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

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 incurs overheads that can adversely affect application performance if applied frequently. 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.

The Equi-partition algorithm [?], which partitions the processors evenly between the unfinished jobs, is an example of a simple scheduler that is non-clairvoyant and preempts only when jobs complete. Motwani *et al.* [?] 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 consider a large class of jobs, characterized by multiple phases of arbitrary nondecreasing and sublinear speedup functions. For this class of jobs 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 knowledge of job characteristics to make its scheduling decisions). Also, for this class of jobs, we provide a tradeoff between performance and the number of preemptions allowed, where the number of preemptions is any bounded number.

*A condensed version of this paper will appear in the Journal of Scheduling

†{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: Microsoft Corporation, 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 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 [?]. 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 [?, ?]. 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 [?, ?, ?], but these results require complete information about 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.* [?]. 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 [?], and later modified by Zahorjan and McCann [?] 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) [?] 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 [?, ?, ?, ?, ?, ?, ?]. 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 [?]. 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.* [?] 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 [?, ?, ?]. 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.* [?] 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 [?] 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.* [?] 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.* [?] 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 $e \approx 2.71$ 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 [?] 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.* [?] 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. 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.

The remainder of this paper is organized as follows. In Section 2, we formally introduce our job model and provide a summary of our results. Section 3 provides a summary of our results. In Section 4, we present bounds for the case when jobs are nondecreasing and sublinear in each phase and the scheduler is allowed at least n preemptions. In Section 5, we show how we can reduce the number of preemptions Equi-partition makes to $\log n$, at a cost of a constant factor increase in the competitive ratio. In Section 6, we consider the case when the scheduler is allowed no preemptions. In Section 7, we allow speedup functions to be either nondecreasing and sublinear, or superlinear. In Section 8, we consider a class of jobs whose speedup functions are *almost* sublinear. In Sections 9 through 11, we consider nondecreasing, *gradual*, and *integer domain* speedup functions, respectively. In Section 12, we consider difficult speedup functions and schedulers that are not allowed to preempt often. In Section 13, we conclude our paper with a brief description of open problems.

Unless otherwise explicitly stated, all logarithms in this paper are in base 2.

2 Scheduler and Job Classes

In this section we define sets of jobs, schedulers, flow time, and competitive ratios. We then define a number of classes of schedulers and job sets. Finally, we summarize bounds on the competitive ratio for these classes.

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 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.

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:

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_k 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. We classify a scheduler by the number of preemptions it makes (which is a measure of its simplicity or practicality), and we classify a job set by the amount of knowledge the scheduler has about the speedup functions in the job set (which varies in practice).

For most of this paper, we assume a given job set J. We omit the subscript J from the names of all schedules, except where the subscripts avoid confusion.

2.2 Classes of Schedulers

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.

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. In practice, preemptions are costly. To provide flexibility in modeling this cost, we consider the number of preemptions allowed to be a parameter m and for every number of preemptions, ranging from none to an infinite number.

A useful time to preempt is when a job completes, so that the processors allocated to it can be reallocated. Ideally $m \geq n$, allowing at least one preemption per job. When $m < n$ preemptions are allowed, we prove asymptotically tight upper and lower bounds on the competitive ratio. For some types of jobs, these bounds show that every additional preemption allowed decreases the competitive ratio. Within this range, we focus on schedulers that allow zero or $\log_2 n$ preemptions. Allowing more than n but still a bounded number of preemptions does not seem to help. We prove this by giving asymptotically matching bounds where the upper bound allows n preemptions and the lower bound allows any bounded number of preemptions (e.g., n, 10^{10^n}, or even $Ackermann(n,n)$). Finally, for other types of jobs, it is helpful for the scheduler to preempt continuously. We consider such schedulers as well.

We now describe these classes of schedulers in more detail. The following is the most restricted class of schedulers $\mathcal{S}$ considered in this paper.

Scheduler Class: Allows Zero Preemptions. Such a scheduler allocates some number of processors (e.g., 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.

A difficult job set for this class of scheduler is one in which all jobs have a *perfect* speedup function $\Gamma(\beta) = \beta$, one job has a large amount of remaining work (say $W_i = p$), and the remaining jobs have essentially no work (say $W_i = \epsilon$). The optimal scheduler will first execute the jobs with ϵ work. Because these jobs complete quickly, the remaining (large) job does not have to wait long before executing. When allocated all p processors, it completes in one time unit. Hence, its completion time is $c_i \approx 1$ and the total flow time is $F(OPT) \approx 1$. The non-clairvoyant scheduler does not know the work of the jobs. One possible scheduling strategy is to allocate all p processors to each of the jobs in turn. However, even if the jobs were executed in a random order, one would expect that half of the small jobs must wait for the large job. This gives a flow time of $\Omega(n)$ and hence a competitive ratio of $\Theta(n)$. Another possible strategy is to allocate p/n processors to each of the job. However, then the job with work $W_i = p$ will take time n, again giving a flow time and a competitive ratio of $\Omega(n)$.

This paper proposes a scheduler that strikes a balance between these two extremes. We refer to it as the $p/\sqrt{n}$-scheduler. 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 (one with no processors allocated to it), until all the jobs have been completed. On the above job set, it has a competitive ratio of $\sqrt{n}$.

If on the completion of a job the processors previously allocated to the job could be partitioned among the remaining jobs, then there is a non-clairvoyant scheduler that performs well for the job set described above. The following is a less restricted class of schedulers $\mathcal{S}$ that allows this.

Scheduler Class: Allows n Preemptions. In the upper bound, n preemptions are allowed. In the lower bound, any bounded number of preemptions is allowed (e.g., n, 10^{10^n}, or even $Ackermann(n, n)$). However, the scheduler is not allowed to preempt continuously.

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 unfinished job in J. That is, for all i and t, if job J_i is unfinished at time t, then $EQUI_J(i,t) = p/n_t$, where n_t is the number of unfinished jobs at time t, and $EQUI_J(i,t) = 0$ otherwise. When a job completes, the processors allocated to it need to be redistributed among the remaining jobs. Hence, the number of preemptions required can be as much as n, one per job. Note that $EQUI$ requires no knowledge of the jobs other than how many jobs are unfinished.

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

Scheduler Class: Allows $\log_k n$ Preemptions.

The parameter m bounds the number of preemptions allowed. For $m < n$, set $m = \log_k n$ and solve for k. Define $EQUI'$ to be the scheduler that is the same as $EQUI$ except it reallocates the processors when the number of unfinished jobs n_t reaches n/k^i for all $1 \leq i \leq \log_k n$.

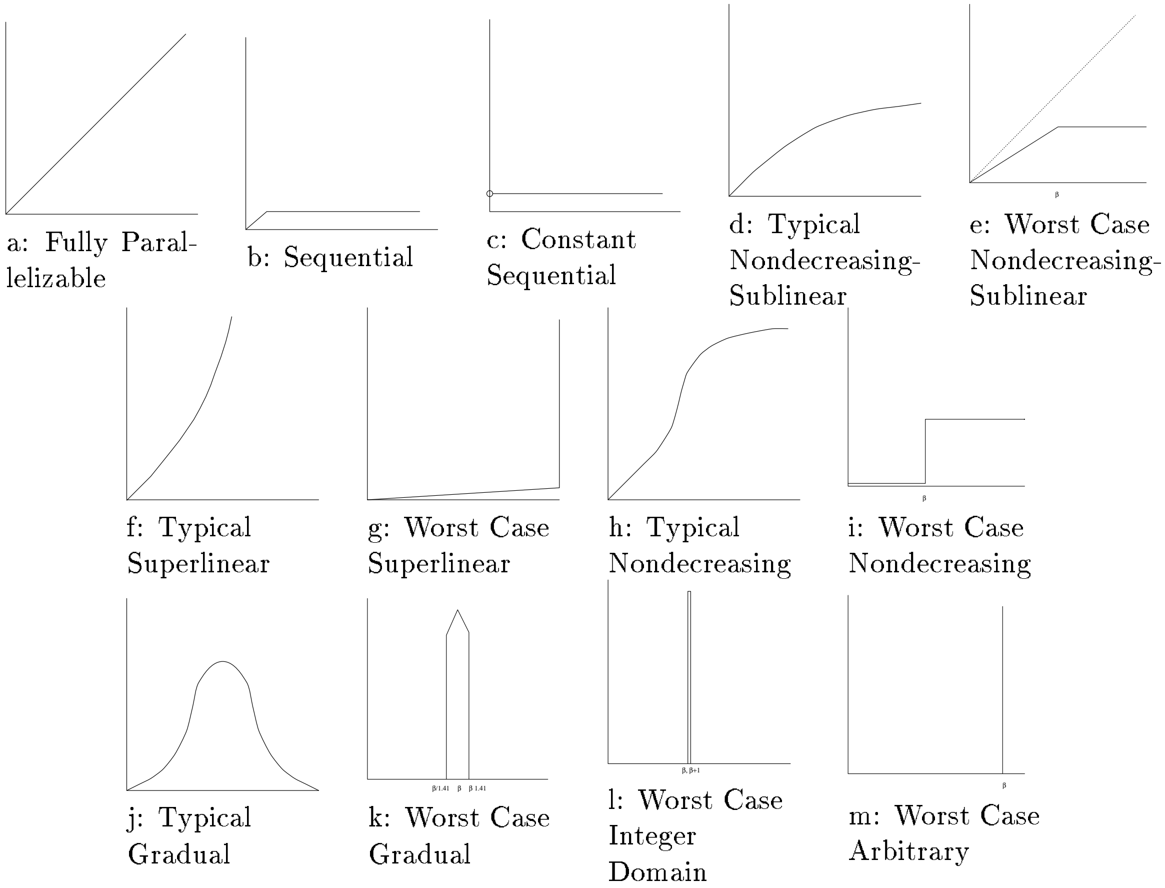
Finally, we have the least restricted class of schedulers of this paper.

Scheduler Class: Allows Continuous Preemptions. The scheduler is allowed to change the processor allocation continuously.

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. We also consider schedulers, hybrids of Equi-partition and Round Robin, that allocate different numbers of processors to jobs during different slices of time. Such non-clairvoyant schedulers are useful when the speedup functions of the jobs favors the job being allocated some number of processors but that number is unknown to the non-clairvoyant scheduler.

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 next. Before presenting the formal definition of each class, we provide motivation for defining that class. See Figure 1 for examples of these speedup functions.



a: Fully Parallelizable

b: Sequential

c: Constant Sequential

d: Typical Nondecreasing-Sublinear

e: Worst Case Nondecreasing-Sublinear

f: Typical Superlinear

g: Worst Case Superlinear

h: Typical Nondecreasing

i: Worst Case Nondecreasing

j: Typical Gradual

k: Worst Case Gradual

l: Worst Case Integer Domain

m: Worst Case Arbitrary

Figure 1: Examples of speedup functions

The class of jobs $\mathcal{J}$ that is most frequently studied in the literature is that of fully parallelizable or perfectly efficient jobs.

Job Class: Fully Parallelizable. Every job phase has the speedup function $\Gamma(\beta) = \beta$. (See Figure 1:a.)

Another common class of job sets consist of jobs that are either fully parallelizable or are sequential. One way to model a *sequential* job is with the speedup function for which $\Gamma(\beta) = \beta$ for $\beta < 1$, but $\Gamma(\beta) = 1$ for $\beta \geq 1$. (See Figure 1:b.) That is, additional processors beyond one do not increase the rate at which work completes. However, for convenience we define a different class of jobs. A job is said to be *constant sequential*, if $\Gamma(\beta) = 1$ for all $\beta > 0$. (See Figure 1:c.) Such jobs complete work at the same rate no matter how many processors are allocated to it. A scheduler that uses no knowledge of the speedup functions of jobs will allocate some processors to such a job, whereas an optimal schedule will assign an infinitesimal number of processors to such jobs. 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.

Job Class: Single Phase, Fully Parallelizable or Constant Sequential. Each job has a single phase that is either fully parallelizable ($\Gamma(\beta) = \beta$, for all β) or constant sequential ($\Gamma(\beta) = 1$, for all $\beta > 0$).

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 [?] 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:d.) If 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. The following is a way of visualizing the concept of sublinear. Draw a line from the origin to the point $(\beta_1, \Gamma(\beta_1))$ and from there to $(\beta_2, \Gamma(\beta_2))$. If the slope of the second line is less than or equal to the slope of the first line for every choice of $\beta_1 < \beta_2$, then Γ is sublinear. For example, note that $\Gamma(\beta) = 100 \cdot \beta$ is sublinear and $\Gamma(\beta) = 2 \cdot \beta - 1$ is not.

The results from Sections 4 and 5 provide a complete tradeoff between the competitive ratio of a scheduler and the number of preemptions it is allowed, where the number of preemptions is any bounded number.

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

When proving both upper and lower bounds on the competitive ratio for a class of speedup functions $\mathcal{J}$, it is useful to consider the worst case function for the class. Suppose all we know about a speedup function is that OPT achieves a rate of $\Gamma(\beta)$ by allocating some number of processors, β, during the phase. Given this, consider the function that has the minimal value of $\Gamma(\beta')$ for every β' such that the function is nondecreasing and sublinear. The resulting function is the worst case nondecreasing sublinear speedup function (Figure 1:e).

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 1: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 space S. If the job is fully parallelizable, then with β processors and space S, 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 $\Gamma(\beta) = \beta^2$.

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 Almost Sublinear. Every speedup function is nondecreasing and almost sublinear.

On the other hand, to be less restrictive, we might want to allow any speedup function that is superlinear. (The worst case function is shown in Figure 1:g.) Hence, we consider the class of job sets $\mathcal{J}$ in which some phases of a job may be nondecreasing and sublinear whereas others may be superlinear. However, every phase of a job is either strictly one or strictly the other, not a combination of both.

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

In contrast to the above restriction, we might consider speedup functions such that for some ranges of processor allocation the function is nondecreasing and sublinear, and for other ranges it is superlinear. (See Figure 1:h for a typical function and 1:i for a worst case function.) To include such functions, we consider the class $\mathcal{J}$ of jobs with nondecreasing speedup functions.

Job Class: Nondecreasing. Every speedup function is nondecreasing.

We argued above that in practice speedup functions are nondecreasing because if a job is assigned more processors than it can use, then the job can simply refuse to use the extra processors. However, in some circumstances the programmer of a parallel application does not know whether more processors will speed up or slow down the computation. The only way of knowing may be to do an impractical amount of testing (i.e., running every phase of the job with every number of processors).

We 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. The following is a reasonable minimal requirement that is general enough to include all but the last two examples in Figure 1. We refer to this class of job sets as *gradual*.

Job Class: Gradual. Every speedup function is gradual. A speedup function is 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 1:j for a typical example and 1:k for a worst case example.) The motivation for the definition is as follows. The speedup function may have some crucial values of β at which Γ changes suddenly. For example, suppose there is a sudden increase in Γ at β_1 processors and then a sudden decrease at β_2 processors. For a scheduler to perform well, it must schedule the job with a number of processors between β_1 and β_2. A non-clairvoyant scheduler does not know which value to allocate. The definition of gradual ensures that the interval $[\beta_1, \beta_2]$ is not too narrow. In particular, it ensures that this interval contains a number of processors that is a power of two. We believe that this is a reasonable assumption in practice. Suppose that a job phase has a good rate $\Gamma(\beta)$ when allocated some number of processors β. Then it is reasonable to assume that if the number of processors allocated to the phase was halved or doubled, then the rate of computation may decrease by a factor of two, or four, maybe even eight, but it will not decrease by non-constant factor. (The reason for requiring the optimal schedule to allocate at least one processor is because otherwise it would be able to take unfair advantage of a speedup function like that in Figure 1:k with β infinitesimally small.)

An even less restricted requirement is referred to as *integer domain*. It requires that the interval $[\beta_1, \beta_2]$ has width at least one.

Job Class: Integer Domain. For every speedup, function $\Gamma(\lfloor\beta\rfloor) \geq \frac{1}{c}\Gamma(\beta)$, for all β.

This ensures that the rate of computation with an integer number of processors is within a constant of that with nearby real numbers of processors. (See Figure 1:l.)

To be complete, the most general class of job sets $\mathcal{J}$ is completely unrestricted. (See Figure 1:m for the worst case function.)

Job Class: Arbitrary. No restrictions at all.

To have have meaningful units of work and rate, we can assume without loss of generality that one processor completes work at a rate of one work unit per time unit (that is, $\Gamma(1) = 1$). In Figure 1, the scale on the rate has been adjusted to emphasize the relevant feature of that class. Similarly, the actual number of processors p has no effect on any of the results (except for those with gradual or integer domain speedup functions). There is a one-to-one reduction from scheduling problems with p processors and with p' processors. This reduction is done by scaling each speedup function appropriately. This is possible because we allow the scheduler to allocate non-integral numbers of processors.

3 Summary of Results

This paper provides matching asymptotic upper and lower bounds on the competitive ratios for all the classes of schedulers and jobs defined in Section 2. Recall that the competitive ratio is

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

where $\mathcal{S}$ is the class of schedulers being considered, $\mathcal{J}$ is the class of job sets being considered, $F(S_J) = \sum_{i \in J} c_i$ is the flow time of job set J under scheduler S, and OPT_J is an optimal (clairvoyant) scheduler for the job set J. Figure 2 summarizes the results of this paper. The rest of this section highlights some of these results.

$\mathcal{J} \setminus \mathcal{S}$	Zero	$\log n$	n	Continuous
Fully Parallelizable	$\Theta(\sqrt{n})$, S6	$[4, 4]$, S5	$[2, 2]$, S4	
Fully Parallelizable or Const. Sequential				
Nondecreasing Sublinear		$[4.69, 7.48]$, S5	$[2.71, 3.74]$, S4	$[2, 3.74]$
Nondecreasing Almost Sublinear	$\Theta\left(n^{\frac{1+\epsilon}{2}}\right)$, S8	$\Theta(n^{\epsilon})$, S8		$[2, 7.48]$, S7
Nondec. Sublinear or Superlinear	$\Theta(n)$, S12			
Nondecreasing				$\Theta(\log n)$, S9
Gradual	∞, S12			$\Theta(\log p)$, S10
Integer Domain				$\Theta(p)$, S11
Arbitrary				∞, S11

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. Motwani *et al.* [?] provide the lower bound of 2 for the fully parallelizable job class. 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.

For the class of fully parallelizable jobs, Motwani *et al.* [?] show that Equi-partition has a competitive ratio of $2 - \frac{2}{n+1}$. 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 $2 - \frac{2}{n+1}$. Motwani *et al.* [?] also prove that no non-clairvoyant scheduler has a better competitive ratio.

If we allow some of the job phases to be fully parallelizable (i.e., $\Gamma(\beta) = \beta$, for all β) and some to be constant 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 constant sequential jobs essentially are wasted. However, the optimal schedule allocates an infinitesimally small number of processors to the constant sequential jobs and all p processors to the fully parallelizable jobs (each job in turn, where a job with smallest work has highest priority). Since $EQUI$ potentially wastes many processors, it is reasonable to believe that the flow time of $EQUI$ could be unboundedly large compared to the flow time of OPT. However, this is not the case. We will prove that with nondecreasing and sublinear speedup functions, $F(EQUI_J)/F(OPT_J) \leq 2 + \sqrt{3} \approx 3.74$ (Theorem 4.1). It is also interesting that the lower bound is not 2, but is at least $e \approx 2.71$ (Theorem 4.3), separating this class from fully parallelizable jobs. On the other hand, the intuition given above suggests that a worst case set of jobs is one for which each of its jobs has a single phase that is either fully parallelizable or constant sequential. However, restricted to such jobs we prove that the competitive ratio is still 2 (Theorem 4.2). The set of jobs used in the lower bound of e consists of jobs with two phases each, the first phase is constant sequential and the second phase is fully parallelizable.

Recall that $EQUI'$, defined in Section 2.2, is defined to be the scheduler that is the same as $EQUI$ except that it reallocates the processors only when the number of unfinished jobs n_t reaches $n/2^i$ for some integer i, and hence preempts only $\log n$ times. Under $EQUI'$, at each point in time, each job has at least half the number of processors as it has under $EQUI$. Hence, the flow time (when the speedup functions are sublinear) increases by at most a factor of two (Theorem 5.1). When no preemptions are allowed, a surprising result is that the $p/\sqrt{n}$-scheduler achieves a competitive ratio of $\sqrt{n}$ for all nondecreasing sublinear speedup functions (Theorems 6.1 and 6.2).

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 7.1). The competitive ratio of $2 + \sqrt{3}$ is achieved when the speedup functions are all nondecreasing and sublinear. On the other hand, if only n preemptions are allowed, the competitive ratio is $\Theta(n)$ (Theorems 12.1 and 8.1). Even if the speedup functions must be almost sublinear, then the competitive ratio still is $\Theta(n^\epsilon)$ (Theorems 8.2 and 8.1).

Now consider job phases whose speedup functions are neither strictly sublinear nor strictly superlinear, but are only restricted to being nondecreasing. See Figure 1:i for the worst case function, which 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 $\Theta(\log n)$ (Theorems 9.1 and 9.2). 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 job 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. Similarly, if speedup functions are allowed whose only restriction is that they are gradual, then the scheduler can allocate every number of processors that is a power of two between 1 and p for a competitive ratio of $\Theta(\log p)$ (Theorems 10.1 and 10.2). If the speedup functions can change drastically for every integer, then the scheduler must allocate every integer number of processors for a ratio of $\Theta(p)$ (Theorems 11.1 and 11.2). 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 (Theorem 11.3).

The above upper bounds require the scheduler to preempt continuously. Again let us consider the situation in which the number of preemptions is restricted. As long as the jobs have nondecreasing speedup functions, a scheduler can achieve a competitive ratio of $\Theta(n)$ (Theorem 12.1). Theorem 8.1 provides the matching lower bound. On the other hand, if the jobs are allowed to have gradual speedup functions, then every non-clairvoyant scheduler that is not allowed to preempt continuously can have an arbitrarily large competitive ratio.

The results are proved in the order mentioned in this section. The reader may wish to refer to Figure 2 as each result is presented.

4 Nondecreasing Sublinear Speedup Functions

This section first proves 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. The final subsection of the section proves a lower bound of $e \approx 2.71$ for the Equi-partition algorithm.

4.1 A Lower Bound for OPT

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 $\int_{OPT(i,t)>0} 1\ dt$.

Lemma 4.1 *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.* [?]) as the squashed area bound and the height bound, respectively.

Proof of Lemma 4.1: 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 with out 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 $i \in [1..p-1]$ 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 will still 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 execute 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 [?, page 124] for a complete proof.) The intuition is that it does not decrease the flow time (average completion time) to have more than one job partially completed, since all unfinished 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 $c''_i = \sum_{j:\pi(j)\geq\pi(i)} s_i/p$ 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 4.1 implies that the flow time of OPT is at least any weighted average of these two quantities. That is,

Corollary 4.2 *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 4.1, we will fix $b = \frac{1}{\sqrt{3}}$.

4.2 Equi-partition Does Well

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

Theorem 4.1 *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 4.1: Observe that the flow time of $EQUI$ is simply the integral over all t of n_t, the number of unfinished 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 4.2. 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 3.)

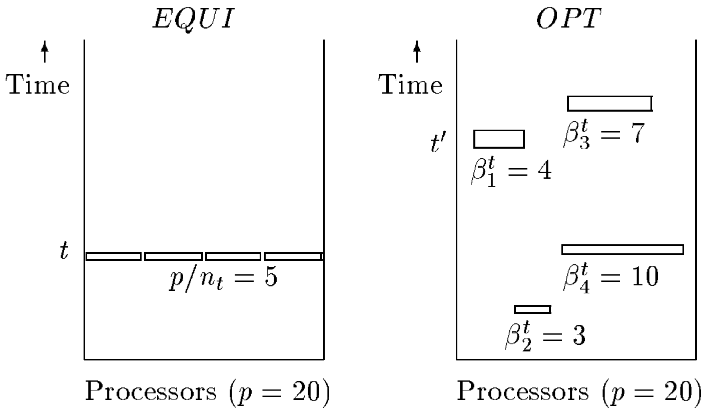


Figure 3: At time t under $EQUI$ there are four unfinished jobs (i.e., $n_t = 4$), hence with $p = 20$ processors each job is allocated 5 processors. The work completed in $EQUI$ for each of these jobs is completed under OPT at different times and with different numbers of processors. The time t' is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

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

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 4.2, 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 unfinished jobs in $EQUI$ at time t such that $p/n_t < \beta_i^t$. Define S_t' to be the set of all unfinished 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

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

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 $\frac{\beta_i^t}{p}\frac{\Gamma_i^q(p/n_t)}{\Gamma_i^q(\beta_i^q)} \geq \frac{1}{n_t}$.

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 large for $EQUI$ as 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 4.1. ■

4.3 A Special Case Where *EQUI* Does Well

As mentioned in Section 3, 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 nondecreasing sublinear speedup functions.

Theorem 4.2 *If J is such that all jobs have one phase that is either fully parallelizable ($\Gamma_i(\beta) = \beta$, for all β) or constant sequential ($\Gamma_i(\beta) = 1$, for all $\beta > 0$), then $F(EQUI_J)/F(OPT_J) \leq 2$.*

Proof of Theorem 4.2: 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 4.1. 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 $F(EQUI)/F(OPT) \leq 2$. ■

4.4 A Lower Bound of e

We now present a lower bound of e (the base of the natural logarithm) on the competitive ratio for multi-phase jobs with nondecreasing and sublinear speedup functions. The bound applies for any non-clairvoyant scheduler that is only allowed to preempt a bounded number of times. Here, the number of preemptions could be n, 10^{10^n}, or even $Ackermann(n, n)$. However, the scheduler is not allowed to preempt continuously. It is our belief that the same bound of e holds in the continuous case, but we have been unable to prove it. The lower bound presented is accomplished 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 4.3 *For the set of job sets with nondecreasing and sublinear speedup functions, the competitive ratio is at least e for any non-clairvoyant scheduler that preempt a bounded number of times.*

Proof of Theorem 4.3: First consider a non-clairvoyant scheduler S that does not preempt at least until some job completes. Suppose that when given n jobs, S initially allocates $p_i = q_i \cdot \frac{p}{n}$ processors to job J_i, where the average q_i is $\frac{1}{n}\sum_i q_i = 1$. Sort and rename the jobs so that $q_1 \leq q_2 \leq \ldots \leq q_n$. Work will be allocated to the jobs so that all the jobs complete at the same time. Hence, under S, they have the same processor allocation for their entire computation.

Consider the following job set J. 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.)

The work of these phases is defined by the sequences t_i and s_i below, and is illustrated in Figure 4 for $n = 8$ and $q_i = 1$ for all i. The sequences are defined recursively as follows:

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

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:

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

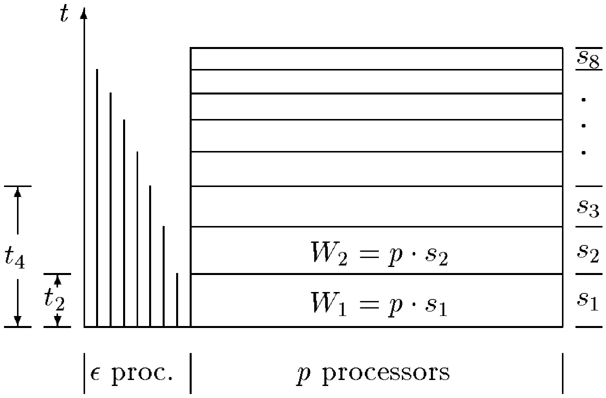


Figure 4: Job set J under OPT, where $n = 8$ and $q_i = 1$ for all i. Each of the phases on the left side of the figure are first phases of jobs and require no processors to complete. (The first phases of jobs 2 and 4 are indicated.) The phases on the right side of the figure are the second phases of jobs.

In the optimal schedule, only an infinitesimal number of processors are allocated to each job whose first phases are unfinished, and p processors are allocated to the job whose first phase is complete but whose second phase is not complete. One can easily prove by induction that job J_i completes in time $t_i + s_i$. For the basis step, 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.

The first phase of job J_i completes work at a rate of 1 even though only an infinitesimal number of processors are allocated to it 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. The second phase then completes in time s_i, giving a total completion time of $t_i + s_i$. Thus, the flow time of J under the optimal schedule is $\sum_{i=1}^{n}(t_i + s_i)$.

Under the schedule S, all n jobs are unfinished 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. Because the scheduler S does not preempt at least until some job completes, we know that job J_i is allocated $p_i = q_i \cdot \frac{p}{n}$ processors until it completes. Therefore it takes $W_i^1 + W_i^2/p_i = t_i + (p \cdot s_i) \,/\, (q_i \cdot \frac{p}{n}) = t_i + q_i\,(1 - t_i/n)\left(\frac{n}{q_i}\right) = n$, contradicting our original assumption. Therefore, the flow time for J under S is n^2. (See Figure 5.)

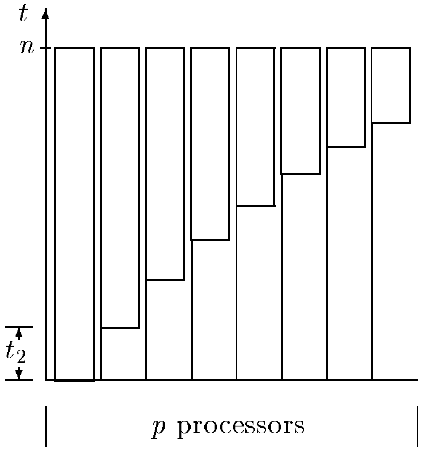


Figure 5: Job set J under S, where $n = 8$ and $q_i = 1$ for all i. Each of the first phases of jobs complete at the same time as they did under OPT, but the second phases are allocated only p/n processors.

What remains is to compute the optimal flow time $\sum_{i=1}^{n}(t_i + s_i)$. Substituting the definition of s_i into the definition of t_i (from Equation (1)) gives $t_i = (1 - \frac{q_{i-1}}{n})\,t_{i-1} + q_{i-1}$. Solving this recurrence gives

$$F \quad = \quad \sum_{i=1}^{n} t_i = \sum_{i=1}^{n}\sum_{j=1}^{i-1} q_j \prod_{k=j+1}^{i-1}\left(1 - \frac{q_k}{n}\right)$$

To begin, suppose that the scheduler S decides to allocate the same number of processors to each job. This sets each q_i to 1 and gives $\sum_{i=1}^{n} t_i = n^2\left(1 - \frac{1}{n}\right)^n \leq \frac{n^2}{e}$. Then $\sum_{i=1}^{n} s_i = \sum_{i=1}^{n}(1 - t_i/n) \leq n$. (We ignore this amount from here on because it is lower order than $F = \sum_{i=1}^{n} t_i$.) The flow time for J under the optimal schedule is $\sum_{i=1}^{n}(t_i + s_i) = \frac{n^2}{e} + O(n)$. Thus, as n approaches infinity, the ratio $F(EQUI)/F(OPT)$ approaches e.

It remains to be seen, however, that the best strategy for scheduler S is to give the same number of processors to each job. The flow time for S is fixed at n^2. S, however, has the freedom to change the values of the q_i in an attempt to increase the flow time F for the optimal. This would decrease the competitive ratio for S. On the other hand, if S assigns different numbers of processors to the jobs, then the adversary has the power to rename the jobs, reordering the q_i and hence make F even smaller. Then only by making all the q_i the same can S prevent the adversary from making F smaller by rearranging the jobs.

When changing the values of the q_i, there are two different effects, both of which indicate that it is to the adversary's advantage to sort the jobs so that $q_1 \leq q_2 \leq \ldots \leq q_n$. The first effect is that in this scheme, the optimal scheduler finishes job J_1 first. Generally, to minimize its flow time, the optimal scheduler should complete jobs with the smallest work first. Because we want all jobs under S to finish at the same time, the job allocated the smallest number of processors under S will be given the smallest amount of work. Hence, J_1 should be chosen to be the one that is allocated the smallest number of processors. The second effect is that job J_1 has the shortest phase 1 and the longest phase 2. Phase 1 does not need processors, while Phase 2 does need processors. Hence J_1 will be delayed the most by not having its share of processors.

We will now prove more formally that subject to $q_1 \leq q_2 \leq \ldots \leq q_n$ and to $\frac{1}{n}\sum_i q_i = 1$, the scheduler S cannot increase F beyond that given with each $q_i = 1$. Let $q_1, \ldots, q_n$ be values that maximize F subject to these conditions. By way of contradiction, assume that all the q_i are not 1. Consider an index l for which q_l is strictly less then q_{l+1}. We will increase q_l and decrease q_{l+1} by δ. Note that this maintains the conditions that $q_1 \leq q_2 \leq \ldots \leq q_n$ and that $\frac{1}{n}\sum_i q_i = 1$. We will prove that this change increases F. This contradicts the fact that the values are those that maximize F. The first step is to rewrite F in a way that emphasizes the q_l and the q_{l+1} factors. Here each C_x (for $1 \leq x \leq 6$) in the expression for F is a positive expression depending neither on q_l nor on q_{l+1}. Then we take the derivative with respect to q_l and q_{l+1}, increasing q_l and decreasing q_{l+1} by δ.

$$\begin{aligned}
F &= \sum_{i=1}^{n}\sum_{j=1}^{i-1} q_j \prod_{k=j+1}^{i-1}\left(1-\frac{q_k}{n}\right) \\
&= C_1 \times 1 \\
&+ C_2 \times q_l \\
&+ C_3 \times q_{l+1} \\
&+ C_4 \times q_l \left(1-\frac{q_{l+1}}{n}\right) \\
&+ C_5 \times \left(1-\frac{q_l}{n}\right) \\
&+ C_6 \times \left(1-\frac{q_l}{n}\right) \times \left(1-\frac{q_{l+1}}{n}\right),
\end{aligned}$$

$$\begin{aligned}
F' & \\
&= C_1 \times 0 \\
&+ C_2 \times 1 \\
&+ C_3 \times (-1) \\
&+ C_4 \times \left(1-\left(\frac{q_{l+1}-q_l}{n}\right)\right) \\
&+ C_5 \times \left(-\frac{1}{n}\right) \\
&+ C_6 \times \left(\frac{q_{l+1}-q_l}{n}\right)\left(\frac{1}{n}\right).
\end{aligned}$$

Later we show that $C_2 = 1$, $C_3 = C_4$, and $C_6 = C_4 \times C_5$. This gives,

$$\begin{aligned}
F' &= 1 - C_4\left(\frac{q_{l+1}-q_l}{n}\right) - C_5\left(\frac{1}{n}\right) + C_4\left(\frac{q_{l+1}-q_l}{n}\right) \times C_5\left(\frac{1}{n}\right) \\
&= \left(1 - C_4\frac{q_{l+1}-q_l}{n}\right) \times \left(1 - C_5\frac{1}{n}\right).
\end{aligned}$$

Finally, we complete the proof that $F' > 0$ by proving that the above two factors are positive. The dot notation below is a indicates to what values i and j in the expression for F need to be fixed. For example, $\cdot i = l+1$ refers to the terms in the expression for F where $i = l+1$.

$$C_2 = {}_{\cdot i=l+1 \cdot j=l} \prod_{k=l+1}^{l-1}\left(1-\frac{q_k}{n}\right) = 1$$

$$C_3 = \sum_{i=l+2}^{n} \cdot j{=}l{+}1 \prod_{k=l+2}^{i-1} \left(1 - \frac{q_k}{n}\right)$$

C_3 is equal to

$$C_4 = \sum_{i=l+2}^{n} \cdot j{=}l \prod_{i=l+2}^{i-1} \left(1 - \frac{q_k}{n}\right) \leq (n - l)$$

We now show that the first factor is positive, i.e., $\left(1 - C_4 \frac{q_{l+1} - q_l}{n}\right) \geq 0$. $C_4 \frac{q_{l+1} - q_l}{n} \leq (n - l) \frac{q_{l+1}}{n} \leq \frac{1}{n} \sum_{k \in [l+1..n]} q_k \leq 1$.

We now show that the second factor is positive, i.e., $\left(1 - C_5 \frac{1}{n}\right) \geq 1$.

$$\begin{aligned} C_5 &= \cdot i{=}l{+}1 \sum_{j=1}^{l-1} q_j \prod_{k=j+1}^{l-1} \left(1 - \frac{q_k}{n}\right) \\ &\leq \sum_{j=1}^{l-1} q_j \leq n \end{aligned}$$

Now, we show that $C_6 = C_4 \times C_5$.

$$\begin{aligned} C_6 &= \sum_{i=l+2}^{n} \sum_{j=1}^{l-1} q_j \prod_{k \in [j+1..l-1, l+2..i-1]} \left(1 - \frac{q_k}{n}\right) \\ &\leq \left(\sum_{i=l+2}^{n} \prod_{k=l+2}^{i} \left(1 - \frac{q_k}{n}\right)\right) \times \left(\sum_{j=1}^{l-1} q_j \prod_{k=j+1}^{l-1} \left(1 - \frac{q_k}{n}\right)\right) \\ &= C_4 \times C_5 \end{aligned}$$

Finally, if we define C_1 to be all terms involving neither q_l nor q_{l+1}, then it is easy to verify that the expression for F is correct.

This completes the proof when the scheduler does not preempt at least until some job completes. Consider now any non-clairvoyant scheduler S that is only allowed to preempt a bounded number of times, for example n, 10^{10^n}, or even $Ackermann(n, n)$. Run S an arbitrarily long length of time without any job completing. Let $t_1, t_2, t_3, \ldots, t_r$ be the times that S preempts before the first jobs completes. Note that this is a finite list because S preempts a bounded number of times. We now have two ways to proceed. The first way is to scale down the set of jobs presented above so that all the jobs under S complete before time t_1. In this case, as above, S does not preempt until after some job completes. The second way is to scale all the jobs up so that time t_r is inconsequentially small with respect to the running times of the job phases. Again, S effectively does not preempt during the execution of the jobs. ■

5 Reducing the Number of Preemptions to $\log n$

As seen, it is useful to preempt when a job completes, so that the processors allocated to it can be reallocated. Ideally $m \geq n$, allowing at least one preemption per job. We now consider the situation when $m < n$ preemptions are allowed. We prove asymptotically tight upper and lower bounds on the competitive ratio. These show that every additional preemption allowed decreases the competitive ratio. Because it leads to a natural scheduler, we focus on $m = \log_2 n$.

5.1 Upper Bound: $EQUI'$

$EQUI$ performs at most n preemptions if presented with n jobs. We now show how to modify the $EQUI$ algorithm to one that performs only $\log_k n$ preemptions. (The parameter m bounds the number of preemptions allowed. For $m < n$, set $m = \log_k n$ and solve for k.) We call this new algorithm $EQUI'$. $EQUI'$ behaves in the same way that $EQUI$ does, but instead of allocating p/n_t processors to each of the n_t unfinished jobs, $EQUI'$ allocates p/n processors to each unfinished job until there are n/k unfinished jobs. At this point, $EQUI'$ allocates $p/(n/k)$ processors to each of the n/k unfinished jobs until there are n/k^2 unfinished jobs, and so on. That is, when the number of unfinished jobs reaches n/k^i for some integer i, $EQUI'$ allocates $p/(n/k^i)$ processors to each of them until there are n/k^{i+1} unfinished jobs. Clearly $EQUI'$ performs at most $\log_k n$ preemptions. An important property of $EQUI'$ is that if there are n_t unfinished jobs, $EQUI'$ allocates at least $p/(k \cdot n_t)$ processors to each of those jobs. We now show that for any job set J, the flow time of $EQUI'$ is within a factor of k of the flow time of $EQUI$.

Lemma 5.1 *For any set of jobs with sublinear speedup functions, $F(EQUI'_J) \leq k \cdot F(EQUI_J)$.*

The intuition behind the proof of the lemma is that with the same number of jobs unfinished, $EQUI'$ allocates at least $1/k$ 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 $1/k$ of the rate under $EQUI$. Hence, the jobs require at most twice the time to complete. The only complication is that the number of unfinished jobs may differ under the two algorithms. In fact, the jobs may complete in a different order.

Proof of Lemma 5.1: 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 kc_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 kc_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 basis step ($i = 0$ 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 kc_i under $EQUI'$. In order to prove that all work completed under $EQUI$ by time c_{i+1} is completed by time kc_{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 $[kc_i, kc_{i+1}]$. (The only reason that it would not be completed during this interval is that it has already completed before time kc_i.)

By the definition of c_i and c_{i+1}, there are exactly $n - i$ unfinished 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 kc_i. Hence, there are at most $n-i$ unfinished jobs running under $EQUI'$ at any point during the time interval $[kc_i, kc_{i+1}]$. Recall that if there are x unfinished jobs, $EQUI'$ allocates at least p/kx processors to each of those jobs. Thus, each job is allocated at least $p/(k(n-i))$ processors under $EQUI'$ in this time interval.

Consider one of the phases J_j^q of job J_j that is partially or fully completed under $EQUI$ during the time interval $[c_i, c_{i+1}]$. Since the speedup function Γ_j^q is sublinear, the rate at which work is completed with at least $p/(k(n-i))$ processors is at least $1/k$ the rate with $p/(n-i)$ processors. However, the time interval $[kc_i, kc_{i+1}]$ 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 $[kc_i, kc_{i+1}]$ as under $EQUI$ during $[c_i, c_{i+1}]$. This completes the induction step and the proof of the lemma. ■

From Theorem 4.1, Theorem 4.2, and Lemma 5.1, we have:

Theorem 5.1 *$EQUI'$ performs at most $\log n$ preemptions. It has a competitive ratio of at most $k \cdot (2+\sqrt{3})$ when the speedup functions are nondecreasing and sublinear and has a competitive ratio of at most $2k$ when all jobs have one phase that is either fully parallelizable or constant sequential.*

5.2 Lower Bound for Fully Parallelizable Jobs and $\log_k n$ Preemptions

The following lower bound proves that for fully parallelizable jobs, the above tradeoff between the number of preemptions and the competitive ratio is tight.

Theorem 5.2 *For any non-clairvoyant scheduler allowed to preempt at most $\log_k n - O(1)$ times on n fully parallelizable jobs, its competitive ratio is at least $2k$.*

Proof of Theorem 5.2: Fix $k > 1$. Let $n = c \times k^{m+2}$, where c is a sufficiently large constant and m is the number of preemptions allowed. Also let $\epsilon > 0$ be some small value.

We prove by induction on m that the competitive ratio is at least $2k$ and that the completion times for the jobs used in the lower bound are at least ϵ^{2m} under the scheduler in question. (This last property is useful in the induction step of the proof.)

For $m = 0$, by definition no preemptions are allowed. Hence, Theorem 6.2 applies, giving a lower bound of $\Omega\left(\sqrt{n}\right) = \Omega\left(\sqrt{c \times k^{m+2}}\right) \geq 2k$, for appropriate constant c. From the proof of Theorem 6.2, the set of jobs given can be such that all jobs complete in time at least $1 > \epsilon^{2m}$.

Assume by way of induction that the induction hypothesis holds for $m \geq 0$. Consider a scheduler S that preempts at most $m+1$ times on $n = c \times k^{(m+1)+2}$ jobs. View each job as having phases. Set $t_1 = \epsilon^{2m+2}$ and assign the first phase of each job the amount of work required so that it completes at time t_1. We will allocate enough work to the remaining phases so that this time $t_1 = \epsilon^{2m+2}$ contributes insignificantly to the flow time of both this scheduler and of the optimal scheduler.

Consider the $\frac{k-1}{k}n$ jobs that are allocated the most processors at time t_1. Together they are allocated at least $\frac{k-1}{k}p$ processors. We will refer to these as the *small jobs*, because they will not have a second phase and hence will complete under S at time t_1. After these jobs complete, the $\frac{k-1}{k}p$ processors allocated to them idle until S preempts, rescheduling them to the remaining $\frac{n}{k}$ jobs.

Consider what S does in the circumstances where the remaining jobs do not complete for an arbitrarily long time. Based on no information other than the number of processors n and the completion time t_1, S must do one of the following things: it can preempt immediately ($t_2 = 0$); it can wait for some period of time $t_2 \leq 1$ before preempting; or it can wait at least until time $t_2 = 1$ to preempt, assuming none of the remaining jobs have completed by this time.

Suppose $t_2 \leq \epsilon^{2m+1}$. That is, the scheduler preempts soon after time t_1. After time $t_1 + t_2$, the scheduler must schedule the remaining phases of the remaining $n' = \frac{n}{k} = c \times k^{m+2}$ jobs with at most m preemptions. From the induction hypothesis, we know that the competitive ratio of this part of the computation is at least $2k$. Moreover, this competitive ratio is not affected significantly by the initial time period of duration $t_1 + t_2 \leq 2\epsilon \cdot \epsilon^{2m}$, because by the induction hypothesis, we can find jobs with the required competitive ratio, whose completion times are at least ϵ^{2m}.

Now suppose $t_2 > \epsilon^{2m+1}$. This is large relative to $t_1 = \epsilon \cdot \epsilon^{2m+1}$. Because no preemptions occur during the time period $[t_1, t_2]$, we know that at least $\frac{k-1}{k}p$ processors idle during this time period.

The construction of the remaining phases of the remaining $n' = \frac{n}{k}$ is similar to that of Motwani *et al.* [?]. These jobs are assigned work so that they each complete at time t_2. Motwani *et al.* show that this competitive ratio is 2. Here, however, the n' jobs have at most $\frac{p}{k}$ processors available to them. Hence, the competitive ratio is at least $2k$. Again, this competitive ratio is not affected significantly by the initial time period of length $t_1 = \epsilon \cdot \epsilon^{2m+1}$, because t_1 is small relative to $t_2 \geq \epsilon^{2m+1}$. ■

5.3 Lower Bound for Nondecreasing Sublinear Jobs and $\log_k n$ Preemptions

Using techniques from the proofs of Theorems 4.3 and 5.2, we can show a lower bound on the competitive ratio when only $\log_k n$ preemptions are allowed and jobs are allowed to have phases that are nondecreasing and sublinear. We present the bound only in the case where $k = 2$ because of the complexity of the expressions resulting from the recurrence relations. However, the same proof technique works for any value of k.

Theorem 5.3 *For any non-clairvoyant scheduler S allowed to preempt at most $\log_2 n - O(1)$ times on n nondecreasing and sublinear jobs, its competitive ratio is at least 4.69.*

Proof of Theorem 5.3: Let $n = c \times 2^{m+2}$, where c is a sufficiently large constant and m is the number of preemptions allowed. We prove by induction on m that the competitive ratio is at least 4.69. The proof follows closely to that of Theorem 5.2. The only change is in the case ($t_2 > \epsilon^{2m+1}$) where the scheduler S does not preempt during the time period $[t_1, t_2]$ even though $\frac{k-1}{k}p = \frac{1}{2}p$ of the processors idle and n' jobs have yet not completed. Instead of using fully parallelizable jobs from the Motwani *et al.* result for the remaining phases of these n' jobs to obtain a bound of $2 \cdot k = 4$, we use n' jobs with a constant sequential phase followed by a fully parallelizable phase, modeled after those in Theorem 4.3 to obtain the bound of 4.69.

The change to the proof of Theorem 4.3 is minor. In the proof of Theorem 4.3, scheduler S allocated $p_i = q_i \cdot \frac{p}{n}$ processors to job J_i. We set the completion time $W_i^1 + W_i^2/p_i$ under S to n and solved for s_i, giving $s_i = q_i\,(1 - t_i/n)$. In our proof here, because S only has access to half as many processors, it allocates only $p_i = \frac{1}{2}q_i \cdot \frac{p}{n'}$, giving $s_i = \frac{1}{2}q_i\,(1 - t_i/n')$. The proof is basically the same that the best the scheduler can do is to allocate all the jobs the same number of processors, i.e., $q_i = 1$. Then, the optimal flow, instead of being $\sum_{i=1}^{n} t_i + s_i = \frac{n^2}{e} + O(n)$, is

$\left(2 \times e^{-\frac{1}{2}} - 1\right)(n')^2 + O(n') = \frac{1}{4.69}(n')^2 + O(n')$. Because the flow time of S is $(n')^2$, this gives a competitive ratio of 4.69, as desired. ■

6 No Preemptions and Nondecreasing Sublinear Speedup Functions

We now consider the class of schedulers that are not allowed any preemptions. We present the result that the $p/\sqrt{n}$-scheduler achieves a competitive ratio of $\sqrt{n}$ for every job set with nondecreasing-sublinear speedup functions. Recall that the $p/\sqrt{n}$-scheduler 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.

6.1 Upper Bound

Theorem 6.1 *The $p/\sqrt{n}$-scheduler S performs no preemptions and has a competitive ratio of $(3+\sqrt{3}) \cdot \sqrt{n}$ for every job set with nondecreasing sublinear speedup functions.*

Proof of Theorem 6.1: Denote by S the schedule resulting from the $p/\sqrt{n}$-scheduler. Let c be the last time under S during which all $\sqrt{n}$ groups of processors are executing a job and let c_i be the completion time of job J_i. Then $F(S) = \sum_i c_i = n \cdot c + \sum_{i:c_i>c}(c_i - c)$. Note that no job starts executing after time c. Otherwise that job would have started at time c with the idle group of processors. Hence, if $c_i > c$, then job J_i is executing under S with $p/\sqrt{n}$ processors, during the entire time period $[c, c_i]$. Because the speedup functions are nondecreasing and sublinear, we know that under the optimal schedule OPT, job J_i, does not complete at a rate more than a factor of $\sqrt{n}$ times faster than that under S, even if given all p processors. Hence, the completion time of job J_i under OPT is at least $(c_i - c)/\sqrt{n}$. Thus, $F(S) = \sum_i c_i = n \cdot c + \sum_{i:c_i>c}(c_i - c) \leq n \cdot c + \sqrt{n} \times F(OPT)$. What remains to be proved is that $F(OPT) \geq \Omega(\sqrt{n} \cdot c)$. From the proof of Theorem 4.1, we have that

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

Here $S_t \cup S_t'$ is the set of $\sqrt{n}$ jobs executed at time t under S, S_t are those for which $p/\sqrt{n} < \beta_i^t$ and S_t' for which $\beta_i^t \leq p/\sqrt{n}$. Let a_t be such that $|S_t| = a_t \cdot \sqrt{n}$. (And so $|S_t'| = (1-a_t) \cdot \sqrt{n}$.) Continuing as in Section 4.2 gives

$$\begin{aligned} F(OPT) \;&\geq\; \int_0^c \left(b \cdot \sum_{i \in S_t} \pi(i) \, \frac{1}{\sqrt{n}} + (1-b) \cdot \sum_{i \in S_t'} 1 \right) dt \\ &\geq\; \int_0^c \left(b \, \frac{(a_t\sqrt{n})^2}{2} \, \frac{1}{\sqrt{n}} + (1-b)(1-a_t)\sqrt{n} \right) dt \\ &=\; \int_0^c \sqrt{n} \left(b \, \frac{a_t^2}{2} + (1-b)(1-a_t) \right) dt \end{aligned}$$

$$
\begin{aligned}
&\geq \int_0^c \sqrt{n}\left(2-\sqrt{3}\right)\, dt \\
&= (2-\sqrt{3})\sqrt{n}\cdot c.
\end{aligned}
$$

Since $F(S) \leq n \cdot c + \sqrt{n} \cdot F(OPT)$ and $F(OPT) \geq (2-\sqrt{3})\sqrt{n} \cdot c$, then we have

$$
\begin{aligned}
\frac{F(S)}{F(OPT)} &\leq \frac{n \cdot c + \sqrt{n} \cdot F(OPT)}{F(OPT)} \\
&\leq \frac{1}{2-\sqrt{3}}\sqrt{n} + \sqrt{n} \\
&= (3+\sqrt{3})\sqrt{n}.
\end{aligned}
$$

■

6.2 Lower Bound

Theorem 6.2 *For every scheduler S that never preempts, there is a set of jobs J that are fully parallelizable and for which $F(S) \geq \sqrt{n} \cdot F(OPT)$.*

Proof of Theorem 6.2: Consider such a scheduler S. The initial allocation of the processors may depend on the number of jobs n, but otherwise does not depend on the set of jobs J. Let $J' \subset J$ be the jobs to which S initially allocates processors. Suppose that S initially allocates fewer than $p/\sqrt{n}$ processors to some job $J_i \in J'$. Let the set of jobs J be such that J_i has work $W_i = p$ and the remaining jobs have essentially no work. Under S, job J_i is allocated this fixed number of processors for the duration of its computation, hence its completion time is at least $\sqrt{n}$ and $F(S) \geq \sqrt{n}$. On the other hand, for this set of jobs J, the optimal schedule initially completes the jobs that require essentially no time and then allocates all p processors to job J_i. This gives a flow time of $F(OPT) \approx 1$ and a competitive ratio of $\sqrt{n}$ for S.

Now assume that S allocates at least $p/\sqrt{n}$ processors to every job in J'. Note $|J'| \leq \sqrt{n}$. For job $J_i \in J'$, let β_i be the number of processors that S allocated to the job. The adversary sets the work W_i for the job to be $\Gamma_i(\beta_i) = \beta_i$, so that the job requires one time unit with this processor allocation. Let the remaining jobs have essentially no work. Under S, each job $J_i \in J'$ completes in time $c_i = 1$. The remaining jobs must wait one time unit for these jobs to complete and then complete in essentially no time. This gives a flow time of $F(S) = n \times 1$. A better, yet sub-optimal scheduler OPT', first completes all the jobs not in J'. These complete in essentially no time. Then OPT' completes the jobs in J' with the same processor distribution as under S. This gives a flow time of $F(OPT') \approx |J'| \times 1 \leq \sqrt{n}$ for a competitive ratio of at least $\sqrt{n}$. ■

7 Nondecreasing Sublinear or Superlinear Speedup Functions

The previous assumption that all speedup functions are sublinear is not true when the jobs are both highly parallel and have a strong time-space tradeoff. We now consider the class of such jobs. To begin we consider schedulers that are able to preempt continuously.

Theorem 7.1 *There is a non-clairvoyant algorithm $HEQUI$ such that $F(HEQUI_J) \le 2 \cdot (2 + \sqrt{3}) \cdot F(OPT_J)$ for every job set J in which each phase of each job is either nondecreasing sublinear or superlinear.*

Proof of Theorem 7.1: The scheduler $HEQUI$ (short for "Hybrid Equi-partition") slices time into units of size Δ. If there are n_t unfinished 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 on jobs with both nondecreasing-sublinear and superlinear phases without knowing which phase is currently being executed. It performs well with the nondecreasing-sublinear phases because half of the time it behaves like $EQUI$ and hence performs on these phases within a factor of two, as proved in Theorem 4.1. Superlinear phases execute the most efficiently when given all p processors. $HEQUI$ performs well on these phases because half of the time it behaves like Round-Robin.

The proof proceeds as in the proof of Theorem 4.1. 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 must now do what is done in Theorem 4.1 except we now 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 4.1 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 $\frac{p/n_t}{\Gamma_i^q(p/n_t)} \le \frac{\beta_i^t}{\Gamma_i^q(\beta_i^t)}$. We replace this with $\frac{p/n_t}{\gamma_i^q(p/n_t)} \le 2 \cdot \frac{\beta_i^t}{\Gamma_i^q(\beta_i^t)}$. 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

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

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

The second place in which the proof of Theorem 4.1 compares $\Gamma_i^q(\beta_i^t)$ and $\Gamma_i^q(p/n_t)$ is that if $\beta_i^t \le p/n_t$, then $\Gamma_i^q(\beta_i^t) \le \Gamma_i^q(p/n_t)$. The new requirement $\Gamma_i^q(\beta_i^t) \le 2 \cdot \gamma_i^q(p/n_t)$ holds because Γ_i^q is nondecreasing. Thus $\Gamma_i^q(\beta_i^t) \le \Gamma_i^q(p/n_t) \le 2 \cdot \gamma_i^q(p/n_t)$.

The remainder of the proof follows that of Theorem 4.1, except that a factor two is introduced at each step. ■

From Theorem 7.1, we know that it is possible to allow the job phases to be superlinear (not just nondecreasing sublinear), but doing so increases the competitive ratio by a factor of two. The following lower bound helps to show that this factor of two in inherent for $HEQUI$, even when the jobs are not allowed to be arbitrarily super linear, but can only be "almost" sublinear. The lower bound indicates that a scheduler needs treat the sublinear and almost sublinear job phases differently.

Theorem 7.2 *The competitive ratio of HEQUI is at least* $(2 - o(1)) \times e \approx 5.42$ *for HEQUI with nondecreasing and almost sublinear jobs.*

Note this lower bound is specific to *HEQUI*. This is why this result is not mentioned in Table 2. Adjusting the length of the Equi-partition and the Round Robin parts of *HEQUI* may lower this constant slightly, but our conjecture, is that no non-clairvoyant scheduler can do much better.

Proof of Theorem 7.2: The technique we use is the same as that for the proof of Theorem 4.3. However, the second phases of the jobs are almost sublinear instead of fully parallelizable. That is, their speedup function is $\Gamma(\beta) = \beta^{1+\epsilon}$.

As before, we define $t_1 = 0$, $s_1 = 1$, $t_i = t_{i-1} + s_{i-1}$, $s_i = 1 - t_i/n$, and $W_i^1 = t_i$. However, the work, W_i^2, of the second phase is changed from $p \cdot s_i$ to $p^{1+\epsilon} \cdot s_i$ so that s_i is still the time to complete the phase when allocated p processors. It follows that the flow time of the optimal algorithm is still $\sum_{i=1}^{n}(t_i + s_i)$.

To prove that the competitive ratio here is twice that in Theorem 4.3, we need only show that the flow time under *HEQUI* for this set of jobs is effectively twice that under *EQUI* for the jobs from Theorem 4.3. The key point is that under *HEQUI*, the first phases of the jobs computes competitively only during the Equi-partition part of *HEQUI*, and that the second phases of jobs do so only during the Round Robin part. Each of these scheduling parts last only for half the time slices.

The first phase of job i makes progress at a rate of 1 whenever it is allocated a nonzero number of processors. However, under *HEQUI*, the job is allocated nonzero processors only during $\frac{1}{2} + \frac{1}{2} \cdot \frac{1}{n}$ fraction of time. Hence, the first phase takes $2 - o(1) \cdot t_i$ time to complete.

The second phase makes progress at a rate of $\frac{1}{2}\Gamma(p/n) + \frac{1}{2n}\Gamma(p) = \frac{1}{2}(p/n)^{1+\epsilon} + \frac{1}{2n}p^{1+\epsilon} = \frac{1}{2}(1 + 1/n^{\epsilon})\, p^{1+\epsilon}/n$. Hence, the work $W_i^2 = p^{1+\epsilon} \cdot s_i$ requires $(2 - o(1)) \cdot s_i \cdot n$ time to complete.

HEQUI takes $2t_i + 2ns_i$ time to finish job J_i, while *EQUI* takes only $t_i + ns_i$ time. This doubles the flow time and the competitive ratio. ■

8 Almost Sublinear Speedup Functions with n or Zero Preemptions

We have seen that non-clairvoyant schedulers can achieve a constant competitive ratio on jobs with superlinear speedup functions by preempting continuously. We now show that if the scheduler is only allowed to preempt a bounded number of times then it cannot perform well.

We have seen that *EQUI* performs competitively when the speedup functions are sublinear and nondecreasing. In this section, we see that weakening the sublinear requirement only slightly, to *almost* sublinear, has a significant effect on the competitive ratio. Recall that in order to include the speedup function $\Gamma(\beta) = \beta^{1+\epsilon}$, we defined almost sublinear to mean that $\beta_1^{1+\epsilon}/\Gamma(\beta_1) \leq \beta_2^{1+\epsilon}/\Gamma(\beta_2)$ whenever $\beta_1 \leq \beta_2$.

8.1 Lower Bound

Theorem 8.1 *Let $\epsilon \geq 0$. Suppose that S is a scheduler that preempts only a bounded number of times. Suppose as well that all the jobs have speedup functions $\Gamma(\beta) = \beta^{1+\epsilon}$ (known by the scheduler), but the scheduler does not know the work for each job. There is a set of jobs J for which $F(S_J)/F(OPT_J) \geq 2 \cdot n^\epsilon \cdot \frac{n}{n+1}$ when $0 \leq \epsilon < 1$ and $F(S_J)/F(OPT_J) \geq n$ when $\epsilon \geq 1$.*

Note that for $\epsilon = 0$, the jobs have speedup functions $\Gamma(\beta) = \beta$ and the competitive ratio is 2.

Proof of Theorem 8.1: As done in the proof of Theorem 4.3, first consider a non-clairvoyant scheduler S that does not preempt at least until some job completes. To account for schedulers that preempt n, 10^{10^n}, or $Ackermann(n, n)$ times, we use the same technique used there. We scale down the work so all the jobs complete before the first preemption.

Suppose the scheduler initially allocates β_i processors to job J_i, where $\sum_i \beta_i \leq p$. The adversary sets the work for job J_i to be $W_i = \Gamma(\beta_i) = \beta_i^{1+\epsilon}$ so that its completion time is $c_i = 1$. (If $\beta_i = 0$, then W_i is set to an infinitesimally small value. Note that this job does not complete until a preemption occurs.) Given these work levels, the flow time is $F(S) = \sum_i c_i = n$. Note that no preemption occurs until time 1, because no job completes until this time.

The optimal scheduler allocates all p processors to each job in shortest work first order. Without loss of generality, assume that $\beta_i \geq \beta_{i+1}$. Then the flow time is $F(OPT) = \sum_i i \cdot W_i/\Gamma(p) = \sum_i i \cdot (\beta_i)^{1+\epsilon}/p^{1+\epsilon} = \sum_i i \cdot (q_i)^{1+\epsilon}$ where $q_i = \beta_i/p$ and $\sum_i q_i = 1$. When $0 \leq \epsilon < 1$, Claim 1 below proves that this is maximized when all q_i have the same value. This gives $F(OPT) \leq \sum_i i \cdot (1/n)^{1+\epsilon} \leq \frac{n+1}{2n^\epsilon}$ and a competitive ratio of $2 \cdot n^\epsilon \frac{n}{n+1}$. Note that for $0 \leq \epsilon < 1$, the competitive ratio of Equi-partition is within a constant factor of this ratio.

When $\epsilon \geq 1$, Claim 1 proves that $\sum_i i \cdot (q_i)^{1+\epsilon}$ is maximized when $q_1 = 1$ and the other $q_i = 0$. This gives $F(OPT) \leq 1$ and a competitive ratio of n. ■

Claim 1 *Subject to $\sum_{i\in[1..n]} q_i = 1$ and to $q_i \geq q_{i+1} \geq 0$ for all $i \in [1..n-1]$, the value of $F = \sum_{i\in[1..n]} i \cdot (q_i)^{1+\epsilon}$ is maximized for $0 \leq \epsilon < 1$ when all the q_i have the same value and for $\epsilon \geq 1$ when $q_1 = 1$ and the other $q_i = 0$.*

Proof of Claim 1: Let $q_1, \ldots, q_n$ be values that maximize the objective function F subject to the constraints. The first step is to prove that this list of values q_i does not have more than one distinct non-zero value. By way of contradiction, assume that q and q' are two non-zero values that the q_i have (and where $q < q'$). We will find a contradiction by finding a way of changing the q_i values in a way that honors the constraints and increases the objective function F.

Let $A = \{i \mid q_i = q\}$ and $B = \{i \mid q_i = q'\}$ be the sets of indices with these values and $a = |A|$ and $b = |B|$ be the numbers of such values. We will change the values of q_i for $i \in A$ from q to $q + \delta/a$ and for $i \in B$ from q' to $q' - \delta/b$. Clearly the constraint $\sum_{i\in[1..n]} q_i = 1$ is maintained. In addition, because q and q' are both non-zero, these q_i can be decreased without violating $q_i \geq 0$. Because all the q_i with these values are being changed, they can be increased or decreased slightly before violating the order constraint $q_i \geq q_{i+1}$. Therefore, this change respects the constraints for both positive and negative sufficiently small δ. Hence, F (with this change) is maximized at $\delta = 0$ only if the second derivative of F with respect to δ is negative. However, it is strictly positive.

$$\frac{d^2F}{d^2\delta} = (1+\epsilon)(\epsilon)(1/a)^2 \cdot i \cdot \sum_{i\in A}(q + \delta/a)^{-1+\epsilon} + (1+\epsilon)(\epsilon)(-1/b)^2 \cdot i \cdot \sum_{i\in B}(q' + \delta/b)^{-1+\epsilon} > 0.$$

Because this list of values q_i does not have more than one non-zero value, because the sequence is sorted in decreasing order, and because $\sum_{i\in[1..n]} q_i = 1$, it follows that there is some value b such that $q_1 = \ldots = q_b = 1/b$ and $q_{b+1} = \ldots = q_n = 0$. This gives $F = \sum_{i\in[1..b]} i \cdot (1/b)^{1+\epsilon} = \frac{b+1}{2b^\epsilon}$. If $0 \leq \epsilon < 1$, this is maximized with $b = n$. If $\epsilon \geq 1$, then is maximized with $b = 1$. ■

8.2 Upper Bound

For completion, we give a matching upper bound to the above lower bound.

Theorem 8.2 *For any $\epsilon \geq 0$ and for any job set J with nondecreasing and almost sublinear speedup functions, $F(EQUI_J)/F(OPT_J) \in O(n^\epsilon)$.*

Proof of Theorem 8.2: The proof is the same as that in Theorem 4.1 except for the relaxation of the sublinear condition. In the proof of Theorem 4.1, when $p/n_t \geq \beta_t$, $EQUI$ performs at least as well as in the proof. When $p/n_t < \beta_i^q$, we had that $\frac{p/n_t}{\Gamma_i^q(p/n_t)} \leq \frac{\beta_i^q}{\Gamma_i^q(\beta_i^q)}$. Now we have that $\frac{p/n_t}{\Gamma_i^q(p/n_t)} \leq \left(\frac{\beta_i^q}{p/n_t}\right)^\epsilon \cdot \frac{\beta_i^q}{\Gamma_i^q(\beta_i^q)}$. However, this additional factor is at most n^ϵ because $\beta_i^q \leq p$ and $n_t \leq n$. The rest of the proof is the same as the proof of Theorem 4.1, except with this additional factor everywhere. ■

The reader may create his or her own definition of "almost" sublinear, which can be substituted into this proof at the appropriate points.

8.3 Zero Preemptions

Now consider the situation in which the scheduler is allowed zero preemptions. Recall the upper and lower bounds from Theorems 6.1 and 6.2 that prove that the competitive ratio is $\Theta(\sqrt{n})$. These can easily be modified to prove that the competitive ratio is $\Theta\left(n^{\frac{1+\epsilon}{2}}\right)$ when the jobs have almost sublinear nondecreasing speedup functions.

9 Nondecreasing Speedup Functions

In practice the assumption that all speedup functions are either nondecreasing and sublinear or are superlinear is not always true. we require the speedup functions to be merely nondecreasing (i.e., if $\beta_1 \leq \beta_2$, then $\Gamma(\beta_1) \leq \Gamma(\beta_2)$). Again we must consider schedulers that are allowed to preempt continuously.

9.1 Upper Bound

Theorem 9.1 *There is a non-clairvoyant algorithm $HEQUI'$ using continuous preemptions 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 9.1: Suppose that under $HEQUI'$ there are n_t unfinished jobs remaining at time t. 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 $\frac{\Delta}{\log(n_t)n_t\beta/p}$.

We now continue as in Theorem 4.1. Our proof here changes that of Theorem 4.1 in the same way as was done for Theorem 7.1. 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 $\frac{p/n_t}{\gamma_i^q(p/n_t)} \le 2\log n \cdot \frac{\beta_i^t}{\Gamma_i^q(\beta_i^t)}$. 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 $p/n_t < \beta_i^t \le \beta$. Therefore, the effective rate of $HEQUI'$ is $\gamma_i^q(p/n_t) \ge \frac{1}{\log(n_t)} \frac{p}{n_t\beta}\Gamma_i^q(\beta)$. Because the speedup functions are nondecreasing and $\beta_i^t \le \beta$, we know that $\Gamma_i^q(\beta_i^t) \le \Gamma_i^q(\beta)$. Also, β is within a factor of two of β_i^t. Thus $\gamma_i^q(p/n_t) \ge \frac{1}{2\log n} \frac{p}{n_t\beta_i^t}\Gamma_i^q(\beta_i^t)$. Rearranging this formula gives the first required statement.

The second required statement is that if $\beta_i^t \le p/n_t$, then $\Gamma_i^q(\beta_i^t) \le 2\log n \cdot \gamma_i^q(p/n_t)$. If $\beta_i^t \le p/n_t$, 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 $\gamma_i^q(p/n_t) \ge \frac{1}{\log(n_t)} \frac{p}{n_t\beta}\Gamma_i^q(\beta) \ge \frac{1}{2\log n}\Gamma_i^q(\beta_i^t)$. The last inequality again uses the fact that the speedup function is nondecreasing.

The remainder of the proof follows that of Theorem 4.1, except that a factor of $2\log n$ is introduced at each step. ■

9.2 Lower Bound

Theorem 9.2 *For any non-clairvoyant scheduler S that is allowed continuous preemptions, there is a set of jobs J all with nondecreasing speedup functions for which $F(S_J)/F(OPT_J) \in \Omega(\log n)$.*

Proof of Theorem 9.2: For each integer $k \in [0..\log n]$ define the speedup function Γ_k to be $\Gamma_k(\beta) = 1$ for $\beta \ge \frac{p}{n}2^k$ and effectively zero for $\beta < \frac{p}{n}2^k$. (See Figure 1:i.) Every job in J will have one such speedup function.

Consider the allocation of processors under scheduler S for the first time unit assuming that during this time period no jobs have completed. Note that this allocation does not depend on the job set J, because the scheduler gains no information about the job set until some job completes. For each job J_i and each integer $j \in [0..\log n]$, define t_i^j to be the amount of this first unit of time during which the number of processors S allocates to job J_i is in the interval $[\frac{p}{n}2^j, \frac{p}{n}2^{j+1})$. Define $s_i = \sum_{j=0}^{\log n} t_i^j \cdot \frac{p}{n}2^j$. This is a lower bound on the processor area consumed by job J_i. Note that the total processor area consumed during this first time unit is at most $1 \cdot p$. It follows that $\sum_i s_i \le p$.

Consider job J_i. To help the adversary choose which speedup function Γ_k to give this job, define $s_i'(k) = \frac{p}{n}2^k \cdot \sum_{j=k}^{\log n} t_i^j$. Later we will see that if Γ_k were to be chosen, then this would be the processor area consumed by job J_i under a scheduler OPT' to complete the same work. Set k_i such that $k \in [0..\log n]$ that minimizes $s_i'(k)$. (For example, if it were the case that $t_i^j = s_i/(\frac{p}{n}2^j \log n)$, then $s_i = \sum_{j=0}^{\log n} t_i^j \cdot \frac{p}{n}2^j$ and for all k, $s_i'(k) = \frac{p}{n}2^k \cdot \sum_{j=k}^{\log n} s_i/(\frac{p}{n}2^j \log n) \le \sum_{l\ge 0} s_i/(2^l \log n) = 2s_i/\log n$.) Claim 2 proves that in general there exists a k_i for which $s_i'(k_i) \le 2s_i/\log n$. Therefore, for this choice of k, the processor area consumed by job J_i under OPT' is a factor of $\Omega(\log n)$ less than that under S. The adversary sets the speedup function of job J_i to be Γ_{k_i} and its work to be $W_i = s'_i(k_i)/(\frac{p}{n}2^{k_i})$ (plus some infinitesimal amount).

Consider the work completed by job J_i under S during this first time unit. For every point in time in which it is allocated at least $\frac{p}{n}2^{k_i}$ processors, work is executing at a rate of one. Otherwise, the rate is effectively zero. Hence, the work that is accomplished is $\sum_{j=k_i}^{\log n} t_i^j = s'_i(k_i)/(\frac{p}{n}2^{k_i}) = W_i$ (minus the infinitesimal amount). Hence, job J_i does not complete until after the end of this first time unit. This gives a flow time of $F(S) \geq n$.

Now consider the following good but possibly suboptimal scheduler OPT'. To each job J_i, the scheduler assigns a group of $\frac{p}{n}2^{k_i}$ processors until J_i completes. However, a processor may be assigned to many jobs. Hence, a job J_i is not allocated any processors until all the processors in its group have completed all the jobs that have a higher priority than J_i (according to a given priority scheme). Define $\pi(i)$ (not a permutation) to be the number of jobs that have a lower priority than job J_i and that are assigned processors from job J_i's processor group. These are the jobs that must wait for job J_i to complete before starting their computation. Define h_i to be the running time of job J_i once allocated processors. It follows that the flow time is $F(OPT') = \sum_{i=1}^{n} \pi(i)h_i$.

Because job J_i has the speedup function Γ_{k_i} and because it is allocated $\frac{p}{n}2^{k_i}$ processors, work executes at a rate of one. Hence, the running time h_i of this job is $W_i = s'_i(k_i)/(\frac{p}{n}2^{k_i}) \leq 2s_i/(\frac{p}{n}2^{k_i}\log n)$. Below we prove that $\pi(i) \leq 2 \cdot 2^{k_i}$. This gives a flow time of $F(OPT') = \sum_{i=1}^{n} \pi(i)h_i \leq \sum_{i=1}^{n}(2 \cdot 2^{k_i})(2s_i/(\frac{p}{n}2^{k_i}\log n)) = \frac{4}{\log n}\frac{n}{p}\sum_i s_i$. Because $\sum_{i=1}^{n} s_i \leq p$, it follows that $F(OPT') \in O(n/\log n)$. Recall that $F(S) \geq n$. Hence, the competitive ratio is $\Omega(\log n)$.

What remains is to fill in the details of the schedule OPT' and to prove for each job J_i that $\pi(i) \leq 2 \cdot 2^{k_i}$. The jobs with speedup function $\Gamma_{\log n}$ (i.e., those requiring all p processors) are given the highest priority and hence are executed first. Then those with $\Gamma_{(\log n)-1}$. Finally, the jobs with speedup function Γ_0 (i.e., those requiring only p/n processors) are given the lowest priority and hence are executed last. For each integer $k \in [0..\log n]$, partition the processors into $n/2^k$ groups of $\frac{p}{n}2^k$ processors each. (Without loss of generality, assume that p and n are both powers of two so that these groups divide evenly.) Define n_k to be the number of jobs J_i for which the speedup function is Γ_k. (Note that $\sum_{k=0}^{\log n} n_k = n$.) Assign $\lfloor \frac{n_k}{n}2^k \rfloor$ jobs with speedup function Γ_k to each group in an arbitrary priority order. (If $\frac{n_k}{n}2^k$ is not an integer, then assign the remaining jobs, one per group, in groups with evenly spaced indices.) Now consider $\pi(i)$, the number of jobs that have a lower priority than job J_i and that are assigned processors from job J_i's processor group. For $j \leq k_i$, there are $2^{k_i}/2^j$ groups of $\frac{p}{n}2^j$ processors that are subsets of J_i's processor group. There are about $\frac{n_j}{n}2^j$ jobs with speedup function Γ_j per group for a total of $\lceil (2^{k_i}/2^j) \cdot (\frac{n_j}{n}2^j) \rceil \leq \frac{n_j}{n}2^{k_i} + 1$ such jobs waiting for J_i. Hence, the total number of jobs waiting on J_i is $\pi(i) \leq \sum_{j \leq k_i}(\frac{n_j}{n}2^{k_i} + 1) \leq 2^{k_i} + k_i \leq 2 \cdot 2^{k_i}$. ■

Claim 2 *Let q_j be such that $\sum_{j=0}^{r} q_j = 1$. There exists an integer $k \in [0, r]$ for which $\sum_{j=k}^{r} q_j/2^{j-k} \leq 2/(r+2)$.*

In our application, $r = \log n$ and $q_j = t_i^j \cdot \frac{p}{n}2^j/s_i$. The requirement that $\sum_{j=0}^{\log n} q_j = 1$ ensures that $s_i = \sum_{j=0}^{\log n} t_i^j \cdot \frac{p}{n}2^j$. Then we choose k to minimize $s'_i(k) = \frac{p}{n}2^k \cdot \sum_{j=k}^{\log n} t_i^j$ which is a factor of s_i bigger than what is in the claim. The claim is only strengthened by allowing the q_j to be negative.

Proof of Claim 2: Define $s(k)$ to be $\sum_{j=k}^{r} q_j/2^{j-k}$. Let the q_j have values that maximize $\text{Min}_k s(k)$ subject to $\sum_{j=0}^{r} q_j = 1$. The first thing to prove is that for such values of q_j, the $s(k)$

has some fixed value s independent of the choice of k. Suppose by contradiction, there is a value k' such that $s(k') > s(k)$ for all $k \neq k'$. Then find values δ_j for $j \in [0..r]$ subject to the following $r+1$ linear constraints. The first constraint is that $\sum_{j=0}^{r} \delta_j = 0$. Then for each $k \neq k'$, include the constraint that $\sum_{j=k}^{r} \delta_j / 2^{j-k} = \delta$. It is not hard to see that these constraints are linearly independent and hence are satisfiable. Then we change the values of each q_j by adding on δ_j. The first constraint maintains the fact that $\sum_j q_j = 1$. The kth constraint increases $s(k)$ by δ. This change will increase $\mathrm{Min}_k s(k)$ as long as the change does not decrease $s(k')$ by too much. However, if δ is sufficiently small this will not happen. Then this contradicts the fact that the q_j were chosen to maximize this minimum.

Using the fact that $s(k) = s$ for all k and that $\sum_{j=0}^{r} q_j = 1$, we can solve that $q_r = s$, $q_j = s/2$ for $j \neq r$, and $s = 2/(r+2)$. ■

10 Gradual Speedup Functions

Even the assumption that all the speedup functions are nondecreasing may not be true in practice. In this section, we only require the speedup functions to be gradual (i.e., there is a constant $c > 0$ such that for every number of processors β and for every factor $a \in [1..2]$ either $\Gamma(a\beta/2) \geq \frac{1}{c}\Gamma(\beta)$ or $\Gamma(a\beta) \geq \frac{1}{c}\Gamma(\beta)$).

10.1 Upper Bound

Theorem 10.1 *There is a non-clairvoyant algorithm $HEQUI''$ such that $F(HEQUI''_J) \leq O(\log p) \cdot F(OPT)$ for every job set J, where each phase of each job has a gradual speedup function.*

Proof of Theorem 10.1: The scheduler $HEQUI''$ is the same as $HEQUI'$ defined in the proof of Theorem 9.1 except that it considers every number of processors β that is a power of two between 1 and p, not just those between p/n_t and p. Note that when $\beta > p/n_t$, $HEQUI''$ cannot execute all the jobs at once, but must do them in $n_t\beta/p$ batches. On the other hand, if $\beta \leq p/n_t$, then all the jobs can be executed at once. Hence, the effective rate of $HEQUI''$ is $\gamma_i^q(p/n_t) = \sum_{\beta=2^k \in [1,p/n_t]} \frac{1}{\log p}\Gamma_i^q(\beta) + \sum_{\beta=2^k \in [p/n_t,p]} \frac{1}{\log p}\frac{p}{n_t\beta}\Gamma_i^q(\beta)$. As in the proof of Theorem 9.1, there are two statements to be proved.

No matter how many processors β_i^t OPT uses at time t, there exists a factor $a \in [1..2]$ such that both $a\beta_i^t/2$ and $a\beta_i^t$ are powers of two (unless $\beta_i^t < 1$, but we do not allow the optimal to take advantage of peaks in the rate for $\beta < 1$). Because the speedup functions are gradual, we know that either $\Gamma(a\beta_i^t/2) \geq \frac{1}{c}\Gamma(\beta_i^t)$ or $\Gamma(a\beta_i^t) \geq \frac{1}{c}\Gamma(\beta_i^t)$. Let β be the value for which this is true. Note that β is within a factor of two of β_i^t.

If $p/n_t \leq \beta_i^t$, then $\gamma_i^q(p/n_t) \geq \frac{1}{\log p}\frac{p}{n_t\beta}\Gamma_i^q(\beta) \geq \frac{1}{2c\log p}\frac{p}{n_t\beta_i^t}\Gamma_i^q(\beta_i^t)$, as is required in the first statement. If $\beta_i^t \leq p/n_t$, then $\gamma_i^q(p/n_t) \geq \frac{1}{\log p}\Gamma_i^q(\beta) \geq \frac{1}{c\log p}\Gamma_i^q(\beta_i^t)$. The remainder of the proof follows that of Theorem 4.1, except that a factor $2c\log p$ is introduced at each step. ■

10.2 Lower Bound

Theorem 10.2 *Consider a non-clairvoyant scheduler S that is allowed continuous preemptions. There is a set of jobs J all with gradual speedup functions for which $F(S_J)/F(OPT_J) \geq \Omega(\log p)$.*

Proof of Theorem 10.2: Every nondecreasing speedup function is gradual. Hence, the lower bound of $\Omega(\log n)$ on the competitive ratio from Theorem 9.2 applies here directly. If $\log n \geq \frac{1}{2}\log p$, then the proof is complete. Otherwise, because $\log n + \log(p/n) = \log p$, it follows that $\log(p/n) \geq \frac{1}{2}\log p$. Therefore, it suffices to prove a lower bound of $\Omega(\log(p/n))$ for the case when $\log n < \frac{1}{2}\log p$.

For each integer $k \in [0..\log(p/n)]$, define the speedup function Γ_k to be $\Gamma_k(\beta) = 1$ for $\beta \in [2^k, 2^{k+1})$ and effectively zero elsewhere. (See Figure 1:k.) As in the proof of Theorem 9.2, consider the allocation of processors under scheduler S for the first time unit assuming that during this time period no jobs have completed. For each job J_i and each $k \in [0..\log(p/n)]$, define t_i^k to be the amount of this first unit of time during which the number of processors allocated to job J_i is in the interval $[2^k, 2^{k+1})$. Let k_i be the index $k \in [0, \log(p/n)]$ for which t_i^k is the smallest. Because $\sum_{k \in [0,\log(p/n)]} t_i^j = 1$, we know $t_i^{k_i} \leq 1/\log(p/n)$. The adversary sets the speedup function of job J_i to be Γ_{k_i} and its work to be $W_i = t_i^{k_i}$ (plus an infinitesimal amount). Hence, job J_i does not complete under S until after the end of this first time unit, giving a flow time of $F(S_J) \geq n$. The optimal scheduler, on the other hand, allocates 2^{k_i} processors to job J_i. Note that all n jobs can be simultaneously allocated its respective number of processors, because no job is allocated more than $\frac{p}{n}$ processors. The completion time of job J_i is $t_i^{k_i}$, the flow time is $F(OPT) = \sum_{i=1}^{n} t_i^{k_i} \leq n/\log(p/n)$, and the competitive ratio is at least $\log(p/n)$. ■

11 Integer Domain and Arbitrary

If we are unable to assume that all the speedup functions are gradual, then we should hope that they are in the integer domain job class (i.e., the speedup functions do not change by more than a constant factor between integer numbers of processors, $\Gamma(\lfloor\beta\rfloor) \geq \frac{1}{c}\Gamma(\beta)$).

Theorem 11.1 *There is a non-clairvoyant algorithm $HEQUI'''$ such that $F(HEQUI_J''') \leq O(p) \cdot F(OPT)$ for every job set J where each phase of each job has an integer domain speedup function.*

Proof of Theorem 11.1: The proof is the same as that of Theorem 10.1 except the scheduler $HEQUI'''$ considers every integer number of processors 1 and p, not just powers of two. ■

Theorem 11.2 *Consider a non-clairvoyant scheduler S that is allowed continuous preemptions. There is a set of jobs J with all integer domain speedup functions for which $F(S_J)/F(OPT_J) \geq \Omega(p)$.*

Proof of Theorem 11.2: The proof is the same as that of Theorem 10.2 except a speedup function is considered for each integer $k \in [1..p]$. (See Figure 1:l.) ■

Theorem 11.3 *Consider a non-clairvoyant scheduler S that allows continuous preemptions. There is a set of jobs J with arbitrary speedup functions for which the competitive ratio is arbitrarily bad.*

Proof of Theorem 11.3: For the jobs considered, there need only be one real number of processors β at which substantial progress is made. (See Figure 1:m.) A non-clairvoyant scheduler has no hope of running the job for a non-infinitesimal amount of time at this exact number of processors. ■

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

Many of the previous upper bounds require the scheduler to preempt continuously. Again let us consider the situation in which the number of preemptions is restricted. As long as the jobs have nondecreasing speedup functions, a scheduler is able to achieve a competitive ratio of $O(n)$ (Theorem 12.1). The matching lower bound is from (Theorem 8.1). On the other hand, if the jobs are allowed to have gradual speedup functions, then every non-clairvoyant scheduler that is not allowed to preempt continually can have an arbitrarily bad competitive ratio.

Theorem 12.1 *There is a simple scheduler that does no preemptions and has a competitive ratio of at most n for every set of jobs with nondecreasing speedup functions.*

Proof of Theorem 12.1: The scheduler S runs the n jobs one at a time with all p processors starting the next job when the previous job completes. Let h_i be the length of time that job J_i is run with all p processors under S. The flow time is $F(S) = \sum_i ih_i$. Let c_i be the completion time of job J_i under OPT. Because the speedup functions are nondecreasing, the jobs under S (when executing) always execute at a rate that is at least as fast as that under OPT. Hence, $h_i \leq c_i$ and $F(S) \leq \sum_i ic_i$. Subject to the constraint that $F(OPT) = \sum_i c_i$, this is maximized when $c_n = F(OPT)$ and $c_i = 0$ for $i < n$. This gives $F(S) \leq n \cdot F(OPT)$. ■

13 Conclusions and Open Problems

In order to better understand the relationship between jobs of a particular class, preemption cost, and scheduling algorithm performance, 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. For nondecreasing sublinear jobs, we have provided a complete tradeoff between the competitive ratio and the number of preemptions allowed, where the number of preemptions is bounded. In particular, for nondecreasing sublinear jobs, Equi-partition performs within a constant factor of optimal when it is allowed at least $\log n$ preemptions.

Besides tightening the bounds in Figure 2, 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? That is, how much of an advantage does the optimal algorithm have over the online algorithm because of its ability to preempt an unlimited number of times (and hence is able to recompute a new distribution of processors to the jobs)?

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