On the Benefits and Limitations of Dynamic Partitioning in Parallel Computer Systems

Mark S. Squillante

IBM T. J. Watson Research Center, Yorktown Heights NY 10598, USA

Abstract. In this paper we analyze the benefits and limitations of dynamic partitioning across a wide range of parallel system environments. We formulate a general model of dynamic partitioning that can be fitted to measurement data to obtain a sufficiently accurate quantitative analysis of real parallel systems executing real scientific and/or commercial workloads. An exact solution of the model is obtained by employing matrix-geometric techniques. We then use this framework to explore the parallel system design space over which dynamic partitioning outperforms other space-sharing policies for a diverse set of application workloads, quantifying the significant performance improvements within these regions. Our results show that these regions and the performance benefits of dynamic partitioning are heavily dependent upon its associated costs, the system load, and the workload characteristics. We also identify the regions of the design space over which dynamic partitioning performs poorly, quantifying the performance degradation and illustrating forms of unstable thrashing.

1 Introduction

The scheduling of processors among parallel jobs submitted for execution is a fundamental aspect of multiprocessor computer systems. A number of scheduling strategies have been proposed for such parallel environments, each differing in the way processors are shared among the jobs. One important class of policies shares the processors by rotating them among a set of jobs in time, and thus are referred to as *time-sharing* strategies. Another particularly important class of scheduling policies is based on *space sharing* where the processors are partitioned among different parallel jobs.

Several approaches have been considered in each of these scheduling classes. Within the space-sharing class, the *static partitioning* of the processors into a fixed number of disjoint sets, each of which are allocated to individual jobs, is a scheduling strategy that has often been employed in a number of commercial systems. This is due in part to its low system overhead and its simplicity from both the system and application viewpoints. The static scheduling approach, however, can lead to relatively low system throughputs and resource utilizations under nonuniform workloads [34, 21, 22, 25, 35], as is common in scientific/engineering computing environments [6]. *Adaptive partitioning* policies, where the number of processors allocated to a job is determined when jobs arrive and depart based on the current system state, have also been considered in a number of research

studies [14, 42, 8, 21, 22, 33, 3, 25]. This approach tends to outperform its static counterparts by adapting partition sizes to the current load. However, the performance benefits of adaptive partitioning can be limited due to its inability to adjust scheduling decisions in response to subsequent workload changes. These potential problems are alleviated under *dynamic partitioning*, where the size of the partition allocated to a job can be modified during its execution, at the expense of increased overhead [40, 4, 14, 42, 9, 16, 21, 22, 35].

The runtime costs of a dynamic partitioning policy are heavily dependent upon the parallel architecture and application workload under consideration. In uniform-access, shared-memory (UMA) systems, these overheads tend to be relatively small and thus the benefits of dynamic partitioning outweigh its associated costs. Several research studies have made this quite clear, showing that dynamic partitioning outperforms all other space-sharing strategies in many UMA environments [40, 14, 42, 9, 16]. In more distributed parallel environments (e.g., non-uniform-access, shared-memory and distributed-memory systems), however, the overheads of a dynamic partitioning policy can be significant due to factors such as data/job migration, processor preemption/coordination and, in some cases, reconfiguration of the application [4, 21, 22, 30]. Even with continuing reductions in the latency of interprocessor communication [41, 39], there are other factors that can cause the cost of repartitioning to be significant for important classes of scientific/engineering applications (e.g., the need to reconfigure the application) [19, 20, 18].

Our objective in this paper is to evaluate the benefits and limitations of dynamic partitioning with respect to other space-sharing strategies across a wide range of parallel system environments, as reflected by the overhead associated with repartitioning and by the efficiency with which the workload utilizes processor allocations. We formulate a general model of dynamic partitioning in parallel computer systems that can be fitted to measurement data to obtain a sufficiently accurate quantitative analysis of real parallel systems executing real scientific and/or commercial workloads. An exact solution of the model is obtained by employing matrix-geometric techniques [24]. In this paper we provide a less formal and rigorous description of our mathematical analysis, and we refer the interested reader to [37, 38] for additional technical details. It is important to note that the computational efficiency of our approach allows us to examine the large design space of diverse parallel environments.

We use this modeling framework to consider the fundamental question: how expensive must the costs of reconfiguration be before it is not beneficial to employ a dynamic partitioning policy? As previously noted, dynamic partitioning has been often shown to outperform other types of space sharing when these overheads are relatively small, such as in UMA environments. In this study we attempt to identify the conditions under which it becomes detrimental to employ dynamic space sharing with respect to other space-sharing policies, the efficiency of the workload, and the costs of repartitioning. Our results provide key insights about these conditions across a diverse set of workloads, showing that the benefits of dynamic partitioning depend heavily upon the application workload as

well as the reconfiguration overhead. We also show that dynamic partitioning provides significant improvements in performance over other forms of space sharing under most workloads when the costs of repartitioning are relatively small, and our results quantify these considerable performance gains. For sufficiently large reconfiguration overheads, however, the costs associated with dynamic partitioning tend to outweigh its benefits, particularly at moderate to heavy system loads, and the degradation in system performance can be quite significant. Our analysis also demonstrates the potential for unstable behavior under dynamic partitioning in these cases, where the system spends a considerable amount of time repartitioning the processors among jobs.

The remainder of the paper is organized as follows. In Section 2 we formulate our model of dynamic space sharing in parallel systems. Section 3 summarizes an exact mathematical analysis of the model, and in Section 4 we provide some of the results of our quantitative analysis. Our concluding remarks are presented in Section 5.

2 Dynamic Partitioning Model

We consider a system consisting of P identical processors that are scheduled according to a dynamic partitioning policy as follows. Let M denote the minimum number of processors allocated to any job, and therefore the maximum number of processor partitions is given by $N = P/M$. If an arrival occurs when $i-1$ jobs are being executed, $1 \leq i \leq N$, then the processors are repartitioned among the i jobs such that each job is allocated (on average) P/i processors. An arrival that finds $i \geq N$ jobs in the system is placed in a first-come first-served (FCFS) *system queue* to wait until a processor partition becomes available. When one of the $i+1$ jobs in execution departs, $1 \leq i < N$, the system reconfigures the processor allocations so that each job receives (on average) P/i processors. A departure when $i > N$ simply causes the job at the head of the system queue to be allocated the available partition, and no repartitioning is performed. The exact details of the processor allocation decisions made by the scheduler in each case, as well as the overheads of making these decisions and of reconfiguring the applications involved, are reflected in the parameter distributions and the state space of the corresponding stochastic process (see Section 3).

Jobs arrive to the system when it contains i jobs according to a phase-type probability distribution $\mathcal{A}_i(\cdot)$ with mean rate λ_i, $i \geq 0$, $\mathcal{A}_{N+k}(\cdot) \equiv \mathcal{A}_N(\cdot)$, $k \geq 0$. When the system is executing i jobs, the service times of each of these jobs are assumed to be independent and identically distributed according to a phase-type distribution $\mathcal{B}_i(\cdot)$ with mean service time $\widetilde{S}_i$, $1 \leq i \leq N$. The times required to repartition the processors among the i jobs being executed (either due to a departure when the system contains $i+1$ jobs or an arrival when the system contains $i-1$ jobs) are assumed to be independent and identically distributed following a phase-type distribution $\mathcal{C}_i(\cdot)$ with mean reconfiguration overhead $\widetilde{R}_i$, $1 \leq i \leq N$. Multiple job arrivals, multiple job departures, and both an arrival and a departure within a small time interval are all assumed to occur with

negligible probability, leading to a quasi-birth-death process [24] (although our analysis is easily extended to handle batch arrivals and/or departures as long as the batch sizes are bounded; see [38]).

The use of phase-type distributions [24] for the parameters of our model is motivated in part by their important mathematical properties, which can be exploited to obtain a tractable analytic model while capturing the fundamental aspects of dynamic partitioning. Just as important, however, is the fact that any real distribution can in principle be represented arbitrarily close by a phase-type distribution. Furthermore, a considerable body of research has examined the fitting of phase-type distributions to empirical data, and a number of algorithms have been developed for doing so [1, 5, 12, 13]. It is also well known that some steady-state measures (e.g., mean waiting time) often depend only upon the first few moments of the parameter distributions (as opposed to their detailed forms) in an important and general class of probability models [26, 27, 28]. We therefore have a general formulation that can be used to provide a sufficiently realistic model and analysis of dynamic partitioning in parallel systems.

3 Mathematical Analysis

The dynamic partitioning model presented in the previous section is represented by a continuous-time Markov chain defined over an infinite, multi-dimensional state space. This Markov chain has a particular structure that we exploit, using matrix-geometric techniques [24], to obtain an exact model solution in an extremely efficient manner. In this section we provide a less formal and rigorous mathematical analysis of the model, and we refer the interested reader to [37, 38] for additional technical details. A closed-form solution for the specific case where the model parameters all have exponential distributions, and an analysis of optimal static partitioning under assumptions corresponding to those in Section 2 are also provided in [37].

The states of the Markov chain are denoted by $(i, \overline{v}_{i,z})$ where the value of i, $i \geq 0$, reflects the total number of parallel jobs in the system, and the value of the vector $\overline{v}_{i,z}$, $1 \leq z \leq D_i$, reflects the states of the phase variables for the model distributions ($\mathcal{A}_i$, $\mathcal{B}_i$, $\mathcal{C}_i$) as well as any other aspects of the system recorded in the state space. The (infinitesimal) rates at which the system moves from one state to another state are defined by the elements of the transition rate matrix for the Markov chain, denoted by $\mathbf{Q}$. We refer to the set of states $\{(i, \overline{v}_{i,1}), \ldots, (i, \overline{v}_{i,D_i})\}$ as *level* i, and D_i denotes the number of states on level i.

The states of the chain are ordered lexicographically, i.e., $(0, \overline{v}_{0,1})$, ..., $(0, \overline{v}_{0,D_0})$, $(1, \overline{v}_{1,1})$, ..., $(1, \overline{v}_{1,D_1})$, $(2, \overline{v}_{2,1})$, $(2, \overline{v}_{2,2})$, ... Using this ordering, we define

$$\boldsymbol{\pi}_i \equiv (\pi(i, \overline{v}_{i,1}),\ \pi(i, \overline{v}_{i,2}),\ \ldots,\ \pi(i, \overline{v}_{i,D_i})), \quad i \geq 0, \tag{1}$$

and

$$\boldsymbol{\pi} \equiv (\boldsymbol{\pi}_0,\ \boldsymbol{\pi}_1,\ \boldsymbol{\pi}_2,\ \ldots). \tag{2}$$

We also define $D \equiv \sum_{i=0}^{N-1} D_i$. The vector $\boldsymbol{\pi}$ is the steady-state probability vector for the Markov chain, and the value of each of its components $\pi(i, \overline{v}_{i,z})$, $i \geq 0$, $1 \leq z \leq D_i$, represents the proportion of time the system spends in state $(i, \overline{v}_{i,z})$ over the long run operation of the system. It is well known that the steady-state probability vector $\boldsymbol{\pi}$ can be obtained by solving the global balance equations

$$\boldsymbol{\pi}\mathbf{Q} \;=\; \mathbf{0}, \tag{3}$$

together with the constraint that the sum of these components must be 1 [10].

We arrange the transition rate matrix $\mathbf{Q}$ of the Markov chain in the same order as the elements of the steady-state probability vector $\boldsymbol{\pi}$, and we block-partition the matrix according to the state space levels. The $\mathbf{Q}$ matrix then has a structure given by

$$\mathbf{Q} \;=\; \begin{bmatrix} B_{00} & B_{01} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots \\ B_{10} & B_{11} & A_0 & \mathbf{0} & \mathbf{0} & \dots \\ \mathbf{0} & A_2 & A_1 & A_0 & \mathbf{0} & \dots \\ \mathbf{0} & \mathbf{0} & A_2 & A_1 & A_0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}, \tag{4}$$

where B_{00}, B_{01}, B_{10}, B_{11} and A_k, $0 \leq k \leq 2$, are finite matrices of dimensions $D \times D$, $D \times D_N$, $D_N \times D$, $D_N \times D_N$ and $D_N \times D_N$, respectively. The key to the matrix-geometric solution method is the repetitive structure beyond a certain point in the matrix $\mathbf{Q}$, which in our case occurs beyond level N.

The block of matrices corresponding to levels 0 through N of the state space, i.e.,

$$\begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix},$$

has the form

$$\begin{bmatrix} \Psi(0) & \Lambda(0) & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \Phi(1) & \Psi(1) & \Lambda(1) & \mathbf{0} & \cdots & & \\ \mathbf{0} & \Phi(2) & \Psi(2) & \Lambda(2) & \cdots & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & & \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \Phi(N) & \Psi(N) \end{bmatrix},$$

where $\Phi(i)$, $\Psi(i)$ and $\Lambda(i)$ have dimensions $D_i \times D_{i-1}$, $D_i \times D_i$ and $D_i \times D_{i+1}$, respectively. Intuitively, the matrix $\Phi(i)$ defines the transitions from states in level i to states in level $i-1$, $1 \leq i \leq N$, $\Psi(i)$ describes the transitions between states within level i, $0 \leq i \leq N$, and $\Lambda(i)$ defines the transitions from states in level i to states in level $i+1$, $0 \leq i \leq N-1$. These matrices (which are dependent upon the number of jobs in the system, as recorded by i) define the exact allocation behavior of the dynamic partitioning policy being modeled, the arrival, service and reconfiguration processes of the workload being modeled, and the various interactions of each of these aspects of the system. The A matrices provide the same functionality for the repeating (homogeneous) portion of the state space, where A_2 (resp., A_0) describes the transitions from states in level i

to states in level $i-1$ (resp., $i+1$) and A_1 defines the transitions between states within level i, $i \geq N+1$.

Given the form in equation (4) for the transition rate matrix of the Markov chain, the solution of the global balance equations in (3) and the normalization constraint can be obtained exactly via matrix-geometric techniques [24]. In particular, the geometric portion of the probability vector, representing when the system has more than N parallel jobs, can be solved as

$$\boldsymbol{\pi}_{N+k} \;=\; \boldsymbol{\pi}_N R^k, \quad k \geq 0, \tag{5}$$

where R is the minimal non-negative matrix that satisfies

$$R^2 A_2 \;+\; R A_1 \;+\; A_0 \;=\; \mathbf{0}. \tag{6}$$

The remaining components of the vector $\boldsymbol{\pi}$ can be found by solving the balance equations for levels 0 through N, which can be written in matrix notation as

$$(\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_N) \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} + RA_2 \end{bmatrix} \;=\; \mathbf{0}, \tag{7}$$

together with the normalization constraint

$$(\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_{N-1})\mathbf{e} \;+\; \boldsymbol{\pi}_N (I-R)^{-1}\mathbf{e} \;=\; 1, \tag{8}$$

where we have made use of equation (5) and $\mathbf{e}$ is the column vector of all ones.

The performance measures of interest can be directly obtained from the steady-state probability vector $\boldsymbol{\pi}$. In particular, the mean number of jobs in the system, the mean job response time, and the percentage of time spent repartitioning processor allocations in steady state are calculated as

$$\overline{N}_{\text{DP}} \;=\; \sum_{k=1}^{N-1} k\,\boldsymbol{\pi}_k \mathbf{e} \;+\; N\,\boldsymbol{\pi}_N (I-R)^{-1}\mathbf{e} \;+\; \boldsymbol{\pi}_N (I-R)^{-2} R\mathbf{e}, \tag{9}$$

$$\overline{T}_{\text{DP}} \;=\; \frac{\sum_{k=1}^{N-1} k\,\boldsymbol{\pi}_k \mathbf{e} \;+\; N\,\boldsymbol{\pi}_N (I-R)^{-1}\mathbf{e} \;+\; \boldsymbol{\pi}_N (I-R)^{-2} R\mathbf{e}}{\lambda}, \tag{10}$$

and

$$p_r \;=\; (\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_{N-1})\nu_b \;+\; \sum_{k=0}^{\infty} \boldsymbol{\pi}_{N+k} \nu_r, \tag{11}$$

respectively, where the binary vectors ν_b and ν_r are used to exclude the state probabilities of states corresponding to when the system is not reconfiguring its processor partitions. The solution of the matrix R, the steady-state probability vector $\boldsymbol{\pi}$, and equations (9) – (11) are all efficiently computed by the routines provided by the MAGUS performance modeling tool [23, 36].

4 Results

Our dynamic partitioning model can be fitted to measurement data to obtain a sufficiently accurate quantitative analysis of dynamic partitioning in real parallel systems executing real scientific and/or commercial workloads. We hope that our model and (exact) analysis, together with such system and workload measurement data, will serve as a basis for further research of dynamic partitioning across different parallel environments.

In the absence of such measurement data, we consider here the performance characteristics of dynamic partitioning under assumptions based on data and results that have appeared in the research literature. Our objective is to quantitatively evaluate the benefits and limitations of dynamic partitioning with respect to other space-sharing strategies as a function of its associated costs and the workload efficiency, and to therefore determine how expensive reconfiguration overheads must be before it is not beneficial to employ a dynamic partitioning policy.

We first provide some technical preliminaries that support the analysis of this section, including our assumptions based upon previous research. Our results, a portion of which are subsequently presented, were obtained with the MAGUS performance modeling tool [23, 36]. We assume throughout that $M = 1$, and thus $N = P$.

4.1 Preliminaries

The execution time of many parallel applications on a fixed number of processors for a given problem size is either constant or bounded between relatively tight upper and lower bounds. It is therefore most appropriate to model the execution time of such an application by a probability distribution with a coefficient of variation[1] close (or equal) to 0. On the other hand, current and expected *workloads* for large-scale parallel computing environments consist of a mixture of such jobs with very different resource requirements, often resulting in a highly variable workload [29, 21, 22, 6]. We thus use model parameter distributions that reflect this variability in the resource requirements of the system workload. Specifically, we consider the service time distributions $\mathcal{B}_i$ to be exponential with mean service times $\widetilde{S}_i$ and we consider the reconfiguration overhead distributions $\mathcal{C}_i$ to be exponential with mean reconfiguration overheads $\widetilde{R}_i$, $1 \leq i \leq N$, which are dependent upon the number of jobs i in the system. This service time assumption matches various instances of a workload based on measurement data of computational fluid dynamics applications [21, 22].[2]

[1] The coefficient of variation is the ratio of the standard deviation to the mean [10]. A deterministic distribution has a coefficient of variation equal to 0.

[2] There exists evidence suggesting that the coefficient of variation for the workload, in many cases, is larger than 1 [21, 22, 6]. We are currently working on results for the case of hyperexponential service time and reconfiguration overhead distributions to address such workloads, noting that the hyperexponential distribution is a very simple instance of a phase-type distribution.

Another important aspect of the parallel jobs comprising the system workload is the efficiency with which these jobs utilize the processors allocated to them. The efficiency of the workload as a whole can be reflected in the service rates $\mu_i \equiv 1/\widetilde{S}_i$ of the model, $1 \leq i \leq N$, where the model parameter μ_i represents the rate at which the system services a workload of i parallel jobs "each" executed on P/i processors.[3] These workload service rates can be represented by

$$\mu_i \;=\; i\,\frac{1}{\tau(P/i)} \;=\; i\,\frac{\mathbf{S}(P/i)}{\tau(1)}, \quad 1 \leq i \leq N, \tag{12}$$

where $\mathbf{S}(\cdot)$ is the workload speedup function and $\tau(P/i)$ is the mean service time of a generic job when the system is servicing a workload of i jobs. Letting $1/\mu \equiv \widetilde{S} \equiv \tau(1)$, we have

$$\mu_i \;=\; i\,\mathbf{S}(P/i)\,\mu, \quad 1 \leq i \leq N. \tag{13}$$

To isolate the key reconfiguration overhead parameter of our analysis, we consider $\widetilde{R}_i = \widetilde{R}$, $1 \leq i \leq N$. Throughout this section we let $\widetilde{S} \equiv \tau(1) = 1000$.

The workload speedup function can be written as [7, 32]

$$\mathbf{S}(n) \;=\; \frac{n}{1 + f_\circ(n)}, \quad n \geq 1, \tag{14}$$

where $f_\circ(\cdot)$ is used here to reflect the various types of overhead that can reduce the workload speedup function from being linear. For a large class of parallel applications, the factor $f_\circ(\cdot)$ is dominated by issues related to communication [7]. It therefore can be approximated within the context of our model by

$$f_\circ(n) \;=\; F\,n^{1/d}, \quad n \geq 1, \tag{15}$$

where F is a constant that depends upon the system architecture and the application workload, and d is the system dimension. In the results that follow, we consider the values $F \in \{0.025, 0.175, 1.225\}$ and $d \in \{2, 3\}$ which cover a range of parameters provided in [7]. We also consider the overheads $f_\circ(n) \in \{0, n-1\}$, which represent the extremes of linear and constant workload speedup functions. Our sole purpose in using this workload formulation is to consider a reasonable range of parallel processing overheads, thus allowing us to examine the effects of different types of parallel workloads in our analysis of dynamic partitioning.

The times at which jobs arrive to the system are defined by the distributions $\mathcal{A}_i$, $0 \leq i \leq N$, which are dependent upon the number of jobs i in the system. These arrival times are most often modeled by a Poisson distribution in the research literature [4, 42, 21, 22, 33, 17, 32, 25]. We thus assume that jobs come

[3] The details of exactly how the dynamic partitioning policy allocates processors to jobs when i does not evenly divide P, as well as the service rates for each of these cases, are easily incorporated in our model (see Sections 2 and 3, and [37, 38]). Here we make the simplifying assumption that the workload speedup function reasonably approximates this information.

to the system according to a Poisson distribution with mean rate $\lambda_i = \lambda$, $0 \leq i \leq N$.[4]

The mean job response time under dynamic partitioning ($\overline{T}_{\text{DP}}$) is obtained via the analysis of Section 3. To support our comparison in the next section, we obtain performance measures for other space-sharing policies as follows. Consider a system in which the processors are statically divided into K partitions each of size P/K, where only values of K that evenly divide P are examined. We refer to this system as SP(K). Under the above model parameter assumptions (see [37] for a more general analysis), this system is equivalent to an M/M/K queue with arrival rate λ and service rate $\mathbf{S}(P/K)\mu$. Hence, the mean job response time in the SP(K) system, denoted by $\overline{T}_{\text{SP(K)}}$, is obtained from the well-known solution of the M/M/K queueing system [10]. The mean response time under the optimal static partitioning policy, for a given arrival rate, is therefore given by

$$\overline{T}_{\text{Opt-SP}}(\lambda) \;=\; \min_{1 \leq K \leq P} \{\, \overline{T}_{\text{SP(K)}}(\lambda) \,\}. \tag{16}$$

Our decision to consider equal-sized processor partitions is motivated by the results of recent studies [25, 15] showing that adaptive/static strategies in which the system is divided into equal-sized partitions outperform other adaptive/static policies when job service time requirements are not used in scheduling decisions. Several recent research studies, under different workload assumptions, have also shown that adaptive partitioning yields steady-state performance comparable to that of the optimal static partitioning policy for a given value of λ [21, 22, 33]. Hence, when this relation holds, the mean job response time under adaptive partitioning is accurately approximated by equation (16) and the results of the next section are also representative of a comparison between adaptive and dynamic partitioning policies.

Each of the various application workloads considered in our study cause the system to saturate (i.e., the response times become unbounded) at different job arrival rates. The parallel system under a workload with perfect linear speedup is the last to saturate with increasing offered load, as this is the most efficient case considered. We therefore use the measure of system utilization under the linear workload as the basis for all of our performance comparisons. More specifically, we use (offered) system load to refer to the ratio λ/λ^*, where λ^* denotes the saturation point for the linear workload. The results that follow for each system are all plotted as functions of system load over the interval (0, 1).

4.2 Comparison of Space-Sharing Policies

Our first set of results identifies the reconfiguration costs for which dynamic partitioning and optimal static partitioning provide identical steady-state performance, as a function of the offered load. In particular, we use a binary search

[4] There exists evidence suggesting that the interarrival times of jobs, in some cases, is more variable than the exponential assumption considered here [6]. We are currently working on results for the case of hyperexponential interarrival times to address such workloads.

on the reconfiguration overhead and iteratively solve our dynamic partitioning model until we find the value of $\widetilde{R}$ that yields the same mean response time as that obtained from equation (16) for a given system load. We note that the solution of our model is computed in an extremely efficient manner, and thus this fixed-point iteration converges *very* rapidly. To simplify our subsequent discussions, we use $\widetilde{R}^*$ to denote the value of $\widetilde{R}$ obtained from this fixed-point iteration. This value is representative of the repartitioning overhead for which both types of space-sharing yield the same performance. Thus, reconfiguration overheads less than $\widetilde{R}^*$ define the regions over which dynamic partitioning outperforms the optimal static policy, whereas dynamic partitioning provides worse performance when the reconfiguration overhead is greater than $\widetilde{R}^*$. Figure 1 plots these response time contours as a function of system load for the different workloads considered and $P = 16$. The y-axis is plotted on a particular log scale. The corresponding results for $P = 32$, $P = 64$ and $P = 128$ are provided in Figures 2, 3 and 4, respectively.

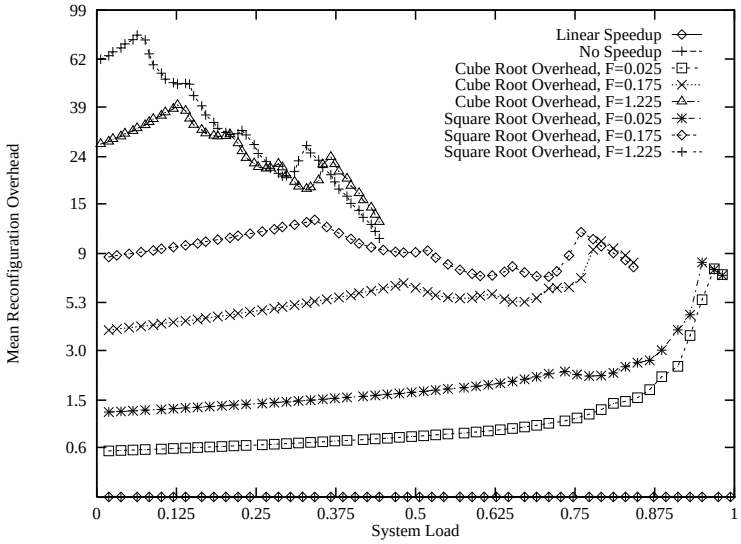

Fig. 1. Response Time Contours (log scale) with respect to Dynamic and Optimal Static Partitioning, for $P = 16$

We first observe that all of the results for the linear workload are equal to zero. This is due to the fact that the SP(1) system is optimal in this case. The optimality of SP(1) for the linear workload follows directly from a result due to Brumelle [2], where it is shown that the mean response time in a $GI/GI/k$ queue with interarrival and service time distribution functions $A(t)$ and $B(t)$, respectively, is greater than or equal to the mean response time in the $GI/GI/1$ queue with the same interarrival time distribution and service time distribution $B(kt)$, provided that the coefficient of variation of $B(\cdot)$ is less than or equal

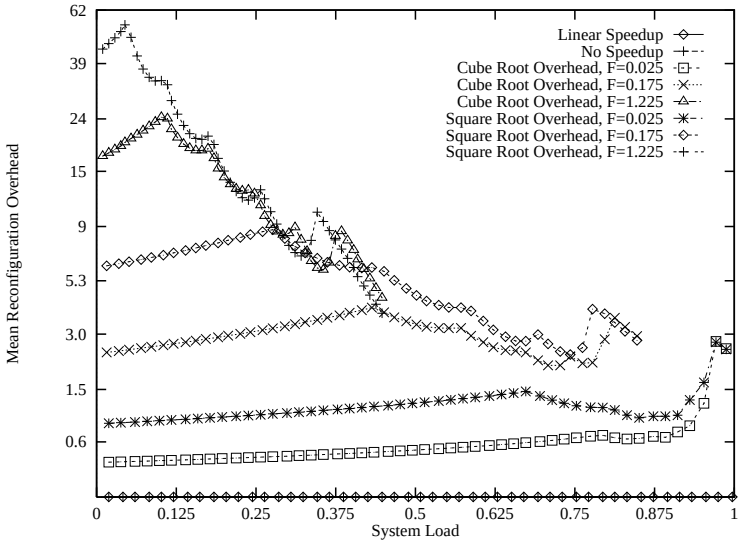

Fig. 2. Response Time Contours (log scale) with respect to Dynamic and Optimal Static Partitioning, for $P = 32$

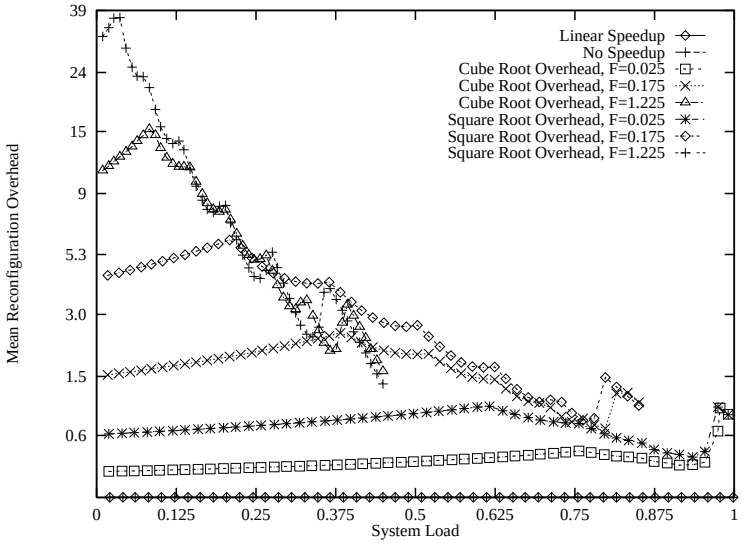

Fig. 3. Response Time Contours (log scale) with respect to Dynamic and Optimal Static Partitioning, for $P = 64$

to 1. Given the optimality of shortest-job-first in uniprocessor systems [11], our results for linear workloads suggest that *time sharing* all of the processors among the jobs (in a sufficiently coarse manner to outweigh the overhead of context switching) may provide the best steady-state performance when the workload

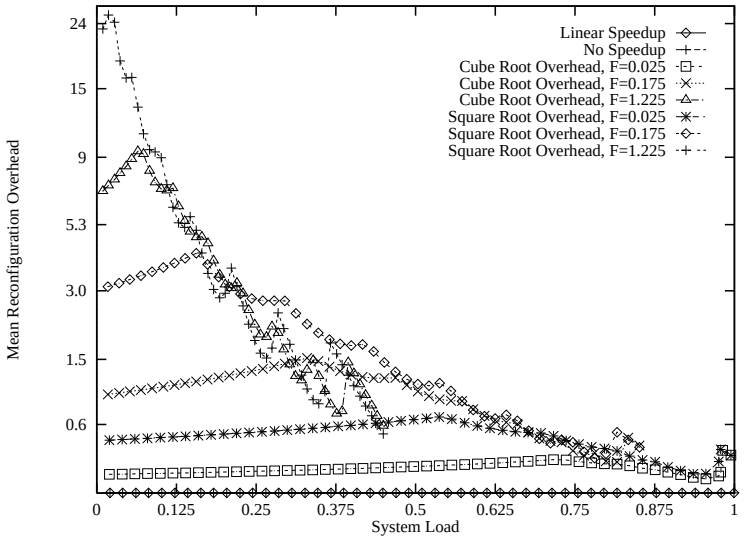

Fig. 4. Response Time Contours (log scale) with respect to Dynamic and Optimal Static Partitioning, for $P = 128$

makes extremely efficient use of the processors. We also note that the benefits of using time sharing together with a static partitioning policy have been observed for a different region of the parallel system design space [31].

We observe that the values of $\widetilde{R}^*$ are also equal to zero for the constant workload. The optimality of the SP(P) system when the workload has a constant speedup function is easily explained by noting that all space-sharing policies besides SP(P) effectively leave processors idle. Thus, we find that dynamic partitioning is a sub-optimal space-sharing strategy at both extremes of the spectrum of parallel workloads.

Turning our attention to the remaining workloads, which are probably more representative of those often found in practice, we observe that the performance of the dynamic partitioning policy is as good as or better than optimal static partitioning even with a relatively large reconfiguration overhead. In particular, the mean cost to repartition the processors can be as large as 78, 55, 37 and 26 on a system with 16, 32, 64 and 128 processors, respectively, and it still may be beneficial to employ a dynamic partitioning policy under light system loads. Note that these overheads are relative to the mean service time of a generic job when executed on a single processor, which has a value of $\tau(1) = 1000$. Note further that larger values of F and smaller values of d imply system workloads with poorer speedup functions (see equations (14) and (15)).

As the system load increases, the general drift for the $\widetilde{R}^*$ values decreases because the large reconfiguration costs that can be tolerated at lighter loads tend to outweigh the benefits of dynamic partitioning at heavier loads. In the limit as the system approaches saturation, the probability that the system repartitions

the processors tends toward 0, i.e., the frequency of reconfigurations decreases to 0 as the Markov chain spends essentially all of its time at or above level N (see [37] for the technical details). It therefore follows that the dynamic partitioning system converges toward SP(P) in the limit as the system approaches saturation.

The scalloped shape of the response time contours for these workloads in Figures 1 – 4 are representative of the response time behavior of the optimal static partitioning policy. Specifically, each of the points where the value of $\widetilde{R}^*$ increases (within a particular region of system load) is due to a change in the value of K employed under the optimal static policy. This in turn causes the response time under dynamic partitioning to be compared with a different static partitioning response time curve, which is further from saturation than the response time curve for the previous measure of load. As previously noted, the different systems (consisting of the various workloads and policies under consideration) saturate at different job arrival rates. This explains why the various response time contours span different intervals of offered load.

Our next set of results quantifies the performance benefits of dynamic partitioning with respect to optimal static partitioning. Comparing the value of $\overline{T}_{\text{Opt-SP}}$ with $\overline{T}_{\text{DP}}$, we obtain the percentage of improvement, or degradation, in mean response time under dynamic partitioning as a function of the system load and the reconfiguration overhead. The results for $P = 64$ and $\widetilde{R} = 0.01, 0.1, 1, 5, 10, 20$ are plotted in Figures 5, 6, 7, 8, 9 and 10, respectively. We first observe that dynamic partitioning provides no performance benefits under workloads with linear or constant speedup functions. In fact, the mean job response time under dynamic partitioning can be significantly worse than that of optimal static partitioning, particularly for heavy traffic intensities and large values of $\widetilde{R}$. This follows directly from our above discussions for the linear and constant workloads.

With respect to the other workloads considered, we observe that dynamic partitioning can provide significant improvements in performance for relatively small reconfiguration overheads. By adjusting scheduling decisions in response to workload changes, the dynamic partitioning policy provides the most efficient utilization of the processors among the various space-sharing strategies when $\widetilde{R}$ is small. These performance benefits tend to increase as the system load rises, since workload changes are more frequent and dynamic partitioning adjusts its processor allocations accordingly to achieve the best steady-state performance. Our results for small reconfiguration costs confirm and help to further explain those previously reported for dynamic partitioning policies in UMA environments.

When the value of $\widetilde{R}$ becomes sufficiently large, however, the overhead of repartitioning the processors tends to outweigh the benefits of dynamic partitioning, particularly at moderate to heavy system loads. Our results clearly show the significant degradation in system performance (with respect to optimal static partitioning) that is possible under large reconfiguration overheads. This is due in part to an unstable characteristic of the dynamic partitioning pol-

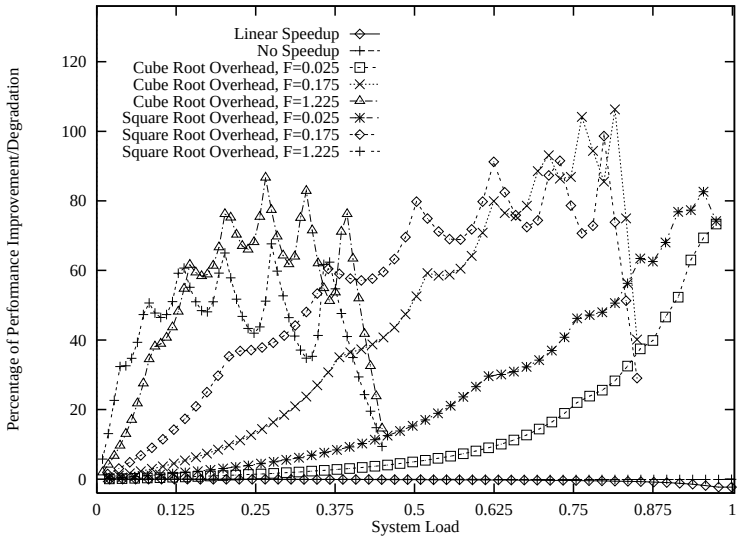

Fig. 5. Response Time Improvements, or Degradations, under Dynamic Partitioning, for $P = 64$ and $\widetilde{R} = 0.01$

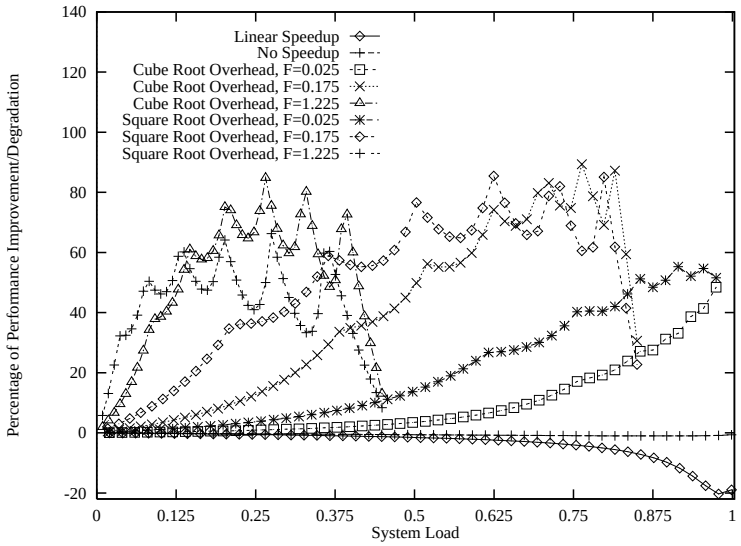

Fig. 6. Response Time Improvements, or Degradations, under Dynamic Partitioning, for $P = 64$ and $\widetilde{R} = 0.1$

icy where the system spends a considerable amount of time repartitioning the processors among jobs. To illustrate this, we plot in Figure 11 the percentage of time that the system spends reconfiguring its processor allocations in steady state as a function of the system load for $P = 64$ and $\widetilde{R} = 20$. The potential for

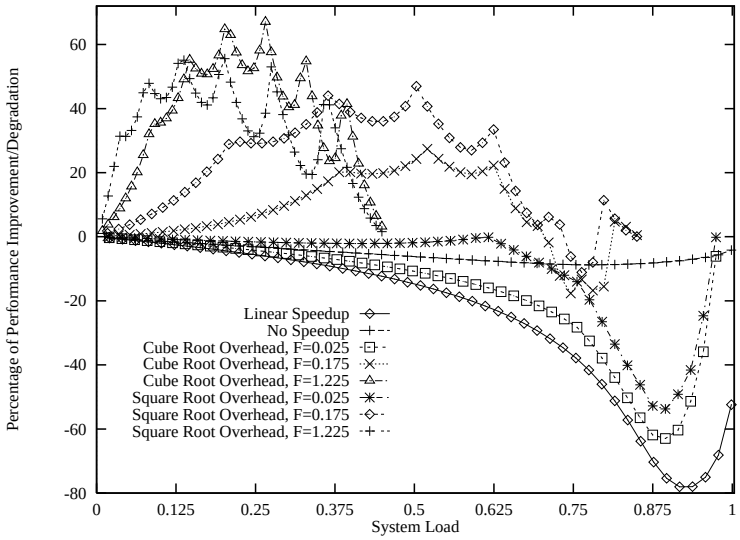

Fig. 7. Response Time Improvements, or Degradations, under Dynamic Partitioning, for $P = 64$ and $\widetilde{R} = 1$

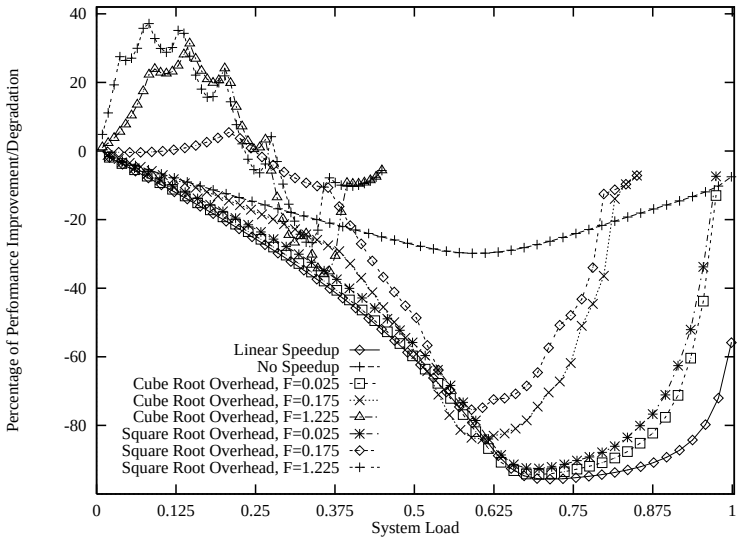

Fig. 8. Response Time Improvements, or Degradations, under Dynamic Partitioning, for $P = 64$ and $\widetilde{R} = 5$

instability under dynamic partitioning is exhibited by the two different phases of the percentage curves, where we observe a sharp increase in the system's reconfiguration of processors toward the end of the first phase, while this factor continually decreases (often linearly) through the second phase. In fact, for all

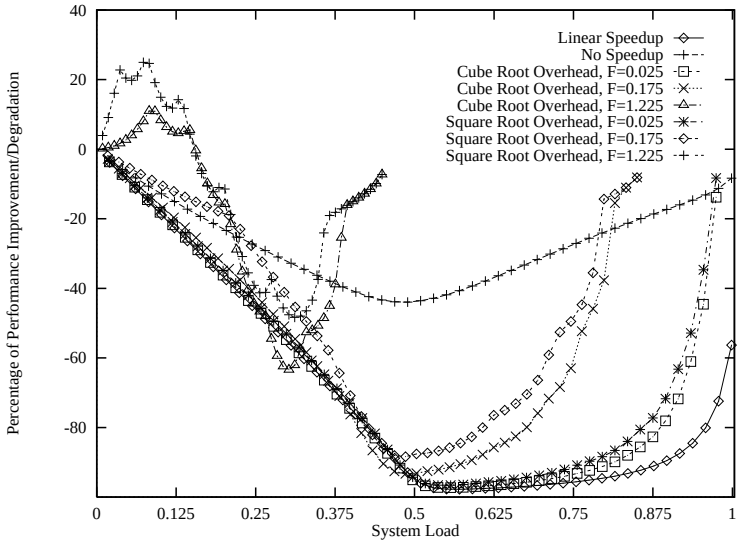

Fig. 9. Response Time Improvements, or Degradations, under Dynamic Partitioning, for $P = 64$ and $\widetilde{R} = 10$

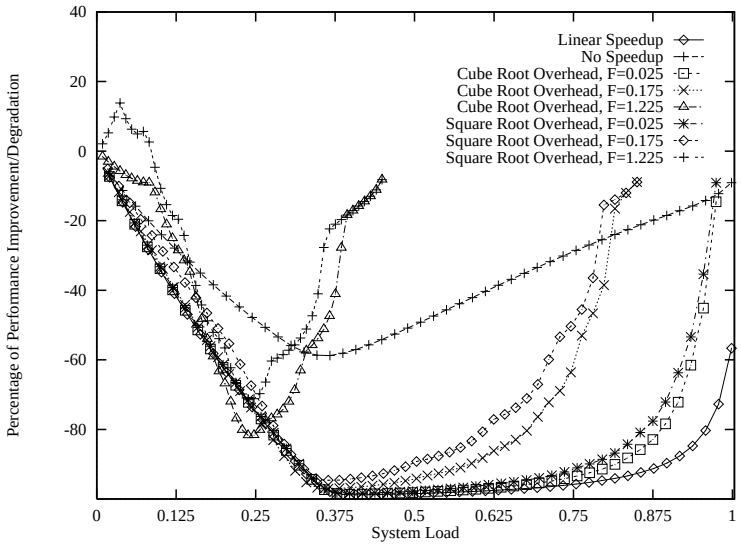

Fig. 10. Response Time Improvements, or Degradations, under Dynamic Partitioning, for $P = 64$ and $\widetilde{R} = 20$

workloads except those with $F = 1.225$, there are intervals of offered load over which the system spends the majority of its time (more than 60%) repartitioning the processors among jobs. This form of *reconfiguration thrashing* clearly must be avoided.

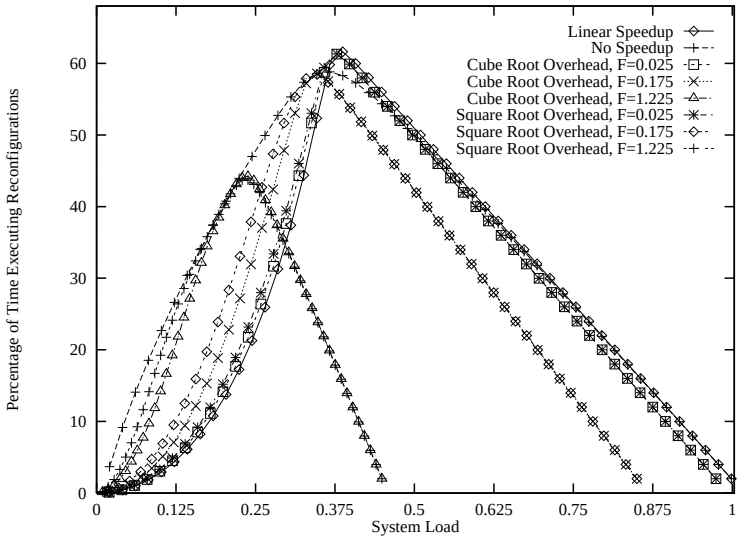

Fig. 11. Percentage of Time Spent Repartitioning the Processors under the Dynamic Policy in Steady State, for $P = 64$ and $\widetilde{R} = 20$

As noted above, the system under dynamic partitioning converges toward $\mathrm{SP}(P)$ in the limit as it approaches saturation, i.e., the percentage of time spent repartitioning the processors tends toward 0 in this limit (see Figure 11). Similarly, the scalloped shape of the response time improvement percentage curves are caused by the exact behavior cited above for the response time contours.

5 Conclusions

In this paper we examined the benefits and limitations of dynamic partitioning with respect to other space-sharing strategies across a wide range of parallel system environments. We formulated a general model of dynamic partitioning that can be fitted to measurement data to obtain a sufficiently accurate quantitative analysis of real parallel systems executing real scientific and/or commercial workloads. An exact solution of the model was then obtained by employing matrix-geometric techniques, the computational efficiency of which allowed us to explore the large parallel system design space. We hope that the model and analysis presented in this paper, together with real measurement data on parallel system and workload characteristics, will serve as a basis for further research of dynamic partitioning across different system architectures and application workloads.

Our results show that the performance benefits of dynamic partitioning are heavily dependent upon its associated costs, the system load and the workload characteristics. When the reconfiguration overhead is relatively small, the performance benefits of dynamic partitioning can be quite significant for most of the

workloads considered. In these cases, the dynamic partitioning policy provides the most efficient utilization of the processors among the various space-sharing strategies by adjusting scheduling decisions in response to workload changes. These performance benefits tend to increase with rising traffic intensities, since workload changes are more frequent and dynamic partitioning adjusts its processor allocations accordingly to achieve the best steady-state, space-sharing performance.

When the reconfiguration costs are sufficiently large, however, this overhead tends to outweigh the benefits of dynamic partitioning, particularly at moderate to heavy system loads, and the degradation in system performance (with respect to the other forms of space sharing) can be quite significant. This is caused in part by a form of reconfiguration thrashing where the system spends a considerable amount of time repartitioning the processors among jobs. In such cases, dynamic partitioning must be employed more selectively. An interruptible list, containing those jobs in execution that are eligible for reconfiguration, can be used to prevent thrashing by removing a job from the list (making it ineligible for repartitioning) for some period of time after it has been reconfigured [21, 22]. Since the costs of reconfiguration often depend upon the problem size [21, 22, 30], having the user provide such information can facilitate even better repartitioning decisions by the scheduling policy. Finally, we believe it will be beneficial to combine dynamic partitioning together with some form of time sharing (in a sufficiently coarse manner to outweigh the costs of context switching) when the reconfiguration overhead is sufficiently large.

Acknowledgements. We thank the anonymous reviewers for several helpful comments that improved the presentation.

References

1. S. Asmussen, O. Nerman, and M. Olsson. Fitting phase type distributions via the EM algorithm. Tech. Rep. 1994:23, Dept. Math., Chalmers Univ. Tech., 1994.
2. S. L. Brumelle. Some inequalities for parallel-server queues. *Op. Res.*, **19**:402–413, 1971.
3. S.-H. Chiang, R. K. Mansharamani, and M. K. Vernon. Use of application characteristics and limited preemption for run-to-completion parallel processor scheduling policies. In *Proc. ACM SIGMETRICS Conf.*, 33–44, 1994.
4. K. Dussa, B. Carlson, L. Dowdy, and K.-H. Park. Dynamic partitioning in transputer environments. In *Proc. ACM SIGMETRICS Conf.*, 203–213, 1990.
5. M. J. Faddy. Fitting structured phase-type distributions. Tech. Rep., Dept. Math., Univ. Queensland, Australia, 1994. To appear, *Appl. Stoch. Mod. Data Anal.*.
6. D. G. Feitelson and B. Nitzberg. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), Springer-Verlag, 1995. Lecture Notes in Computer Science Vol. 949.
7. G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker. *Solving Problems on Concurrent Processors Volume I: General Techniques and Regular Problems.* Prentice Hall, 1988.

8. D. Ghosal, G. Serazzi, and S. K. Tripathi. The processor working set and its use in scheduling multiprocessor systems. *IEEE Trans. Soft. Eng.*, **17**:443–453, 1991.
9. A. Gupta, A. Tucker, and S. Urushibara. The impact of operating system scheduling policies and synchronization methods on the performance of parallel applications. In *Proc. ACM SIGMETRICS Conf.*, 1991.
10. L. Kleinrock. *Queueing Systems Volume I: Theory.* John Wiley and Sons, 1975.
11. L. Kleinrock. *Queueing Systems Volume II: Computer Applications.* John Wiley and Sons, 1976.
12. A. Lang. Parameter estimation for phase-type distributions, part I: Fundamentals and existing methods. Tech. Rep. 159, Dept. Stats., Oregon State Univ., 1994.
13. A. Lang and J. L. Arthur. Parameter estimation for phase-type distributions, part II: Computational evaluation. Tech. Rep. 160, Dept. Stats., Oregon State Univ., 1994.
14. S. T. Leutenegger and M. K. Vernon. The performance of multiprogrammed multiprocessor scheduling policies. In *Proc. ACM SIGMETRICS Conf.*, 226–236, 1990.
15. R. K. Mansharamani and M. K. Vernon. Properties of the EQS parallel processor allocation policy. Tech. Rep. 1192, Univ. Wisconsin, Comp. Sci. Dept., 1993.
16. C. McCann, R. Vaswani, and J. Zahorjan. A dynamic processor allocation policy for multiprogrammed shared-memory multiprocessors. *ACM Trans. Comp. Syst.*, **11**(2):146–178, 1993.
17. C. McCann and J. Zahorjan. Processor allocation policies for message-passing parallel computers. In *Proc. ACM SIGMETRICS Conf.*, 19–32, 1994.
18. N. H. Naik, V. K. Naik, and M. Nicoules. Parallelization of a class of implicit finite difference schemes in computational fluid dynamics. *Intl. J. High-Speed Comp.*, **5**, 1993.
19. V. K. Naik. Performance effects of load imbalance in parallel CFD applications. In *Proc. SIAM Conf. Par. Proc.*, 1992.
20. V. K. Naik. Scalability issues for a class of CFD applications. In *Proc. Scal. High Perf. Comp. Conf.*, 268–275, 1992.
21. V. K. Naik, S. K. Setia, and M. S. Squillante. Performance analysis of job scheduling policies in parallel supercomputing environments. In *Proc. Supercomputing '93*, 824–833, 1993.
22. V. K. Naik, S. K. Setia, and M. S. Squillante. Scheduling of large scientific applications on distributed memory multiprocessor systems. In *Proc. SIAM Conf. Par. Proc. Sci. Comp.*, 913–922, 1993.
23. R. D. Nelson and M. S. Squillante. The MAtrix-Geometric qUeueing model Solution package (MAGUS) user manual. Tech. Rep. RC, IBM Res. Div., 1994.
24. M. F. Neuts. *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach.* The Johns Hopkins Univ. Press, 1981.
25. E. Rosti, E. Smirni, L. W. Dowdy, G. Serazzi, and B. M. Carlson. Robust partitioning policies of multiprocessor systems. *Perf. Eval.*, **19**:141–165, 1994.
26. R. Schassberger. Insensitivity of steady-state distributions of generalized semi-Markov processes, part I. *Ann. Prob.*, **5**(1):87–99, 1977.
27. R. Schassberger. Insensitivity of steady-state distributions of generalized semi-Markov processes, part II. *Ann. Prob.*, **6**(1):85–93, 1978.
28. R. Schassberger. Insensitivity of steady-state distributions of generalized semi-Markov process with speeds. *Advs. Appl. Prob.*, **10**:836–851, 1978.
29. R. Schreiber and H. D. Simon. Towards the teraflops capability for CFD. In H. D. Simon, editor, *Parallel CFD - Implementations and Results Using Parallel Computers.* MIT Press, 1992.

30. S. K. Setia. *Scheduling on Multiprogrammed, Distributed Memory Parallel Computers.* PhD thesis, Dept. Comp. Sci., Univ. Maryland, College Park, MD, 1993.
31. S. K. Setia, M. S. Squillante, and S. K. Tripathi. Processor scheduling on multiprogrammed, distributed memory parallel computers. In *Proc. ACM SIGMETRICS Conf.*, 158–170, 1993.
32. S. K. Setia, M. S. Squillante, and S. K. Tripathi. Analysis of processor allocation in multiprogrammed, distributed-memory parallel processing systems. *IEEE Trans. Par. Dist. Syst.*, **5**(4):401–420, 1994.
33. S. K. Setia and S. K. Tripathi. A comparative analysis of static processor partitioning policies for parallel computers. In *Proc. MASCOTS '93*, 1993.
34. K. C. Sevcik. Characterizations of parallelism in applications and their use in scheduling. In *Proc. ACM SIGMETRICS Conf.*, 171–180, 1989.
35. K. C. Sevcik. Application scheduling and processor allocation in multiprogrammed parallel processing systems. *Perf. Eval.*, **19**:107–140, 1994.
36. M. S. Squillante. MAGIC: A computer performance modeling tool based on matrix-geometric techniques. In *Proc. Intl. Conf. Mod. Tech. Tools Comp. Perf. Eval.*, 411–425, 1991.
37. M. S. Squillante. Analysis of dynamic partitioning in parallel systems. Tech. Rep. RC 19950, IBM Res. Div., 1995.
38. M. S. Squillante. On the benefits and limitations of dynamic partitioning in parallel computer systems. Tech. Rep. RC 19951, IBM Res. Div., 1995.
39. C. A. Thekkath and H. M. Levy. Limits to low-latency communication on high-speed networks. *ACM Trans. Comp. Syst.*, **11**(2):179–203, 1993.
40. A. Tucker and A. Gupta. Process control and scheduling issues for multiprogrammed shared-memory multiprocessors. In *Proc. ACM Symp. Op. Syst. Prin.*, 159–166, 1989.
41. T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active messages: A mechanism for integrated communication and computation. In *Proc. Intl. Symp. Comp. Arch.*, 256–266, 1992.
42. J. Zahorjan and C. McCann. Processor scheduling in shared memory multiprocessors. In *Proc. ACM SIGMETRICS Conf.*, 214–225, 1990.