Competitively Scheduling Tasks with Intermediate Parallelizability

Sungjin Im
Electrical Engineering and
Computer Science
University of California
Merced, CA 95344
sim3@ucmerced.edu

Benjamin Moseley
Toyota Technological Institute
Chicago, IL 60637
moseley@ttic.edu

Kirk Pruhs
Dept. of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260
kirk@cs.pitt.edu

Eric Torng
Dept. of Computer Science
and Engineering
Michigan State University
East Lansing, MI 48824
torng@msu.edu

ABSTRACT

We introduce a scheduling algorithm Intermediate-SRPT, and show that it is $O(\log P)$-competitive with respect to average waiting time when scheduling jobs whose parallelizability is intermediate between being fully parallelizable and sequential. Here the parameter P denotes the ratio between the maximum job size to the minimum. We also show a general matching lower bound on the competitive ratio. Our analysis builds on an interesting combination of potential function and local competitiveness arguments.

Categories and Subject Descriptors

F.2.2 [**Nonnumerical Algorithms and Problem**]: Sequencing and scheduling

General Terms

Algorithms, Theory

Keywords

Scheduling, Parallelization, Speedup curves.

1. INTRODUCTION

Due to the effects of Moore's law, around a decade ago chip makers such as Intel hit a thermal wall, where the cost of cooling became prohibitive if all switches were devoted to a single high speed processor. In response the chip makers abruptly switched to predominantly producing multiprocessor chips [11]. The advantage of multiprocessor chips is that k processors with speed s/k would use only about $1/k^2$ fraction of the dynamic power of a single speed s processor (assuming the standard cube-root rule relationship between dynamic power and speed), but potentially would have the same processing capability; of course, fully utilizing the processing capability of a multiprocessor is a grand challenge. Our focus here is on one of these challenges, namely the scheduling of tasks.

One (not universally accepted) vision of the future is articulated by Anant Agarwal, CEO of Tilera [12].:

> "I would like to call it a corollary of Moore's Law that the number of cores will double every 18 months."

Tilera currently produces products with order of 10^2 processors [1], and products with order of 10^3 processors are in research and development [2]. In such settings, there will likely often be more processors than tasks, and thus a scheduler would have to partition the processors among the tasks. To achieve optimal performance, the scheduler must consider the parellizability of tasks when partitioning and scheduling. For example, whereas some highly parallel tasks might be sped up almost linearly when assigned additional processors and thus benefit greatly from being given more processors, other highly sequential tasks might not be sped up at all when assigned additional processors. In between these two extremes lie perhaps the majority of tasks which have intermediate levels of parallelizability.

The initial motivating questions for the research that we report on here are:

> What is the best algorithm to schedule jobs with intermediate parallelizability, and what worst-case relative error guarantee does this algorithm give?

Under standard assumptions (which we will elaborate on momentarily) it is clear how to optimally schedule n fully parallelizable tasks on m processors: all m processors are allocated to the task with the least amount of unprocessed work. Let us call this algorithm Parallel-SRPT. Further it is known how to schedule n fully sequential tasks on m processors in an optimally competitive way: the up to m

SPAA'14, June 23–25, 2014, Prague, Czech Republic.
Copyright 2014 ACM 978-1-4503-2821-0/14/06 ...$15.00.
http://dx.doi.org/10.1145/2612669.2612682.

tasks with the least unprocessed work are each allocated one processor. Let us call this algorithm Sequential-SRPT. For sequential jobs with sizes are between 1 and P, Sequential-SRPT is $O(\log P)$-competitive, with respect to the objective of average waiting time [10]. Further, this competitive ratio is best possible for online algorithms [10].

It was previously not known how to schedule jobs of intermediate parallelizability in an optimally competitive way, and it was not clear a priori what the best scheduling policy would be. Presumably the "right" algorithm should agree with Parallel-SRPT when jobs are fully parallelizable, and agree with Sequential-SRPT when the jobs are sequential. After a moment's reflection, the most obvious property that both Parallel-SRPT and Sequential-SRPT share is that they schedule jobs in such a way as to maximize the rate of reduction of the fractional number of unfinished jobs, under the assumption that the original size of each job was its current size. So perhaps the most natural candidate for the best algorithm to schedule tasks with intermediate parallelizability would again be to assume that the remaining unfinished work of each job was its original work, and then greedily maximize the rate that the fractional number of unfinished jobs is being reduced by. Quite surprisingly (at least to us) we show in Section 3 that the competitive ratio of this natural hybrid algorithm is large.

Our main result is a less obvious algorithm (though still simple and natural), which we call Intermediate-SRPT, that we show is optimally competitive for all intermediate levels of parallelizability. We now describe the Intermediate-SRPT algorithm, introduce our natural model for intermediate parallelizability, state our upper bound on the competitive ratio of the Intermediate-SRPT algorithm, and finally state our general matching lower bound on the competitive ratio of any algorithm.

Intermediate-SRPT Algorithm Description: If there are at least m tasks, the m tasks with the least unprocessed work are each allocated one processor (this is like Sequential-SRPT). If there are strictly fewer than m tasks, the processors are evenly partitioned among the tasks (this is essentially the Round Robin or Processor Sharing Algorithm).

Modeling Intermediate Parallelizability: We assume that for each job j there exists an $\alpha_j \in (0, 1)$ such that the speedup curve for job j is $\Gamma_j(x) = x$ for $x \leq 1$, and $\Gamma_j(x) = x^{\alpha_j}$ for $x \geq 1$. The speedup curve gives the rate that work is processed if the job is allocated x processors. Note that $\alpha_j = 0$ corresponds to a sequential job and $\alpha_j = 1$ corresponds to a fully parallelizable job. We believe speed-up curves of the form x^{α_j} give a natural way of interpolating intermediate degrees of parallelizability without being grounded in any specific machine model.

Theorem 1. *For jobs of intermediate parallelizability, the algorithm Intermediate-SRPT has competitive ratio* $O(1) \cdot 4^{1/(1-\alpha)} \log P$ *with respect to average waiting time, where* $\alpha = \max_j \alpha_j$*. In particular, this holds for the special case that each* $\alpha_j = \alpha$.

Theorem 2. *For all* $\alpha \in [0, 1)$*, the competitive ratio of every algorithm with respect to average waiting time, restricted to instances with tasks with speedup curves of the form* $\Gamma_j(x) = x$ *for* $x \leq 1$*, and* $\Gamma_j(x) = x^{\alpha}$ *for* $x \geq 1$*, is* $\Omega(\log P)$.

Taken together, these results show (again somewhat surprisingly to us) that scheduling jobs that are even slightly less than fully parallelizable is more like scheduling sequential jobs than like scheduling fully parallelizable jobs. The lower bound for the natural hybrid algorithm shows that its "error" is that it will sometimes allocate too many processors to one job. This is the right strategy if the jobs are fully parallelizable, but can lead to a large relative error if the jobs are even a bit less than fully parallelizable. The algorithm Intermediate-SRPT corrects this error by functioning as Sequential-SRPT when the system is underloaded, and sharing the processors equally when the system is overloaded. Theorem 1 and Theorem 2 together establish that the optimal competitive ratio jumps from 1 to $\Theta(\log P)$ the instant $\alpha < 1$.

Theorem 1 is proved in Section 2. Theorem 2, is proved in Section 4. But first, we review standard modeling assumptions and notation, and review the most closely related papers in the literature.

1.1 Standard Modeling Assumptions and Notation

There are m identical unit-speed processors. Each task/job j has three characteristics: a release time r_j when it arrives in the system, a size or work amount $p_j \in [1, P]$ specifying the amount of processing that has to be performed on job j to finish it, and a speed-up curve $\Gamma_j(x)$ specifying the rate at which work on job j is processed if assigned x processors. A job is fully parallelizable if $\Gamma_j(x) = x$, and is sequential if $\Gamma_j(x) = x$ for $x \leq 1$, and $\Gamma_j(x) = 1$ for $x \geq 1$. A job j has intermediate parallelizability if there exists an $\alpha \in (0, 1)$ such that the speed-up curve for j is $\Gamma_j(x) = x$ for $x \leq 1$, and $\Gamma_j(x) = x^{\alpha}$ for $x \geq 1$. If $p_j(t)$ is the amount of unprocessed work on job j at time t, then the fractional number of jobs at time t is $\sum_j \frac{p_j(t)}{p_j}$.

The flow/waiting/response time for job j in a schedule S is $F_j^S = C_j^S - r_j$ which is the length of time between when the job is released and when the job is completed in schedule S, and the average flow/waiting/response time of the schedule is $\sum_j F_j^S / n$. Within the context of this paper, the competitive (approximation/worst-case) ratio of an online scheduling algorithm A is the maximum over all inputs I with job sizes in the range $[1, P]$, of the ratio between total flow time for the schedule produces by A on I and the optimal flow time for instance I (this is essentially just a measure of worst-case relative error).

1.2 Related Literature

The speedup curves models was introduced into the literature in [5], who showed that equally partitioning the processors among the jobs is 2-competitive for total waiting time if jobs have arbitrary speedup curves and all jobs are released at the same time.

The other standard way to measure the quality of an online scheduling algorithm, beside competitive ratio, is resource/speed augmentation analysis [9, 13]. An online algorithm A is s-speed c-competitive if for all inputs I, the cost for A on I with s-speed processors is at most c times the optimal cost for I on speed 1 processors. An algorithm is scalable if it is $(1 + \epsilon)$-speed $O(1)$-competitive for all fixed constant $\epsilon > 0$.

[4] showed that partitioning the processors equally amongst the jobs is $(2+\epsilon)$-speed $O(1)$-competitive with respect to average waiting time for jobs with arbitrary speedup curves. [6] showed that the algorithm that partitions the processors equally amongst the latest arriving jobs is scalable.

[3] gives essentially optimally competitive algorithms for scheduling jobs with arbitrary speedup curves in a setting of identical speed scalable processors where the objective is total waiting time plus energy (in this setting one essentially gets speed augmentation for free). As [3] focused on non-clairvoyant scheduling algorithms, the competitive ratios were super-constant. [7] shows a scalable algorithm is achievable when the scheduler has access to a job's parallelizability. [14, 16] give essentially optimally competitive algorithms for scheduling jobs with arbitrary speedup curves for the objective of maximum waiting time. [17] considers scheduling jobs with arbitrary speedup curves and with precedence constraints.

[10] also shows that the competitive ratio with respect to average waiting time of Sequential-SRPT is $O(\log \frac{n}{m})$, and give a general matching lower bound for all online algorithms.

There is a large literature on online scheduling. One good survey for providing background on related results is [15].

2. ANALYSIS OF INTERMEDIATE-SRPT

Our goal in this section is to prove Theorem 1, which upper bounds the competitive ratio for the Intermediate-SRPT algorithm.

2.1 Analysis Overview

Our analysis will be based on a somewhat novel combination of potential function and local competitiveness arguments. Let $A(t)$ and $OPT(t)$ be the unfinished jobs at time t in the algorithm's and optimal solution's schedules, respectively. In Subsection 2.3, we will define a potential function $\Phi(t)$ that satisfies the following standard properties:

- Boundary Condition: $\Phi(0) = \Phi(\infty) = 0$.
- Discontinuous Changes Condition: the potential function can only decrease when a job arrives, or is completed by our algorithm or the optimal solution.
- Continuous Changes Condition: at any time t when no job arrives or completes, $|A(t)| + \frac{d}{dt}\Phi(t) \le c|OPT(t)|$.

By integrating over time, one can see that the existence of such a potential function suffices to show that the algorithm is c-competitive for the total flow time objective. We refer the reader to [8] for details.

The novelty in our analysis lies in proving the Continuous Changes Condition. Most analyses based on potential functions rely on resource (speed) augmentation to prove this condition. We will partition time into overloaded and underloaded times. Let $\mathcal{O}$ denote the set of overloaded times t when $A(t) \ge m$, and $\mathcal{U}$ denote the set of underloaded times when $A(t) < m$. Theorem 1 then follows easily from the following three lemmas. Intuitively Lemma 1 shows that during the overloaded times, the unfinished jobs for the algorithm can be charged to the unfinished jobs for optimal at that time (a local competitiveness argument). Intuitively Lemma 2 shows that during the overloaded times, the increase in the potential function can be charged to the unfinished jobs for optimal at that time. Together Lemma 1 and Lemma 2 show that the Continuous Changes Condition holds at overloaded times. Lemma 3 then shows that the Continuous Change Condition holds at underloaded times.

LEMMA 1. *At all times $t \in \mathcal{O}$,*

$$|A(t)| \le m(3 + \log P) + 2|OPT(t)|$$

LEMMA 2. *At all times $t \in \mathcal{O}$,*

$$\frac{d}{dt}\Phi(t) \le O(1)4^{1/(1-\alpha)} \log P \ |OPT(t)|$$

LEMMA 3. *At all times $t \in \mathcal{U}$,*

$$|A(t)| + \frac{d}{dt}\Phi(t) \le O(1)2^{1/(1-\alpha)} \ |OPT(t)|$$

In Subsection 2.2, we prove Lemma 1. In Subsection 2.3, we define the potential function Φ and prove the Boundary Condition, and the Discontinuous Changes Condition. In subsection 2.4 we prove Lemma 2. In subsection 2.5 we prove Lemma 3.

2.2 Local Competitiveness During Overloaded Times

This section is devoted to proving Lemma 1 and is an adaptation of a similar result from [10]. We will need to define additional notation. At any time, we classify jobs based on remaining length. A job whose remaining length is in $[2^k, 2^{k+1})$ is in class k for integer $0 \le k \le k_{max} = \lfloor \log P \rfloor$. Note that the number of initial job classes is $\lceil \log P \rceil$. We define one special class -1 to denote jobs whose remaining length is strictly less than 1.

For scheduling algorithm S, let $\delta^S(t)$ denote the number of jobs that are alive at time t in schedule S and $V^S(t)$ denote the total volume of this schedule, where the volume is defined to be the sum of remaining lengths of jobs that are still alive. Note that $\delta^A(t) = |A(t)|$ and $\delta^{OPT}(t) = |OPT(t)|$. We define the volume difference $\Delta V(t) = V^A(t) - V^{OPT}(t)$. For function $f \in \{V, \Delta V, \delta\}$, we define $f_{\ge h, \le k}(t)$ to be the function f restricted to jobs in class at least h and at most k. We similarly define $f_{=k}(t)$. In Lemma 4 we bound the volume by which our algorithm can be behind optimal, and then use this Lemma in the proof of Lemma 5, which bounds the number of jobs by which our algorithm can be behind optimal. It is easy to see that Lemma 1 immediately follows from Lemma 5 and the observation that the number of jobs in class -1 is at most m.

LEMMA 4. *For any time $t \in \mathcal{O}$,*

$$\Delta V_{\le k}(t) \le m2^{k+1}$$

PROOF. First, for time t, we define time t' to be the earliest time such that $[t', t) \in \mathcal{O}$. Next, we define t_k to be the latest time in $[t', t)$ prior to time t in which a job of class strictly higher than k was processed by some machine. If there is no such time t_k, then we set $t_k = t'$.

We first observe that $\Delta V_{\le k}(t_k) \le m2^{k+1}$. By the definition of t_k, it follows that for any time $t_k - \epsilon$ for any $\epsilon > 0$, $\delta^A_{\le k}(t_k - \epsilon) \le m - 1$. It may be the case that some job enters class k at time t_k by the algorithm's processing, but this only means that $\delta^A_{\le k}(t_k) \le m$ when restricted to jobs that arrived strictly prior to time t_k. The volume of such jobs is restricted to at most $m2^{k+1}$ because each such job has a maximum remaining length of 2^{k+1}. Finally, jobs that arrive at time t_k do not affect $\Delta V_{\leq k}(t_k)$ since such jobs increase both $V^{OPT}_{\leq k}(t)$ and $V^{A}_{\leq k}(t)$.

We next observe that $\Delta V_{\leq k}(t) \leq \Delta V_{\leq k}(t_k)$. This follows because by the definition of $\mathcal{O}$, each machine is processing one job during $[t_k, t]$ and by the definition of t_k, each job processed cannot be in a class larger than k. Thus, our algorithm completes as much work on jobs in classes at most k during this time period as OPT and the result follows. □

Lemma 5. *For any time $t \in \mathcal{O}$,*

$$\delta^{A}_{\geq 0, \leq k_{max}}(t) \leq m(k_{max}+2) + 2\delta^{OPT}_{\leq k_{max}}(t)$$

Proof. We formulate $\delta^{A}_{\geq 0, \leq k_{max}}(t)$ as follows:

$$\begin{aligned}
\sum_{k=0}^{k_{max}} \delta^{A}_{=k}(t) &\leq \sum_{k=0}^{k_{max}} \frac{V^{A}_{k}(t)}{2^k} \\
&= \sum_{k=0}^{k_{max}} \frac{\Delta V_{=k}(t) + V^{OPT}_{=k}(t)}{2^k} \\
&= \sum_{k=0}^{k_{max}} \frac{\Delta V_{\leq k}(t) - \Delta V_{\leq k-1}(t)}{2^k} + \frac{V^{OPT}_{=k}(t)}{2^k} \\
&\leq \frac{\Delta V_{\leq k_{max}}(t)}{2^{k_{max}}} + \sum_{k=0}^{k_{max}-1} \frac{\Delta V_{\leq k}(t)}{2^{k+1}} \\
&\qquad - \frac{\Delta V_{\leq -1}(t)}{2^0} + 2\delta^{OPT}_{\geq 0, \leq k_{max}}(t) \\
&\leq 2m + \sum_{k=0}^{k_{max}-1} m + \delta^{OPT}_{\leq -1}(t) + 2\delta^{OPT}_{\geq 0, \leq k_{max}}(t) \\
&\leq m(k_{max}+2) + 2\delta^{OPT}_{\leq k_{max}}(t).
\end{aligned}$$

The first inequality follows since 2^k is the minimum remaining length of any job in class k. The fourth inequality follows by assuming the jobs in δ^{OPT}_{k} have remaining length 2^{k+1}. The fifth inequality follows from the previous lemma, observing that we can eliminate the negative term and add a positive term $\delta^{OPT}_{\leq -1}(t)$. □

2.3 Potential Function Analysis

In this section, we define the potential function Φ, and then we prove the Boundary Condition and the Discontinuous Changes Condition.

Definition of the Potential Function: Let $p^{A}_{i}(t)$ and $p^{OPT}_{i}(t)$ denote the remaining processing time of job i in the algorithm's and optimal solution's schedules at time t, respectively. Let $z_i(t) = \max\{p^{A}_{i}(t) - p^{OPT}_{i}(t), 0\}$. Recall that $A(t)$ and $OPT(t)$ denote the unfinished jobs in the algorithm's and optimal solution's schedules, respectively. Let $\texttt{rank}(i,t) = \min\{m, \sum_{j \in A(t), r_j \leq r_i} 1\}$ where without loss of generality we assume that each job arrives at a unique time. Note that $\texttt{rank}(i,t) \leq m$ for all i and t. We define the potential function as follows:

$$\Phi(t) = 16 \sum_{i \in A(t)} \frac{z_i(t)}{\Gamma_i(m/\texttt{rank}(i,t))}$$

Throughout the analysis, the following simple lemma will be useful.

Proposition 1. *For any B and C where $B \geq C$ and any job j, it is the case that $\frac{\Gamma_j(B)}{\Gamma_j(B)} \leq \frac{B}{C}$.*

Proof. The proposition follows immediately by the assumption that Γ_j is a concave function and $\Gamma_j(0) = 0$. □

Boundary Condition: It is easy to see that $\Phi(0) = \Phi(\infty) = 0$ from the definition of the potential function Φ.

Discontinuous Changes Condition: First consider when a job arrives at time t. In this case there is no change in the potential function. This is because the $\texttt{rank}$ for every job remains the same for all jobs that arrive before time t. Further, for the job i that arrives at this time, $z_i(t) = 0$. Thus, there is no change in the potential. Next observe that optimal completing a job has no effect on the potential. Now consider the case where the algorithm completes some job i at time t. In this case, the potential function can only decrease. To see that this is the case, consider any job $j \in A(t)$. If $r_j < r_i$, then there is no change in job j's term in the potential function. However, if $r_j > r_i$ then $\texttt{rank}(j,t)$ may decrease by at most one. Since Γ_j is non-decreasing, $\Gamma_j(m/\texttt{rank}(j,t))$ can only increase for a job j where $r_j \geq r_i$. Since this is in the denominator of the term in the potential function corresponding to job j and $z_j(t)$ is non-negative, the potential function can only decrease.

2.4 Potential Function Change During Overloaded Times

In this subsection we prove Lemma 2. If $|A(t)| \geq 10m \log P$, then Lemma 2 immediately follows from Lemmas 6 and 7. If $40 \cdot 4^{1/(1-\alpha)} \log P \; |OPT(t)| \geq |A(t)|$, then Lemma 2 immediately follows from Lemma 7. If $m \leq |A(t)| \leq 10m \log P$ and $40 \cdot 4^{1/(1-\alpha)} \log P \; |OPT(t)| \leq |A(t)|$ (which in turn implies that $|OPT(t)| \leq \frac{1}{4} \cdot \frac{1}{4^{1/(1-\alpha)}} m$), then Lemma 2 immediately follows from Lemmas 8 and 9.

First we consider the case where the algorithm has a large number of jobs compared to m.

Lemma 6. *If $|A(t)| \geq 10m \log P$, then $|OPT(t)| \geq |A(t)|/2 - 2m \log P \geq |A(t)|/4$.*

Proof. The lemma immediately follows from Lemma 1 by noticing that $t \in \mathcal{O}$. □

Lemma 7. *At all times t, the rate of increase in the potential due to optimal processing the jobs is at most $16(|A(t)| + |OPT(t)|)$.*

Proof. Let $q^{OPT}_{i}(t)$ be the number of machines assigned to job i by OPT at time t. The change in the potential due to optimal processing the jobs can then be bounded as follows:

$$\begin{aligned}
& 16 \sum_{i \in OPT(t)} \frac{\Gamma_i(q^{OPT}_{i}(t))}{\Gamma_i(m/\texttt{rank}(i,t))} \\
\leq\; & 16 \sum_{i \in OPT(t)} \frac{\Gamma_i(q^{OPT}_{i}(t))}{\Gamma_i(m/|A(t)|)} \quad \text{[Since } \Gamma_i \text{ is non-decreasing]} \\
\leq\; & 16|OPT(t)| + 16 \sum_{i \in OPT(t)} \frac{q^{OPT}_{i}(t)}{m/|A(t)|} \quad \text{[By Proposition 1]} \\
=\; & 16|OPT(t)| + 16|A(t)| \sum_{i \in OPT(t)} \frac{q^{OPT}_{i}(t)}{m} \\
\leq\; & 16(|A(t)| + |OPT(t)|)
\end{aligned}$$

The second inequality holds since for each job i with $\frac{q^{OPT}_{i}(t)}{m/|A(t)|} \leq 1$, it is the case that $\frac{\Gamma_i(q^{OPT}_{i}(t))}{\Gamma_i(m/|A(t)|)} \leq 1$. □

Lemma 8. *At any time t where $|OPT(t)| \leq m$, the rate of increase in the potential due to optimal processing the jobs is at most $16m^{\alpha}|OPT(t)|^{1-\alpha}$.*

Proof. As before, let $q_i^{OPT}(t)$ be the number of machines assigned to job i by OPT at time t. Let Γ be a function such that $\Gamma(x) = x$ for $0 \leq x \leq 1$ and $\Gamma(x) = x^{\alpha}$ for $x \geq 1$. Recall that $\texttt{rank}(i,t) \leq m$ for all i and t from the definition of $\texttt{rank}$. The change in the potential due to optimal processing the jobs can then be bounded as follows:

$$\begin{aligned}
& 16 \sum_{i \in OPT(t)} \frac{\Gamma_i(q_i^{OPT}(t))}{\Gamma_i(m/\texttt{rank}(i,t))} \\
\leq\ & 16 \sum_{i \in OPT(t)} \frac{\Gamma_i(q_i^{OPT}(t))}{\Gamma_i(m/m)} \quad \text{[Since } \Gamma_i \text{ is non-decreasing]} \\
=\ & 16 \sum_{i \in OPT(t)} \Gamma_i(q_i^{OPT}(t)) \quad \text{[Since } \Gamma_i(1) = 1\text{]} \\
\leq\ & 16 \sum_{i \in OPT(t)} \Gamma(q_i^{OPT}(t)) \\
\leq\ & 16|OPT(t)|(m/|OPT(t)|)^{\alpha} \quad \text{[Due to the concavity of } \Gamma\text{]} \\
=\ & 16m^{\alpha}|OPT(t)|^{1-\alpha}
\end{aligned}$$

□

Lemma 9. *At any time t where $m \leq |A(t)| \leq 10m \log P$ and $|OPT(t)| \leq \frac{1}{4} \cdot \frac{1}{4^{1/(1-\alpha)}} m$, the rate of increase in the potential due to the algorithm processing jobs is at most $-4m$.*

Proof. When $|A(t)| \geq m$ the algorithm assigns the shortest m jobs each on a unique machine. Let $A'(t)$ denote these m jobs. Notice that $z_i(t)$ decreases at a rate of one for each job in $A'(t) \setminus OPT(t)$. Thus, we have that the change in the potential due to the algorithm is at most:

$$\begin{aligned}
& -16 \sum_{i \in A'(t) \setminus OPT(t)} \frac{1}{\Gamma_i(m/\texttt{rank}(i,t))} \\
\leq\ & -16 \sum_{i \in A'(t) \setminus OPT(t)} \frac{\texttt{rank}(i,t))}{m} \\
\leq\ & -\frac{16}{m} \sum_{i=1}^{|A'(t) \setminus OPT(t)|} i \\
\leq\ & -\frac{16}{m} \frac{(3m/4)^2}{2} \\
\leq\ & -4m
\end{aligned}$$

The first inequality easily follows from Proposition 1 and by observing that $m/\texttt{rank}(i,t)) \geq 1$ since $\texttt{rank}(i,t) \leq m$. The second to last inequality holds since $|OPT(t)| \leq (1/4)m$ and $|A'(t)| = m$. □

2.5 Underloaded Times

Our goal in this subsection is to prove Lemma 3. Let t be a time such that $|A(t)| \leq m$. If $|OPT(t)| \geq \frac{1}{16}|A(t)|$, then Lemma 3 immediately follows from Lemma 7; the potential can only decrease when the algorithm processes jobs. Hence we assume that $|OPT(t)| \leq \frac{1}{16}|A(t)|$.

First we bound the increase in the potential function due to the processing of optimal. Again, let $q_i^{OPT}(t)$ be the number of processors assigned to job i at time t by OPT. The increase in the potential is at most the following.

$$\begin{aligned}
& 16 \sum_{i \in OPT(t)} \frac{\Gamma_i(q_i^{OPT}(t))}{\Gamma_i(m/\texttt{rank}(i,t))} \\
\leq\ & 16 \sum_{i \in OPT(t)} \frac{\Gamma_i(q_i^{OPT}(t))}{\Gamma_i(m/|A(t)|)} \quad \text{[Since } \Gamma_i \text{ is non-decrasing]} \\
\leq\ & 16|OPT(t)| + 16 \sum_{i \in OPT(t), q_i^{OPT}(t) \geq m/|A(t)|} \frac{(q_i^{OPT}(t))^{\alpha_i}}{(m/|A(t)|)^{\alpha_i}} \\
\leq\ & 16|OPT(t)| + 16 \sum_{i \in OPT(t)} \frac{(q_i^{OPT}(t))^{\alpha}}{(m/|A(t)|)^{\alpha}}
\end{aligned}$$

The second to last inequality holds for the following reason. Consider any job i such that $\frac{q_i^{OPT}(t)}{(m/|A(t)|)^{\alpha_i}} \leq 1$. Then we have $\frac{(q_i^{OPT}(t))^{\alpha_i}}{(m/|A(t)|)^{\alpha_i}} \leq 1$. Hence the total contribution of such jobs is at most $16|OPT(t)|$. Since our goal is to bound the total change of the potential plus $|A(t)|$ by $|OPT(t)|$, we will ignore $16|OPT(t)|$, and proceed with our string of inequalities.

$$\begin{aligned}
16 \sum_{i \in OPT(t)} \frac{(q_i^{OPT}(t))^{\alpha}}{(m/|A(t)|)^{\alpha}} &\leq 16 \sum_{i \in OPT(t)} \frac{(m/|OPT(t)|)^{\alpha}}{(m/|A(t)|)^{\alpha}} \\
&= 16|OPT(t)| \frac{(m/|OPT(t)|)^{\alpha}}{(m/|A(t)|)^{\alpha}} \\
&\leq 16|OPT(t)|^{1-\alpha}|A(t)|^{\alpha} \\
&\leq 16\Big(\frac{1}{2^{\alpha+2}}|A(t)| + 2^{\frac{\alpha+2}{1-\alpha}\alpha}|OPT(t)|\Big)
\end{aligned}$$

The first inequality is immediate from the fact that $0 \leq \alpha < 1$. The last inequality can be easily shown by considering two cases whether $|A(t)| \geq 2^{\frac{\alpha+2}{1-\alpha}}|OPT(t)|$ or not.

Now we consider the decrease in the potential function due to the algorithm processing jobs. When $|A(t)| < m$ then the algorithm gives each job equal share of every processor. Thus, for all $i \in A(t) \setminus OPT(t)$ it is the case that $z_i(t)$ decreases at a rate of $\Gamma_i(m/|A(t)|)$. Thus, we have the decrease due to the algorithm is as follows.

$$\begin{aligned}
& -16 \sum_{i \in A(t) \setminus OPT(t)} \frac{\Gamma_i(m/|A(t)|)}{\Gamma_i(m/\texttt{rank}(i,t))} \\
\leq\ & -16 \sum_{i \in A(t) \setminus OPT(t), \texttt{rank}(i,t) \geq |A(t)|/2} \frac{\Gamma_i(m/|A(t)|)}{\Gamma_i(2m/|A(t)|} \\
\leq\ & -16(|A(t) \setminus OPT(t)| - |A(t)|/2)(1/2)^{\alpha} \\
\leq\ & -16(1 - \frac{1}{2} - \frac{1}{16})|A(t)|(1/2)^{\alpha} \\
\leq\ & -6|A(t)|(1/2)^{\alpha}
\end{aligned}$$

So far we have shown that

$$\begin{aligned}
\frac{d}{dt}\Phi(t) &\leq 16\Big(\frac{1}{2^{\alpha+2}}|A(t)| + 2^{\frac{\alpha+2}{1-\alpha}\alpha}|OPT(t)|\Big) - 6|A(t)|(1/2)^{\alpha} \\
&\leq -|A(t)| + O(1)2^{1/(1-\alpha)}|OPT(t)|
\end{aligned}$$

This completes the proof.

3. LOWER BOUND FOR GREEDY ALGORITHM

In this section, we prove that the following natural greedy hybrid of Parallel-SRPT and Sequential-SRPT has a super-logarithmic lower bound on its competitive ratio.

Description of Greedy Algorithm: At all times allocate processors to jobs in such a way as to maximize the instantaneous rate at which the fractional number of unfinished jobs would be decreased, if it was the case that the original work of each job was its remaining unprocessed work. Using a simple exchange argument one can prove that if each job j has a speedup curve of the form $\Gamma_j(x) = x^\alpha$ for $\alpha \in (0, 1)$, then this policy can be implemented in the following greedy way: We arbitrarily number the processors from 1 to m. At each decision point, the machines schedule jobs in order from machine 1 to machine m. When it is machine i's turn to schedule a job, let $p(i, j)$ be the number of processors from 1 to $i - 1$ that have been assigned to job j. Processor i chooses job j that maximizes $\frac{\Gamma(p(i,j)+1)-\Gamma(p(i,j))}{p_j(t)}$.

Lemma 10. *This Greedy algorithm has a competitive ratio that is* $\Omega(\max\{P, n^{1/3}\})$.

Proof. Let $\epsilon = 1 - \alpha$. Consider an input instance where $m - m^{1-\epsilon}$ jobs of size m are released at time 0. From time 0 to time $m - \frac{1}{m^{1-\epsilon}}$, one job of size 1 is released every $\frac{1}{m^{1-\epsilon}}$ time units. Finally, at time $m + 1$, we release a job of size 1 every $\frac{1}{m^{1-\epsilon}}$ time units for $X = m^2$ time units (a total of $Xm^{1-\epsilon}$ jobs are released in this final phase).

This greedy algorithm will devote all m machines to the 1 job of size 1 and complete it just as the next size 1 job arrives. This follows by considering the last processor m. It balances the choice of $\frac{m^{1-\epsilon}-(m-1)^{1-\epsilon}}{1}$ versus $\frac{1}{m}$. Given that $\epsilon > 0$, it will always choose to assign the machine to the size 1 job.

At time m, this greedy algorithm will still have all $m - m^{1-\epsilon}$ jobs of size m remaining. In this next unit of time, it can only complete at most m units of work on these $m - m^{1-\epsilon}$ jobs; in particular, it cannot finish any of these jobs and cannot reduce the processing time to less than 1. After time $m + 1$, it will assign all m processors to the newly arrived job until the stream ends. The total flow time incurred will thus be m for the jobs of size 1 released prior to time m, X for the jobs of size 1 released after time $m + 1$, and $(m - m^{1-\epsilon} + 1)(X + m + 1)$ for the jobs of size m up to the end of the long stream. We ignore the flow time incurred to complete these long jobs after the end of the stream. The dominant term is $(m - m^{1-\epsilon})X$ for the size m jobs during the long stream.

On the other hand, an alternative algorithm (not necessarily optimal but simple to conceptualize) will assign $m - m^{1-\epsilon}$ machines to the size m jobs from time 0 to time m completing them by time m. On the remaining $m^{1-\epsilon}$ machines, it assigns one machine to each job of size 1 as that job arrives. Because it operates efficiently, each such machine will complete its assigned size 1 job exactly 1 time unit later. During this time, each job will complete just as its machine is needed to schedule the next arriving size 1 job because the number of jobs that arrive during 1 unit of time is exactly $m^{1-\epsilon}$. By time $m + 1$, this algorithm will have completed all of these jobs and will now devote all m machines to the stream of size 1 jobs that arrive completing each one just as the next arrives. Before time $m + 1$, this algorithm incurs a total time of $m^{2-\epsilon}$ for the size 1 jobs since each of these jobs is scheduled immediately on one processor. The large jobs each complete within m time units of arrival for a total flow time of $m^2 - m^{2-\epsilon}$. Finally, during the stream of length X, each job incurs a flow time of $\frac{1}{m^{1-\epsilon}}$ for a total flow time of X.

The $\Omega(P)$ bound follows from the observation that $P = m$. The $\Omega(n^{1/3})$ bound follows from the observation that $n = \Theta(m^{3-\epsilon})$. □

4. GENERAL LOWER BOUND

Our goal in this section is to prove Theorem 2, which gives a logarithmic lower bound on the competitive ratio of any algorithm. This lower bound is an adaptation of the lower bound proof from [10]. The proof is slightly more complex because online algorithms can exploit the fact that the jobs have intermediate parallelizability to catch up on jobs that they should have finished earlier.

We construct a family of input instances parameterized by α where each instance is composed of two parts. In the first part, jobs are released in phases. Each phase has long jobs and short jobs that force the online algorithm to choose between completing almost all of the short jobs before the halfway point of a phase or completing all the jobs in the phase by the end of the phase. The family of input instances is structured such that any deterministic online algorithm on at least one instance in the family must face a time T where it has at least $\Omega(m \log P)$ unfinished jobs whereas the optimal algorithm at the same moment in time will have at most $m/2$ unfinished jobs. The second part of the input instance starts at time T and presents a stream of m jobs of size 1 for P^2 consecutive starting times.

We formally define the family of input instances as follows. First, we need to define the following terms. Let $\epsilon = 1 - \alpha$. We define a length reduction factor $r = 1/2(1 - \frac{1}{2^\epsilon})$; the length of the long jobs will be multiplied by r (equivalently divided by a factor of $1/r$) in each phase of the input instance. We choose the number of machines m such that $\frac{1}{2} \frac{2^\epsilon - 1}{2^\epsilon + 1} \frac{m}{2}$ is an integer. We choose the longest job length P such that the maximum number of phases $L = 1/2 \log_{\frac{1}{r}} P$ is an integer and $\log^2_{\frac{1}{r}} P < \frac{1}{4} \frac{2^\epsilon - 1}{2^\epsilon + 1} P^{1/2}$.

The first part has at most $L = 1/2 \log_{\frac{1}{r}} P$ phases numbered from 0 to $L - 1$. Each phase $0 \le i \le L - 1$ has a phase length $p_i = Pr^i$ and a start time $s_i = \sum_{j=0}^{i-1} p_j$. During phase i, $m/2$ long jobs of length p_i are released at time s_i, and m short jobs of length 1 are released at times $s_i + j$ for $0 \le j \le p_i/2 - 1$.

The adversary begins by releasing the jobs in phase 0 starting at time $s_0 = 0$. In general, suppose the adversary has released the jobs in phase i starting at time s_i where $i \le L - 1$. The adversary decides at time $s_i + p_i/2$ whether or not to (i) begin the second part of the input instance at time $s_i + p_i/2$ or (ii) to release the next set of jobs starting at time s_{i+1} as follows. If the online algorithm has at least $m \log_{\frac{1}{r}} P$ remaining work from length 1 jobs released in phase i at time $s_i + p_i/2$, then the adversary begins the second part of the input instance at time $s_i + p_i/2$. Otherwise, if $i < L - 1$, then the adversary releases the jobs in phase $i + 1$ starting at time s_{i+1}. If $i = L - 1$, then the adversary starts the second part of the input instance at time $s_i + p_i$. This leaves us with two possible cases. In the first case, the adversary starts the second part of the input instance at some time $T = s_i + p_i/2$ where $0 \leq i \leq L-1$. In the second case, the adversary starts the second part of the input instance at time $T = s_{L-1} + p_{L-1}$.

We now argue that for both cases, the optimal flow time is bounded by $O(mP^2)$. As in [10], we define a notion of a standard schedule for phase i that has the goal of completing all jobs released in phase i by time $s_i + p_i$. Each of the $m/2$ long jobs are processed non-preemptively by one machine for the entire phase. For the m length 1 jobs released at time $s_i + k$ where $0 \leq k \leq p_i/2 - 1$, $m/2$ of them are completed by using $m/2$ machines at time $s_i + k$ and the other $m/2$ are completed using $m/2$ machines at time $s_i + k + p_i/2$. The total flow time of this standard schedule for phase i is $2mp_i + m/2(p_i/2)^2$.

We now show that the optimal flow time is $O(mP^2)$ for the first case by giving a specific schedule with flow time $O(mP^2)$. The standard schedule is used for all phases up to but not including phase i. For phase i, the $m/2$ long jobs are ignored and each length 1 job is assigned its own machine immediately upon arrival. Thus, by time $T = s_i + p_i/2$, only the $m/2$ long jobs of phase i remain. For time $T + k$ where $0 \leq k \leq P^2 - 1$, the m jobs of length 1 released at time $T + k$ are each assigned their own machine and completed by time $T + k + 1$. Finally, at time $T + P^2$, each of the $m/2$ long jobs of size p_{L-1} are assigned to 2 machines and completed by time $T + P^2 + p_i/2^\alpha$. Clearly, the overall flow time for this feasible schedule is $O(mP^2)$.

We now show that the optimal flow time is $O(mP^2)$ for the second case by giving a specific schedule with flow time $O(mP^2)$. The standard schedule is used for all phases. Thus, by time $T = s_{L-1} + p_{L-1}/2$, no jobs remain. For time $T + k$ where $0 \leq k \leq P^2 - 1$, the m jobs of length 1 released at time $T + k$ are each assigned their own machine and completed by time $T + k + 1$. Clearly, the overall flow time for this feasible schedule is $O(mP^2)$.

We now show that the online flow time for both cases is at least $\Omega(mP^2 \log_{\frac{1}{r}} P)$. By the definition of the first case, the online algorithm has at least $m \log_{\frac{1}{r}} P$ remaining work from the length 1 jobs released in phase i at time $T = s_i + p_i/2$. Thus, the online algorithm has at least $m \log_{\frac{1}{r}} P$ unfinished jobs from time T to time $T + P^2$. Using only the flow time from this time interval, we see that the online algorithm incurs a total flow time of at least $mP^2 \log_{\frac{1}{r}} P$ and the theorem follows for this case.

We now consider the second case. In our analysis, we opt for simplicity rather than proving the most accurate bound. The first key observation is that in phase i for $0 \leq i \leq L-1$, online completes at least $mp_i/2 - m \log_{\frac{1}{r}} P$ of the total available work from the length 1 jobs by time $s_i + p_i/2$, the halfway point of phase i. This means that at most $m \log_{\frac{1}{r}} P$ work can be completed on the long jobs, possibly from earlier phases, during the time interval $[s_i, s_i + p_i/2]$.

We will prove that at time T, the amount of unfinished work from the $m/2$ long jobs from phase i for $0 \leq i \leq L-1$ is at least $\frac{1}{2} \frac{2^\epsilon - 1}{2^\epsilon + 1} \frac{m}{2} p_i$. This implies that the number of long jobs with remaining length at least 1 from phase i at time T is at least $\frac{1}{2} \frac{2^\epsilon - 1}{2^\epsilon + 1} \frac{m}{2}$. Given that there are $L = 1/2 \log_{\frac{1}{r}} P$ phases, we have that the total number of jobs with remaining length at least 1 at time T is at least $L \frac{1}{2} \frac{2^\epsilon - 1}{2^\epsilon + 1} \frac{m}{2} = \frac{1}{2}(\log_{\frac{1}{r}} P) \frac{1}{2} \frac{2^\epsilon - 1}{2^\epsilon + 1} \frac{m}{2}$ which is $\Omega(m \log_{\frac{1}{r}} P)$. Thus, the total flow time incurred in interval $[T, T + P^2]$ is $\Omega(mP^2 \log_{\frac{1}{r}} P)$.

Consider the $m/2$ long jobs from phase i. From our previous observation, we can complete at most $(L-i)m \log_{\frac{1}{r}} P \leq \frac{m}{2} \log_{\frac{1}{r}} \log_{\frac{1}{r}} P \leq \frac{1}{2} \frac{2^\epsilon - 1}{2^\epsilon + 1} \frac{m}{2} P^{1/2} \leq \frac{1}{2} \frac{2^\epsilon - 1}{2^\epsilon + 1} \frac{m}{2} p^i$ work on these $m/2$ jobs during the first half of phases i to $L-1$. During the second half of phases i to $L-1$, the best we can do is devote 2 machines to each job for the entire second half of these phases; note that we ignore the processing required by any unfinished length 1 jobs from phase i in the second half of phase i. Given that the phase lengths form a geometric progression with multiplicative factor r, the total time available in these second halves of phases is strictly less than $\frac{p_i}{2} \frac{1}{1-r}$. Thus, the total amount of work that can be completed in the second half of these phases is strictly less than $\frac{m}{2} 2^\alpha \frac{p_i}{2} \frac{1}{1-r}$ which is equal to $\frac{m}{2} \frac{2}{2^\epsilon + 1} p_i$. Thus, considering only the second half of these phases, there is strictly more than $\frac{2^\epsilon - 1}{2^\epsilon + 1} \frac{m}{2} p_i$ unfinished work for these $m/2$ long jobs from phase i at time T. Taking into account how much work can be done in the first half of these phases, we see that the total unfinished work on the $m/2$ long jobs from phase i at time T is at least $\frac{1}{2} \frac{2^\epsilon - 1}{2^\epsilon + 1} \frac{m}{2} p^i$, and the theorem follows for the second case.

Acknowledgment

Sungjin Im's work was supported in part by NSF Award CCF-1008065, and was partially done while the author was at Duke. Kirk Pruhs' work was supported in part by NSF grants CCF-1115575, CNS-1253218, and an IBM Faculty Award.

5. REFERENCES

[1] `http://www.tilera.com/`.

[2] `http://projects.csail.mit.edu/angstrom/`.

[3] Ho-Leung Chan, Jeff Edmonds, and Kirk Pruhs. Speed scaling of processes with arbitrary speedup curves on a multiprocessor. *Theory Comput. Syst.*, 49(4):817–833, 2011.

[4] Jeff Edmonds. Scheduling in the dark. *Theor. Comput. Sci.*, 235(1):109–141, 2000.

[5] Jeff Edmonds, Jarek Gryz, Dongming Liang, and Renée J. Miller. Mining for empty spaces in large data sets. *Theor. Comput. Sci.*, 296(3):435–452, 2003.

[6] Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. *ACM Transactions on Algorithms*, 8(3):28, 2012.

[7] Kyle Fox, Sungjin Im, and Benjamin Moseley. Energy efficient scheduling of parallelizable jobs. In *SODA*, pages 948–957, 2013.

[8] Sungjin Im, Benjamin Moseley, and Kirk Pruhs. A tutorial on amortized local competitiveness in online scheduling. *SIGACT News*, 42(2):83–97, 2011.

[9] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

[10] Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. *Journal of Computer and Systems Sciences*, 73(6):875–891, 2007.

[11] John Markoff. Intel's big shift after hitting technical wall. *New York Times*, May 2004.

[12] Rick Merritt. CPU designers debate multi-core future. *EE Times*, February 2008.

[13] Cynthia A. Phillips, Clifford Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002.

[14] Kirk Pruhs, Julien Robert, and Nicolas Schabanel. Minimizing maximum flowtime of jobs with arbitrary parallelizability. In *WAOA*, pages 237–248, 2010.

[15] Kirk Pruhs, Jiri Sgall, and Eric Torng. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter Online Scheduling. 2004.

[16] Julien Robert and Nicolas Schabanel. Non-clairvoyant batch sets scheduling: Fairness is fair enough. In *ESA*, pages 741–753, 2007.

[17] Julien Robert and Nicolas Schabanel. Non-clairvoyant scheduling with precedence constraints. In *SODA*, pages 491–500, 2008.