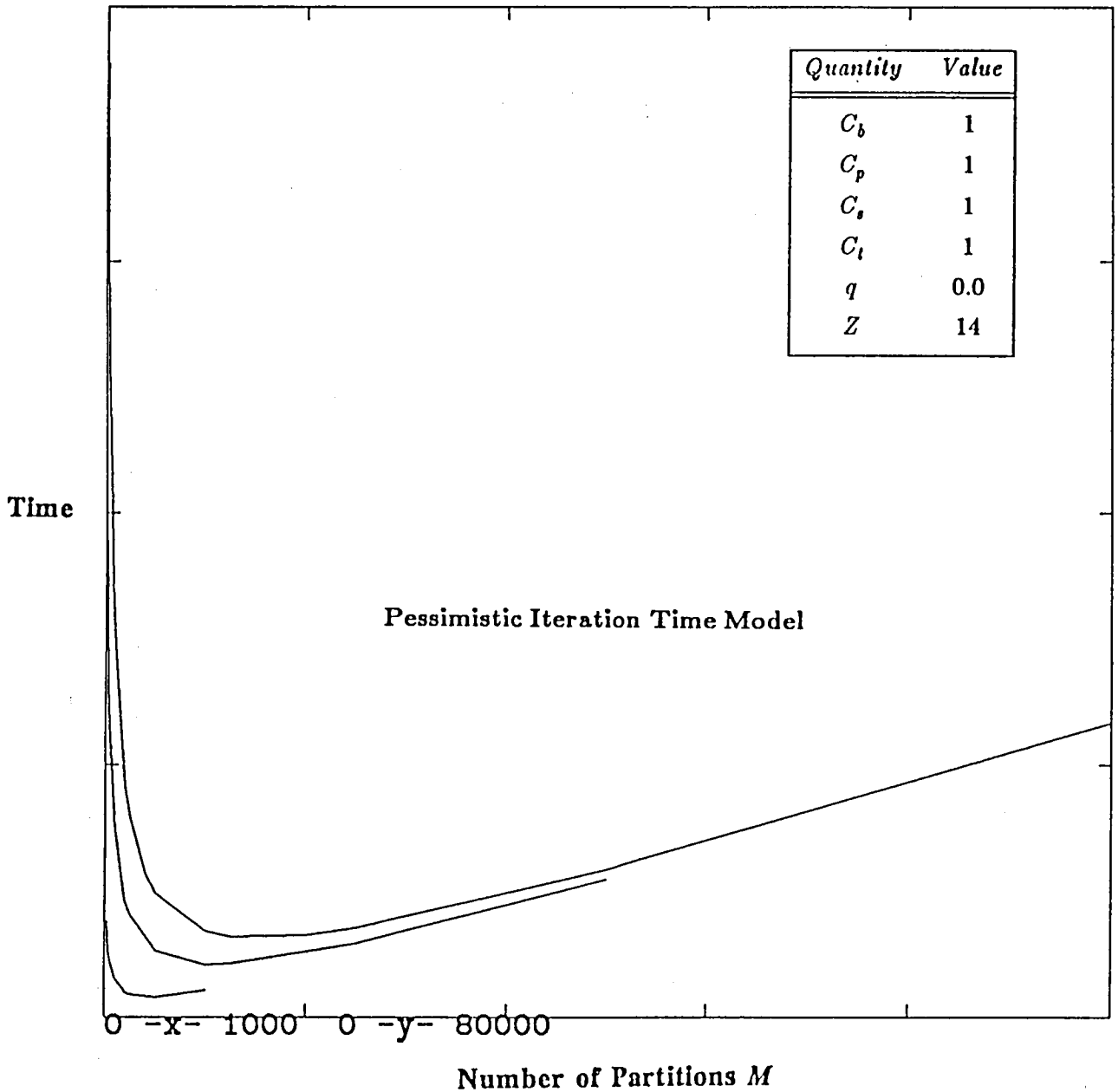


Figure XI Iteration Time for $N = 100, 500, 1000$

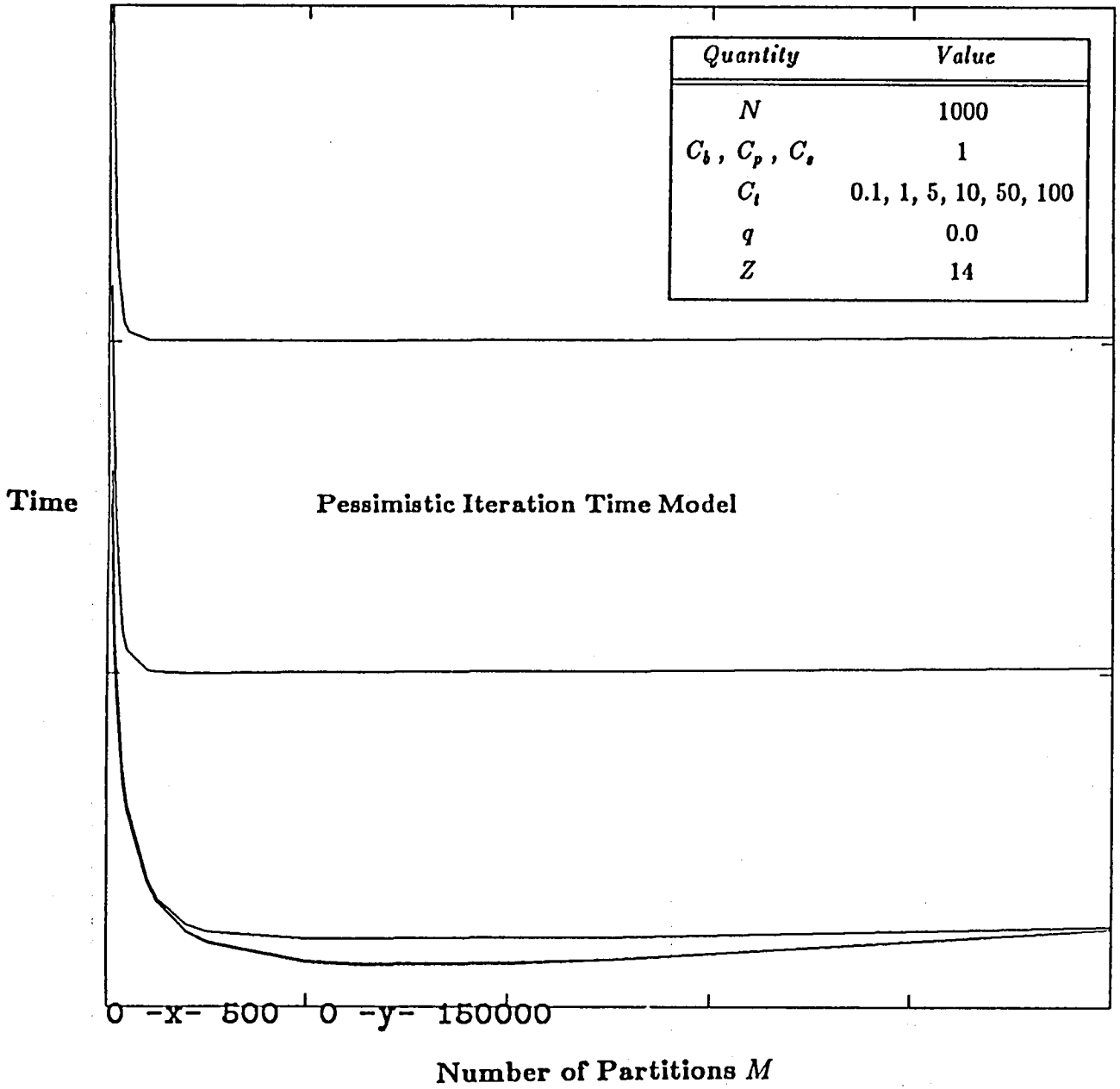


Figure XII Iteration Time for Varying Transmission Costs

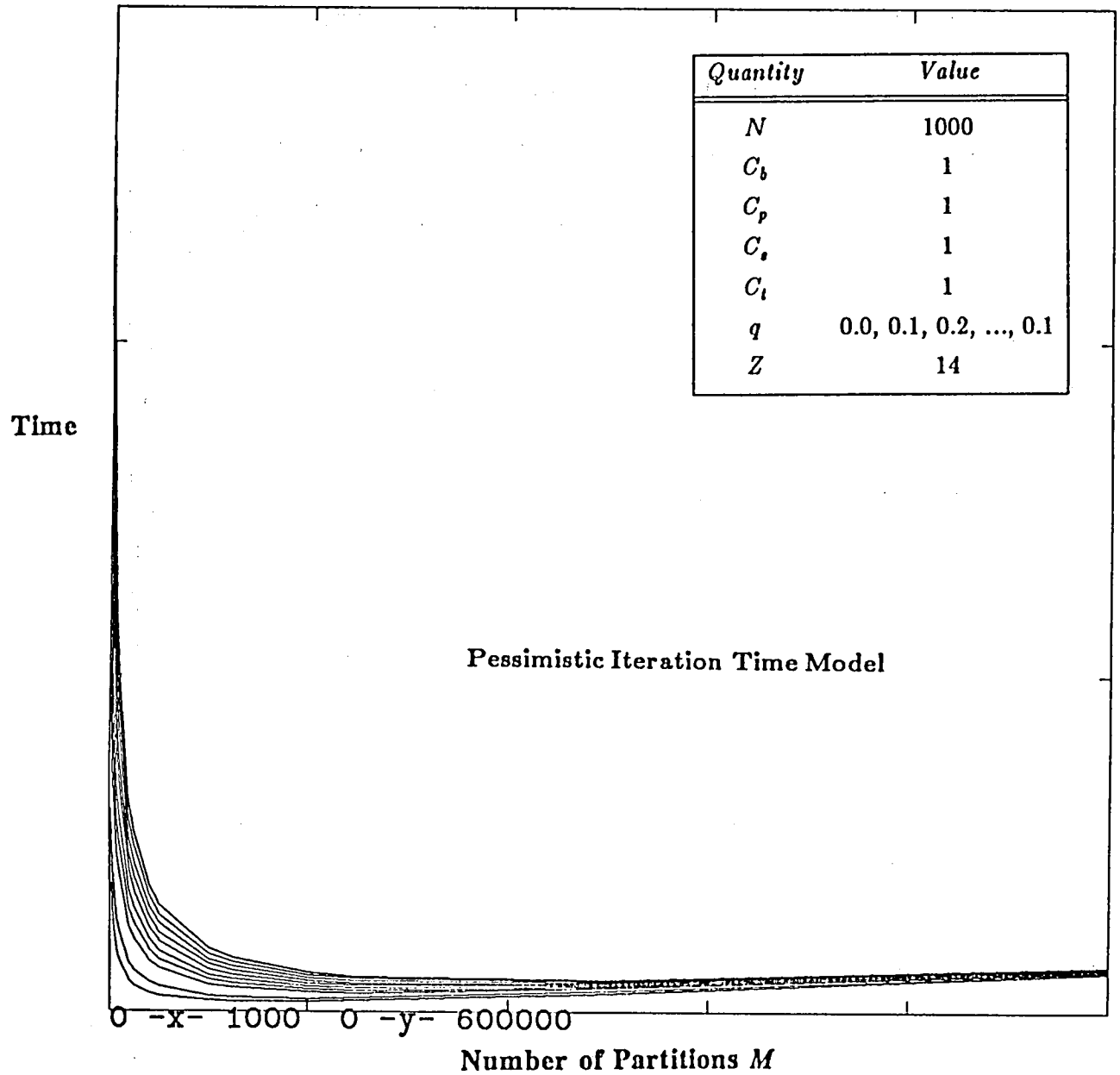


Figure XIII Iteration Time for Varying Matrix Sparsity

1. Report No. NASA CR-172457 ICASE Report No. 84-35	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Parallel, Iterative Solution of Sparse Linear Systems: Models and Architectures		5. Report Date August 1984 6. Performing Organization Code
7. Author(s) Daniel A. Reed and Merrell L. Patrick		8. Performing Organization Report No. 84-35 10. Work Unit No.
9. Performing Organization Name and Address Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23665		11. Contract or Grant No. NAS1-17070 NAS1-17130 13. Type of Report and Period Covered Contractor Report
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546		14. Sponsoring Agency Code 505-31-83-01

15. Supplementary Notes

Langley Technical Monitor: J. C. South, Jr.
Final Report

16. Abstract

Solving large, sparse, linear systems of equations is a fundamental problem in large scale scientific and engineering computation. A model of a general class of asychronous, iterative solution methods for linear systems is developed. In the model, the system is solved by creating several cooperating tasks that each compute a portion of the solution vector. A data transfer model predicting both the probability that data must be transferred between two tasks and the amount of data to be transferred is presented. This model is used to derive an execution time model for predicting parallel execution time and an optimal number of tasks given the dimension and sparsity of the coefficient matrix and the costs of computation, synchronization, and communication.

The suitability of different parallel architectures for solving randomly sparse linear systems is discussed. Based on the complexity of task scheduling, one parallel architecture, based on a broadcast bus, is presented and analyzed.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
random sparse linear systems, parallel iterative methods, queuing network models, interprocessor communication networks, task allocation, bus architecture, optimal partitioning.	62 - Computer Systems 64 - Numerical Analysis Unclassified - Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	46	A03

For sale by the National Technical Information Service, Springfield, Virginia 22161

NASA-Langley, 1984