A parallel workload model and its implications for processor allocation

Allen B. Downey *

Abstract

We develop a workload model based on observations of parallel computers at the San Diego Supercomputer Center and the Cornell Theory Center. This model gives us insight into the performance of strategies for scheduling moldable jobs on space-sharing parallel computers. We find that Adaptive Static Partitioning (ASP), which has been reported to work well for other workloads, does not perform as well as strategies that adapt better to system load. The best of the strategies we consider is one that explicitly reduces allocations when load is high (a variation of Sevcik's A+ *strategy [10]).*

Keywords: *parallel, space-sharing, partitioning, scheduling, allocation, moldable, multiprocessor.*

1 Introduction

Space-sharing, distributed-memory multiprocessors, like the Intel Paragon, the Cray Research T3E and the IBM SP2, are often used in supercomputing environments to run scientific applications. These environments typically have the following characteristics:

- For batch processing, jobs do not share processors, but rather allocate a *cluster* of processors exclusively and *run to completion*. Many of these machines also have an interactive partition that uses timesharing, but in the environments we have observed, the vast majority of computation is done in batch mode. Thus, this paper only addresses scheduling strategies for batch partitions (pure space-sharing).
- Many jobs on these systems are *moldable*, meaning that they are capable of running on a range of cluster sizes. On the other hand, the programming models used for scientific applications usually do not generate jobs that can change cluster sizes dynamically. Thus, once a job begins execution, its cluster size is fixed.

In current systems, users choose cluster sizes for their jobs by hand, and the system does not have the option of allocating more or fewer than the requested number of processors. The factors that should influence this choice include the characteristics of the program, the load on the system and the performance requirements of the user. But users generally do not have the information, tools, or inclination to consider all of these factors. Allowing the system to make this decision has the potential to improve system utilization and reduce users' wait times.

Toward this end, prior studies have proposed *allocation strategies* that choose cluster sizes automatically. Most of these studies evaluate the proposed strategies with analysis and simulation based on hypothetical workloads. Each of the strategies in this paper has been evaluated in at least one prior study, under a variety of workload models [4] [10] [9] [11] [6] [1] [7]. Chiang et al. compare the performance of these strategies over a wide range of workload parameters, and argue that the discrepancies among various studies are due to differences in the hypothesized workloads [1].

The goal of this paper is to focus debate by constructing a new workload model based on observations of space-sharing systems running scientific workloads. By studying real systems, we would like to find the the range of workload parameters most descriptive of current environments. With a more narrow range of workload parameters, it is possible to examine the proposed strategies in more detail and gain insight into the reasons for their success or failure.

1.1 Outline

Section 2 presents a model that predicts the speedup of a job as a function of its cluster size and two job parameters, the average and variance of parallelism. Section 3 presents out workload model, which specifies the distributions of those parameters. Section 4 describes the simulator we used to evaluate the scheduling strategies described in Section 5. Section 6 presents the results of these simulations. Section 7 summarizes our findings.

*University of California at Berkeley and San Diego Supercomputer Center, P.O. Box 85608, San Diego, CA 92186. Supported by Advanced Research Projects Agency/ITO, Distributed Object Computation Testbed, ARPA order No. D570, Issued by ESC/ENS under contract #F19628-96-C-0020. email: downey@sdsc.edu, http://www.sdsc.edu/~downey

2 The speedup model

One way to model a parallel program is with a parallelism profile, which is the distribution of available parallelism during the execution of the program [10]. Given a job's parallelism profile, it is possible to derive its speedup curve. It is common to summarize parallelism profiles with the two parameters A and V, which are the average and variance in parallelism, calculated in the usual way for distributions.

Our speedup model is based on the assumption that for given values of A and V we can approximate the speedup curve by constructing a hypothetical parallelism profile with the appropriate parameters and deriving the corresponding speedup curve. In a prior study, we showed that the resulting family of speedup curves captures the behavior of a variety of parallel scientific applications on a variety of architectures [3]. The model is based on two families of profiles, one for programs with low V, one for programs with high V.

2.1 Low variance model

Figure 1a shows a hypothetical parallelism profile for a program with low variance in parallelism. The degree of parallelism is A for all but some fraction σ of the duration ($0 \leq \sigma \leq 1$). The remaining time is divided between a sequential component and a high-parallelism component. The average parallelism of this profile is A; the variance of parallelism is $V = \sigma(A-1)^2$.

A program with this profile would have the following speedup as a function of the cluster size n:

$$S(n) = \begin{cases} \frac{An}{A+\sigma/2(n-1)} & 1 \leq n \leq A \\ \frac{An}{\sigma(A-1/2)+n(1-\sigma/2)} & A \leq n \leq 2A-1 \\ A & n \geq 2A-1 \end{cases} \quad (1)$$

2.2 High variance model

In the low variance model, σ cannot exceed 1, so the variance of parallelism, V, cannot exceed $(A-1)^2$. In this section, we propose an extended model in which σ can exceed 1 and the variance is unbounded. The two models can be combined naturally because when $\sigma = 1$, the two models are identical, and for both models $V = \sigma(A-1)^2$.

From the latter property we derive the semantic content of σ — it is approximately the square of the coefficient of variation of parallelism, CV^2. This approximation follows from the definition of coefficient of variation, $CV = \sqrt{V}/A$. Thus, CV^2 is $\sigma(A-1)^2/A^2$, which for large A is approximately σ.

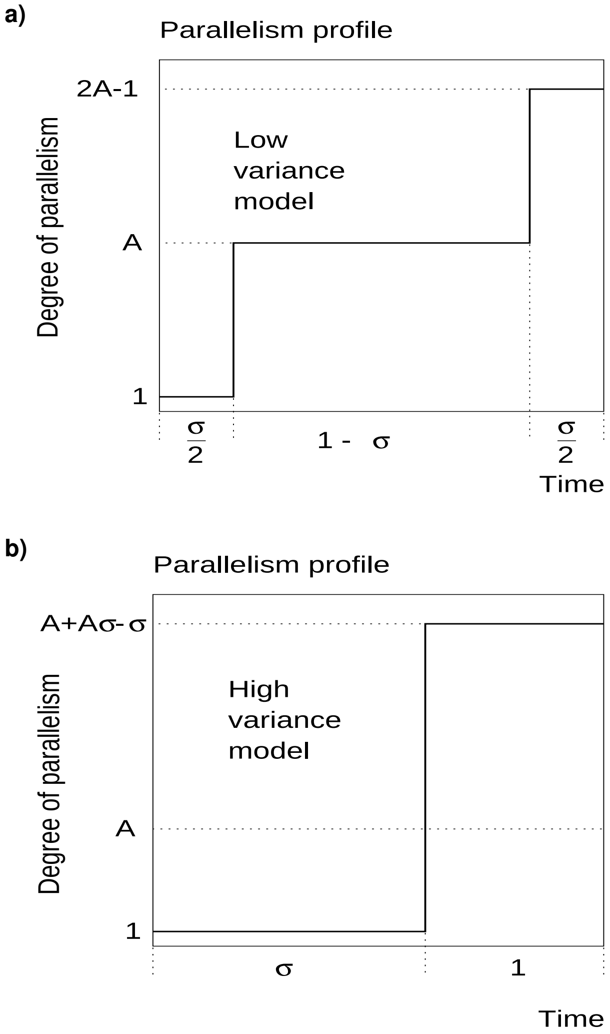

Figure 1. The hypothetical parallelism profiles we use to derive the speedup model.

Figure 1b shows a hypothetical parallelism profile for a program with high variance in parallelism. A program with this profile would have the following run time as a function of cluster size:

$$S(n) = \begin{cases} \frac{nA(\sigma+1)}{\sigma(n+A-1)+A} & 1 \leq n \leq A + A\sigma - \sigma \\ A & n \geq A + A\sigma - \sigma \end{cases} \quad (2)$$

Figure 2 shows speedup curves for a range of values of σ (and $A = 64$). When $\sigma = 0$ the curve matches the theoretical upper bound for speedup, bound at first by the hardware limit (linear speedup) and then by the software limit (the average parallelism A). As σ approaches infinity, the curve approaches the theoretical lower bound on speedup derived by Eager et al. [4]: $S_{min}(n) = An/(A+n-1)$.

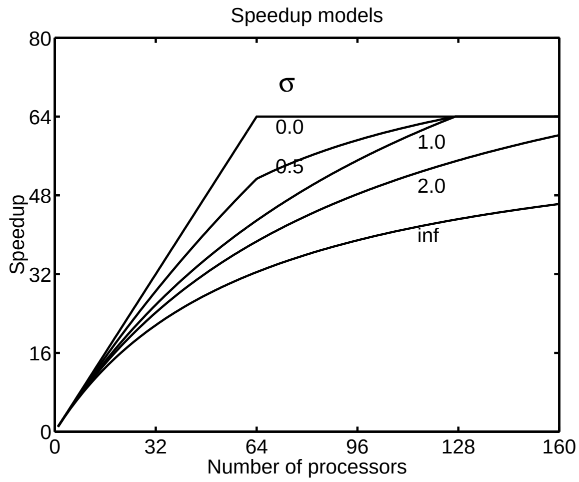

Figure 2. Speedup curves for various values of σ.

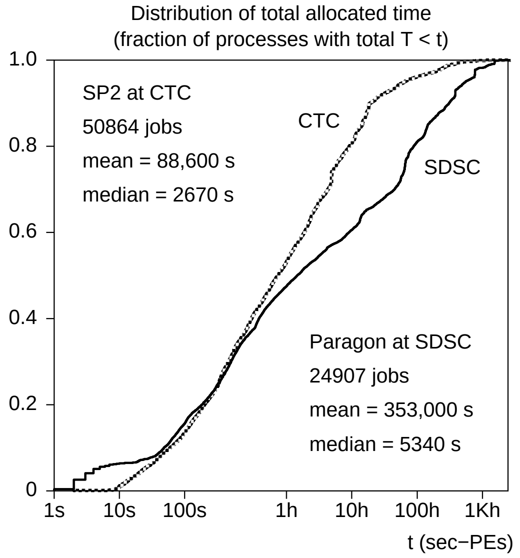

Figure 3. Distribution of total allocated time for the Paragon at SDSC the SP2 at CTC.

3 Distribution of parameters

To evaluate allocation strategies, we use a simulator based on an *abstract workload model*. On existing systems, we often collect statistics about actual (concrete) workloads; for example, we might know the duration and cluster size of each job. Observed workloads depend on the job mix, the properties of the hardware, and the behavior of the job scheduler. Thus, it may not be correct to use a concrete workload from one system to simulate and evaluate another. Our goal is to create an abstract workload that separates the characteristics of the job mix from the effect of the system.

It is particularly important to use an abstract workload model to test for starvation. If a system tends to provide poor service for certain jobs, then users will not submit jobs of that type. If we used observations from such a system to build a workload model, we might conclude that starvation is not a problem, since we would never see jobs starving. An abstract workload model that reflected users' intentions (as opposed to their observed behavior) would show the true effects of starvation.

In this section we use observations of the Intel Paragon at the San Diego Supercomputer Center and the IBM SP2 at the Cornell Theory Center to develop an abstract workload model.

3.1 Distribution of lifetimes

Ideally, we would like to know the distribution of L, the *sequential lifetime*, for a real workload. Sequential lifetime is the time a job would take on a single processor. If we knew L, A and σ, we could calculate the speedup, $S(n, A, \sigma)$, on n processors and the run time, L/S. But for most jobs we do not know L; often it is not even defined, because memory requirements prevent some jobs from running on a single processor. On the other hand, we do know the *total allocated time*, T, which is the product of wall clock lifetime and cluster size. For programs with linear speedup, T equals L, but for programs with sublinear speedups, T can be much larger than L.

Figure 3 shows the distribution of total allocated time for jobs from the Paragon at SDSC and the SP2 at CTC. On both machines, the distribution is approximately uniform (linear) in log space; thus, we call it a *uniform-log distribution*. We know of no theoretical reason that the distribution should have this shape, but it seems to be pervasive among batch workloads. We have observed similar distributions on the Cray C90 at SDSC, and other authors have reported similar distributions on other systems [5][12].

What is the relationship between the distribution of T (which we can observe) and the distribution of L (which we would like to know)? For some allocation policies, the two distributions might have different shapes; for example, if large jobs are given small clusters, then the distribution of T will have a longer tail than the distribution of L. On the other hand, all the strategies in this paper give more processors to large jobs; in this case, the two distributions will have the same general shape, although with different parameters. The simulations confirm this expectation. The distribution of L, which is an input to the simulator, is uniform-log. For all of the simulated strategies, the resulting distribution of T is also approximately uniform-log.

In our simulations, L is distributed between 7 seconds

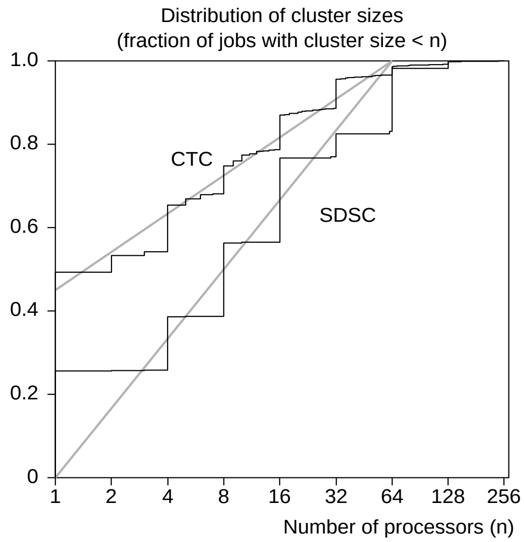

Figure 4. Distribution of cluster sizes for the workloads from SDSC and CTC. The gray lines show the distributions we used for the workload model.

and 45 hours. The median of this distribution is 18 minutes; the mean is 271 minutes. These parameters were chosen in order to make the average number of jobs per day consistent with the observed workloads. In other simulations, we have used different parameters, but within a wide range of values, they do not affect the relative performance of the strategies.

3.2 Distribution of average parallelism

To choose speedup curves for the workload model, we would like to know the parallelism profiles of real jobs. But the parallelism profile reflects *potential* parallelism, as if there were an unbounded number of processors available, and in general it is not possible to derive this information by observing the execution of a job. On the other hand, we may be able to infer from real workloads the distributions of the parallelism parameters A and V.

In the accounting data we have from SDSC and CTC, we do not have information about the average parallelism of jobs, but we do know the cluster size the user chose for each job. We hypothesize that these cluster sizes, in the aggregate, reflect the parallelism of the workload.

This hypothesis is based on a model of user behavior:

- In general, users care about their jobs' turnaround time; thus, they tend to allocate more processors to jobs with large potential speedups.
- On the other hand, users are allocated limited computing time; thus, they avoid allocating processors they cannot use efficiently.

In the aggregate, this user behavior results in cluster sizes that are roughly proportional to the average parallelism of each job. Thus, we expect the distribution of A to have the same shape as the distribution of cluster sizes.

Figure 4 shows this distribution for the workloads from SDSC and CTC. In both cases, most jobs have cluster sizes that are powers of two. Neither the Paragon nor the SP2 require power-of-two cluster sizes, but in both cases the interface to the queueing system suggests powers of two and few users have any reason to resist the combination of suggestion and habit. We believe that the step-wise pattern in the distribution of cluster sizes reflects this habit and not the true distribution of A. Thus for our workload model, we use the uniform-log distributions shown as gray lines in the figure.

The SDSC model, with parameters $A_{min} = 1$ and $A_{max} = 64$, fits the observed distribution well, although there are more sequential jobs in the observed workload than in the model.

The CTC distribution contains even more sequential jobs, most likely because their SP2 has more memory on each node than most workstations, and provides some software that is not available on workstations. Thus, many users submit sequential jobs to the SP2 that they would ordinarily run on workstations. To model this workload, we made 45% of the jobs sequential, and chose the average parallelism for the others from a uniform-log distribution. The top gray line in the figure shows this distribution.

The results in Section 6 are based on the SDSC model. Because the CTC model has less available parallelism, it yields much longer residence times (50 to 90% longer) but has no effect on the relative performance of the strategies.

Although the observed cluster sizes suggest the shape of the distribution of A, they do not necessarily reflect the range. One possibility is that users, trying to minimize the accounting cost of their jobs, choose cluster sizes significantly less than A. If so, then our model would underestimate the available parallelism in the workload. Thus, we also simulated a distribution of A with $A_{max} = 4N = 256$. In simulation, this increase in available parallelism reduces average residence times by 10 to 20% (depending on load), but has no effect on the relative performance of the strategies.

3.3 Distribution of variance in parallelism

In general there is no way to measure the variance in potential parallelism of existing codes explicitly. In previous work, we proposed a way to infer this value from observed speedup curves [3]. To test this technique, we collected speedup curves for a variety of scientific applications running on a variety of parallel computers. We found that σ, which approximates the coefficient of variation of parallelism, was typically in the range 0 to 2, with occasional higher values.

Although these observations suggest a range of values for σ, they do not tell us its distribution in a real workload. For this study, we use a uniform distribution between 0 and 2. The specific shape of this distribution has little effect on the relative performance of the allocation strategies. For example, we also simulated a high-variance workload (all jobs have $\sigma = 2$) and a linear-speedup workload ($\sigma = 0$). Under the high-variance workload, the residence times for all strategies increase by 20%, but their relative performance is the same. For the linear-speedup workload, residence times are consistently 20% lower than under the uniform workload.

4 Simulations

To evaluate scheduling strategies for parallel applications, we used the models in the previous section to generate workloads, and then used a simulator to construct schedules for each workload according to each strategy. We compare these schedules using several summary statistics as performance metrics (see Section 4.1).

Our simulations try to capture the daily cycle that has been observed in several supercomputing environments [5] [12]:

- In the early morning there are few arrivals, system utilization is at its lowest, and queue lengths are short.
- During the day, the arrival rate exceeds the departure rate and jobs accumulate in queue. System utilization is highest late in the day.
- In the evening, the arrival rate falls but the utilization stays high as the jobs in queue begin execution.

To model these variations, we divided each simulated day into two 12-hour phases: during the day-time phase, jobs arrive according to a Poisson process and either begin execution or join the queue. During the night-time phase, no new jobs arrive, but the existing jobs continue to run until all queued jobs have been scheduled.

Figure 5 shows one of the schedules generated by the simulator. Hour 0, at the top of the page, is roughly 8am; time proceeds down the page. The length of each job indicates its duration. The width of each job is proportional to its cluster size. The jobs have been arranged for visual clarity rather than according to the topology of their allocations; thus, the width of the figure is greater than the number of processors in the system, and empty spaces do not necessarily indicate idle processors. Jobs which were queued have a black tail representing their queue time. The night-time interval, during which there are no additional arrivals, is shaded.

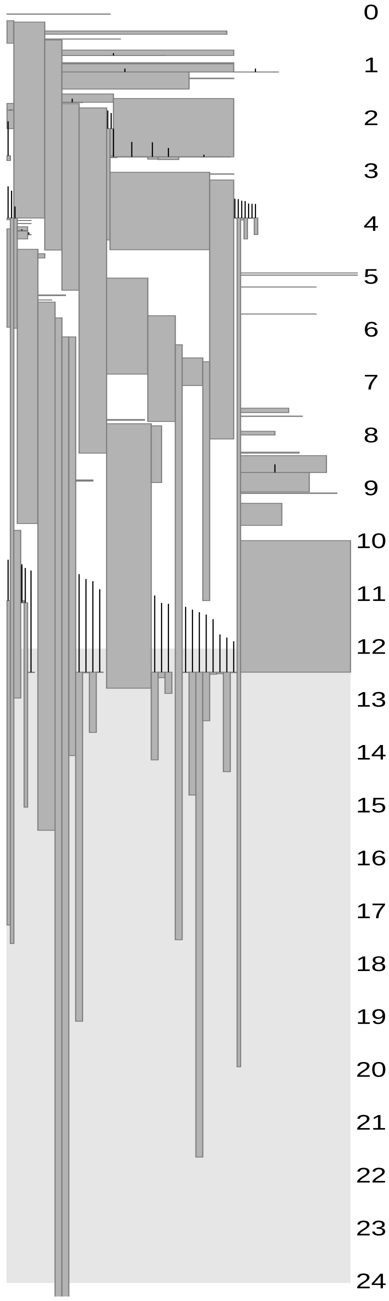

Figure 5. A typical daily schedule, with offered load $\rho = 0.6$, using the ASP strategy (see Section 5.4).

We choose the day-time arrival rate in order to achieve a specified offered load, ρ. We define the offered load as the total sequential load divided by the processing capacity of the system: $\rho = \lambda \cdot E[L]/N$, where λ is the arrival rate (in jobs per second), $E[L]$ is the average sequential lifetime (271 minutes in our simulations), and N is the number of processors in the system (64 in our simulations).

4.1 Metrics

As jobs enter and leave the system, we collect the following statistics:

Utilization: Utilization is efficiency averaged over processors and time. Efficiency is S/n, the ratio of speedup to cluster size. The efficiency of an idle processor is zero.

Average turnaround time: Turnaround time is the time between the arrival and completion of a job; i.e. the sum of its queue time and its run time.

Average slowdown: Slowdown is the ratio of turnaround time to the shortest possible turnaround time, as if the job had run on a dedicated machine. In other words,

$$\text{slowdown} = \frac{\text{queue time} + R(n)}{R(N)} \tag{3}$$

where $R(n)$ is the run time on the allocated cluster size, n, and $R(N)$ is the hypothetical run time on all N processors. Slowdown is a useful performance metric because it gives equal weight to all jobs regardless of length, whereas average turnaround time tends to be dominated by long jobs. Also, slowdown may better represent users' perception of system performance, since it measures delays relative to job duration. For example, a long queue time is more acceptable for a long job than for a short one. Slowdown captures this implicit cost function.

5 Allocation Strategies

Scheduling strategies consist of a *queueing strategy* that chooses which queued job to run and an *allocation strategy* that chooses how many processors to allocate to each job.

The queueing strategy we consider is first-in-first-out (FIFO). The advantages of FIFO are predictability (it is easier to estimate when a queued job will begin execution) and avoidance of starvation (there is no danger of stranding a large job in queue while smaller, later arrivals run). The disadvantages are possibly lower utilization (a job at the head of the queue might leave processors idle waiting for a large cluster) and large slowdowns (giving priority to short jobs would reduce the average slowdown over all jobs).

The following sections describe the allocation strategies we consider. None of these strategies is work-conserving, meaning that in some circumstances processors may be left idle that could have been allocated to a job. Rosti et al. [7] show that non-work-conserving strategies are best if the workload contains jobs with limited parallelism, if the arrival process is bursty, or if the distribution of job lifetimes is highly variable. The workloads we observed meet all these criteria.

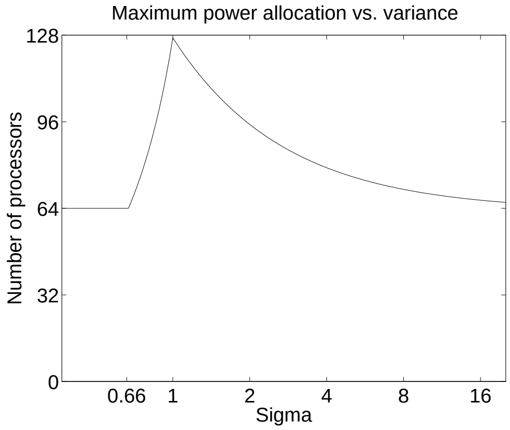

Figure 6. The processor working set, pws, for a range of values of σ ($A = 64$).

5.1 PWS

For most jobs, speedup is an increasing function of cluster size and efficiency is a decreasing function. Thus, the product of speedup and efficiency, power, has a maximum point called the *processor working set*, or pws.

Eager et al. suggest that pws is an "appropriate" allocation for a parallel job. Ghosal et al. [6] show that allocation can be formulated as a cost-benefit tradeoff in which the pws maximizes the benefit (speedup) per unit of cost (additional processors).

For our hypothesized speedup model (Eqns. 1 and 2) we can find the point of maximum power analytically:

$$pws = \begin{cases} A & \sigma \leq 2A/(3A-1) \\ \frac{\sigma(A-1/2)}{1-\sigma/2} & \sigma \geq 2A/(3A-1) \\ (A\sigma + A - \sigma)/\sigma & \sigma \geq 1 \end{cases} \tag{4}$$

Figure 6 shows pws for a range of values of σ (with A fixed at 64). For small values of σ, the processor working set is A, which is in accord with the heuristic that the number of processors allocated to a job should be equal to its average parallelism.

But as σ approaches 1, pws increases quickly to $2A - 1$. For $\sigma > 1$, pws decreases and approaches $A - 1$ asymptotically. This result is surprising because it violates the intuition that the optimal allocation for a job should decrease as the variance in parallelism increases, which is one of the assumptions of Sevcik's allocation strategy (Section 5.3).

Eager et al. prove that for any parallelism profile, pws is bounded by $A/2$ and $2A - 1$. Within our speedup model, pws never approaches the lower bound, but it reaches the upper bound when $\sigma = 1$.

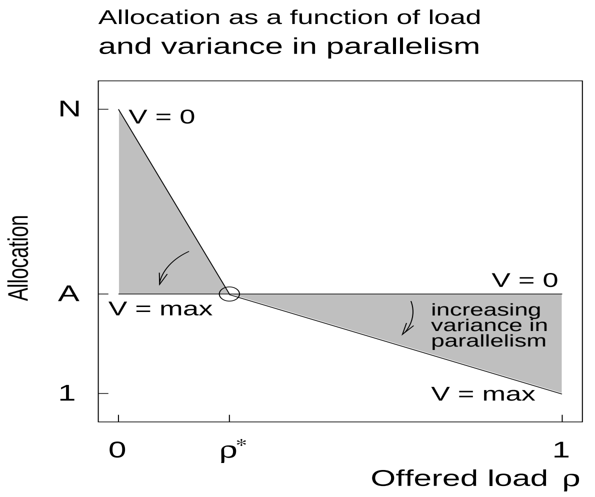

Figure 7. The relationship proposed by Sevcik between average parallelism, A, variance of parallelism, V, system load, ρ and cluster size.

5.2 AVG and MAX

Several authors have proposed the idea that a job should be allocated a number of processors equal to A, the average parallelism. Eager et al. suggest that this strategy should do well because A is always within a factor of two of pws. Interestingly, we find that this strategy, AVG, performs *better* than PWS, which it is supposed to approximate.

Another strategy suggested by the speedup curves in Figure 2 is MAX, which allocates enough processors to achieve the maximum speedup for the job; in other words, it allocates the minimum n such that $S(n) = A$. For our speedup model:

$$max = \begin{cases} A & \sigma = 0 \\ 2A - 1 & 0 < \sigma \leq 1 \\ A + A\sigma - \sigma & \sigma \geq 1 \end{cases} \qquad (5)$$

5.3 Sevcik's allocation strategy

Sevcik [10] has proposed an allocation strategy that chooses cluster sizes as a function of the average parallelism of the job, A, the variance in parallelism, V, and the offered load, ρ. This strategy is based on the intuition that cluster sizes should be large when A is large, but should get smaller as V or ρ increases. Sevcik calls this strategy A+; we call our variation of it SEV. The primary distinction is that we use σ as a measure of variance where Sevcik uses V. Because σ approximates the coefficient of variation, it is parameterless, whereas V increases with A.

Figure 7 shows Sevcik's strategy graphically: (1) for very low loads, cluster sizes vary from N for jobs with low

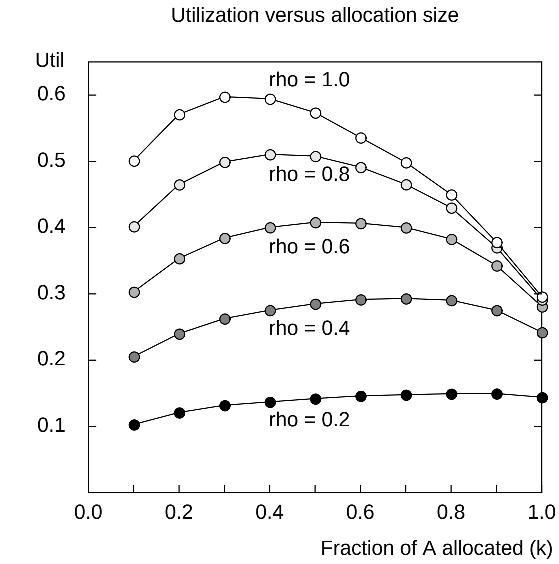

Figure 8. Each curve shows the average utilization of a simulated 64-processor system with offered load ρ. The parameter k is the fraction of A processors assigned to each job. Each data point is the average of 60 simulated days.

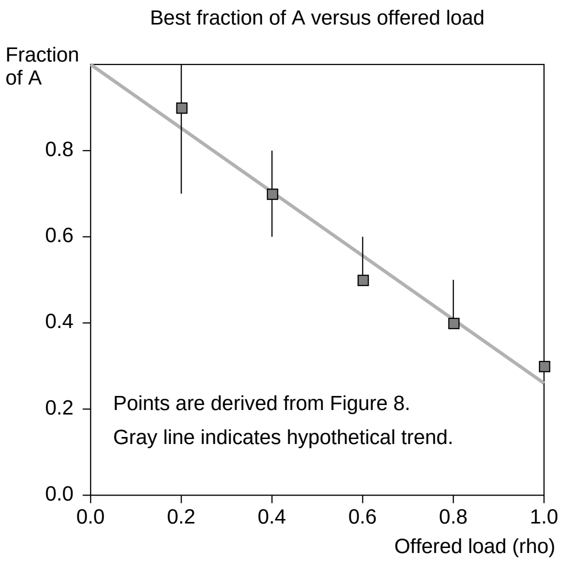

Figure 9. For each value of ρ we have chosen the value of k that maximizes utilization; the vertical error bars show the range of values where this maximum may fall. The gray line extrapolates until $k = 1$, which is approximately at $\rho = 0$.

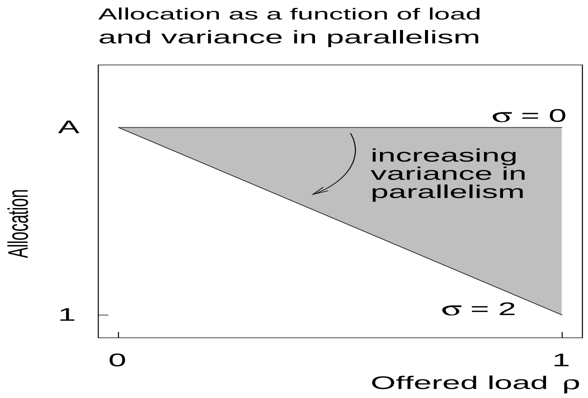

Figure 10. A simplified version of Sevcik's strategy with $\rho^* = 0$.

V to A for jobs with high V, (2) at some moderate load ρ^*, all jobs are allocated A processors, and (3) for high loads, cluster sizes vary from A for jobs with low V, to 1 for jobs with high V.

The value of ρ^* is the load at which the optimal allocation per process is A. Sevcik chooses the value of ρ^* based on a queueing model with an exponential distribution of job lifetimes ($CV = 1$); for this workload ρ^* is roughly 0.25. As CV increases, we expect ρ^* to decrease, reducing cluster sizes in order to lessen the danger of assigning a large cluster to a very long job.

To find the value of ρ^* for our workload, we ran our simulator with a range of offered loads ρ and with a range of values of a parameter, k, which is the fraction of A processors allocated to each job. Thus, the cluster size for each job is kA. We expect ρ^* to be the value of ρ for which the optimal allocation is A; in other words, the offered load for which $k = 1$ yields the best performance.

Figure 8 shows the average utilization of a simulated 64-processor system with values of $\rho = 0.2$, 0.4, 0.6, 0.8 and 1.0, and values of k between 0.1 and 1.0. Figure 9 shows the optimal value of k chosen for each value of ρ. The gray line indicates the hypothetical linear trend.

We conclude: (1) Sevcik's assumption — that the optimal allocation size should decrease linearly as the load increases — is at least approximately correct, and (2) the value of ρ^* is at or near zero. It is difficult to make the latter claim precise, since for low loads ($\rho < 0.4$) the value of the parameter has almost no effect on performance, and hence "optimal value" has little meaning.

With $\rho^* = 0$, this allocation strategy can be simplified as in Fig. 10. As discussed above, we use σ in place of V. In accordance with the workload model, the maximum value of σ is 2.

5.4 ASP

The strategies we have described so far have been based on the idea that the optimal allocation for a job depends on the characteristics of the program. For purposes of comparison, we also consider a form of *adaptive static partitioning* (ASP) based on the notion that it is preferable to have as many jobs running as possible, rather than waiting in queue, even if they run on small cluster sizes.

Thus at the time of an arrival or completion, the ASP strategy allocates free processors to the jobs in queue, starting at the head of the queue and continuing in round-robin fashion, with the constraint that no job is given more processors than it can use (Eqn. 5).

Chiang et al. [1] report that this strategy does well for workloads in which parallelism does not vary greatly from job to job. For workloads with large variance in A, though, their results are consistent with ours — the performance of ASP is relatively poor. We explain this result in more detail in Section 6.1.

6 Results

AVG, MAX, PWS and SEV each choose cluster sizes for jobs based on their speedup characteristics. It is not apparent, though, whether the derived sizes should be considered the only acceptable allocation, or an upper bound, or even a lower bound.

In other words, when a job arrives at the head of the queue and the number of free processors is less than the desired cluster size, it is not clear whether the job should allocate the available (small) cluster and run immediately or wait in queue until more processors are freed. In a prior study, we evaluated two variations of each strategy [2]:

Greedy strategies: The job at the head of the queue must allocate any available processors and begin execution immediately.

Stubborn strategies: Jobs wait in queue until their derived cluster size is available.

We found that greedy strategies are better than stubborn strategies by all metrics. We also evaluated hybrid strategies that can allocate fewer than the ideal number of processors (greedy), but which nevertheless impose some lower bound on cluster sizes (stubborn). We found that the pure greedy strategy is better than hybrid strategies with even moderate minimum cluster sizes.

In this paper we consider only greedy strategies; thus, we will assume that there is no lower bound on the cluster size for a job. Of course, many real jobs have memory requirements that prevent them from running on a small cluster. Nevertheless, we have found that moderate memory

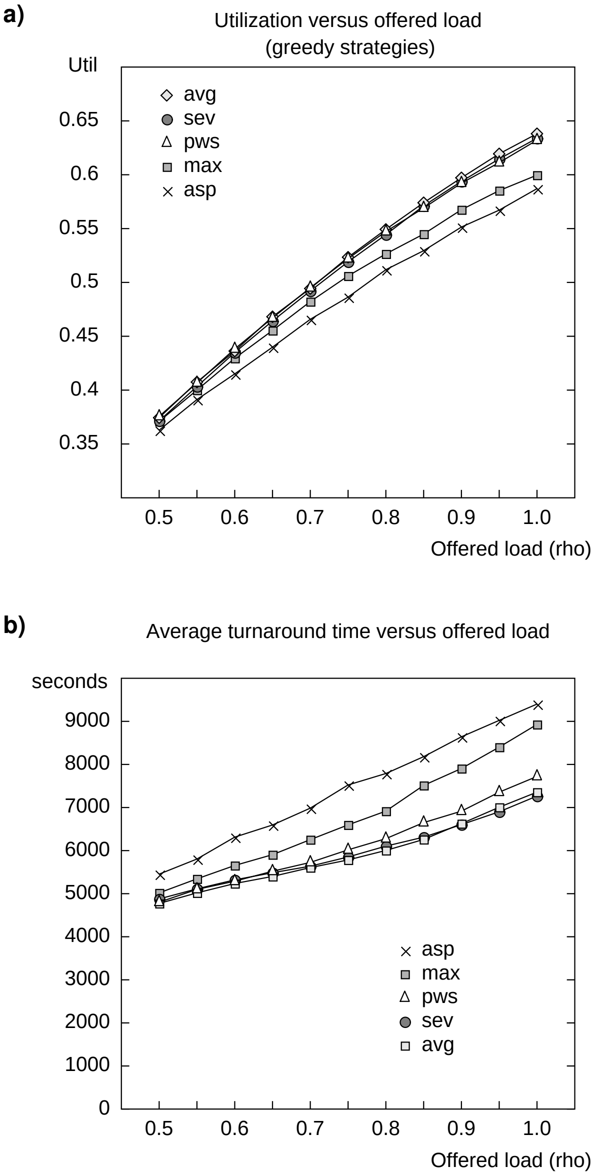

Figure 11. Comparison of the allocation strategies (a) by utilization and (b) by average turnaround time.

requirements do not significantly impair the strategies we evaluate. Setia [8] has examined this effect in more detail and finds that wide-ranging minimum cluster sizes do affect performance. Thus, an important topic for future work is a model of the memory requirements (minimum cluster sizes) of real workloads.

Figure 11 shows the performance of the allocation strategies by utilization (averaged over 120 simulated days) and turnaround time (averaged over approximately 15000 jobs). The size of the data markers (squares, circles, etc.) is approximately two standard errors. Thus, non-overlapping data markers indicate statistically significant differences.

For loads higher than 0.7, the performance of PWS is slightly (but significantly) worse than that of AVG, by both metrics. This is surprising because according to the cost-benefit analysis of Ghosal et al., PWS should choose cluster sizes that are optimal. The reason this analysis fails is that it assumes that the opportunity cost of a processor does not depend on the state of the system. In fact, the opportunity cost increases with system load, as it becomes more likely that there is another job that can use an additional processor more efficiently.

6.1 A closer look

In order to investigate the relationships among various performance metrics, we set $\rho = 0.75$ and examine the proposed strategies in more detail. Table 1 shows several summary statistics for each strategy[1].

Strategies that allocate large cluster sizes tend to have high measured loads — they leave fewer idle processors. But load only measures how many processors are busy, not how effectively they are being used. Since utilization also considers the efficiency of running jobs, it is a better indicator of performance than load.

It is often observed that there is a conflict between maximizing system utilization and minimizing average turnaround time. For the strategies we considered, though, the two metrics are consistent; whatever strategy resulted in the highest utilization also yielded the lowest average turnaround times. On the other hand, slowdown and turnaround time are not always consistent; for example, SEV suffers somewhat longer turnaround times than AVG, but yields much better slowdowns. Since slowdown reflects users' perception of system performance, it might be preferable to choose a strategy that achieves minimal slowdowns, even with (moderately) longer turnaround times.

There is a clear relationship between average cluster size and average queue time. The strategies that allocate the fewest processors per job have the shortest queue times. But the tradeoff is that smaller cluster sizes result in longer run times. The proposed strategies operate at different points along this tradeoff:

- SEV allocates the smallest cluster sizes, resulting in the shortest queue times, but incurs the longest run times.

- AVG and PWS reduce run times by allocating more processors per job, but this benefit is almost exactly

[1] We report the 90th percentile of slowdown rather that its mean because the distribution of slowdowns is long-tailed, and for such distributions order statistics (median and other percentiles) are more robust, and hence more meaningful, than moment statistics (mean, variance, etc.).

Table 1. Performance metrics for the five allocation strategies.

	measured load average	utilization	average turnaround time	average queue time	average cluster size (CV)	90th percentile slowdown
SEV	.64	.52	5858	204	7.8 (1.04)	11.6
AVG	.70	.52	5782	372	9.4 (1.07)	35.3
PWS	.73	.52	6017	566	10.2 (1.10)	77.8
MAX	.81	.51	6597	1115	10.7 (1.11)	249
ASP	.77	.49	7510	402	9.9 (1.24)	63.6

balanced by greater queue times; thus, the utilization of the three strategies is the same and the average turnaround times are similar.

- On the other hand, MAX allocates so many processors per job that the additional queue time overwhelms the saved run time. Utilization for this strategy is lower and turnaround times are higher.

SEV yields by far the lowest slowdowns, in part because it takes load into account explicitly and reduces cluster sizes when load is high. The other strategies achieve a similar effect implicitly, by using the number of free processors as a proxy for the current load, but this implicit load-sensitivity is less precise — these methods sometimes allocate too many processors during a momentary lull. Figure 5 provides an example. At Hour 10, a large job arrives during a lull and allocates a large fraction of the machine. While that job runs (for over two hours) 16 smaller jobs arrive and wait in queue for the large job to complete.

Of the strategies we consider, only SEV would avoid this mistake by restricting the cluster size of the large job in anticipation of future arrivals. Of course, the disadvantage of SEV is that it is necessary to know the offered load a priori, or estimate it dynamically. Our simulation of this strategy assumes optimistically that the offered load is known exactly.

6.2 ASP-max

We have discussed how a momentary lull causes some strategies to allocate too many processors, but there is also the danger that a momentary surge in queue length will cause the system to allocate too few processors. Most of the strategies avoid this type of error by allowing the job at the head of the queue to choose a cluster size regardless of queue length. ASP is the only strategy that takes queue length into account, and it suffers for it.

The reason ASP performs relatively poorly is that it is most likely to make extreme allocations (very large or very small) during short-term variations in load (lulls and surges). To support this claim we examined the variance in cluster size for each strategy. As expected, ASP has a higher CV of cluster size than any of the other strategies (1.24 compared to a range of 1.04 to 1.11). We conclude that the success of a scheduling strategy lies in its ability to adjust to long-term changes in load (as in the daily cycle we model) without over-reacting to short-term variations.

Chiang et al. [1] suggest an improvement on ASP, called ASP-max, that imposes a limit on the number of processors a job can allocate. In their study, this strategy performs significantly better than ASP. There are several problems with this strategy, though. The first is that there is no obvious way to choose the maximum cluster size. We ran simulations with this limit set to 32, 16, 8 and 4 processors (on a 64-node system). The best choice depends on system load; for loads less than $\rho = 0.8$, it is near 8; for higher loads, it is near 4. A second problem is that users are likely to object to limits as low as 10% of the system size. What is the point of a massively-parallel computer if even large, highly-parallel jobs are allowed only a few processors? The final problem is that even with the best choice of the parameter, the performance of ASP-max is only a little better than that of MAX, and still worse than the other strategies. Furthermore, to make the comparison fair, we should add a hand-tuned parameter to the other strategies. Their performance would probably improve, too.

6.3 Sensitivity to variance

Although most of the strategies use σ to choose cluster sizes, they use this information in different ways. Figure 12 shows the relationship between cluster size and σ for each strategy (with A fixed). The two strategies that are most sensitive to variance are SEV, which decreases cluster sizes linearly as σ increases, and PWS, which has the counter-intuitive relationship derived in Section 5.1.

In this section, we investigate whether the performance of the strategies depends on these particular relationships, or whether similar, variance-insensitive strategies (3/2 AVG and Simplified SEV) might do as well. The insensitive

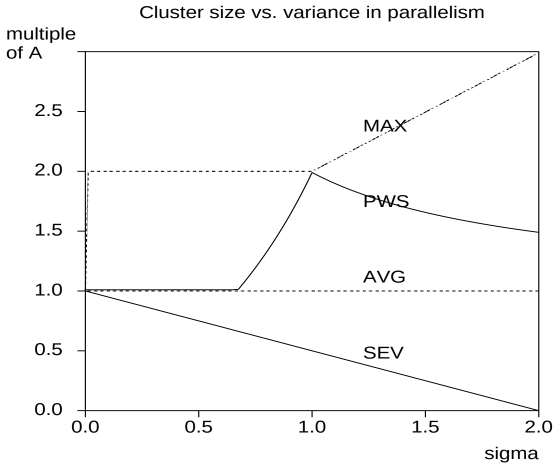

Figure 12. The relationship between cluster size and variance in parallelism (σ) for each of the proposed strategies.

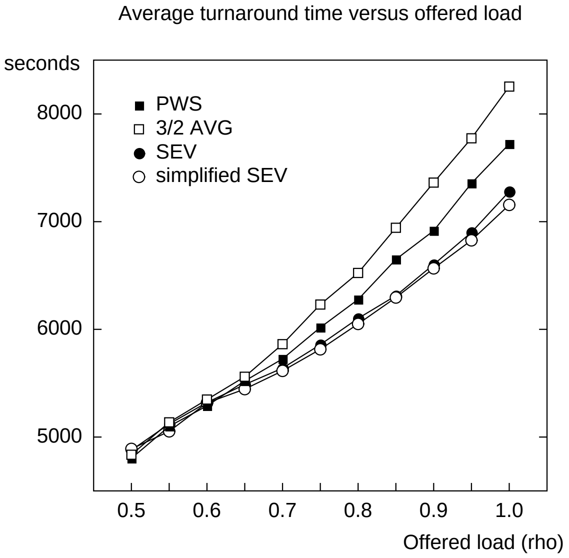

Figure 13. Turnaround time as a function of offered load, comparing two of the proposed strategies with variance-insensitive versions.

strategies are designed to allocate (on average) the same number of processors as their sensitive counterparts: under 3/2 AVG the maximum cluster size for each job is $3/2A$; under Simplified SEV the maximum cluster size is calculated by Sevcik's method, except that for all jobs σ is considered to be 1. In simulation, the average cluster sizes for the sensitive and insensitive strategies are almost identical.

Figure 13 shows that the average turnaround time for Simplified SEV is not significantly different from that of SEV; in other words, variance-sensitivity has no (statistically significant) effect on the performance of SEV. This contradicts one of the underlying assumptions of Sevcik's method, that cluster size should decrease as variance increases.

On the other hand, the strange relationship between σ and cluster size used by PWS *does* have a significant impact on performance. The variance-insensitive version of this strategy does significantly worse by all metrics. Although this result is surprising, it has no practical importance, since AVG, which is variance-insensitive, performs better than even the variance-sensitive version of PWS. Thus we conclude that knowing variance in parallelism is not useful for allocation.

7 Conclusions

- One of the strategies recommended in other studies (ASP) did not perform well for our workload. We show that this policy is too sensitive to short-term variations in system load. It performs worse than strategies that ignore queue length when choosing cluster sizes.

- The program characteristics we examined, average and variance of parallelism, are useful for choosing the upper bound on cluster size (and thereby imposing a lower bound on efficiency). We found, though, that strategies that considered variance of parallelism were no better than those that considered only average parallelism.

- The processor working set, or "knee" of the speedup curve, is not an optimal allocation.

- It is not necessary to set maximum cluster sizes precisely. There is a tradeoff between large clusters/long queues and small clusters/short queues, but within a wide range, system performance (measured by average turnaround time) does not vary greatly.

- Of several strategies with equivalent turnaround times, Sevcik's strategy, which uses a priori knowledge about system load to restrict cluster sizes, yielded the lowest slowdowns. Depending on what metric matters most to users, this strategy might be the best choice.

- One of Sevcik's underlying assumptions — that cluster sizes should decrease linearly as load increases — has been validated. The other underlying assumption — that jobs with a more variable parallelism profile should allocate fewer processors — has been contradicted.

- Lifetimes for batch jobs on supercomputers are distributed uniformly in log space. The uniform-log model is useful for summarizing these distributions and generating stochastic workloads. The observed distributions have coefficients of variation between 2 and 4.

7.1 Future work

When a job arrives at the head of the queue, there is a conflict between the system, which would like the job to begin execution as soon as possible, and the job, which might enjoy a shorter turnaround time by waiting until a larger cluster size is available. In continuing work, we are investigating the following questions:

- How much do individual jobs benefit by waiting for larger clusters?
- How much do these stubborn jobs hurt the system as a whole by increasing the queue times of other jobs?
- In a real system, is it tenable for the scheduler to make decisions that are contrary to the immediate interests of users, or will users subvert such a system?

Our goal is to find an allocation policy that maintains acceptable overall performance without creating incentives for users to manipulate the system for their own benefit.

Acknowledgements

I would like to thank George Kremenek at SDSC and Steven Hotovy at CTC for their help collecting accounting data. I am also grateful to Reagan Moore at SDSC, Jennifer Schopf and Rich Wolski at U.C. San Diego, and Emilia Rosti at the University of Milan for reading drafts of this paper and providing valuable comments.

References

[1] 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 *Proceedings of the 1994 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, 1994.

[2] A. B. Downey. A parallel workload model and its implications for processor allocation. Technical Report CSD-96-922, University of California at Berkeley, 1996.

[3] A. B. Downey. A model for speedup of parallel programs. Technical Report CSD-97-933, University of California at Berkeley, 1997.

[4] D. L. Eager, J. Zahorjan, and E. L. Lazowska. Speedup versus efficiency in parallel systems. *IEEE Transactions on Computers*, 38(3):408–423, March 1989.

[5] 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, Springer-Verlag LNCS Vol 949*, pages 337–360, April 1995.

[6] D. Ghosal, G. Serazzi, and S. K. Tripathi. The processor working set and its use in scheduling multiprocessor systems. *IEEE Transactions on Software Engineering*, 17(5):443–453, May 1991.

[7] E. Rosti, E. Smirni, G. Serazzi, and L. W. Dowdy. Analysis of non-work-conserving processor partitioning policies. In *Job Scheduling Strategies for Parallel Processing, Springer-Verlag LNCS Vol 949*, pages 165–181, April 1995.

[8] S. K. Setia. The interaction between memory allocation and adaptive partitioning in message-passing multicomputers. In *Job Scheduling Strategies for Parallel Processing, Springer-Verlag LNCS Vol 949*, pages 146–164, April 1995.

[9] S. K. Setia and S. K. Tripathi. A comparative analysis of static processor partitioning policies for parallel computers. In *Proceedings of the Internationsal Workshop on Modeling and Simulation of Computer and Telecommunications Systems (MASCOTS)*, January 1993.

[10] K. C. Sevcik. Characterizations of parallelism in applications and their use in scheduling. *Performance Evaluation Review*, 17(1):171–180, May 1989.

[11] E. Smirni, E. Rosti, L. W. Dowdy, and G. Serazzi. Evaluation of multiprocessor allocation policies. Technical report, Vanderbilt University, 1993.

[12] K. Windisch, V. Lo, D. Feitelson, B. Nitzberg, and R. Moore. A comparison of workload traces from two production parallel machines. In *6th Symposium on the Frontiers of Massively Parallel Computation*, 1996.