Non-clairvoyant Multiprocessor Scheduling of Jobs with Changing Execution Characteristics *

Jeff Edmonds
York University,
Toronto, Canada
jeff@cs.yorku.ca

Donald D. Chinn
University of Washington,
Seatle, USA
dci@cs.washington.edu

Tim Brecht
University of Waterloo,
Waterloo, Canada
brecht@cs.uwaterloo.ca

Xiaotie Deng†
City University of Hong Kong,
Kowloon, Hong Kong SAR, China
deng@cs.cityu.edu.hk

work done while together at
York University
Toronto, Canada

Abstract

In this work theoretically proves that Equi-partition efficiently schedules multiprocessor batch jobs with different execution characteristics. Motwani *et al.*show that the mean response time of jobs is within two of optimal for fully parallelizable jobs. We extend this result by considering jobs with multiple phases of arbitrary nondecreasing and sublinear speedup functions. Having no knowledge of the jobs being scheduled (*non-clairvoyant*) one would not expect it to perform well. However, our main result shows that the mean response time obtained with Equi-partition is no more than $2 + \sqrt{3} \approx 3.73$ times the optimal. The paper also considers schedulers with different numbers of preemptions and jobs with more general classes of speedup functions. Matching lower bounds are also proved.

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 and Equi-partition 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 [19]. In multiprocessor systems a similar preemptive algorithm, dynamic Equi-partition (DEQ), can be used to achieve similar performance when preemption costs are not prohibitively large [3, 4]. 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 [25, 26, 21], but these results require complete information about jobs in the system.

*To appear in the Journal of Scheduling

†All four authers were partially supported by NSERC. Deng was also funded by a grant of HKRGC (CityU 1049/98E) and a grant of City University of Hong Kong.

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.* [25]. Our metric of performance is the mean response time of the jobs. To provide flexibility in modeling these costs, we classify scheduling algorithms by the number of preemptions they are allowed, ranging from none to an infinite number. We also explore job classes, categorized by their execution characteristics. We examine how well a scheduler can perform if it is presented with jobs from a particular class of jobs. Another way to view these job classes is that if a system administrator knows what kind of jobs are scheduled on the system, he or she can choose a non-clairvoyant algorithm based on this information. Our goal is to find practical algorithms that have good analytic properties.

We study the simple Equi-partition algorithm, for which an equal number of processors is assigned to every job. The approach of Equi-partition was first introduced to parallel scheduling by Tucker and Gupta as a *process control* policy [24], and later modified by Zahorjan and McCann [27] to dynamically adjust processor allocations as job requirements for processors change. This algorithm is known as dynamic equi-partition (DEQ).

We show that the Equi-partition 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 knowledge of 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 $\log_k n$ with an extra constant multiplicative factor of k loss in performance.

This result is perhaps most interesting when compared with the existing bound (4 times optimal) [4] 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. That is, for the job class for which the result holds, there is little advantage in preempting an arbitrarily large number of times. 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 parallel 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 [2, 13, 17, 18, 24, 27, 1]. 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. That is, we adopt a batch job processing model. It would be more general to allow jobs to arrive at arbitrary times. 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 since it is most often the measure of interest to users of such systems.

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 [11]. 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, some of which

include: sequential, fully parallelizable, sublinear, superlinear and nondecreasing. To be more general, we allow jobs to have multiple phases, each of which is defined by an amount of remaining work and a speedup function.

Most scheduling results depend heavily 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 unfinished 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 scheduling algorithms used in some previous work are computationally intensive. Depending on the scheduling problem, finding the optimal schedule may be NP-complete. (For example, see Turek *et al.* [25] for a sample of such results.) Even if the algorithm is polynomial-time, it may not be practical in a real-time situation. For example, the scheduler may need to find a perfect matching. With the goal of practicality in mind, we consider only computationally simple algorithms.

A *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 [10, 16, 23]. In our situation, the competitive ratio considers the best scheduling algorithm among 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 being considered (i.e., batch, multiple 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 an 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.* [19] 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 $1/p$ times its execution time with one processor. Motwani *et al.* 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 at least $2 - \frac{2}{n+1}$.

Deng and Koutsoupias [5] 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.* [4] 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.* [25] consider a general job model where jobs consist of a single phase and have speedup functions that are nondecreasing and sublinear. Without using preemptions they achieve the impressive competitive ratio of two. 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 $\epsilon \approx 2.71$ for Equi-partition when the jobs have nondecreasing sublinear speedup functions, thus separating this class of jobs from fully parallelizable jobs with respect to Equi-partition.

Prior to our result, Kalyanasundaram and Pruhs [9] 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.* [19] 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 their BALANCE scheduler $(1+\epsilon)p$ processors and only giving the optimal scheduler p processors. After our work, Edmonds proved that $EQUI$ with $(2+\epsilon)p$ processors has a constant competitive ratio for jobs with arbitrary arrival times and arbitrary nondecreasing sublinear speedup functions [8].

2 Jobs and Schedulers

In this section we define sets of jobs, schedulers, flow time, and competitive ratios. Finally, we summarize our results.

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 q_i phases $\langle J_i^1, J_i^2, \ldots, J_i^{q_i} \rangle$ 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 *remaining 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.

The most important examples are *fully parallelizable* (perfectly efficient) job phases and *sequential* job phases. In fully parallel work, increasing the fraction of the processing power devoted to this work by a multiplicative factor of f increases the rate at which remaining work decreases by a factor of f. It has the speedup function $\Gamma(\beta) = \beta$. (See Figure 1:a.) Work is *sequential* if increasing the fraction of the processing power that this work receives does not increase the rate at which the remaining work decreases. In practice, the rate decreases when allocated few than one processor. (See Figure 1:b.) Though all of our results apply to such phases, we will call a phase "sequential" if it completes work at a rate of 1 even when absolutely no processors are allocated to it. It has the speedup function $\Gamma(\beta) = 1$ for $\beta \geq 1$. (See Figure 1:c.) Such a job is worse for a non-clairvoyant scheduler and better for the optimal scheduler. Note that the purpose of sequential jobs is to simplify algorithm analysis; there is no claim that they model actual speed-up curves found in any application.

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 1:a-i.) This is a 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). Nguyen, Vaswani, and Zahorjan [20] provide experimental evidence that this might be possible.

The rate at which job J_i completes work, $\Gamma_i^q(\beta)$, 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 for phase q of job i 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 1:a-e.)

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

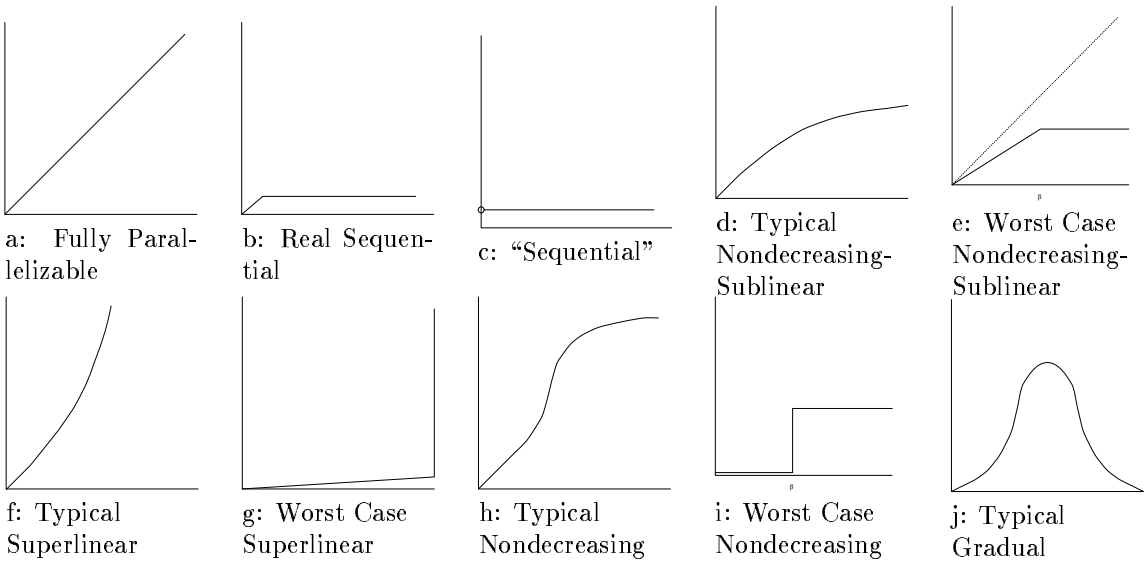

Figure 1: Examples of speedup functions

1. For all times 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.

All schedulers considered in this paper are computationally simple and most preempt a bounded number of times. 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.

In contrast to the schedulers $S \in \mathcal{S}$, the optimal scheduler OPT has unlimited computation power, is allowed an unbounded number of preemptions, and has complete knowledge (i.e., work and speedup function) of all the phases of each job.

The main result of the paper is that with nondecreasing and sublinear speedup functions, $F(EQUI_J)/F(OPT_J) \leq 2 + \sqrt{3} \approx 3.74$ (Theorem 3.1). This is surprising for the following reason. If we allow some of the job phases to be fully parallelizable (i.e., $\Gamma(\beta) = \beta$, for all β) and some to be sequential

(i.e., $\Gamma(\beta) = 1$, for all $\beta > 0$), then we would expect the competitive ratio to be unbounded, because a non-clairvoyant scheduler is unable to distinguish between these two types of phases and the processors allocated to the sequential jobs essentially are wasted. However, the optimal schedule allocates only one processor to the sequential jobs and the rest to the fully parallelizable jobs. Since the non-clairvoyant scheduler potentially wastes many processors, it is reasonable to believe that its the flow time could be unboundedly large compared to the flow time of OPT. However, this is not the case.

The other results of the paper consider schedulers with different numbers of preemptions and jobs with more general classes of speedup functions, along with matching lower bounds. The results are summarized in Figure 2.

<table>
<tr><th>$\mathcal{J} \setminus \mathcal{S}$</th><th>Zero</th><th>$\log n$</th><th>$n$</th><th>Continuous</th></tr>
<tr><td>Fully Parallelizable</td><td rowspan="2">$\Theta(\sqrt{n})$, S4.2</td><td rowspan="2">[4, 4], S4.1</td><td colspan="2" rowspan="2">[2, 2] Mot, S3.3</td></tr>
<tr><td>Fully Parallelizable or Sequential</td></tr>
<tr><td>Nondecreasing Sublinear</td><td>[4.69, 7.46], S4.1</td><td>[2.71, 3.73], S3</td><td>[2, 3.73]</td></tr>
<tr><td>Nondec. Sublinear or Superlinear</td><td colspan="3" rowspan="2">$\Theta(n)$, S5</td><td>[2, 7.46], S5.1</td></tr>
<tr><td>Nondecreasing</td><td>$\Theta(\log n)$, S5.2</td></tr>
<tr><td>Gradual</td><td colspan="3" rowspan="2">∞, S5</td><td>$\Theta(\log p)$, S5.3</td></tr>
<tr><td>Arbitrary</td><td>∞, S5</td></tr>
</table>

Figure 2: The columns in the table are for the classes of schedulers $\mathcal{S}$ that are non-clairvoyant and allow zero, $\log n$, n, and continuous 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, along with the section of the paper in which it is proved. Entries with the same bounds are grouped together. For each grouping, only one lower and one upper bound needs to be proved.

3 Nondecreasing Sublinear Speedup Functions

This section first proves the main result that Equi-partition on any job set with nondecreasing and sublinear speedup functions has competitive ratio between $e \approx 2.71$ and $2 + \sqrt{3} \approx 3.73$. The first step is to review a general lower bound on the total completion time for the optimal scheduler. Then for the class of jobs with nondecreasing sublinear speedup functions, an upper bound on its total completion time for Equi-partition is proved. This provides the upper bound of $2 + \sqrt{3}$ on the competitive ratio.

The final subsection of the section proves the lower bound of $e \approx 2.71$ for the Equi-partition algorithm. This is interesting because it beats the Motwani *et al.* [19] competitive ratio of 2 for fully parallelizable jobs.

This section proves another surprising result. One would expect that the worst case set of jobs with nondecreasing and sublinear speedup functions would be one in which all jobs are either fully parallelizable or sequential. However, it turns out that for this case the competitive ratio is 2, matching the Motwani *et al.*. (See Figure 1:a and c for graphs of parallelizable and sequential speedup functions.

3.1 A Lower Bound for OPT

We give two lower bounds for the total completion 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 $\int_{OPT(i,t)>0} 1\,dt$.

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 (see Turek *et al.* [25]) as the squashed area bound and the height bound, respectively.

Proof of Lemma 3.1: For the squashed area bound, we change OPT into OPT'' so that:

1. The total processor area consumed for each job is the same. That is, $s_i = s_i''$.
2. If c_i and c_i'' are the last times processors are assigned to job J_i under OPT and under OPT'', respectively, then $F(OPT) = \sum_{i=1}^{n} c_i \geq \sum_{i=1}^{n} c_i''$.
3. $\sum_{i=1}^{n} c_i'' = \frac{1}{p}\sum_{i=1}^{n} \pi(i)s_i$.

OPT'' might not be a legal schedule for the set of jobs J. However, from this the bound $F(OPT) = \sum_{i=1}^{n} c_i \geq \frac{1}{p}\sum_{i=1}^{n} \pi(i)s_i$ follows.

The change from OPT to OPT'' is done in two steps. We first change OPT to OPT' so that at any given time, the function OPT' allocates all p processors to exactly one job. That is, for all t, $OPT'(i,t) = p$ for some i, and $OPT'(j,t) = 0$ for all $j \neq i$. Assume without loss of generality that the jobs are sorted by completion time (that is, $c_i \leq c_{i+1}$ for all i). Let $a_{i,j}$ denote the processor area consumed by J_j during the time interval $[c_i, c_{i+1}]$, for each $i \in [1..n-1]$ and $j \in [i+1, n]$. For each interval $[c_i, c_{i+1}]$ we *squash* the processor area used by each job within that interval. More formally, define $OPT'(i+1,t)$ to be p for the first $a_{i,i+1}/p$ time units in the time interval $[c_i, c_{i+1}]$, define $OPT'(i+2,t)$ to be p for the next $a_{i,i+2}/p$ time units, and so on until we finally define $OPT'(n,t)$ to be p for the last $a_{i,n}/p$ time units in the interval. (These are the last time units in the interval because $\sum_{j\in[i+1,n]} a_{i,j}/p = c_{i+1} - c_i$.) This new function OPT' might not be a legal schedule. If the speedup function is not fully parallelizable, then the extra processors will not be utilized as efficiently and the required work will not get completed. Let c_i' be the last time processors are assigned to job J_i in OPT'. This time still will be within the time interval $[c_{i-1}, c_i]$. Hence, $\sum_i c_i \geq \sum_i c_i'$. Also, the processor area s_i for each job has not changed.

We now have a function OPT' such that at any time, exactly one job is allocated all p processors. This situation is analogous both to that when there is a single processor and to that where the jobs have with perfect speedup functions.

It is well-known that the way to minimize the sum of the completion times in this problem is to use the Least Work First schedule. (See Sevcik [22, page 124] for a complete proof of this.) The intuition is that it does not decrease the total completion time (average completion time) to have more than one job partially completed, since all uncompleted jobs must wait while a job is being completed. Thus, it is optimal to complete the jobs one at a time and in order of shortest completion time.

Let OPT'' be the schedule where the blocks of time that each job is executing are moved into one continuous time interval and then the jobs are completed in this order. The intuition is that the flow time can only improve by this change, i.e., $\sum_i c_i' \geq \sum_i c_i''$. The length of time that job J_i executes in OPT'' is s_i/p, because its processor area has been squashed across all p processors. Hence, π, which is the permutation of jobs sorted in reverse order by s_i, is the reverse order of the jobs being executed and the completion time for job J_i is $c_i'' = \sum_{j:\pi(j)\geq\pi(i)} s_j/p$ and $\sum_i c_i'' = \frac{1}{p}\sum_j \pi(j)s_j$. Therefore OPT'' has the three properties stated and the proof for the squashed area bound is complete.

For the second lower bound, we observe that the completion time c_i of a job is at least the time h_i that OPT spends executing the job. Hence, $F(OPT) = \sum_i c_i \geq \sum_i h_i$. ■

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

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

For our result in Theorem 3.1, we will fix $b = \frac{1}{\sqrt{3}}$.

3.2 Equi-partition Does Well

We now present the result that Equi-partition has a competitive ratio of at most $2 + \sqrt{3} \approx 3.73$ 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 total completion 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 total completion 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 in J_i^q 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 3.)

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 $EQUI$:

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

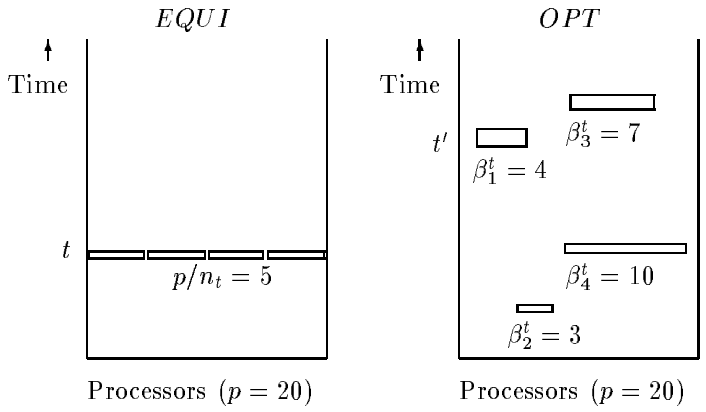

Figure 3: At time t under $EQUI$ there are four uncompleted jobs (i.e., $n_t = 4$), hence with $p = 20$ processors each job is allocated 5 processors. The work completed in $EQUI$ for each of these jobs is completed under OPT at different times and with different numbers of processors. The time t' is indicated for job 1.

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 $EQUI$:

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

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

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

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 3, 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

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

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

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

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,

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

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 $\frac{1}{\sqrt{3}}(a_t^2/2) + (1 - \frac{1}{\sqrt{3}})(1-a_t)$ 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. ■

3.3 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 sequential, since $EQUI$ wastes processors on the 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 nondecreasing sublinear speedup functions.

Theorem 3.2 *If J is such that all jobs have one phase that is either fully parallelizable ($\Gamma_i(\beta) = \beta$, for all β) or sequential ($\Gamma_i(\beta) = 1$, for all $\beta > 0$), then $F(EQUI_J)/F(OPT_J) \leq 2$.*

Proof of Theorem 3.2: Let A be the set of fully parallelizable jobs, and let B be the set of sequential jobs. We can apply the squashed area lower bound to jobs in A and the height lower bound to jobs in B. The flow time of all the jobs together is at least the sum of the flow times of each subset of jobs when considered separately. This gives

$$F(OPT) \geq \frac{1}{p}\sum_{i \in A} \pi(i)s_i + \sum_{j \in B} h_j$$

where π sorts the jobs in A in decreasing order according to their processor area consumed s_i.

The proof proceeds in a similar manner as in Theorem 3.1. Define A_t to be the set of jobs in A that are unfinished using $EQUI$ at time t. Define B_t to be the set of jobs in B that are unfinished using $EQUI$ at time t. For $i \in A_t$, $\frac{\beta_i^t}{p}\frac{\Gamma_i^q(p/n_t)}{\Gamma_i^q(\beta_i^t)} = \frac{1}{n_t}$ because $\Gamma_i^q(\beta) = \beta$, and for $j \in B_t$, $\frac{\Gamma_j^q(p/n_t)}{\Gamma_j^q(\beta_j^t)} = 1$ because $\Gamma_j^q(\beta) = 1$. Let $|A_t| = a_t \cdot n_t$. (And so $|B_t| = (1 - a_t) \cdot n_t$.) As before, $\sum_{i \in A_t} \pi(i) \geq \sum_{i=1}^{a_t \cdot n_t} i \geq (a_t \cdot n_t)^2/2$. Hence,

$$\begin{aligned} F(OPT) &\geq \int_0^\infty \left(\sum_{i \in A_t} \pi(i) \frac{\beta_i^t}{p} \frac{\Gamma_i^q(p/n_t)}{\Gamma_i^q(\beta_i^t)} + \sum_{j \in B_t} \frac{\Gamma_j^q(p/n_t)}{\Gamma_j^q(\beta_j^t)} \right) \, dt \\ &= \int_0^\infty \left(\sum_{i \in A_t} \pi(i) \frac{1}{n_t} + \sum_{j \in B_t} 1 \right) \, dt \\ &\geq \int_0^\infty n_t \left(\frac{a_t^2}{2} + (1 - a_t) \right) \, dt. \end{aligned}$$

This quadratic equation is minimized when $a_t = 1$ (i.e., the job set consists only of fully parallelizable jobs), and so $(a_t^2/2) + (1 - a_t) \geq 1/2$. As before, $F(EQUI) = \int_0^\infty n_t \, dt$, and so $F(EQUI)/F(OPT) \leq 2$.

There is a subtle point that was glossed over in this proof. The standard definition of a sequential job is one for which additional processors beyond one do not increase the rate at which work completes, i.e. the speedup function is $\Gamma(\beta) = 1$ for $\beta \geq 1$. On the other hand, $\beta \leq 1$ occurs when one processor is time sharing for β fraction of the time, giving $\Gamma(\beta) = \beta$. The above proof, however, assumes that $\Gamma(\beta) = 1$ for all β and for all sequential jobs. We can get around this problem as follows. Under $EQUI$ at each point in time t all jobs are allocated the same number of processors β_t. If $\beta_t \leq 1$, then effectively all jobs are fully parallelizable. In this case, the Motwani upper bound of 2 applies. If $\beta_t \geq 1$, then for the sequential jobs $\Gamma(\beta) = 1$ as required. ■

3.4 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 3.3 *For the set of job sets with nondecreasing and sublinear speedup functions, the competitive ratio of $EQUI$ is at least e, when $p >> n$.*

Proof of Theorem 3.3: 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 sequential phase. (That is, $\Gamma_i^1(\beta) = 1$, for all $\beta \geq 1$ and $i \in [1..n]$.) The second phase is a fully parallelizable phase. (That is, $\Gamma_i^2(\beta) = \beta$, for all β and i.)

The work of these phases is defined by the sequences t_i and s_i below, and is illustrated in Figure 4 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:

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

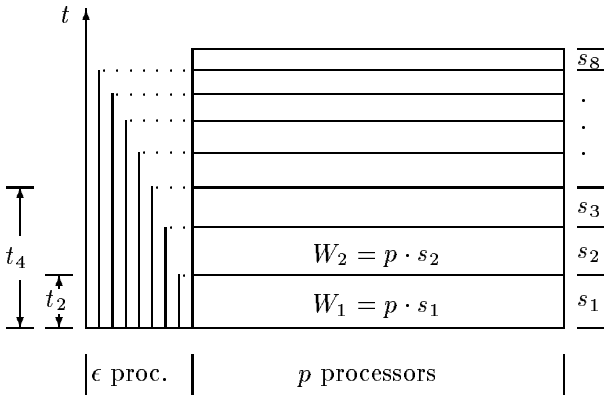

Figure 4: Job set J under OPT, where $n = 8$. Each of the phases on the left side of the figure are first phases of jobs and require no processors to complete. (The first phases of jobs 2 and 4 are indicated.) The phases on the right side of the figure are the second phases of jobs.

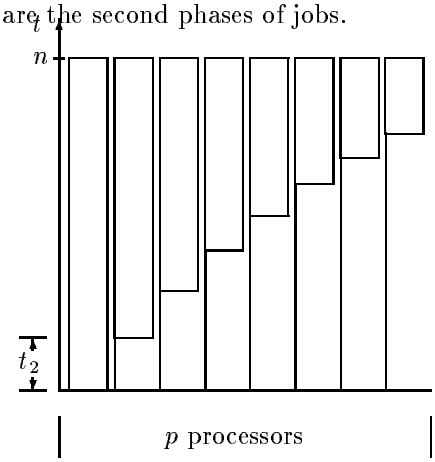

Figure 5: Job set J under $EQUI$, where $n = 8$. Each of the first phases of jobs complete at the same time as they did under OPT, but the second phases are allocated only p/n processors.

In the optimal schedule, only one processor is allocated to each job whose first phases are uncompleted, and the remaining processors are allocated to the job whose first phase is complete but whose second phase is not complete. (Because $p >> n$, this job is allocated at effectively all p processors. To simplify the calculations, we will assume that it is actually allocated all p processors.) Now, 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 total completion 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

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

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 total completion 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 total completion time for J under $EQUI$ is n^2. (See Figure 5.)

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

4 Reducing the Number of Preemptions

Preemption allows a scheduling algorithm to adapt to the uncertain and changing nature of jobs and workloads and hence is an important tool to reduce the total completion time. Unfortunately, it may incur large overheads when applied frequently. Hence, $EQUI$ performs at most n preemptions if presented with n jobs. We present one modification of the $EQUI$ schedule that performs only $\log_2 n$ preemptions and another modification that uses no preemptions at all.

In this section, we show that for any job set J, the total completion time of the $\log_2 n$ preemption $EQUI$ is within a factor of two of the total completion time of normal n preemption $EQUI$. This automatically gives an upper bound on the competitive ratio of $2 \cdot (2+\sqrt{3}) \approx 7.46$ for jobs with nondecreasing and sublinear speedup functions. We are able to prove a lower bound for this algorithm of $\frac{1}{2 \times e^{-\frac{1}{2}} - 1} = 4.69$. The proof technique is identical to that for Theorem 3.3 (though the result is less than the $2 \cdot e$ that one might expect) and hence will not be presented. See [6].

$EQUI'$ is defined to be the scheduler that is the same as $EQUI$ except that it reallocates the processors when the number of uncompleted jobs n_t is reduced to $n/2^i$ for all $1 \leq i \leq \log n$.

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

We now show how to modify the $EQUI$ schedule to one that performs only $\log_2 n$ preemptions. We call this new schedule $EQUI'$. $EQUI'$ behaves in the same way that $EQUI$ does, but instead of allocating p/n_t processors to each of the n_t uncompleted jobs, $EQUI'$ allocates p/n processors to each uncompleted job until there are $n/2$ uncompleted jobs. At this point, $EQUI'$ allocates $p/(n/2)$ processors to each of the $n/2$ uncompleted jobs until there are $n/4$ uncompleted jobs, and so on. That is, when the number of uncompleted jobs reaches $n/2^i$ for some integer i, $EQUI'$ allocates $p/(n/2^i)$ processors to each of them until there are $n/2^{i+1}$ uncompleted jobs. Clearly $EQUI'$ performs at most $\log_2 n$ preemptions. An important property of $EQUI'$ is that if there are n_t uncompleted jobs, $EQUI'$ allocates at least $p/(2 \cdot n_t)$ processors to each of those jobs. We now show that for any job set J, the total completion time of $EQUI'$ is within a factor of two of the total completion time of $EQUI$.

Lemma 4.1 *For any set of jobs with sublinear speedup functions,* $F(EQUI'_J) \leq 2 \cdot F(EQUI_J)$.

The intuition behind the proof of the lemma is that with the same number of jobs uncompleted, $EQUI'$ allocates at least half as many processors to each job as $EQUI$. Because all speedup functions are sublinear, work completes under $EQUI'$ on these jobs at a rate that is at least half of the rate under $EQUI$. Hence, the jobs require at most twice the time to complete. The only complication is that the number of uncompleted jobs may differ under the two schedules. In fact, the jobs may complete in a different order.

Proof of Lemma 4.1: Let c_i be the completion time of job J_i under $EQUI$, and sort the jobs by increasing c_i. We prove by induction on i that every job completes at least as much work after time $2c_i$ under $EQUI'$ as it does after time c_i under $EQUI$. In particular, job J_i, which completes at time c_i under $EQUI$, completes at time at most $2c_i$ under $EQUI'$. From this the lemma follows.

For convenience, we define job J_0 to be a job of one phase with zero work, and so $c_0 = 0$. Hence, the base case ($i = 0$ and $c_0 = 0$) is trivial. For the induction step, suppose that for every job, all work completed under $EQUI$ by time c_i is completed by time $2c_i$ under $EQUI'$. In order to prove that all work completed under

$EQUI$ by time c_{i+1} is completed by time $2c_{i+1}$ under $EQUI'$, it is sufficient to consider an arbitrary job J_j and prove that the work it completes under $EQUI$ during the time interval $[c_i, c_{i+1}]$ could be completed under $EQUI'$ during the time interval $[2c_i, 2c_{i+1}]$. (The only reason that it would not be completed during this interval is that is already completed before time $2c_i$.)

By the definition of c_i and c_{i+1}, there are exactly $n-i$ uncompleted jobs running under $EQUI$ during the time interval $[c_i, c_{i+1}]$, and hence each job is allocated $p/(n-i)$ processors under $EQUI$. By the induction hypothesis, under $EQUI'$ jobs $J_1, \ldots, J_i$ have completed by time $2c_i$. Hence, there are at most $n-i$ uncompleted jobs running under $EQUI'$ at any point during the time interval $[2c_i, 2c_{i+1}]$. Recall that if there are x uncompleted jobs, $EQUI'$ allocates at least $p/2x$ processors to each of those jobs. Thus, each job is allocated at least $p/(2(n-i))$ processors under $EQUI'$ in this time interval.

Consider one of the phases J_j^q of job J_j that has completed at least in part under $EQUI$ during the time interval $[c_i, c_{i+1}]$. Since the speedup function Γ_j^q is sublinear, the rate work is completed with at least $p/(2(n-i))$ processors is at least half the rate with $p/(n-i)$ processors. However, the time interval $[2c_i, 2c_{i+1}]$ is twice as long as the interval $[c_i, c_{i+1}]$. Hence, at least as much work on this phase can be completed under $EQUI'$ in $[2c_i, 2c_{i+1}]$ as under $EQUI$ during $[c_i, c_{i+1}]$. This completes the induction step and the proof of the lemma. ■

From Theorem 3.1 and Lemma 4.1, we have:

Theorem 4.1 *$EQUI'$ performs at most $\log_2 n$ preemptions. It has a competitive ratio of at most $2 \cdot (2+\sqrt{3})$ when the speedup functions are nondecreasing and sublinear.*

It is easy to see that for any $k \geq 1$ we can define $EQUI'$ so that it preempts at most $\log_k n$ times and has a competitive ratio of at most $k \cdot (2+\sqrt{3})$. As said, we are able to prove a lower bound for this algorithm of $\frac{1}{2 \times e^{-\frac{1}{2}} - 1} = 4.69$. See [6].

4.2 No Preemptions

The non-preemptive scheduling algorithm for which we are able to prove that the competitive ratio is $\Theta(\sqrt{n})$ is as follows. It partitions 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. See [6].

5 More General Classes of Speedup Functions

The main results apply only for jobs with nondecreasing sublinear speedup functions. This section proves some tight bounds for more general classes of speedup functions.

Suppose that doubling the number of processors doubles not only the number of operation completed per second but also doubles the total amount of memory. In this case, the rate at which work is completed will more than double for a job has both strong parallelizability and a strong time-space tradeoff. Such jobs do not have a sublinear speedup function. A typical one is given in Figure 1:f and a worst case one in Figure 1:g. The first result defines an *RoundRobin/EQUI* like scheduler that achieves a $2 \cdot (2+\sqrt{3}) \approx 7.46$ competitive ratio when each phase of each job is either nondecreasing sublinear or superlinear.

To be even more general, we should allow job phases whose speedup functions are neither strictly sublinear nor strictly superlinear, but are only restricted to being nondecreasing. A typical one given in Figure 1:h and a worst case one in Figure 1:i. We define another *RoundRobin/EQUI* like scheduler that achieves a $\Theta(\log n)$ competitive ratio for such jobs. The lower bound is not included in the paper, but can be found in [6].

Suppose now that the system does not know how many processors can be allocated to a phase of a job before the communication costs between the additional processors actually slows down the computation. In

such a case, the jobs do not have nondecreasing speedup functions. It is unreasonable, however, to consider completely arbitrary speedup functions. We say a speedup function is gradual if halving or doubling the number of processors allocated to the phase does not change the rate of computation by more than some fix constant factor. A typical one given in Figure 1:j. We define yet another *RoundRobin/EQUI* like scheduler that achieves a $\Theta(\log p)$ competitive ratio for such jobs.

Finally, if the speedup functions can change drastically (for every real number), then the scheduler has no chance of allocating a correct number of processors. Hence, the competitive ratio is unbounded. These results are not included. See [6].

One problem with these three *RoundRobin/EQUI* like schedulers is that they preempt continuously. Again let us consider the situation in which the number of preemptions is restricted. We are able to prove that every non-clairvoyant scheduler that is not allowed to preempt continuously has a competitive ratio of $\Theta(n)$ as long as the jobs have nondecreasing speedup functions and an arbitrarily large competitive ratio if the jobs are allowed to have gradual speedup functions. See [6].

5.1 Nondecreasing Sublinear or Superlinear Speedup Functions

When each job phase is allowed to be either nondecreasing-sublinear or superlinear, the scheduler will want to execute the nondecreasing-sublinear phases using an Equi-partition algorithm and will want to execute the superlinear jobs with an allocation of p processors using a Round Robin algorithm. By continuously switching between the two approaches, a scheduler is able to achieve a competitive ratio that is only twice the value $2+\sqrt{3}$.

Theorem 5.1 *There is a non-clairvoyant algorithm $HEQUI$ such that $F(HEQUI_J) \leq 2\cdot(2+\sqrt{3})\cdot F(OPT_J)$ for every job set J in which each phase of each job is either nondecreasing sublinear or superlinear.*

Proof of Theorem 5.1: The scheduler $HEQUI$ (short for "Hybrid Equi-partition") slices time into block containing Δ time units. If there are n_t uncompleted jobs, then for $\Delta/2$ time units, $HEQUI$ allocates p/n_t processors to each job. For the remaining $\Delta/2$ time units, each of the n_t jobs in turn is allocated all p processors for $\Delta/2n_t$ time units. Notice that as Δ approaches zero, the number of preemptions $HEQUI$ approaches infinity.

$HEQUI$ must perform well both with nondecreasing-sublinear job phases and with superlinear ones without knowing which are which. It 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.

The proof proceeds as in the proof of Theorem 3.1. Recall that in that proof, for each block of work in each job, the rate $\Gamma_i^q(\beta_i^t)$ at which OPT executes this work is compared to the rate $\Gamma_i^q(p/n_t)$ at which $EQUI$ executes the same work. We now must do the same except compare the rate $\Gamma_i^q(\beta_i^t)$ with the rate at which $HEQUI$ executes the same work. Let $\gamma_i^q(p/n_t)$ be the effective rate of $HEQUI$ as Δ approaches 0. From the definition of $HEQUI$, $\gamma_i^q(p/n_t) = \frac{1}{2}\Gamma_i^q(p/n_t) + \frac{1}{2}\frac{\Gamma_i^q(p)}{n_t}$.

There are two places in which the proof of Theorem 3.1 compares $\Gamma_i^q(\beta_i^t)$ and $\Gamma_i^q(p/n_t)$. The first place requires that if $p/n_t < \beta_i^t$ then $\frac{p/n_t}{\Gamma_i^q(p/n_t)} \leq \frac{\beta_i^t}{\Gamma_i^q(\beta_i^t)}$. We replace this with $\frac{p/n_t}{\gamma_i^q(p/n_t)} \leq 2 \cdot \frac{\beta_i^t}{\Gamma_i^q(\beta_i^t)}$. Note that the statement is a factor of two weaker, resulting in the result here being a factor of two weaker. Substituting the effective rate of $HEQUI$ gives the required bound to be

$$\frac{p/n_t}{\frac{1}{2}\Gamma_i^q(p/n_t) + \frac{1}{2}\frac{\Gamma_i^q(p)}{n_t}} \leq 2 \cdot \frac{\beta_i^t}{\Gamma_i^q(\beta_i^t)}. \tag{2}$$

If Γ_i^q is sublinear, then $\frac{p/n_t}{\Gamma_i^q(p/n_t)} \leq \frac{\beta_i^t}{\Gamma_i^q(\beta_i^t)}$ because $p/n_t < \beta_i^t$. If Γ_i^q is superlinear, then $\frac{p}{\Gamma_i^q(p)} \leq \frac{\beta_i^t}{\Gamma_i^q(\beta_i^t)}$ because $p \geq \beta_i^t$. In either case, Equation 2 is satisfied.

The second place in which the proof of Theorem 3.1 compares $\Gamma_i^q(\beta_i^t)$ and $\Gamma_i^q(p/n_t)$ is that if $\beta_i^t \leq p/n_t$, then $\Gamma_i^q(\beta_i^t) \leq \Gamma_i^q(p/n_t)$. The new requirement $\Gamma_i^q(\beta_i^t) \leq 2 \cdot \gamma_i^q(p/n_t)$ holds because Γ_i^q is nondecreasing, giving that $\Gamma_i^q(\beta_i^t) \leq \Gamma_i^q(p/n_t) \leq 2 \cdot \gamma_i^q(p/n_t)$.

The remainder of the proof follows that of Theorem 3.1, except that a factor two is introduced at each step. ■

5.2 Nondecreasing Speedup Functions

Now consider job phases whose speedup functions are neither strictly sublinear nor strictly superlinear, but are only restricted to being nondecreasing (i.e., if $\beta_1 \leq \beta_2$, then $\Gamma(\beta_1) \leq \Gamma(\beta_2)$). The worst case function (see is constant except for a sudden increase in the computation rate of the phase at some number of processors β. This value is known to the optimal scheduler OPT, but not to the non-clairvoyant scheduler S. In such a case, one would expect the competitive ratio of S to be large. If the scheduler S allocates fewer than β processors to the job, then the phase makes little progress while "wasting" the processors allocated. If it allocates many more than β processors, then those beyond β are wasted. However, we show that a non-clairvoyant scheduler can achieve a competitive ratio of $\Theta(\log n)$. For each number of processors that is a power of two between p/n and p, the scheduler runs each of the jobs with that number of processors for a small slice of time in a Round Robin fashion. It follows that for each phase for at least a $1/\log n$ fraction of the time, the phase is either allocated within a factor of two of the optimal number of processors or is allocated more than enough processors while doing Equi-Partition.

Theorem 5.2 *There is a non-clairvoyant algorithm $HEQUI'$ such that $F(HEQUI'_J) \leq O(\log n) \cdot F(OPT)$ for every set of jobs J with nondecreasing speedup functions.*

Proof of Theorem 5.2: Suppose that at time t under $HEQUI'$ there are n_t jobs remaining uncompleted. For every number of processors β that is a power of two and between p/n_t and p, the scheduler $HEQUI'$ executes each of the jobs for a slice of time while allocating it β processors. When allocating β processors per job, the scheduler is able to execute p/β jobs in parallel and hence requires $n_t\beta/p$ stages to execute each of the n_t jobs for a slice of time. Therefore, $HEQUI'$ executes each of the jobs with β processors for a time slice of length $\frac{\Delta}{\log_2(n_t)n_t\beta/p}$.

We now continue as in Theorem 3.1. Our proof here changes that of Theorem 3.1 in the same way as was done for Theorem 5.1. The effective rate of $HEQUI'$ is $\gamma_i^q(p/n_t) = \sum_{\beta=2^k \in [p/n_t,p]} \frac{1}{\log(n_t)} \frac{p}{n_t\beta} \Gamma_i^q(\beta)$. Recall that there are two statements that need to be proved.

The first required statement is that if $p/n_t < \beta_i^t$, then $\frac{p/n_t}{\gamma_i^q(p/n_t)} \leq 2\log n \cdot \frac{\beta_i^t}{\Gamma_i^q(\beta_i^t)}$. Note the result is a factor of $2\log n$ weaker because this statement is a factor of $2\log n$ weaker. Suppose that $p/n_t < \beta_i^t$ and let β be the smallest power of two that is at least β_i^t. $HEQUI'$ executes job J_i for a slice of time with β processors because $p/n_t < \beta_i^t \leq \beta$. Therefore, the effective rate of $HEQUI'$ is $\gamma_i^q(p/n_t) \geq \frac{1}{\log(n_t)} \frac{p}{n_t\beta} \Gamma_i^q(\beta)$. Because the speedup functions are nondecreasing and $\beta_i^t \leq \beta$, we know that $\Gamma_i^q(\beta_i^t) \leq \Gamma_i^q(\beta)$. Also, β is within a factor of two of β_i^t. This gives that $\gamma_i^q(p/n_t) \geq \frac{1}{2\log n} \frac{p}{n_t\beta_i^t} \Gamma_i^q(\beta_i^t)$. Rearranging this gives the required statement.

The second required statement is that if $\beta_i^t \leq p/n_t$, then $\Gamma_i^q(\beta_i^t) \leq 2\log n \cdot \gamma_i^q(p/n_t)$. If $\beta_i^t \leq p/n_t$, then let β be the smallest power of two that is at least p/n_t. Note that when $HEQUI'$ executes the jobs with β processors, it can execute all n_t of the jobs in at most two stages. The effective rate of $HEQUI'$ is $\gamma_i^q(p/n_t) \geq \frac{1}{\log(n_t)} \frac{p}{n_t\beta} \Gamma_i^q(\beta) \geq \frac{1}{2\log n} \Gamma_i^q(\beta_i^t)$. The last inequality again uses the fact that the speedup function is nondecreasing.

The remainder of the proof follows that of Theorem 3.1, except that a factor of $2\log n$ is introduced at each step. ■

See [6] for a matching lower bound.

5.3 Gradual

We might also want to include in our consideration 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. We say a speedup function is gradual if halving or doubling the number of processors allocated to the phase does not change the rate of computation by more than some fix constant factor. We consider an algorithm that for each $j \in [1..n]$ and $i \in [1..\log p]$ runs job J_j with 2^i processors for a small slice of time. This ensures that each job is running $\frac{1}{\log p}$ fraction of the time with a number of processors $2^{i'}$ that is within a factor c as fast as that assigned to the job by the adversary. We prove that this algorithm is $\Theta(\log p)$ competitive. We prove a matching lower bound. The proof, however, is not included. See [6].

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. Open problems include:

- 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 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 their arrival times are generally unpredictable. An open problem is to provide results in this environment. Kalyanasundaram and Pruhs [9] provide some results in this area.

References

[1] T. Brecht and K. Guha. Using parallel program characteristics in dynamic multiprocessor allocation policies. *Performance Evaluation*, 27 & 28:519–539, Oct. 1996.

[2] 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.

[3] X. Deng and P. Dymond. On multiprocessor system scheduling. *Journal of Combinatorial Optimization* Vol. 1, 1998, pp. 377-392, a special issue on Scheduling on Parallel/Distributed Systems.

[4] X. Deng, N. Gu, T. Brecht, and K. Lu. Preemptive scheduling of parallel jobs on multiprocessors. *SIAM J. Comput.* 30(1): 145-160 (2000)

[5] X. Deng and E. Koutsoupias. Competitive implementation of parallel programs. *Algorithmica* Vol. 23 No.1, 1999, pp.14-30.

[6] J. Edmonds, T. Brecht, D. Chinn, and X Deng. Non-clairvoyant Multiprocessor Scheduling of Jobs with Changing Execution Characteristics. Technical Report York, 2000.

[7] J. Edmonds, "Scheduling in the dark", STOC 1999 and *Blum's Special Issue of the Journal of Theoretic Computer Science,* 1999, and *Proc.* 31^{st} *Ann. ACM Symp. on Theory of Computing.*

[8] J. Edmonds. Scheduling in the Dark. In *Blum's Special Issue of the Journal of Theoretic Computer Science,* 1999 and in *Proc.* 31^{st} *Ann. ACM Symp. on Theory of Computing*, pp. 179-188, 1999.

[9] 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 and *JACM*, 2000.

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

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

[12] S.T. Leutenegger, and R.D. Nelson. Analysis of Spatial and Temporal Scheduling Policies for Semi-Static and Dynamic Multiprocessor Environments. RC 17086 (75594), IBM T. J. Watson Research Center, Yorktown Heights, NY, August, 1991.

[13] 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.

[14] W. Ludwig, and P. Tiwari. The Power of Choice in Scheduling Parallel Tasks. TR 1190 Computer Science Department, University of Wisconsin, Madison, November, 1993.

[15] R. Mansharamani, and M.K. Vernon. Qualitative Behavior of the EQS Parallel Processor Allocation Policy. TR 1192, Computer Sciences Department, University of Wisconsin, Madison, November, 1993.

[16] 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.

[17] 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.

[18] 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.

[19] 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 and Theoretical Comperter Science, Volume 130, pages 17-47, 1994.

[20] T. Nguyen, R. Vaswani, and J. Zahorjan. Maximizing speedup through self-tuning of processor allocation. In *Proceedings of the 10th International Parallel Processing Symposium*, pages 463–468, Waikiki, HI, Apr. 1996.

[21] 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.*

[22] K. Sevcik. Application scheduling and processor allocation in multiprogrammed parallel processing systems. *Performance Evaluation*, 19(2-3):107–140, March 1994.

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.