Non-clairvoyant Multiprocessor Scheduling of Jobs with Changing Execution Characteristics

EXTENDED ABSTRACT*

Jeff Edmonds† Donald D. Chinn† Tim Brecht† Xiaotie Deng †

Department of Computer Science

York University

North York, ONT M3J 1P3

Abstract

A multiprocessor system is unlikely to have access to information about the execution characteristics of the jobs it is to schedule. In this work, we are interested in scheduling algorithms for batch jobs that require no such knowledge (such algorithms are called *non-clairvoyant*).

Preemptive scheduling (i.e., redistribution of processors) is important to reduce mean response time in multiprocessor systems, especially in the widely available network of workstations. Preemption is a method to adapt to the uncertain and changing nature of jobs and workloads. Unfortunately, preemption may incur large overheads if it is applied frequently. To account for the cost preemptions, we consider a number of simple scheduling algorithms classified by the number of preemptions they are allowed, ranging from none to an infinite number.

The Equi-partition algorithm [18], which partitions the processors evenly between the uncompleted jobs, is an example of a simple scheduler that is non-clairvoyant and preempts only when jobs complete. Motwani *et al.* [15] show that the mean response time of jobs is within two of optimal for fully parallelizable jobs. Since parallel programs can have a wide variety of execution characteristics in practice, we consider a number of classifications of jobs according to how well they are able to utilize processors. Moreover a job may have both sequential and parallel phases in its computation. Hence, we allow jobs to have multiple phases, each of which may have different execution characteristics.

For each of these preemptive models and for each of these job classifications, we provide asymptotically tight bounds on the mean response time of non-clairvoyant scheduling algorithms. For example, we show that in the worst case, the mean job response time obtained with Equi-partition is $2 + \sqrt{3} \approx 3.74$ times that obtained with an optimal algorithm (which may preempt processors any number of times and may use job characteristics to make its scheduling decisions) for a large class of jobs, characterized by multiple phases of arbitrary nondecreasing and sublinear speedup functions.

1 Introduction

The study of parallel and distributed computer system performance is generally more difficult than that of uniprocessor systems. One important property of general purpose computer systems is the unknown nature of job execution. For uniprocessor systems, preemptive scheduling strategies, such as Round Robin, use no information about job characteristics. The cost of preemption can be amortized by giving jobs remaining in the system a quantum of processor time proportional to how long they have been in the system [15]. In multiprocessor systems a similar preemptive algorithm, dynamic Equi-partition (DEQ), can be used when preemption costs are not prohibitively large[1], to achieve similar performance [2, 3]. However, overheads incurred due to preemptive scheduling algorithms may be much larger in parallel and distributed systems, and especially in the networks of workstations model. When the overhead is prohibitive, then results from theoretical studies on non-preemptive execution of parallel jobs may be more relevant [10, 19, 20, 16], but those results would require complete information of jobs in the system.

In this work, we consider the scheduling problem on a p processor system where n jobs all arrive at time 0 and no other jobs arrive thereafter. We present a new job model that applies to a large class of parallel jobs, including those job models discussed in Turek *et al.* [19]. Our metric of performance is the mean response time of the jobs.

*This paper appeared in *Proceedings of the Twenty-ninth Annual ACM Symposium on the Theory of Computing*, El Paso, TX, pages 120-129, 1997.

†{jeff, dci, brecht, deng}@cs.yorku.ca. Edmonds, Brecht, and Deng are supported by NSERC Canada. Chinn was supported in part by NSERC as a Postdoctoral Fellow at York University. Chinn's current address: One Microsoft Way, Redmond, WA 98052; dchinn@microsoft.com. Deng can also be reached at: Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong.

[1] The approach of Equi-partition was first introduced to parallel scheduling by Tucker and Gupta as a *process control* policy [18], and later modified to the preemptive scheduling environment as DEQ by Zahorjan and McCann [21].

To account for the cost of preemptions, we explore a range of schedulers classified by the number of preemptions they make. We also explore job classes, categorized by their execution characteristics. These job classes can be viewed as representing the amount of information a scheduler knows about the jobs it schedules. See Figure 2 for a summary of our results.

We study the simple Equi-partition algorithm, for which an equal number of processors is assigned to every job. We show that this algorithm (which performs at most n preemptions) achieves a performance within $2 + \sqrt{3}$ times the optimum schedule (which may preempt processors any number of times and may use job characteristics to make its scheduling decisions) when the jobs are from a fairly large class. The number of preemptions in Equi-partition can be further reduced to $\log_2 n$ with an extra constant multiplicative factor of two loss in performance.

This result is perhaps most interesting when compared with the existing bound (4 times optimal) [3] for the dynamic Equi-partition algorithm (DEQ). Our new bound for Equi-partition is tighter than the previous bound for DEQ, even though Equi-partition uses significantly fewer preemptions and does not use any job execution characteristics, whereas DEQ does. A possible interpretation of this result is that it provides theoretical evidence that algorithms that do not use information about job execution characteristics to frequently reallocate processors may not have to pay excessively large performance penalties (in terms of mean job response times).

The network of workstations model is an extreme case of distributed memory systems, for which frequent preemptions of executing jobs and reassignments of processors are costly. Our results show that for a large class of parallel jobs, provably near-optimal mean response time can be achieved with few reassignments of processors. Of course, much more research is required to make this theoretical understanding useful in a practical setting. In fact, performance in such systems has been already studied using simulation, experimental, and queuing theoretical approaches [1, 8, 9, 12, 13, 14, 18, 21]. In this perspective, our research constitutes a theoretical confirmation of these efforts.

1.1 Modeling Job Execution

In our model, all jobs arrive at time zero (batch). It would be more general to allow jobs to arrive at anytime. However, this makes the scheduling problem much more difficult and is left as an open problem.

Before a scheduler can attempt to find the best schedule, a measure of the success of a schedule needs to be defined. The two measures used most frequently are the final completion time of all the jobs (makespan) and the mean response time of the jobs (average completion time). Other measures take into account the level of fairness given to each individual job. We use the mean response time in this paper.

The parallelism profile of a job, defined as the number of processors an application is capable of using at any point in time during its execution, was introduced by Kumar [7]. More generally, a speedup function, Γ, specifies the rate at which work is completed as a function of the number of processors allocated to it. Since parallel programs can have a wide variety of execution characteristics in practice, we consider a number of different classifications of jobs according to how well they are able to utilize processors: sequential, fully parallelizable, sublinear, superlinear, nondecreasing, etc. To be more general, we allow jobs to have multiple phases, each of which is defined by an amount of work and a speedup function.

Most scheduling results heavily depend on the scheduler knowing the characteristics of the jobs being scheduled. Hence, to various degrees of success, compilers and run-time systems attempt to give hints to the scheduler. We, however, consider non-clairvoyant schedulers that have no information about the jobs other than the number of uncompleted jobs in the system. Our results show that even without such compiler or run-time hints and without many preemptions, schedulers can perform well.

The schedulers in some results are computationally intensive. Finding the optimal may be NP-complete. Even when polynomial, the algorithm may (e.g., involving finding a perfect matching) not be practical in a real time situation. We, however, consider only computationally simple algorithms.

Competitive ratio is a formal way of evaluating algorithms that are limited in some way, (e.g., limited information, computational power, or number of preemptions). This measure was first introduced in the study of a system memory management problem [6, 11, 17]. In our situation, the competitive ratio considers the best scheduling algorithm from amongst those being considered (i.e., non-clairvoyant, reasonable computation time, and a limited number of preemptions). Then it considers the worst case set of jobs for that scheduler from amongst those being considered (i.e., batch, multi phases, and some class of speedup function). How well this scheduler performs on this set of jobs is then compared with how well the optimal scheduler performs on this same set of jobs. Note that the optimal scheduler is fully clairvoyant, has unbounded computational power, and is allowed unbounded number of preemptions. The ratio of these mean response times is known as the competitive ratio of the class of schedulers on the class of jobs.

1.2 Related Results

Motwani *et al.* [15] show that for any uniprocessor system, any non-clairvoyant algorithm has a competitive ratio of at least $2 - \frac{2}{n+1}$. This lower bound extends to multiprocessor systems where the jobs are fully parallelizable. A job is *fully parallelizable* if for any p, its execution time when given p processors is p times less than its execution time with one processor. They also give some upper and lower bounds on the tradeoff between preemptions and competitive ratio. These, however, apply only to the single processor model.

A worst case set of jobs for Equi-partition consists of

n jobs each with work $W_i = p$. In Equi-partition, each job is allocated p/n processors and hence completes at time $c_i = n$. The flow is $F(EQUI) = \sum_i c_i = n^2$. The optimal schedule, on the other hand, executes the job with least work first. The completion time of job J_i is $c_i = i$ and the flow is $F(OPT) = \sum_i i = n(n+1)/2$. Hence, the competitive ratio is $2 - \frac{2}{n+1}$.

Deng and Koutsoupias [4] discuss how well a job is able to utilize processors, using a DAG model to represent the data-dependency within the job. Their lower bounds for the DAG model are not applicable to the phase job model here.

Deng *et al.* [3] show that DEQ, an algorithm similar to Equi-partition, achieves the same competitive ratio $2 - \frac{2}{n+1}$ for parallel jobs with a single phase, and is $4 - \frac{4}{n+1}$-competitive in a job model that allows jobs to have multiple phases. In this job model, each phase q of job i is fully parallelizable for any allocation of processors up to some number P_i^q, but achieves a speedup of P_i^q for any allocation greater than P_i^q. DEQ uses these values P_i^q to make its scheduling decisions.

Turek *et al.* [19] consider a general job model where jobs consist of a single phase and have speedup functions that are nondecreasing and sublinear. They achieve the impressive competitive ratio of two, even with no preemptions. However, the algorithm requires complete knowledge of the jobs' workload and speedup functions and a perhaps excessive computation time of $O(n(n^2+p))$.

In contrast, we show that the simple Equi-partition algorithm achieves a competitive ratio of $2+\sqrt{3}$ where jobs have multiple phases of different nondecreasing sublinear speedup functions. This scheduler does require up to n preemptions, but is non-clairvoyant and computationally simple. We also prove a lower bound of $e \approx 2.71$ for Equi-partition when the jobs have nondecreasing sublinear speedup functions, thus separates these kinds of jobs from fully parallelizable jobs with respect to Equi-partition.

Prior to our result, Kalyanasundaram and Pruhs [5] consider the model in which jobs can arrive at arbitrary times. In this model, it is more difficult to find good schedulers. In fact, Motwani *et al.* [15] prove that no non-clairvoyant scheduler can achieve a competitive ratio better than $\Omega(n/\log n)$ even when all the jobs are fully parallelizable. On the other hand, Kalyanasundaram and Pruhs achieve a competitive ratio of $1 + \frac{1}{\epsilon}$ by giving the their BALANCE scheduler $(1+\epsilon)p$ processors and only giving the optimal scheduler p processors.

In contrast, while their results only work for fully parallelizable jobs, ours work for a wide range of classes of speedup functions while not giving the scheduler extra processors.

In Section 2, we formally introduce our job model and provide a summary of our results. In Sections 3 and 4, we present upper and lower bounds for the case when jobs are nondecreasing and sublinear in each phase and the scheduler is allowed at least n preemptions. In Sections 5, we summarize the other results in the full paper. In Section 6, we conclude our paper with open problems. In this extended abstract, we define a variety of job classes, but we will provide the full proof only for nondecreasing sublinear jobs. In the full paper, we will provide proofs for all of the bounds mentioned in Figure 2.

2 Summary of the Results

In this section we define sets of jobs, schedulers, flow time, and competitive ratios. We then define a number of classes of schedulers and of job sets. Figure 2 summarizes our results.

2.1 Sets of Jobs and Schedulers

We consider a set of n jobs, all of which arrive at time zero, that are to be executed on p processors. A set of jobs J is defined to be $\{J_1, \ldots, J_n\}$ where *job* J_i has a sequence of phases $\left\langle J_i^1, J_i^2, \ldots, J_i^{q_i} \right\rangle$ and each phase is an ordered pair $\langle W_i^q, \Gamma_i^q \rangle$. The quantity W_i^q is a nonnegative real number, called the *work*, and Γ_i^q is a function, called the *speedup function*, that maps a nonnegative real number to a nonnegative real number. $\Gamma_i^q(\beta)$ represents the rate at which work is executed for phase q of job i when given β processors.

A schedule S allocates the p processors for each point in time to the jobs in the given jobs set J in a way such that all the work completes. More formally, a *schedule $\mathcal{S}_J$ for a given job set J with n jobs on p processors* is a function from $\{1, \ldots, n\} \times [0, \infty)$ to $[0, p]$ such that

1. For all t, $\sum_{i=1}^{n} S_J(i,t) \leq p$, and
2. For all i, there exist $0 = c_i^0 < c_i^1 < \ldots < c_i^{q_i}$ such that for all $1 \leq q \leq q_i$, $\int_{c_i^{q-1}}^{c_i^q} \Gamma_i^q(S_J(i,t))\, dt = W_i^q$. If $c_i^0, c_i^1, \ldots, c_i^{q_i}$ are the smallest such values that satisfy this condition, then the *completion time of phase q of job i under S* is c_i^q, for all $1 \leq q \leq q_i$.

Condition 1 above ensures that at most p processors are allocated at any given time. Condition 2 ensures that before a phase of a job begins, all of the previous phases of the job must have completed. Note that we allow a job to be allocated a non-integral number of processors. The *completion time of a job i*, denoted c_i, is the completion time of the last phase of job i (that is, phase q_i of job i).

Throughout this paper, we refer to an algorithm for producing schedules as a *scheduler*, and we identify a scheduler with the schedule it produces. The goal of the scheduler is to minimize the average completion time, $\frac{1}{n}\sum_{i \in J} c_i$, of all the jobs it must schedule. This goal is equivalent to minimizing the *flow time of J under scheduler S*, denoted $F(S_J)$, which is $\sum_{i \in J} c_i$. We use the *competitive ratio* of a scheduler to categorize it. The competitive ratio of a schedule over a class of schedules is

$$\mathrm{Min}_{S \in \mathcal{S}} \mathrm{Max}_{J \in \mathcal{J}} F(S_J)/F(OPT_J),$$

where $\mathcal{S}$ is the class of schedulers being considered, $\mathcal{J}$ is the class of job sets being considered, and OPT_J is an

optimal (unrestricted) scheduler for the job set J. This paper proves relatively tight upper and lower bounds on this competitive ratio for several classes of schedulers $\mathcal{S}$ and jobs sets $\mathcal{J}$. (See Figure 2.) For the upper bounds, we present a scheduler $S \in \mathcal{S}$ that performs within the stated ratio of the optimal for every job set $J \in \mathcal{J}$ considered. For the lower bounds, we construct for each possible scheduler, a job set J on which the scheduler performs poorly. (For the lower of e for Equi-partition, we only manage to prove the bound for the specific scheduler in question.)

In the next subsections, we classify schedulers by the number of preemptions they make and we classify job sets by the types of speedup functions they have.

2.2 Classes of Schedulers

All schedulers considered in this paper are simple. They are *non-clairvoyant*, meaning that they have no knowledge of the work W_i^q or the speedup functions Γ_i^q of the jobs in the set J. Initially, their only knowledge is the number of jobs n and the number of processors p. They are also able to detect when a job completes. They are not able to detect when a particular phase of a job completes.

The classes of schedulers $\mathcal{S}$ considered differ from each other in the number of preemptions that are allowed. A *preemption* occurs when a job that is currently being executed with some nonzero number of processors is allocated either more processors or fewer processors. We consider the classes of schedulers $\mathcal{S}$ that allow zero, $\log_2 n$, n, and an unbounded number of preemptions. Each class is a proper subclass of the next.

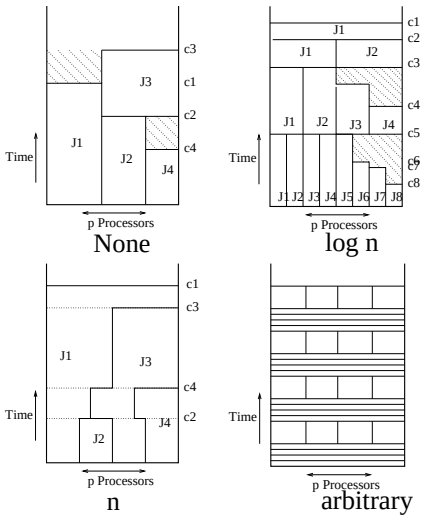

Figure 1: Examples of schedules with zero, $\log_2 n$, n, and an unbounded number of preemptions

Scheduler Class: Allows Zero Preemptions. Such a scheduler allocates some number of processors (e.g., zero, $\log_2 n$, n, and an unbounded number of preemptions. Each class is a proper subclass of the next.p/n or p) to some of the jobs. Once a job starts executing, the number of processors allocated to it must not change. However, when a job completes, the processors that had been allocated to it can be allocated to any job that has not yet been allocated processors.

Scheduler Class: Allows n Preemptions. Such a scheduler is allowed to reallocate all the processors every time some job completes. Note that a preemption of possibly all the jobs occurs at most n times.

An example of such a scheduler that is often used in practice is called *Equi-partition*. We define $EQUI_J$ to be the schedule that allocates an equal number of processors to each uncompleted job. That is, for all i and t, if job i is uncompleted at time t, then $EQUI_J(i,t) = p/n_t$, where n_t is the number of uncompleted jobs at time t, and $EQUI_J(i,t) = 0$ otherwise. Note that $EQUI$ requires no knowledge of the jobs other than how many jobs are uncompleted.

A class of schedulers of an intermediate level proposed in this paper is the following.

Scheduler Class: Allows $\log n$ Preemptions. Such a scheduler is allowed, for example, to reallocate the processors when the number of uncompleted jobs n_t reaches $n/2^i$ for all $1 \leq i \leq \log n$.

Scheduler Class: Allows an Arbitrary Number of Preemptions. The scheduler is allowed to change the processor allocation continuously (or arbitrarily often).

An example of such a scheduler used in practice is *Round Robin*. The jobs take turns being allocated all p processors for some small slice of time.

The Optimal Scheduler: In contrast, the optimal scheduler OPT, that these others are compare against, has complete knowledge (i.e., work and speedup function) of all the phases of each job, has unlimited computation power, and is allowed an unbounded number of preemptions.

2.3 Classes of Speedup Functions

We now describe the different classes of job sets $J \in \mathcal{J}$ that we consider. The work of each job phase is never restricted, and hence we only consider different numbers of phases and different classes of speedup functions. Each class is a proper subclass of the previous.

Job Class: Fully Parallelizable. Every job phase has the speedup function $\Gamma(\beta) = \beta$. (See Figure 3:a.)

Job Class: Single Phase, Fully Parallelizable or Constant Sequential. Each job has a single phase that is either fully parallelizable or constant sequential.

The *constant sequential* speedup function is $\Gamma(\beta) = 1$, for all $\beta > 0$. (See Figure 3:b). Such jobs complete work at the same rate no matter how many processors are allocated to it.

Job Class: Nondecreasing Sublinear. Every speedup function is nondecreasing and sublinear.

A speedup function Γ is *nondecreasing* if $\Gamma(\beta_1) \leq \Gamma(\beta_2)$ whenever $\beta_1 \leq \beta_2$. A job phase with a nondecreasing speedup function executes no slower if it is allocated more processors. (See Figure 3:a-h.) This is a

$\mathcal{J} \setminus \mathcal{S}$	Zero	$\log_2 n$	n	Arbitrary
Fully Parallelizable	$\Theta(\sqrt{n})$	[??, 4]	[2, 2]	
Fully Parallelizable or Const. Sequential				
Nondecreasing Sublinear		[??, 7.48]	[2.71*, 3.74]	
Nondecreasing Almost Sublinear	$\Theta\left(n^{\frac{1+\epsilon}{2}}\right)$	$\Theta(n^\epsilon)$		[??, 7.48]
Nondec. Sublinear or Superlinear	$\Theta(n)$			
Nondecreasing				$\Theta(\log n)$
Gradual	∞			$\Theta(\log p)$
Integer Domain				$\Theta(p)$
Arbitrary				∞

Figure 2: The columns in the table are for the classes of schedulers $\mathcal{S}$ that are non-clairvoyant and allow zero, $\log_2 n$, n, and an arbitrary number of preemptions, respectively. Each row represents a different class $\mathcal{J}$ of job sets. For each entry, the lower and the upper bound on the competitive ratio is given. Entries with the same bounds are grouped together. For each grouping, only one lower and one upper bound needs to be proven. The asterisk indicates a lower bound for the *EQUI*-like scheduler, not the entire class of schedulers.

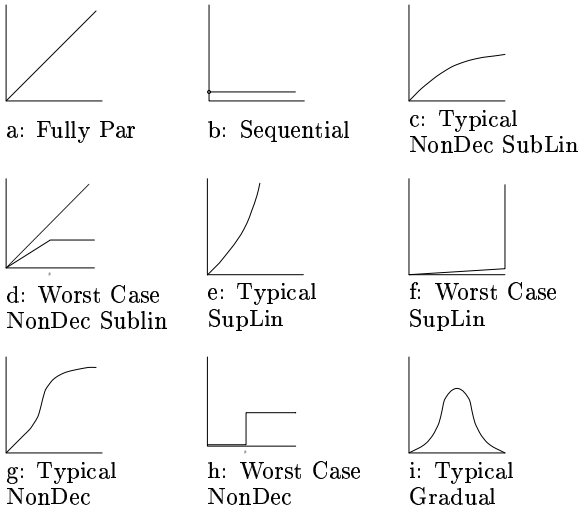

Figure 3: Examples of speedup functions.

reasonable assumption if in practice a job can determine whether it can use additional processors to speed its execution and can refuse to use some of the processors allocated to it (in the case that it cannot use additional processors).

The rate $\Gamma_i^q(\beta)$ at which a job completes work is a useful concept when considering the time until that job completes. However, when considering the completion times of all the jobs simultaneously, a more useful concept is $\Gamma_i^q(\beta)/\beta$, which is the work completed by the job per time unit per processor. One way of viewing this concept is to consider the *processor area* consumed by a job. This is measured in processor-time units. For example, if a job is allocated β processors for t time units, then the processor area consumed is βt. If β processors are allocated for the duration of J_i^q, then $W_i^q/\Gamma_i^q(\beta)$ is its execution time and $(\beta/\Gamma_i^q(\beta)) \cdot W_i^q$ is the processor area consumed.

A speedup function Γ is *sublinear* if $\beta_1/\Gamma(\beta_1) \leq \beta_2/\Gamma(\beta_2)$ whenever $\beta_1 \leq \beta_2$. A sublinear speedup function is one in which the processor area consumed per unit of work completed does not decrease when more processors are allocated to the associated job. (See Figure 3:a-d.) If for $\beta_1 < \beta_2$, β_1 processors can simulate the execution of β_2 processors in a factor of at most β_2/β_1 more time, then the speedup function is sublinear.

Job Class: Each Phase either Nondecreasing Sublinear or Superlinear. Every speedup function either is nondecreasing and sublinear or is superlinear.

In practice, job phases can have *superlinear* speedup functions, i.e., $\beta_1/\Gamma(\beta_1) \geq \beta_2/\Gamma(\beta_2)$ whenever $\beta_1 \leq \beta_2$. (See Figure 3:e-f.) Such speedup functions occur in parallel programs with a strong time-space tradeoff. For example, suppose we have a job that with one processor takes time T/S when given S space. If the job is fully parallelizable, then with β processors and S space, the required time is $T/(S\beta)$. Suppose also that when given β processors, the job has $S = c\beta$ space, where c is the amount of space in one processor. Then the time required would be $T/c\beta^2$. Thus, the speedup function for the job is $\Gamma(\beta) = \beta^2$.

Job Class: Nondecreasing Almost Sublinear. Every speedup function is nondecreasing and almost sublinear.

Though in practice the speedup functions might be superlinear, they are not likely be extremely superlinear. For example, we might want to allow the speedup

function $\Gamma(\beta) = \beta^{1+\epsilon}$ to be included for some small $0 < \epsilon \leq 1$. Note that this is almost linear. To capture this idea, we define *almost sublinear* to mean that $\beta_1^{1+\epsilon}/\Gamma(\beta_1) \leq \beta_2^{1+\epsilon}/\Gamma(\beta_2)$ whenever $\beta_1 \leq \beta_2$. Note this is less restrictive than sublinear.

Job Class: Nondecreasing. Every speedup function is nondecreasing. (See Figure 3:a-h.)

Job Class: Gradual. Every speedup function is gradual.

In practice, the point at which more processors slow down the job may not be known. This leads to jobs whose rate of computation both increase and decrease with the number of processors allocated to them. It is unreasonable, however, to consider completely arbitrary speedup functions. The following is a reasonable minimal requirement that is general enough to include most speedup functions.

A speedup function is said to be *gradual* (with respect to some constant $c > 1$) if for every number of processors β and for every value $a \in [1..2]$ either $\Gamma(a\beta/2) \geq \frac{1}{c}\Gamma(\beta)$ or $\Gamma(a\beta) \geq \frac{1}{c}\Gamma(\beta)$. In addition, we require that for a gradual speedup function, $\Gamma(\beta) = 0$ for all $\beta < 1$. (See Figure 3:i.)

Job Class: Integer Domain.

A speedup function is said to have an *integer domain* if $\Gamma(\lfloor\beta\rfloor) \geq \Gamma(\beta)$, for all β.

Job Class: Arbitrary. No restrictions at all.

Our results are summarized in Figure 2.

3 Nondecreasing Sublinear Speedup Functions

This section proves that Equi-partition has a competitive ratio of $2 + \sqrt{3}$ when the jobs have nondecreasing sublinear speedup functions. This result at first is surprising. This class of jobs includes both fully parallelizable jobs and constant sequential jobs. Under $EQUI$, the processors allocated to the sequential jobs are wasted, whereas an optimal schedule will assign an infinitesimal number of processors to such job. On the other hand, the following is intuition why a constant ratio is reasonable. If more than half the jobs are sequential, then it does not matter how the jobs are scheduled. The flow time is dominated by the sequential jobs. If fewer than half the jobs are sequential, then $EQUI$ waists fewer than half the processors on these jobs.

This section first states a lower bound on the flow time for the optimal scheduler OPT. Then for the class of jobs with nondecreasing sublinear speedup functions, an upper bound on its flow time is proved. This provides the upper bound of $2 + \sqrt{3}$ on the competitive ratio.

We give two lower bounds for the flow time for OPT. These bounds are based on the amount of processor area OPT uses in completing jobs and the amount of time OPT spends in completing jobs. Formally, the processor area used by OPT to execute job i, denoted s_i, is $\int_{OPT(i,t)>0} OPT(i,t)\ dt$. The time OPT spends to execute job i, denoted h_i, is $\int_{OPT(i,t)>0} 1\ dt$. (To simplify the discussion, we assume that $\Gamma(0) = 0$ for all speedup functions. Hence, a schedule must allocate a nonzero number of processors to make progress on a job.)

Lemma 3.1 *For any job set J, let $\pi(i)$ be the permutation of jobs sorted in reverse order by s_i. (If job i has the largest s_i, then $\pi(i) = 1$.)*

1. $F(OPT) \geq \frac{1}{p}\sum_{i=1}^{n} \pi(i)s_i$, *and*
2. $F(OPT) \geq \sum_{i=1}^{n} h_i$.

The two bounds are known in the literature as the squashed area bound and the height bound, respectively. See Turek *et al.* [19] or the full version of this paper for a complete proof.

Lemma 3.1 implies that the flow time of OPT is at least any weighted average of these two quantities. That is,

Corollary 3.2 *For any $0 \leq b \leq 1$, $F(OPT) \geq b \cdot \frac{1}{p}\sum_{i=1}^{n} \pi(i)s_i + (1-b) \cdot \sum_{i=1}^{n} d_i$.*

We now present the result that Equi-partition has a competitive ratio of at most $2 + \sqrt{3} \approx 3.74$ when all job phases have nondecreasing and sublinear speedup functions.

Theorem 3.1 *For any job set J with nondecreasing and sublinear speedup functions, $F(EQUI_J) \leq (2+\sqrt{3}) \cdot F(OPT_J)$.*

Proof of Theorem 3.1: Observe that the flow time of $EQUI$ is simply the integral over all t of n_t, the number of uncompleted jobs at time t. That is, $F(EQUI) = \int_0^\infty n_t\ dt$. We now compare the flow time of $EQUI$ to OPT using the lower bound of Corollary 3.2. The first step is to prove a lower bound on the total time h_i and processor area s_i that OPT spends on a job in terms of what is happening in $EQUI$. This is done separately for each job J_i.

Consider a job J_i. We first arbitrarily partition the time $EQUI$ spends on J_i (i.e., when $EQUI(i,t) > 0$) into infinitesimal blocks $[t, t + \Delta t]$. Then we partition the time OPT spends on J_i, (i.e., when $OPT(i,t') > 0$) into infinitesimal blocks $[t', t' + \Delta t']$ in such a way that there is a bijection between the blocks $[t, t + \Delta t]$ under $EQUI$ and the blocks $[t', t' + \Delta t']$ under OPT. The correspondence is that the same block of work of the job J_i is completed during corresponding blocks in the two different schedules. This correspondence is a bijection because both schedules complete all the work for job J_i. For each block of time, we bound separately the total time h_i and processor area s_i that OPT spends on J_i during this time.

More formally, consider one of the time blocks $[t, t + \Delta t]$ under $EQUI$. Suppose that at time t, phases $J_i^1, \ldots, J_i^{q-1}$ are complete and $W < W_i^q$ work is completed under $EQUI$. Let t' be the latest time in which the same work has been completed for J_i under OPT. Note that t' depends on which job J_i is being considered. Let $\Delta t'$ be time duration that OPT spends completing the same work that $EQUI$ completes in this

block of time. Even though the same work of J_i is completed during corresponding blocks of time $[t, t+\Delta t]$ and $[t', t'+\Delta t']$, the lengths of these time blocks will be different because the work is being completed at different rates. (See Figure 4.)

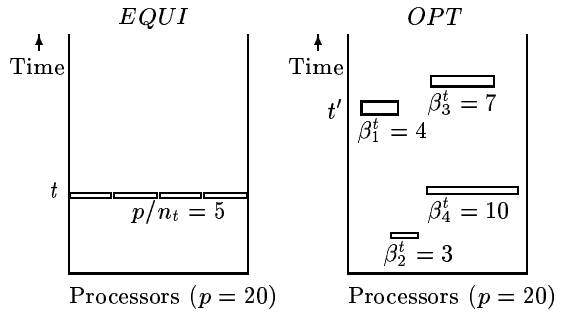

Figure 4: At time t under $EQUI$ there are four uncompleted jobs (i.e., $n_t = 4$), hence with $p = 20$ processors each job is allocated 5 processors. The work completed in $EQUI$ for each of these jobs is completed under OPT at different times and with different numbers of processors. The time t' is indicated for job 1.

By definition, $EQUI$ allocates p/n_t processors to job J_i at time t, where n_t is the number of jobs uncompleted at this time. Denote by β_i^t the number $OPT(i,t')$ of processors OPT allocates to J_i at time t'. If we allow Δt and $\Delta t'$ to become infinitesimal, then we can assume without loss of generality that these schedules assign this fixed number of processors during the duration of the respective intervals $[t, t+\Delta t]$ and $[t', t'+\Delta t']$. Hence we can conclude that during the interval $[t, t+\Delta t]$, the amount of work completed for J_i under $EQUI$ is $\Delta w = \Gamma_i^q(p/n_t) \cdot \Delta t$ and the time required to complete the same amount of work under OPT is $\Delta t' = \frac{\Delta w}{\Gamma_i^q(\beta_i^t)} = \frac{\Gamma_i^q(p/n_t)}{\Gamma_i^q(\beta_i^t)} \Delta t$.

Recall that h_i denotes the total time that OPT spends on job J_i. This is, of course, the sum of the durations of the blocks $[t', t'+\Delta t']$. We use our correspondence between the blocks $[t', t'+\Delta t']$ under OPT and the blocks $[t, t+\Delta t]$ under $EQUI$ to express h_i in terms of the schedule $EQUI$:

$$\begin{aligned} h_i &= \int_{t':OPT(i,t')>0} 1 \; dt' \\ &= \int_{t:EQUI(i,t)>0} \frac{\Gamma_i^q(p/n_t)}{\Gamma_i^q(\beta_i^t)} \; dt. \end{aligned}$$

The total processor area consumed by OPT on job J_i is denoted by s_i. This is equal to the sum of the processor areas consumed by OPT during each of the blocks of time $[t', t'+\Delta t']$, which is $OPT(i,t') \cdot dt' = \beta_i^t \cdot dt'$. We again use our correspondence between the blocks to express s_i in terms of the schedule $EQUI$:

$$\begin{aligned} s_i &= \int_{t':OPT(i,t')>0} OPT(i,t') \; dt' \\ &= \int_{t:EQUI(i,t)>0} \beta_i^t \frac{\Gamma_i^q(p/n_t)}{\Gamma_i^q(\beta_i^t)} \; dt. \end{aligned}$$

Substituting the definitions of s_i and h_i into the lower bound of Corollary 3.2, we get

$$\begin{aligned} F(OPT) \;\geq\; & \frac{b}{p} \sum_{i=1}^{n} \pi(i) \left(\int_{t:EQUI(i,t)>0} \beta_i^t \frac{\Gamma_i^q(p/n_t)}{\Gamma_i^q(\beta_i^t)} \; dt \right) \\ & + (1-b) \sum_{i=1}^{n} \left(\int_{t:EQUI(i,t)>0} \frac{\Gamma_i^q(p/n_t)}{\Gamma_i^q(\beta_i^t)} \; dt \right). \end{aligned}$$

Define S_t to be the set of all uncompleted jobs in $EQUI$ at time t such that $p/n_t < \beta_i^t$. Define S_t' to be the set of all uncompleted jobs in $EQUI$ at time t such that $p/n_t \geq \beta_i^t$. Intuitively, S_t is the set of jobs that receive fewer processors under $EQUI$ than under OPT for the work executed at time t under $EQUI$ and so these jobs are at least as work efficient under $EQUI$, since all speedup functions are sublinear, whereas S_t' is the set of jobs that receive at least as many processors under $EQUI$ than under OPT and so execute no slower under $EQUI$, since all speedup functions are nondecreasing. (In Figure 4, jobs 1 and 2 are in S_t, and jobs 3 and 4 are in S_t'.) By observing that $S_t \cup S_t'$ is the set of all jobs for which $EQUI(i,t) > 0$, we can interchange the summations with the integrals. Then by including only some of these jobs in each sum, we get that

$$\begin{aligned} F(OPT) \;\geq\; & \int_0^\infty \left(b \cdot \sum_{i \in S_t} \pi(i) \frac{\beta_i^t}{p} \frac{\Gamma_i^q(p/n_t)}{\Gamma_i^q(\beta_i^t)} \right. \\ & \left. + (1-b) \cdot \sum_{i \in S_t'} \frac{\Gamma_i^q(p/n_t)}{\Gamma_i^q(\beta_i^t)} \right) \; dt. \end{aligned}$$

Suppose $J_i \in S_t$. Then $p/n_t < \beta_i^t$, and so $EQUI$ allocates fewer processors than OPT does. Since Γ_i^q is sublinear, the instantaneous rate at which processor area is consumed per unit of work for a higher allocation of processors is at least that of a lower allocation of processors. That is, $\beta_i^t/\Gamma_i^q(\beta_i^t) \geq (p/n_t)/\Gamma_i^q(p/n_t)$, where q is the phase of job i executing at time t under $EQUI$. Rearranging this gives $\frac{\beta_i^t}{p} \frac{\Gamma_i^q(p/n_t)}{\Gamma_i^q(\beta_i^t)} \geq \frac{1}{n_t}$.

Now suppose $J_i \in S_t'$. Then $p/n_t \geq \beta_i^t$, and so $EQUI$ allocates at least as many processors as OPT. But since Γ_i^q is nondecreasing, the rate at which work of phase q of job i is being completed is at least as great

for $EQUI$ than for OPT. That is, $\frac{\Gamma_i^q(p/n_t)}{\Gamma_i^q(\beta_i^t)} \geq 1$. This gives us

$$F(OPT) \geq \int_0^\infty \left(b \sum_{i \in S_t} \pi(i) \frac{1}{n_t} + (1-b) \sum_{i \in S_t'} 1 \right) dt.$$

Let $|S_t| = a_t \cdot n_t$. (And so $|S_t'| = (1-a_t) \cdot n_t$.) The value a_t is the fraction of unfinished jobs in $EQUI$ at time t that are in S_t. Because π is a permutation, there is at most one $i \in S_t$ such that $\pi(i) = 1$, one $i \in S_t$ such that $\pi(i) = 2$, etc. Since there are only $a_t \cdot n_t$ jobs in S_t, it follows that $\sum_{i \in S_t} \pi(i)$ is at least $\sum_{i=1}^{a_t \cdot n_t} i \geq (a_t \cdot n_t)^2/2$. Thus,

$$\begin{aligned} F(OPT) &\geq \int_0^\infty \left(b \frac{(a_t n_t)^2}{2 \cdot n_t} + (1-b)(1-a_t) n_t \right) dt \\ &= \int_0^\infty n_t \left(b \frac{a_t^2}{2} + (1-b)(1-a_t) \right) dt. \end{aligned}$$

We now choose $b = \frac{1}{\sqrt{3}}$. Since we do not know what a_t is, we must consider the value of a_t that minimizes the right hand side of the equation. The minimum of $\frac{1}{\sqrt{3}}(a_i^2/2) + (1 - \frac{1}{\sqrt{3}})(1-a_t)$ over all $0 \leq a_t \leq 1$ is $(2-\sqrt{3})$, which implies that

$$F(OPT) \geq \int_0^\infty n_t \left(2 - \sqrt{3}\right) dt.$$

But $F(EQUI) = \int_0^\infty n_t \, dt$, giving $F(OPT) \geq (2-\sqrt{3}) \cdot F(EQUI) = 1/(2+\sqrt{3}) \cdot F(EQUI)$. This concludes the proof of Theorem 3.1. □

4 A Lower Bound of e for $EQUI$

We now present a lower bound of e (the base of the natural logarithm) on the competitive ratio of $EQUI$ in our multi-phase job model. We do this by presenting an infinite sequence of job sets of increasing size such that in the limit, the competitive ratio of $EQUI$ is at least e.

Theorem 4.1 *For the set of job sets with nondecreasing and sublinear speedup functions, the competitive ratio of EQUI is at least e.*

Proof of Theorem 4.1: Consider the following job set J, consisting of n jobs. Each job in J consists of two phases. The first phase of the jobs is a constant sequential phase. (That is, $\Gamma_i^1(\beta) = 1$, for all β and i.) The second phase is a fully parallelizable phase. (That is, $\Gamma_i^2(\beta) = \beta$, for all β and i.)

Job set J under OPT, where $n = 8$. Each of the phases on the left side of the figure are first phases of jobs and require no processors to complete. (The first phases of jobs 2 and 4 are indicated.) The phases on the right side of the figure are the second phases of jobs.

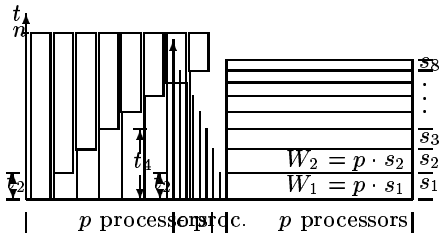

Figure 5: $EQUI$ allocates only p/n processors to each job. Those allocated to sequential phases are waisted. All jobs finish at time n. OPT allocates all p processor to the next fully parallelizable phase. This fully parallelizable phase completes just as the fully parallelizable phase of the next job is ready.

Job set J under $EQUI$, where $n = 8$. Each of the first phases of jobs complete at the same time as they did under OPT, but the second phases are allocated only p/n processors.

The work of these phases is defined by the sequences t_i and s_i below, and is illustrated in Figure 5 for $n = 8$. The sequences are defined recursively as follows:

$$\begin{aligned} t_1 &= 0 \quad , \quad s_1 = 1 \\ t_i &= t_{i-1} + s_{i-1} \quad , \quad s_i = 1 - t_i/n \end{aligned} \tag{1}$$

The quantity t_i is the time required for the first phase of job i when allocated any number of processors, and s_i is the time needed for the second phase of job i when allocated p processors. From t_i and s_i, we define the work of phases in J as follows:

$$\begin{aligned} W_i^1 &= t_i \quad , \quad \text{for all } 1 \leq i \leq n \\ W_i^2 &= p \cdot s_i \quad , \quad \text{for all } 1 \leq i \leq n \end{aligned}$$

In the optimal schedule, only an infinitesimal number of processors are allocated to each job whose first phases are uncompleted, and p processors are allocated to the job whose first phase is complete but whose second phase is not complete. Note that even though only an infinitesimal number of processors are allocated to them, the first phases of jobs complete work, since $\Gamma_i^1(\epsilon) = 1$. One can easily prove by induction that job J_i completes in time $t_i + s_i$. For the basis case, the first phase of J_1 takes $t_1 = 0$ time and so the second phase starts at time 0. This second phase requires s_1 time when allocated p processors.

For job J_i, work is completed at a rate of 1 because $\Gamma_i^1(\epsilon) = 1$. Hence, this phase requires t_i time. The work of each phase is constructed so that $t_i = t_{i-1} + s_{i-1}$. Hence, the first phase of job i completes exactly when the second phase of job $i-1$ completes, allowing the second phase of J_i to be allocated all p processors at that point. Thus, the flow time of J under the optimal schedule is $\sum_{i=1}^n (t_i + s_i)$.

We solve for t_i by substituting the definition of s_i

into the definition of t_i (from Equation (1)). The solution is

$$t_i = n - n\left(\frac{n-1}{n}\right)^{i-1}.$$

From this we get that $\sum_{i=1}^{n} t_i = n^2\left(1-\frac{1}{n}\right)^n \leq \frac{n^2}{e}$. Then $\sum_{i=1}^{n} s_i = \sum_{i=1}^{n}(1-t_i/n) = O(n)$. The flow time for J under the optimal schedule is $\sum_{i=1}^{n}(t_i+s_i) = \frac{n^2}{e} + O(n)$.

Under the $EQUI$ schedule, all n jobs are uncompleted until time n. To see this, suppose to the contrary that there were some job that completed before time n. Let J_i be the first such job. Then it is allocated p/n processors until it completes. Therefore it takes $W_i^1 + W_i^2/(p/n) = t_i + s_i \cdot n = n$, contradicting our original assumption. Therefore, the flow time for J under $EQUI$ is n^2. (See Figure 5.)

Thus, as n approaches infinity, the ratio $F(EQUI)/F(OPT)$ approaches e. □

5 Summary of Other Results

5.1 A Special Case Where $EQUI$ Does Well

It is reasonable to believe that the worst case amongst jobs with nondecreasing sublinear speedup functions occurs when all jobs are either fully parallelizable or constant sequential, since $EQUI$ wastes processors on the constant sequential jobs whereas OPT does not. However, we can show that in such cases, the competitive ratio is at most 2, beating the lower bound of e for multiple phases of either fully parallelizable or constant sequential.

5.2 Reducing the Number of Preemptions to $\log_2 n$

Let $EQUI'$ be the scheduler that is like $EQUI$ except that it reallocate the processors only when the number of uncompleted jobs reaches $n/2^i$ for all $1 \leq i \leq \log_2 n$. Note, this scheduler preempts at most $\log_2 n$ times. (See Figure 1.) For nondecreasing sublinear speedup functions, $EQUI'$ performs within a factor of two as well as $EQUI$ because the number of processors a job has under $EQUI'$ is always within a factor of two of that under $EQUI$. Hence, it has a competitive ratio of $\Theta\left(2\cdot(2+\sqrt{3})\right)$.

5.3 No Preemptions and Nondecreasing Sublinear Speedup Functions

We also consider the class of schedulers that are not allowed any preemptions. We define the $p/\sqrt{n}$-scheduler to partition the processors into $\sqrt{n}$ groups of $p/\sqrt{n}$ processors each. Each group is allocated to a different job. When a job completes, the group is allocated to another job, until all the jobs have been completed. Note that this scheduler never preempts (i.e., changes the number of processors allocated to a job once it starts.) We prove that this scheduler achieves a competitive ratio of $\sqrt{n}$ for every job set with nondecreasing sublinear speedup functions.

5.4 Nondecreasing Sublinear or Superlinear Speedup Functions

We define a scheduler $HEQUI$ that must perform well both with nondecreasing sublinear job phases and with superlinear ones without knowing which are which. It will performs well with the nondecreasing sublinear phases because half of the time it behaves like $EQUI$ and hence performs on these within a factor of two as well as proved in Theorem 3.1. Superlinear phases execute the most efficiently when given all p processors. $HEQUI$ performs well on these because half of the time it behaves like Round Robin. (See Figure 1.) This scheduler must be able to preempt an arbitrary number of times, but achieves a competitive ratio of $2\cdot(2+\sqrt{3})$.

5.5 Nondecreasing Speedup Functions and Gradual Speedup Functions

Jobs that have nondecreasing speedup functions or that have gradual speedup functions may execute efficiently only when allocated a specific number of processors. (See Figure 3:h-i) However, a non-clairvoyant scheduler does not know this number of processors. If, however, the scheduler executes each job with 2^k processors for $\frac{1}{\log_2 p}$ fraction of the time, then for each job, for at least a $\frac{1}{\log_2 p}$ fraction of the time, the job will be executed with a number of processors that is within a factor of two of its efficient number of processors. In this way, the scheduler achieves a competitive ratio of $\Theta\left(\frac{1}{\log_2 p}\right)$.

5.6 Jobs with Difficult Speedup Functions and Schedulers that Cannot Preempt Continuously

The previous upper bounds require the scheduler to preempt continuously. If the number of preemptions is restricted, then a lower bound of $\Omega(n)$ is proved. As long as the jobs have nondecreasing speedup functions, a scheduler is able to achieve this ratio. The scheduler runs the n jobs one at a time with all p processors, starting the next job when the previous job completes. On the other hand, the competitive ratio can be arbitrarily bad when the speedup functions are not nondecreasing.

5.7 Lower Bounds

We prove matching lower bounds for all of the results stated in Figure 2.

6 Conclusions and Open Problems

We have provided asymptotically tight bounds on the competitive ratio of non-clairvoyant scheduling algorithms for a range of job classes and a range of allowable number of preemptions. The following are possible open problems to consider.

How much does clairvoyance help? For each entry in Figure 2, what is the competitive ratio when the scheduler is given complete knowledge, but limited in the number of preemptions?

How much does computation help? For each entry in the Figure 2, what is the competitive ratio of the best algorithm to an optimal one that is also limited in the number of preemptions?

Our work applies to the case when all jobs arrive at time 0. In a practical scheduling environment, jobs arrive periodically and are their arrival times are generally unpredictable. An open problem is to provide results in this environment. Kalyanasundaram and Pruhs [5] provide some results in this area.

References

[1] S. H. Chiang, R. K. Mansharamani, and M. 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*, pages 33–44, 1994.

[2] X. Deng and P. Dymond. On multiprocessor system scheduling. In *Seventh ACM Symposium on Parallel Architectures and Algorithms*, June 1996.

[3] X. Deng, N. Gu, T. Brecht, and K. Lu. Preemptive scheduling of parallel jobs on multiprocessors. In *Seventh Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 159–167, Atlanta, Georgia, January 1996.

[4] X. Deng and E. Koutsoupias. Competitive implementation of parallel programs. In *Fourth Annual ACM–SIAM Symposium on Discrete Algorithms,*, pages 455–461, 1993.

[5] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In *Proceedings of the 36th Symposium on Foundations of Computer Science*, pages 214–221, October 1995.

[6] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:79–119, 1988.

[7] M. Kumar. Measuring parallelism in computation-intensive scientific/engineering applications. *IEEE Transactions on Computers*, 37(9):1088–1098, September 1988.

[8] S. Leutenegger and R. Nelson. Analysis of spatial and temporal scheduling policies for semi-static and dynamic multiprocessor environments. Technical Report RC 17086 (75594), IBM T. J. Watson Research Center, Yorktown Heights, NY, August 1991.

[9] S. Leutenegger and M. Vernon. The performance of multiprogrammed multiprocessor scheduling policies. In *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 226–236, Boulder, Colorado, May 1990.

[10] W. Ludwig and P. Tiwari. The power of choice in scheduling parallel tasks. Technical Report TR 1190, Computer Science Department, University of Wisconsin, Madison, November 1993.

[11] M. Manasse, L. McGeoch, and D. Sleator. Competitive algorithms for on-line problems. In *Proceedings of the Twentieth Annual ACM Symposium on the Theory of Computing*, pages 322–333, 1988.

[12] R. Mansharamani and M. Vernon. Qualitative behavior of the EQS parallel processor allocation policy. Technical Report TR 1192, Computer Sciences Department, University of Wisconsin, Madison, November 1993.

[13] C. McCann, R. Vaswani, and J. Zahorjan. A dynamic processor allocation policy for multiprogrammed, shared memory multiprocessors. *ACM Transactions on Computer Systems*, 11(2):146–178, May 1993.

[14] C. McCann and J. Zahorjan. Scheduling memory constrained jobs on distributed memory parallel computers. In *Proceedings of International Joint Conference on Measurement and Modeling of Computer Systems, ACM SIGMETRICS 95 and Performance 95*, pages 208–219, 1995.

[15] R. Motwani, S. Phillips, and E. Torng. Non-clairvoyant scheduling. In *Proceedings of the 4th Annual ACM/SIAM Symposium on Discrete Algorithms*, pages 422–431, Austin, Texas, January 1993.

[16] U. Schwiegelshohn, W. Ludwig, J. Wolf, J. Turek, and P. Yu. Smart SMART bounds for weighted response time scheduling. To appear in *SIAM Journal on Computing*.

[17] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[18] A. Tucker and A. Gupta. Process control and scheduling issues for multiprogrammed shared-memory multiprocessors. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pages 159–166, 1989.

[19] J. Turek, W. Ludwig, J. L. Wolf, L. Fleischer, P. Tiwari, J. Glasgow, U. Schwiegelshohn, and P. S. Yu. Scheduling parallelizable tasks to minimize average response time. In *6th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 200–209, June 1994.

[20] J. Turek, U. Schwiegelshohn, J. Wolf, and P. Yu. Scheduling parallel tasks to minimize average response time. In *Proceedings of the 5th SIAM Symposium on Discrete Algorithms*, pages 112–121, 1994.

[21] J. Zahorjan and C. McCann. Processor scheduling in shared memory multiprocessors. In *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 214–225, Boulder, Colorado, May 1990.