A Methodology for the Evaluation of Multiprocessor Non-Preemptive Allocation Policies

E. Smirni*, E. Rosti†, L.W. Dowdy‡, G. Serazzi§

Abstract

In this paper a methodology for the evaluation of non-preemptive adaptive policies for multiprogramming multiprocessor systems is proposed. An evaluation criterion is derived, based on the systematic and consistent comparison of several allocation policies in a common framework. The comparison of the policies identifies the characteristic performance behavior of this type of policies. It suggests that equipartitioning strategies are an effective solution to the allocation problem when the workload components are statistically similar. The envelope of static equipartitioning policies is proposed as an evaluation criterion for non-preemptive adaptive policies. This criterion offers a guideline for the design of new policies and is useful in assessing the performance of existing ones. Analytical validation of the proposed criterion is presented for a simple case.

1 Introduction

The increasing power of hardware coupled with inherent limitations on software parallelism suggest the use of multiprogramming to provide better utilization of multiprocessor systems. With multiprogramming, various parallel applications are allowed to execute simultaneously on a set of processors. Since the late 80's several processor allocation strategies have been developed to implement multiprogramming. A convenient classification of processor allocation strategies distinguishes them as *preemptive* and *non-preemptive*.

*Department of Computer Science - University of Illinois, Urbana, Illinois, USA

†Dipartimento di Scienze dell'Informazione - Università di Milano, Italy

‡Department of Computer Science - Vanderbilt University - Tennessee, USA.

§Dipartimento di Elettronica e Informazione - Politecnico di Milano - Italy.

This work was partially supported by Italian M.U.R.S.T. 40% and 60% projects and by sub-contract 19X-SL131V from the Oak Ridge National Laboratory managed by Martin Marietta Energy Systems, Inc. for the U.S. Department of Energy under contract no. DE-AC05-84OR21400.

Non-preemptive policies do not permit processor preemption during application execution. By definition, non-preemptive policies space share the multiprocessor among the applications, i.e., partition the system into disjoint subsets of processing elements called partitions, each assigned to a different application. Non-preemptive policies can be further classified as static or adaptive. With static non-preemptive policies, the number and size of partitions is fixed for the system's lifetime [Sevc89, GST91, MEB91]. When an application is submitted for execution, it is assigned one of the partitions. With adaptive non-preemptive policies, the partition size assigned to an application is computed before the application starts execution and may vary in successive application executions depending upon the current state of the system and possibly other parameters [Sevc89, ZM90, NSS93, Sevc94, RSDSC94].

With preemptive policies, an executing application may be interrupted and part of its processors may be assigned to another application. Time sharing policies, by definition, are preemptive as tasks of various applications may be assigned to each processor. Processors may service their tasks in a round robin or priority based scheme [Oust82, LV90]. Space sharing policies may be preemptive if the partition allocated to an application is allowed to change during its execution. The number of processors assigned to an application changes depending on the system load [TG89, MZ94] or on changes in the applications' parallelism [ZM90]. Preemptive space sharing policies are also known as dynamic policies [TG89, DCDP90, ZM90]. Hybrid partitioning schemes are also preemptive, where the multiprocessor is divided into possibly non disjoint partitions. In each partition, tasks of different applications execute in time sharing mode [ZB91, SST93].

The focus of this paper is on non-preemptive policies as they represent a viable solution for multiprogramming parallel systems. Non-preemptive policies are a simple on-line alternative to off-line allocation policies based on users' requests exclusively [Int93]. Off-line allocations, either in batch or interactive mode, are often the strategy used on actual parallel machines that operate in production mode. By assigning the number of processors requested by the user, the application's execution time is guaranteed but not the waiting time. Non-preemptive policies are easy to implement and their run-time overhead is minimum as they do not context switch processors among different applications. Non-preemptive policies are not sensitive to the underlying hardware platform, therefore their implementation overhead is independent of the specific architecture. Non-preemptive adaptive policies are interesting for medium size parallel systems where application performance and system throughput are equally important targets.

In this paper a methodology for the evaluation of non-preemptive adaptive policies is proposed. The methodology results from the systematic and consistent comparison in a common framework of selected representative policies that have appeared in the literature. The comparison demonstrates the characteristic behavior of this type of policies. It shows that the various policies considered achieve comparable performance when the allocation algorithm tends to divide the system into partitions of equal size. Such a result suggests that equipartitioning strategies may be an effective solution to the allocation problem under given circumstances. Based upon these considerations, a performance evaluation criterion is derived for the design of new policies and the assessment of the performance of existing ones. Such a criterion is based upon the envelope of the performance curves of the static equipartitioning policies, i.e., the policies that partition the system into a fixed number of equally sized partitions at system configuration time. Depending on the workload characteristics and intensity, such a multiprogramming scheme may yield dramatically different performance. However, by tuning the number and size of partitions to the system load and the workload characteristics, good performance is achieved on a limited system load subrange. Analytical validation of the proposed criterion is presented.

This paper is organized as follows. Section 2 describes the policies considered. The comparison framework used for the comparison of the various policies is described in Section 3 and results of the comparison are presented in Section 4. The evaluation methodology and the performance evaluation criterion are presented in Section 5. Section 6 concludes the paper.

2 The Policies

In order to investigate the existence of a common performance behavior for non-preemptive adaptive policies and the influence of the parameters used, a set of policies of such a class are presented and compared. Among the several policies proposed in the literature, the more representative of the various classes of strategies have been selected.

The design philosophy of non-preemptive adaptive policies assumes that the partition size assigned to an application (job) is determined based on the current state of the system and possibly other parameters (e.g., the application average parallelism). It is decided before the job starts execution and does not change during the entire job's execution. Policies may require input parameters either of configuration type (i.e., those that depend upon system hardware or workload characteristics) or of run-time type (i.e., those that depend upon the current run-time state of the system). In the following sections, emphasis is placed on the required input parameters since they directly affect the cost and feasibility of policy implementation.

2.1 The Algorithms

In this section, the selected non-preemptive adaptive policies are presented. Before describing the policies, the definitions of the performance indices used by the policies are recalled. Average parallelism A of an application is the average number of processors utilized by the application assuming that an unlimited number of processors is available. Maximum (minimum) parallelism M (m) of an application is the maximum (minimum) number of processors that are simultaneously busy when a sufficiently large number of processors is available. Processor working set PWS of an application is the minimum number of processors that maximizes the efficacy, which is defined as S_p^2/p, where S_p is the job speedup when p processors are assigned to the job [GST91]. P is the total number of processors in the system.

The policies considered are: A+ [Sevc89]; run-to-completion (RTC) [ZM90]; processor working set (PWS) [GST91]; threshold (THR) [MEB91]; robust adaptive (RA) [RSDSC94]. The policy description follows.

A+

The A+ policy [Sevc89] is based on the application's average parallelism. In the lack of precise information, the policy assumes that the application minimum and maximum parallelism are 1 and P, respectively. It also assumes knowledge of the variance V in the application's parallelism. When the variance V is not known, then it is drawn from a uniform distribution over the interval $[0, (P-A) \times (A-1)]$. At each system load level, defined as $\rho = \frac{\lambda * T(1)}{P}$, where $T(1)$ is equal to the sequential execution time of the application when only one processor is allocated, all jobs are allocated the minimum between the available processors and an equal number of processors p, which is given by:

$$p = A + \frac{V}{(P-A)(A-1)}[h(1,P) - A] \tag{1}$$

where $h(1,P)$ is the function that adjusts the assignment depending upon the system load ρ:

$$h(x,y) = \begin{cases} y - \rho \times \frac{y-A}{\pi}, & \text{if } \rho \le \pi \\ x + (1-\rho) \times \frac{A-x}{1-\pi}, & \text{if } \rho \ge \pi \end{cases} \tag{2}$$

In the definition of $h(x,y)$, π is a parameter that roughly indicates the load level beyond which assigning A processors is no longer the optimal allocation. Following the rule of thumb indicated in [Sevc89], π can be set to .25. The minimum required parameters for the A+ policy are the workload arrival rate λ, the sequential execution time of the workload $T(1)$, and the average parallelism A.

Run-to-Completion

The run-to-completion (RTC) policy [ZM90] requires that the maximum parallelism of each application is known. The policy also assumes that the execution rate of the application when it is assigned p processors is known, for any p. Upon arrival to the system, a job is assigned the minimum between its maximum parallelism and the number of free processors. When a job finishes, the released processors are distributed one at a time to the waiting job i whose current assignment of processors, p_i, is less than its maximum parallelism and for which the expected improvement in elapsed execution time $T(p_i) - T(p_i + 1)$ is the largest. No job is assigned more than one processor if there are jobs that have not been allocated any processor yet. When all the released processors have been assigned, jobs begin execution.

Processor Working Set

The concept of processor working set is used as a configuration parameter for several adaptive policies [GST91]. Of the several possible variations, only the overall best policy is described here. This policy aims to assign to each job a number of processors that is equal to the job's PWS. The system waiting queue is searched for the first job whose PWS is less than or equal to the number of free processors. This job is allocated its PWS. When no job is found whose PWS can be satisfied, then the currently free processors are allocated to the first job in the waiting queue.

Threshold Policy

Partitioning policies that dynamically switch between different partition sizes chosen within a predefined set as a function of the number of jobs in the system are called threshold policies (THR) [MEB91]. A threshold policy is defined by the set of threshold queue length values $\mathcal{H} = \{h_0, h_1, \ldots, h_{n+1}\}$ and the corresponding set of partition sizes $\mathcal{P} = \{p_0, p_1, \ldots, p_n\}$. If the number of jobs in the system is between h_i and h_{i+1}, then the partition size is p_i. To guarantee good performance, the threshold-partition pairs must be tuned to the workload's characteristics. In the limit, as the difference $h_{i+1} - h_i$ becomes 1 for all i's, with $\mathcal{P} = \{P, \lfloor\frac{P}{2}\rfloor, \lfloor\frac{P}{3}\rfloor, \lfloor\frac{P}{4}\rfloor, \ldots\}$ the corresponding set of partition sizes, the THR policy converges to the equipartitioning dynamic policy. In the original version such policies are preemptive. Here, the non-preemptive version is considered. In this case, when a threshold is crossed, the partition size is adjusted for the jobs yet to be scheduled. When a running job completes execution, the released processors are assigned to the waiting jobs according to the (possibly new) partition size.

Robust Adaptive

Of the several policies studied in [RSDSC94], a robust adaptive policy (RA) that computes the partition size based only on run-time parameters is described here. A state dependent equipartitioning scheme that adapts to transient workload behavior without requiring any *a priori* knowledge about the workload characteristics is realized. The system is in an ideal state when the number of partitions is the same as the number of jobs in the waiting queue. The equally sized partitions are obtained as the total number of processors divided by the number of partitions. If the number of waiting jobs surpasses the number of partitions, the number of partitions is adjusted to the number of jobs waiting in the queue. On the contrary, if several jobs depart the system while fewer new jobs have arrived, the number of partions is decreased by 1 at each scheduling round, as long as the number of jobs waiting in the queue remains smaller than the number of partitions and there are at least two idle partitions of the current partition size.

2.2 Policy Characterization

Policies can be characterized based on the parameters they require. Such parameters contribute to the policy implementation costs. The A+ policy requires the maximum, minimum, and average parallelism, the variance in the applications' parallelism, and the workload arrival rate. However, the maximum and minimum parallelism, as well as the variance in the parallelism, can be guessed from system configuration parameters (i.e., P and 1 for maximum and minimum parallelism respectively) or from predefined distributions (e.g., the variance). The RTC policy requires the knowledge of the application's entire execution signature and maximum parallelism, and uses the number of idle processors and the queue length to compute the actual allocation size. The PWS policy bases its decisions only upon the application pws, as long as there are enough idle processors. The THR policy requires the threshold-partition sets, which define the partition sizes for all number of jobs in the system. The RA computes the partition size based on the queue length only.

The various parameters are illustrated in Figure 1 together with the policy that uses them. Parameters such as the number of idle processors and the queue length are measured at run-time. All policies, except for the RTC poplicy, use the number of idle processors to ensure the feasibility of the computed partition (i.e., verify that there are enough processors available for allocation). Therefore their name is not associated with it. Parameters such as average parallelism and job arrival rate are configuration and workload parameters, whose derivation require specific knowledge of the workload and of the system.

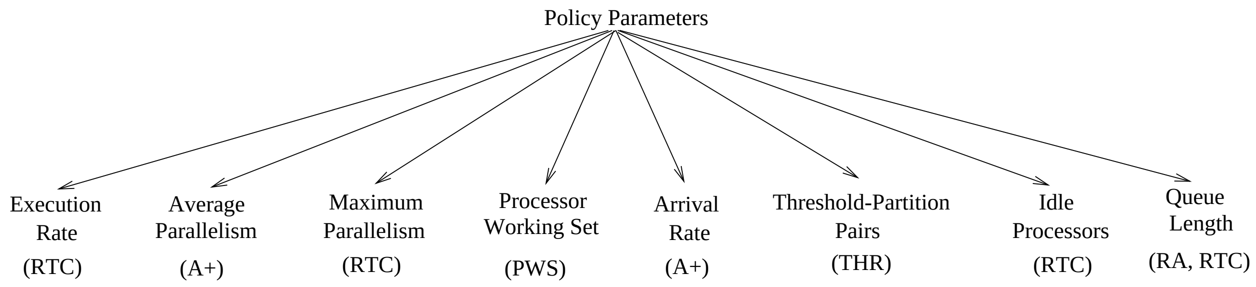

Figure 1: Characteristic policy parameters.

Unlike run-time parameters, which can be easily estimated, parameters that require specific knowledge of the workload and of the system are not always available in real environments, or they are difficult to acquire. In order to obtain the applications' execution times, the average parallelism, the maximum parallelism, and the processor working set, prior executions of the applications on various numbers of processors are required. Estimates of the execution times may be available, although precise evaluations are not possible without executing the applications. Tuning the threshold-partition pairs to the workload characteristics requires experimentation with various potential pairs. Using only run-time parameters, such as the queue length and the number of free processors, provides for low policy implementation costs in actual systems.

3 The Evaluation Framework

In order to compare the performance of the selected allocation policies, a common framework for comparisonis established such that the policies can be evaluated under the same workload conditions. The multiprocessor system is modeled as an open queueing network with one multiple server station (the multiprocessor) with a single queue. Arriving jobs join the waiting queue from which they are scheduled according to the specific allocation algorithm. Jobs arrive in a Poisson process and are served in first-come first-served order. Jobs are considered statistically identical and receive an exponentially distributed service time[1], whose mean value depends upon the number of allocated processors. The scheduler is activated every time a job arrives or departs. It determines the current partition size, as specified by the scheduling policy. Since only non-preemptive adaptive policies are considered, the scheduler overhead is neglected.

The workload is described by the average execution rates of each job for all possible processor assignments. The execution rates indicate job scalability as a function of the number of processors assigned. The effects of different architectural features are incorporated in the workload model. No explicit assumptions about the architecture are made.

A functional form is used to express the workload execution rates. Let μ_p be the execution rate of a job with p processors. Then,

$$\mu_p = \frac{1}{T(p)} = \mu_1 * \frac{1-\tau^p}{1-\tau} \quad p \geq 1,\ 0 \leq \tau < 1 \tag{3}$$

Eq. 3 characterizes a curve where, at each step, the increment is a constant fraction of the increment at the previous step, that is, $(\mu_{p+1} - \mu_p)/(\mu_p - \mu_{p-1}) = \tau$. When τ is approximately 0, the execution rate (hence the speedup curve) is flat (i.e., the job is intrinsecally sequential). At the other extreme,

[1] Experiments with various interarrival and service time distributions were also run, but are not reported here since the relative performance of the policies remains the same [Smi95].

when τ is approximately 1, the speedup is linear. For values of τ in [0,1), the speedup curve is concave increasing. The full workload execution rate is known to the allocation policies that uses it in their computation (see e.g., RTC). Four values of τ and μ_1 have been selected to characterize the execution rates and speedup curves (see Figure 2) of the workloads used in the following sections. They best represent the execution rates experimentally derived on an Intel Paragon for a matrix transposition kernel executed on 16 processors [SR95].

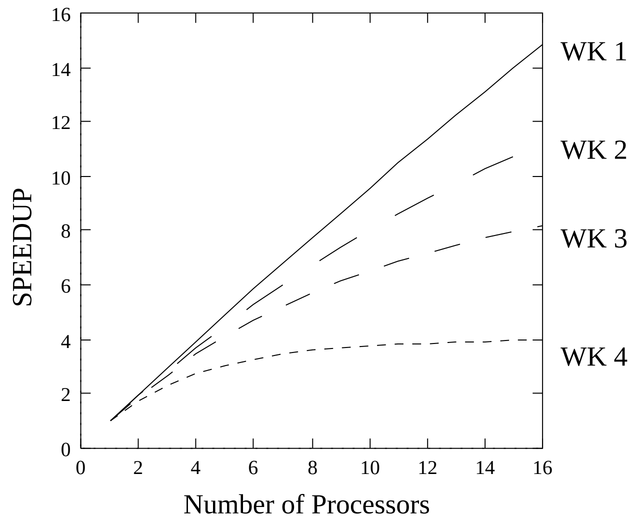

Figure 2: Speedup curves of the four representative workloads for a 16 processor system.

Table 1 summarizes the workload functional forms and parameters (i.e., the maximum parallelism M, the average parallelism A, the processor working set PWS, τ, and μ_1) of the four basic workload types used for policy evaluation. The same input stream of jobs is considered for all the policies.

All workload types have maximum parallelism equal to the system size since the functional form in Eq. 3 is monotonic increasing. This does not limit the generality of the analysis in this paper since interesting parallel applications usually scale well in systems like the one considered here.

workload	A	M	PWS	τ	μ_1	μ_p
WK1	15	16	16	0.99	0.555	$55.556(1-0.99^p)$
WK2	11	16	16	0.95	0.833	$16.676(1-0.95^p)$
WK3	8	16	12	0.90	0.487	$4.8780(1-0.90^p)$
WK4	4	16	4	0.75	0.625	$2.5(1-0.75^p)$

Table 1: Parameters of the four synthetic workloads.

To compare the policies, single class and multiclass workloads are considered. Six workload mixes are selected (see Table 2), four of which represent single class workloads (MIX1 – MIX4) and two represent multiclass workloads (MIX5 and MIX6). Multiclass workloads have been considered as they reflect actual system workloads and they adequately model execution time variability.

	WK1	WK2	WK3	WK4
MIX1	100%	–	–	–
MIX2	–	100%	–	–
MIX3	–	–	100%	–
MIX4	–	–	–	100%
MIX5	25%	25%	25%	25%
MIX6	–	50%	50%	–

Table 2: Workload mixes used for policy evaluation.

4 Policy Comparison

The five allocation policies are compared on the six workload mixes across the range of system loads via simulation. The parameters each policy requires are those reported in Table 1 or derived from it. The objective function used is *system power*, defined as the ratio of the system throughput to the average job response time [?]. Power is selected because it balances two conflicting but equally important performance metrics, i.e., throughput and response time. Throughput is maximized when the system is fully utilized, which typically leads to poor response time. Alternatively, response time is minimized when jobs are executed alone in the system, which implies poor throughput. Maximizing power balances these effects.

The entire range of system load, defined as $\rho = \frac{\lambda * T(1)}{P}$, is simulated and results are reported as a function of ρ. All simulations are run with 95% confidence intervals. The confidence intervals are not indicated in the figures for the sake of clarity.

The policy comparisons are illustrated in Figure 3. The relative policy performance varies depending on the system load and on the workload mix. Each policy was implemented assuming that the parameters it requires are available. Specific observations are possible regarding each policy. The focus here is on the common performance behavior.

When the policy allows for smaller partitions as the system load increases, performance is sustained across the entire range of system load. Policies like PWS and THR that do not allow for partition fragmentation at high load levels may reach high power values but saturate before the load has reached 100% of system capacity. With THR, adding smaller partitions to $\mathcal{P}$, the partition size set, accounted for a significant reduction in power. Therefore partition sizes that yielded better performance on a limited range of system load were preferred. With PWS, fragmentation is possible only if the jobs comprising the workload have widely different PWS's (see Figure 3 for MIX5).

Using workload characteristics in scheduling decisions does not provide for better performance. Policies that use only run-time parameters, such as RA, perform as good, if not better, than policies that base their decisions upon workload parameters such as average parallelism or execution signature.

Except for specific cases, such as A+ on MIX1, PWS on all mixes but MIX1, THR on MIX2, MIX4, and MIX5, all policies exhibit similar behavior. The range of maximum power defined by the policies is reasonably narrow to associate a common performance behavior to all of them.

Table 3 qualitatively summarizes the results of the policy comparison from a performance point of view for the six workload mixes, based on the graphs plotted in Figure 3. The relative performance of each policy on the six examined workloads is characterized as "good", "average", or "poor" according to the following criteria.

- Policies that achieve high power values (i.e., within 10% of the maximum power among all policies) over all system loads are characterized as "good".
- Policies that reach reasonable power values (i.e., within 25% of the maximum power among all policies) for a broad subrange of the potential system loads (i.e., at least 75% of system load range), are considered "average".
- Policies that perform well only over a limited subrange of the load (i.e., less than 75% of system load range), or achieve low power values (i.e., below 25% of the maximum) are classified as "poor".

	MIX1	MIX2	MIX3	MIX4	MIX5	MIX6
PWS	good	poor	poor	poor	average	poor
RTC	average	good	good	good	average	average
A+	poor	average	good	good	good	average
THR	good	good	average	average	average	average
RA	average	good	good	good	good	good

Table 3: Summary of policy performance for the six workload mixes.

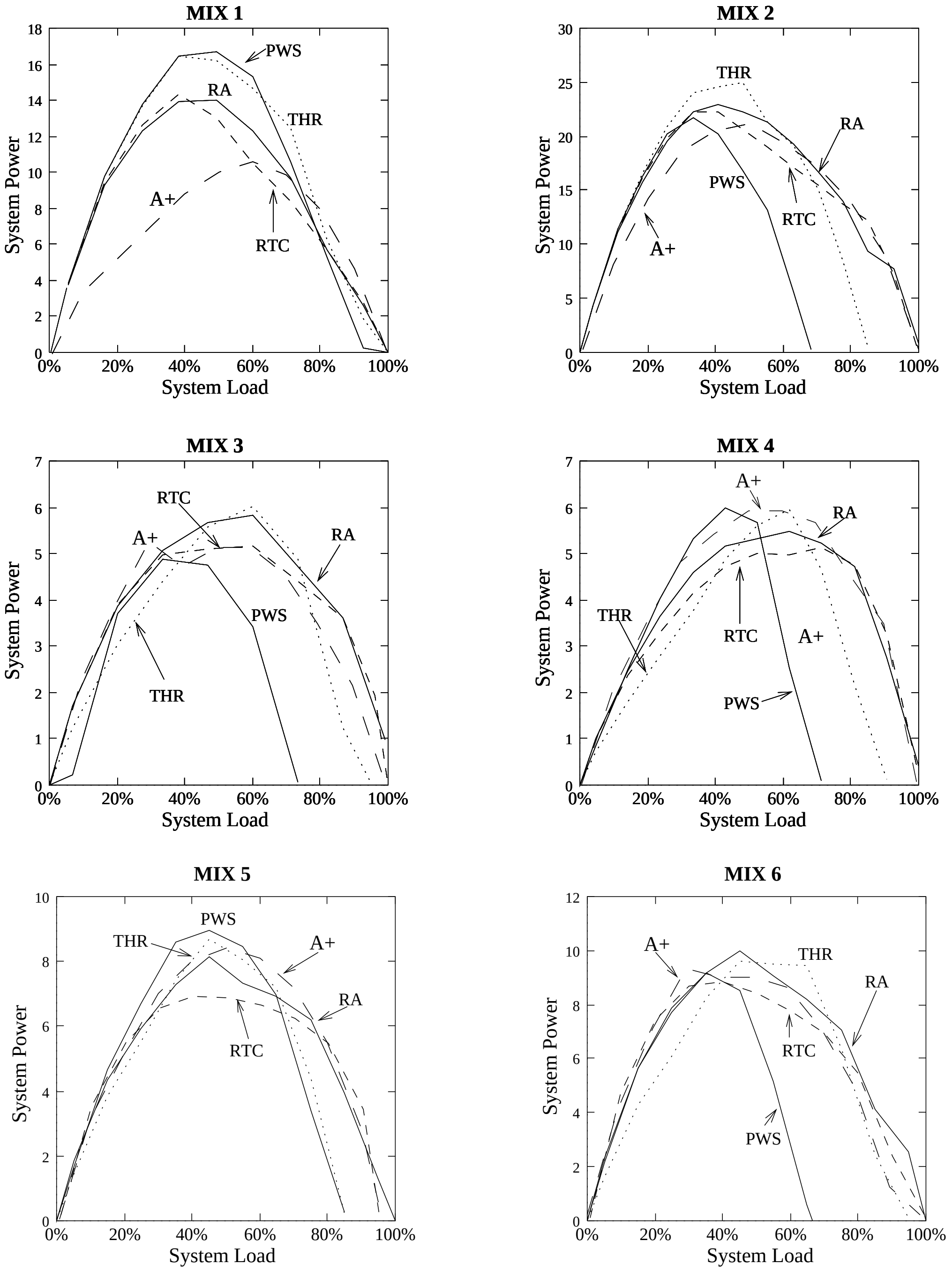

Figure 3: System power for the six workload mixes as a function of the system load.

Table 3 and Figure 3 confirm expected results:

- knowing workload parameters such as maximum and average parallelism or the execution rate does not significantly improve the performance of non-preemptive adaptive policies;
- policies based on run-time parameters (e.g., RA) generally give good performance;
- adaptive equipartitioning schemes are an overall good choice also when the workload characteristics are ignored (e.g., RA compared to RTC or A+)

The results of the comparison suggest that the ideal policy would be one whose power is a collection of the highest values of all power curves across all policies for all workload mixes and for all system loads (see Figure 3). In the next section a characterization of the performance of such a policy is presented and derived in terms of simple components. This gives a criterion for evaluating the effectiveness of any newly proposed policy.

5 Evaluation Methodology

The uniform comparison of non-preemptive adaptive policies presented in the previous section suggests that it may be possible to identify a common performance target that characterizes the maximum power achievable by this type of allocation policies. Such a target would be an evaluation criterion for guiding the design of new policies and assessing the performance of existing ones. In this section the performance achievable with static equipartitioning policies is identified as characterizing the common behavior observed in the previous section. The performance of the dynamic space sharing policy is also identified as the optimal performance bound.

The proposed evaluation methodology consists of four steps:

1. characterization of the system workload using the speedup curve of its parallel applications;
2. identification of the adaptive policy performance target by constructing the *envelope* of the power curves of the static equipartitioning policies;
3. identification of the dynamic policy performance bound by deriving the power curve for the dynamic space sharing policy;
4. evaluation of the performance of the scheduling policy of interest.

A good non-preemptive adaptive policy would achieve power values in the range of the envelope curve, or possibly above it, in the area between it and the curve of the dynamic policy.

In what follows the envelope curve is derived and it is shown that, although effective, it is not a strict upper bound for the performance of all non-preemptive adaptive policies. However, the exhaustive search of the policy space of a limited system confirms the quality of the proposed criterion.

5.1 The Envelope Criterion

The common philosophy of the policies investigated in Section 4 suggests that an equipartitioning strategy tuned for the specific workload intensity and characteristics performs well. In order to derive the performance target in an easy to reproduce way, the family of non-preemptive static equipartitioning policies (SEP) is considered. These policies [Sevc89, EZL89, ZM90, GST91], partition the multiprocessor into disjoint subsets of processors of equal size at system configuration time. For a fixed number of partitions, n, and a single class workload, the system is modeled as an $M/M/n$ queue. It is well known that for each number of partitions (i.e., for each policy) and for a given workload, there is a limited subrange of the system load where the policy performs well. The size and location of the optimal subrange of each policy depend upon the partition size, the system load, and the characteristics of the workload. As an example, the power curves for the possible SEP policies for a 16 processor system (i.e., namely SEP16, SEP8, SEP4, SEP2, and SEP1 for partitions of 16, 8, 4, 2, and 1 processor, respectively) are reported in Figures 4(a) and 4(b) for WK1 and WK3, respectively. Each curve in the figure is analytically derived, by solving the $M/M/n$ model corresponding to the specific SEPp policy, where $n = 16/p$ is the number of partitions.

As observed in Figure 4, no single SEP policy is optimal, nor effective, over the entire system load range. However, the collection of the pieces where each curve is best exhibits good overall behavior. The coarse granularity of the discrete partitions sizes accounts for possibly jagged behavior of such a strict envelope that simply connects the optimal portions of each curve. Larger systems where the variety of partition sizes of the SEP policies is greater present smoother strict envelopes. Therefore the envelope assumed as a performance target for non-preemptive adaptive policies is the one obtained as though a continuum of partition sizes were possible. It is the concave curve associated with the outermost points of the collection of all SEP power curves across the range of system load.

Figure 5 shows the envelope (bold line) added to Figures 3(a) and 3(c). As seen, the envelope captures the best features of the policies compared in the previous section.

Figure 5 also includes the power curve of the dynamic equal partitioning policy with no repartitioning overhead. The dynamic policy is preemptive and assumes that it is possible to redistribute processors to jobs during execution at no cost. Upon job arrival or departure, the processors are dynamically redistributed among all jobs in the system to enforce partitions of equal size. The dynamic curve is derived analytically by solving the corresponding Markov chain with arival rate

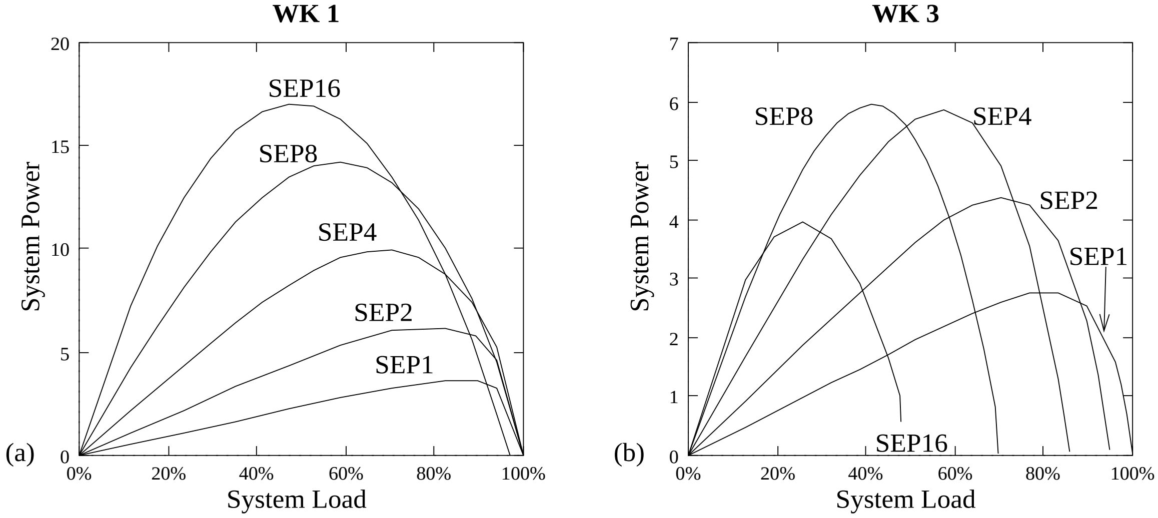

Figure 4: Power curves of the Static Equipartitioning Policies (SEPn, where n is the partition size) as obtained from the solution of their analytical models.

λ and service rates $n\mu(p)$, where $n = 1, \ldots, P$ is the number of jobs currently in execution and p is the partition size assigned to each of them. The policy defines a performance upper bound as it optimally switches between equal sized partitions as the number of jobs in the system increases. When the overhead associated with processor reallocation is considered, the performance of the dynamic policy quickly drops. By varying the amount oerhead, the performance curve spans the area between the dynamic policy curve and the envelope. When no repartitioning overhead is considered, the dynamic policy gives a theoretical upper bound to all non-preemptive policies. However, depending upon the workload scalability, this bound may not be useful as an effective evaluation criterion. If the workload scales well (e.g., WK1, see Figure 2), the gap between the power curve of the non-preemptive policies and the theoretical bound is small (see Figure 5(a)). As illustrated in Figure 5(b), the gap between the envelope and the curve of the dynamic policy increases in the case of less linear workloads (e.g., WK3). The less linear the workload, the wider this gap becomes. In such a case, application with poor speedup characteristic cannot exploit large partitions, which, however, cannot be reassigned to other jobs, thus inflating jobs' waiting time. The dynamic policy immediately switch to smaller partitions as soon as new jobs arrives, thus minimizing their waiting time.

The envelope as a performance criterion has been successfully examined via simulation in multiclass workloads whose composition varies dynamically over time and for various interarrival and service time distributions [Smi95]. In both single class and multiclass cases, the envelope serves as a criterion for policy performance evaluation since it is robust with respect to various workload

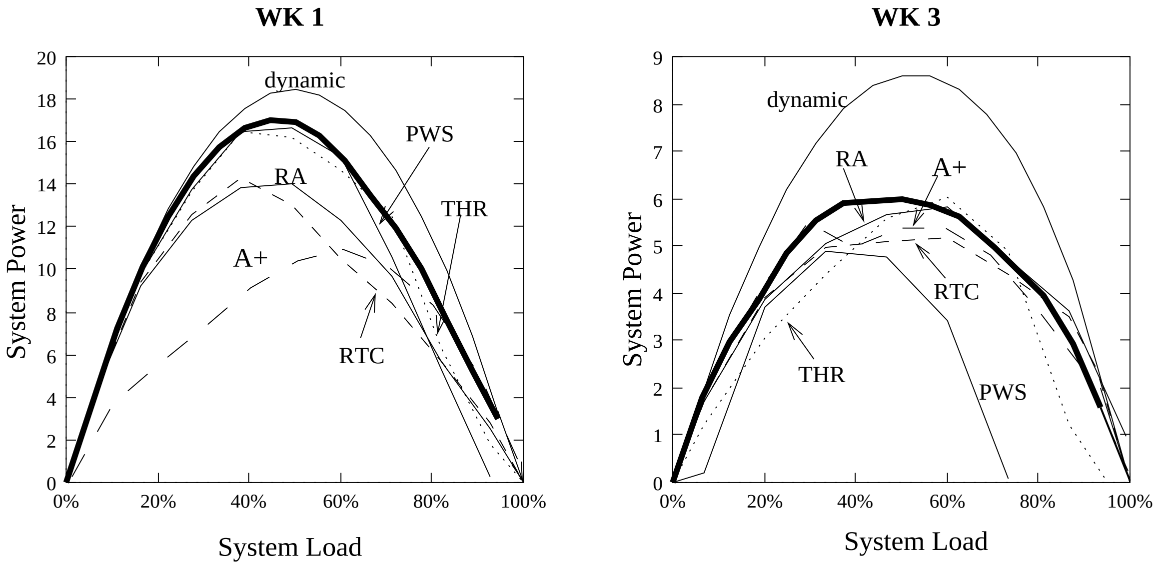

Figure 5: Envelope of the SEP policies (bold line) compared to the power of the non-preemptive adaptive policies and to the power of the preemptive dynamic policy.

types and across the range of system loads. Allocation policies whose performance closely matches that of the envelope would be very effective policies. Whether the envelope could set a performance bound such as the dynamic policy is the subject of the following section.

5.2 Bound Analysis

In this section, Markovian models of non-preemptive policies are used to investigate the existence of non-preemptive policies whose power curves exceed the envelope and approach the bound of the dynamic policy. From all possible non-preemptive policies, the threshold policy is selected as a good candidate since its behavior closely resembles the behavior of both the SEP and the dynamic policies. The dynamic policy can be approximated to any degree of accuracy, up to convergence, by the preemptive threshold policies (see Section 2.1). Since the focus of the paper is on non-preemptive adaptive policies, the non-preemptive version is investigated here, using Markovian analysis.

Assuming exponential interarrival and service times, with mean $1/\lambda$ and $1/\mu$ respectively, the system behavior under the allocation policy is described by a Continuous Time Markov Chain (CTMC). In order to keep the analysis tractable from a computational point of view, the simple case of policies with only one threshold, THR1, is analyzed. Let $(q_0^{p_0} q_1^{p_1} \ldots q_n^{p_n})$ denote the system state when there are q_k jobs executing on p_k processors each. $q_k^{p_k}$ will only appear in the state description if it is non-zero. $(-)$ indicates the idle state. When $p_k = 0$, jobs are waiting for service in the queue. As an example, state $(1^0 2^8)$ indicates that there is 1 job whose partition size is zero (i.e., waiting in the queue), two jobs executing on 8 processors each, and no job executing on 16 processors (the component is missing).

With a single threshold, two partition sizes are possible. With the single threshold set at 1, the two partitions are selected as the whole system (i.e., 16 processors) and half the system (i.e., 8 processors). If the number of jobs in the system $q = \sum_{i=0}^{i=n} q_i^{p_i}$ becomes greater than the threshold, then the smaller partition size (half the system in this case) is used for scheduling new jobs. Likewise, if q is less than or equal to one (i.e., the threshold) then the larger target partition size (the entire system) is used for scheduling jobs.

When the exponential service times for both partition sizes have average $1/\mu_{16}$ and $1/\mu_8$, respectively, the CTMC describing the system behavior under THR1 policy is shown in Figure 6. If the threshold policy has h threshold values, then the corresponding CTMC has $h+1$ horizontal chains. Each horizontal chain is associated with a partition size. As the number of chains increases, the complexity of the analytical solution increases dramatically. Due to this complexity, the focus is on a single threshold policy, THR1.

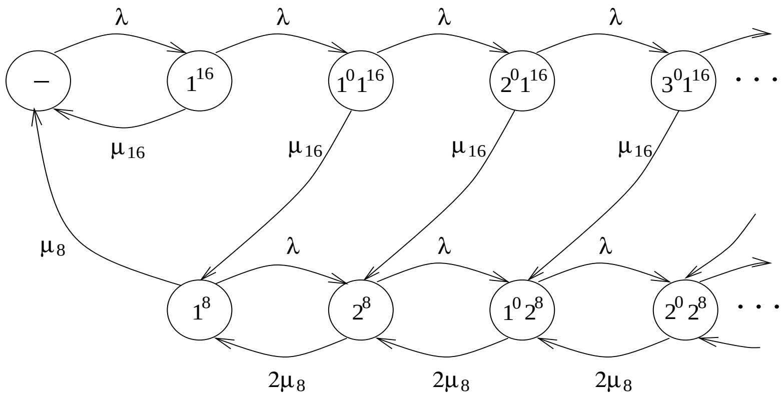

Figure 6: Continuous Time Markov Chain of the THR1 policy.

The Markov chain of Figures 6 is solved analytically using standard techniques (see Appendix A) and the system power is derived. Figure 7 shows the power achieved by THR1 in comparison with the dynamic policy and the envelope (dashed line) for the case of a linear (WK1) and a non-linear workload (WK2). When the workload is linear (see Figure 7(a)), the THR1 policy achieves higher values than the envelope when the system load ranges between 70% to 90% of the system load capacity. In the case of WK2 (see Figure 7(b)) the load range where THR1 exceeds the envelope is between 30% to 60% of system load capacity. As the workload scalability decreases (e.g., WK3 and WK4), THR1 exceeds the envelope at lower system load levels. The possibility of switching to smaller partitions when the workload has poor speedup characteristics improves performance as applications cannot take advantage of large partitions

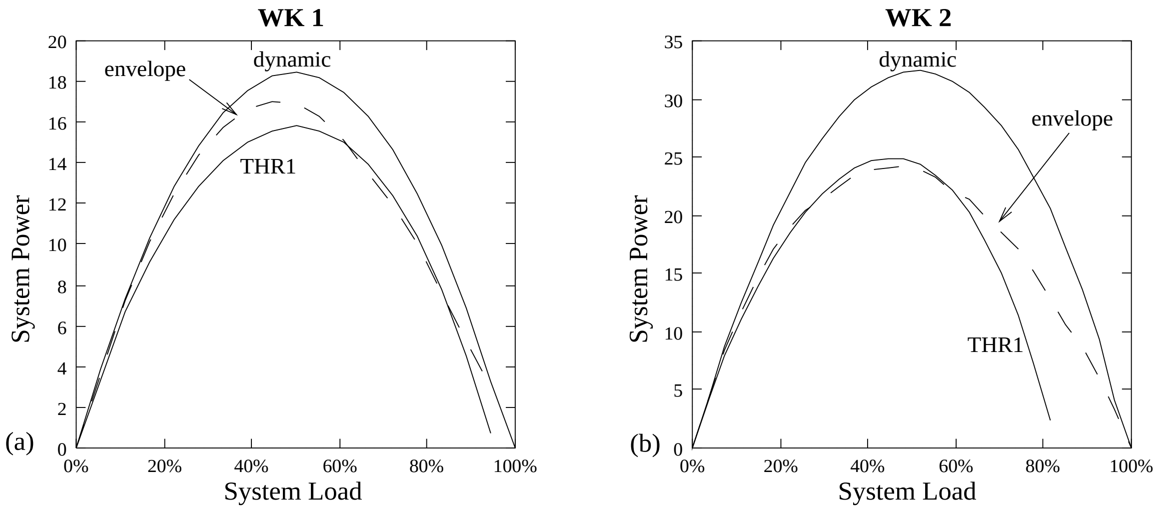

Figure 7: Power curves of the envelope (dashed line), THR1 (dotted line), and dynamic policy (solid line).

In all situations, the envelope is not a strict upper bound. However, it is not exceeded significantly by other non-preemptive policies considered. Thus, according to all experimental evidences, the envelope is a good evaluation criterion for the set of all non-preemptive policies.

As evidence to the quality of the envelope, the state space of all possible policies of a limited system has been explored. A system of limited size with finite capacity (i.e., where the total number of jobs allowed in the system is finite) is used to exhaustively search for the best non-preemptive policy. To show the existence, or the lack thereof, of policies that perform better than the envelope, an exhaustive enumeration of all possible non-preemptive policies is performed. The results of the exhaustive search are illustrated in Figure 8, which plots the power curves of the 511 policies found for the system considered. The graph confirms that the envelope of the SEP policies for the system considered achieves the best power values. However, there is a point where another policy, a threshold policy, performs as good as the envelope. (See Appendix B for the details.)

As the counterexample to the 16 processor system shows, the theoretical results obtained for the exhaustive search of the closed system do not generalize to systems with infinite capacity. However, this example supports the quality of the envelope as the power of an overall good policy for systems where arrivals are not discouraged beyond a given limit. Furthermore, it seems unlikely that policies that perform better than the curve enveloping the SEP curves and exhibit comparable robustness exist. Such a characteristic and its simple derivation based on the SEP curves make the

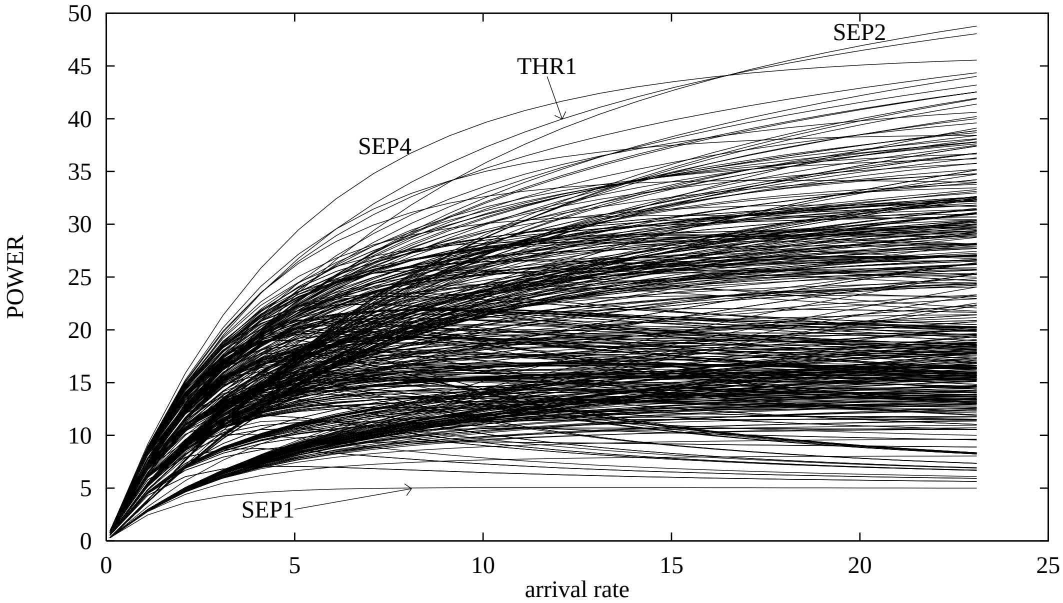

Figure 8: Power curves of all 511 possible non-preemptive policies for 4 processor system with finite capacity of size 2 on WK2.

envelope a convenient and helpful criterion for policy evaluation.

6 Conclusions

In this paper, a methodology for the evaluation of non-preemptive adaptive policies for processor allocation in multiprocessor systems is proposed. Various issues related to the evaluation of such policies have been addressed. It consists of four steps: the characterization of the workload submitted to the system, the construction of the envelope of the static equipartitioning policies power curves as a performance target, the construction of the optimal performance bound given by the dynamic space sharing policy, and finally the assessment of the performance of the policy under evaluation. Such a methodology is based upon the results of a consistent and systematic comparison of several representative non-preemptive adaptive policies under a common framework. The policies are: the run-to-completion (RTC) policy, the processor working set (PWS) policy, the A+ policy, the threshold policy (THR), and the robust adaptive (RA) policy. Policies have been compared across single class and multiclass workloads and across various system load levels. The issue of parameters required for policy implementation has been addressed and the advantages of policies based on run-time parameters have been indicated.

The policy comparison has led to the definition of a performance evaluation criterion. A performance evaluation criterion for non-preemptive policies, the envelope of the power curves of static equipartitioning policies, has been proposed. The envelope provides a uniform criterion of comparison against which all non-preemptive policies can be evaluated. The envelope is easy to derive since the basic components, the static equipartitioning policies, admit analytical models that can be solved with little effort. Although it is not a strict upper bound, no policy that performs considerably better than the envelope is known. The exhaustive search of all possible policies for a small system with limited capacity supports the quality of the envelope as a performance target of non-preemptive policies.

References

[EZL89] D. L. Eager, J. Zahorjan, and E. D. Lazowska, "Speedup versus efficiency," *IEEE Transaction on Software Engineering*, Vol 38(3), 1989, pp. 408-423.

[DCDP90] K. Dussa, B. M. Carlson, L. W. Dowdy, and K.-H. Park, "Dynamic partitioning in a transputer environment," *Proc. ACM SIGMETRICS*, 1990, pp. 203-213.

[GST91] D. Ghosal, G. Serazzi, and S. K. Tripathi, "Processor working set and its use in scheduling multiprocessor systems," *IEEE Trans. Software Eng.*, Vol 17(5), 1991, pp. 443-453.

[Int93] Intel Corporation, **Paragon OSF/1 User's Guide**, 1993.

[LV90] S. T. Leutenegger and M. K. Vernon, "The performance of multiprogrammed multiprocessor scheduling policies," *Proc. ACM SIGMETRICS*, 1990, pp. 226-236.

[MEB91] S. Majumdar, D. L. Eager, and R. Bunt, "Characterization of programs for scheduling in multiprogrammed parallel systems," *Performance Evaluation*, Vol 13(2), 1991, pp. 109-130.

[MVZ93] C. McCann, R. Vaswani, and J. Zahorjan, "A dynamic processor allocation policy for multiprogrammed shared memory multiprocessors," *ACM Trans. Comput. Syst.*, Vol 11(2), 1993, pp. 146-178.

[MZ94] C. McCann, J. Zahorjan, "Processor allocation policies for message-passing parallel computers," *Proc. ACM SIGMETRICS*, 1994, pp. 19-32.

[NSS93] V.K. Naik, S.K. Setia, M.S. Squillante, "Performance analysis of job scheduling policies in parallel supercomputing environments," Proc. of *Supercomputing 93*, 1993, pp. 824-833.

[Oust82] J. Ousterhout, "Scheduling techniques for concurrent systems," *Proc. 3rd International Conference on Distributed Computing Systems*, 1982, pp. 22-30.

[RSDSC94] E. Rosti, E. Smirni, L. W. Dowdy, G. Serazzi, and B. M. Carlson, "Robust partitioning policies for multiprocessor systems," *Performance Evaluation*, Vol 19(2-3), March 1994, pp. 141-165.

[SST93] S. K. Setia, M. S. Squillante, and S. K. Tripathi, "Processor scheduling in multiprogrammed, distributed memory parallel computers," *Proc. ACM SIGMETRICS*, 1993, pp. 158-170.

[Sevc89] K. C. Sevcik, "Characterization of parallelism in applications and their use in scheduling," *Proc. ACM SIGMETRICS*, 1989, pp. 171-180.

[Sevc94] K. C. Sevcik, "Application scheduling and processor allocation in multiprogrammed multiprocessors," *Performance Evaluation*, Vol 19(2-3), March 1994, pp. 107-140.

[Smi95] Smirni E., "Processor allocation and thread placement policies in parallel multiprocessor systems," Ph.D. Dissertation, CS-Dept. Vanderbilt University, TN, May 1995.

[SR95] Smirni E., Rosti E., "Modeling speedup of SPMD applications on the Intel Paragon: a case study," Proc. of *High-Performance Computing and Networking 95 Europe*, B. Hertzberger, G. Serazzi Eds., Lecture Notes in Computer Science 919, pp 94-101, Italy, May 1995.

[TG89] A. Tucker and A. Gupta, "Process control and scheduling issues for multiprogrammed shared-memory multiprocessors," *Proc. of the 12th ACM Symposium on Operating Systems Principles*, 1989, pp. 159-166.

[ZM90] J. Zahorjan and C. McCann, "Processor scheduling in shared memory multiprocessors," *Proc. ACM SIGMETRICS*, 1990, pp. 214-225.

[ZB91] S. Zhou and T. Brecht, "Processor pool-based scheduling for large-scale NUMA multiprocessors," *Proc. ACM SIGMETRICS*, 1991, pp. 133-142.

A Appendix A

The steady state probabilities $P_{(q_0^{p_0}\, q_1^{p_1})}$ for the THR1 policy are derived in this appendix. Let $\{(-)$, $(0^0 1^8)$, $(i^0 1^{16})$ for $i \geq 0$, $(j^0 2^8)$ for $j \geq 0\}$ be the set of all system states under the THR1 threshold policy as described by the Markov chain in Figure 6. By solving the global balance equations of the system, the steady state probabilities are given by:

$$\pi_{-} = \frac{\mu_{16}(\lambda + \mu_{16})(2\mu_8 - \lambda)}{2\mu_8\lambda^2 + 4\lambda\mu_8\mu_{16} - \lambda\mu_{16}^2 + 2\mu_8\mu_{16}^2} \tag{4}$$

$$\pi_{1^8} = \frac{\lambda^2}{\mu_8(\lambda + \mu_{16})}\pi_{-} \tag{5}$$

$$\pi_{i^0 1^{16}} = \left(\frac{\lambda}{\lambda + \mu_{16}}\right)^{i+1} \pi_{-} \qquad \text{for} \quad i \geq 0 \tag{6}$$

$$\begin{aligned} \pi_{i^0 2^8} &= \sum_{k=1}^{i+1} \left(\frac{\lambda}{2\mu_8}\right)^k \left(\frac{\lambda}{\lambda + \mu_{16}}\right)^{i+3-k} \pi_{-} + \\ & \frac{\lambda^2}{\mu_8(\lambda + \mu_{16})} \left(\frac{\lambda}{2\mu_8}\right)^{i+1} \pi_{-} \text{ for } \quad i \geq 0 \end{aligned} \tag{7}$$

Using the state probabilities, the closed form expression for the system power can be computed.

B Appendix B

This appendix illustrates the details of the exhaustive search of all possible non-preemptive policies for a small case. A system of limited size (4 processors) with finite capacity (i.e., where the total number of jobs allowed in the system is finite) is used to exhaustively search for the best non-preemptive policy. When the capacity is infinite, the number of policies is infinite and an exhaustive search is not feasible. By limiting the capacity, an exhaustive enumeration of all possible non-preemptive policies is performed to show the existence, or the lack thereof, of policies that perform better than the envelope.

The system is modeled as a queueing system with finite capacity of size 2. Jobs arriving when there are already two jobs in the system are ignored. The system state is described, as before, by the vector of queue lengths for all possible assignments, $(q_0^{p_0} q_1^{p_1} \ldots q_n^{p_n})$, for non-zero queue lengths. The possible assignments are 0, 1, 2, 3, and 4 processors, respectively. The set of feasible states is: $\{(-), (1^1), (1^2), (1^3), (1^4), (1^0 1^1), (1^0 1^2), (1^0 1^3), (1^0 1^4), (1^1 1^2), (1^1 1^3), (2^1), (2^2)\}$. States where the multiprocessor is idle and there are jobs waiting for service are excluded from the state space. A policy is defined as a set of states and a set of deterministic transition rules that specify the successor state for each state of the set. As an illustration, Figure 9 represents the entire state space and three possible non-preemptive policies, SEP2 (dotted line), SEP4 (dashed line), and THR1 (solid line). THR1 is a threshold policy with 1 threshold set at 1. If the multiprocessor is idle, the whole system (4 processors) is assigned to the incoming job (i.e., a transition to state (1^4)). If a second job arrives, it waits in the queue (i.e., transition to state $(1^0 1^4)$). When the second job arrives, the threshold is corssed and the system switches to a different partition size, 2 in this case, which will be assigned as soon as enough free processors will be available. When the first job finishes, the second job is scheduled on the new partition size, half of the multiprocessor (i.e., a transition to state (1^2)). State (1^2) is not desirable because system resources are not fully utilized. State ($1^0 1^4$) is not desirable because there is a job waiting. States (1^4) and (2^2) are "good states" because they maximize system utilization and minimize response time. These are the states the dynamic policy would traverse.

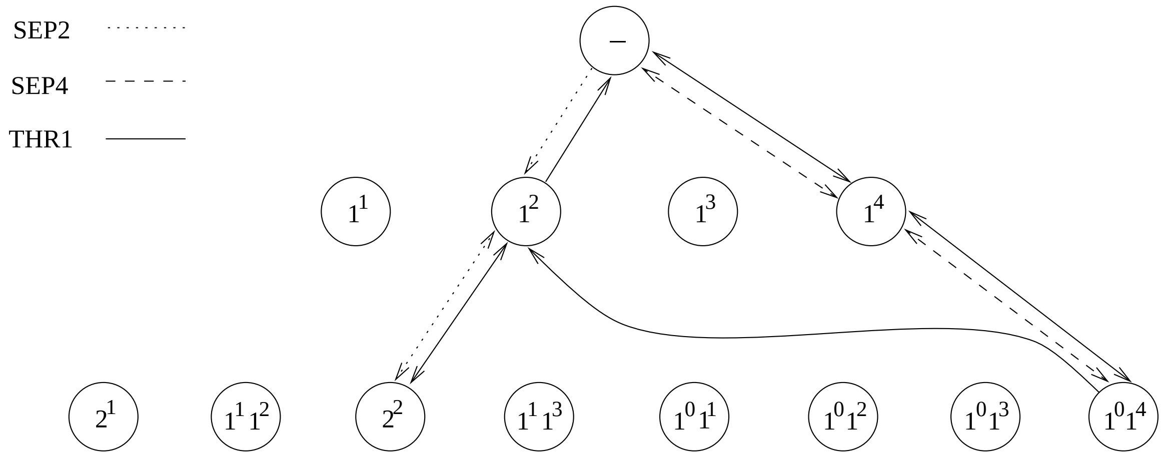

Figure 9: State space for 4 processor system with finite capacity of size 2 and possible policies.

The behavior of any policy can be described by a CTMC. Figure 10 shows the Markov diagrams of the SEP2, SEP4, and THR1 policies. As the diagrams show, THR1 is obtained by the composition of SEP2 and SEP4. It can also be observed to be the initial portion of the CTMC reported in Figure 6 truncated when the number of customers in the system has reached the system capacity (i.e., 2). The first three states of the upper chain and the first two of the lower chain are considered. In the system with finite capacity the RA policy is equivalent to THR1, while PWS and RTC are equivalent to SEP4.

A recursive procedure is used to search the state space to identify all possible non-preemptive policies for the system considered. Starting from the initial state ($-$), the state space is searched breadth first, exploring the states where there is initially only one job in the system. From one state, all the other states are considered as possible successors. If a state is not a feasible successor (e.g., the number of jobs in the system does not change, or the partition sizes associated with the two states do not match), it is discarded, and another state is examined. When a feasible sequence of states is found the corresponding policy is analytically evaluated. The exhaustive search of the system state space yields 511 policies. Policies that keep the whole system idle when there are jobs waiting for service have been neglected as obviously inefficient policies. The large number of policies is due to the fact that a different action can be specified for each feasible state. All combinations of

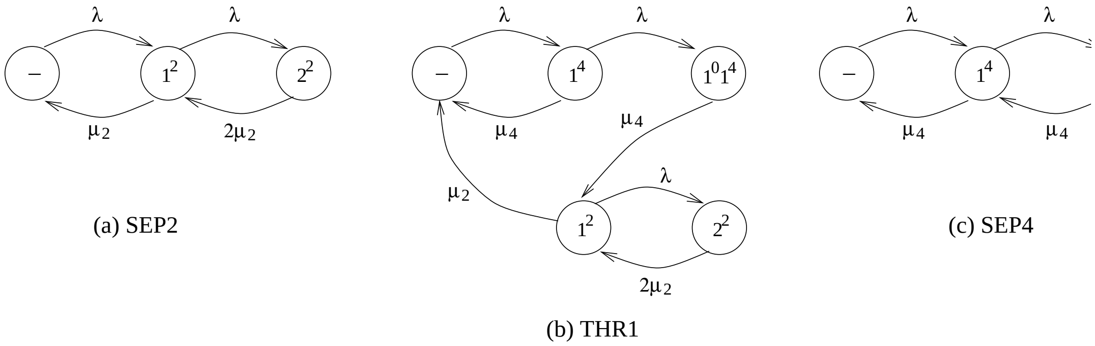

Figure 10: Markov diagrams for SEP2 (a), THR1 (b), and SEP4 (c) for a 4 processor system with finite capacity of size 2.

system states and transition rules are examined. If the capacity increases to 3, more than 100,000 policies exist.

The power curve of each of the 511 policies has been derived via Markov analysis for the various single class workloads. Figure 8 illustrates the set of curves for the 511 different policies for WK2. Because the capacity is finite, the system does not saturate and the power curves do not drop to 0. Although this figure is extremely dense, it is quite informative. Different policies are optimal at different arrival rates. The two best policies are static equipartitioning policies, namely SEP4 at low load and SEP2 at high load. The most inefficient curve in Figure 8 corresponds to SEP1. In this case, the power curve is very low because at most one processor is assigned to every job in the system. The system is heavily underutilized and its power curve is low.

It is interesting to note that there is one policy that is as good as both SEP2 and SEP4 at exactly one point, where the SEP4 and SEP2 curves cross. This particular policy is a THR1 threshold policy with the small partition size set to 2, the large partition size set to 4, and the threshold value set to 1. Unlike the SEP policies, THR1 adapts to the system load by assigning 4 processors to a job arriving at an idle system and 2 processors in all other cases. The crossover point for the three policies (i.e., THR1, SEP4, and SEP2) always exists, but its location depends upon the type of workload. This property can be proven analytically. When the capacity is finite, no policy outperforms the envelope of the SEP policies.