Speed Scaling of Processes with Arbitrary Speedup Curves on a Multiprocessor

Nikhil Bansal[*] Ho Leung Chan [†] Jeff Edmonds[‡] Kirk Pruhs[§]

"With multi-core it's like we are throwing this Hail Mary pass down the field and now we have to run down there as fast as we can to see if we can catch it."
— David Patterson, UC Berkeley computer science professor

Abstract

We consider the setting of a multiprocessor where the speeds of the m processors can be individually scaled. Jobs arrive over time and have varying degrees of parallelizability. A nonclairvoyant scheduler must assign the jobs to processors, and scale the speeds of the processors. We consider the objective of energy plus flow time. For jobs that my have side effects or that are not checkpointable, we show an $\Omega(m^{(\alpha-1)/\alpha^2})$ bound on the competitive ratio of any deterministic algorithm. For jobs without side effects that may be efficiently check pointed, we give an $O(\log m)$-competitive algorithm. Thus for jobs that may have side effects or that are not checkpointable, the achievable competitive ratio grows quickly with the number of processors, but for check-pointable jobs without side effects, the achievable competitive ratio grow slowly with the number of processors. We then lower bound of $\Omega(\log^{1/\alpha} m)$ on the competitive ratio of any algorithm for checkpointable processes without side effects. Finally we slightly improve the upper bound on the competitive ratio for the single processor case, which is equivalent to the case that all jobs are fully parallelizable, by giving an improved analysis of a previously proposed algorithm.

This is a regular (not short) submission.

[*]IBM. nikhil@us.ibm.com.

[†]MPI. hlchan@cs.hku.hk

[‡]York University, Canada. jeff@cs.yorku.ca. Supported in part by NSERC Canada.

[§]Computer Science Department. University of Pittsburgh. kirk@cs.pitt.edu. Supported in part by an IBM faculty award, and by NSF grants CNS-0325353, CCF-0514058, IIS-0534531, and CCF-0830558.

1 Introduction

Due to the power related issues of energy and temperature, major chip manufacturers, such as Intel, AMD and IBM, now produce chips with multiple cores/processors, and with dynamically scalable speeds, and produce associated software, such as Intel's SpeedStep and AMD's PowerNow, that enables an operating system to manage power by scaling processor speed. Currently most multiprocessor chips have only a handful of processors, but chip designers are agreed upon the fact that chips with hundreds to thousands of processors chips will dominate the market in the next decade. The founder of chip maker Tilera asserts that a corollary to Moore's law will be that the number of cores/processors will double every 18 months [18].

According to the well known cube-root rule, a CMOS-based processor running at speed s will have a dynamic power P of approximately s^3. In the algorithmic literature, this is usually generalized to $P = s^\alpha$. Thus in principle, p processors running at speed s/p could do the work of one processor running at speed s but at $1/p^{\alpha-1}$ of the power. But in spite of this, chip makers waited until the power costs became absolutely prohibitive before switching to multiprocessor chips because of the technical difficulties in getting p speed s/p processors to come close to doing the work of one speed s processor. This is particularly true when you have many processors, and few processes, where these processes have widely varying degrees of parallelizability. That is, some processes may be be considerably sped up when simultaneously run on to multiple processors, while some processes may not be sped up at all (this could be because the underlying algorithm is inherently sequential in nature, or because the process was not coded in a way to make it easily parallelizable). To investigate this issue, we adopt the following general model of parallelizability used in [12, 14, 23, 24]. Each process consists of a sequence of phases. Each phase consists of a positive real number that denotes the amount of work in that phase, and a speedup function that specifies the rate at which work is processed in this phase as a function of the number of processors executing the process. The speedup functions may be arbitrary, other than we assume that they are nondecreasing (a process doesn't run slower if it is given more processors), and sublinear (a process satisfies Brent's theorem, that is increasing the number of processors doesn't increase the efficiency of computation).

The operating system needs a *process assignment* policy for determining at each time, processors (if any) a particular process is assigned to. We assume that a process may be assigned to multiple processors. In tandem with this, the operating system will also need a *speed scaling* policy for setting the speed of each processor. In order to be implementable in a real system, the speed scaling and process assignment policies must be online since the system will not in general know about processes arriving in the future. Further, to be implementable in a generic operating system, these policies must be nonclairvoyant, since in general the operating system does not know the size/work of each process when the process is released to the operating system, nor the degree to which that process is parallelizable. So a nonclairvoyant algorithm only knows when processes have been released and finished in the past, and which processes have been run on each machine at each time in the past.

The operating system has competing dual objectives, as it both wants to optimize some schedule quality of service objective, as well as some power related objective. In this paper, we will consider the formal objective of minimizing a linear combination of total response/flow time (the schedule objective) and total energy used (the power objective). (In the conclusion, we we will discuss the relationship between this energy objective and a temperature objective). This objective of flow plus energy has a natural interpretation. Suppose that the user specifies how much improvement in flow, call this amount ρ, is necessary to justify spending one unit of energy. For example, the user might specify that he is willing to spend 1 erg of energy from the battery for a decrease of 6 micro-seconds in flow. Then the optimal schedule, from this user's perspective, is the schedule that optimizes $\rho = 6$ times the energy used plus the total flow. By changing the units of either energy or time, one may assume without loss of generality that $\rho = 1$.

So the problem we want to address here is how to design a nonclairvoyant process assignment policy and a speed scaling policy that will be competitive for the objective of flow plus energy.

The case of a single processor was considered in [10]. In the single processor case, the parallelizability of the processes is not an issue. If all the processes arrive at time 0, then the in the optimal schedule, the power at time t is $\Theta(n_t)$, where n_t is the number of unfinished processes at time t. The algorithm considered in [10] runs at a constant factor faster speed than this, say $(1+\delta)n_t^{1/\alpha}$. The process assignment algorithm considered in [10] is Latest Arrival Processor Sharing (LAPS). LAPS was proposed in [14] in the context of running processes with arbitrary speed-up curves on fixed speed processors, and it was shown to be scalable, $(1+\epsilon)$-speed $O(1)$-competitive, for the objective of total flow time in this setting. LAPS is parameterized by a constant $\beta \in (0, 1]$, and shares the processing power evenly among the βn_t most recently arriving processes. Note that is speed scaling policy and LAPS are both nonclairvoyant. [10] showed that, by picking δ and β appropriately, that the resulting algorithm is $4\alpha^3(1 + (1 + \frac{3}{\alpha})^\alpha)$ competitive for the objective of flow plus energy on a single speed scalable processor. In particular, note that this algorithm is $O(1)$-competitive in the case that α is constant, such as when the cube-root rule holds.

1.1 Our Results

Here we consider extending the results in [10] to a multiprocessor setting. It is straight-forward to note that if all of the work is parallelizable, then the multiprocessor setting is essentially equivalent to the uniprocessor setting. To gain some intuition of the difficulty that varying speed-up curves pose, let us first consider an instance of one processes that may either be sequential or parallelizable. If an algorithm runs this process on few of processors, then the algorithm's competitive ratio will be bad if the process is parallelizable, and the optimal schedule runs the process on all of the processors. If the algorithm wanted to be competitive on flow time, it would have to run too fast to be competitive on energy. If an algorithm runs this process on many of processors, then the algorithm's competitive ratio will be bad if the process is sequential, and the optimal schedule runs the process on few processors. If the algorithm wanted to be competitive on energy, it would have to run too slow to be competitive on flow time. Formalizing this argument, we show in section 2 a lower bound on the competitive ratio of any deterministic algorithm of $\Omega(m^{(\alpha-1)/\alpha^2})$ (with an additional assumption that we will now discuss).

At first glance, such a strong lower bound for such an easy instance leads one to conclude that there is really no way that the scheduler can be expected to produce reasonable schedules. But on further reflection, one realizes that an underlying assumption in this lower bound is that only one copy of a process can be run. If a process does not have side effects, that is if the process doesn't change/effect anything external to itself, then this assumption is not generally valid. One could run multiple copies of a process simultaneously, with each copy being run on a different number of processors, and halt computation when the first copy finishes. Unfortunately, we also show in section 2 an $m^{\Omega(1/\alpha)}$ lower bound on the competitive ratio of any deterministic algorithm of this type, again using a one process instance.

Contemplating this second lower bound, one gains the intuition that what the algorithm must be able to accomplish is that at all times, all copies process work at the speed of the fastest copy at that time. If the process had small state, so that the overhead of checkpointing isn't prohibitive, one might reasonably approximate this by checkpointing each copy periodically,and then restarting each copy from the point of execution of the copy that made the most progress. In section 3, we formalize this intuition. We give a process assignment algorithm MultiLAPS, which is a modification of LAPS. We show that, by combining MultiLAPS with the natural speed scaling algorithm from say [10], one obtains an $O(\log m)$ competitive algorithm if all copies process work at the rate of the fastest copy. There are two steps in the analysis of MultiLAPS. The first step is to show that there is a worst-case instance where every speed-up curve is parallelizable up to some number of processors, and then is constant. This shows that the worst-case speed-up curves for speed scalable processors are more varied than for fixed speed processors, where it is sufficient to restrict attention to only parallel and sequential speed-up curves [12, 14, 23, 24]. The second step in the analysis of MultiLAPS is a reduction to the analysis of LAPS in a uniprocessor setting.

In section 4 we then lower bound of $\Omega(\log^{1/\alpha} m)$ on the competitive ratio of any nonclairvoyant algorithm for checkpointable processes without side effects. In fact, this lower holds for randomized algorithms against an oblivious adversary, and even if the rate that a process is processes is the sum (not the maximum) rate of the various copies.

Thus in summary, for processes that may have side effects, or that are not checkpointable, the achievable competitive ratio grows quickly with the number of processors. But for checkpointable processes without side effects, the achievable competitive ratio grows slowly with the number of processors. This shows the importance of being able to efficiently checkpoint multiple copies of a process, in a setting of processes with varying degrees of parallelizability and individually speed scalable multiprocessors.

Finally, in section 5 show how improve the competitive analysis for the LAPS algorithm in the single processor setting. The key to this improved analysis is the use of a convexity technique, introduced in [4], that replaces the use of Young's inequality in [10]. We obtain a competitive ratio of $32(\alpha-1)^2/\ln^2\alpha$. For large α, this improves the bound from $O(\alpha^3)$ to $O(\alpha^2/\log^2\alpha)$. And when the cube-root rule holds, this reduces the competitive ratio from 1032 to a mere 213.

1.2 Related results

We start with some results in the literature about scheduling with the objective of total flow time on a single fixed speed processor. It is well known that the online clairvoyant algorithm Shortest Remaining Processing Time (SRPT) is optimal. The competitive ratio of deterministic nonclairvoyant algorithm is $\Omega(n^{1/3})$, and the competitive ratio of every randomized algorithm against an oblivious adversary is $\Omega(\log n)$ [19]. A randomized version of the Multi-Level Feedback Queue algorithm is $O(\log n)$-competitive [8, 16]. The non-clairvoyant algorithm Shortest Elapsed Time First (SETF) is scalable, that is, $(1+\epsilon)$-speed $O(1)$-competitive [15]. SETF shares the processor equally among all processes that have been run the least.

We now consider scheduling processes with arbitrary speed-up curves on fixed speed processors for the objective of total flow time. The algorithm Round Robin RR (also called Equipartition and Processor Sharing) that shares the processor equally among all processes is $(2+\epsilon)$-speed $O(1)$-competitive [12]. As mentioned before, LAPS is scalable [14].

We now consider speed scaling algorithms on a single processor for the objective flow plus energy. [1, 22] give efficient offline algorithms. We give a brief overview of the clairvoyant algorithms are given in [1, 2, 5, 6, 17]. The scheduling algorithm, that uses Shortest Remaining Processing Time (SRPT) for process assignment and power equal to one more than the number of unfinished processes for speed scaling, is $(3+\epsilon)$-competitive for the objective of total flow plus energy on arbitrary-work unit-weight processes, even if the power function is arbitrary. So clairvoyant algorithms can be $O(1)$-competitive independent of the power function. [10] showed that nonclairvoyant algorithms can not be $O(1)$-competitive if the power function is growing too quickly. The scheduling algorithm, that uses Highest Density First (HDF) for process assignment and power equal to the fractional weight of the unfinished processes for speed scaling, is $(2+\epsilon)$-competitive for the objective of fractional weighted flow plus energy on arbitrary-work arbitrary-weight processes. An $O(1)$-competitive algorithm for weighted flow plus energy can then be obtained using the known resource augmentation analysis of HDF [9]. [2, 17] extend some of the results for the unbounded speed model to an upper bound on the speed of a processor.

There are many related scheduling problems with other objectives, and/or other assumptions about the machine and instance. Surveys can be found in [20, 21].

1.3 Formal Problem Definition and Notation

In this section, we review the formal definitions. An instance consists of a collection $J = \{J_1, \ldots, J_n\}$ where *job* J_i has a *release/arrival time* r_i and a sequence of phases $\left\langle J_i^1, J_i^2, \ldots, J_i^{q_i} \right\rangle$. Each phase is an ordered pair $\langle w_i^q, \Gamma_i^q \rangle$, where w_i^q is a positive real number that denotes the amount of *work* in the phase and Γ_i^q is a function, called the *speedup function*, that maps a nonnegative real number to a nonnegative real number. $\Gamma_i^q(\rho)$ represents the rate at which work is executed for phase q of job i when given ρ processors running at speed 1. If the these processors are run at speed s, then work is processed at a rate of $s\Gamma_i^q(\rho)$.

A schedule specifies for each time and each processor, which job is run and a nonnegative real speed. We assume that if several processor on working on the same instance/copy of a job, that they must run at the same speed. But different copies can run at different speeds. Preemption is allowed and has no cost; a preempted job can resume at the point of preemption. When running at speed s, a processor consumes $P(s) = s^\alpha$ units of energy per unit time, where $\alpha > 1$ is some fixed constant. We call $P(s)$ the *power function.* The *completion* time of a job J_i, denoted C_i, is the completion time of the last phase of the job. A job is said to be *alive* at time t, if it has been released, but has not completed, i.e., $r_i \leq t \leq c_i$. The *response/flow time* of job J_i, $C_i - r_i$, is the length of the time interval during which the job is *alive.* Let n_t be the number of jobs alive at time t. Then another formulation of total response time is $\int_0^\infty n_t dt$.

A phase of a job is *parallelizable* if its speedup function is $\Gamma(\rho) = \rho$. Increasing the number of processors allocated to a parallelizable job by a factor of s increases the rate of processing by a factor of s. A phase of a job is *parallel up to p processors* $\Gamma(\rho) = \rho$ for $\rho \leq p$, and $\Gamma(\rho) = p$ for $\rho > p$. A phase is *sequential* if it is parallel up to one processor. Formally, a speedup function Γ is *nondecreasing* if and only if $\Gamma(\rho_1) \leq \Gamma(\rho_2)$ whenever $\rho_1 \leq \rho_2$. Formally, a speedup function Γ is *sublinear* if and only if $\Gamma(\rho_1)/\rho_1 \geq \Gamma(\rho_2)/\rho_2$ whenever $\rho_1 \leq \rho_2$.

2 Lower Bound for Single Copy and Non-Checkpointing Algorithms

In this section we show that the competitive ratio must grow quickly with the number of processors if only one copy of each job can be running, or if multiple copies are allowed, but no checkpointing is allowed.

Theorem 1. *Any algorithm that only runs one copy of each job must be $\Omega(m^{(\alpha-1)/\alpha^2})$ competitive.*

Proof. We consider an instance of one job with unit work. This work will either be sequential or parallelizable. Assume that the algorithm runs this on p processors at speed s.

Consider the case that the work is parallelizable. The flow time for the algorithm is $\frac{1}{sp}$, and the energy cost is $\frac{1}{sp} \cdot ps^\alpha = s^{\alpha-1}$. Thus the cost is this case is $\Omega(\max(s^{\alpha-1}, \frac{1}{sp}))$. These terms are equal if $s = (1/p)^{1/\alpha}$, and hence the cost is $\Omega(\frac{1}{p^{1-1/\alpha}})$. The optimum can run the job on all m

processors, and (by plugging m instead of p in the expression above) incurs a cost of $O(\frac{1}{m^{1-1/\alpha}})$. Thus the competitive ratio is this case is $\Omega((\frac{m}{p})^{1-1/\alpha})$.

Now consider the case that the work is sequential. For the algorithm, the flow time is $1/s$ and the energy cost for the algorithm is then $ps^{\alpha-1}$. These terms are equal if $s = (1/p)^{1/\alpha}$ and hence the cost is $\Omega(p^{1/\alpha})$. The optimum can run the job on one processor and incur a cost of $O(1)$. Thus the competitive ratio is this case is $\Omega(p^{1/\alpha})$.

These two lower bounds on the competitive ratio are equal if $p = m^{1-1/\alpha}$ and hence the competitive ratio is at least $p^{1/\alpha} = m^{(\alpha-1)/\alpha^2}$. □

Theorem 2. *Consider the instance of a single job. Consider algorithms that run multiple copies of that job, where each copy is run on a fixed number of processors at a fixed speed, and the copies do not communicate/interact with each other. Then the competitive ratio for all such algorithms is* $m^{\Omega(1/\alpha)}$.

Proof. Let us consider a job with phase $i \in [\log_2(m)]$ having work $w_i = 2^{(1-1/\alpha)i}$ and speedup function $\Gamma_{2^i}(\rho)$. Opt clearly allocates $\rho_{\langle o,i\rangle} = 2^i$ processors of speed $s_{\langle o,i\rangle} = \frac{1}{[\rho_{\langle o,i\rangle}]^{1/\alpha}} = 2^{-(1/\alpha)i}$ to the i^{th} phase for a cost of

$$\begin{aligned}
\sum_i \left[F_{\langle o,i\rangle} + E_{\langle o,i\rangle}\right] &= \sum_i \left[\frac{w_i}{\Gamma_{2^i}(\rho_{\langle o,i\rangle})s_{\langle o,i\rangle}} + \rho_{\langle o,i\rangle} s^{\alpha}_{\langle o,i\rangle} \frac{w_i}{\Gamma_{2^i}(\rho_{\langle o,i\rangle})s_{\langle o,i\rangle}}\right] \\
&= \sum_i \left[\frac{[2^{(1-1/\alpha)i}]}{[2^i][2^{-(1/\alpha)i}]} + [2^i][2^{-(1/\alpha)i}]^{\alpha} \frac{[2^{(1-1/\alpha)i}]}{[2^i][2^{-(1/\alpha)i}]}\right] \\
&= \sum_i 2 = 2\log_2 m
\end{aligned}$$

The winning group in the online algorithm allocates ρ speed s processors for the entire job for a cost of

$$\begin{aligned}
\sum_i \left[F_{\langle a,i\rangle} + E_{\langle a,i\rangle}\right] &= \sum_i \left[\frac{w_i}{\Gamma_{2^i}(\rho)s} + \rho s^{\alpha} \frac{w_i}{\Gamma_{2^i}(\rho)s}\right] \\
&= \sum_{i \le log_2\rho} \left[\frac{[2^{(1-1/\alpha)i}]}{[2^i]s} + \rho s^{\alpha}\frac{[2^{(1-1/\alpha)i}]}{[2^i]s}\right] + \sum_{log_2\rho < i} \left[\frac{[2^{(1-1/\alpha)i}]}{\rho s} + \rho s^{\alpha}\frac{[2^{(1-1/\alpha)i}]}{\rho s}\right] \\
&= \left[\frac{1}{s} + \rho s^{\alpha}\frac{1}{s}\right]\left[\sum_{i \le log_2\rho} \frac{1}{[2^{(1/\alpha)i}]}\right] + \left[\frac{1}{\rho s} + \rho s^{\alpha}\frac{1}{\rho s}\right]\left[\sum_{log_2\rho < i} [2^{(1-1/\alpha)i}]\right] \\
&= \Omega\left(\left[\frac{1}{s} + \rho s^{\alpha-1}\right][1] + \left[\frac{1}{\rho s} + s^{\alpha-1}\right]\left[m^{(1-1/\alpha)}\right]\right) \\
&= \Omega\left(\left[\rho^{1/\alpha}\right][1] + \left[\frac{1}{\rho^{1-1/\alpha}}\right]\left[m^{(1-1/\alpha)}\right]\right) \\
&= \Omega\left(m^{1/\alpha - 1/\alpha^2}\right)
\end{aligned}$$

The second to last equality comes from noting that the optimal setting for s is $s = \frac{1}{\rho^{1/\alpha}}$. The last equality comes from noting that the optimal setting for ρ is $\rho = m^{1-1/\alpha}$. □

3 Analysis of MultiLAPS

This section considers jobs without side effects, so that multiple copies of a job can be run simultaneously. We show that MultiLAPS, defined below, is $O(\frac{\alpha^2}{\log^2 \alpha}(3^\alpha + 2^\alpha \log m))$-competitive.

> **MultiLAPS(β).** Let $0 < \beta < 1$ be any given real number. Consider any time t. Let n_a be the number of active jobs at t, $s_a = n_a^{1/\alpha}$, and $\mu = 1/3$. For each of the $\lceil \beta n_a \rceil$ active jobs with the latest release times, MultiLAPS allocates a group of $p = \mu \frac{m}{\lceil \beta n_a \rceil}$ processors with speed $s = \frac{1}{\mu}(\frac{1}{m})^{1/\alpha} s_a$. Furthermore, for each of these $\lceil \beta n_a \rceil$ jobs, MultiLAPS allocates an additional $\log p$ groups of processors, where for $i = 1, \ldots, \log p$, the i-th group consists of 2^i processors with speed $2(\frac{1}{2^i})^{1/\alpha}$. We can check that the total number of processors allocated is $\lceil \beta n_a \rceil p + \lceil \beta n_a \rceil \sum_{i=1}^{\log p} 2^i \leq m$.

To analyze MultiLAPS, we first show that there is a worst case instance containing only jobs with phases that are parallel up to p_o processors for some $p_o > 0$, which may be different for each phases. It is similar to [12, 14, 23, 24], in which they show that the worst case instance for their problems contains only parallelizable or sequential jobs. For any algorithm A and instance J, we denote $F_A(J)$ and $E_A(J)$ as the total flow time and energy incurred, respectively, when J is scheduled by A. Denote $cost_A(J) = F_A(J) + E_A(J)$. We will use M as a short-hand for MultiLAPS. Let Opt be the optimal algorithm that always minimizes total flow time plus energy.

Lemma 3. *Let J be any input instance. There is an instance J' in which all phases are parallel up to p_o processor, for their respective $p_o > 0$, such that $F_M(J') = F_M(J)$, $E_M(J') = E_M(J)$, $cost_{\text{Opt}}(J') \leq cost_{\text{Opt}}(J)$.*

Proof. We construct J' by modifying each job in J as follows. Consider an infinitesimally small phase of a job in J with size w and speedup function Γ. Let p_o be the number of processors that Opt allocates to this phase when scheduling J. We modify this phase so that the new speedup function is $\Gamma'(\rho) = \frac{\rho}{p_o}\Gamma(p_o)$ for $\rho \leq p_o$ and $\Gamma'(\rho) = \Gamma(p_o)$ for $\rho \geq p_o$. Note that MultiLAPS may process this phase by a number of groups, where the i-th group has ρ_i processors of speed s_i. Due to the modification of speedup function, the rate of processing for the i-th group changes from $\Gamma(\rho_i)s_i$ to $\Gamma'(\rho_i)s_i$. Since Γ is sublinear, $\Gamma(\rho_i)s_i \geq \Gamma'(\rho_i)s_i$. Hence we can decreases the size of this phase so that the time when this phase is first completed remains the same in MultiLAPS. Note that the schedule of MultiLAPS remains the same for J and J', while Opt may get better performance due to the reduction of job size. Finally, we can normalize the size and the speedup function by the same factor of $\frac{p_o}{\Gamma'(p_o)}$ so that the schedules of both MultiLAPS and Opt remain the same while $\Gamma'(p_o)$ becomes p_o and the phase is parallel up to ρ_o processors. □

Theorem 4. *When allowing multiple copies of a job,* MultiLAPS *is* $O(\frac{\alpha^2}{\log^2 \alpha}(3^\alpha + 2^\alpha \log m))$-*competitive for total flow time plus energy.*

Proof. To analyze the performance of MultiLAPS, by Lemma 3, we only need to consider instance with phases that are parallel up to p_o processors for some $p_o > 0$. Consider any such instance J. Our aim is to transform it into another instance J' for the following setting.

> **Variable Processor Setting.** Jobs are coming online and all phases of the jobs are either parallelizable or sequential. There is a supply of unit-speed processors. At any time, the online algorithm decides the set of jobs to be run and the number of processors allocated to each job. When totally s_a processors are being allocated at some time, the energy usage per unit time is s_a^α. The objective is to minimize the total flow time plus energy usage.

Note that we can define LAPS in the Variable Processor Setting such that at any time with n_a active jobs, it allocates totally $n_a^{1/\alpha}$ processors evenly to the $\lceil \beta n_a \rceil$ jobs with the latest release time. Our transformation guarantees that the total cost of MultiLAPS with input J is at most $O(\log m)$ times the total cost of LAPS with input J', where the cost of LAPS refers to its cost in the Variable Processor Setting. Furthermore, the optimal cost for J is close to the optimal cost for J' in Hence, by observing performance of LAPS in the Variable Processor Setting, it implies the performance of MultiLAPS in the original setting. Details are as follows.

We first describe our transformation. We transform each job in J into a job in J' by modifying each phase of the original job. At each point in time, consider a phase and the group in MultiLAPS with the highest processing rate. Let Γ_{p_o} be the speedup function of the phase, which is parallel up to p_o processors. We say the phase is currently "saturated" if $\mu \frac{m}{\lceil \beta n_a \rceil} \leq p_o$, and "unsaturated" otherwise. Note that $\mu \frac{m}{\lceil \beta n_a \rceil}$ is the number of processors in the group with the most processors in MultiLAPS. Thus, a phase is saturated if all groups in MultiLAPS are processing in the parallel range or Γ_{p_o}, and unsaturated otherwise.

Transformation and effect on MultiLAPS **and** LAPS**.** Our transformation will ensure that the set of active jobs at any time and the running time of each job are the same in the two algorithms, as follows. If the phase is saturated, then the group with the highest processing rate in MultiLAPS is the one with $\mu \frac{m}{\lceil \beta n_a \rceil}$ processors of speed $\frac{1}{\mu}(\frac{1}{m})^{1/\alpha} s_a$, giving a rate of $\frac{m}{\lceil \beta n_a \rceil}(\frac{1}{m})^{1/\alpha} s_a = m^{1-1/\alpha} \cdot \frac{s_a}{\lceil \beta n_a \rceil}$. We modify this phase into fully parallelizable and scaling down the size by a factor of $m^{1-1/\alpha}$. Note that the processing rate of LAPS is $\frac{s_a}{\lceil \beta n_a \rceil}$, so it will complete the phase using the same time. If a phase is unsaturated, consider the biggest group in MultiLAPS that is processing in the parallel range, i.e., the group with 2^i processors such that 2^i is maximized and at most p_o, where p_o is the value the speedup function Γ_{p_o}. We notice that $2^i \leq p_o < 2 \cdot 2^i$. The processing rate of this group is is $2^i \cdot 2(\frac{1}{2^i})^{1/\alpha} = 2 \cdot 2^{1-1/\alpha} \geq p_o^{1-1/\alpha}$. Notice that the processing rate of MultiLAPS for this phase is the maximum of all groups, so the rate is at least $r \geq p_o^{1-1/\alpha}$. We modify this phase into sequential and scale down the size by a factor of r. Note that the processing rate of LAPS is 1, so it will complete the phase using the same time as MultiLAPS.

Consider at any time with n_a active jobs in both MultiLAPS and LAPS. The rate of energy usage in MultiLAPS is the sum of energy usage in the m processors. Note that each of the $\lceil \beta n_a \rceil$ latest released jobs, MultiLAPS allocates p processors of speed s, where $p = \mu \frac{m}{\lceil \beta n_a \rceil}$ and $s = \frac{1}{\mu}(\frac{1}{m})^{1/\alpha} s_a$, and another $\log p$ groups where the i-th group has 2^i processors of speed $2(\frac{1}{2^i})^{1/\alpha}$. Hence, the total rate of energy usage is $\lceil \beta n_a \rceil \left(\mu \frac{m}{\lceil \beta n_a \rceil} \cdot (\frac{1}{\mu}(\frac{1}{m})^{1/\alpha} s_a)^\alpha + \sum_{i=1}^{\log p} 2^i (2(\frac{1}{2^i})^{1/\alpha}))^\alpha \right) \leq (\frac{1}{\mu})^{\alpha-1} n_a + 2^\alpha \beta \log p \;\, n_a$. For LAPS, it uses $n_a^{1/\alpha}$ processors and incurs a rate of energy usage of n_a. Since $p \leq m$ and $\log p \leq \log m$, we conclude that $E_M(J) \leq ((\frac{1}{\mu})^{\alpha-1} + 2^\alpha \beta \log m) E_{\text{LAPS}}(J')$. As shown previously, $F_M(J) = F_{\text{LAPS}}(J')$, so so we have $cost_M(J) \leq ((\frac{1}{\mu})^{\alpha-1} + 2^\alpha \beta \log m) cost_{\text{LAPS}}(J')$.

Effect on Opt **and** Opt$'$**.** Intuitively, we want lower bound $cost_{\text{Opt}}(J)$ by the optimal cost of scheduling J' in the variable processor setting. Then the competitiveness of LAPS would imply the competitiveness of MultiLAPS. But the exactly lower bound for $cost_{\text{Opt}}(J)$ is a bit technical, as follows. Recall that Opt is the optimal schedule for J, and depending on the operation of MultiLAPS, a phase is defined as either saturated or unsaturated. Define J_{sat} to be the instance obtained from J by removing all unsaturated phases in each job and directly concatenating the saturated phases. Define J_{uns} conversely, i.e., the instance obtained by removing all saturated phases. Define J'_{par} to be the instance consisting of the jobs corresponding to the phases in J_{sat}. Note that each phase in J_{sat} is transformed into a parallel phase, so J'_{par} consists of only parallel phases. Define J'_{seq} conversely, i.e., the instance corresponding to J_{uns} which consists of sequential phases only. Then, the lower bound we want to show is $cost_{\text{Opt}}(J) \geq cost_{\text{Opt}'}(J'_{seq}) + cost_{\text{Opt}'}(J'_{par})$, where the $cost$ function is calculated in their respective settings. We can show easily that LAPS is $O(\frac{\alpha^2}{\log^\alpha})$-competitive, and furthermore, $cost_{\text{LAPS}}(J') \leq O(\frac{\alpha^2}{\log^\alpha})(cost_{\text{Opt}'}(J'_{seq}) + cost_{\text{Opt}'}(J'_{par}))$. Then together with $cost_M(J) \leq ((\frac{1}{\mu})^{\alpha-1} + 2^\alpha \beta \log m) cost_{\text{LAPS}}(J')$, the theorem follows by setting $\mu = \frac{1}{3}$ and β to $\ln \alpha/(4(\alpha-1))$. Note that the effect of β is on determining the competitive ratio of LAPS only.

So it remains to show the above mentioned lower bound. Obviously, Opt can only gain by scheduling J_{sat} and J_{uns} separately, i.e., $cost_{\text{Opt}}(J) \geq cost_{\text{Opt}}(J_{sat}) + cost_{\text{Opt}}(J_{uns})$. We show that Opt$'$ can also schedule J'_{par} and J'_{seq} with costs at most that for J_{sat} and J_{uns} in Opt, respectively, as follows. Consider a parallel phase in J_{sat}. Let ρ_o and s_o be the number of processors and speed allocated by Opt to this phase. Then Opt is processing at a rate of $\rho_o s_o$. Define Opt$'$ so that it allocates $\rho'_o = (\frac{1}{m})^{1-1/\alpha} \rho_o s_o$ unit speed processors to this phase. Note that the phase after transformation is parallelizable, so Opt$'$ is processing at a rate of $(\frac{1}{m})^{1-1/\alpha} \rho_o s_o$. Since the transformation scale down the size by a factor of $(\frac{1}{m})^{1-1/\alpha}$, Opt$'$ will complete the phase in the same time. The rate of energy usage in Opt is $E = \sum_j \rho_{o,j} (s_{o,j})^\alpha$, where the sum is over all jobs j it is processing and $\rho_{o,j}$ and $s_{o,j}$ are the number and speed of the processors allocated to j. Keeping $R = \sum_j \rho_{o,j} \;\; s_{o,j}$ fixed, E is minimized by having all the $s_{o,j}$ to be the same fixed value s_o. This gives $R = \sum_j \rho_{o,j} s_o = s_o m$ and $E \geq \sum_j \rho_{o,j} \;\; {s_o}^\alpha = {s_o}^\alpha m = (\frac{R}{m})^\alpha m = (\frac{1}{m})^{\alpha-1} R^\alpha$. In contrast, the rate of energy usage in Opt$'$ is $E' = (\rho'_a)^\alpha$, where ρ'_a is the total number of processors allocated. By the construct of Opt$'$, $\rho'_a = \sum_j (\frac{1}{m})^{1-1/\alpha} \cdot \rho_{o,j} \;\; s_{o,j} = (\frac{1}{m})^{1-1/\alpha} R$. This give $E = ((\frac{1}{m})^{1-1/\alpha} R)^\alpha = (\frac{1}{m})^{\alpha-1} R^\alpha$ which is no more than that of Opt. It means that $cost_{\text{Opt}}(J_{sat}) \geq cost_{\text{Opt}'}(J'_{par})$.

Finally, consider a unsaturated phase in J_{uns}. We allow Opt to schedule each job in J_{uns} independently and it only improves Opt. Assume the speedup function of the phase parallel up to p_o. Let $\rho_{o,j} \leq p_o$ be the number of processors allocated to this phase by Opt. We can show that in a single job case, the total flow time plus energy incurred for Opt is at least $\frac{w}{(\rho_{o,j})^{1-1/\alpha}}$ Opt$'$ will allocate zero processor to the phase and the rate of processing is 1. Note that size of the phase is smaller in J' by a factor of at least $p_o^{1-1/\alpha} \geq \rho_{o,j}^{1-1/\alpha}$. Hence, $cost_{\text{Opt}}(J_{uns}) \geq cost_{\text{Opt}'}(J'_{seq})$, and it completes the proof. ☐

The multiplicative constant in our upper bound on the competitive ratio for MultiLAPS can be improved from exponential in α to something more like $O(\alpha^2/\log^2 \alpha)$ using techniques from [11, 13, 14] at the cost of complicating the proof somewhat.

4 Lower Bound for Checkpointable Multiple Copies

We show here that even if the rate that the work is processed is the sum of the rate of the copies, that every algorithm is poly-log competitive.

Theorem 5. *The competitive ratio for every nonclairvoyant algorithm, is $\Omega\left(\log^{1/\alpha} m\right)$, even if the rate that a process is processed is the sum rate of each of the copies. This lower bound also holds for randomized algorithms against an oblivious adversary.*

Proof. Let us first consider what the optimal allocation is for a job with w work and speed up function $\Gamma_{\rho_o}(\rho)$, i.e. is fully parallelizable up to some parameter ρ_o and then is sequential, namely $\Gamma_{\rho_o}(\rho) = \rho$ when $\rho \leq \rho_o$ and $\Gamma_{\rho_o}(\rho) = \rho_o$ when $\rho \geq \rho_o$. Suppose the optimal algorithm Opt allocates ρ speed s machines. Within the parallelizable range, its flow plus energy cost is $F_o + E_o =$

$\frac{w}{\rho s} + \rho s^\alpha \frac{w}{\rho s} = w[\frac{1}{\rho s} + s^\alpha \frac{1}{s}]$. This is optimized with $s = (\frac{1}{\rho})^{1/\alpha}$ for a cost of $2w\frac{1}{\rho^{1-1/\alpha}}$. This is optimized with ρ as big as possible, namely ρ_o. Within the sequential range, its flow plus energy cost is $F + E = \frac{w}{s} + \rho s^\alpha \frac{w}{s}$, which is also optimized with $s = (\frac{1}{\rho})^{1/\alpha}$ giving a cost of $2w\rho^{1/\alpha}$. In contrast, this is optimized by setting ρ as small as possible, again ρ_o. In conclusion, Opt, knowing the parameter ρ_o of the speedup function, allocates $\rho = \rho_o$ machines, each of speed $s = \frac{1}{\rho_o^{1/\alpha}}$.

The instance will be chosen from a number of potential instances. For each $j \in [0, \log(m)]$, instance J_j will consist of one job with work $w_j = 2^{(1-1/\alpha)j}$ and speedup function $\Gamma_{2^j}(\rho)$. As we have seen, on instance J_j, Opt allocates $\rho_j = 2^j$ machines, each of speed $s_j = (2^j)^{-1/\alpha}$. Its cost is $F_j + E_j = \frac{w_j}{\rho_j s_j} + \rho_j {s_j}^\alpha \frac{w_j}{\rho_j s_j} = 2$.

Now consider any deterministic nonclairvoyant algorithm A allowing multiple copies of a job. Rounding the number of machines a copy is run on to a factor of two doesn't change the objective by more than a constant factor, and there is no benefit from running two copies on an equal number of machines. Since the algorithm is nonclairvoyant, it will gain no information about q until some copy finishes. Since the power function is convex, it is best for the algorithm to run each copy at constant speed. Thus we can assume that the algorithm runs $\log m$ copies of the job, with copy i run on 2^i machines at at some constant speed s_i. Note that the algorithm can set $s_i = 0$ if it doesn't want to run a copy on that many machines. The rate that the i^{th} group consumes energy is $E'_i = \rho_i {s_i}^\alpha = 2^i {s_i}^\alpha$ and the total rate that A consumes energy is $E' = \sum_i E'_i$.

Let $R_{\langle i,j\rangle} = \Gamma_{2^j}(2^i)s_i$ denote the rate that the i^{th} group of machines completes job J_j. Because we are assuming that the work completed on a job is the sum of that completed by the groups working on it, we have that $R_j = \sum_i R_{\langle i,j\rangle}$ is the rate that A completes work on job J_j and we have that $T_j = \frac{w_j}{R_j}$ is the time until the job is complete. The adversary chooses j to maximize this time. We bound this maximum as follows:

$$\begin{aligned}
\frac{1}{T} &= \min_j \frac{1}{T_j} \\
&\leq \frac{1}{\log m}\sum_j \frac{1}{T_j} \\
&= \frac{1}{\log m}\sum_j \frac{\sum_i R_{\langle i,j\rangle}}{w_j} \\
&= \frac{1}{\log m}\sum_i\sum_j \frac{\Gamma_{2^j}(2^i)s_i}{w_j} \\
&= \frac{1}{\log m}\sum_i s_i\left[\sum_{j\in[0,i]} \frac{\Gamma_{2^j}(2^i)}{w_j} + \sum_{j\in[i+1,\log m]} \frac{\Gamma_{2^j}(2^i)s_i}{w_j}\right] \\
&= \frac{1}{\log m}\sum_i s_i\left[\sum_{j\in[0,i]} \frac{2^j}{2^{(1-1/\alpha)j}} + \sum_{j\in[i+1,\log m]} \frac{2^i}{2^{(1-1/\alpha)j}}\right] \\
&= \frac{1}{\log m}\sum_i s_i\left[\sum_{j\in[0,i]} 2^{(1/\alpha)j} + 2^i\cdot \sum_{j\in[i+1,\log m]} \frac{1}{2^{(1-1/\alpha)j}}\right] \\
&= \frac{O(1)}{\log m}\sum_i s_i\left[2^{(1/\alpha)i} + 2^i\cdot\frac{1}{2^{(1-1/\alpha)i}}\right] \\
&= \frac{O(1)}{\log m}\sum_i {E'_i}^{1/\alpha}
\end{aligned}$$

Subject to $E' = \sum_i E'_i$, the sum $\sum_i E_i'^{1/\alpha}$ is maximized by setting each E'_i to $\frac{E'}{\log m}$ giving

$$\frac{1}{T} \leq \frac{O(1)}{\log m} \sum_i \left(\frac{E'}{\log m} \right)^{1/\alpha} = O(1) \left(\frac{E'}{\log m} \right)^{1/\alpha}$$

The total cost for A is

$$F_a + E_a = T + E'T = (1 + E')T \geq \Omega \left((1 + E') \left(\frac{\log m}{E'} \right)^{1/\alpha} \right) \geq \Omega \left(\log^{1/\alpha} m \right)$$

To prove the lower bound against a randomized algorithm, Yao's technique states that it is sufficient to consider a deterministic algorithm averaged over inputs according to a fixed input distribution. We bound the expected value of $\frac{1}{T}$ when J_j is chosen randomly for $j \in [0, \log m]$. What remains is to show that $1/\exp[\frac{1}{T}] \leq \exp[T]$. This follows from the fact that $\sum_j \frac{1}{T_j}$ subject to $\sum_j T_j = R$ is minimized by making all the T_j the same, which follow from the concavity of $\frac{1}{x}$. $\square$

5 A Better Analysis for Fully Parallel Work

Theorem 6. *For a single processor,* LAPS *is* $32(\alpha - 1)^2 / \ln^2 \alpha$ *competitive.*

For space reasons, this proof is moved to the appendix.

6 Conclusion

In summary, we have shown that for jobs that may have side effects, the achievable competitive ratio grows quickly with the number of processors. And for jobs without side effects, the achievable competitive ratio grows slowly with the number of processors. This shows the importance of being able to run multiple copies of a job in a setting of jobs with varying degrees of parallelizability and individually speed scalable multiprocessors.

There seem to be several interesting lines of research spawned by these results. Most obviously, the upper and lower bounds on the competitive ratio for jobs without side effects are not quite tight. It is plausible that one could obtain tight upper and lower bounds by being more careful in the analyses. One might also consider the situation where the power objective is temperature. It is not clear how to best formalize this problem. The most obvious approach is to include a constraint on temperature, that is you can not exceed the threshold of the processor. If the processor cools according to Newton's law, then the temperature is approximately the maximum energy used over any time interval of a particular length, where the length of the interval is determined by the cooling parameter of the device [3]. If the intervals are long, then the temperature constraint is essentially an energy constraint. But optimizing any reasonable scheduling objective, subject to an energy constraint, is known to be difficult [7].

Acknowledgments: We thank Tak Wah Lam, Lap Kei Lee, and Alberto Marchetti-Spaccamela for helpful discussions.

References

[1] Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, 3(4), 2007.

[2] N. Bansal, H.L. Chan, T.W. Lam, and L.K. Lee. Scheduling for bounded speed processors. In *Proc. of International Colloquium on Automata, Languages and Programming, ICALP*, pages 409 – 420, 2008.

[3] N. Bansal, T. Kimbrel, and K. Pruhs. Speed scaling to manage energy and temperature. *JACM*, 54(1), 2007.

[4] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Improved bounds for speed scaling in devices obeying the cube-root rule. In *submitted for publication*, 2008.

[5] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. In *SODA*, 2009.

[6] Nikhil Bansal, Kirk Pruhs, and Cliff Stein. Speed scaling for weighted flow time. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 805–813, 2007.

[7] Nikhil Bansal, Kirk Pruhs, and Cliff Stein. Speed scaling for weighted flow time. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 805–813, 2007.

[8] Luca Becchetti and Stefano Leonardi. Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. *J. ACM*, 51(4):517–539, 2004.

[9] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Online weighted flow time and deadline scheduling. *J. Discrete Algorithms*, 4(3):339–352, 2006.

[10] Ho-Leung Chan, Jeff Edmonds, Tak-Wah Lam, Lap-Kei Lee, Alberto Marcheti-Spaccamela, and Kirk Pruhs. Nonclairvoyant speed scaling for flow and energy. In *STACS*, 2009.

[11] J. Edmonds, S. Datta, , and P. Dymond. Tcp is competitive against a limited adversary. In *SPAA*, pages 174–183, 2003.

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

[13] Jeff Edmonds. On the competitiveness of aimd-tcp within a general network. In *LATIN*, pages 577–588, 2004.

[14] Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. In *SODA*, 2009.

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

[16] Bala Kalyanasundaram and Kirk Pruhs. Minimizing flow time nonclairvoyantly. *J. ACM*, 50(4):551–567, 2003.

[17] T.W. Lam, L.K. Lee, Isaac To, and P. Wong. Speed scaling functions for flow time scheduling based on active job count. In *Proc. of European Symposium on Algorithms, ESA*, 2008, to appear.

[18] Rick Merritt. Cpu designers debate multi-core future. *EE Times*, June 2008.

[19] Rajeev Motwani, Steven Phillips, and Eric Torng. Nonclairvoyant scheduling. *Theorertical Computer Scienc3*, 130(1):17–47, 1994.

[20] Kirk Pruhs. Competitive online scheduling for server systems. *SIGMETRICS Performance Evaluation Review*, 34(4):52–58, 2007.

[21] Kirk Pruhs, Jiri Sgall, and Eric Torng. Online scheduling. In *Handbook on Scheduling*. CRC Press, 2004.

[22] Kirk Pruhs, Patchrawat Uthaisombut, and Gerhard J. Woeginger. Getting the best response for your erg. *ACM Transactions on Algorithms*, 4(3), 2008.

[23] Julien Robert and Nicolas Schabanel. Non-clairvoyant batch sets scheduling: Fairness is fair enough. In *European Symposium on Algorithms*, pages 741–753, 2007.

[24] Julien Robert and Nicolas Schabanel. Non-clairvoyant scheduling with precedence constraints. In *Symposium on Discrete Algorithms*, pages 491–500, 2008.

A Proof of Theorem 6

Here we give our improved analysis of the LAPS algorithm for a single processor. An essentially isomorphic analysis works for parallel work in a multiprocessor setting.

We use a potential function argument. The potential function we use is identical to the one used in [10], however we do a tighter analysis using a convexity approach. For any time t, let $G_a(t)$ and $G_o(t)$ be the total flow time plus energy incurred by to time t by LAPS and the optimal algorithm Opt, respectively. To show c-competitiveness it suffices to give a potential function Φ with the following properties.

- *Boundary condition:* $\Phi = 0$ before any job is released and $\phi \geq 0$ after all jobs are completed.
- *Job arrival:* When a job is released, Φ does not increase.
- *Job completion:* When a job is completed by LAPS or Opt, Φ does not increase.
- *Running condition:* At any other time, the rate of change of $G_a(t)$ and $G_o(t)$ satisfy $\frac{dG_a(t)}{dt} + \frac{d\Phi}{dt} \leq c \cdot \frac{dG_o}{dt}$.

Let $n_a(t)$ and $s_a(t)$ be the number of active jobs and speed under LAPS and define $n_o(t)$ and $s_o(t)$ similarly for Opt. Then, $\frac{dG_a(t)}{dt} = n_a(t) + s_a(t)^\alpha$ and similarly $\frac{dG_o(t)}{dt} = n_o(t) + s_o(t)^\alpha$. We define the following potential Φ.

Potential function $\Phi(t)$. Consider any time t. For any job j, let $q_a(j,t)$ and $q_o(j,t)$ be the remaining work of j at time t under LAPS and Opt, respectively. Let $\{j_1, \ldots, j_{n_a(t)}\}$ be the set of active jobs in LAPS, ordered by their release time such that $r(j_1) \leq r_(j_2) \ldots$. Then,

$$\Phi(t) = \gamma \sum_{i=1}^{n_a(t)} (i^{1-1/\alpha} \cdot \max\{0, q_a(j_i,t) - q_o(j_i,t)\})$$

We will specify the constant γ later. We call $i^{1-1/\alpha}$ the *coefficient* of j_i

It is easily checked that the boundary condition holds. The job arrival condition holds as $q_a(j,t) - q_o(j,t) = 0$ upon arrival of job j. For job completion, if LAPS completes a job j, the term for j in Φ is removed. The coefficient of any other job either stays the same or decreases, so Φ does not increase. If Opt completes a job, Φ does not change. It remains to show the running condition.

We consider $\frac{d}{dt}\Phi$ as the combined effect of Opt and LAPS. The worst case is that Opt is processing the job with the largest coefficient, i.e., $n_a^{1-1/\alpha}$. Thus, the rate of change of Φ due to Opt is is at most $\gamma n_a^{1-1/\alpha} s_o$.

We say a job j is *lagging* at t is j has more remaining work in LAPS than Opt at t. Assume at time t, LAPS is processing ℓ jobs that are not lagging, and hence LAPS is processing $\lceil \beta n_a \rceil - \ell$ lagging jobs. Let j_i be one of these lagging jobs. We notice that j_i is among the $\lceil \beta n_a \rceil$ active jobs with the latest release times. Thus, the coefficient of j_i is at least $(n_a - \lceil \beta n_a \rceil + 1)^{1-1/\alpha}$. Also, j_i is being processed at a speed of $s_a / \lceil \beta n_a \rceil$, so $q_a(j_i,t)$ is decreasing at this rate. LAPS is processing at least $\lceil \beta n_a \rceil - \ell$ such lagging jobs, so the rate of change of Φ due to LAPS is more negative than

$$\gamma \left(\lceil \beta n_a \rceil - \ell \right) \left(n_a - \lceil \beta n_a \rceil + 1 \right)^{1-1/\alpha} \left(\frac{-s_a}{\lceil \beta n_a \rceil} \right) \leq -\gamma (1 - \frac{\ell}{\beta n_a})(1-\beta) n_a.$$

The rhs above follows as $s_a = n_a^{1/\alpha}$ and $-\lceil \beta n_a \rceil + 1 \geq -\beta n_a$.

The running condition. Our aim is to show that $n_a + s_a^\alpha + \frac{d}{dt}\Phi \leq c(n_o + s_o^\alpha)$, where c is the competitive ratio. Note that $s_a^\alpha = n_a$ and $n_o \geq \ell$. By bounding $\frac{d}{dt}\Phi$ using our previous discussion, the equation to prove becomes

$$2n_a + \gamma n_a^{1-1/\alpha} s_o - \gamma(1 - \frac{\ell}{\beta n_a})(1-\beta)n_a \leq c(\ell + s_o^\alpha)$$

Moving all terms to the right hand side, we obtain the following **convex function** of s_o.

$$f(s_o) = cs_o^\alpha - \gamma n_a^{1-1/\alpha} s_o + c\ell + \gamma(1 - \frac{\ell}{\beta n_a})(1-\beta)n_a - 2n_a \geq 0 \tag{1}$$

Since $f(s_o)$ is convex, to prove it is always positive, we only need to consider the two cases of $s_o = 0$ and s_o such that $f'(s_o) = 0$. For $s_o = 0$, (1) becomes

$$c\ell + \gamma(1 - \frac{\ell}{\beta n_a})(1-\beta)n_a - 2n_a \geq 0 \tag{2}$$

Now $f'(s_o) = 0$ when $s_o = (\frac{\gamma}{c\alpha})^{1/(\alpha-1)} n_a^{1/\alpha}$. Plugging this into (1), we need to show that

$$\gamma(\frac{1}{\alpha} - 1)(\frac{\gamma}{c\alpha})^{\frac{1}{\alpha-1}} n_a + c\ell + \gamma(1 - \frac{\ell}{\beta n_a})(1-\beta)n_a - 2n_a \geq 0 \tag{3}$$

It suffices to show (3) since it implies (2). Let us set $c = \gamma(1-\beta)/\beta$. Later we will set $\beta < 1/2$, so that $c \geq \gamma$. Now the terms containing ℓ cancel, and dividing throughout by n_a, the lhs of (3) reduces to $-\gamma(1 - \frac{1}{\alpha})(\frac{\gamma}{c\alpha})^{\frac{1}{\alpha-1}} + \gamma(1-\beta) - 2$. As $c \geq \gamma$ and $(1 - \frac{1}{\alpha}) < 1$, it suffices to show

$$-\gamma\alpha^{-\frac{1}{\alpha-1}} + \gamma(1-\beta) - 2 \geq 0. \tag{4}$$

Now, define $x = \alpha^{-1/(\alpha-1)}$, and let $y = 1 - x$. Now, $\ln x = -\ln\alpha/(\alpha-1)$, and hence

$$-\ln\alpha/(\alpha-1) = \ln x = \ln(1-y) = -y - \frac{y^2}{2} - \frac{y^3}{3} - \ldots \geq -y - y^2 - y^3 \ldots = \frac{-y}{1-y}.$$

Thus $y \geq \ln\alpha/(\ln\alpha + \alpha - 1) \geq \ln\alpha/(2(\alpha-1))$. Hence $x \leq 1 - \ln\alpha/(2(\alpha-1))$, and (4) holds if

$$\gamma\left(\frac{\ln\alpha}{2(\alpha-1)} - \beta\right) - 2 \geq 0.$$

We set $\beta = \ln\alpha/(4(\alpha-1))$ and $\gamma = 8(\alpha-1)/(\ln\alpha)$ and observe that this suffices to satisfy to inequality above. The competitive ratio $c = \gamma(1-\beta)/\beta \leq \gamma/\beta = 32(\alpha-1)^2/\ln^2\alpha$.