NASA Contractor Report 178150

ICASE REPORT NO. 86-45

ICASE

DYNAMIC REMAPPING OF PARALLEL COMPUTATIONS WITH VARYING RESOURCE DEMANDS

David Nicol
Joel Saltz

Contract Nos. NAS1-17070, NAS1-18107
July 1986

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

Dynamic Remapping of Parallel Computations with Varying Resource Demands

David M. Nicol
Joel H. Saltz

Institute for Computer Applications in Science and Engineering

Abstract: A large class of computational problems are characterized by frequent synchronization, and computational requirements which change as a function of time. When such a problem must be solved on a message passing multiprocessor machine, the combination of these characteristics lead to system performance which decreases in time. Performance can be improved with periodic redistribution of computational load; however, redistribution can exact a sometimes large delay cost. We study the issue of deciding when to invoke a global load remapping mechanism. Such a decision policy must effectively weigh the costs of remapping against the performance benefits. We treat this problem by constructing two analytic models which exhibit stochastically decreasing performance. One model is quite tractable; we are able to describe the optimal remapping algorithm, and the optimal decision policy governing when to invoke that algorithm. However, computational complexity prohibits the use of the optimal remapping decision policy. We then study the performance of a general remapping policy on both analytic models. This policy attempts to minimize a statistic $W(n)$ which measures the system degradation (including the cost of remapping) per computation step over a period of n steps. We show that as a function of time, the expected value of $W(n)$ has at most one minimum, and that when this minimum exists it defines the optimal fixed-interval remapping policy. Our decision policy appeals to this result by remapping when it estimates that $W(n)$ is minimized. Our performance data suggests that this policy effectively finds the natural frequency of remapping. We also use the analytic models to express the relationship between performance and remapping cost, number of processors, and the computation's stochastic activity.

This research was supported by the National Aeronautics and Space Administration under NASA Contract Numbers NAS1-17070, NAS1-18107, while the authors were in residence at ICASE, Mail Stop 132C, NASA Langley Research Center Hampton VA 23665.

N86-30379 #

1. Introduction

Many computational problems assume a discrete model of a physical system, and calculate a set of values for every domain point in the model. These values are often functions of time, so that it is intuitive to think of the computation as marching through time. When such a problem is mapped onto a message passing multiprocessor machine or a shared memory machine with fast local memories, regions of the model domain are assigned to each processor. The running behavior of such a system is often characterized as a sequence of steps, or iterations. During a step, a processor computes the appropriate values for its domain points. At the step's end, it communicates any newly computed results required by other processors. Finally, it waits for other processors to complete their computation step, and send it data required for the next step's computation.

The computational work associated with each portion of a problem's subdomain may change over the course of solving the problem. This may be true because the behavior of the modeled physical system may change with time. The distribution of computational work over a domain may also change in problems without explicit time dependence. For example, during the course of solving a problem, more work may be required to resolve features of the emerging solution. Since time stepping is often used as a means for obtaining a steady state solution, there is considerable overlap between the above mentioned categories. We call these types of problems *varying demand distribution problems.* Because of the synchronization between steps, the system execution time during a step is effectively determined by the execution time of the slowest, or most heavily loaded processor. We can then expect system performance to deteriorate in time, as the changing resource demand causes some processor to become proportionally overloaded. One way of dealing with this problem is to periodically redistribute, or remap load among processors.

Changing distributions of computational work over a domain arise through the use of adaptive methods in the solution of hyperbolic partial differential equations. These solutions place extra grid points in some regions of the problem domain in order to resolve all features of the solution to the same accuracy [7],[8],[20],[33], [24],[45]. A number of studies have investigated methods for redistributing load in message passing multiprocessors for this type of problem [6],[21],[22]. Changing distributions of computational work can also occur when vortex methods are applied to the numerical simulation of incompressible flow fields. In these methods, invicid fluid dynamics is modeled by parcels of vorticity which induce motion in one another [1], [27]. The number of vortices corresponding to a given region in the domain varies during the course of the solution of a problem. Methods for dynamically redistributing work in this problem have been investigated [2].

In multirate methods for the solution of systems of ordinary differential equations[46], different variables in the system of equations are stepped forward with different timesteps. The size of the timesteps in the system is generally equal to that of a globally defined largest timestep divided by an integer. The size of the different timesteps utilized may vary during the course of solving the problem, and hence the computational work associated with the integration of a given set of variables may change. Another class of problems which may have varying resource demands are adaptive methods for solving elliptic partial differential equations where iterations on a sequence of adaptively defined meshes are carried out [4],[28],[10],[3],[47]. Generally both the total amount of computational work required by each of the meshes and the distribution of work within the domain changes as one moves from one mesh to the next. There are also non-numerical parallel computations which can exhibit varying computational requirements. In time driven discrete event simulations [18], one simulates the interactions over time of a set of objects. Responsibility for a subset of objects is assigned to each processor. Over the course of the simulation, subsets of objects may differ in activity, and hence in their computational requirements. This problem may also arise in parallel simulations which proceed in a loosely synchronized manner, such as those described in [11], [13], [26], [32], [34], [30].

There are two fundamentally different approaches to such remapping. The decentralized load balancing approach is usually studied in the context of a queueing network [17],[19],[29],[40],[41],[43],[44]. Load balancing questions are then focused on "transfer policies", and "location policies"[17]. A transfer policy governs whether a job arriving at a service center is kept or is routed elsewhere for processing. A location policy determines which service center receives a transferred job. Decentralized balancing seems to be the natural approach when jobs are independent, and a global view of balancing would not yield substantially better load distributions.

However, a large class of computations is not well characterized by a job arrival model, and it may be advantageous to take a global, or centralized perspective when balancing. We will call a global balancing mechanism "mapping" to distinguish it from the localized connotations of the term load balancing. A centralized mapping mechanism can exploit full knowledge of the computation and its behavior. Furthermore, dependencies between different parts of a computation can be complex, making it difficult to dynamically move small pieces of the computation from processor to processor in a decentralized way. Global mapping is natural in a computational environment where other decisions are already made globally, e.g. convergence checking in an iterative numerical method. Yet the execution of a global mapping algorithm may be costly, as may the subsequent implementation of the new workload distribution. A number of authors have considered global mapping policies under varying model assumptions, for example, see [12], [16], [23], [42] , [5] , [9]. A comparison between global and decentralized mapping strategies is reported in [25].

For the types of problems we describe, *remapping* the load with a global mechanism is tantamount to repartitioning the set of model domain points in regions, and assigning the newly defined regions to processors. A mapping algorithm of this sort is studied in [6] and the performance of this mapping algorithm in the context of vortex methods is investigated in [2]. Decision policies determining *when* a load should be remapped become quite important. The overhead associated with remapping can be high, so it is important to balance the overhead cost of remapping with the expected performance gain achieved by remapping. While this is a generic problem, the details of load evolution, of the remapping mechanism, and of various overhead costs are system and computation dependent. In order to study general properties of remapping decision policies, it is necessary to *model* the behavior of interest, and evaluate the performance of decision policies on those models. Remapping is treated this way in [31] under the assumption that the parallel computation has multi-phase behavior. In the present paper we consider remapping of varying demand distribution problems using two different stochastic models. An overview by the present authors introducing some of the ideas developed in this paper is presented in [37].

The evaluation of policies for memory management in multiprogrammed uniprocessor systems has successfully employed a number of stochastic models to reflect the memory requirements of typical programs [15], [39]. In these models, the principal of memory reference locality plays a central role. Evaluating policies for scheduling a remapping in message passing machines is somewhat similar in spirit to the evaluation of paging algorithms in multiprogrammed uniprocessor systems. The principal of locality that we attempt to capture here is the locality of resource demand. The computational work corresponding to the problem region assigned to a given processor will often vary in a gradual fashion. In this paper we consider two models which describe this evolution probabilistically. The first model assumes that the computational requirements of each partition region behaves as a Markov chain, independently of any other region; this is called the Multiple Markov chain (MUM) model. The MUM model has the advantage of being analytically tractable in several ways. However, for many problems it may not be reasonable to assume independence in load evolution between partition regions. We address this issue with a second, less tractable model, the Load Dependency (LD) model. These models attempt to capture the dynamics by which the distribution of computational load changes in time, and are characterized by a small number of important parameters. Through the use of these load evolution models, we are able to evaluate policies for deciding when load should be remapped.

In this paper we propose a policy which attempts to minimize the statistic $W(n)$ measuring the average (over n steps) system degradation per step (including the cost of a remapping). We show that as a function of n, the expected value of $W(n)$ has at most one minimum. If the minimum occurs at $\hat{n}$, then we show that the optimal fixed-interval remapping policy is to remap every $\hat{n}$ steps. These results support the general philosophy of the heuristic. Empirical studies based on our models show that the heuristic is effective on different models of load behavior. We also describe analytical work which looks at the relative effects the model parameters on remapping frequency.

This paper is organized as follows. Section 2 describes the Multiple Markov chain model of computational variation, and shows how it captures the drifting computational load phenomenon. Section 3 discusses optimal algorithms for both determining *how* to remap, and for determining dynamically *when* to remap. The high computational expense of computing the optimal decision policy leads us to define a simple inexpensive heuristic policy in section 4, where we also discuss the heuristic's performance. Section 5 then presents an analysis of the statistic used by the heuristic, and shows analytically that the heuristic is well motivated. Section 6 presents an alternate model of load variation, and shows how the statistic of section 4 is also effective with this model. Section 7 summarizes our results, and the appendices treat technical issues in detail.

2. Multiple Markov Chain Model

A processor is assigned a region of the problem domain. At each step, the processor needs to perform a certain amount of computation related to that region. Upon completion of this computation, it may send messages to other processors, reporting newly calculated results. Before advancing to the next step, the processor then synchronizes as required by the computation. The computational requirements of the region may vary gradually from step to step. The MUM model characterizes this variation by modeling the work demands on each processor using a Markovian birth-death process. The state s of the chain is a positive integer describing the execution time of the processor at a step. We also assume that $s \leq L$ for some L. The transition probabilities out of s reflect the principle of locality, where all one step transitions are to neighboring states. When s is between 2 and $L-1$, the probability that the chain will make a one step transition to state $s+1$ is $p/2$, the probability of a one step transition to $s-1$ is $p/2$ and the probability that the chain will not make a transition is $1-p$. For $s = 1$ or $s = L$, the state remains the same with probability $1 - p/2$, and moves to the single neighboring state with probability $p/2$.

The processors are modeled by a collection of independent, identically distributed Markov chains. We let $T_j(n)$ represent the time required by the *jth* processor to complete the *nth* step. Assuming N processors, the time required for the system as a whole to complete the *nth* step is given by

$$T_{\max}(n) = \max_{1 \leq j \leq N}\{T_j(n)\}.$$

The average processor execution time during the *nth* step is

$$\overline{T}(n) = \frac{1}{N}\sum_{j=1}^{N} T_j(n).$$

Then the average processor utilization during the *nth* step is

$$\rho(n) = \frac{\overline{T}(n)}{T_{\max}(n)}$$

and the average period of time that a processor is idle waiting for other processors to finish step n is consequently given by $T_{\max}(n)-\overline{T}(n)$. Finally, we assume that the states of every chain at step 0 are identical.

An intuitive feel for the behavior of the MUM model is gained by examining graphs of particular performance measures. Figure 1a depicts the behavior of the MUM model for varying numbers of chains. The performance shown is the average (per step) processor utilization as a function of step, taken over 500 simulations or sample paths, where $p = 0.5$ and each chain has 19 states. Performance declines more quickly and to a lower level as one simulates a problem with an increasingly large number of independent processors. For a given number of chains, the performance decline arises from the fact that the expected value of $\overline{T}(n)$ remains relatively constant as n increases, while the expected value of $T_{\max}(n)$ increases in n. Figure 1b depicts the performance of single sample paths of the MUM model using varying numbers of chains, where as before $p = 0.5$ and each chain has 19 states. Note that the decline in performance as a function of step is true only in the sense of comprising a long term trend; each curve has many local maxima and minima. This point is particularly important, because any dynamic real time remapping policy mechanism is concerned with the current single sample path defined by the computation's execution.

3. Optimal MUM Model Load Balancing

We now consider the problems associated with remapping a computation described by a MUM model. There are two basic issues, *how* to remap, and *when* to remap. The tractability of the MUM model allows us to describe optimal policies for both of these issues. We treat the remapping mechanism by demonstrating that under certain regularity conditions, the obvious technique of assigning equal loads to each processor is optimal. However, we will note that deviation from these regularity conditions can cause this technique to be sub-optimal. We treat the remapping decision problem in the framework of a Markov decision process. We are then able to *symbolically* express an optimal remapping decision policy in terms of a solution to a set of equations. However, the complexity of solving these equations grows exponentially in the size of the problem, so that this technique is not useful for any but the smallest sized problems. This realization leads us in section 4 to consider a sub-optimal but computationally simple decision heuristic.

Consider the situation where the computation has completed step n, and we wish to redistribute the computational load. For every processor j, we can determine its load during step n, and can determine that step n required $T_j(n)$ processing time. We then suppose that it is possible to repartition the problem domain into N regions with equal (or near equal) computational demand during step n. This new partition is implemented for step $n + 1$. We call this mechanism *average value* remapping. While the optimality of this policy may seem trivially obvious, later reflection shows that this may not always be the case. For example, if one processor's computational load tends to evolve faster than others', we may want to assign it *less* than the average value load upon remapping. Called padding [38], such a policy accepts slightly decreased immediate performance in order to forestall the active processor from quickly degrading performance. Nevertheless, under certain conditions (which exclude this difficulty), we can show that the average value load policy is optimal. The proof of this claim is somewhat technical, and is provided in Appendix A. To formally state the claim, we define

$$T_j(n+d,s_j) = T_j(n+d) \text{ given that } T_j(n) = s_j$$

and

$$T_{\max}(n+d,s_1,\cdots,s_N) = \max_{1\le j\le N}\{T_j(n+d,s_j)\}.$$

Then

THEOREM 3.1 : Assume that the Markov chains are homogeneous, and unbounded (no maximum nor minimum state), and that the transition probabilities are unaffected by remapping. Suppose $\sum_{j=1}^{N} T_j(n,s_j) = K$. For every $i = 1,2,\cdots,N$, define

$$a_i = \begin{cases} \lfloor K/N \rfloor + 1 & \text{if } i < K \bmod N \\ \lfloor K/N \rfloor & \text{if } i \ge K \bmod N \end{cases} .$$

Then for every $d \ge 0$ we have

$$E[T_{\max}(n+d,s_1,\cdots,s_N)] \ge E[T_{\max}(n+d,a_1,\cdots,a_N)].$$

□

Theorem 3.1 is a strong statement of the average value remapping policy's optimality. It says that employing this policy at step n minimizes the expected execution time of *every* future step, if no further remapping is permitted after step n.

The second major remapping issue is *when* to remap. If there were no cost associated with remapping, then we would remap frequently to maximize processor utilization. With an increasing cost of remapping, we will want to remap less and less often, to better amortize the cost of remapping over a larger number of computation steps. We can employ a Markov decision process framework [35] to formally state the remapping decision issue. Within this framework we are able to symbolically describe the optimal remapping decision policy.

Our notation concerning Markov decision processes is taken largely from [35]. Consider a stochastic process whose state we observe at each of a sequence of times $t = 0, 1, \cdots$. Let I be the set of all possible states. At each time j, the state of the process is discerned to be some $s \in I$. Then a decision is made, choosing some action a from a finite set A; the choice of action a while in state s incurs a cost $c(s,a)$. $c(s,a)$ may be random; we assume that $E[c(s,a)]$ is finite for all states s and actions a. The decision process then passes into another state. The probability $p_{sq}(a)$ of passing into state q from s is dependent on the action a chosen in state s. The expected total cost of a decision policy is the expected sum of the costs incurred at each decision step. An *optimal* decision policy minimizes the expected total cost.

We restrict our attention to the class of *stationary* decision policies, those policies which are deterministic functions of the discerned state. The following useful theorem concerning optimal stationary decision policies is given by [35].

Let $V(s)$ be the expected total cost of the process which starts in state s, and which is governed by the optimal stationary policy. Then,

$$V(s) = \min_{a \in A} \left\{ c(s,a) + \sum_{q \in I} p_{sq}(a) V(q) \right\}, \tag{1}$$

□

The function $V(s)$ is known as the *optimal cost function.* From state s, the optimal stationary decision is the choice of action which minimizes the right hand side of equation (1). We now formulate the MUM remapping decision problem in terms of a Markov decision process.

The state of our decision process at step n is the vector of chain states $<T_1(n), \cdots, T_N(n), n>$ along with the step number, n. For any decision state $S = <s_1, \cdots s_N, n>$, let $J(S)$ denote the set of states reachable from $<s_1, \cdots s_N, n>$ in one step , and let $A(S)$ denote the system state achieved by performing average value remapping on S. Each $T = <t_1, \cdots, t_N, n+1> \in J(<s_1, \cdots s_N, n>)$ has a transition probability

$$Prob\{T \mid S\} = \prod_{i=1}^{N} Prob\{t_i \mid s_i\}$$

where $Prob\{t_i \mid s_i\}$ is the probability of chain i passing from state s_i into state t_i in one step. The execution cost of state S is simply

$$EC(S) = \max_{1 \le i \le N}\{s_i\},$$

and the delay cost of doing a remapping is C. This cost includes both the communication costs and the computational overhead required for performing a remapping operation. This delay cost C is in general expected to be a function of the state S, although we will suppress this dependence here for the sake of simplicity. We can observe the system state S only by allowing the system to execute the step during which state S is achieved. Thus the decision process state encodes the performance of the last step's execution. If we remap from state S, this assumption implies that the next system state will be a member of $J(A(S))$. Equation (1) may now be written as

$$V(S) = \min \begin{cases} EC(S) + C + \sum_{U \in J(A(S))} Prob\{U\}V(U) \\ EC(S) + \sum_{U \in J(S)} Prob\{U\}V(U) \end{cases} \tag{2}$$

where the top equation on the right-hand side is the cost function associated with remapping, and the bottom equation is associated with not remapping. According to the theory of Markov decision processes, the optimal decision to make from state S is the decision which minimizes the right-hand side of equation (2). If the number of steps taken by the system is some random variable with finite mean, then the system of equations given by (2) can be solved, at least in theory. In practice, we have not found any means of substantially reducing the enormous state space addressed by these equations, and so the exact solution of the optimal remapping decision policy is restricted to relatively small systems. Furthermore, the optimal Markov decision process as formulated here is applicable only to the MUM load model. It depends on foreknowledge of the precise probabilistic structure of the MUM model and on the independence of the MUM model chains. All of these issues make this decision model an unrealistic candidate as a decision mechanism. In the next section we introduce a heuristic which avoids these difficulties.

4. Stop At Rise Decision Policy

Because of the difficulties inherent in the optimal MUM model remapping decision policy, we consider a sub-optimal heuristic called the Stop At Rise (SAR) policy. The SAR policy attempts to minimize the average time per step that a processor spends inactive due to synchronization or remapping delay. In this section we describe SAR and present performance data.

Any remapping decision policy must attempt to reconcile the costs of remapping against the increasing execution costs suffered by not remapping. However, the increasing execution costs are *future* costs, and are consequently uncertain. The optimality of the MUM Markov decision process policy stems from its explicit consideration of all possible future activity and costs. A real policy cannot afford this computational luxury. Instead, we turn to SAR, a "greedy" policy, which attempts to remap so that the average long-term processor idle time since the last remapping is minimized. This calculation of idle (or wasted) time includes the time C spent in one remapping operation. Supposing that the load was last remapped n steps ago (say, just before step 1), the average processor idle time per step that we achieve by remapping immediately is denoted $W(n)$, and is given by

$$W(n) = \frac{\sum_{j=1}^{n}(T_{\max}(j) - \overline{T}(j)) + C}{n}.$$

$W(n)$ explicitly embodies two of the costs a remapping policy must manage. C is the delay cost of remapping once, $\sum_{j=1}^{n}(T_{\max}(j) - \overline{T}(j))$ is interpreted as the cost of not remapping. However, under the greedy philosophy we have adopted, this latter cost is a past cost of not remapping, rather than a future cost.

It is instructive to consider the behavior of $W(n)$ as a function of n. Figure 2a plots $W(n)$ for single sample paths when $p = 0.5$, $C = 8.0$, and the chains have 19 states. Paths from 2, 8, and 32 chain systems are shown. Figure 2b plots $E[W(n)]$ under these same parameter values. Both graphs show $W(n)$'s marked propensity to drop, be minimized over some section of its domain, and then begin and continue to rise. This behavior can be explained in terms of $W(n)$'s definition. There is a tendency for $W(n)$ to decrease in n, as increasing n amortizes the cost C over a larger number of computation steps. But there is also a tendency for $W(n)$ to increase in increasing n, as we may expect $\sum_{j=1}^{n}\overline{T}(j)/n$ to remain relatively constant, and $\sum_{j=1}^{n}T_{\max}(j)/n$ to increase with n. The tendency to decrease dominates initially, but has less effect on $W(n)$ as n grows. The tendency to increase then becomes predominate. We once again note that this tendency is an *expected* tendency. The precise behavior of $W(n)$ for a given sample path will vary with that sample path. Whereas it is reasonable to expect that $E[W(n)]$ has a single local minimum (a topic we discuss in section 5), the values of $W(n)$ for a given sample path may exhibit multiple local minima. The significant implication of these observations is that we do not have the option of remapping at a time step $\hat{n}$ with any assurance that $W(\hat{n})$ will minimize the statistic. We can however remap once the first *local* minimum of $W(n)$ is detected. We thus choose to remap at the first step $\hat{n}$ after the last remapping such that $W(\hat{n}) > W(\hat{n}-1)$. This policy of remapping when a local minimum of $W(n)$ is first detected is labeled the SAR (Stop At Rise) policy.

We studied the performance of the SAR policy by comparing it to three other policies: the optimal policy, the "remap every m steps" policy, and the policy which never remaps. It is possible to compute the expected time required to complete small sized problems when the optimal Markov decision policy is utilized to decide when to remap. Figure 3 compares a performance metric for the Markov decision policy, SAR, and a non-remapped system for three chains, 100 steps, and various remapping costs. The SAR data is the average of 500 simulation runs for each value of *C*. As the remapping cost increases, the discrepancy between the performance obtained through the optimal decision policy and SAR increases. With increasing remapping cost, both the performance of the optimal decision policy and of SAR approach the performance obtained when no remapping is performed.

The performance metric used in figure 3 for all policies depicted in that figure is an estimate of processor utilization: the *ratio* of the *expected* $\sum_{j=1}^{n}\overline{T}(j)$ to the *expected* total time spent by the system to solve the problem, including the cost of all remappings. This measure is useful in figure 3 as it is straightforward in the case of the optimal policy to calculate the expected time required to complete a problem as well as the expected $\sum_{j=1}^{n}\overline{T}(j)$. For all subsequent figures, performance data is obtained by simulation, and the easily computed average performance over all simulations is utilized, i.e. the *mean* of the *ratio* of the $\sum_{j=1}^{n}\overline{T}(j)$ to the total time spent by the system to solve the problem, including the cost of all remappings. Both performance measures were computed for all simulations, and found to differ from each other by less than one percent.

One simple but intuitive remapping policy is the "remap every *m* steps" policy, or *fixed interval* policy. This policy is insensitive to statistical variations in a system's performance, and requires pre-run-time analysis to determine an effective value of *m*. However, we might well choose to employ a fixed interval policy if it is costly to measure system performance at every step. In this case, we would attempt to choose *m* to optimize the system's *expected* performance. In figure 4, we compare the performance obtained through the use of: (1) SAR , (2) the fixed interval policy for a wide range of values of *m*, and (3) not remapping at all. The performance obtained in a system using the MUM model with eight independent processors is depicted. Each problem consists of 400 steps, each data point is obtained through 200 simulations, and remapping costs of 2 and 8 are assumed. For the SAR policy, we plotted performance against the calculated average number of steps between consecutive remappings. In the fixed interval policy, we plotted performance against *m*, the fixed number of steps between remappings. The number of steps between remappings has no meaning when no remapping is done, the performance obtained when no remapping occurs is plotted as a straight horizontal line to facilitate comparison with the other results. The calculation of the performance obtained through the use of the optimal Markov decision policy is not practical in this case due to the long run times and large memory requirements that would be required.

It is notable that SAR's performance was comparable and in fact slightly higher than that obtained by remapping at the optimal fixed interval. The average number of elapsed steps between SAR remappings corresponds closely to the optimal fixed interval remapping policy. Similar results were obtained in other cases using the MUM load model. These results are encouraging for two reasons. Since SAR adapts to statistical variations in the system's behavior, we would hope that it can outperform a non-adaptive policy. Our data shows that SAR outperforms the **optimal** fixed interval policy. Secondly, SAR appears to find the "natural frequency" of remapping for a given remapping cost. While the exact number of steps between remappings may vary with the system's sample path, the average number of steps between remappings is close to that of the optimal fixed interval policy. Note also that the performance obtained by SAR is markedly superior to the performance obtained when no remapping is performed. From extensive simulation results not presented here, we found that the difference between the performance obtained by SAR and the performance obtained when no remapping is performed increases with the number of chains. This is consistent with the observed results in figures 3 and 4.

In the face of uncertainity about future problem behavior, it is reasonable to design a remapping decision policy which optimizes performance locally in time. The SAR policy does this by attempting to minimize $W(n)$, a statistic which measures performance since the last remapping. Performance experiments show that the SAR policy effectively finds the "natural frequency" of remapping as a function of the rate at which resource demand changes, and the cost of remapping. As such, SAR is a promising policy for real remapping situations. In section 6 we demonstrate that the SAR policy can also be effectively employed with computational models other than the MUM model.

5. Analysis of $E[W(n)]$

In this section we analyze the behavior of $E[W(n)]$. First we show that if the difference between expected maximum load and average load is an increasing function, then $E[W(n)]$ has at most one local minimum as a function of n. We then show that if $E[W(n)]$ has a minimum at $\hat{n}$, then the fixed-interval decision policy of remapping every $\hat{n}$ steps minimizes the expected loss per step of any fixed-interval policy, including the policy which prohibits remapping. These results are independent of the MUM model; we then present conditions on MUM parameters for which these results apply, and show that MUM assumptions allow us to compute $E[W(n)]$ exactly for all n. Finally, we analyze the effects that problem behavior and remapping costs have on optimal (fixed-interval) remapping frequency. This analysis gives us both a qualitative and a quantitative grasp of the relationships between the different factors involved in a decision to remap. Furthermore, since our SAR data indicates that SAR adaptively finds the optimal fixed-interval remapping frequency, we expect these relationships to hold true for SAR.

We will first show that under a reasonable hypothesis, the expected time wasted per step, $E[W(n)]$, can have no more than one local minimum. Moreover, the step $\hat{n}$ at which the minimum of $E[W(n)]$ occurs is a monotone increasing function of the cost of remapping C. In order to minimize wasted time per step, if remapping becomes more costly, the number of steps between successive remappings must increase. After remapping we assume that the expected partitioning of load between processors becomes increasingly uneven. The hypothesis used in the theorem formalizes this and supposes that the expected difference between the maximum order statistic and the average is monotone increasing with n.

THEOREM 5.1 : Suppose $E_i = E[T_{\max}(i)] - E[\overline{T}(i)]$ is monotone increasing in i. Then $E[W(n)]$ has at most one minimum and if this minimum exists, it is a monotone increasing function of the remapping cost C.

PROOF: Let $\delta(n) = E[W(n+1)] - E[W(n)]$, and let

$$\kappa(n) = nE_{n+1} - \sum_{i=1}^{n} E_i.$$

From the definitions of $\delta(n)$, $E[W(n)]$ E_i and $\kappa(n)$, it is straightforward to show that

$$\delta(n) < 0 \quad \textit{iff} \quad \kappa(n) < C$$

and that

$$\delta(n) > 0 \quad \textit{iff} \quad \kappa(n) > C.$$

Using the assumption that E_i is monotone increasing, we first show that $\kappa(n)$ is monotone increasing. This follows, since

$$\kappa(n+1) - \kappa(n) = \left[(n+1)E_{n+2} - \sum_{i=1}^{n+1} E_i\right] - \left[nE_{n+1} - \sum_{i=1}^{n} E_i\right]$$

$$= (n+1)(E_{n+2} - E_{n+1}) \geq 0.$$

For the sake of contradiction, assume that there are local minima at $\hat{n}_1$ and at $\hat{n}_2$; without loss of generality we suppose that $\hat{n}_2 > \hat{n}_1$. Since $\hat{n}_1$ is a local minimum, $E[W(\hat{n}_1+1)] > E[W(\hat{n}_1)]$, so that $\delta(n_1) > 0$ and hence $\kappa(\hat{n}_1) > C$. If $\hat{n}_2$ is also a local minimum, then $E[W(\hat{n}_2)] < E[W(\hat{n}_2-1)]$, so that $\delta(\hat{n}_2-1) < 0$ and hence $\kappa(\hat{n}_2-1) < C$. If $\hat{n}_1 = \hat{n}_2-1$ we have a direct contradiction, since κ cannot be both greater than and less than C at the same point. If $\hat{n}_1 < \hat{n}_2-1$, we have a contradiction as $\kappa(n)$ is a monotone increasing function of n.

Now, if the minimum $\hat{n}$ exists it is the largest n for which $\kappa(n) < C + 1$. Since $\kappa(n)$ is a monotone increasing function of n, it follows immediately that the $\hat{n}$ minimizing $E[W(n)]$ is a monotone increasing function of C.

□

One may expect that following a remapping, the difference in the time per step required by the maximally loaded processor and the average processor will tend to increase in time. Theorem 5.1 states that as long as $E[T_{\max}(i)]$ - $E[\overline{T}(i)]$ is a monotone increasing function of i, then $E[W(n)]$ has at most one minimum. Under ideal circumstances, an existing minimum defines the optimal fixed-interval remapping policy, a fact demonstrated by Theorem 5.2.

THEOREM 5.2 : Suppose that remapping after n steps resets the E_i sequence, so that the expected loss between remappings is $\sum_{i=1}^{n} E_i + C$. If $E[W(n)]$ is minimized at $\hat{n}$, then the optimal fixed-interval remapping policy (including the policy which never balances) is the policy which remaps every $\hat{n}$ steps.

PROOF: The average loss at the *kth* remapping is

$$\frac{k\left[\sum_{i=1}^{n} E_i + C\right]}{kn} = E[W(n)].$$

Thus as $k \rightarrow \infty$, the limiting average loss is simply $E[W(n)]$, which is minimized when $n = \hat{n}$. To see that $E[W(\hat{n})]$ is less than the limiting average loss per step of never remapping, we observe that

$$E[w(n)] = \frac{\sum_{i=1}^{n} E_i}{n}$$

is the average loss per step without remapping, and that

$$E[W(n)] = E[w(n)] + \frac{C}{n}.$$

Since $E[W(n)]$ is increasing and the difference between $E[W(n)]$ and $E[w(n)]$ gets arbitrarily small with increasing n, the fact that $E[W(n)]$ is increasing for $n > \hat{n}$ ensures that

$$E[W(\hat{n})] \leq \lim_{n \to \infty} E[w(n)].$$

□

While recognizing that the reseting assumption upon which Theorem 5.2 rests may not be met in practice, the theorem's statement supports the idea of SAR: seeking to remap when $W(n)$ is minimized.

It is reasonable to investigate the conditions under which the simulated loads produced by the MUM model lead to a sequence of E_i that is monotone increasing. We assume that all chains begin at the "middle" state. We will show that as long as it is relatively unlikely that a processor will require the maximum possible time L to complete a step, we are guaranteed a monotone increasing sequence of E_i. Theorem 5.3 below demonstrates that a sufficient condition that ensures that the situation described above will prevail is that the transition probability p be less than or equal to 2/3.

We state two lemmas, which are used to prove Theorem 5.3. An essential tool used in the statement of one of these lemma's is the theory of stochastic variability (see [36]). A random variable X is said to be *stochastically more variable* than random variable Y, denoted $X \geq_v Y$, if $E[g(X)] \geq E[g(Y)]$ for every increasing convex function g. $X \geq_v Y$ intuitively means that X places more probability on occurrences of high sample values than does Y. For our purposes, a result given in [36] is very important: if $X_1, \cdots, X_k$ is a group of independent random variables, $Y_1, \cdots, Y_k$ is a group of independent random variables, and $X_i \geq_v Y_i$ for $i = 1,2, \cdots, k$, then

$$E[\max(X_1, \cdots, X_k)] \geq E[\max(Y_1, \cdots, Y_k)]. \tag{3}$$

Also, for every $j = 1,2,..., L$, and step n, we denote the probability mass function

$$p_j(s;n) = Prob\{T_j(n) = s\}.$$

The proofs of the following two lemmas are detailed, and have been relegated to Appendix B.

LEMMA 5.1 : Assume that at step 0, all chains are in state $(L+1)/2$, i.e., for all j, $p_j((L+1)/2;0) = 1$; also assume that $p \leq 2/3$. Then for all steps n, we have $p_j(k-1;n) \leq p_j(k;n)$ for $k \leq (L+1)/2$, and $p_j(k+1;n) \leq p_j(k;n)$ for $k \geq (L+1)/2$.

□

Lemma 5.1 simply identifies conditions which ensure that the probability mass function for a chain's state distribution is unimodal at every step. The next lemma shows that if the probability of chain j being in state L is always less than or equal to the probability of its being in any other given state, then $T_j(n)$ is stochastically more variable than $T_j(n-1)$.

LEMMA 5.2 : If for every n,

$$Prob\{T_j(n) = L\} = \min_k \left\{ Prob\{T_j(n) = k\} \right\},$$

then $T_j(n) \geq_v T_j(n-1)$ for all n.

□

Note that if the conditions of Lemma 5.1 are satisfied, then the conditions of Lemma 5.2 are also satisfied. In this case, we also have

THEOREM 5.3 : If $Prob\{T_j(0) = (L+1)/2\} = 1$, and $p \leq 2/3$, then $E[W(n)]$ has at most one local minimum. If the minimum exists, it is a monotone increasing function of the remapping cost C.

PROOF: The condition that $Prob\{T_j(n) = L\} = \min_k \left\{ Prob\{T_j(n) = k\} \right\}$ follows immediately from Lemma 5.1, hence by Lemma 5.2 we have the conclusion that $T_j(n) \geq_v T_j(n-1)$ for all n, and all j. Since the chains are independent, it follows from (3) that

$$E[T_{\max}(n)] \geq E[T_{\max}(n-1)].$$

The average time spent by the processors performing computations during step i is

$$\overline{T}(i) = \frac{1}{N}\sum_{j=1}^{N} T_j(i),$$

so that the expected value of $\overline{T}(i)$ is

$$E[\overline{T}(i)] = \frac{1}{N}\sum_{j=1}^{N} E[T_j(i)].$$

Because we assumed that all chains j are initially in the middle state $(L+1)/2$, and the chains are symmetric, it follows that $E[T_j(i)] = (L+1)/2$ for all i. But since $E[T_{\max}(i)]$ increases in i and $E[\overline{T}(i)]$ is constant, we see that E_i is monotone increasing in i. Our conclusion then follows from Theorem 5.1.

□

It is interesting to note that the result above does not depend on the MUM chains being identically distributed; Theorem 5.3 applies if every chain's transition probability is less than 2/3.

The simplicity of the MUM model allows us to derive an exact expression for $E[W(n)]$. In theory, we could use this expression to find the $\hat{n}$ which minimizes $E[W(n)]$. However, this expression is computationally cumbersome, and does not immediately lend itself to useful interpretation. We therefore also present two more tractable approximations to $E[W(n)]$, and comment on the relationships they show between MUM model parameters.

It is straightforward to compute $E[W(n)]$ for the MUM model. For each processor j, we define the probability state vector describing the distribution of $T_j(m)$ as $\mathbf{p}_j(m) = (p_j(1;m), \cdots ,p_j(L;m))$ We also define $\mathbf{M}$ as the matrix of the Markov chain's one step transition probabilities. The probability state vector for the *mth* step is given by the vector-matrix product

$$\mathbf{p}_j(m)=\mathbf{p}_j(0)\mathbf{M}^m.$$

Note that this expression depends on the distribution of chain j's initial state. The cumulative distribution function for $T_j(m)$ may be written as

$$P_j(s;m) = Prob\{T_j(m) \le s\} = \sum_{i=1}^{s} p_j(i;m).$$

The time required by a N processor system to complete step m is given by the distribution of the maximum order statistic of the processors' states at step m. For every state s and step m, the probability that the maximum state exceeds s is equal to one minus the probability that all processors have a state less than s at step m. The cumulative distribution function for the maximum order statistic at step m is thus

$$\begin{aligned} Prob\{\max_j\{T_j(m)\} > s\} &= 1 - Prob\{T_j(m) \le s, j = 1,2, \cdots ,N\} \\ &= 1 - \prod_{j=1}^{N} P_j(s;m). \end{aligned}$$

It is well known that for any non-negative discrete random variable X, $E[X] = \sum_{a\ge 0} Prob\{X > a\}$. The mean time for the system to compute step m is thus expressed as

$$E[T_{\max}(m)] = \sum_{s=1}^{L} \left[1-\prod_{j=1}^{N} P_j(s-1;m) \right].$$

As mentioned previously in this section, the expected value of $\overline{T}(m)$ is

$$E[\overline{T}(m)] = \frac{1}{N}\sum_{j=1}^{N} E[T_j(m)].$$

Furthermore, $E[T_j(m)]$ may be written as

$$E[T_j(m)]=\sum_{s=1}^{L}\left[1-P_j(s-1;m)\right].$$

We can now derive an expression for $E[W(n)]$. By definition,

$$E[W(n)]=\frac{\sum_{m=1}^{n}\left[E[T_{\max}(m)]-E[\overline{T}(m)]\right] + C}{n}. \tag{4}$$

Substituting the expressions derived above into (4), we obtain

$$E[W(n)] = \frac{\sum_{m=1}^{n}\sum_{s=1}^{L}\left[\frac{1}{N}\sum_{j=1}^{N}P_j(s-1;m)-\prod_{j=1}^{N}P_j(s-1;m)\right] + C}{n}. \tag{5}$$

The expression above may be used to calculate the expected value of the time lost per processor as a function of the number of steps since the last remapping. In principle, we could then *compute* the $n = \hat{n}$ which minimizes expression (5). However, this precise formulation does not give us any insight into the qualitative effects that the MUM model parameters have on $\hat{n}$. To attempt to gain this insight, we considered two approximations for $E[W(n)]$ which better express relationships between the MUM model parameters. Since SAR appears to adaptively find the optimal fixed-interval remapping frequency, we expect that these relationships affect SAR remapping frequency in the same way. The first approximation we describe is asymptotic in the number of processors; the second approximation uses an upper bound on the max order statistic of a symmetric random variable.

Our first approximation assumes that the number of states in a processor's chain is odd, and denotes the "middle" state by $K = (L + 1)/2$. The following lemma shows how $\hat{n}$ depends on the MUM model parameters L and C as the number of processors gets large.

LEMMA 5.3 : As $N \rightarrow \infty$, then

$$\hat{n} = \begin{cases} \sqrt{2C} & \text{if } C < \frac{(L-K-1)^2}{2} \\ L-K-1 & \text{if } \frac{(L-K-1)^2}{2} \leq C < \frac{(L-K-1)(L-K)}{2} \\ \textit{no minimum} & \textit{otherwise} \end{cases}$$

□

The proof of Lemma 5.3 is given in Appendix B.

Lemma 5.3 shows that for small values of C, $\hat{n}$ increases in C as a square root, until C reaches a threshold. For values of C larger than this threshold, $E[(W)]$ cannot be minimized, which implies that the cost of remapping is too high to ever consider remapping. We recognize this critical threshold as essentially the expected processor state squared, divided by two. Lemma 5.3 thus helps to quantify the role that the remapping cost plays in the MUM model. It identifies a relationship between permissible remapping costs and processor execution time, showing that remapping improves performance even if the remapping cost is relatively large compared to processor execution time.

The asymptotic assumptions underlying Lemma 5.3 lead to one discomfitting effect: the variation in the Markov chains does not play a role in determining $\hat{n}$. A second approximation to $E[W(n)]$ is more sensitive to this variation. Lemma 5.4 states the general dependence of $\hat{n}$ on the number of processors N, the cost C of remapping, and the MUM transition parameter p.

LEMMA 5.4 : Using an approximation given by [14], $\hat{n}$ is a function of $\frac{C}{Nd(N)\sqrt{p}}$, where $d(N) \approx N^{-1/2}$.
□

The proof of Lemma 5.4 is also given in Appendix B.

We have noted that as the cost of remapping increases, it makes sense to remap less frequently so as to better amortize the cost of remapping over a larger number of computational steps. For the MUM model, $\hat{n}$ is actually a function of the expression $\frac{C}{Nd(N)\sqrt{p}}$, where $d(N) \approx N^{-1/2}$. The cost C of remapping, the number of processors N and the activity p of the processors together determine the value of $\hat{n}$. One can see that increasing the number of processors, and increasing the activity associated with each processor leads to a reduction in the number of steps between remappings analogous to that obtained by decreasing the cost of remapping.

This section has examined the statistic $W(n)$. We have demonstrated general conditions which ensure that $E[W(n)]$ has at most one minimum, and that the existence of the minimum at $\hat{n}$ implies that remapping every $\hat{n}$ steps is the optimal fixed-interval remapping policy. This result supports our use of the SAR policy by suggesting that performance gains are achieved by minimizing $W(n)$. We then looked at $E[W(n)]$ specifically under the MUM model assumptions. We showed general conditions ensuring that $E[W(n)]$ has at most one minimum; we showed that $E[W(n)]$ is computable, and analyzed the interrelationships between MUM model parameters by looking at how they affect the expected frequency of remapping.

6. Load Dependency Model

While the MUM model is analytically tractable, some of its assumptions may not be realized in practice. For example, MUM assumes that a processor's load drift is stochastically independent of any other processor's load drift. It is easy to construct examples where this assumption is violated. This flaw could be corrected by allowing correlation between chains' transitions, but then an appropriate model of correlation would have to be determined. MUM also assumes homogeneous Markov chains; there is no problem in allowing heterogeneous chains, but the analysis we have developed does not apply to such a model. More seriously, the MUM model implicitly assumes that the transitional behavior of a processor's computational load is determined by the processor, rather than the load. This assumption is embodied in the assumption of transitional invariance under remapping, used to prove Theorem 3.1. This flaw is corrected in a model where the distinction between a processor and its load is clearly drawn. We call this the Load Dependency (LD) model.

The LD model directly simulates the spatial distribution of computational load in a domain. We consider a two dimensional plane in which activity occurs, for example, a factory floor. To simulate this activity we impose a dense regular grid upon the plane; each square of the grid defines an *activity point.* We suppose that activity in the plane is discretized in simulation time, and model the behavior of activity as follows. Each time step a certain amount of activity may occur at an activity point. This activity is simulated (for example, arrival of parts to a manufacturing assembly station), causing a certain amount of computation. By the next time step some of that activity may have moved to neighboring activity points. This movement of activity simulates the movement of physical objects in a physical domain, and is modeled by the movement of *work units.* A work unit is always positioned at some activity point, and has a weight describing its computational demand at that activity point. From one time step to the next, a work unit may move from an activity point to a neighboring activity point; this movement is governed probabilistically. In the LD model, the probability that a work unit will move from one activity point to another, as the problem goes from one time step to the next is called the *transition probability* linking the two activity points.

We employ binary dissection [6] to partition the activity points into *N activity regions*, where the points in an activity region form a rectangular mass. The weight of an activity point is taken to be the sum of the weights of work units at the point, at the time that the partitioning is performed. The computational load on a processor during a time step is found by adding the weights of all work units resident on activity points assigned to that processor.

In a wide variety of problems, including those mentioned in section 1 as examples of varying demand distribution problems, data dependencies are quite local. Decomposition of a domain into contiguous regions with a relatively small perimeter to area ratio is thus generally desirable for reducing the quantity of information that must be exchanged between partitions. Furthermore, due to the local nature of the data dependencies, the communication required between partitions in a binary dissection will generally be greatest in partitions that are in physical proximity. The analysis in [6] shows that this type of partition is effective for static remapping, and is easily mapped onto various types of parallel architectures. Estimates are also obtained of the communication costs incurred when binary dissection is used to partition a problem's domain, and the resulting partitions are mapped onto a given architecture. The communication cost estimates obtained by such analysis are inevitably problem, mapping and architecture dependent. This binary dissection is briefly described in Appendix C.

A processor's load changes from one time step to the next when a work unit either moves to an activity point assigned to another processor, or similarly moves from an activity point in a different processor. This explicit modeling of work unit movement removes the most serious flaw with the MUM model. Unlike the MUM model, the change in a processor's computational load from one time step to the next is explicitly dependent on its own load, and on the loads of processors with neighboring activity regions.

To ensure the correctness of the simulation, we require that all computation associated with a time step be completed before the simulation advances to the next time step. Thus, as in the MUM model, the time required to complete a time step is the maximum computation time among all processors. Again like the MUM model, as time progresses any initial balance will disappear, and average processor utilization will drop. This is particularly true if the work unit movement probabilities are anisotropic.

The SAR policy can also be used with the LD model, since the $W(n)$ statistic requires only the mean processor execution time, the maximum processor execution time per time step, and the remapping cost C. The performance of SAR on the LD model was examined by once again comparing SAR to the performance of fixed interval polices. Figure 5 plots expected processor utilization as a function of time for remapping costs of 50 and 100 work units, when a 64 by 64 mesh of activity points is initialized with one work unit per activity point, and 16 processors are employed. The transition probabilities are anisotropic (given in the figure legend), so that the work tends to drift to the upper right portion of the mesh over time. Not taken into account here is the cost of the interprocessor communication that occurs at the end of each step when partitions exchange newly computed results. As was observed in the MUM model the the performance of the SAR rule and the average number of elapsed steps between SAR remappings corresponded closely to that of the fixed interval leading to the optimal performance. In figure 5, the performance of SAR for a given cost is superior to that obtained from fixed load balancing at the optimal frequency. In other simulations, the performance obtained from SAR was comparable to, but slightly below that obtained from the optimal fixed load balancing method. Note that the performance obtained by SAR in figure 5 is markedly greater than that obtained when no remapping is performed.

7. Summary

To date, most load balancing problems addressed in the literature concern systems which can be modeled by a queueing network. A large class of parallel computations are not well modeled by queues and job arrivals, particularly those solving scientific problems. Yet the time-variant behavior of these computations, coupled with their synchronization needs, creates the following performance problem. Good processor utilization requires that the computational load be balanced between processors, yet a good balance can't be sustained because of the variation in the computational workload. To treat this problem, we need to both model the phenomenon of performance degradation, and develop remapping decision policies which effectively determine when the computational load should be remapped onto the parallel machine . This paper has addressed both issues. We describe two different models of load evolution. One model is simpler than the other, and can be analyzed. The other model better captures a means by which a processor's load changes in time. We have developed and studied an adaptive remapping decision policy SAR which proves to be effective on both models. SAR does not depend on the details of the model structure; rather, it attempts to minimize a statistic which measures the long-term average system degradation (including that due to remapping) as a function of time. We have also analytically demonstrated conditions ensuring that the statistic's mean has a single local minimum, and that the optimal fixed-interval remapping policy is to remap when the statistic's mean is minimized. These analytic results validate SAR's approach. Because of its appealing empirical and analytical properties, SAR is a promising candidate for use in an actual parallel system.

Appendix A

In this appendix we prove Theorem 3.1. The assumptions we use for its proof are

- The Markov chains are homogeneous, and unbounded (no maximum nor minimum state).
- The Markov chains' transition probabilities are unaffected by remapping.

Both assumptions are used for analytic tractability. Under these assumptions, we will show that the average value remapping policy is optimal in the sense that among all remapping schemes we could apply at time n, the average value remapping scheme minimizes $E[T_{\max}(n+d)]$ for all $d > 0$. The use of the average remapping scheme therefore minimizes the expected duration of every future computation step.

Our basic tool for establishing the average value policy's optimality is the theory of stochastic variability, alluded to in section 5. Recall that a random variable X is said to be *stochastically more variable* than random variable Y, denoted $X \geq_v Y$, if $E[g(X)] \geq E[g(Y)]$ for every increasing convex function g. This definition immediately implies that if $X \geq_v Y$, then $E[X] \geq E[Y]$. For non-negative random variables, an equivalent definition [36] is that

$$\int_a^\infty Prob\{X > t\}\, dt \geq \int_a^\infty Prob\{Y > t\}\, dt \quad \text{for all } a \geq 0.$$

For our purposes, the following result, also from [36], is important. if $X_1, \cdots, X_n$ is a group of independent random variables, $Y_1, \cdots, Y_n$ is a group of independent random variables, and $X_i \geq_v Y_i$ for $i = 1,2, \cdots, n$, then

$$g(X_1, \cdots, X_n) \geq_v g(Y_1, \cdots, Y_n) \tag{6}$$

for all increasing convex functions g.

We will first show that average value remapping is optimal in a system with two chains. We observe that if $|T_1(n) - T_2(n)| \leq 1$, then there is no benefit to be gained from remapping. We consequently assume that if we remap at n, then $|T_1(n) - T_2(n)| > 1$. Our results stem principally from the following lemma.

LEMMA A-1 : Let X and Y be independent, identically distributed, integer valued random variables. If $a > b-1$, then

$$\max\{a+X, b+Y\} \geq_v \max\{a-1+X, b+1+Y\}.$$

PROOF: Let g be any increasing convex function, and let

$$D_g(x,y) = g(\max\{a+x, b+y\}) - g(\max\{a-1+x, b+1+y\}).$$

We will argue that $E[D_g(X,Y)] \geq 0$, which will prove the lemma. We consider the value of

$$\max\{a+x, b+y\} - \max\{a-1+x, b+1+y\} \tag{7}$$

as a function of x and y. When $a-1+x \geq b+1+y$ this difference is easily seen to be 1. When $a-1+x \leq b+1+y$ and $a+x > b+y$ expression (7) is equal to $(a-b-1) + (x-y)$; furthermore, $b-a < x-y$. But since x and y are integer valued, we have $b-a+1 \leq x-y$, so that expression (7) is non-negative. In the cases considered, the increasing nature of g ensures that $D_g(x,y) \geq 0$. Finally, when $a+x \leq b+y$, expression (7) equals -1. Suppose then that $X = \hat{x}$, $Y = \hat{y}$, and $a-b \leq \hat{y}-\hat{x}$. Since X and Y are independent and identically distributed, we have

$$Prob\{X = \hat{x}, Y = \hat{y}\} = Prob\{X = \hat{y}, Y = \hat{x}\}.$$

Thus for any samples of $X = \hat{x}$ and $Y = \hat{y}$ which cause (7) to be -1, it is equally likely that $X = \hat{y}$ and $Y = \hat{x}$, in which case (7) is equal to 1. Since $\hat{y} \geq \hat{x}$, we have $\max\{a+\hat{y}, b+\hat{x}\} \geq \max\{a+\hat{x}, b+\hat{y}\}$. It then follows from the increasing convexity of g that $D_g(\hat{y}, \hat{x}) \geq | D_g(\hat{x}, \hat{y}) |$. The importance of this observation is that every sample of X and Y which causes $D_g(x,y)$ to be negative is counter-balanced by an equally probable sample of X and Y which yields $D_g(x,y)$ with a larger positive magnitude. It follows then that $E[D_g(X, Y)] \geq 0$, which proves the lemma.

□

To apply this lemma to our problem, we note that if X is the Markov chain single step random variable, and if $X(d)$ denotes a d-fold convolution of X, then

$$T_{\max}(n+d,a,b) = \max\{a+X_1(d),\ b+X_2(d)\}$$

where $X_1(d)$ and $X_2(d)$ are independent. It follows immediately from Lemma A-1 that $T_{\max}(n+d,a,b) \geq_v T_{\max}(n+d,a-1,b+1)$. We then use this result to show that average value remapping is optimal for a N chain system.

THEOREM 3.1 : Assume that the Markov chains are homogeneous, and unbounded (no maximum nor minimum state), and that the transition probabilities are unaffected by remapping. Suppose $T_1(n,s_1) + T_2(n,s_2) + \cdots + T_N(n,s_N) = K$. For every $i = 1,2,\cdots,N$, define

$$a_i = \begin{cases} \lfloor K/N \rfloor + 1 & \text{if } i < K \bmod N \\ \lfloor K/N \rfloor & \text{if } i \geq K \bmod N \end{cases}.$$

Then for every $d \geq 0$,

$$E[T_{\max}(n+d,s_1,\cdots,s_N)] \geq E[T_{\max}(n+d,a_1,\cdots,a_N)].$$

PROOF: Without loss of generality, assume that $s_1 \geq s_2 \geq \cdots \geq s_N$. Note first that

$$T_{\max}(n+d,s_1,\cdots,s_N)$$
$$= \max\{\ \max\{T_1(n+d,s_1),T_N(n+d,s_N)\},\ T_2(n+d,s_2),\cdots,T_N-1(n+d,s_N-1)\}.$$

Now

$$\max\{T_1(n+d,s_1),T_N(n+d,s_N)\} \geq_v \max\{T_1(n+d,s_1-1),T_N(n+d,s_N+1)\},$$

so that by (6),

$$T_{\max}(n+d,s_1,\cdots,s_N) \geq_v T_{\max}(n+d,s_1-1,\cdots,s_N+1).$$

This argument applies to any set of $s_1,\cdots,s_N$ such that $\sum_{j=1}^{N} s_j = K$. We may therefore apply the argument repeatedly to find that

$$T_{\max}(n+d,s_1,\cdots,s_N) \geq_v T_{\max}(n+d,a_1,\cdots,a_N).$$

The theorem's conclusion follows immediately.

□

Appendix B

In this appendix we prove Lemmas 5.1, 5.2, 5.3, and 5.4. Lemmas 5.1 and 5.2 are used in section 5 to prove Theorem 5.1. That theorem gives sufficient conditions for the difference $E[T_{\max}(i)] - E[\overline{T}(i)]$ arising from the MUM model to be monotone increasing.

LEMMA 5.1 : Assume that at step 0, all chains are in state $(L+1)/2$, i.e., for all j, $p_j((L+1)/2;0) = 1$; also assume that $p \le 2/3$. Then for all steps n, we have $p_j(k-1;n) \le p_j(k;n)$ for $k \le (L+1)/2$, and $p_j(k+1;n) \le p_j(k;n)$ for $k \ge (L+1)/2$.

PROOF:

We induct on n. For n=0, the claim is trivially true since $p_j((L+1)/2;0) = 1$. Assume that the claim is true for n, we shall show that it is also valid for n+1. By the symmetry of the probability mass function $p_j(k;n)$, it suffices to only prove that $p_j(k-1;n) \le p_j(k;n)$ for $k \le (L+1)/2$. There are three cases to consider. Each case will employ the Chapman-Kolmogorov difference equations [36] to describe the state probabilities at step n+1 in terms of the state probabilities at step n.

Case 1: $2 < k < (L+1)/2$: Now

$$\begin{aligned} p_j(k;n+1) - p_j(k;n+1) &= \frac{p}{2}(p_j(k-1;n) - p_j(k-2;n)) \\ &\quad + (1-p)(p_j(k;n) - p_j(k-1;n)) + \frac{p}{2}(p_j(k+1;n) - p_j(k;n)) \\ &\ge 0 \end{aligned}$$

since $p_j(s;n) \ge p_j(s-1;n)$ for $1 \le s \le (L+1)/2$ by the induction hypothesis.

Case 2: $k = 2$: Noting that

$$\begin{aligned} p_j(1;n+1) &= (1 - \frac{p}{2})p_j(1;n) + \frac{p}{2}p_j(2;n) \\ &= \frac{p}{2}p_j(1;n) + (1-p)p_j(1;n) + \frac{p}{2}p_j(2;n) \end{aligned}$$

we have

$$p_j(2;n+1) - p_j(1;n+1) = \frac{p}{2}(p_j(1;n) - p_j(1;n)) + (1-p)(p_j(2;n) - p_j(1;n))$$
$$+ \frac{p}{2}(p_j(3;n) - p_j(2;n))$$
$$\geq 0$$

again by the induction hypothesis.

Case 3: $k = (L+1)/2$:

Here we must make assumptions on the allowable values of p. Utilizing the induction hypothesis to assume that $p_j((L+1)/2 - 1;n) \geq p_j((L+2)/2 - 2;n)$, we obtain the following bound on $p_j((L+1)/2 - 1;n+1)$.

$$p_j((L+1)/2 - 1;n+1) = \frac{p}{2}p_j((L+1)/2;n) + (1-p)p_j((L+1)/2 - 1;n) + \frac{p}{2}p_j((L+1)/2 - 2;n)$$
$$\leq \frac{p}{2}p_j((L+1)/2;n) + (1 - \frac{p}{2})p_j((L+1)/2 - 1;n).$$

Utilizing the symmetry of the probability mass function we obtain:

$$p_j((L+1)/2;n+1) = \frac{p}{2}p_j((L+1)/2 + 1;n) + (1-p)p_j((L+1)/2;n) + \frac{p}{2}p_j((L+1)/2 - 1;n)$$
$$= (1-p)p_j((L+1)/2;m) + (p)p_j((L+1)/2 - 1;m).$$

Thus when $\frac{p}{2} \leq 1-p$, or equivalently $p \leq \frac{2}{3}$, we obtain $p_j((L+1)/2 - 1;n) \leq p_j((L+1)/2;n)$.

□

Note that when $p > \frac{2}{3}$, Lemma 5.1's conclusion is not true for a three state chain. When $p = \frac{2}{3} + \varepsilon$ and n=1, it is straightforward to verify that $p_j(1;1) = \frac{1}{3} + \frac{\varepsilon}{2}$, $p_j(2;1) = \frac{1}{3} - \varepsilon$, and $p_j(3;1) = \frac{1}{3} + \frac{\varepsilon}{2}$.

Next we prove Lemma 5.2.

LEMMA 5.2 : If for every n

$$Prob\{T_j(n) = L\} = \min_k \left\{ Prob\{T_j(n) = k\} \right\},$$

then $T_j(n) \geq_v T_j(n-1)$ for all n.

PROOF: The $\geq_v$ order relation is discussed in Appendix A. We will simplify notation in this proof by writing $T_j(n)$ simply as $T(n)$, suppressing the index j. We will demonstrate that

$$\int_a^\infty Prob\{T(n) > t\}\, dt \geq \int_a^\infty Prob\{T(n-1) > t\}\, dt \quad \text{for all } a \geq 0. \tag{9}$$

We first note that this relationship is true for $0 \leq a \leq 1$. This follows since the case where $a = 0$ describes the means (which are equal), and $T(k) \geq 1$ for all k implies that

$$\int_a^1 Prob\{T(n) > t\}\, dt = \int_a^1 Prob\{T(n-1) > t\}\, dt.$$

Supposing that $a > 1$, we observe that for any t,

$$Prob\{T(n) > t\} = Prob\{T(n-1) + X_n > t\} \tag{10}$$

where X_n is the random step taken by the chain between steps $n-1$ and n. By the theorem of total probability,

$$\begin{aligned} Prob\{T(n-1) + X_n > t\} = {} & Prob\{X_n = -1\} Prob\{T(n-1) > t+1 \mid X_n = -1\} \\ & + Prob\{X_n = 0\} Prob\{T(n-1) > t \mid X_n = 0\} \\ & + Prob\{X_n = 1\} Prob\{T(n-1) > t-1 \mid X_n = 1\}. \end{aligned}$$

We then integrate this expression with respect to t from a to ∞. By adjusting the bounds of integration so that every inequality is expressed with respect to t alone, then factoring out integrals from a to ∞, and finally combining those factored integrals into the term $\int_a^\infty Prob\{T(n-1) > t\}\, dt$, it is shown directly that

$$\int_a^\infty Prob\{T(n) > t\}\,dt \;=\; \int_a^\infty Prob\{T(n-1) > t\}\,dt$$

$$+\; Prob\{X_n = 1\}\int_{a-1}^{a} Prob\{T(n-1) > t \mid X_n = 1\}\,dt$$

$$-\; Prob\{X_n = -1\}\int_{a}^{a+1} Prob\{T(n-1) > t \mid X_n = -1\}\,dt.$$

Noting that $Prob\{X_n = -1\} = Prob\{X_n = 1\}$, we see that relation (9) in this case is satisfied if and only if

$$\int_{a-1}^{a} Prob\{T(n-1) > t \mid X_n = 1\}\,dt \;\geq\; \int_{a}^{a+1} Prob\{T(n-1) > t \mid X_n = -1\}\,dt.$$

The knowledge that $X_n = 1$ implies only that $T(n-1) \neq L$; thus

$$Prob\{T(n-1) > t \mid X_n = 1\} \;=\; Prob\{T(n-1) > t \mid T(n-1) \neq L\}.$$

It is then mechanical to show that

$$Prob\{T(n-1) > t \mid X_n = 1\} \;=\; \frac{Prob\{T(n-1) > t\} - Prob\{T(n-1) = L\}}{Prob\{T(n-1) \neq L\}}.$$

Since $X_n = -1$ only if $T(n-1) \neq 1$, a similar exercise demonstrates that

$$Prob\{T(n-1) > t \mid X_n = -1\} \;=\; \frac{Prob\{T(n-1) > t\}}{Prob\{T(n-1) \neq 1\}}.$$

We then note that by symmetry of $T(n-1)$'s distribution, the denominators of these last two quotients are equal. Thus we see that to satisfy (9), we need only show that

$$\int_{a-1}^{a} Prob\{T(n-1) > t\}\,dt \;-\; Prob\{T(n-1) = L\} \;\geq\; \int_{a}^{a+1} Prob\{T(n-1) > t\}\,dt\,. \tag{11}$$

Since $T(n-1)$ is an integer valued random variable, $Prob\{T(n-1) > t\}$ is a decreasing step function of t, with steps occurring at integer values of t. Furthermore, the change in the function's value at $t = j$ is precisely $-Prob\{T(n-1) = j\}$. Letting $j(t)$ denote the smallest integer larger than t, we see that the assumption that $Prob\{T(n-1) = k\}$ is minimized at $k = L$ implies that for all positive t,

$$Prob\{T(n-1) > t\} - Prob\{T(n-1) = L\} \geq Prob\{T(n-1) > t\} - Prob\{T(n-1) = j(t)\} \quad (12)$$
$$= Prob\{T(n-1) > t+1\}.$$

We can now integrate both sides of (12) from $a-1$ to a, change variables and obtain (11). This shows that (9) is satisfied, so that $T(n) \geq_v T(n-1)$ as claimed.

□

We next provide proofs to Lemmas 5.3 and 5.4, which describe how the $\hat{n}$ minimizing $E[W(n)]$ behaves as a function of MUM model parameters. Lemma 5.3 gives $\hat{n}$ as the number of processors approaches ∞. The basic idea behind Lemma 5.3's proof is that with an infinite number of processors, the maximum state will always advance forward one state until the maximum state L is reached. After this point, the maximum state will always be L. We now substantiate these claims.

We have assumed that the Markov chains are all identical, have a odd number of states and that the initial state K of each Markov chain is equal to $(L+1)/2$. We hence assume that $P_j(s;0) = 0$ for $1 \leq s < K$ and $P_j(s;0) = 1$ for $K \leq s \leq L$. Again to simplify notation, we suppress the dependence of the cumulative distribution function $P_j(s;n)$ on j, since all chains are identical. Then rewriting equation (5):

$$E[W(n)] = \frac{\sum_{m=1}^{n}\sum_{s=1}^{L}\left[P(s-1;m)-(P(s-1;m))^N\right] + C}{n}. \quad (13)$$

For $m \leq L-K-1$, all chains must be in states numbered less than or equal to $K+m$ as the state number of a chain can increase by at most one per step. Consequently for $s \geq K+m$, the probability that $T(m)$ is equal to s is zero, and hence $P(s;m) = 1$ for $s \geq K+m-1$. When $m > L-K-1$, every state has a non-zero probability of occupancy so that $P(s;m) = 1$ only for $s = L$. Thus

$$\lim_{N\to\infty} P(s-1;m)^N = \begin{cases} 1 & \text{if } K+m+1 \le s \le L \\ 0 & \textit{otherwise} \end{cases} .$$

It follows that as $N \to \infty$,

$$\sum_{s=1}^{L} P^N(s-1;m) = L-(K+m) \quad m \le L-K-1$$
$$\sum_{s=1}^{L} P^N(s-1;m) = 0 \qquad\qquad m > L-K-1$$

The expected state of each chain is equal to K for all n due to the symmetry of the Markov chains' transition probabilities and the symmetry of the initial probability distribution. Hence, $\sum_{s=1}^{L}(1-P(s-1;m)) = K$ for all s and thus $\sum_{s=1}^{L}P(s-1;m) = L-K$. Substituting the above into (13) yields

$$E[W(n)] = \begin{cases} \dfrac{n}{2}+\dfrac{1}{2}+\dfrac{C}{n} & n \le L-K-1 \qquad \text{(B.1)} \\ L-K+\dfrac{2C-(L-K-1)(L-K)}{2n} & n \ge L-K-1 \qquad \text{(B.2)} \end{cases} . \tag{14}$$

Note that (B.1) and (B.2) take on identical values at $L-K-1$, and hence (14) is continuous.

We are now in a position to find the $\hat{n}$ that minimizes the asymptotic form of $E[W(n)]$. In order to derive simple expressions for $\hat{n}$ we shall allow n to assume any real value. The location of the minimum $\hat{n}$, if it exists, depends on the remapping cost C in a way that is stated by Lemma 5.3.

LEMMA 5.3 : As $N\to\infty$, then

$$\hat{n} = \begin{cases} \sqrt{2C} & \text{if } C < \dfrac{(L-K-1)^2}{2} \\ L-K-1 & \text{if } \dfrac{(L-K-1)^2}{2} \le C < \dfrac{(l-K-1)(L-K)}{2} \\ \textit{no minimum} & \textit{otherwise} \end{cases}$$

PROOF: We will find the minimum values n_1 of (B.1) and n_2 of (B.2) when either exist. The continuity of (14) is then used to make statements about the location and existence of the minimum value of (14). Simple calculus shows that if C is positive, then $n/2 + 1/2 + C/n$ has a local minimum at $n_1 = \sqrt{2C}$. This local minimum lies in (B.1)'s functional range only if $C \le \dfrac{(L-K-1)^2}{2}$; (B.1) is otherwise minimized at its domain endpoint $n_1 = L-K-1$. Having established n_1's form, we consider the minimization of (B.2) at n_2. We observe that (B.2) is a strictly increasing function of n when $C < \frac{(L-K-1)(L-K)}{2}$. Consequently (B.2) is minimized at its domain endpoint $n_1 = L-K-1$. For $C \geq \frac{(L-K-1)(L-K)}{2}$, (B.2) is a constant or decreasing function, and hence has no minimum.

Thus when $C < \frac{(L-K-1)^2}{2}$, we have $n_1 = \sqrt{2C} < L-K-1$ and $n_2 = L-K-1$; it follows that $\hat{n} = \sqrt{2C}$. For $\frac{(L-K-1)^2}{2} \leq C < \frac{(L-K-1)(L-K)}{2}$, both n_1 and n_2 equal $L-K-1$. For $C \geq \frac{(L-K-1)(L-K)}{2}$, (B.1) is increasing over $[0, L-K-1]$, and (B.2) is non-decreasing over $[L-K-1, \infty]$. Recalling that (14) is continuous, we see then that under these conditions (14) has no local minimum.

□

Lemma 5.4 shows how different MUM model parameters affect the optimal static remapping frequency $\hat{n}$. Unlike Lemma 5.3, it specifically incorporates the "activity parameter" p. The analysis leading to Lemma 5.4's statement makes use of the bounds on the expectation of the maximum order statistic of a set of random variables with the same symmetric distribution. These bounds prove to be quite tight for relatively small numbers of chains and consequently provide a useful approximation for the expectation [14]. We also make the approximation that each Markov chain has an *infinite* number of states. As before, the one step transition probability from state s to state $s+1$ is $p/2$, from state s to state $s-1$ is $p/2$, and the probability of remaining in s is $1-p$. This analysis takes into account the effect of the number of chains on the form of $E[W(n)]$, and allows an exploration of the relationship between the steps between remappings, and the cost of remapping, the number of chains and the chains' activities.

LEMMA 5.4 : Using an approximation given by [14], $\hat{n}$ is a function of $\frac{C}{Nd(N)\sqrt{p}}$, where $d(N) \approx N^{-1/2}$.

PROOF: Since the state distribution for a chain $T(n)$ at time n is symmetric, we may apply analysis in [14] to obtain

$$\frac{E[T_{\max}(m) - \bar{T}(m)]}{(Var(T(m)))^{1/2}} \leq Nd(N)/2.$$

This expression is equivalent to

$$E[T_{\max}(m) - \bar{T}(m)] \leq \frac{Nd(N)(Var(T(m)))^{1/2}}{2} \tag{15}$$

where

$$d(N) = \left[\frac{2[1-\frac{1}{\binom{2N-1}{N-1}}]}{2N-1}\right]^{1/2}.$$

Now for any time step n, $E[W(n)]$ may be written as

$$E[W(n)] = \frac{\sum_{m=1}^{n}(E[T_{\max}(m) - \bar{T}(m)]) + C}{n},$$

which may be approximated using equation (15) by

$$E[W(n)] = \frac{Nd(N)\sum_{m=1}^{n}\sqrt{(Var(T(m)))^{1/2}} + C}{2n}. \tag{16}$$

The next step is to use the properties of the MUM model to rewrite the expression for $E[W(n)]$ in a closed form that depends on the Markov chain transition probabilities. In order to do this we must derive an expression for $Var(T(m))$. Under the assumption that a processor's state is not bounded from above or below, $T(m)$ may be written as

$$T(m) = T(0)+\sum_{k=1}^{m} X_k$$

where for all k,

$$X = X_k = \begin{cases} -1 & \textit{with probability } p/2 \\ 0 & \textit{with probability } 1-p. \\ 1 & \textit{with probability } p/2 \end{cases}$$

Under the MUM model, $Prob\{T(0) = K\} = 1$; i.e. we have deterministically specified $T(0)$. It follows that

$$\begin{aligned} Var(T(m)) &= \sum_{k=1}^{m} Var(X_k) \\ &= mVar(X) = mp. \end{aligned}$$

We then obtain the following approximation for $E[W(n)]$,

$$E[W(n)] = \frac{Nd(N)\sqrt{p}\sum_{m=1}^{n}\sqrt{m} + C}{n}. \tag{17}$$

Analysis entirely similar to that establishing Theorem 5.1 verifies that the expression above has at most local minimum. Since

$$E[W(n)] \propto \frac{\sum_{m=1}^{n}\sqrt{m} + \frac{C}{Nd(N)\sqrt{p}}}{n},$$

the point $\hat{n}$ minimizing $E[W(n)]$ (if that point exists) is determined by a function F of $\frac{C}{Nd(N)\sqrt{p}}$:

$$\hat{n} = F\left[\frac{C}{Nd(N)\sqrt{p}}\right].$$

□

To evaluate the degree to which the approximations made affect the predicted form of $E[W(n)]$, a comparison between expression (17) and simulation results are depicted in figure 6. This figure portrays results for eight chains and a variety of remapping costs. Each simulation curve depicting $E[W(n)]$ is obtained from 100 sample paths. Equation (17) yields a good approximation for eight or fewer chains; for larger numbers of chains the upper bound $d(N)$ becomes an increasingly poor approximation. In tests using twenty chains, we noticed a large discrepancy between simulation results and the values given by (17).

Appendix C

In this appendix we outline the binary partitioning algorithm described by [6]. When the LD model is partitioned by this algorithm, every activity point is assigned a weight equal to the sum of its current work units' weights. This gives rise to a matrix of weights, where each matrix entry corresponds to an activity point. For every column k, we let $CL(k)$ be the sum of weights in all columns $j \leq k$. Similarly, we let $CR(k)$ be the sum of weights in all columns $m \geq k$. The first step in the binary dissection is to determine the column $\hat{k}$ which minimizes $\min\{|CL(\hat{k}) - CR(\hat{k}+1)|, |CL(\hat{k}) - CR(\hat{k}-1)|\}$. The matrix is split between the two columns minimizing the magnitude of this difference. The column partitioning of a matrix is illustrated by figure 7a. Then the same procedure is applied to the two resulting matrices, except that the sum of row weights is considered, rather than the sum of column weights. As illustrated by figure 7b, at the completion of this step there will be four matrices of potentially varying dimensions, such that the sums of weights in the matrices are approximately equal. The procedure of dividing once by columns and twice by rows may be applied recursively to each of the resultant matrices. If the ratio of matrix points to number of partitions is small, binary dissection may yield partitions which are relatively imbalanced. We can expect increasingly balanced partitions as this ratio increases.

References

[1] C. Anderson and C. Greengard, "On Vortex Methods", *SIAM J. Numer. Anal. 22*,,413-440, 1985.

[2] S. Baden, "Dynamic Load Balancing of a Vortex Calculation Running on Multiprocessors", *University of California Berkley Computer Science Technical Report*, 1986.

[3] R. E. Bank, "A Multi-Level Iterative Method for Nonlinear Elliptic Equations" In *Elliptic Problem Solvers* ed. Martin Schultz, Academic Press, New York, 1981.

[4] D. Bai and A. Brandt, "Local Mesh Refinement Multilevel Techniques", *Dept of Appl. Math., Weizmann Institute of Science Report,* 1984

[5] J. A. Bannister, K. S. Trivedi, "Task Allocation in Fault-Tolerant Distributed Systems", *Acta Informatica, 20,* 1983, 261-281.

[6] M.J. Berger and S. Bokhari, "The Partitioning of Non-Uniform Problems",*ICASE Report No. 85-55*, November 1985.

[7] M.J. Berger and A. Jameson, "Automatic Adaptive Grid Refinement for the Euler Equations",*AIAA Journal, 23,* 1985, 561-568.

[8] M.J. Berger and J. Oliger, "Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations", *J. Comp. Phys., 53,* 1984, 484-512.

[9] S. Bokhari, "Partitioning Problems in Parallel, Pipelined, and Distributed Computing", *ICASE Report No. 85-54*, November 1985.

[10] A. Brandt, "Multilevel Adaptive Solutions to Boundary Value Problems", *Math. Comp. 31,* 1977, 333-390.

[11] K. M. Chandy, J. Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs", *IEEE Trans. on Software Engineering, SE-5, 5,* (September 1979), 440-452.

[12] W. W. Chu, L. J. Holloway, M. Lan, K. Efe, "Task Allocation in Distributed Data Processing", *Computer, 13, 11*, November 1980, 57-69.

[13] A. I. Concepcion, *Distributed Simulation on Multi-Processors: Specification, Design, and Architecture*, Ph.D. Dissertation, Wayne State University, January 1985.

[14] H. A. David, *Order Statistics*, John Wiley and Sons, New York, 1981.

[15] P. J. Denning, "Working Sets Past and Present", *IEEE Trans. on Software Engineering, SE-6, 1,* January 1980, 64-84.

[16] A. Dutta, G. Koehler, A. Whinston, "On Optimal Allocation in a Distributed Processing Environment", *Management Science, 28, 8 (August 1982), 839-853.*

[17] D.L. Eager, E.D. Lazowska, J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems", *IEEE Trans. on Software Eng., SE-12, (May 1986), 662-675.*

[18] G. S. Fishman, *Principles of Discrete Event Simulation*, Wiley & Sons, New York, 1978.

[19] G. J. Foschini, "On Heavy Traffic Diffusion Analysis and Dynamic Routing in Packet Switched Networks", in *Computer Performance*, K. M. Chandy and M. Reiser Eds., New York, North-Holland, 1977.

[20] W.D. Gropp, "Local Uniform Mesh Refinement with Moving Grids", *Yale Technical Report YALEU/DCS/RR-313,* April 1984.

[21] W.D. Gropp, "Local Uniform Mesh Refinement on Loosely-Coupled Parallel Processors ", *Yale Technical Report YALEU/DCS/RR-352,* December 1984.

[22] W.D. Gropp, "Dynamic Grid Manipulation for PDEs on Hypercube Parallel Processors ", *Yale Technical Report YALEU/DCS/RR-458,* March 1986.

[23] D. Gusfield, "Parametric Combinatorial Computing and a Problem of Program Module Distribution", *Journal of the ACM, 30, 3*, July 1983, 551-563.

[24] A. Harten and J.M. Hyman, "Self-Adjusting Grid Methods for One-Dimensional Hyperbolic Conservation Laws", *Los Alamos Report LA-9105,* 1981.

[25] M. A. Iqbal, J. H. Saltz, S. H. Bokhari, "Performance Tradeoffs in Static and Dynamic Load Balancing Strategies", Proceedings of the 1986 International Conference on Parallel Processing.

[26] D. R. Jefferson, H. Sowizral, "Fast Concurrent Simulation Using the Time Warp Mechanism", *Rand Report to the Air Force FN-1906-AFFR, Dec. 1982.*

[27] A. Leonard, "Vortex Methods for Flow Simulation", *J. Comp. Phys. 37*, , 289-335,1980.

[28] S. McCormick and J. Thomas, "The Fast Adaptive Composite Grid Method for Elliptic Equations", *Math. Comp., 46* , 1986, 439-456.

[29] L. M. Ni, C. Xu, and T.B. Gendreau, "A Distributed Drafting Algorithm for Load Balancing", *IEEE Trans. on Software Engineering, SE-11, 10*, October 1985, 1153-1161.

[30] D.M. Nicol, P.F. Reynolds Jr,"The Automated Partitioning of Simulations for Parallel Execution", *University of Virginia Dept. of Computer Science Technical Report* TR-85-15, August 1985.

[31] D. M. Nicol, P. F. Reynolds, Jr., "An Optimal Repartitioning Decision Policy", ICASE Report 86-7, February 1986.

[32] J. K. Peacock, E. Manning, J. W. Wong, "Synchronization of Distributed Simulation Using Broadcast Algorithms", *Computer Networks, 4*, 1980, 3-10.

[33] M. M. Rai, T.L. Anderson, "The Use of Adaptive Grid Generation Method for Transonic Airfoil Flow Calculations", *AIAA Paper 81-1012,* , June 1981.

[34] P. F. Reynolds, Jr., "A Shared Resource Algorithm for Distributed Simulation", *Proceedings of the Ninth Annual International Computer Architecture Conference*, Austin, Texas, April 1982, 259-266.

[35] S. Ross, *Applied Probability Models with Optimization Applications*, Holden and Day, San Fransisco, 1971.

[36] S. Ross, *Stochastic Processes*, Wiley and Sons, New York, 1983.

[37] J.H. Saltz and D.M. Nicol, "Statistical Methodologies for the Control of Dynamic Remapping", ICASE Report 86-46, July 1986, to appear in the *Proceedings of the Army Research Workshop on Parallel Processing and Medium Scale Multiprocessors* , Palo Alto, California, January 1986.

[38] R. Smith, J. Saltz, "Performance Analysis of Strategies for Moving Mesh Control", *Proceedings of the CMG XV International Conference on the Management and Performance Evaluation of Computer Systems, 1984*, 301-308.

[39] J. R. Spirn, *Program Behavior: Models and Measurements*, Elsevier North-Holland Inc, New York, 1977.

[40] J. A. Stankovic, "An Application of Bayesian Decision Theory to Decentralized Control of Job Scheduling", *IEEE Trans. on Computers, C-34*, 2 (Feb 1985), 117-130.

[41] J. A. Stankovic, K. Ramamritham and S. Cheng, "Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems", *IEEE Trans. on Computers, C-34, 12 (December 1985), 1130-1143.*

[42] H. S. Stone, "Critical Load Factors in Distributed Computer Systems", *IEEE Trans. on Software Engineering, SE-4, 3* (May 1978), 254-258.

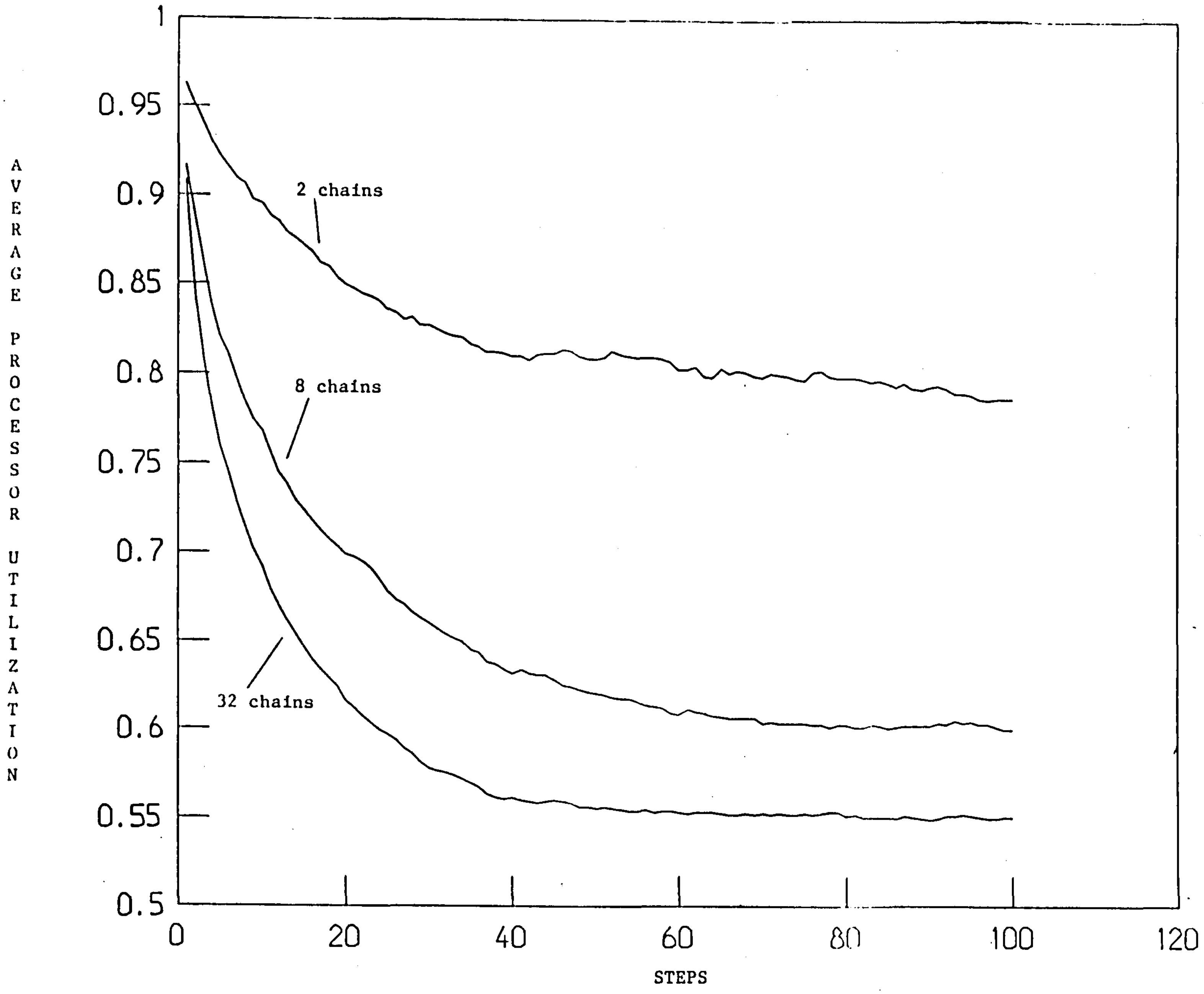
[43] D. Towsley, "Queueing Network Models with State-Dependent Routing", *Journal of the ACM*, 27, 2 (April 1980) 323-337.

[44] A. N. Tantawi and D. Towsley, "Optimal Static Load Balancing", *Journal of the ACM, 32*, 2 (April 1985), 445-465.

[45] W. Usab and E.M. Murman, "Embedded Mesh Solutions of the Euler Equation Using a Multiple-Grid Method",*Proceedings of the AIAA Computational Fluid Dynamics Conference* , Danvers, Mass., Paper 83-1946, July 1983.

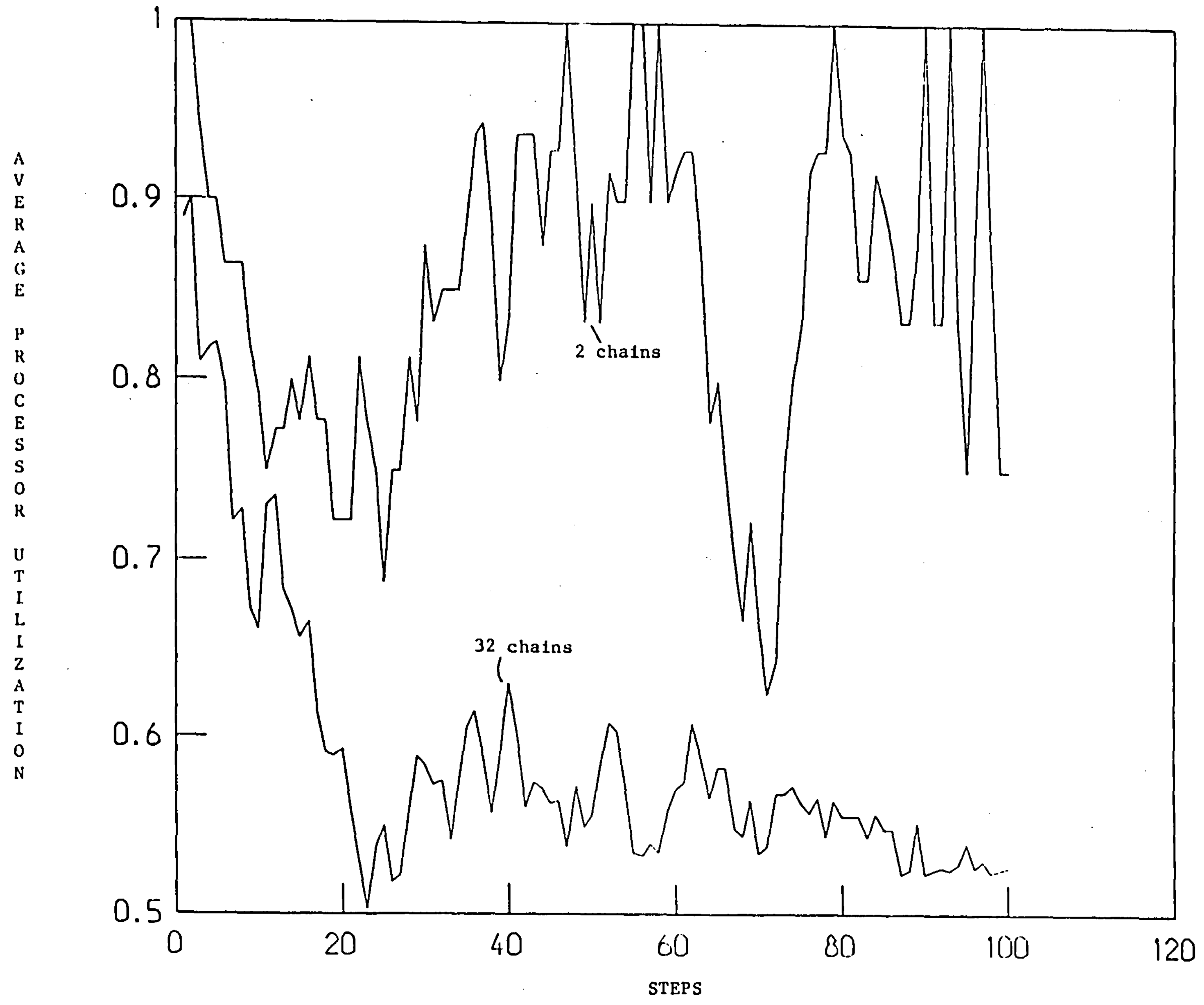
[46] D.R. Wells, "Multirate Linear Multistep Methods for the Solution of Systems of Ordinary Differential Equations",*University of Illinois Report UIUCDCS-R-82-1093*, July 1982.

[47] O.C. Zienkiewicz and A.W. Craig, "Adaptive Mesh Refinement and A Posteriori Error Estimation for the p-Version of the Finite Element Method" In *Adaptive Computational Methods for Partial Differential Equations*, edited by Ivo Babuska, SIAM, Philadelphia, 1983.



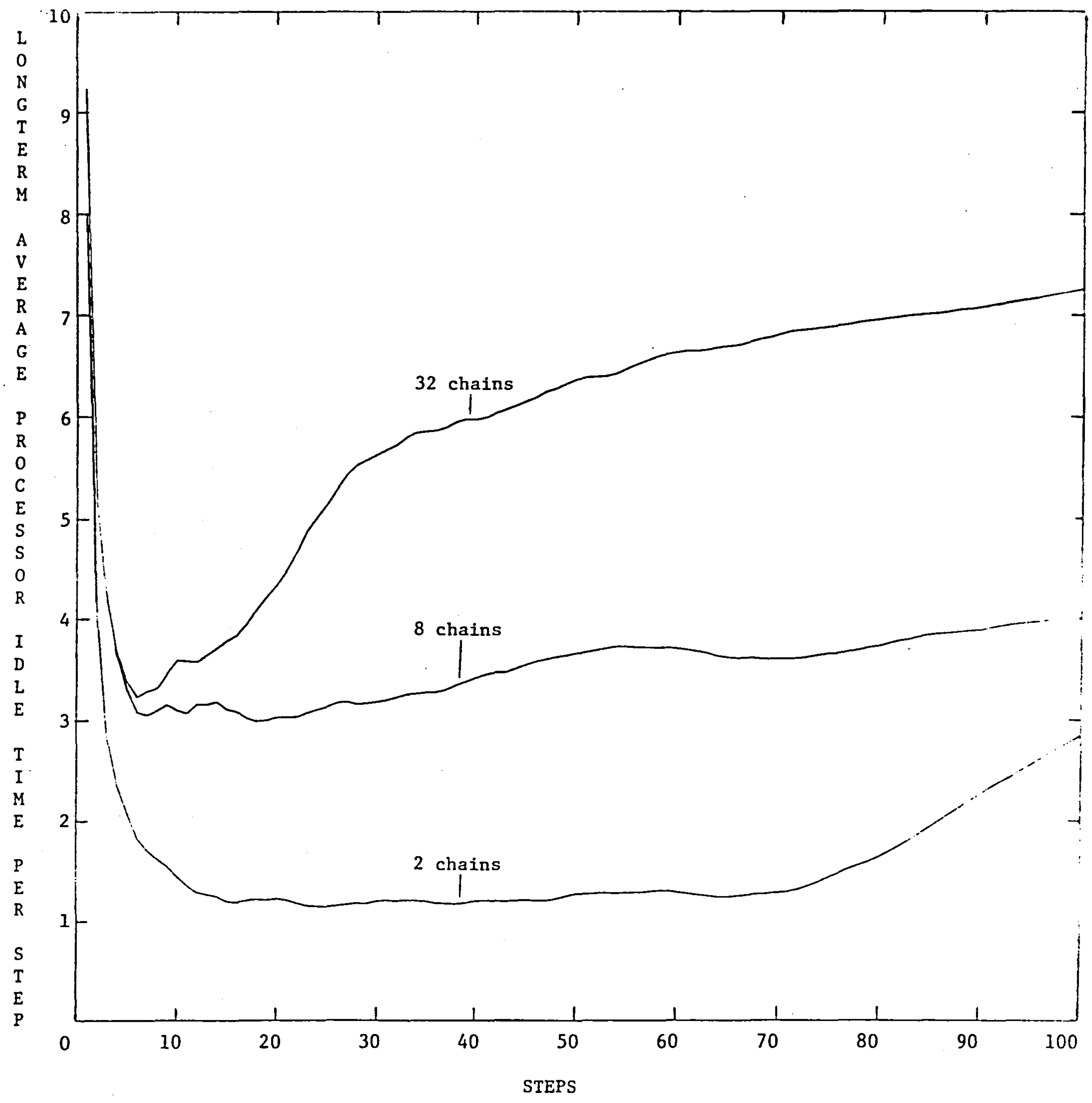
MUM Model: Expected Average Processor Utilization as a Function of Step Estimated from 500 Sample Paths. Each chain has 19 states, p = 0.5.

Figure 1a



MUM Model: Average Processor Utilization as a Function of Step-Single Sample Path. Each chain has 19 states, p = 0.5.

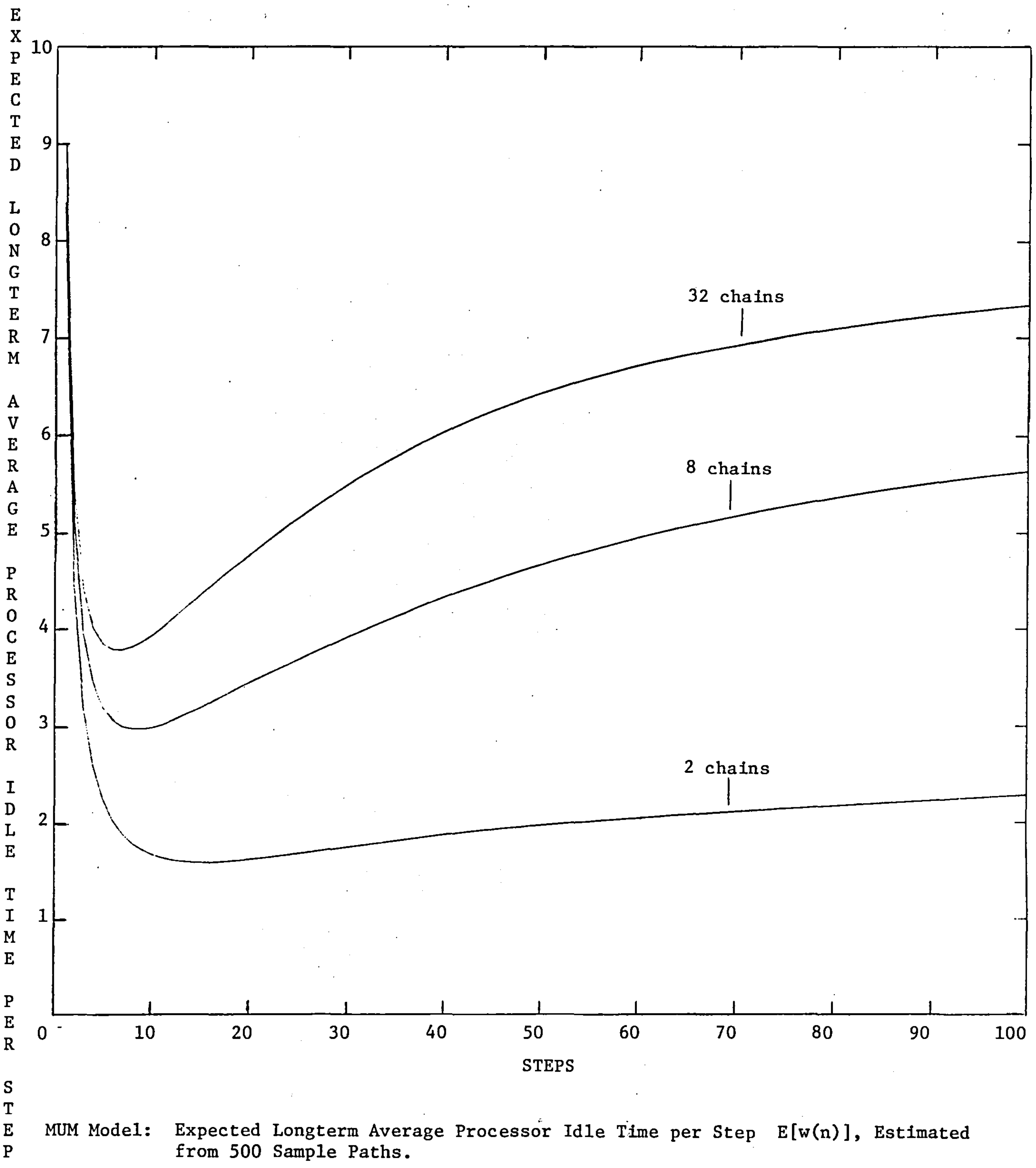
Figure 1b



MUM Model: Longterm Average Processor Idle Time per Step w(n) from Single Sample Path.

Each chain has 19 states, p = 0.5, load balancing cost = 8.

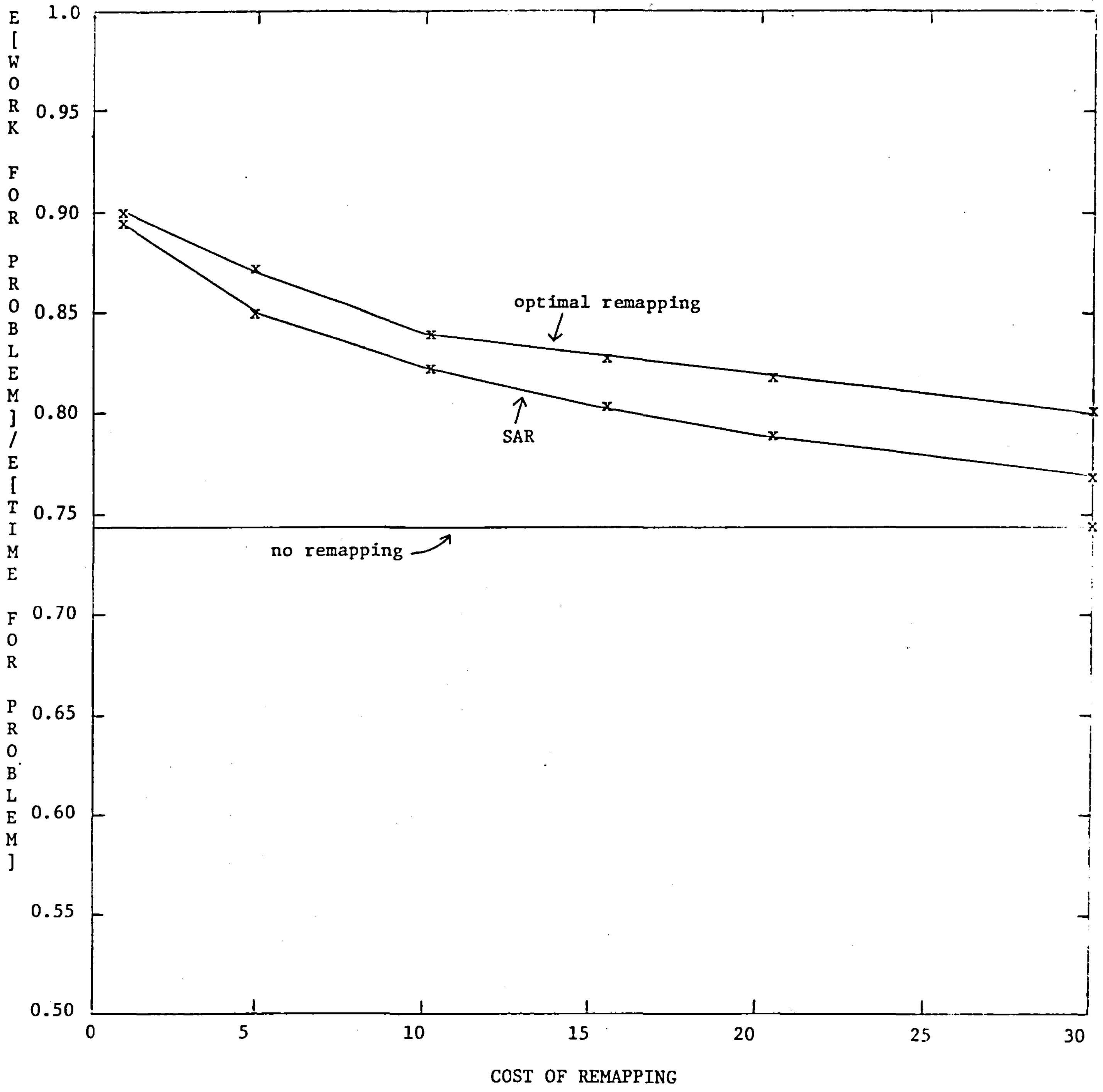
Figure 2a



MUM Model: Expected Longterm Average Processor Idle Time per Step $E[w(n)]$, Estimated from 500 Sample Paths.

Each chain has 19 states, $p = 0.5$, load balancing cost = 8.

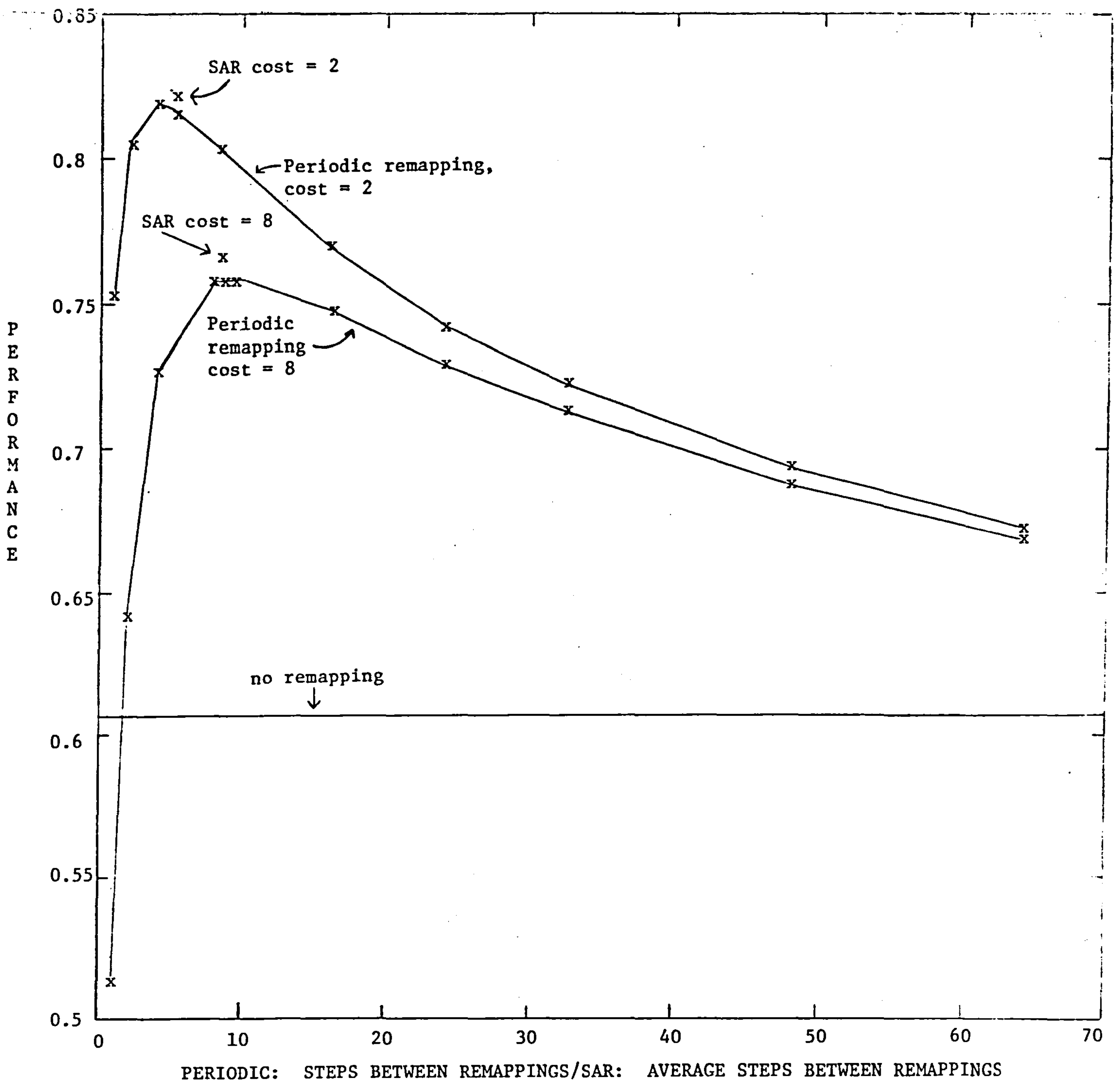
Figure 2b



MUM Model: Performance of Optimal Remapping Decision Policy Compared with Performance of SAR.

Three chains, 100 steps, each chain has 19 states, p = 0.5, SAR performance calculated from 500 sample paths.

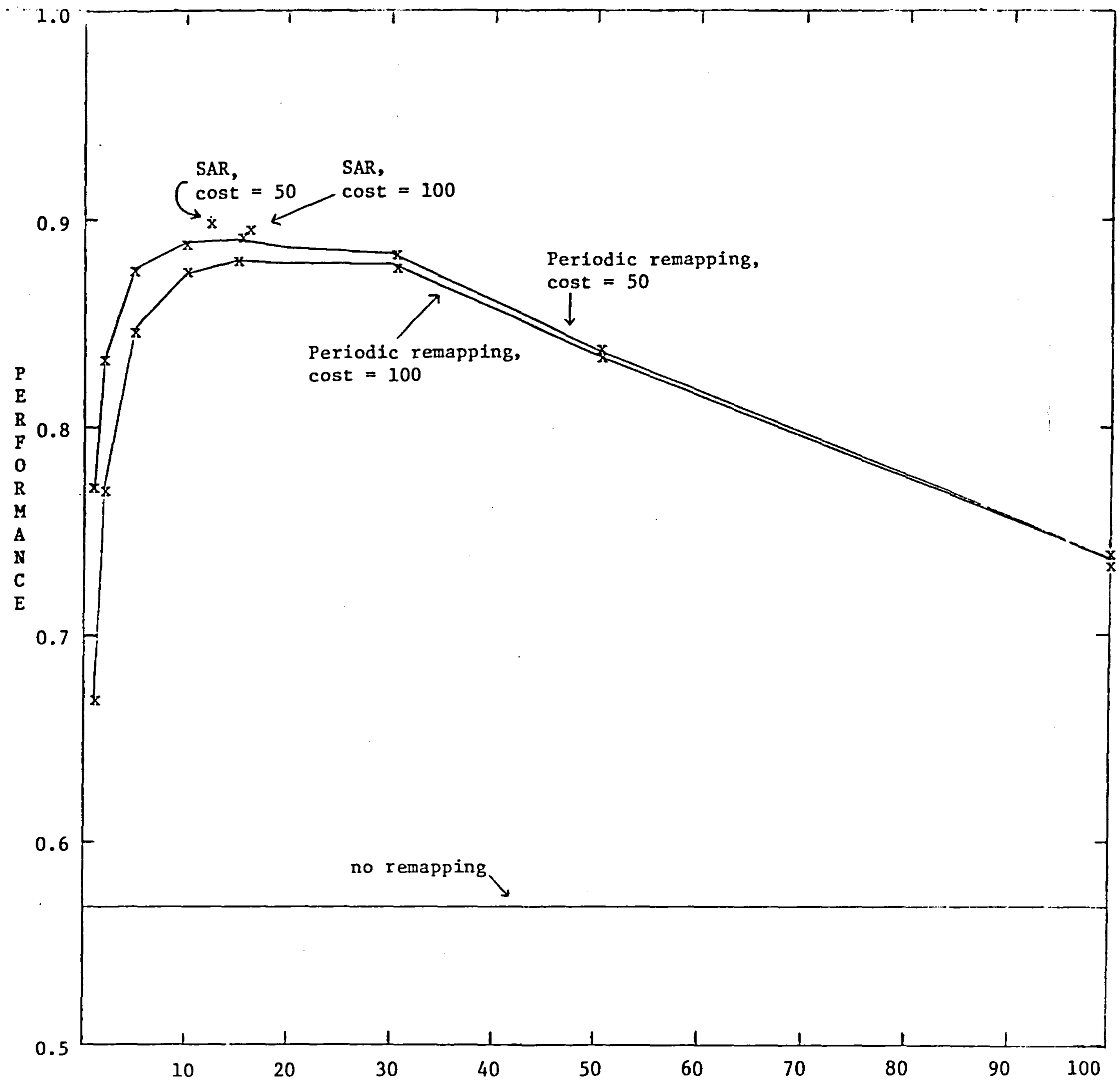
Figure 3



MUM Model: Performance of SAR Compared with Performance of Periodic Remapping.

Eight chains, 400 steps, each chain has 19 states, p = 0.5, each data point calculated from 200 sample paths.

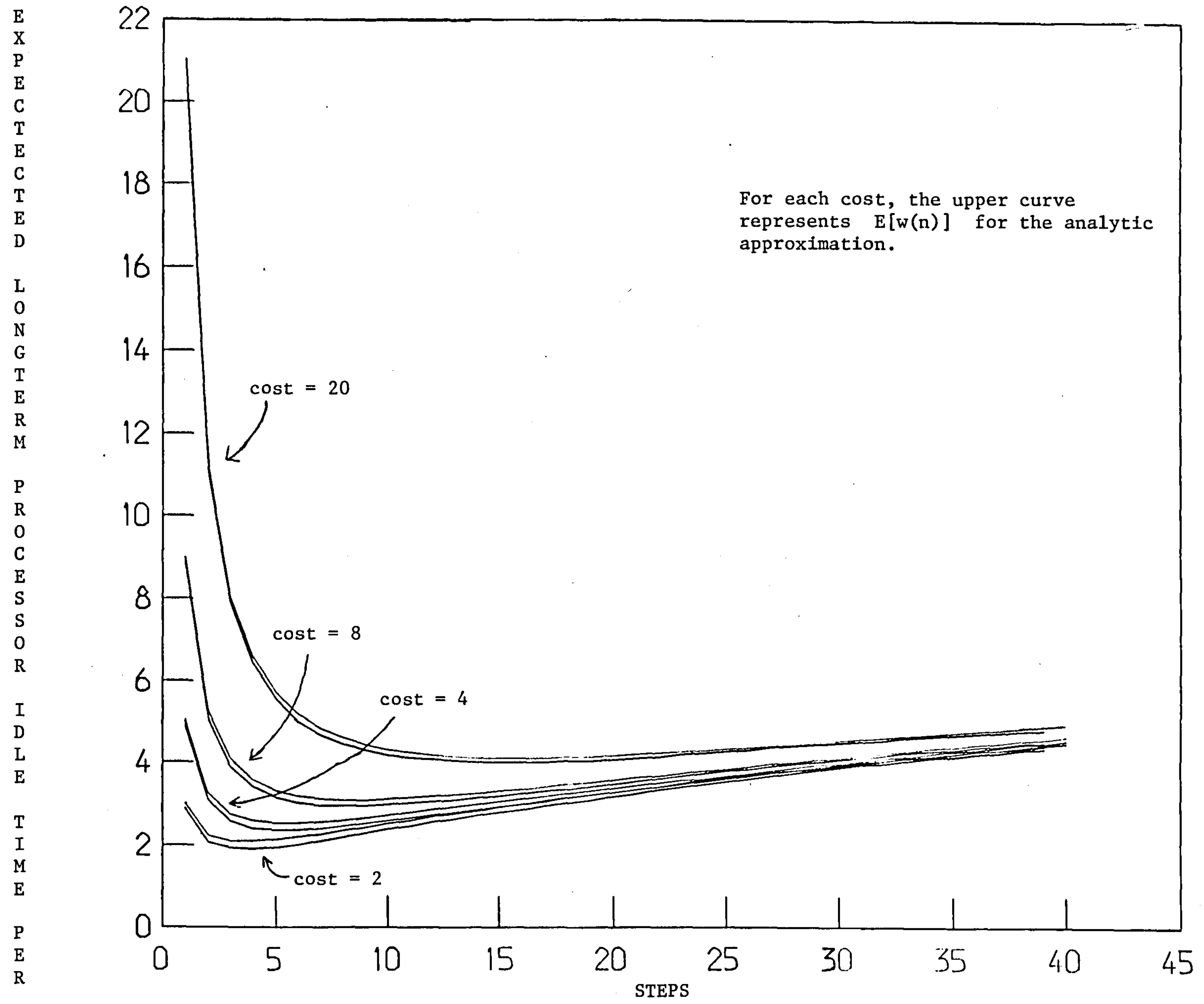
Figure 4



LD Model: Performance of SAR Compared with Performance of Periodic Remapping

64 by 64 activity array initialized with one work unit per activity point. Work unit transition probabilities: up - 0.1, right - 0.1, down - 0.05, left - 0.05. Each data point calculated from 50 sample points.

Figure 5



MUM Model: Analytic Approximation vs. Simulation Derived Values for Expected Longterm Average Processor Idle Time per Step E[w(n)]. Eight chains, each chain has 19 states, p = 0.5, each simulated curve estimated from 100 sample paths.

Figure 6

$$\begin{array}{cccccccc}
1 & 2 & 4 & 1 & 1 & 2 & 9 & 9 \\
1 & 3 & 2 & 4 & 2 & 3 & 9 & 9 \\
4 & 6 & 4 & 3 & 9 & 4 & 8 & 0 \\
3 & 3 & 3 & 6 & 2 & 7 & 4 & 3 \\
1 & 2 & 4 & 1 & 1 & 2 & 4 & 1 \\
7 & 0 & 8 & 5 & 4 & 4 & 5 & 5 \\
5 & 1 & 1 & 1 & 0 & 0 & 2 & 3 \\
9 & 3 & 2 & 1 & 8 & 6 & 2 & 4
\end{array}
\qquad\qquad
\begin{array}{ccccc|ccc}
1 & 2 & 4 & 1 & 1 & 2 & 9 & 9 \\
1 & 3 & 2 & 4 & 2 & 3 & 9 & 9 \\
4 & 6 & 4 & 3 & 9 & 4 & 8 & 0 \\
3 & 3 & 3 & 6 & 2 & 7 & 4 & 3 \\
1 & 2 & 4 & 1 & 1 & 2 & 4 & 0 \\
7 & 0 & 8 & 5 & 4 & 4 & 5 & 5 \\
5 & 1 & 1 & 1 & 0 & 0 & 2 & 3 \\
9 & 3 & 2 & 1 & 8 & 6 & 2 & 4
\end{array}$$

Figure 7a: *Column Partition*

$$\begin{array}{c|c}
\begin{array}{ccccc}
1 & 2 & 4 & 1 & 1 \\
1 & 3 & 2 & 4 & 2 \\
4 & 6 & 4 & 3 & 9 \\
3 & 3 & 3 & 6 & 2 \\
\hline
1 & 2 & 4 & 1 & 1 \\
7 & 0 & 8 & 5 & 4 \\
5 & 1 & 1 & 1 & 0 \\
9 & 3 & 2 & 1 & 8
\end{array}
&
\begin{array}{ccc}
2 & 9 & 9 \\
3 & 9 & 9 \\
4 & 8 & 0 \\
\hline
7 & 7 & 3 \\
2 & 4 & 1 \\
4 & 5 & 5 \\
0 & 2 & 3 \\
6 & 2 & 4
\end{array}
\end{array}$$

Figure 7b: *Column and Row Partitions*

Standard Bibliographic Page

1. Report No. NASA CR-178150 ICASE Report No. 86-45	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle DYNAMIC REMAPPING OF PARALLEL COMPUTATIONS WITH VARYING RESOURCE DEMANDS		5. Report Date July 1986
		6. Performing Organization Code
7. Author(s) David M. Nicol, Joel H. Saltz		8. Performing Organization Report No. 86-45
9. Performing Organization Name and Address Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23665-5225		10. Work Unit No.
		11. Contract or Grant No. NAS1-17070, NAS1-18107
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546		13. Type of Report and Period Covered Contractor Report
		14. Sponsoring Agency Code 505-31-83-01

15. Supplementary Notes

Langley Technical Monitor: J. C. South

Submitted to IEEE Transactions on Computers

Final Report

16. Abstract A large class of computational problems are characterized by frequent synchronization, and computational requirements which change as a function of time. When such a problem must be solved on a message passing multiprocessor machine, the combination of these characteristics lead to system performance which decreases in time. Performance can be improved with periodic redistribution of computational load; however, redistribution can exact a sometimes large delay cost. We study the issue of deciding when to invoke a global load remapping mechanism. Such a decision policy must effectively weigh the costs of remapping against the performance benefits. We treat this problem by constructing two analytic models which exhibit stochastically decreasing performance. One model is quite tractable; we are able to describe the optimal remapping algorithm, and the optimal decision policy governing when to invoke that algorithm. However, computational complexity prohibits the use of the optimal remapping decision policy. We then study the performance of a general remapping policy on both analytic models. This policy attempts to minimize a statistic W(n) which measures the system degradation (including the cost of remapping) per computation step over a period of n steps. We show that as a function of time, the expected value of W(n) has at most one minimum, and that when this minimum exists it defines the optimal fixed-interval remapping policy. Our decision policy appeals to this result by remapping when it estimates that W(n) is minimized. Our performance data suggests that this policy effectively finds the natural frequency of remapping. We also use the analytic models to express the relationship between performance and remapping cost, number of processors, and the computation's stochastic activity.

17. Key Words (Suggested by Authors(s)) load balancing, dynamic load balancing, remapping, adaptive computation, optimal partitioning, multiprocessors	18. Distribution Statement 61 - Computer Programming and Software 62 - Computer Systems Unclassified - unlimited

19. Security Classif.(of this report) Unclassified	20. Security Classif.(of this page) Unclassified	21. No. of Pages 55	22. Price A04

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

NASA Langley Form 63 (June 1985)

End of Document