LOOSELY COUPLED PARALLEL SYSTEMS-A NEW APPROACH TO SOLVING MANAGEMENT SCIENCE PROBLEMS

C.Y. Teresa Lam
Department of Industrial
and Operations Engineering
The University of Michigan
Ann Arbor, MI 48109

John P. Lehoczky
Department of Statistics
Carnegie Mellon University
Pittsburgh, PA 15213

Technical Report 89-25

August 1989

Revised August 1990

Loosely Coupled Parallel Systems - A New Approach to Solving Management Science Problems

C. Y. Teresa Lam*
Department of Industrial and Operations Engineering
The University of Michigan
Ann Arbor, MI 48109

John P. Lehoczky*
Department of Statistics
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This paper addresses the modelling and performance evaluation of a loosely coupled parallel computer system. This is a new type of computer system which can be used to solve many important problems in operations research, management science and statistics. The system provides affordable parallel computation in that it can be created on a network of workstations or even microcomputers. No only do these systems offer substantial, yet affordable, computing power, they raise interesting modelling and performance evaluation problems.

This paper describes loosely coupled parallel computing systems and presents many examples of their application. The system is modelled by the superposition of renewal processes, and some associated performance evaluation problems are studied. Data from the execution of large problems on a loosely coupled system is presented and compared with models. The models are shown to fit very well to the data.

(LOOSELY COUPLED PARALLEL SYSTEMS; DATA FLOW SYSTEM; PERFORMANCE EVALUATION; TRADEOFFS BETWEEN OVERHEAD AND BLOCK SIZE)

*Research Supported in part by a grant from the National Science Foundation DMS-87-02537.

1 Introduction

In recent years, there has been a profound increase in the sophistication of affordable computer technology available to most corporations and universities. Microcomputers now have the computing power of the mainframe computers of five years ago, and scientific workstations have become widely affordable. Only sophisticated parallel computers and supercomputers are not generally available for large scale operations research, management science or statistics problems. This increase in computing power has allowed researchers and analysts to address problems of ever increasing size and complexity, and this means that the available computing power never seems to be adequate.

This paper discusses loosely coupled computer systems consisting of networks of microcomputers or workstations. Such networks can be used to create a large grained parallel computer system and, therefore, can be used to solve a large class of problems arising in operations research, management science and statistics. Consequently, this approach to computing offers the promise of making parallel computing widely available for a wide range of important problems. Moreover, loosely coupled computer systems also raise very interesting questions in stochastic modelling, performance evaluation and performance optimization. It is these latter issues which are the focus of this paper.

Loosely coupled computer systems lie toward one end of the spectrum of distributed computer systems and are sometimes referred to as workstation-LANs (Local Area Network). Each of the workstations has its own private memory, and communication among processors is done exclusively over the LAN. Such systems permit parallel processing on a single problem; however, it must be *coarse grained parallelism*, that is the amount of computing done by a single processor between communication steps is relatively large. These systems are in marked contrast to tightly coupled systems (e.g. systems with shared memory multiprocessors) for which *fine grained parallelism* with more frequent interprocessor communication is practical. While tightly coupled systems are suitable for both coarse and fine grained parallelism, these systems do not yet represent an easily affordable technology, whereas loosely coupled systems of workstations are increasingly inexpensive and commonplace. The survey paper by Bal, Steiner and Tanenbaum (1989) offers an introduction to distributed computing systems and programming languages which are suitable for writing distributed programs. The interested reader is strongly encourage to read the survey article of Carriero and Gelenter (1989) which provides an excellent introduction to the various concepts of parallelism and to writing parallel programs.

The loosely coupled approach to parallel computing applies to problems that can be decomposed into smaller subproblems, each of which can be solved independently and concurrently. A surprisingly large number of important problems fit into this framework including problems ranging from combinatorial optimization to computing the equilibrium distribution of a large Markov chain to doing multivariate time series analysis and forecasting. A discussion of these and other examples is given in Section 3.

A loosely coupled computer system utilizes one of the processors as a parent, master or coordinating processor, while the other processors in the network are child or servant processors. The master processor takes the large problem and divides it into pieces or subproblems. These subproblems are sent over the network to the child processors, and these processors execute the subproblems along with any other work they may be doing. Upon completion of a subproblem, the child sends the results over the network back to the master. The master receives the results of the subproblems and sends more subproblems for solution back to the child processors. This process continues until the original large problem is solved. For this system to be practical, the communication time cost must be kept to a minimum. This means that the subproblems sent to the child processors must be relatively large so that the time to communicate the problem and its solution are a small fraction of the computation time required for a subproblem, that is the parallelism is coarse grained. In addition, the child processors must be able to solve a subproblem independently of the other subproblems. Many problems have this form; however, they may require the subproblems to be of small size which will make the communication costs too high.

As the above discussion suggests, loosely coupled computing systems are not only an important new approach to offering affordable, but powerful computing, they are also interesting systems to model and optimize. One must develop models for their behavior and optimize system performance given consideration to quantities such as the time spent in communication, the size of the subproblems and the load on each of the child processors.

The requirement for the implementation of a loosely coupled system is a collection of microcomputers or workstations connected by a local area network. The computers must have an operating system capable of interprocessor communication. The network used for the experiments reported in this paper was the Carnegie Mellon Department of Statistics network of Microvax workstations using the VMS operating system and connected by Decnet. Nevertheless, the concepts discussed in this paper apply to more modest networks such as those consisting of IBM PS/2 processors using OS-2 or Macintosh II processors. The approach can be applied to a wide range of problems, but the communication overhead must be kept relatively small to make this approach attractive.

This paper is organized as follows: In Section 2, the loosely coupled parallel computer system developed by Eddy and Schervish (1986) is described. A description of some of the problems in operations research and statistics that can be implemented by such a system is given in Section 3. Sections 4 and 5 develop a stochastic model for loosely coupled computer systems, and we focus on minimizing the expected completion time of a task. Numerical examples are given in Section 6 to illustrate the method and system performance. Simulations are carried out in Section 7 to validate exponential approximations used in the numerical examples. Finally, Section 8 of this paper discusses areas of future research.

2 Description of a Loosely Coupled System

The Department of Statistics of Carnegie Mellon University has a cluster of workstations consisting of Microvax II and III processors which communicate via DECnet over Ethernet cables. Eddy and Schervish (1986) developed a set of FORTRAN subroutines which allowed for the creation of processes on each of these machines which can communicate with each other. Eddy and Schervish (1986) presented the results of using this loosely coupled system to solve a discrete finite inference problem. They were able to divide the large problem up efficiently among the other processors in the network and reported dramatic reductions in elapsed computation times. This example is discussed briefly in the next section and more extensively in Section 6. The system described by Eddy and Schervish is general enough to be able to handle any problem in which the task is decomposable into parts which can be performed independently of each other. The system they created works as follows. There are essentially two programs, and there are two types of processes on the network. One process will be called the *parent*, while the others will be called the *child.* One of the two programs is run by the parent, and the other program is run by all of the children simultaneously. Both programs are required to perform interprocess communication, which will be described in more detail below. The parent program divides the task into decomposable parts which are called messages and sends the messages to the children via the communication network. Each message consists of sufficient information for the child to execute or carry out the required calculations. When a child finishes its calculations, it sends its results back to the parent. The parent is responsible for combining the individual results in an appropriate fashion. This system is an example of a data flow system as described by O'Leary and Stewart (1985). The concept of a data flow algorithm is that, once the program has started, the flow of data controls the distribution of work and the program itself. In the Eddy and Schervish system, after the child begins its work, program control is handled by the return of results from the child which initiates the subsequent sending of the next message to it.

Network communication

Communication between parent and child is done over DECnet. There are two communication channels, one called the *mailbox* channel and the other called the *data* channel. The mailbox channel is used to keep track of the network status. For example, when each child is created, the parent sends no data until it receives a message in its mailbox saying that the child has been created. When a child dies, the parent is sent a mailbox message saying the child is dead, and the parent must reassign the task on which the child was working to a different processor. The data channel is used by the parent to send its messages (data) to the child and by the child to return the results of its processing.

Asynchronous communication

Because the program control is data driven, the parent only deals with data arriving from a child when it arrives. That is, the parent does not wait for messages or data, but rather issues asynchronous read requests and then goes on with whatever other work it may be performing. For example, upon starting a child process, the parent opens the data and mailbox channels and issues an asynchronous read for a mailbox message. Then it goes to the next child and does the same until it has done this for all available children or is interrupted by the arrival of a mailbox message. When it reads a mailbox message saying that a child is alive, it issues an asychronous read for a mailbox message, sends data to the child, and then issues an asynchronous read for data. When the parent reads the results, it sends more data and issues another asynchronous read command for more data. The child on the other hand, operates synchronously by reading data from the parent, performing its computations, sending the results back to the parent, and waiting for the next set of data. The child may perform other tasks while waiting for the next message. The crucial features of this system are that

1. the parent goes back to whatever it was doing after it issues an asychronous read request and
2. when an asynchronous read request is answered, the parent is interrupted from what it was doing and deals with the message it received. (The one exception to this is if the parent is already reading the answer to an asynchronous read when another one is also answered, the second and all subsequent answers queue up and are dealt with in order of arrival).

The parent process

Because the answers to asynchronous reads interrupt the parent and begin execution of a separate set of code, they behave like subprocesses. In fact, the program flow following one of these asynchronous answers is completely separate from the basic parent program. The basic parent program consists solely of the following:

1. Initialize with input data.
2. Loop through the children one at a time.
   - Open a link, if it is not currently open.
   - Issue an asynchronous read request for mailbox message.
3. Wait some fixed amount of time.
4. Return to step 2.

All of the data handling is done by the data subprocess described below. Each subprocess is initiated when a read is answered by one of the children. Hence, the subprocess is associated with that child for the duration of its existence.

The mailbox subprocess

The first thing the mailbox subprocess does is check to see if the message is a birth message or a death message. A birth message means that the child is alive. In this case, data is sent to the child, and an asynchronous read on the data channel is issued for the results. A death message means that the child is dead, and must be removed from the list of living children. In addition, if the child had been working on a set of data, the data must be requeued for delivery to another child at a later time. If, on the other hand, the child is dead because the parent killed it (when there is no data left to be sent), we need only remove it from the list of living children. When the last living child dies and there are no more data to be sent, results are summarized and execution ceases.

The data subprocess

The data subprocess begins when the asynchronous read for the first set of results issued by the mailbox subprocess is answered. At that time, the results are accumulated. If there are no more data, a special data set is sent to the child which causes it to cease execution and send a death message back the parent on the mailbox channel. If there is still data to be sent, the child receives the next packet of data. The subprocess then issues an asynchronous read for the results and quits. Notice that this last part of the subprocess is identical to part of the mailbox subprocess. In fact, these two subprocesses use the same code (which is reentrant for this purpose). As pointed out in Schervish (1988), it is possible to implement a data-flow system on any network of processors that supports some simple communication primitives, and nearly every computer installation has a network of processors. Programming a data-flow system is simple, it is not necessary to have any special purpose languages or special understanding to program the system. Similar systems can therefore be developed on other networks of processors, including networks of microcomputers running operating systems such as OS-2. Besides the system of loosely coupled processors described above, there are many other distributed parallel processing systems. For example, Whiteside and Leichter (1988) implemented a distributed parallel processing system based on the "Linda" programming constructs on a local area network of computers. Their system allows a single application program to utilize many machines on the network simultaneously. Gardner et. al. (1986) described a library of utilities that support distributed concurrent computing on a local area network of computers. The reader should also consult the article by Carriero and Gelernter (1989) described earlier.

3 Examples

There are many examples of numerical algorithms in which calculations can be done in parallel using the software system described in Section 2. These include problems in statistics, management science and engineering.

(a) Finding the Maximum Likelihood Estimators
Suppose that we want to find the maximum likelihood estimators of a complicated log-likelihood function based on a large data set. The evaluation of the log-likelihood for any single parameter value may take a long time if the dataset is large or density function is complicated. Typically, a maximization program must evaluate the log-likelihood for many different parameter values in order to estimate the gradient and the Hessian numerically between every two iterations. An attractive approach is to divide the large dataset into smaller sub-datasets and let the processors calculate the log-likelihood of the sub-datasets in parallel. As soon as a processor finishes its calculation, it sends the result back to the parent where it is added to the accumulated contributions of the log-likelihood from other processors. Once the log-likelihoods of all the sub-datasets are calculated and added to the total, one has the log-likelihood evaluated at a particular value of the parameters. The maximization program can continue to evaluate the log-likelihood at another parameter value using the processors in parallel or to update the maximum likelihood estimator.

(b) Monte-Carlo Analysis
Suppose we want to calculate K quantities based on a large number N of simulations. We can divide N into small pieces of size M, and have each message cause a child to execute M simulations. Let the processors do the M simulations in parallel. If the K quantities we want to calculate are all sums, each child can accumulate the results from its simulations. As soon as it

finishes the calculations, it sends the results back to the parent where they are combined with the accumulated contributions of the quantities from other processors. This distributed approach is especially attractive for simulation based statistical analysis based on Gibbs sampling, a topic of intense current interest, see Gelfand and Smith (1990) for an introduction.

(c) Asynchronous Iterative Method for Solving a System of Equations
Suppose we want to solve the system of equations $\underset{\sim}{x} = F(\underset{\sim}{x})$, where $\underset{\sim}{x} \in \mathcal{R}^n$ and n is large. For example, this problem might arise in computing the equilibrium distribution of a finite state Markov chain numerically. To compute a solution iteratively, one begins with some starting value $\underset{\sim}{x}_0$. One implements an iteration procedure by computing a sequence of successive approximations, $\underset{\sim}{x}_{n+1} = G(\underset{\sim}{x}_n)$ where G may equal F or be some modification of F to accelerate the rate of convergence. This can be done using a loosely coupled system by having each child compute a subset of the components of $\underset{\sim}{x}_{n+1}$ using all of the components of $\underset{\sim}{x}_n$. The parent combines the results to form $\underset{\sim}{x}_{n+1}$ and repeats this process until it has converged. It is possible to speed this process by using asynchronous iterations. In this case, the parent does not wait for all child processors to send their results before sending the next problem. Rather it sends the partially updated vector $\underset{\sim}{x}_{n+1}$. This approach can be much faster than synchronous iteration and will converge under mild conditions, see Baudet (1978) for details.

(d) Knapsack Problem and the Traveling Salesman Problem
These problems frequently are solved by branch and bound methods which dynamically construct a tree of simpler subproblems, from whose solutions the solution of the original problem is obtained. The parallel methods use the parent process to generate the tree of subproblems and monitor progress, and distribute the solution of the subproblems to the various computers on the network.

(e) Multiprocess Models
Schervish and Tsay (1988) developed Bayesian models for autoregressive processes which allow for changes in the model to occur. For example, the models allow a particular observation to be treated as an outlier, that is it may have a distribution other than the one used to derive the autoregressive model. Since we can rarely, if ever, be certain that an observation is an outlier, we might wish to consider two different models, one which says that the observation is an outlier, and another which says that it is not. The posterior probabilities of these two models will tell us how likely it is that the observation is indeed an outlier. Next consider the possibility that two different observations are outliers. It could be that either of them, both, or neither are outliers. This results in 4 possibilities. Indeed, if there are n observations, there are 2^n possibilities. For even moderate sized n, this is too many possibilities to consider. The method of Schervish and Tsay (1988) is to ignore all but a fixed number of the possibilities with the highest posterior probabilities. The larger that fixed number, the longer an unlikely possibility can be entertained to see if later data might make it appear more likely. In order to keep large numbers of possible models available, Schervish and Tsay (1988) divide the possible models among several processors and allow each processor to work on one model at a time as each new observation arrives. After Bayes' theorem has been applied to a new observation, the possible models with the highest posterior probability are maintained.

(f) Discrete-Finite Models

In 1986, Eddy and Schervish studied models for discrete random variables specified by probabilities which can only assume a finite number of values, d. They assumed a discrete prior distribution which is constrained to be equal to one of m possible values, and they wanted to calculate the predictive distribution of a single future observation. They claimed that when $d = 10$ and $m = 300$, the number of possible models is 6×10^{16}, and the estimated time to compute the predictive distribution of a single future observation is roughly one million years of CPU time on a VAX 11/750. Consequently, they considered two smoothness conditions for their models and hence reduced the number of model vectors in the calculation. These, combined with very high speed computation made some previously infeasible discrete-finite models computationally tractable. In particular, they made use of the local area network and different numbers of DEC VAXes on their network in order to perform most of the required computations in parallel. As pointed out in their paper, this breakthrough has two important advantages. First, it dramatically reduces the "wall clock time" required to perform discrete-finite calculations. Second, it provides a numerically more stable algorithm for computing the results. This is because parallel computation also facilitates the partitioning of numerical results into sums which can be accumulated with greater numerical accuracy.

(g) A Charged Particle Transport Program

The BOHR program was used by Olson and Salop (1977) to study early events in the radiation damage of semiconductors. These computations examine the interactions between a high-energy incoming projectile and a target system. For example, one computation studies an argon projectile scattering from a silicon target. This Monte Carlo technique uses a few random numbers to initialize the projectile direction and velocity. It then follows the trajectory by numerically integrating the classical equations of motion until the collision is completed. The results of the collision are then assessed. Electrons in the target, for instance, may have been ejected, excited, or transferred to the projectile. The recoil energy of the target is computed. After this, another trajectory is set up and followed. One run of BOHR consists of following thousands of trajectories for which various possible outcome results are averaged, counted and tabulated. The parallel implementation of this is straightforward, since each trajectory is completely independent of the next. Each child repeatedly follows one trajectory and sends the results back to the parent. The technique proposed by Percus and Kalos (1987) can be used to provide each child processor with its own independent sequence of random numbers. The parent receives and tabulates trajectory results until the required number has been received and writes the output file.

4 Message sizes

Ideally, if a loosely coupled system behaves in a deterministic way, the communication time between processors is linear in the message size. As soon as the smaller pieces are sent to the processors, each processor starts working on the subproblem at a constant rate. The elapsed time to complete the problem will then be minimized by sending one message to each processor so that all the subtasks finish at the same time. However, in a loosely coupled time-sharing system, calculations are performed at different rates by the same processor at different times.

The rate depends on the demands on that particular processor at that time. If one of the processors has considerably more work to do from other users than the other processors, then sending a large computation to each processor can be counterproductive. It is better to divide the work into a large number of messages and to send them to the children as they are needed. That is, when a child finishes a message and returns the results to the parent, the parent then sends the child the next available message. There are advantages and disadvantages to having small message sizes. One advantage of small sizes is that a slow child (either a slow CPU or a busy machine) will receive only a few messages, leaving the bulk of the work to be performed by those children who have the time and resources to do it. An added advantage to partitioning the computation into small parts is that the numbers being combined are more nearly the same size than would result from a serial algorithm which increases numerical accuracy. A definite advantage to small messages appears when one considers the possibility of one child "dying" prematurely. All the work done by that child since it last reported results to the parent is lost and must be redone. Since the network is not as reliable as one would wish, and systems sometimes crash for unanticipated reasons, it pays to have small messages. On the negative side, if there are too many messages (hence very small ones), then most of the processors answer back almost immediately to the parent, and they request more work to be sent. Time is therefore wasted, while the parent deals with all the asynchronous requests from the other processors. At the same time it can cause a communication bottleneck for the parent. Furthermore, there is an overhead associated with each subtask. This is the total communication time for the parent to send a message to the other processors and for the other processors to report back to the parent. As the number of subtasks increases, the total overhead of the task increases as well. An understanding of the underlying working environment and a flexible message distribution system are therefore required to optimize the total elapsed time to complete a task on a loosely coupled parallel system. In the next section of this paper, we present a study of the tradeoffs between the overhead and the size of the subproblems.

5 Tradeoffs between overhead and block size

As mentioned in the previous section, if a problem is divided into subproblems and if the size (in term of blocks) of each subproblem (message) is too large, then time will be wasted at the end of the computation in waiting for the last processor to finish. On the other hand, if the message size is too small, then there is communication bottleneck for the parent processor. The overhead associated with each subproblem is defined to be the mean total communication time for the parent to send a message to the child and for the child to send the results back to the parent when the child has completed the subproblem. In this section, we study the tradeoffs between the overhead and the size of the subproblems.

It is assumed that there is a computation or problem of size w, meaning that a single processor working alone at normal rate 1 would require w time units to complete it. Assume there are p loosely coupled independent processors working in parallel. Suppose that the problem is divided into n equal size subproblems. Let T_i be the total time taken by the ith processor to receive a subproblem from the parent, to complete it and to send the results back. Assume that T_i

is random with mean $\dfrac{w}{r_i n} + a_i + b_i x$ and variance σ_i^2. Here, r_i is the rate associated with the ith processor, a_i is the fixed overhead, b_i is the rate at which messages are transmitted and x is the message size. The subproblem size, $\dfrac{w}{n}$, and the message size x are not related. Let $c_i = a_i + b_i x$ where c_i can be thought of as the mean overhead time of the ith processor. Assume that as soon as the parent receives the results from the ith processor, it immediately sends out another message to the ith processor. Let $N_i(t)$ be the number of subproblems that the ith processor has completed at time t. Then $N_i(t)$ is a renewal process, and the successive occurrence times between events are $\{T_{ik}\}_{k=1}^{\infty}$, where T_{ik} are independent and identically distributed and have the same distribution as T_i for all $k = 1, 2, \ldots$ From now on, let us assume that the number of subproblems, n, is large. By the Renewal Central Limit Theorem given in Karlin and Taylor (1975), as $t \to \infty$,

$$N_i(t) \sim N\left(\frac{t}{\mu_i}, \frac{\sigma_i^2 t}{\mu_i^3}\right),$$

where

$$\mu_i = \mathcal{E}(T_i) = \frac{w}{r_i n} + c_i, \quad Var(T_i) = \sigma_i^2.$$

Since all processors are independent, it follows that as $t \to \infty$,

$$\sum_{i=1}^{p} N_i(t) \sim N\left(t \sum_{i=1}^{p} \frac{1}{\mu_i}, t \sum_{i=1}^{p} \frac{\sigma_i^2}{\mu_i^3}\right). \tag{5.1}$$

Let $\mathcal{T}_{p,k}$ be the time for the system of p processors to complete k subproblems. From Lam and Lehoczky (1991), we can invert the asymptotic result given by (5.1) so that as $k \to \infty$,

$$\mathcal{T}_{p,k} \sim N\left(\frac{k}{\sum_{i=1}^{p} \frac{1}{\mu_i}}, \frac{k \sum_{i=1}^{p} \frac{\sigma_i^2}{\mu_i^3}}{\left(\sum_{i=1}^{p} \frac{1}{\mu_i}\right)^3}\right). \tag{5.2}$$

At time $\mathcal{T}_{p,n-p+1}$, one of the processors completes a subproblem. However, there are no more subproblems to be sent. From Result (5.2) above, $\mathcal{T}_{p,n-p+1}$ is asymptotically normal with mean depending on $n - p + 1$ and μ_i, $i = 1, 2, \ldots, p$. Let $T_{p,n}$ be the random time required for the system to complete the n messages. Then

$$T_{p,n} = \mathcal{T}_{p,n-p+1} + Y_{p,n}$$

where $Y_{p,n} = \max_{j \neq i}(Y_j(\mathcal{T}_{p,n-p+1}))$ if the $(n - p + 1)$th subproblem is finished by the ith processor. $Y_j(\mathcal{T}_{p,n-p+1})$ is the remaining life distribution of the jth processor, $j = 1, 2, \ldots, p$, at time $\mathcal{T}_{p,n-p+1}$. Let $\underset{\sim}{R}_{p,n} = (Y_1(\mathcal{T}_{p,n-p+1}), \ldots, Y_p(\mathcal{T}_{p,n-p+1}))$ and $\underset{\sim}{r} = (r_1, \ldots, r_p) > 0$. From Lam (1990),

$$\lim_{n\to\infty} \mathcal{P}(\underset{\sim}{R}_{p,n} \le \underset{\sim}{r}) = \sum_{i=1}^{p} \frac{\frac{1}{\mu_i}}{\frac{1}{\mu}} \left[\prod_{j=1, j\neq i}^{p} \int_0^{r_j} \frac{(1 - F_j(y))}{\mu_j} dy \right] \tag{5.3}$$

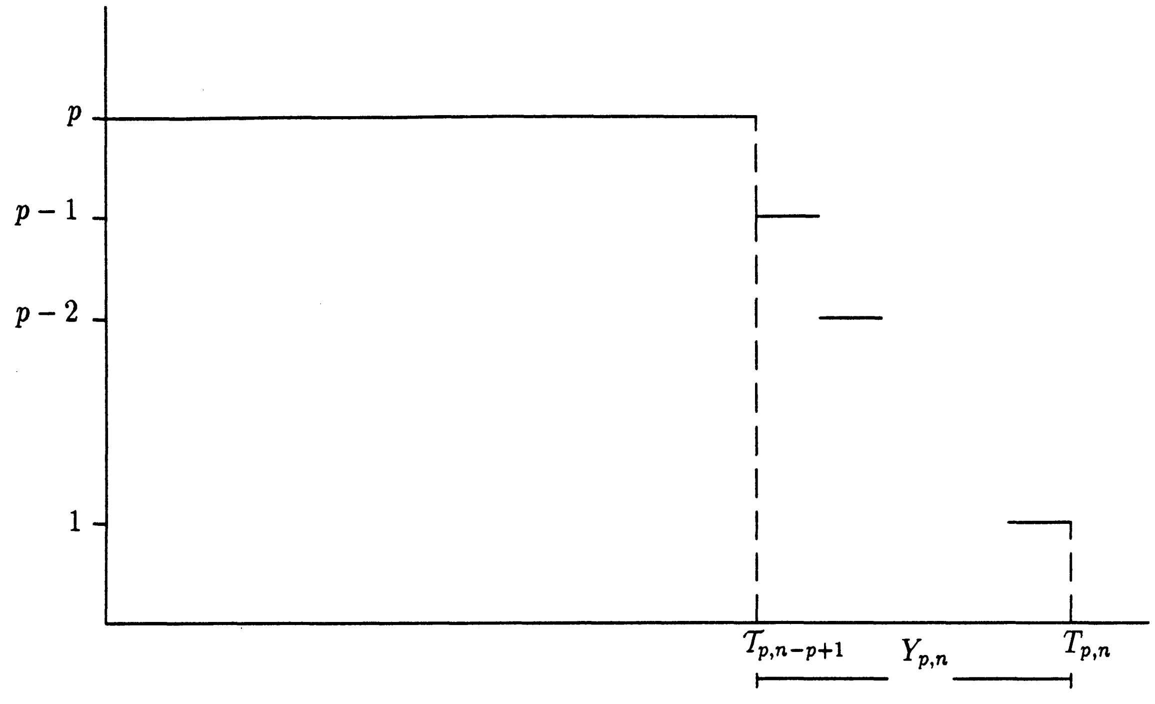

Figure 5.1: Number of processors working on the problem versus time

where F_j is the distribution function of T_j and $\frac{1}{\mu} = \sum_{j=1}^{p} \frac{1}{\mu_j}$. This means that as $n \to \infty$, knowing that the $(n-p+1)$th subproblem is completed by the ith processor, then all the $Y_j(\mathcal{T}_{p,n-p+1})$, $j \neq i$, are independent, and they follow the limiting remaining life distribution of the individual renewal process. Let $I_{p,n-p+1}$ be the indicator such that $I_{p,n-p+1} = i$ if the $(n-p+1)$th subproblem is finished by the ith processor. Then for any $y > 0$,

$$\begin{aligned}
\lim_{n\to\infty} \mathcal{P}(Y_{p,n} \le y) &= \lim_{n\to\infty} \sum_{i=1}^{p} \mathcal{P}(Y_{p,n} \le y, I_{p,n-p+1} = i) \\
&= \lim_{n\to\infty} \sum_{i=1}^{p} \mathcal{P}(\bigcap_{j=1, j\neq i}^{p} \{Y_j(\mathcal{T}_{p,n-p+1}) \le y\}, I_{p,n-p+1} = i) \\
&= \lim_{n\to\infty} \mathcal{P}(\bigcap_{j=1}^{p} \{Y_j(\mathcal{T}_{p,n-p+1}) \le y\}) \\
&= \lim_{n\to\infty} \mathcal{P}(\underset{\sim}{R}_{p,n} \le y \underset{\sim}{1}),
\end{aligned}$$

where $\underset{\sim}{1}$ is a column vector of 1's. Hence by (5.3) above,

$$\lim_{n\to\infty} \mathcal{P}(Y_{p,n} \le y) = \sum_{i=1}^{p} \frac{\frac{1}{\mu_i}}{\frac{1}{\mu}} \Big[\prod_{j=1, j\neq i}^{p} \int_0^y \frac{1 - F_j(x)}{\mu_j}\, dx \Big]. \tag{5.4}$$

From Equation (5.4), one can then calculate $\mathcal{E}(\lim_{n\to\infty} Y_{p,n})$, and hence an approximation for the mean limiting time to complete the problem. We obtain

$$\mathcal{E}(\lim_{n\to\infty} T_{p,n}) \doteq \frac{n-p+1}{\sum_{i=1}^{p} \frac{1}{\mu_i}} + \mathcal{E}(\lim_{n\to\infty} Y_{p,n}).$$

When n is large, $\mathcal{E}(\lim_{n\to\infty} Y_{p,n})$ is small compared to $(n-p+1)/\sum_{i=1}^{p} \frac{1}{\mu_i}$. Furthermore, result (5.2) holds for all distributions of T_i. This suggests that the distribution of T_i has very little effect on the mean limiting completion time of the problem.

Special case

In the case when all the T_i are exponentially distributed with the same parameter, i.e. $r_i = r$, $c_i = c$, $\mu_i = p\mu = \frac{w}{rn} + c$ and $\sigma_i^2 = (p\mu)^2$ for $i = 1, 2, \ldots, p$, then $\sum_{i=1}^{p} N_i(t)$ is a renewal process, and the interarrival time is again exponentially distributed. Furthermore,

$$\mathcal{E}(T_{p,n-p+1}) = \left(\frac{w}{rn} + c\right)\left(\frac{n-p+1}{p}\right)$$

and

$$\mathcal{E}(Y_{p,n}) = \sum_{i=1}^{p-1} \mathcal{E}(T_{i,1}) = \left(\frac{w}{rn} + c\right)\sum_{i=1}^{p-1} \frac{1}{i}.$$

Hence,

$$\mathcal{E}(T_{p,n}) = (\frac{w}{rn} + c)(\frac{n}{p} + H(p) - 1), \tag{5.5}$$

where

$$H(p) = \sum_{i=1}^{p} \frac{1}{i}.$$

The expected elapsed time $\mathcal{E}(T_{p,n})$ is minimized when

$$n = \sqrt{\frac{pw(H(p)-1)}{cr}}. \tag{5.6}$$

The second derivative of $\mathcal{E}(T_{p,n})$ with respect to n is always positive. This implies that $\mathcal{E}(T_{p,n})$ is a convex function of n. Note that the optimal n increases as the size of the problem w or the number of processors p increase. This is reasonable because when w is large or p increases, we would want to divide the problem into small sized subproblems so as to minimize the end effect, $\mathcal{E}(Y_{p,n})$. The optimal n decreases as c increases. When the mean overhead c is large, we would want to divide the problems into fewer subproblems each with large size. Also,

$$\begin{aligned}
Var(T_{p,n}) &= Var(\mathcal{T}_{p,n-p+1}) + Var(Y_{p,n}) \\
&= (n-p+1)\mu^2 + \sum_{i=1}^{p-1} Var(\mathcal{T}_{i,1}) \\
&= (n-p+1)\mu^2 + \sum_{i=1}^{p-1}\left(\frac{p\mu}{i}\right)^2 \\
&= (\frac{w}{rn}+c)^2(\frac{n}{p^2}+L(p)-\frac{1}{p}) \qquad (5.7)
\end{aligned}$$

where

$$L(p) = \sum_{i=1}^{p}\frac{1}{i^2}.$$

From Result (5.7), we can conclude that as n increases, the variance of the time to complete the problem decreases. Also, as w or c increase, the variance of the completion time increases. In the next section, two numerical examples are given to illustrate the applications of the results in this section.

6 Numerical examples

In this section, the performance of the approximate analysis given in Section 5 is compared with the actual execution of two problems carried out on a loosely coupled parallel computing system. In particular, the observed completion times of the numerical examples on the parallel system are compared to the expected completion times computed from the theoretical models. We should expect similar conclusions to apply for loosely coupled parallel implementations of Monte-Carlo analyses, knapsack problems, traveling salesman problems and many other management science applications for which the computation can be broken into independent parts.

(a) Hierarchical models for serially correlated economic data

Suppose that there are g distinct groups of stocks, and for group p, $p = 1, 2, \ldots, g$, there are m_p individual stocks. For the ith stock in group p, the daily stock closing prices are recorded. Individual stocks within the same group share some common characteristics. We are interested in studying the differences among the groups of stocks and differences with a single stock group. Consider the following model for the daily stock closing prices:

$$y_{ijp} = f(\mu_{ip}, A_{ip}^1, \ldots, A_{ip}^k) + w_{ijp} \qquad (6.1)$$

where p is the index for group, $p = 1, 2, \ldots, g$; i is the index for individual stock, $i = 1, \ldots, m_p$ and, j is the index for observations, $j = 1, 2, \ldots, n_{ip}$. y_{ijp} is, therefore, the jth observation of the ith stock in group p, while $f(\mu_{ip}, A_{ip}^1, \ldots, A_{ip}^k)$ and w_{ijp} represent the deterministic and stochastic components of the daily price respectively. $(\mu_{ip}, A_{ip}^1, A_{ip}^2, \ldots, A_{ip}^k)^t$ is a random vector of characteristics associated with the ith stock in group p. Assume that for each $p = 1, 2, \ldots, g$ and $i = 1, 2, \ldots, m_p$,

$$\begin{bmatrix} \mu_{ip} \\ A_{ip}^1 \\ A_{ip}^2 \\ \vdots \\ A_{ip}^k \end{bmatrix} \sim N\left(\begin{bmatrix} \lambda_{0p} \\ \lambda_{1p} \\ \lambda_{2p} \\ \vdots \\ \lambda_{k,p} \end{bmatrix}, \begin{bmatrix} \sigma_0^2 & & & & \\ & \sigma_1^2 & & 0 & \\ & & \sigma_2^2 & & \\ & 0 & & \ddots & \\ & & & & \sigma_k^2 \end{bmatrix} \right)$$

Also, assume that the random vectors $(\mu_{ip}, A_{ip}^1, A_{ip}^2, \ldots, A_{ip}^k)^t$ are independent for all $p = 1, 2, \ldots, g$ and $i = 1, 2, \ldots, m_p$. The above model enables us to assess differences among groups of stocks and at the same time, it incorporates variation across individual stocks within the same group. Furthermore, let

$$\begin{aligned} \underset{\sim}{\beta}_{ip} &= (\mu_{ip}, A_{ip}^1, \ldots, A_{ip}^k)^t \\ \underset{\sim}{\beta}_{0p} &= (\lambda_{0p}, \lambda_{1p}, \ldots, \lambda_{kp})^t \end{aligned}$$

and

$$\Sigma_0 = \text{diag}(\sigma_0^2, \sigma_1^2, \ldots, \sigma_k^2).$$

Assume $\underset{\sim}{\beta}_{ip}$ follows a prior $N(\underset{\sim}{\beta}_{0p}, \Sigma_0)$. For each fixed i and p, let $\{w_{ijp}\}_{j=1}^{n_{ip}}$ be an ARMA(ζ, s) process, i.e.

$$w_{ijp} - \sum_{u=1}^{\zeta} \phi_u w_{i,j-u,p} = a_{ijp} - \sum_{v=1}^{s} \theta_v a_{i,j-v,p}$$

where a_{ijp} is normally distributed with mean 0 and variance σ^2. The processes $\{w_{ijp}\}_{j=1}^{n_{ip}}$, $p = 1, 2, \ldots, g$ and $i = 1, 2, \ldots, m_p$ are all independent of each other but they all have the same time series parameters. In the case that the function $f(\mu_{ip}, A_{ip}^1, \ldots, A_{ip}^k)$ is linear in $\mu_{ip}, A_{ip}^1, \ldots, A_{ip}^k$, the likelihood function of this model can be derived easily, for more details, see Greenhouse, Kass, Lam and Tsay (1990).

System performance

With $\zeta = 1$, $s = 0$, $k = 1$, $g = 2$, $m_p = 2500$ and $n_{ip} = 100$ for all $i = 1, \ldots, 2500$ and $p = 1, 2$, the time taken to find the maximum likelihood estimators of the unknown parameters is 13 hours 26 minutes and 35.25 seconds (48,395.25 seconds) on a single Microvax II. Five Microvax II's are used to do the calculation in parallel. The result is given in Table 1 and plotted in Figure (6.1). The observed data are divided into three groups, daytime batch jobs (9am to 5 pm), evening time batch jobs (5pm to 1 am) and night time batch jobs (1am to 8am). In this particular example, there is a total of 5,000 stocks in two groups. Each message consists of telling the child to calculate the likelihood function for a subset of stocks. For example, when there are 100 messages, each message tells the child to calculate the likelihood function for 50 stocks at a time. The result is sent back to the parent which is added to the accumulated contribution of the likelihood function from the other children. On average, the time taken to evaluate the likelihood function once using a single Microvax II is 46 minutes and

	time (seconds)		
number of messages	first run	second run	third run
5	30843.95	20088.23	26334.97
10	24546.89	19404.92	17941.06
20	23163.30	18376.62	17456.01
25	18576.02	19811.04	15029.49
40	21693.64	19545.32	18942.60
50	20622.79	18733.76	18541.44
100	15866.47	13930.14	13199.70
125	14969.35	13482.20	13901.44
200	13502.17	13924.26	15868.98
250	17968.39	18421.23	18866.52
500	15467.40	19030.14	18728.52
1000	14964.12	20246.73	20591.94
1250	17678.10	21153.52	16075.82
2500	20882.46	20419.54	22940.89
5000	25228.65	28796.02	28455.84

Table 1: Times to compute maximum likelihood estimator

6.46 seconds (2766.46 seconds). The communication time for each message between the parent and the children can vary from 1.21 to 1.39 seconds. Most of the time it varies from 1.21 to 1.24 seconds. The time required by that part of the computation which cannot be performed in parallel is estimated to be 4131.89 seconds. From Table 1 above, the best timing obtained for the five Microvax II system is 13199.70 seconds. This is remarkably close to the best timing we could have expected from the system, namely (48395.25-4131.89)/5+4135.89=12984.56 seconds. We have a speedup of almost a factor of 5.

Let I be the total number of likelihood function evaluations before the function meets some convergence criterion. For example, we may want the relative change of the function values in two consecutive steps to be less than some small number ϵ. In this example, ϵ is equal to 0.005. Let T_s be the time of the part of the computation which cannot be performed in parallel. Let $T_{p,i}$ be the time to compute the likelihood function in the ith evaluation. $T_{p,i}$ is therefore the part of the work that can be done in parallel. We want to minimize $T_s + \sum_{i=1}^{I} T_{p,i}$, the time to complete the maximum likelihood estimation. Putting $w = 2766.46$, $c = (1.21 + 1.24)/2 = 1.225$, $r = 1$ (corresponding to night time batch jobs) and $p = 5$ in Expression (5.6) in the previous section, we have the optimal $n = 120.38$ or $\log(n) = 4.79$. With the above values of w, c, r and p, we can plot Expression (5.5) as a function of $\log(n)$, and it is given in Figure (6.2). Figure (6.2) also shows the total observed elapsed time of the system for different message sizes at night time and the 2(standard deviation) envelop curves. From Figure (6.2), we can conclude that the exponential assumption seems to be appropriate to predict the optimal number of equal size

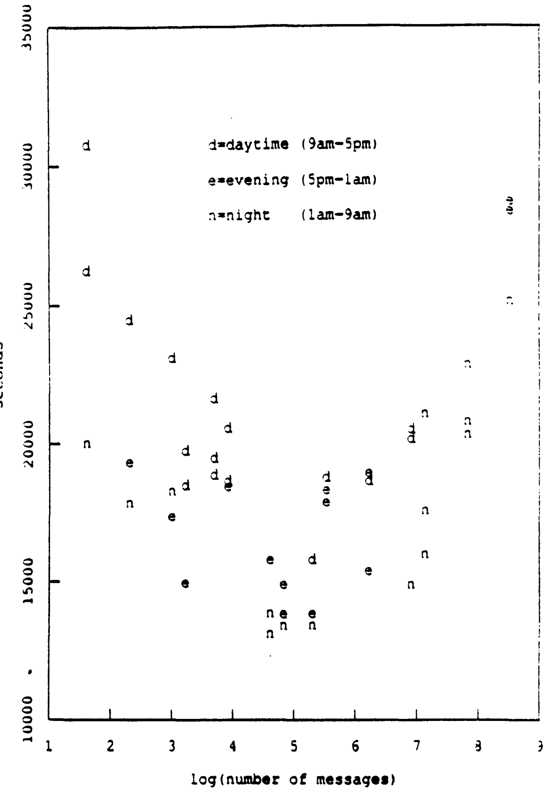

Figure 6.1 : Observed data of hierarchical models

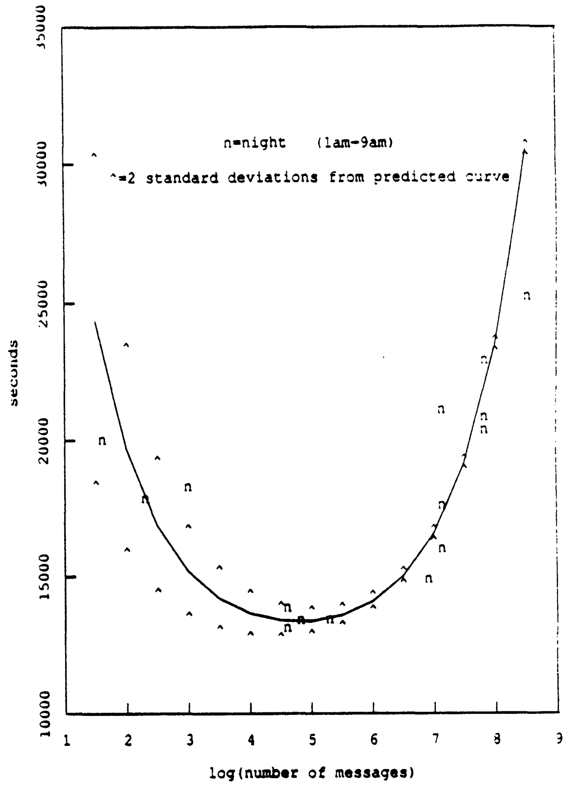

Figure 6.2 : Hierarchical models for serially correlated data

messages into which to divide our problem. The plotted curve tells us the mean completion time of the problem for different message sizes at night time. In this example, the predicted curve is not as flat as the one given in the next example. This is because the mean communication time in this example is much larger than that for discrete-finite models considered next. The elapsed time to complete the problem is therefore more sensitive, and it increases sharply as the number of messages increases. We see that the expected curve fits the data very well. Most of the observed data lie inside 2 standard deviations of the predicted curve. Similar analyses can be carried out for daytime and evening batch jobs. In these cases, we would expect the working rate r to be relatively smaller than that of night time jobs.

(b) Discrete-Finite Models [Eddy and Schervish]

Suppose X is a random variable, and it can assume one of d possible values $x_1, \ldots, x_d$. These values can be numerical or nominal, vector or scalar. We call this set of possible values the observation space. The distribution of X consists of a vector $\underset{\sim}{p} = (p_1, \ldots, p_d)^t$, where

$$p_j = \mathcal{P}(X = x_j), \tag{6.2}$$

and

$$\underset{\sim}{p}^t \ \underset{\sim}{1} = 1. \tag{6.3}$$

Since all calculations are discrete and finite, assume each p_j is constrained to equal one of m possible values, $v_1, \ldots, v_m$. For simplicity, we only consider the case where the $\{v_k\}$ lies on a grid that is equally spaced in $[0,1]$; that is, suppose that

$$v_k = (k-1)/(m-1).$$

One need only consider that subset of the collection of m^d "possible" vectors $\underset{\sim}{p}$ which satisfy Equation (6.3). When $\underset{\sim}{p}$ is specified, all inference about X can be based on Equation (6.2). Call $\underset{\sim}{p}$ the model vector and the set of all possible $\underset{\sim}{p}$ vectors the model space. Next, assume that one is interested in making inferences about a subset of some sequence $X_1, X_2, \ldots$ of observables which are exchangeable, in the sense that their labels provide no information about their joint distribution. Assume each X_i must equal one of the values $x_1, \ldots, x_d$. A theorem of de Finetti (1937) (also see Hewitt and Savage, 1955) shows that conditional on some vector $\underset{\sim}{p}$ satisfying Equation (6.3), the X_i are independent with distribution given by Equation (6.2). Once again, the model space is a finite collection of vectors, say, $\{\underset{\sim}{r}_1, \ldots, \underset{\sim}{r}_t\}$, with $\underset{\sim}{r}_s = (r_{1s}, \ldots, r_{ds})^t$. For convenience, let the distribution of $\underset{\sim}{p}$ be uniform. Use the subscript i to index possible values. Since, for each i,

$$\mathcal{P}(X_i = x_j \mid \underset{\sim}{p}) = p_j \tag{6.4}$$

The conditional distribution of $\underset{\sim}{p}$ given any finite subset X^* of $X_1, X_2, \ldots$ can be calculated as follows. For $j = 1, \ldots, d$, let n_j be the number of observed X_i's in X^* equal to x_j so that $(n_1, n_2, \ldots, n_d)$ has a multinomial distribution conditional on $\underset{\sim}{p}$, we obtain

$$\mathcal{P}(\underset{\sim}{p} = \underset{\sim}{r}_s \mid X^*) = K^{-1} \prod_{j=1}^{d} (r_{js})^{n_j} \quad \text{for } s = 1, \ldots, t \tag{6.5}$$

where

$$K = \sum_{u=1}^{t} \prod_{j=1}^{d} (r_{ju})^{n_j}.$$

The joint distribution of any further set of X's is given by Equations (6.4) and (6.5) and conditionally independent given $\underset{\sim}{p}$. For example, the predictive distribution of a single future X is

$$\mathcal{P}(X = x_j \mid X^*) = \sum_{s=1}^{t} r_{js} \mathcal{P}(\underset{\sim}{p} = \underset{\sim}{r}_s \mid X^*). \tag{6.6}$$

System performance
From Eddy and Schervish (1986), with $n = 15$ observations on a variable assuming $d = 10$ different values, the amount of time to calculate the predictive distribution of one future observation on a VAX 11/750 for various grid sizes m is given in Table 2. The estimated time referred to in the last line of Table 2 is approximately equal to one million years. To make the discrete finite calculation more feasible in a reasonable amount of time, Eddy and Schervish (1986) first put some restrictions on the model vectors, and then use parallel processing to do the calculation.

Smoothness Specify some value ϵ and allow only those vectors $\underset{\sim}{r}_s$ with adjacent coordinates closer than ϵ to be included in the model space.

m	number of $\underset{\sim}{p}$'s	time (seconds)
10	48,620	29.53
15	817,190	431.5
20	6,906,900	3529
300	6×10^{16}	3×10^{13}

Table 2: Times to compute predictive distribution

Parallel processing Eddy and Schervish (1986) used $n = 14$ observations on a variable assuming $d = 19$ values (0 to 9 in steps of 0.5) with a model space having $m = 22$ and a smoothness criterion $\epsilon = 0.095$. The total number of model vectors is 38,226,040 and the calculation of the predictive distribution of one future observation took 11 hours and 8 minutes on a single Microvax II (40,100 seconds).

- **First system:** It consists of 8 nodes, six Microvax II's, the 11/750 and one Microvax I. This is roughly equivalent to seven Microvax II's since the 11/750 runs about 80% as fast and the Microvax I's run about 20% as fast as the Microvax II's.
- **Second system:** It consists of 15 nodes, eight Microvax II's, the 11/750 and six Microvax I's. This is roughly equivalent to ten Microvax II's.

Figure (6.3) shows the total elapsed time for each system for several different sizes of messages. The horizontal axis gives the natural logarithm of the number of messages rather than message size. The pattern confirms that when there are few messages (hence large ones), time will be wasted waiting for the last slow machine to finish its last message. And when there are too many messages (hence very small ones), time is wasted while the parent deals with all of the asynchronous read requests being answered almost immediately and the extra overhead from message communication. Figure (6.3) also tells us that the total elapsed time is insensitive for $\log(n)$ lying between 7.0 and 9.0, or for n lying between 1,097 and 8,103. This means that the total elapsed time does not change much whether we divide the computation into 1,097 messages or 8,103 messages. It was reported in Eddy and Schervish (1986) that the mean overhead in this example varied from 0.13 to 0.15 seconds regardless of which system was used or how many messages were sent. Putting $w = 40,100$, $c = (0.13 + 0.15)/2 = 0.145$ and $r = 1$ in Expression (5.6), we have

- When $p = 7$, $n = 1,756$ and $\log(n) = 7.47$.
- When $p = 10$, $n = 2,310$ and $\log(n) = 7.745$.

With the above values for w, c and r, we can also plot Expression (5.5) as a function of $\log(n)$, and it is given in Figure (6.4). From Figure (6.4), we can conclude that the exponential assumption appears to be appropriate to help us to predict the optimal number of messages into which to

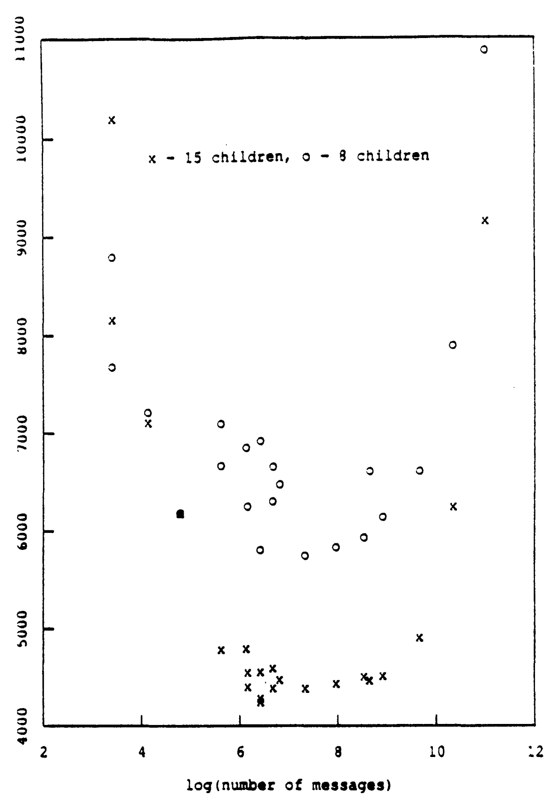

Figure 6.3 : Observed Data for Discrete-Finite Models

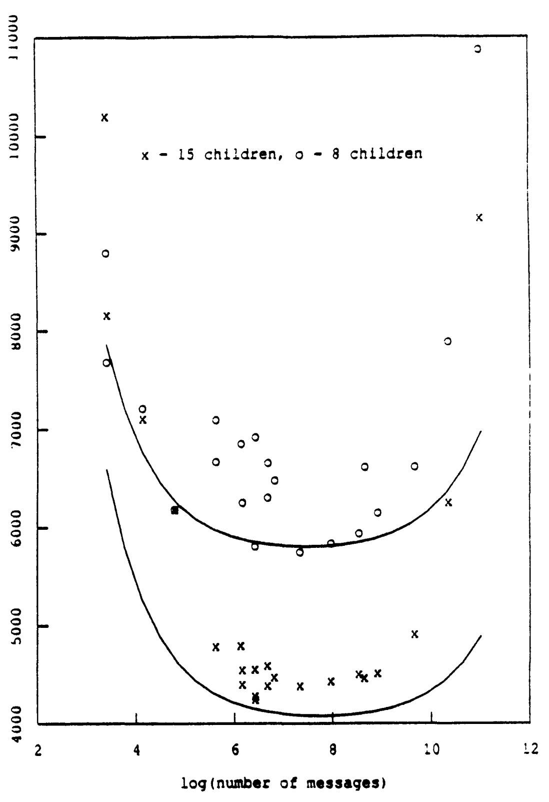

Figure 6.4 : Observed and Fitted Data for Discrete-Finite Models

divide the problem. The plotted expected time curve for the system to complete the problem seems to fit the observed data quite well. The predicted values from the model are slightly smaller than the observed data, this is because we approximate the first system (8 children) by 7 Microvax II's and the second system (15 children) by 10 Microvax II's. The actual total observed overhead should therefore be larger than that from the model. In Figure (6.5), we plot the observed data, predicted curve and the predicted curve plus 2(standard deviation) of the second system with 15 children running in parallel. As mentioned in the previous section, the standard deviation decreases as n increases. From Figure (6.5), we can see that when n is large, the observed data are more than 2 standard deviations larger than the predicted curve. This is because in our model of the superposition of p renewal processes, we have assumed that the parent can read the answer of all the asynchronous read of the children at the same time. In reality, the parent in the software system developed by Eddy and Schervish (1986) can only answer one child at a time. All the others queue up and are dealt with in the order of arrival. When n is large, most of the processors answer back almost immediately. At the same time, they request more work to be sent. Time is, therefore, wasted in waiting for the parent to deal with all the asynchronous requests from all the processors. In Figure (6.6), we plot the observed data, the predicted curve and the predicted curve plus 2(standard deviation) of the first system with 8 children running in parallel. It has similar interpretations as in Figure (6.5).

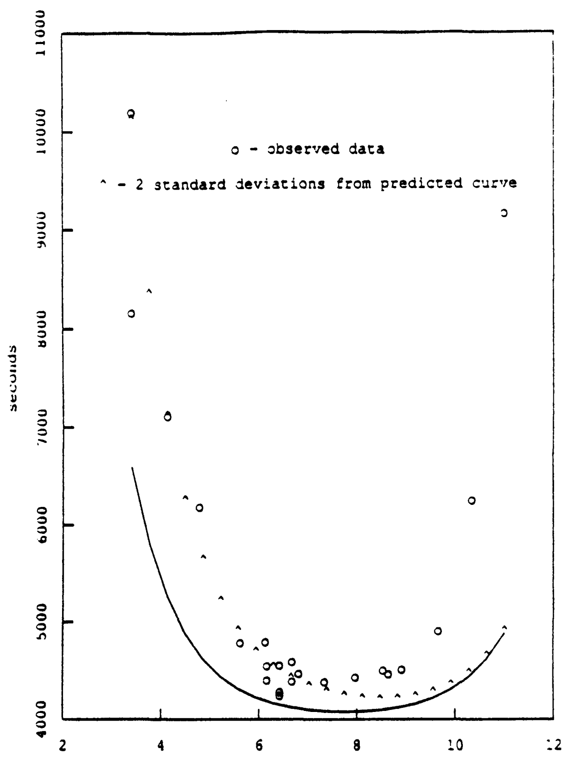

Figure 6.5 : 15 children - Discrete-Finite Models

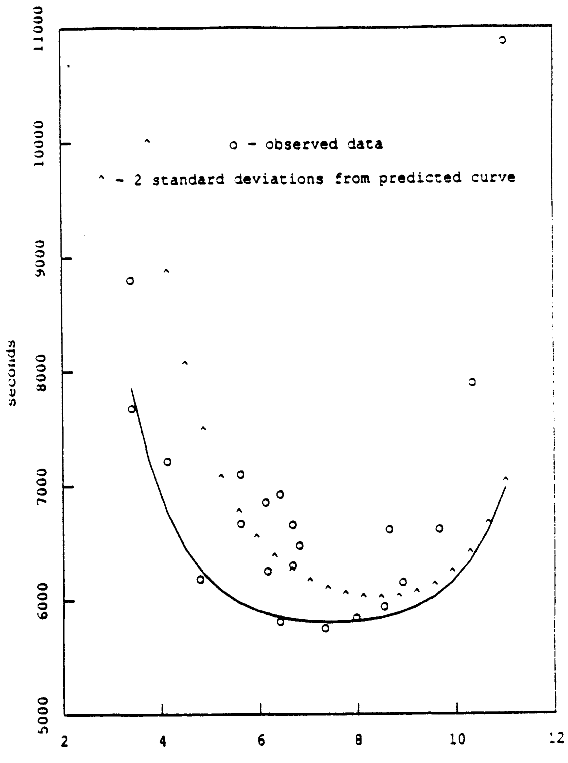

Figure 6.6 : 8 children - Discrete-Finite Models

7 Simulations

The two numerical examples in the previous section both suggest that exponential assumption may be appropriate to help us in choosing an optimal number of subproblems into which to divide our problem. We know that each processor has a maximum rate R if all its resources are devoted to the subtask. Each subtask has a fixed amount of work. Thus, there is a lower bound $\frac{w}{nR} + c$ on the amount of time to finish a subtask. Also, the overhead should be constant and not stochastic. These suggest that the exponential assumption for the total time taken by the child to receive a subproblem from the parent, to complete it and to send the results back is incorrect. However, from Section 5, we know that the distribution of this total time has very little effect on the limiting expected completion time of the problem. In the special case that all the processors are homogeneous and the times to complete the subproblems are all exponentially distributed with the same parameter, the completion time of the problem can be derived easily. The exponential distribution is therefore chosen so that the calculation of the completion time of the problem is simplified. Furthermore, from Cox (1962), we know that the superposition of indefinitely many independent and identically distributed equilibrium renewal processes behaves locally like a Poisson process. 'Local' behavior when p, the number of equilibrium renewal processes, is large refers to behavior of the superposed process over time periods short compared with individual renewal times. In particular, if the mean interarrival time of the renewal processes is μ, then the interarrival time in the superposition of equilibrium renewal processes is asymptotically exponentially distributed with parameter μ/p as $p \to \infty$. Also, the adjacent intervals in the superposed process are asymptotically independently distributed for large p. The exponential assumption is therefore a reasonable approximation in the case when the number of messages, n, is large. Large n means that each processor works on many subproblems and therefore reaches equilibrium when it starts to work on its last subproblem. We can therefore approximate $\mathcal{E}(\lim_{n\to\infty} Y_{p,n})$ in Section 5 by the corresponding result when all the underlying renewal processes are independent and identical Poisson Processes. The exponential approximation should improve as the number of processors increases.

Simulations are carried out to verify the heuristic argument in previous paragraph. Consider the superposition of p independent and identically distributed renewal processes. The interarrival arrival time of the individual renewal processes follow a gamma distribution with parameters g and h. The mean interarrival time is gh. g is the shape parameter and h is the scale parameter. In the discrete-finite models example in Section 6, $w = 40100$ seconds and $c = 0.145$ seconds. Simulations are carried out for $g = 2, 3, 4, 5$. For each g, h is chosen so that the mean interarrival time is $gh = \frac{w}{n} + c$ for different number of messages n. For different combination of g, n and p, the completion time of the job is generated 100 times and the average completion time is calculated. The results are plotted in Figures (7.1) to (7.4). The figures confirm that

1. For large n, exponential approximations perform very well. This is not surprising because the limiting expected completion time of the problem is insensitive to the distribution of the interarrival time of the individual renewal processes.

2. For large n, exponential approximations improve as p increases. From the figures, the absolute error of the approximation decreases for $\log(n) \geq 8$ as p increases.

3. Exponential approximations fail for small n. However, loosely coupled parallel system is only useful if w is large and c, the overhead time is relatively small. Hence, in most applications, we should expect the optimal number of subproblems into which to divide our problem to be reasonably large.

4. Our objective here is to find an optimal way to divide our problem into equal size subproblems. In all four figures, exponential approximation perform very well in predicting the optimal number of equal size messages into which to divide our problem. By varying g between 2 to 5, we demonstrate that the exponential approximation is useful for distributions with different shapes.

5. From the figures, when n is small, exponential approximation uniformly overestimate the completion time. This is not surprising because exponential distribution has constant failure rate but gamma distribution with shape parameter $g \geq 1$ has failure rate that increases over time.

Although simulations are only carried out using gamma distributions, we should expect similar results to hold for other distributions. The results in Section 6 and this section all confirm that both exponential approximation and infinite subproblems assumption are very useful in helping us to predict the optimal number of messages into which to divide the problem. In particular, exponential approximation only works well when the number of subproblems are reasonably large. As mentioned above, this is usually the case if the problem size w is large and overhead is relatively small.

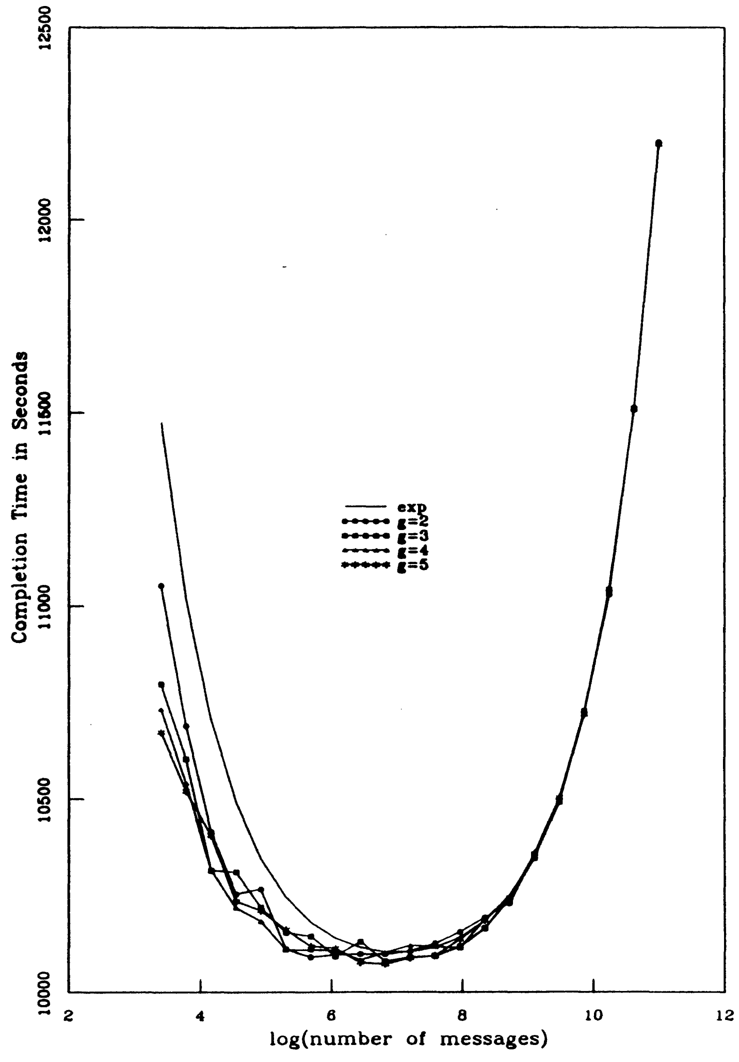

Figure 7.1 : Simulation Results with p=4

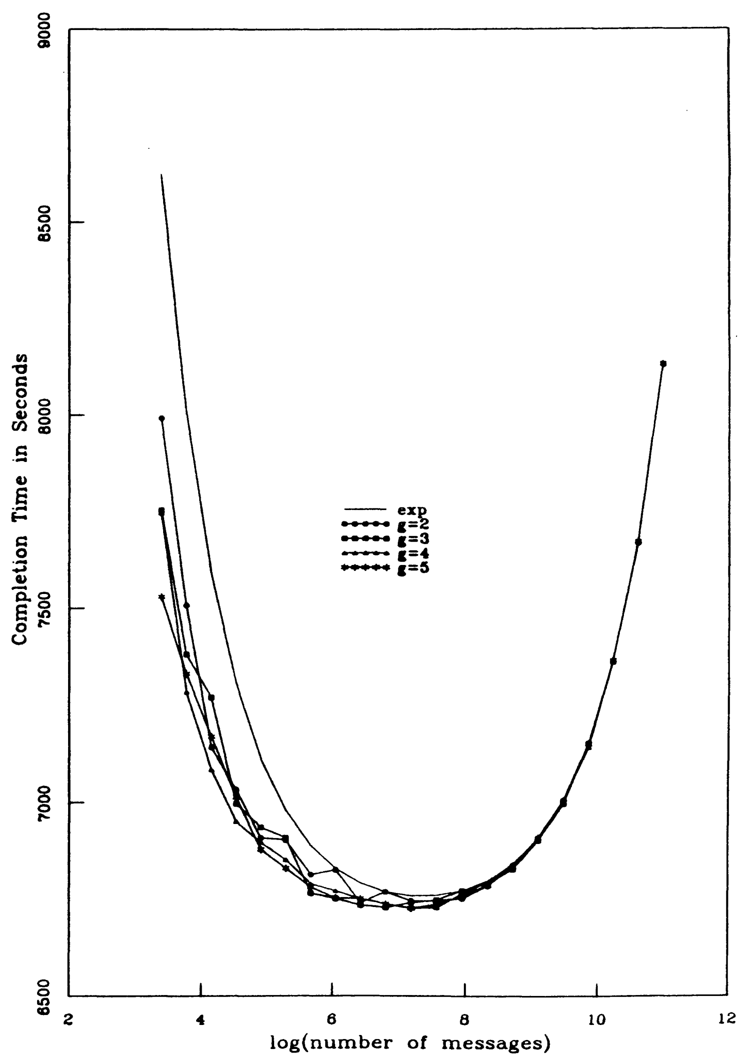

Figure 7.2 : Simulation Results with p=6

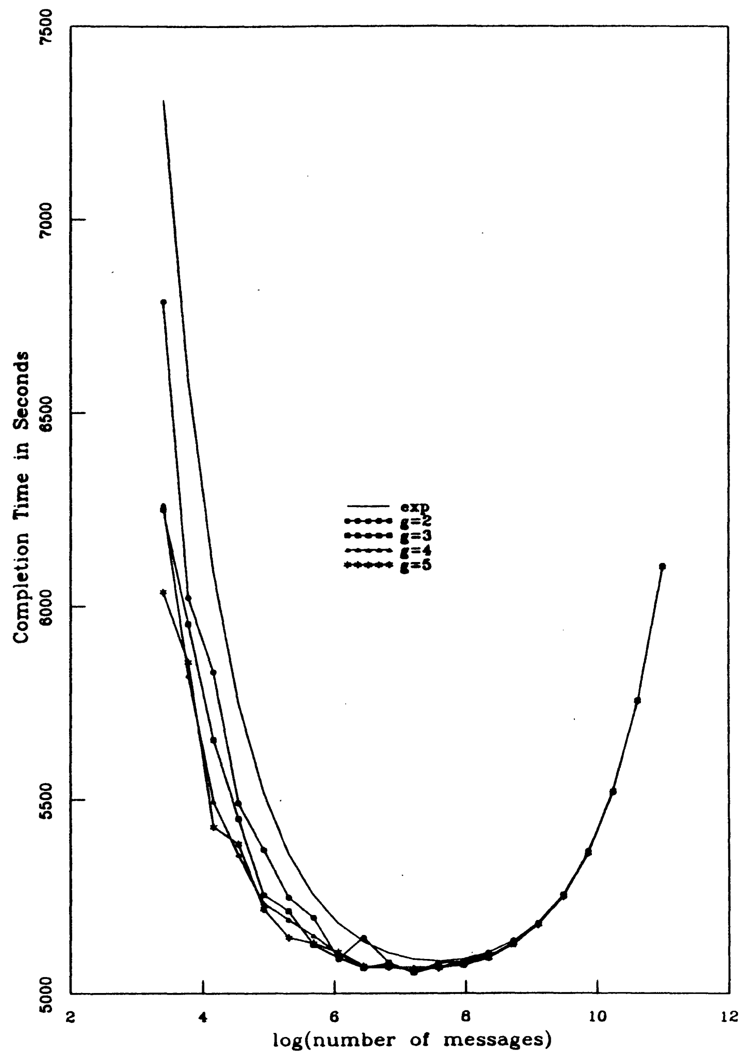

Figure 7.3 : Simulation Results with p=8

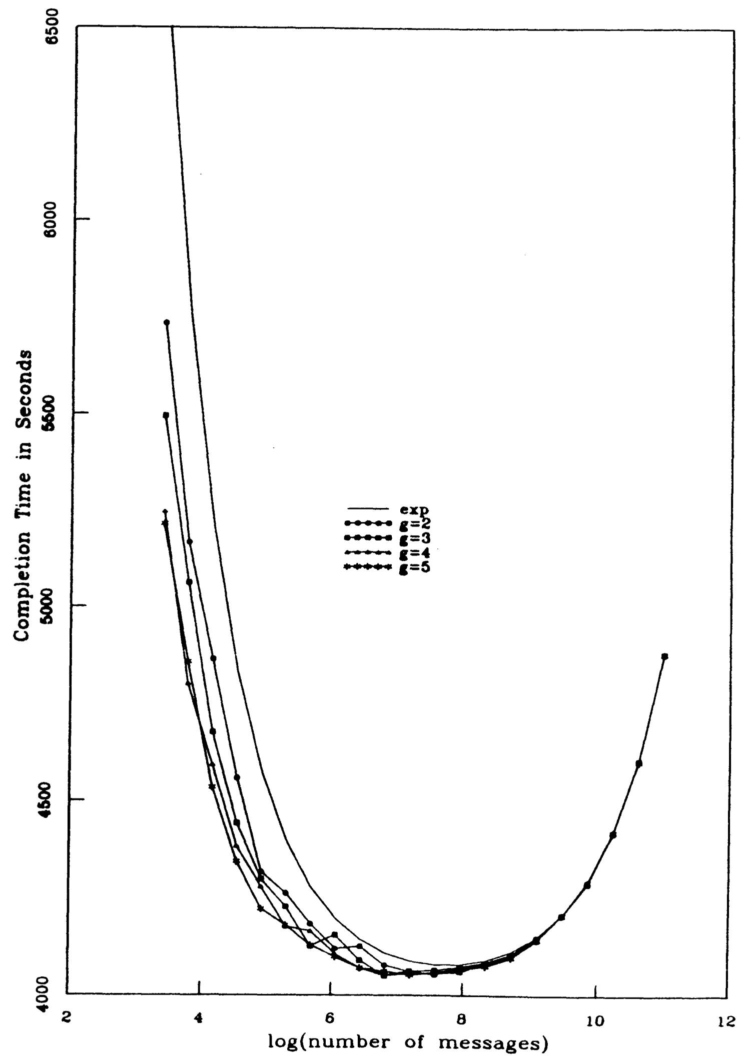

Figure 7.4 : Simulation Results with p=10

8 Conclusion and Future Research

In this paper, our objective has been to describe a new computational method for solving certain complex numerical problems arising in operations research and statistics and to find an optimal way to divide our problem into equal size subproblems. In particular, we minimize the total elapsed time required to compute the solution. However, alternative criteria may also be considered. For example, there may be some cost functions associated with the number of machines we use to work on the problem in parallel. Also, there may be another cost function for the expected completion time of the problem. Our objective is to minimize the total expected cost to complete the task.

In Section 5, $\mathcal{T}_{p,n-p+1}$ was defined to be the time for the loosely coupled parallel system to complete $n-p+1$ subproblems. That is, at the time $\mathcal{T}_{p,n-p+1}$, one of the processors completed a subproblem, and there were no more subproblems to be sent. The time to complete the problem was therefore equal to $\mathcal{T}_{p,n-p+1}$, plus the maximum of the time for the other $p-1$ processors to complete their last subproblems. In practice, at time $\mathcal{T}_{p,n-p+1}$, one may want to resend some of the subproblems that are being worked on by the other processors. The parent only accumulates the results of those subproblems from the first child who reports back. By resending subproblems, time will not be wasted in waiting for a busy machine to complete its last subproblem. It may be useful to extend the model in this paper to allow for resending of subproblems.

As mentioned in Section 1, the network is not as reliable as one would wish, and systems sometimes crash for unanticipated reasons. If one of the child "dies" prematurely, all work done by that child, since it last reported results to the parent, is lost and must be redone. One may want to extend the model in Section 5 to include this case.

So far, it has been assumed that the problem size is known, and the problem is divided into equal size subproblems. In practice it may not be possible to know the size of a problem beforehand. Moreover, even if the problem size is known, one may not want to divide it into equal size subproblems. It is more efficient to send larger messages at the beginning. As the amount of remaining work decreases, one would want to decrease the message size. This would minimize the total overhead time for the whole problem and would also reduce the end effect in which one must wait for the slowest processor to finish. The goal is, therefore, to develop a dynamic control policy in which the transmission of work to the individual processors is a function of the environment state or the time taken by each processor to complete its work if the actual environment state is unobservable.

References

[1] Bal, H. E., Steiner, J. G. and Tanenbaum, A. S. (1989) Programming languages for distributed computer systems *ACM Computing Surveys*, **21**,261-322.

[2] Baudet, G. M. (1978) Asynchronous Iterative Methods for Multiprocessors. *Journal of ACM,* Vol 25, No. 2, 226-244.

[3] Carriero, N. and Gelernter, D., (1989), How to write parallel programs: A guide to the perplexed, *ACM Computing Surveys*, **21**, 323-358.

[4] Cox, D. R. (1962) *Renewal Theory.* John Wiley, New York.

[5] De Finetti, B. (1937) Foresight: Its logical laws, its subjective sources. *Ann. Inst. Henri Poincare,* **7**, 1-68.

[6] Eddy, W. F. and Schervish, M. J. (1986) Discrete-Finite inference on a network of vaxes. *Computer Science and Statistics: Proceedings of the 18th Symposium on the Interface,* ed. T. J. Boardman, Washington, DC: American Statistical Association.

[7] Gardner, T. J., Gerard, I. M., Mowers, C. R., Nemeth, E., and Schnabel, R. B. (1986) DPUP: A distributed processing utilities package. *ACM SIGNUM Newsletter,* **21**, Number 4, 5-19.

[8] Gelfand, A. E. and Smith, A. F. M. (1990), Sampling-based approaches to calculating marginal densities, *Jour. Am. Stat. Ass.*, **85**, 398-409.

[9] Greenhouse, J. B., Kass, R. E., Lam C. Y. T. and Tsay, R. S. (1990) A hierarchical model for serially correlated data : analysis of biological rhythm data. Technical report 90-13. Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor.

[10] Hewitt, E. and Savage, L. J. (1955) Symmetric measures on cartesian products. *Transactions of the American Mathematical Society,* **80**, 470-501.

[11] Karlin, S and Taylor, H. M. (1975) *A first course in Stochastic Processes.* Second Edition, Academic Press.

[12] Lam, C. Y. T. and Lehoczky, J. P. (1991) Superposition of renewal processes. To appear in *Advances in Applied Probability.*

[13] Lam, C. Y. T. (1990) Limiting distribution of remaining life and current life in the superposition of renewal processes. Technical report 90-12. Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor.

[14] O'Leary, D. P. and Stewart, G. W. (1985) Dataflow algorithms for parallel matrix computations. *Communications of the ACM,* **28**, 840-853.

[15] Olson, R. E. and Salop A. (1977) Charge-transfer and impact-ionization cross sections for fully and partially stripped positive ions colliding with atomic hydrogen. *Phys. Rev. A,* 16(2):531-541.

[16] Percus, O. E. and Kalos M. H. (1987) *Random Number Generators for Ultracomputers.* Ultracomputer note 114, New York University, Division of Computer Science, 251 Mercer Street, New York, NY 10012.

[17] Schervish, M. J. (1988) Applications of Parallel Computation to Statistical Inference. *Journal of the American Statistical Association,* Vol 83, No. 404, 976-983.

[18] Schervish, M. J. and Tsay, R. S. (1988) Bayesian modeling and forecasting in large scale time series. To appear in *Bayesian Analysis of Time Series and Dynamic Models,* J. C. Spall, Ed., Marcel Dekker, New York.

[19] Whiteside, R. A. and Leichter, J. S. (March 1988) *Using Linda for supercomputing on a local area network.* Yale University, Dept. Comp. Sci. Tech. Memo.