NASA Contractor Report 178344

ICASE REPORT NO. 87-49

ICASE

OPTIMAL DYNAMIC REMAPPING OF PARALLEL COMPUTATIONS

David M. Nicol

Paul F. Reynolds, Jr.

Contract No. NAS1-18107
July 1987

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

Operated by the Universities Space Research Association

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

Optimal Dynamic Remapping of Parallel Computations

David M. Nicol
Institute for Computer Applications in Science and Engineering
and
The College of William and Mary

Paul F. Reynolds, Jr.
University of Virginia

ABSTRACT

A large class of computations are characterized by a sequence of phases, with phase changes occurring unpredictably. We consider the decision problem regarding the remapping of workload to processors in a parallel computation when (i) the utility of remapping and the future behavior of the workload is uncertain, and (ii) phases exhibit stable execution requirements during a given phase, but requirements may change radically between phases. For these problems a workload assignment generated for one phase may hinder performance during the next phase. This problem is treated formally for a probabilistic model of computation with at most two phases. We address the fundamental problem of balancing the expected remapping performance gain against the delay cost.

Stochastic dynamic programming is used to show that the remapping decision policy minimizing the expected running time of the computation has an extremely simple structure: the optimal decision at any decision step is followed by comparing the probability of remapping gain against a threshold. However, threshold calculation requires a priori estimation of the performance gain achieved by remapping. Because this gain may not be predictable, we examine the performance of a heuristic policy that does not require estimation of the gain. In most cases we find nearly optimal performance if remapping is chosen simply when the probability of improving performance by remapping is high. The heuristic's feasibility is demonstrated by its use on an adaptive fluid dynamics code on a multiprocessor. Our results suggest that except in extreme cases, the remapping decision problem is essentially that of dynamically determining whether gain can be achieved by remapping after a phase change; precise quantification of the decision model parameters is not necessary. Our results also suggest that this heuristic is applicable to computations with more than two phases.

This research was supported in part by the National Aeronautics and Space Administration under NASA Contract NAS1-18107 while the first author was in residence at ICASE, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23665. It was also supported in part by the Virginia Center for Innovative Technology while both authors were in residence at the University of Virginia, Department of Computer Science, Thornton Hall, Charlottesville, VA 22903.

1. Introduction

An important issue in parallel processing is the assignment of workload to processors. A common model of this problem is to assume that a program is composed of a number of communicating modules, and that each module is to be assigned to a processor in the target parallel system. The assignment algorithm takes a global view of the system, and must consider processors' capacity, any special affinity a module has for a processor (e.g. a module may require assignment to a processor with a floating point accelerator), module execution requirements, inter-module communication, and any access to files and data structures that a module may require. An assignment algorithm may assign files to storage devices as well, so we will speak of the *mapping* of the computation, rather than the assignment of modules. One might also view this as a static scheduling of the computation's components. We will assume that global communication is significantly more costly than local communication, as in a message passing architecture.

Any reasonable mapping algorithm must take into account the expected behavior of the mapped computation, because the efficiency of a parallel computation depends heavily on how well its mapping both exploits available parallelism, and minimizes the communication and synchronization overhead. Both of these factors are determined by the underlying stochastic behavior of the computation. Loosely defining a *phase* to be a period during which a computation's behavior exhibits some characteristic particular to the phase, we are interested in computations having multiple phases. Multiple phases may arise in a number of ways. Some algorithms explicitly have different phases and different behavior during each phase; a computation may also change its behavior in response to the behavior of partial results within the computation or in response to input that drives it. If during run-time a phase changes and causes a mismatch between mapping and behavior, performance will deteriorate. It may then be desirable to dynamically remap the computation. Because of the complicated considerations often involved in task and file assignment, it may not be feasible to allow processors to move modules, files, and data structures around in a dynamic decentralized fashion. A global mapping (or remapping) algorithm is better able to consider all aspects of the assignment problem, especially if the parallel system is tightly coupled. In this paper we view dynamic remapping as the dynamic invocation of a static mapping or scheduling algorithm, as a function of the computation's observed behavior.

Parallel scientific computations typically divide workload by partitioning a grid-like data domain. Such computations are composed of numerical calculations at each point in a discretized spatial (or transformed) domain. Workload assignment involves partitioning the domain into *regions* which are then mapped to processors [2]. A processor is responsible for all computations on grid points within regions assigned to it. Data partitioning is attractive because it can effectively exploit parallelism. Partitions are relatively simple to compute, and static partitions are simple to manage at run-time. To dynamically remap a data partitioned computation, a new partition and mapping is centrally computed and reported to the processors who then exchange whatever data is necessary to effect the change. Remapping might be desirable if the number of domain points in a processor's region changes, or if the calculations required at the domain points change. A good example of this phenomenon is the behavior of adaptive gridding techniques used in the solution of time-dependent partial differential equations [3]; grid points are adaptively added and subtracted from the domain in order to resolve interesting (and transient) features in the solution. While the solution is well behaved very few additional grid points are required. Grid points are added in response to some phenomenon appearing (unpredictably) in the solution. Since the domain is originally partitioned by physical region, a large patch of additional grid points can be defined within one processor's region, creating imbalanced workload and poor performance. We will later illustrate this situation with a fluid dynamics code that employs dynamic regridding. Many other numerical techniques adaptively change the grid in response to solution behavior and may thus exhibit phase-like behavior, for examples see [1].

In both of the fore-mentioned types of computation, we can expect run-time behavior to change abruptly, and to the detriment of run-time performance. If the behavior of the computation is completely understood and predictable, then the costs and benefits of remapping can be calculated and weighed against each other. When the phase changes, the action minimizing the computation's remaining execution time can be calculated simply, making the decision problem a nonissue. However, it is much more reasonable to assume that the computation's behavior is not completely understood, and that there is uncertainty in the length of the computation, the occurrence of the phase change, and in the benefits of remapping. This is likely to be the case if the computation is driven in real-time by external data. Dynamic remapping might still improve performance, but remapping raises a number of issues including (1) whether to use global remapping or decentralized and localized remapping, (2) determining when a phase change occurs, and whether remapping after the change will improve performance, (3) determining a new mapping and its implementation, (4) estimating the gains and costs of remapping, (5) determining the performance loss of not remapping after a phase change, and (6) optimally choosing when to remap. This paper treats only the latter issue, which must balance all costs, gains, and uncertainities involved in the decision to remap. We do illustrate solutions to the other issues for our model fluids problem; in general the other issues are problem and system dependent, and are important research issues in their own right. We focus on the remapping decision problem because it brings whatever solutions are found for all of the other issues (aside from (1)) together in a cohesive bundle.

Most of the literature related to remapping takes rather different views of the problem. The work reported in [8], [11], and [29] essentially presumes that jobs arrive at a central dispatcher that assigns jobs to processors. This model of computation behavior does not fit our view of parallel computations. A more recent body of work including [10], [16], [26], [27], [30] allows decentralized assignment decisions to be made dynamically. We feel that it is also important to investigate centralized assignment decisions, especially as technology is successfully driving down the cost of inter-processor communication. Static and dynamic task assignment algorithms are presented in [2], [4], [7], [9], [12], [15], [13], [21], [28]. The dynamic assignment algorithms consider restricted classes of computations; the static assignment algorithms might be used in conjunction with a remapping decision policy if the statically assigned computations abruptly change behavior. A treatment of dynamic remapping of parallel computations whose behaviors change constantly and gradually is given in [20]; we focus here on behavior that changes radically, and abruptly. This paper is an extension of earlier treatments of this problem in [18]; the major difference is this paper's inclusion of multiprocessor results while the work in [18] studies simulations of this problem. Our approach is a variation on aspects of the broad treatment of change detection under uncertainty given in [22]. Our model modifies this analysis by using a different decision cost structure, and by assuming a random computation duration.

This paper shows how to model the remapping decision problem formally as a Markov decision process. Under this model, we demonstrate that the decision policy minimizing the expected running time of the computation has a simple threshold structure: at every decision step in the computation we compute the gain probability that remapping immediately will improve performance, and compare that probability to a threshold pre-calculated specifically for that decision step. Remapping is chosen if the probability exceeds the threshold, otherwise the present mapping is retained. Recognizing that some parameters needed for the threshold calculation may not be known in practice, we investigate the performance of a decision heuristic that chooses remapping as soon as the gain probability exceeds some fixed threshold. This heuristic does not presume knowledge of the performance gain achievable by remapping. The heuristic was tested on an adaptive fluid dynamics code implemented on a multiprocessor, and was found in most cases to give nearly optimal performance. Our results show that for these types of problems the key issue is not *how much* performance gain is possible by remapping, but *whether any* gain is possible. While the model is formulated to allow at most two phases, our results suggest that more complicated models are not necessary. So long as the phases do not occur more rapidly than remapping's ability to amortize costs with gains, our model can be applied by always considering "the next" phase change.

The remainder of the paper is organized as follows. Section 2 describes an adaptive fluid dynamics code used to illustrate concepts employed by our analytic model; we also use it to report on dynamic remapping's measured performance. Section 3 describes our analytic decision model, and identifies important functions related to the remapping problem. Section 4 discusses the optimal decision policy, showing that it is a threshold policy. An optimal algorithm for approximating these thresholds is discussed, and the behavior of the optimal thresholds is examined. Section 5 reports on the performance of an adaptive fluids code which employs dynamic remapping; this code is implemented on the Flex/32 multiprocessor[14]. Performance using dynamic remapping is shown to be substantially superior to the performance of static mapping. Section 6 presents our conclusions, and the Appendix treats analytic issues in detail.

2. A Remapping Model Problem

A one-dimensional fluid dynamics code serves as a model problem. The code numerically simulates the density $\rho(r,t)$ and velocity $v(r,t)$ of a compressible fluid flowing through a tube, as a function of position r and time t. Our implementation employs the ETBFCT code [5] which solves the general continuity equation

$$\frac{\partial\rho(r,t)}{\partial t} = \frac{\partial\left[\rho(r,t)v(r,t)\right]}{\partial r} + \frac{\partial D_1(r,t)}{\partial r} + C_1\frac{\partial D_2(r,t)}{\partial r} + D_3(r,t)$$

where C_1, D_1, D_2, D_3 are problem dependent constants or functions. For simplicity our experiments set all of these to zero. The one dimensional tube model is initially discretized with a grid having one point every h spatial units; we will call this the *coarse* grid. We assume that the value of ρ is known at every grid point at time $t = 0$, that the fluid flows from left to right, and that the density and velocity values of fluid entering the tube are given as needed. Presented with the density and velocity values of ρ throughout the domain at time t_0 and the density of fluid entering the tube at time $t_0 + \Delta t$, ETBFCT numerically integrates the continuity equation in t to solve for fluid behavior at time $t_0 + \Delta t$.

Variant computational behavior is caused by employing an adaptive gridding technique proposed in [3]. Every $5\Delta t$ time units the solution behavior is examined, and additional *fine* grid points with a spacing of $h/6$ are added in subregions where the examination predicts that truncation error will exceed a user defined limit (this mechanism is quite detailed and is beyond the scope of this paper). Every coarse grid point in such a region has a corresponding fine grid point at exactly the same tube position. Similarly, fine grid points are removed from regions where they are no longer needed (coarse grid points are never removed). Figure 1 illustrates how patches of grid points might be applied to the domain. At time t_0, the computation first integrates only the coarse grid, from time t_0 to time $t_0 + \Delta t$. Then, each fine grid is integrated 6 times where each integration uses a time step of $\Delta t/6$; boundary values at the ends of subgrids are interpolated from the corresponding coarse grid values. After fully integrating the fine grids, function values at coarse grid points covered by fine grid points are replaced with the improved function values from the corresponding fine grid points. In its fullest generality, the method described in [3] allows a recursive attachment of finer grids to fine grids. For our purposes only two levels of grid points are necessary to cause load imbalance.

Two factors dominate the execution time of a parallel implementation of our fluids code: communication cost and computation cost. Using ETBFCT the density at position r_0 at time $t_0 + \Delta t$ depends functionally on the density values for all grid points between $r_0 - 5h$ and $r_0 + 5h$ at time t_0 (approximately 50 floating point operations are used to update the density at a coarse grid point). This dependence requires inter-processor communication for points that are functionally dependent but reside on different processors. The domain decomposition and assignment illustrated in figure 2 assigns an equal length of domain to every processor. If processor j is assigned a partition interval containing points $[r_0, \cdots, r_0 + Mh]$, then processor j must send points $[r_0, \cdots ,r_0{+}4h]$ to processor $j{-}1$, and will receive points $[r_0{-}6h, \cdots, r_0{-}h]$ from processor $j{-}1$. A similar exchange with the

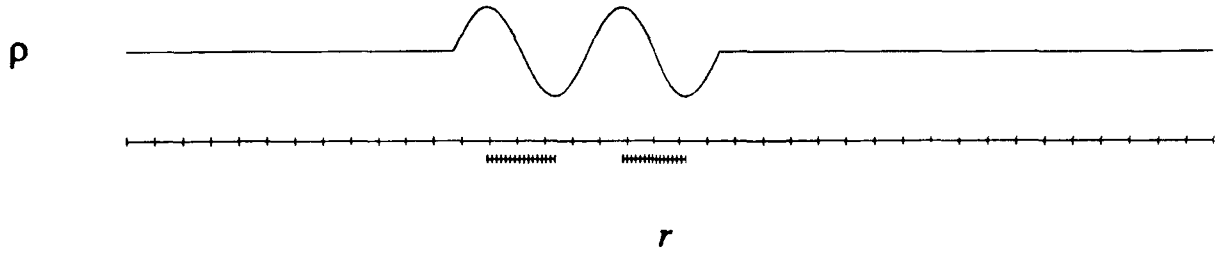

Fig. 1 *Regridding in One-Dimensional Domain*

interval's right boundary points is undertaken with processor j+1. When the computation uses only a coarse grid, this *standard assignment* minimizes the volume of data that must be communicated between processors. Fine grid points tend to be placed in regions where the solution is changing sharply, and these regions tend to migrate from left to right as the computation progresses. The dynamic nature of fine grid creation and migration can pose severe load balancing problems if the standard assignment is used, again shown in figure 2. Figure 3 illustrates the concept of *wrapping*: more partitions than processors are defined and then the partition assignment "wraps" around all processors. Each partition again covers an equal length of the domain. Fine grid intervals have a larger chance of being spread across processors, and so a better load balance can be expected.[1] However, a wrapped assignment requires more inter-processor communication due to the increased number of partition interfaces. In fact, a wrapped assignment requires more computation because of a fixed additional computation cost associated with every partition regardless of size. Consequently, the standard assignment is preferred when little or no dynamic regridding is occurring, while a wrapped assignment is preferred when dynamic regridding causes serious load balancing problems.

Consider the situation where the tube initially has nearly constant density and velocity. As long as fluid entering the tube continues to have this density and velocity very little regridding will occur. Now imagine that the behavior of the inflowing fluid is determined by a real-time process that at some time begins to behave as a wave, and sometime later returns to constant behavior. When a moving wave is in the tube, regridding may cause severe load imbalance. There is a short period between when the wave first enters the tube, and when it has propagated far enough into the tube to allow a wrapped assignment to be of some benefit. We say that the computation's phase changes at the end of this period rather than at the beginning. Consequently the computation's phase change lags behind the physical system's change at the tube's entry point. Another phase change occurs close to the time

[1] A study of wrapped assignments is given in "Principles for Problem Aggregation and Assignment in Medium Scale Multiprocessors", D.M. Nicol and J.H. Saltz, ICASE Report 87-39.

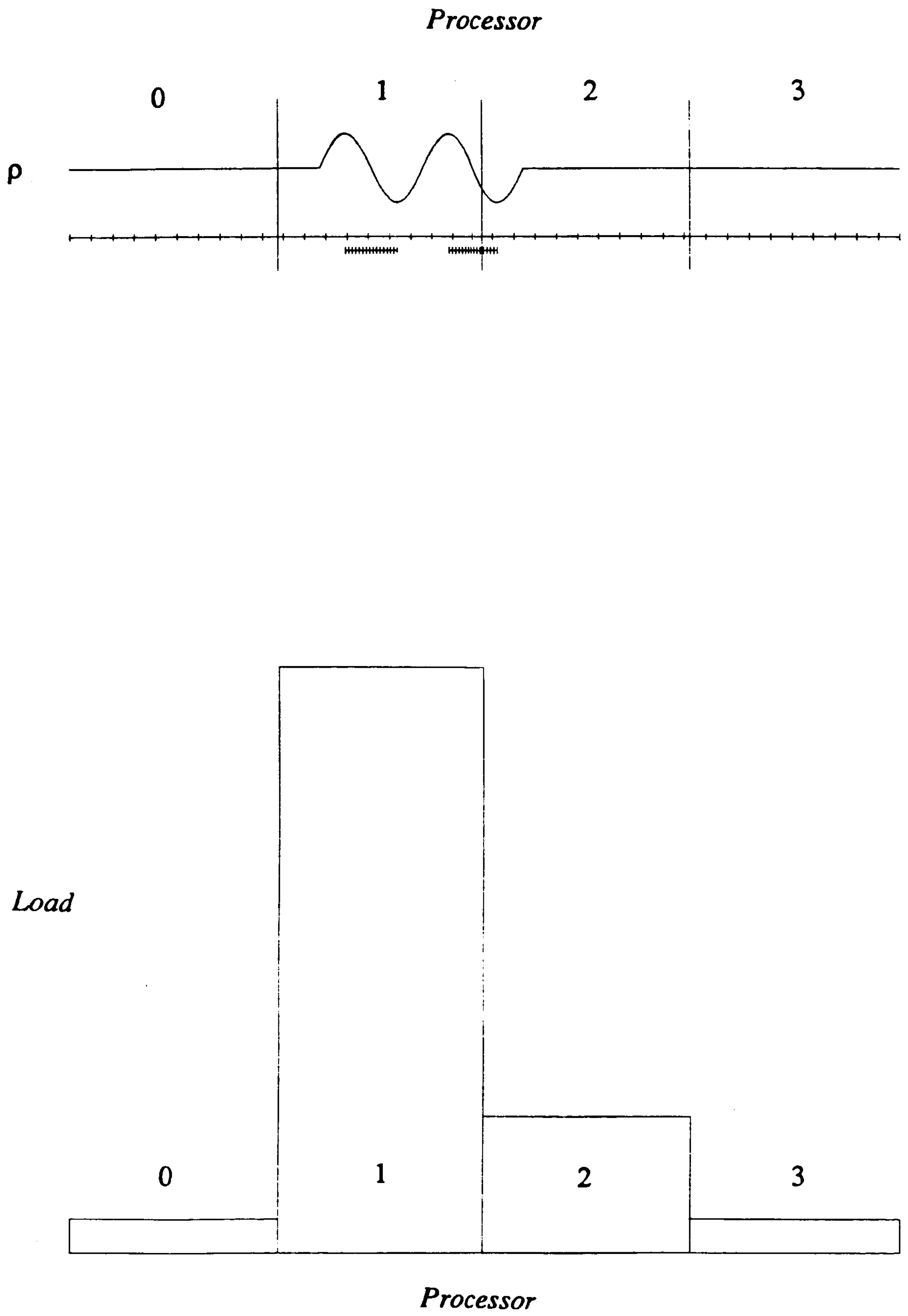

Fig. 2 *Standard Assignment and Load Balance*

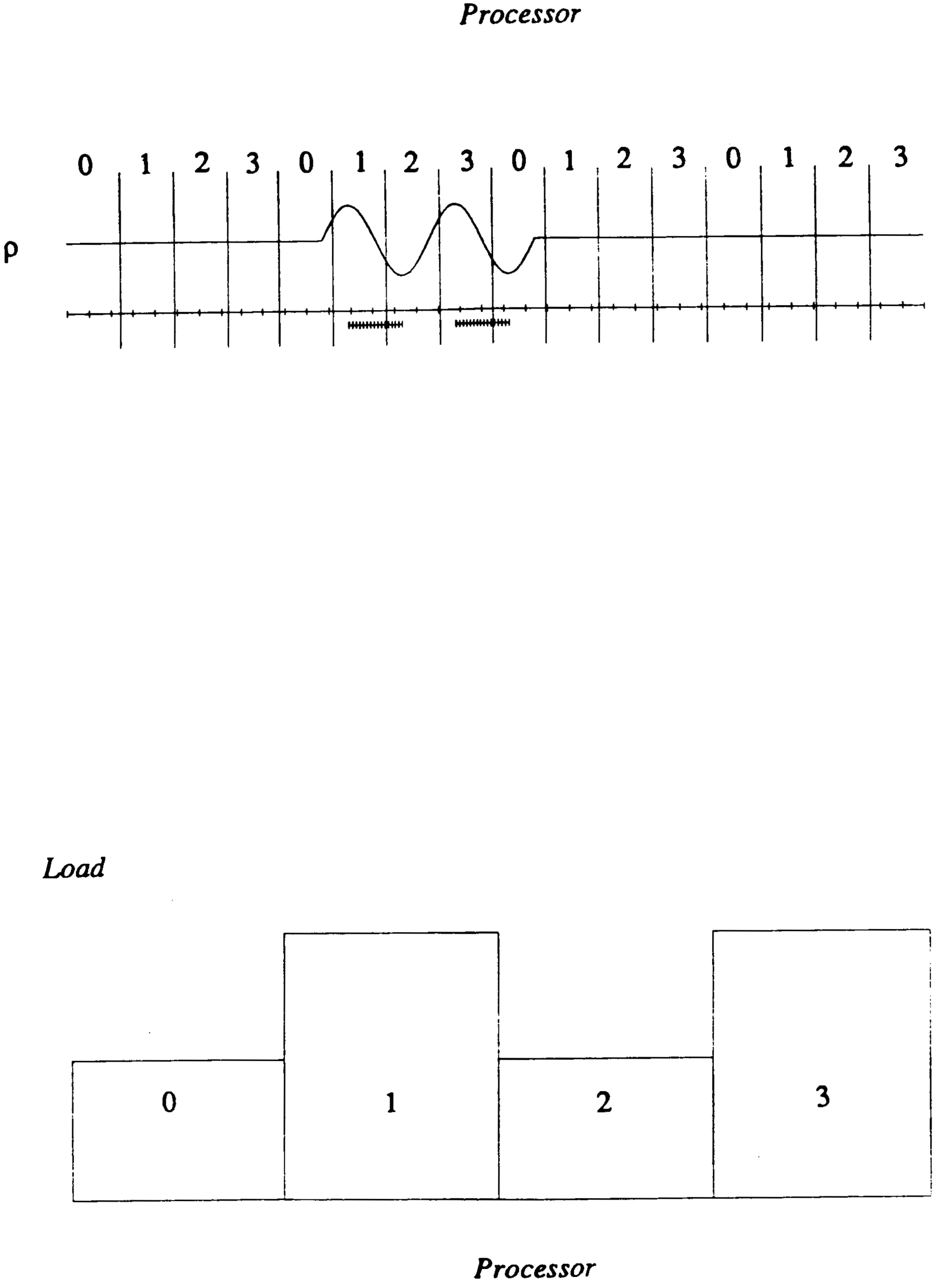

Fig. 3 *Wrapped Assignment and Load Balance*

where the wave completely leaves the tube. Again we define the computation's phase change in terms of when it is advantageous to remap back to the standard assignment.

At the first phase change, switching from the standard mapping to a wrapped mapping may improve the load balance at the price of more communication. A number of issues have to be dealt with if dynamic remapping is to be successfully employed. First, a supervisory process must be able to determine whether performance is better under the standard mapping or under a wrapped mapping. This is accomplished at every regridding time, by requiring every processor to describe (or approximate) to the supervisor its new spatial distribution of grid points. The supervisor then has a picture of workload distribution throughout the entire domain. The supervisor can use its knowledge of the number of operations done at a grid point to estimate computation costs under different mappings. It is important to note that a certain amount of estimation error is unavoidable, and that the potential for a wrong estimate must be dealt with by a remapping policy. A second issue is how we view the remapping problem. This paper considers only one option, that remapping is an activity that occurs infrequently and in response to significant behavior change. Another option is to remap more frequently and tailor each new assignment to the computation's behavior as observed at the time of remapping [20]. A third issue concerns the mechanics of changing from the standard assignment to a wrapped assignment. Because both the standard and wrapped assignments are defined in terms of fixed length domain intervals, it is straightforward for a processor to break up its single large interval into smaller intervals, and to calculate each small interval's new processor location. All data associated with a small interval is communicated to the appropriate processor; consequently, remapping requires a substantial volume of grid points and function values to be communicated. A fourth issue is the one addressed formally by this paper: the decision when and if to adopt a wrapped assignment. The key factors making the remapping decision a non-trivial problem are the uncertainty in the performance estimates under different mappings, the uncertainty in the fluid's future behavior, and uncertainty in the time of termination (which may depend on solution behavior). The mathematical model developed in the next section deals formally with these factors.

3. A Remapping Decision Model

Our application of decision process theory requires that we identify a sequence of *decision steps* in the computation. Every time a decision step is reached, a supervisory process decides whether or not to remap, based on the supervisory processes' knowledge of the computation and behavior within different phases. Our model problem's decision steps are the regridding times. Iterations in an iterative numerical program can serve as decision steps, and natural decision steps could be found in an embedded real-time system that periodically calls monitoring tasks. We define *cycle i* to be the period of computation between decision steps i and i+1. The time required to execute a cycle, a *cycle time*, is assumed to be random with a known and finite mean. For the model problem this randomness describes uncertainty in cycle execution time, particularly as regridding and its effect on performance is not completely predictable. We do *not* assume that cycle time distributions are independent. For simplicity's sake we assume that at most one phase change will occur during the course of the computation. Our empirical results show that carefully determined decision policies are not needed, suggesting that the additional complexity of mathematically treating more than one phase change is not worthwhile.

In the model problem, the time required to execute a cycle depends both on the mapping in use, and the phase. Consequently we define four mean cycle times in the table below:

Cycle Time	Mapping	Phase
e_F	Original	first
e_B	Original	second
e_P	New	first
e_R	New	second

In the model problem $e_P \geq e_F$ reflects the additional communication cost of a wrapped assignment when load balancing is not needed; likewise $e_B \geq e_R$ reflects the bad performance caused by load imbalance under the standard assignment.

Our decision model describes different types of uncertainty in the remapping problem. The uncertainty in termination time is captured by allowing the number of cycles in the computation, N, to be random. We assume that N is bounded from above by some constant M, and that N is independent of the occurrence of a phase change (the independence assumption can be relaxed, but it complicates the analysis). Another source of uncertainty arises as we cannot be sure when or if better performance is possible by remapping. In the model problem this is the uncertainty in when a wave may enter the tube and cause a phase change. This uncertainty is modeled by presuming that the occurrence of a phase change leading to potential remapping gain is random. For the sake of tractability we suppose that the index of the cycle during which the phase change occurs is geometrically distributed, with parameter ϕ. ϕ may be thought of as a phase change rate. Consequently, for every n, ϕ is the probability that the phase change happens at cycle n given that it has not happened before cycle n. Another source of uncertainty is related, but is perhaps more subtle. At a decision step we will employ some (problem dependent) mechanism to test for remapping gain. This mechanism might look for a decline in processor utilizations, or it might be able to examine what code has recently been executed. For example, grid point distributions are used in the model problem to estimate costs under different mappings. Based on such examinations, the mechanism can give us some indication of whether a new mapping is called for, but we cannot be certain that the mechanism is absolutely reliable. It might prematurely report the possibility of gain, or it might fail to report an existing possibility of gain. In the model problem this type of uncertainty arises because only a simple model of program behavior is used to estimate costs; the simplicity introduces some approximation error. This type of uncertainty is modeled by assuming that every invocation of the mechanism has a probability α of prematurely reporting possible gain, and a probability β of failing to report an existing possible gain.

At every decision step in the computation the decision policy decides whether or not to remap. The decision algorithm consults the gain testing mechanism described above, but is distinct from that mechanism. This is especially important because the gain testing mechanism can give a false report. Faced by high remapping costs and poor performance in the event of premature remapping, we do not want to remap simply because one consultation of the gain testing mechanism suggests remapping. Instead, we will compute a *gain probability* that is calculated as a function of the response received from the gain testing mechanism. The gain probability calculated at step n is the probability that if we remap immediately, then the subsequent per cycle execution time would be smaller than if we do not remap. For the model problem this is the same as the probability that the phase has changed (since the phase change there is defined in terms of when remapping leads to performance gains). Formally, let X be the random variable describing the first decision step after which remapping leads to performance gain. We assume that remapping at step n when $n \geq X$ leads to better performance, while remapping at n when $n < X$ does not. The gain probability at step n is denoted p_n, and is simply

$$p_n = Prob\{X \leq n\}.$$

The probability p_0 is taken to be zero. If the first gain test detects no possibility for gain, we have *evidence* that there is no immediate gain from remapping. But the probability of remapping gain can no longer be zero since it is possible for a phase change to have occurred during the first cycle, and it is possible for remapping gain to be achieved, and it is possible that the test mechanism failed to detect the potential gain. The true value of the gain probability in this case depends on the values of ϕ (the probability that the phase changes in the first cycle) and β (the probability that an existing phase change is not detected). Similar observations hold if potential gain is reported. Bayes' Theorem [25] gives us a mechanism for calculating this probability. To use Bayes' Theorem at the *nth* step it is necessary to first compute an *a priori* probability $p_a(p)$, which is the pre-test probability that gain is possible at the *nth* step, given that the probability of possible gain at the $(n-1)st$ step was p:

$$p_a(p) = p + (1 - p)\phi. \tag{1}$$

This expression adds the probability that gain is already possible at the $(n-1)st$ step to the probability that it is not possible at step $n-1$ but will become possible precisely at step n. The gain testing mechanism's report at step n gives further infomation about the possibility of gain; Bayes' Theorem is used to combine that information with $p_a(p)$. If A is an event space partitioned by events $A_1, A_2, \cdots A_k$, and if B is some observed event (not necessarily in A), then Bayes' Theorem states that the *a posteriori* conditional probability of A_i given that B is observed is

$$Prob\{A_i \mid B\} = \frac{Prob\{A_i\}Prob\{B \mid A_i\}}{\sum_{j=1}^{k} Prob\{A_j\}Prob\{B \mid A_j\}}.$$

Let A_1 be the event that gain is possible at step n, A_2 be the event that gain is not possible at step n, and let B represent the gain detection mechanism report. $Prob\{A_1\}$ prior to B's observation is $p_a(p)$, and $Prob\{A_2\} = 1 - p_a(p)$; $1 - \beta$ is the probability that the mechanism reports gain given that gain is possible; and α is the probability that gain is reported given that gain is not possible. If a positive gain report is observed, we use the formulation described above to calculate p_n

$$p_n = p^c(p) = \frac{p_a(p)\cdot(1 - \beta)}{p_a(p)\cdot(1 - \beta) + (1 - p_a(p))\cdot\alpha}. \tag{2}$$

Given a negative indication of potential gain, p_n is similarly defined by $p^{\bar{c}}(p)$:

$$p_n = p^{\bar{c}}(p) = \frac{p_a(p)\cdot\beta}{p_a(p)\cdot\beta + (1 - p_a(p))\cdot(1 - \alpha)}. \tag{3}$$

We require one other related probability. Let $q^c(p)$ be the probability that the gain detection mechanism reports potential gain at step n, given that $p_{n-1} = p$. By conditioning on whether gain is actually possible by step n, it is not difficult to see that

$$q^c(p) = p_a(p)(1 - \beta) + (1 - p_a(p))\alpha. \tag{4}$$

The probability of the mechanism reporting no gain at step n given $p_{n-1} = p$ is simply $q^{\bar{c}}(p) = 1 - q^c(p)$.

A decision to remap incurs two explicit costs. First, there is a delay cost of calculating the remapping, followed by the cost of implementing the new mapping. The first cost does not exist in the model problem, as the wrapped assignment is either computed prior to run-time, or at the time of the gain testing. We combine the two explicit remapping costs costs into a single remapping overhead cost D. We summarize our decision model definitions below.

General decision processes define process *states*. At step n the state of our remapping decision process is the pair $<p_n,n>$. At every decision step, the option chosen by a decision process incurs a cost whose value depends on the chosen option, and on the current process state. The goal is to find a decision policy that minimize the expected sum of costs incurred by the decisions. The costs incurred by our remapping decision process model the expected cycle time of the cycle following the decision, and any remapping overhead costs. For example, if a phase change has occurred but the old mapping is retained, then e_B is the expected cycle time of the next cycle. If remapping is chosen when gain is achievable, then an overhead cost of D is suffered, but then every remaining cycle has a mean

Notation	Definition
n	Decision Step Number
N	Random number of decision points
M	Upper Bound on N
N_n	N given $N \geq n$
$\hat{N}_n$	$E[N_n]$
e_F	Pre-Gain Execution Time, Original Mapping
e_B	Post-Gain Execution Time, Original Mapping
e_P	Pre-Gain Execution Time, New Mapping
e_R	Post-Gain Execution Time, New Mapping
D	Delay to Calculate and Implement New Mapping
α	Gain Test False Alarm Error
β	Gain Test Missed Gain Error
ϕ	Phase Change Gain Rate
$p_a(p)$	A Priori Probability of Gain At Next Decision Step
$p^c(p)$	A Posteriori Probability of Gain After Positive Gain Observation
$p^{\bar{c}}(p)$	A Posteriori Probability of Gain After Negative Gain Observation
$q^c(p)$	Probability of Observing Gain Next Observation
$q^{\bar{c}}(p)$	Probability of Not Observing Gain Next Observation

execution time of e_R. The total computation execution time is the sum of all cycle times plus remapping overhead; an optimal decision policy minimizes the expectation of this sum. We see that the optimal decision policy should depend somehow on the various costs and gains involved in the remapping process, the remaining length of the computation, and the degree of our certainty that gain is possible. One way to express the inter-relationships among all of these concerns is as a stochastic dynamic programming problem. Given gain probability p at step n, let $V(<p,n>)$ denote the expected remaining execution time of the computation if we use the optimal decision policy. In the parlance of Markov decision processes [23], $V(<p,n>)$ is the optimal (stationary) cost function. If we choose to retain the old mapping at step n, the next cycle's expected execution time is

$$pe_B + (1-p)e_F.$$

Now let $E_v(<p,n>)$ be the expected remaining execution time after step n+1, using the optimal decision policy, given gain probability p at step n and retention of the mapping at step n. Since $E_v(<p,n>)$ describes optimal decision policy costs after step n+1, it is stated in terms of $V(<\cdot,n+1>)$. $E_v(<\cdot,n>)$ is a function expressing expected values, taken with respect to the probability of reporting potential gain at step n+1. Given $p_n = p$, p_{n+1} is equal to $p^c(p)$ if the mechanism at step n+1 reports potential gain, this occurs with probability $q^c(p)$. Similar observations apply in the event that no change is observed. It follows that

$$E_v(<p,n>) \;=\; q^c(p)V(<p^c(p),n+1>) + q^{\bar{c}}(p)V(<p^{\bar{c}}(p),n+1>). \tag{5}$$

The expected execution time remaining after step n, achieved by keeping the old mapping now and thereafter using the optimal decision policy is consequently

$$C_t(<p,n>) = pe_B + (1-p)e_F + E_v(<p,n>). \tag{6}$$

We call C_t the *retain cost function.*

We similarly define $C_m(<p,n>)$, the *remap cost function.* $C_m(<p,n>)$ is the expected remaining execution time achieved by choosing to remap now. By choosing to remap, we immediately incur an overhead cost D. If remapping gain is actually possible, every remaining cycle will have mean time e_R; there are a mean number $\hat{N}_n - n + 1$ cycles remaining (recall that $\hat{N}_n$ is the expected value of N given $N \geq n$). Letting X be the first step after which gain is possible, we see that the remap cost function at $<p,n>$ conditioned on $X \leq n$ is

$$C_m(<p,n> \mid X \leq n) = D + (\hat{N}_n - n + 1)e_R.$$

If remapping is chosen prematurely, we assume that this is determinable after suffering the cost D [2], that the computation uses the old mapping for cycle n, and that the decision process is free to choose remapping at a later step. Knowing that remapping is premature affects the gain probability: it becomes zero. The remap cost function at $<p,n>$ conditioned on $X > n$ is thus

$$C_m(<p,n> \mid X > n) = D + e_F + E_v(<0,n>).$$

Combining these expressions gives the unconditional remap cost function

$$C_m(<p,n>) = D + p(\hat{N}_n - n + 1)e_R + (1 - p)(e_F + E_v(<0,n>)). \tag{7}$$

This formulation assumes that once remapping is chosen and gain is found, the decision process stops and no further remapping is considered. This is not realistic for a computation with more than two phases, but does allow a tractable treatment of the remapping decision problem. We will later argue that a more complicated multi-phase model is probably not necessary. Finally, we will find it convenient to define $C_l(<p,n>)$ and $C_m(<p,n>)$ for $n > N$: $C_l(<p,n>) = 0$ and $C_m(<p,n>) = D$. It is clear that once the computation stops we want to "retain" and not "remap".

The optimal decision from state $<p,n>$ is given in terms of $C_m(<p,n>)$ and $C_l(<p,n>)$. The principle of optimality states that

$$V(<p,n>) = \min \begin{cases} C_m(<p,n>) \\ C_l(<p,n>) \end{cases},$$

and that the optimal decision at step n given $p_n = p$ is the decision whose cost function minimizes the

[2] Our results are not substantially affected if we do not assume this checking ability. The form of the remap cost function changes, but the single threshold decision policy is still derivable. The model fluids problem is actually better suited to the modified formulation.

right hand side of the equation above. The following section shows that $V(< \cdot ,n>)$ has a useful structure, and that the optimal decision from state $<p,n>$ is nicely characterized.

4. Optimal Decision Policy Thresholds

The gain probability is a key factor in our decision process. In this section we show that the optimal decision policy is a threshold policy: for every step n there is a threshold $\pi_n \in [0,1]$ such that the optimal decision in state $<p,n>$ is to remap if $p > \pi_n$, and retain if $p \leq \pi_n$. We then show why exact calculation of the thresholds $\{\pi_n\}$ is computationally intractable, report on an optimal approximation technique having bounded error, and graphically illustrate the behavior of the $\{\pi_n\}$ as a function of n.

The following lemma provides the fundamental reasons for the optimal policy structure. Its proof is somewhat lengthy, and is given in the Appendix.

LEMMA 1 :

- For all n, $C_m(<p,n>)$ is a linear function of p;
- For all n, $C_t(<p,n>)$ is a piecewise linear concave function of p;
- There exists $n_0 \in [0,\infty]$ such that for all $n \geq n_0$, $C_t(<p,n>) \leq C_m(<p,n>)$ for all $p \in [0,1]$;
- If $n < n_0$, $C_m(<1,n>) \leq C_t(<1,n>)$.

□

Consider the implications of Lemma 1. For any step $n \geq n_0$, the retain decision cost function is always less than the remap decision cost function, implying that we should retain regardless of the value of the gain probability. In this case, the optimal decision threshold is degenerate, $p_n = 1$. For $n < n_0$ we know that the linear remapping cost function is less than the concave retain cost function at $p = 1$. It is therefore geometrically impossible for $C_t(<\cdot,n>)$'s functional curve to intersect $C_m(<\cdot,n>)$'s functional curve more than once, as illustrated by figure 4. If $C_t(<\cdot,n>)$ and $C_m(<\cdot,n>)$ intersect at $\pi_n < 1$, then $C_t(<p,n>)$ is less than $C_m(<p,n>)$ for $p \in [0,\pi_n]$, and $C_m(<p,n>)$ is less than $C_t(<p,n>)$ for $p \in [\pi_n,1]$. It follows that the optimal decision from state $<p,n>$ is to retain if $p \leq \pi_n$, and to remap if

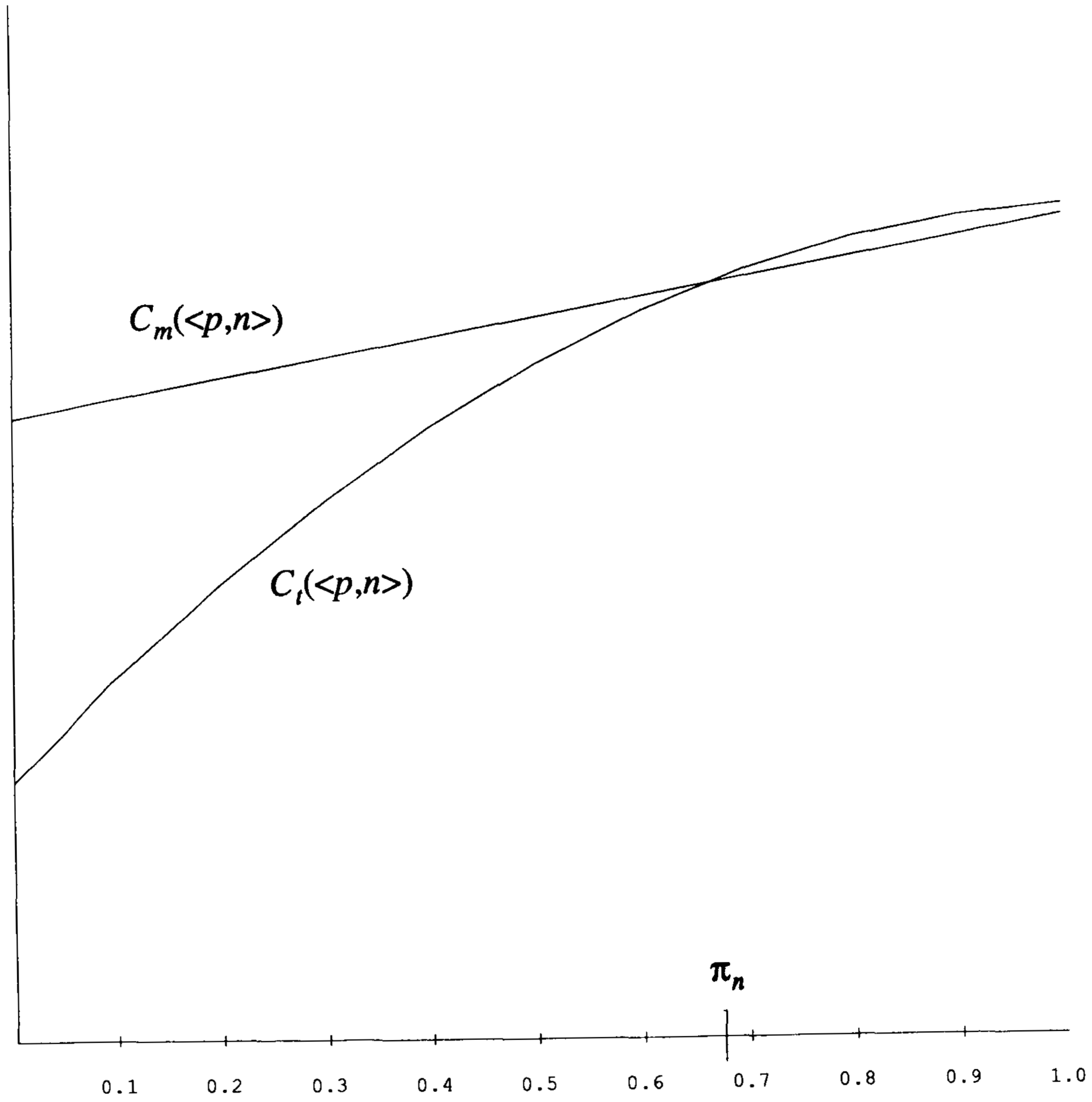

Fig. 4 *Intersection of $C_t(<p,n>)$* and $C_m(<p,n>)$

$p > \pi_n$. We summarize this result:

THEOREM 1 : For every step n, there exists a π_n such that the optimal decision from $<p,n>$ is to remap if and only if $p > \pi_n$.

□

If all of the decision model parameters are known, then in theory we can solve the equations describing $V(<p,n>)$ and determine each optimal threshold. In practice there are significant obstacles to this procedure. Most importantly, we may not be able to quantify the model parameters. In addition, a computational problem arises from the fact that the optimal cost functions $V(< \cdot ,n>)$ are all piece-wise linear. If a piece-wise linear function changes its linear description at domain point d, we will call d a *transition* point. For any piece-wise linear function g on [0, 1], let $S(g)$ be the set of g's transition points. In [17] we show that

$$S(C_t(< \cdot ,n-1> \mid N=m)) \;=\; \{q \geq 0 \mid p^c(q) = d \;\; or \;\; p^{\bar{c}}(q) = d \text{ for some } d \in S(V(< \cdot ,n> \mid N=m)\}.$$

This means that every transition point for $V(< \cdot ,n> \mid N=m)$ can give rise to two distinct transition points for $C_t(< \cdot ,n-1> \mid N=m)$. Any of these transition points greater than π_{n-1} will not appear in $S(V(< \cdot ,n-1> \mid N=m))$; nevertheless, we see that the number of line segments defining $V(< \cdot ,n>)$ essentially doubles at every step of the recursive solution. Then in general, an exact solution is not computationally feasible. However, in [17] we describe an approximation procedure that estimates $V(< \cdot ,n>)$ to any desired degree of accuracy. Furthermore, at every step this procedure is linear in the number of transition points, and over a broad class of approximations, it minimizes the number of transition points required to achieve the desired accuracy. Our computational experience with this approximation shows that it is quite efficient . With the scale of parameter values we used, generally fewer than 200 transition points in the approximation bounded the error in approximating $V(< \cdot ,n>)$ for each n by 10^{-5}.

Figure 5 illustrates the behavior of the optimal thresholds for four different sets of parameter values. For computational simplicity, the number of cycles in the computation was assumed to be constant. The optimal thresholds tend to remain relatively constant, except for n nearest N. Note also that as the per cycle gain $G = e_B - e_R$ increases (for fixed D), the converged value of the optimal threshold decreases. This is completely intuitive, because the smaller the gain from remapping is, the more certain we should be that gain is possible before choosing remapping and suffering the attendant overhead. Another tendency is that the region where $\pi_n = 1$ increases as D increases. It is also intuitively obvious that this should be true, because the expected overall remapping gain depends on the expected remaining length of the computation. If this is small, then a large remapping overhead may not be amortized, and it is better to simply suffer the "bad" cycle times e_B until termination. While the behavior of the optimal thresholds is quite intuitive, it is also important to remember that N is random, and is not known prior to the computation. Optimal remapping requires the computation to rely upon the threshold's ability to balance costs and uncertainties. The optimal decision thresholds precisely account for termination time uncertainty in a way that the unaided computation cannot.

5. Remapping the Model Problem on a Multiprocessor

To illustrate the utility of dynamic remapping, the model fluids problem described in section 2 was implemented on the Flex/32 [14] multiprocessor at the NASA Langley Research Center. The Flex has 20 processors, each having about 1M bytes of local memory. All processors have bus access to a 2M byte common memory. The model problem was written using a message passing paradigm; the Flex's common memory was used solely to implement message passing. In addition, the message-passing software was written to allow the simulation of longer message passing delays than would otherwise be suffered. Our experiments employed this feature to demonstrate that the heuristic need not precisely quantify costs in order to work well.

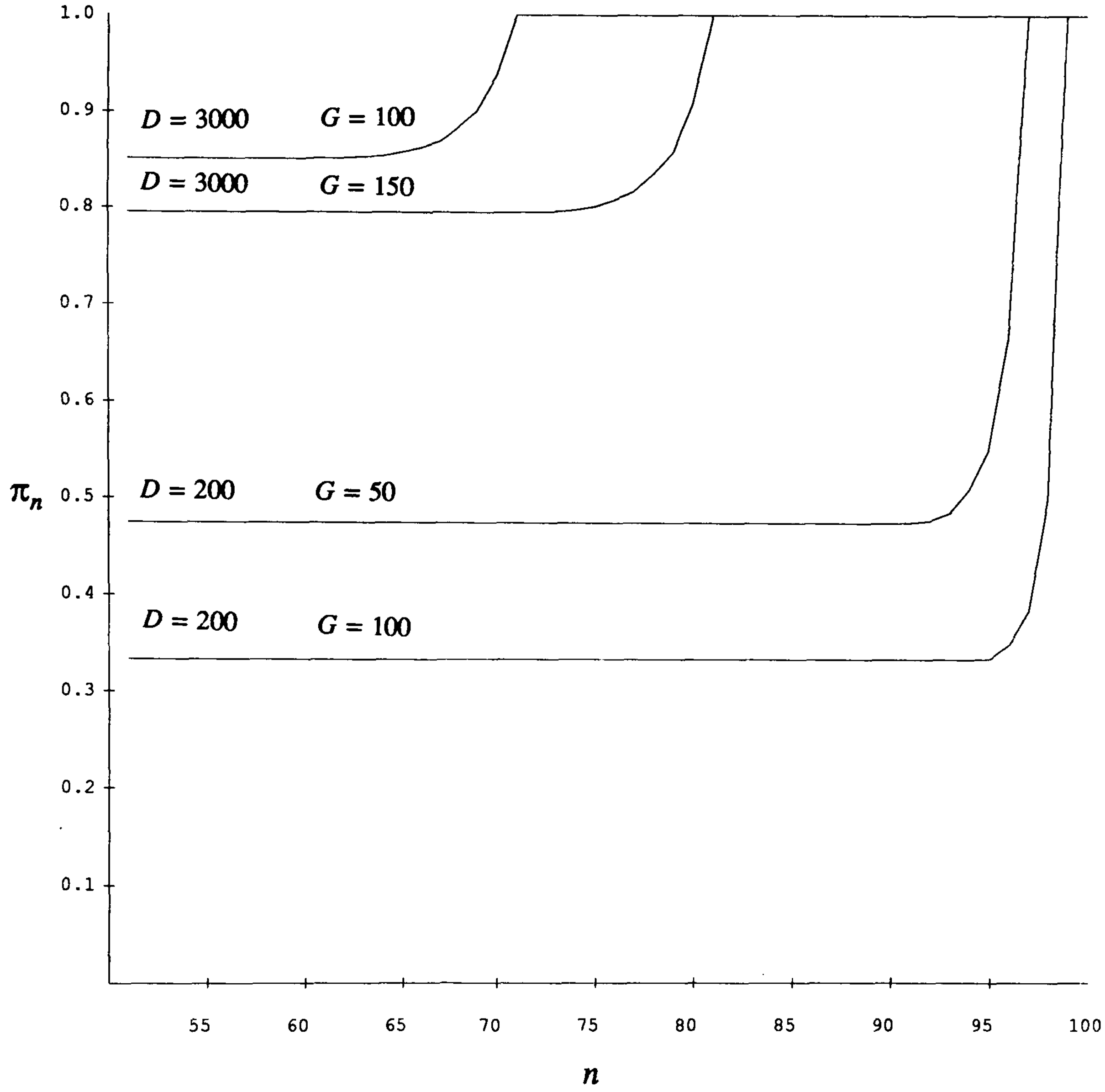

Fig. 5 *Behavior of π_n*

512 coarse grid points were used, spread across sixteen processors. Each computation consisted of 400 basic time-steps, leading to 80 decision steps. A computation tended to require a few minutes of wallclock running time (the best speedup for this problem size was only about 7; speedup improves by increasing the size of the grid, but increasing the size of the grid increases the wallclock running time, a scarce resource). Our experiments compared the performance of the heuristic remapping policy (with $\rho = 0.7$) with the performance of (1) a static standard assignment, (2) a static wrapped assignment with four partitions to each processor, and (3) an optimal policy created by solving for the $\{\pi_n\}$ using estimated problem parameters [3] . Our experiments show that if the entry time of a wave is unknown, then our remapping heuristic strongly outperforms either static mapping. The heuristic and the optimal policy are seen to have nearly identical performance. These results were achieved under varying simulated costs of communication.

Our experiments use a coarse grid spacing of $h = 0.25$, and a basic time step of $\Delta t = 0.025$. The fluid velocity at a point is taken to be the fluid density at the point. The fluid density is initially constant at 5.0 throughout the tube. Entering wave densities are centered at 5.0, have amplitude 0.25, have period 8.0, and last for 1.6 units of time. ϕ is taken to be 1/400, α and β are both taken to be 0.1.

The cycle time values required for the optimal policy calculation were estimated as shown below by averaging measured cycle times within different phases.

Parameter	*Seconds*
e_F	*1.5*
e_P	*2.75*
e_B	*9.96*
e_R	*7.4*
D	*1.2*

There is a discrepancy between the behavior of the model problem and the analytic model: cycle times

[3] The optimal policy here used a slightly different decision model. The modified model does not employ a checking mechanism after computing a new partition, so that a premature decision causes performance to suffer due to unnecessarily high communication costs. This latter decision model better reflects the behavior of the model problem.

actually depend on whether the entire wave is in the tube. As more of a wave enters the tube, more regridding occurs in the wave region, and the cycle times rise. Consequently there is a transition period between the initial entry of the wave, and the time when the entire wave is present in the tube. The analytic model does not capture the transition period, and consequently the "optimal" decision policy thresholds we calculate are not truly optimal. The effects of this error are discernible, but not profound.

Figure 6 plots the relative difference between running times under constant threshold remapping, and the two static mapping policies discussed in section 2. Figure 6 plots data from runs which did not artificially delay communication, and from runs which did. The extra simulated communication costs added approximately three and one half milliseconds to the transmission time of every 32 word message packet. The vertical axis plots the relative difference $(T_s-T_d)/T_d$, where T_s is the running time under a static mapping, and T_d is the running time under dynamic remapping. For example, a plotted value of 0.5 for some statically mapped problem means that the static mapping's running time is 50% worse than that of dynamic remapping for that same problem. The horizontal axis plots the fraction of the computation occurring before the wave enters the tube. This fraction has a strong impact on the running times of the static mappings. When the wave is present for almost the entire computation, then naturally the standard static mapping will suffer severely while the static wrapped assignment enjoys the benefits of load balancing. Similarly, if the wave does not enter the tube until near the computation's termination then the static standard mapping will enjoy low communication volume while the static wrapped assignment is forced to communicate much more boundary information. The critical observation is that chosing a static mapping when there is uncertainty in the wave's entry creates significant risk that the wrong static mapping is chosen. This same observation will hold if the computation undergoes more than one phase change: significant performance losses are suffered whenever the mapping is mis-matched with the computation's phase. It is clear that dynamic remapping effectively deals with these performance problems. It is also clear that while the additional communication delay does affect performance, the heuristic still works well. This is important, because the gain

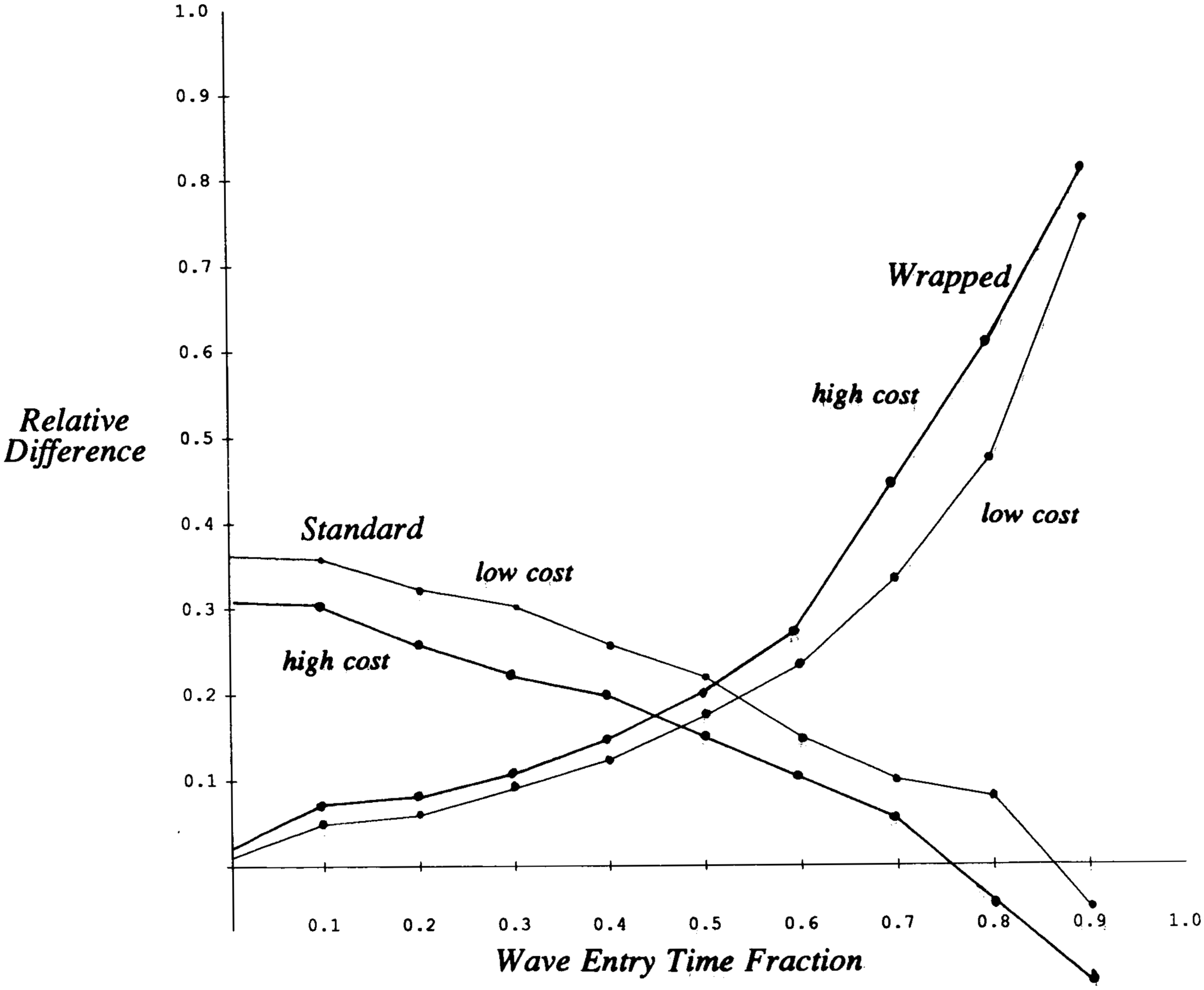

Fig. 6 *Performance of Static Mapping Relative to Dynamic Remapping*

measurement mechanism did not employ estimates of communication costs. Similar results were noted with other choices of problem parameters, and other fluid behavior (e.g. an entering square wave).

We did not plot the performance of the optimal policy because it is so close to the heuristic's. For all but the rightmost sample point the optimal policy was less than 0.1 percent faster than the heuristic (and never slower). In fact, the optimal thresholds (0.925) during the early part of the low-cost-communication computation were significantly higher than the fixed threshold (0.7) ; before deciding to remap the optimal policy generally required one more positive gain report than the heuristic policy did. The heuristic did 3% better than "optimal" when the wave entered just at the end of the computation. This minor discrepancy arises because the analytic model over-estimates the benefit of remapping during the wave's entry transition period. The heuristic's near optimality supports the same observation made with the simulation study reported in [19].

The data presented in figure 6 indicates that precise quantification of remapping gain is not necessary. The gain testing mechanism was written so that communication costs were not considered, yet the heuristic worked nearly optimally for two different communication cost functions. The poor showing of the static wrapped assignment shows the severe effect that unnecessary communication has on performance, and yet the remapping policy provides substantial gains without any consideration at all of this cost. The fact that there *is* gain is enough to achieve that gain. The relatively low cost of remapping is a strong contributing factor to this good performance. We can expect the volume of remapping communication (and hence its cost) to be much larger for adaptive gridding in two and three dimensions. In this case intelligent remapping decision policies are even more important to ensure good performance.

The statistical updating of p_n plays an important role in the heuristic's good performance. Our experience with the code shows that false alarms from the gain testing mechanism occur often enough to make them a potential problem. If we were to remap *whenever* the gain mechanism indicated gain was possible we would often be premature. The Bayesian updating of p_n combined with a reasonably high decision threshold provides good protection against premature remapping.

Another important factor in the heuristic's good performance is the fact that the length of time between a phase change and program termination was generally long enough to amortize the cost of remapping. The heuristic should work equally well on computations with more than one phase, provided that the average time between phase changes is long enough and the remapping gain large enough to amortize the cost of remapping between phases.

The data reported here is obviously not an exhaustive performance study of dynamic remapping, although extensive testing of a simulation model[19] supports the conclusions suggested by our fluids code data, and supports our contention that the two-phase model presented here is adequate for computations with more phases. Our current research includes more extensive testing of different types of remapping on different problems and different architectures. Rather, the data we present should be taken as an existence proof of the utility of dynamic remapping. Much more work is needed before the user of a general parallel code can intelligently determine whether his code benefits from remapping, and what form that remapping should take.

6. Conclusions

An effective mapping of workload to processors in a parallel processing system must make certain assumptions about the computation's running behavior. The behavior of many computations is characterized as a sequence of phases, where behavior within a phase is fairly stable, but the behavior between two phases can be quite different. It is therefore possible for a mapping to become ineffective when a phase change occurs, so that dynamically *remapping* the computation may be required to maintain good performance. The decision to remap must take into account the performance gains and costs involved, and must deal with uncertainty in remapping gains, the computation's future behavior, and the computation's termination time. We have modeled this decision problem with a Markov decision process, and determined the structure of the optimal decision policy. However, calculation of this policy requires estimation of parameters that may not be known a priori. We therefore studied the performance of a simple threshold heuristic that does not assume knowledge of remapping's costs and gains. A study of this heuristic on a multiprocessor fluids code shows that this heuristic works remarkablely well, and shows that the remapping decision problem does not require precise estimates of these parameters. The key issue for the remapping decision problem is thus seen to be the relatively accurate assessment of when remapping leads to performance gains.

There are certainly limitations to the approach we've taken here. To use our policy it is necessary to have enough fore-knowledge of the computation to be able to write code that dynamically analyzes behavior and looks for remapping gain. It was straightforward to analyze the model fluids problem, but even there we needed to know how much work was associated with every grid point, and that the work at every point was essentially identical. We anticipate that looking for remapping gain is a non-trivial problem with more complicated non-numerical codes. Until a method is developed to automatically test for remapping gain for a general problem, our approach is not a good candidate for an automatic load balancing policy at the operating system level. Nevertheless, there will always be codes that people agonize over in order to get the best possible performance; when those codes have unpredictable phase-like behavior our approach can significantly improve performance.

Appendix

In this appendix we prove Lemma 1 from section 3, and discuss an algorithm for approximating $V(<\cdot, n>)$. We first prove Lemma 1.

Proof of Lemma 1

Some of our analysis conditions on the value of N. We use the notation $g(<p,n>|N=m)$ to denote the value of function g at state $<p,n>$ given that $N = m$. We will also say that a function g is *plcc* if it is piece-wise linear, continuous, and concave.

Lemma 1's first claim is that for every n, $C_m(<p,n>)$ is a linear function of p. This is easily seen from its definition in equation (6). Lemma 1's second claim is that for every n, $C_m(<p,n>)$ is a *plcc* function of p. This result follows primarily from the following lemma reported in [22] and stated in terms of our notation:

LEMMA A-1 : Suppose that $N = m$. If $V(<p,n+1>|N=m)$ is a *plcc* function of p, then $E_v(<p,n>|N=m)$ is a *plcc* function of p.
□

Lemma 1's second claim is established by showing that for every fixed $n \geq 0$, $V(<p,n>)$ and $C_t(<p,n>)$ are *plcc* functions of p. First condition on $N = m$ for some m. Recalling that we have defined $C_t(<p,n>) = 0$ and $C_m(<p,n>) = D$ for $n > m$, it is clear that $V(<p,n>) = C_t(<p,n>) = 0$ for $n > m$. For $n \leq m$ we will show inductively that $V(<p,n>|N=m)$ and $C_t(<p,n>|N=m)$ are *plcc* functions of p. For the base case let $n = m$. For any $p \in [0,1]$,

$$C_t(<p,m>|N=m) = pe_B + (1-p)e_F + E_v(<p,m>|N=m)$$

$$= pe_B + (1-p)e_F$$

since $V(<p,m+1>|N=m) = 0$ for all p. We also observe that $C_m(<p,m>|N=m)$ is *plcc* since it is linear. The class of *plcc* functions is closed under the point-wise minimum operation; $V(<p,m>|N=m)$ must also be *plcc*, establishing the induction base.

For the induction hypothesis we suppose that both $C_t(<p,n+1>|N=m)$ and $V(<p,n+1>|N=m)$ are *plcc* functions of p for some $n \le m-1$. Lemma A-1, and the closure of *plcc* functions under addition and point-wise minimum again ensure that $C_t(<p,n>)$ and $V(<p,n>)$ are *plcc* functions of p, completing the induction.

To complete the proof, we note that the class of *plcc* functions is also closed under scalar multiplication, and observe that

$$V(<p,n>) = \sum_{m=1}^{M} Prob\{N=m\} \cdot V(<p,n>|N=m)$$

and

$$C_t(<p,n>) = \sum_{m=1}^{M} Prob\{N=m\} \cdot C_t(<p,n>|N=m)$$

To help establish Lemma 1's third and fourth claims, we analyze the values of $C_m(<p,n>)$ and $C_t(<p,n>)$ at $p=1$. Key results are given by Lemma A-2.

LEMMA A-2 : Either

(i) $V(<1,n>) = C_m(<1,n>)$ for all n for which $Prob\{N=n\} \neq 0$; or

(ii) $V(<1,n>) = C_t(<1,n>)$ for all n for which $Prob\{N=n\} \neq 0$; or

(iii) There exists an n_0 (possibly ∞) such that for all $n < n_0$, $V(<1,n>) = C_m(<1,n>)$, and for all $n \ge n_0$ for which $Prob\{N=n\} \neq 0$, $V(<1,n>) = C_t(<1,n>)$.

PROOF: We condition on $N=m$, for any $0 \le m \le M$. Let K be the largest integer such that $(e_B - e_R) \cdot K \le D$. Simple algebra (omitted here) establishes the inductive proof that for all n such that $m - K < n \le m$,

$$V(<1,n>|N=m) = C_t(<1,n>|N=m) = (m-n+1)e_B;$$

and that for $0 \le n \le m-K$,

$$V(<1,n>|N=m) = C_m(<1,n>|N=m) = D + (m-n+1)e_R$$

and

$$C_t(<1,n>|N{=}m) = e_B + D + (m - n)e_R.$$

Define $d(n|N{=}m)$ to be the conditional difference $C_m(<1,n>|N{=}m) - C_t(<1,n>|N{=}m)$, and $d(n)$ to be the unconditional difference $C_m(<1,n>) - C_t(<1,n>)$. From the equations above, we see that

$$d(n|N{=}m) = \begin{cases} D & \text{for } n > m \\ D - (e_B - e_R)\cdot(m - n + 1) & \text{for } m - K < n \le m. \\ (e_R - e_B) & \text{for } n \le m - K \end{cases}$$

It follows from the definition of K that $d(n|N{=}m)$ is an increasing function of n. From this we infer that $d(n)$ increases in n, since

$$d(n{+}1) - d(n) = \sum_{m=1}^{M} Prob\{N = m\}\left[d(n{+}1|N{=}m) - d(n|N{=}m)\right] \ge 0.$$

This relation shows that $C_m(<1,n>) - C_t(<1,n>)$ is increasing in n. Then case (i) occurs if $d(n)$ is negative for all n, case (ii) occurs if $d(n)$ is positive for all n, and case (iii) occurs if $d(n)$ changes sign at $n = n_0$.

□

To establish Lemma 1's third claim we will show that $C_t(<p,n>)$ is linear in p whenever $n \ge n_0$. Since $C_m(<\cdot,n>)$ is always linear, and $C_t(<0,n>) \le C_m(<0,n>)$ for all n, and $C_t(<1,n>) \le C_m(<1,n>)$ when $n \ge n_0$, it follows directly that C_t and C_m cannot intersect, so that $C_t(<p,n>) \le C_m(<p,n>)$ for all $p \in [0, 1]$.

LEMMA A-3 : If $n \ge n_0$, then $C_t(<p,n>)$ is linear in p, and $V(<p,n>) = C_t(<p,n>)$ for all $p \in [0,1]$.

PROOF: We proceed by induction. M is the largest integer such that $Prob\{N = M\} \ne 0$, so that $V(<p,M{+}1>) = 0$ for all p and

$$V(<p,M>) = Prob\{N = M\}\cdot\min\begin{cases} D + pe_R + (1 - p)e_F \\ pe_B + (1 - p)e_F \end{cases}.$$

Presuming that $n_0 < M$, we have $C_t(<1,M>) \le C_m(<1,M>)$ and $C_t(<0,M>) \le C_m(<0,M>)$, so that

$$V(<p,M>) = Prob\{N = M\}\, C_t(<p,M>) = Prob\{N = M\} \left[e_B + (1 - p)e_F \right]$$

which is linear in p. The induction base is thus satisfied.

For the induction hypothesis, we suppose there is an $n > n_0$ such that $V(<p,n+1>) = C_t(<p,n+1>)$ for all $p \in [0,1]$, and that $C_t(<p,n+1>)$ is linear in p. We know that

$$C_t(<p,n>) = Prob\{N \geq n\} \left[p \cdot e_B + (1 - p)e_F \right] + q^c(p) \cdot V(<p^c(p),n+1>) + q^{\bar{c}}(p) \cdot V(<p^{\bar{c}}(p),n+1>),$$

and the induction hypothesis states that

$$V(<p,n+1>) = A \cdot p + B$$

for some A and B. Equations (2), (3), and (4) show that that $p^c(p) = \dfrac{p_a(p)(1 - \beta)}{q^c(p)}$ and that $p^{\bar{c}}(p) = \dfrac{p_a(p)\beta}{q^{\bar{c}}(p)}$; it follows that

$$C_t(<p,n>) = pe_B + (1 - p)e_F + Ap_a(p) + B$$

which is linear in p since $p_a(p)$ is linear in p. Since $C_m(<p,n>)$ and $C_t(<p,n>)$ cannot intersect it follows directly that $C_m(<p,n>)$ exceeds $C_t(<p,n>)$ for all $p \in [0, 1]$. Thus $V(<p,n>) = C_t(<p,n>)$, completing the induction.

□

References

[1] I. Babuska , Ed. *Adaptive Computational Methods for Partial Differential Equations*, SIAM, Philadelphia, 1983.

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

[3] M.J. Berger, J. Oliger, Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations, *Journal of Computational Physics*, vol. 53, (1984), 484-512.

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

[5] D.L. Book, Ed., *Finite-Difference Techniques for Vectorized Fluid Dynamics Calculations*, Springer-Verlag, New York, 1981.

[6] G. Browning, H.-O. Kreiss, J Oliger, Mesh Refinement, *Mathematics of Computation*, Vol. 27, (1973), pp 29-39.

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

[8] Y. Chow and W. Kowhler, Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System, *IEEE Trans. on Computers, C-28, 5*, (May 1979), 354-361.

[9] T. C. Chou and J. A. Abraham, Load Balancing in Distributed Systems, *IEEE Trans. on Software Eng., 8*, 4 (July 1982), 401-412.

[10] D.L. Eager, E.D. Lazowska, J. Zahorjan, Adaptive Load Sharing in Homogeneous Distributed Systems, *IEEE Trans. on Software Engineering*, Vol SE-12. No. 5, (May 1986), 662-675.

[11] 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.

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

[13] P.R. Ma, E.Y.S. Lee, and M. Tsuchiya, A Task Allocation Model for Distributed Computing Systems, *IEEE Trans. on Computers, C-31, 1*, January 1982, 41-47.

[14] N. Matelan, The Flex/32 MultiComputer, *Proc. 12th International Symposium on Computer Architecture*, Computer Society Press, Los Alamitos, CA, June 1985, 209-213.

[15] D.I. Moldovan and J.A.B. Fortes, Partitioning and Mapping Algorithms into Fixed Size Systolic Arrays, *IEEE Trans. on Computers, C-35, 1,* (January 1986), 1-12.

[16] 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.

[17] D. M. Nicol and P. F. Reynolds, Jr., The Automated Partitioning of Simulations for Parallel Execution, *University of Virginia Department of Computer Science Tech Report TR-85-15,* August 1985.

[18] D. M. Nicol and P. F. Reynolds, Jr., An Optimal Repartitioning Decision Policy, *ICASE Report No. 86-7,* Feb., 1986.

[19] D. M. Nicol and P. F. Reynolds, Jr., Dynamic Remapping Decisions in Multi-Phase Parallel Computations, *ICASE Report No. 86-48,* Sept., 1986.

[20] D. M. Nicol, J. H. Saltz, Dynamic Remapping of Parallel Computations With Varying Resource Demands, *ICASE Report No. 86-45, July 1986.*

[21] C.C. Price, U.W. Pooch, Search Techniques for a Nonlinear Multiprocessor Scheduling Problem, *Naval Research Logistics Quarterly, 29, 2,* June 1982, 213-233.

[22] A. Rapoport, W. E. Stein, and G. J. Burkheimer, *Response Models for Detection of Change,* D. Reidel Publishing Company, Boston, 1979.

[23] S. Ross, *Applied Probability Models with Optimization Applications,* Holden-Day, San Francisco, 1970.

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

[25] S. A. Schmitt, *An Elementary Introduction to Bayesian Statistics,* Addison-Wesley, 1969.

[26] 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.

[27] 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.*

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

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

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

Standard Bibliographic Page

1. Report No. NASA CR-178344 ICASE Report No. 87-49	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle OPTIMAL DYNAMIC REMAPPING OF PARALLEL COMPUTATIONS		5. Report Date July 1987
		6. Performing Organization Code
7. Author(s) David M. Nicol and Paul F. Reynolds, Jr.		8. Performing Organization Report No. 87-49
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. 505-90-21-01
		11. Contract or Grant No. 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

15. Supplementary Notes

Langley Technical Monitor:
Richard W. Barnwell

Submitted to the IEEE Trans. Comput.

Final Report

16. Abstract

A large class of computations are characterized by a sequence of phases, with phase changes occurring unpredictably. We consider the decision problem regarding the remapping of workload to processors in a parallel computation when (i) the utility of remapping and the future behavior of the workload is uncertain, and (ii) phases exhibit stable execution requirements during a given phase, but requirements may change radically between phases. For these problems a workload assignment generated for one phase may hinder performance during the next phase. This problem is treated formally for a probabilistic model of computation with at most two phases. We address the fundamental problem of balancing the expected remapping performance gain against the delay cost.

Stochastic dynamic programming is used to show that the remapping decision policy minimizing the expected running time of the computation has an extremely simple structure: the optimal decision at any decision step is followed by comparing the probability of remapping gain against a threshold. However, threshold calculation requires a priori estimation of the performance gain achieved by remapping. Because this gain may not be predictable, we examine the performance of a heuristic policy that does not require estimation of the gain. In most cases we find nearly optimal performance if remapping is chosen simply when the probability of improving performance by remapping is high. The heuristic's feasibility is demonstrated by its use on an adaptive fluid dynamics code on a multiprocessor. Our results suggest that except in extreme cases, the remapping decision problem is essentially that of dynamically determining whether gain can be achieved by remapping after a phase change; precise quantification of the decision model parameters is not necessary. Our results also suggest that this heuristic is applicable to computations with more than two phases.

17. Key Words (Suggested by Authors(s)) load balancing, parallel computations, multiprocessors, Markov decision processors		18. Distribution Statement 61 - Computer Programming and Software 66 - Systems Analysis Unclassified - unlimited	
19. Security Classif.(of this report) Unclassified	20. Security Classif.(of this page) Unclassified	21. No. of Pages 38	22. Price A03

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