Adaptive Scheduling with Parallelism Feedback

Kunal Agrawal Yuxiong He Wen Jing Hsu Charles E. Leiserson

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

Multiprocessor scheduling in a shared multiprogramming environment is often structured as two-level scheduling, where a kernel-level job scheduler allots processors to jobs and a user-level task scheduler schedules the work of a job on the allotted processors. In this context, the number of processors allotted to a particular job may vary during the job's execution, and the task scheduler must adapt to these changes in processor resources. For overall system efficiency, the task scheduler should also provide parallelism feedback to the job scheduler to avoid the situation where a job is allotted processors that it cannot use productively.

We present an adaptive task scheduler for multitasked jobs with dependencies that provides continual parallelism feedback to the job scheduler in the form of requests for processors. Our scheduler guarantees that a job completes near optimally while utilizing at least a constant fraction of the allotted processor cycles. Our scheduler can be applied to schedule data-parallel programs, such as those written in High Performance Fortran (HPF), *Lisp, C*, NESL, and ZPL.

Our analysis models the job scheduler as the task scheduler's adversary, challenging the task scheduler to be robust to the system environment and the job scheduler's administrative policies. For example, the job scheduler can make available a huge number of processors exactly when the job has little use for them. To analyze the performance of our adaptive task scheduler under this stringent adversarial assumption, we introduce a new technique called "trim analysis," which allows us to prove that our task scheduler performs poorly on at most a small number of time steps, exhibiting near-optimal behavior on the vast majority.

To be precise, suppose that a job has work T_1 and critical-path length T_∞ and is running on a machine with P processors. Using trim analysis, we prove that our scheduler completes the job in $O(T_1/\widetilde{P} + T_\infty + L \lg P)$ time steps, where L is the length of a scheduling quantum and $\widetilde{P}$ denotes the $O(T_\infty + L \lg P)$-trimmed availability. This quantity is the average of the processor availability over all time steps excluding the $O(T_\infty + L \lg P)$ time steps with the highest processor availability. When $T_1/T_\infty \gg \widetilde{P}$ (the job's parallelism dominates the $O(T_\infty + L \lg P)$-trimmed availability), the job achieves nearly perfect linear speedup. Conversely, when $T_1/T_\infty \ll \widetilde{P}$, the asymptotic running time of the job is nearly the length of its critical path.

Categories and Subject Descriptors D.4.1 [*Software*]: Operating Systems - process management; F.2 [*Theory of Computation*]: Analysis of Algorithms and Problem Complexity.

General Terms Algorithms, Performance, Theory.

Keywords Adaptive scheduling, Adversary, Critical path, Data-parallel computing, Greedy scheduling, Instantaneous parallelism, Job scheduling, Multiprocessing, Multiprogramming, Parallelism feedback, Parallel computation, Processor allocation, Task scheduling, Two-level scheduling, Space sharing, Trim analysis, Work.

1. Introduction

The scheduling of a collection of parallel jobs onto a multiprocessor is an old and well-studied topic of research [15,17,18,22,31,36, 39, 50, 53, 54]. In this paper, we study so-called ***space-sharing*** [25] for parallel jobs, where jobs occupy disjoint processor resources, as opposed to ***time-sharing*** [25], where different jobs may share the same processor resources at different times. Space-sharing schedulers can be implemented using a two-level strategy [25]: a kernel-level ***job scheduler*** which allots processors to jobs, and a user-level ***task scheduler*** which schedules the tasks belonging to a given job onto the allotted processors. In this paper, we study how the task scheduler for a job can provide provably effective feedback to the job scheduler on the job's parallelism. Our adaptive scheduler can be applied to schedule data-parallel languages such as High Performance Fortran (HPF) [26], *Lisp [35], C* [45], NESL [4, 6], and ZPL [16].

Like earlier research on task scheduling of data-parallel languages [3, 6, 24, 33, 43], we model the execution of a parallel job as a dynamically unfolding directed acyclic graph (dag) of ***tasks***, where each node in the dag represents a unit-time task. An edge represents a serial dependency between tasks. The ***work*** T_1 corresponds to the total number of unit-time tasks in the dag and the ***critical-path length*** T_∞ corresponds to the length of the longest chain of dependencies in the dag. A task becomes ***ready*** to be executed when all its parents have been executed. Each job has its own task scheduler, and the task scheduler operates in an online manner, oblivious to the future characteristics of the dynamically unfolding dag.

Most prior work on task scheduling for multitasked jobs deals with ***nonadaptive*** scheduling [5, 6, 10, 14, 30, 43], where the job scheduler allots a fixed number of processors to the job for its entire lifetime. For jobs whose parallelism is unknown in advance

This research was supported in part by the Singapore-MIT Alliance and NSF Grant ACI-0324974. Yuxiong He is a Visiting Scholar at MIT CSAIL and a Ph.D. candidate at the National University of Singapore. Wen Jing Hsu is a Visiting Scientist at MIT CSAIL and Associate Professor at Nanyang Technological University.

PPoPP'06 March 29–31, 2006, New York, New York, USA.

and which may change during execution, this strategy may waste processor cycles [50], because a job with low parallelism may be allotted more processors than it can productively use. Moreover, in a multiprogrammed environment, nonadaptive scheduling may not allow a new job to start, because existing jobs may already be using most of the processors.

With ***adaptive*** scheduling [1] (called "dynamic" scheduling in many papers), the job scheduler can change the number of processors allotted to a job while the job is executing. Thus, new jobs can enter the system, because the job scheduler can simply recruit processors from the already executing jobs and allot them to the new job. Unfortunately, as with a nonadaptive scheduler, this strategy may cause waste, because a job with low parallelism may still be allotted more processors than it can productively use.

The solution we present is an adaptive scheduling strategy where the task scheduler provides ***parallelism feedback*** to the job scheduler so that when a job cannot use many processors, those processors can be reallotted to jobs with ample need. Based on this parallelism feedback, the job scheduler adaptively changes the allotment of processors according to the availability of processors in the current system environment and the job scheduler's administrative policy.

The question of how the job scheduler should partition the multiprocessor among the various jobs has been studied extensively [17, 18, 22, 23, 31, 36, 38, 39, 41, 46, 47, 55], but the administrative policy of the job scheduler is not the focus of this paper. Instead, we study the problem of how the task scheduler provides effective parallelism feedback to the job scheduler without knowing the future progress of the job, the future availability of processors, or the administrative priorities of the job scheduler.

Various researchers [17, 18, 31, 39, 55] have used the notion of ***instantaneous parallelism***,[1] the number of processors the job can effectively use at the current moment, as the parallelism feedback to the job scheduler. Although using instantaneous parallelism for parallelism feedback is simple, it can cause gross misallocation of processor resources [48]. For example, the parallelism of a job may change substantially during a scheduling quantum, alternating between parallel and serial phases. The sampling of instantaneous parallelism at a scheduling event between quanta may lead the task scheduler to request either too many or too few processors depending on which phase is currently active, whereas the desirable request might be something in between. Consequently, the job may systematically waste processor cycles on the one hand or take too long to complete on the other.

In this paper, we present an adaptive greedy task scheduler, called A-GREEDY, which provides parallelism feedback. A-GREEDY guarantees not to waste many processor cycles while simultaneously ensuring that the job completes quickly. Instead of using instantaneous parallelism, A-GREEDY provides parallelism feedback to the job scheduler based on a single summary statistic and the job's behavior on the previous quantum. Even though A-GREEDY provides parallelism feedback using the past behavior of the job and we do not assume that the job's future parallelism is correlated with its history of parallelism, our analysis shows that A-GREEDY schedules the job well with respect to both waste and completion time.

Our scheduling model is as follows. We assume that time is broken into a sequence of equal-size ***scheduling quanta*** $1, 2, \ldots$ of length L, and the job scheduler is free to reallocate processors between quanta. The quantum length L is a system configuration parameter chosen to be long enough to amortize the time to reallocate processors among the various jobs and the time to perform various other bookkeeping for scheduling, including time for the task scheduler to communicate with the job scheduler, which typically involves a system call. Between quanta $q-1$ and q, the task scheduler determines its job's ***desire*** d_q, which is the number of processors the job wants for quantum q. The task scheduler provides the desire d_q to the job scheduler as its parallelism feedback. The job scheduler follows some processor allocation policy to determine the ***processor availability*** p_q, which is the number of processors the job is entitled to get for the quantum q. To make the task scheduler robust to different system environments and administrative policies, our analysis of A-GREEDY assumes that the job scheduler decides the availability of processors as an adversary.

The number of processors the job receives for quantum q is the job's ***allotment*** $a_q = \min\{d_q, p_q\}$, which is the smaller of its desire and the processor availability. For example, suppose that a job requests $d_q = 5$ processors and the job scheduler decides that the availability is $p_q = 10$. Then, the job is allotted $a_q = \min\{d_q, p_q\} = \min\{5, 10\} = 5$ processors. If the availability is only $p_q = 3$, however, the job's allotment is $a_q = \min\{5, 3\} = 3$. Once a job is allotted its processors, the allotment does not change during the quantum. Consequently, the task scheduler must do a good job before a quantum of estimating how many processors it will need for all L time steps of the quantum, as well as do a good job of scheduling the tasks on the allotted processors.

In an adaptive setting where the number of processors allotted to a job can change during execution, both $T_1/\overline{P}$ and T_∞ are lower bounds on the running time, where $\overline{P}$ is the mean of the processor availability during the computation. An adversarial job scheduler, however, can prevent any thread scheduler from providing good speedup with respect to the mean availability $\overline{P}$ in the worst case. For example, if the adversary chooses a huge number of processors for the job's processor availability just when the job has little instantaneous parallelism, no adaptive scheduling algorithm can effectively utilize the available processors on that quantum.

We introduce a technique called ***trim analysis*** to analyze the time bound of adaptive thread schedulers under these adversarial conditions. From the field of statistics, trim analysis borrows the idea of ignoring a few "outliers." A ***trimmed mean***, for example, is calculated by discarding a certain number of lowest and highest values and then computing the mean of those that remain. For our purposes, it suffices to trim the availability from just the high side. For a given value R, we define the ***R-high-trimmed mean availability*** as the mean availability after ignoring the R steps with the highest availability. A good thread scheduler should provide linear speedup with respect to an R-trimmed availability, where R is as small as possible.

Our A-GREEDY algorithm uses a greedy scheduler [9, 14, 30], which, when coupled with A-GREEDY's parallelism-feedback strategy, is provably effective. We prove that A-GREEDY guarantees linear speedup with respect to the $O(T_\infty + L \lg P)$-trimmed availability. Specifically, consider a job with work T_1 and critical-path length T_∞ running on a machine with P processors and a scheduling quantum of length L. A-GREEDY completes the job in $O(T_1/\widetilde{P} + T_\infty + L \lg P)$ time steps, where $\widetilde{P}$ denotes the $O(T_\infty + L \lg P)$-trimmed availability. Thus, the job achieves linear speed up with respect to $\widetilde{P}$ when $T_1/T_\infty \gg \widetilde{P}$, that is, when the job's parallelism dominates the $O(T_\infty + L \lg P)$-trimmed availability. In addition, we prove that the total number of processor cycles wasted by the job is $O(T_1)$, representing at most a constant factor overhead.

The remainder of this paper is organized as follows. Section 2 presents the adaptive greedy task scheduler A-GREEDY. Section 3 provides a trim analysis for the special case of A-GREEDY when the scheduling quantum has length 1, *i.e.* task schedulers request the processors from the job scheduler at each time step, and Sec-

[1] These researchers actually use the general term "parallelism," but we prefer the more descriptive term.

tion 4 extends this trim analysis to the general case. Section 5 discusses how A-GREEDY can be applied to schedule data-parallel jobs with bounded time, space, and waste. Section 6 describes related work, and Section 7 offers some concluding remarks.

2. The Adaptive Greedy Algorithm

This section presents the adaptive greedy task scheduler A-GREEDY. Before each quantum, A-GREEDY provides parallelism feedback to the job scheduler based on the job's history of utilization using a simple multiplicative-increase, multiplicative-decrease algorithm. A-GREEDY classifies quanta as "satisfied" versus "deprived" and "efficient" versus "inefficient." Of the four possibilities of classification, however, A-GREEDY only uses three: inefficient, efficient and satisfied, and efficient and deprived. Using this three-way classification and the job's desire for the previous quantum, it computes the desire for the next quantum.

To classify a quantum q as satisfied versus deprived, A-GREEDY compares the job's allotment a_q with its desire d_q. The quantum q is ***satisfied*** if $a_q = d_q$, that is, the job receives as many processors as A-GREEDY requested on its behalf from the job scheduler. Otherwise, if $a_q < d_q$, the quantum is ***deprived***, because the job did not receive as many processors as A-GREEDY requested.

Classifying a quantum as efficient versus inefficient is more complicated. We define the ***usage*** u_q of a quantum q as the amount of work completed by the job during the quantum, which is to say, the total number of unit-time tasks in the dag that were completed during the quantum. The maximum possible usage for a quantum q is La_q, where L is the length of quanta and a_q is the job's allotment for quantum q. A-GREEDY uses a ***utilization parameter*** δ, where $0 < \delta \leq 1$, as a threshold to differentiate between efficient and inefficient quanta. Typical values for δ might be 90–95%. We call a quantum q ***efficient*** if $u_q \geq \delta La_q$, that is, the usage is at least a δ fraction of the maximum possible usage, in which case the job wastes few (at most $(1-\delta)La_q$) processor cycles. We call a quantum q ***inefficient*** otherwise.

A-GREEDY calculates the desire d_q of the current quantum q based on the previous desire d_{q-1} and the three-way classification of quantum $q-1$ as inefficient, efficient and satisfied, and efficient and deprived. The initial desire is $d_1 = 1$. A-GREEDY uses a ***responsiveness parameter*** $\rho > 1$ to determine how quickly the scheduler responds to changes in parallelism. Typical values of ρ might range between 1.2 and 2.0. Figure 1 shows the pseudocode of A-GREEDY for one quantum. The algorithm takes as input the quantum q, the utilization parameter δ, and the responsiveness parameter ρ. Intuitively, it operates as follows:

- If quantum $q-1$ was inefficient, A-GREEDY overestimated the desire. In this case, A-GREEDY does not care whether the quantum is satisfied or deprived, and it decreases the desire (line 4) in quantum q.
- If quantum $q-1$ was efficient and satisfied, the job effectively utilized the processors that A-GREEDY requested on its behalf. Thus, A-GREEDY speculates that the job can use more processors and increases the desire (line 6) in quantum q.
- If quantum $q-1$ was efficient but deprived, the job used all the processors it was allotted, but A-GREEDY had requested more processors for the job than the job actually received from the job scheduler. Since A-GREEDY has no evidence whether the job could have used all the processors requested, it maintains the same desire (line 7) in quantum q.

Remarkably, this simple algorithm provides strong guarantees on waste and performance.

```
A-GREEDY(q, δ, ρ)
1  if q = 1
2     then d_q ← 1                      ▷ base case
3  elseif u_{q-1} < Lδa_{q-1}
4     then d_q ← d_{q-1}/ρ              ▷ inefficient
5  elseif a_{q-1} = d_{q-1}
6     then d_q ← ρd_{q-1}               ▷ efficient and satisfied
7  else d_q ← d_{q-1}                   ▷ efficient and deprived
8  Report desire d_q to the job scheduler.
9  Receive allotment a_q from the job scheduler.
10 Greedily schedule on a_q processors for L time steps.
```

Figure 1: Pseudocode for the adaptive greedy algorithm. A-GREEDY provides parallelism feedback to a job scheduler in the form of a desire for processors. Before quantum q, A-GREEDY uses the previous quantum's desire d_{q-1}, allotment a_{q-1}, and usage u_{q-1} to compute the current quantum's desire d_q based on the utilization parameter δ and the responsiveness parameter ρ.

3. Trim Analysis for Unit Quanta

This section uses a trim analysis to analyze A-GREEDY for the special case where $L = 1$, that is, where each quantum is a ***unit*** quantum consisting of only a single time step. For unit quanta, adaptive scheduling can be done efficiently using instantaneous parallelism as feedback. Surprisingly, A-GREEDY's algorithm for desire estimation, which only uses historical information, provides nearly as good time bounds. Moreover, as we shall see in Section 4, these bounds can be extended to the case when $L \gg 1$. In contrast, although a task scheduler based on instantaneous parallelism can be used for unit quanta, a straightforward extension to $L \gg 1$ would require the task scheduler to know the future parallelism of the job for all L time steps of the quantum. The analysis for unit quanta given in this section gives intuition for the effectiveness of A-GREEDY's strategy for desire estimation.

For unit quanta, we shall prove that A-GREEDY with utilization parameter $\delta = 1$ completes a job with work T_1 and critical-path length T_∞ in at most $T \leq T_1/\widetilde{P} + 2T_\infty + \log_\rho P + 1$ time steps, where P is the number of processors in the machine and $\widetilde{P}$ is the $(2T_\infty + \log_\rho P + 1)$-trimmed availability. In contrast, a greedy task scheduler that uses instantaneous parallelism as feedback completes the job in at most $T \leq T_1/\widetilde{P} + T_\infty$ time steps, where $\widetilde{P}$ is the T_∞-trimmed availability. Thus, even without up-to-date information on instantaneous parallelism, A-GREEDY operates nearly as efficiently. Moreover, the total number of processor cycles wasted by A-GREEDY in the course of the computation is bounded by ρT_1. (Instantaneous parallelism wastes none.)

To prove the completion-time bounds, we use a trim analysis. We label each quantum as either ***accounted*** or ***deductible***. Accounted quanta are those where $u_q = p_q$, that is, the usage equals the processor availability. The deductible quanta are those where $u_q < p_q$. Our trim analysis will show that when we ignore the relatively few deductible quanta, we obtain linear speedup on the more numerous accounted quanta.

We first relate the labeling of accounted and deductible to the three-way classification of quanta as inefficient, efficient and satisfied, and efficient and deprived.

Inefficient: In an inefficient quantum q, we have $u_q < a_q \leq p_q$, that is, the job uses fewer processors than it was allotted, and therefore it uses fewer processors than those available. Thus, inefficient quanta are deductible quanta, irrespective of whether they were satisfied or deprived.

Efficient and satisfied: On an efficient quantum q, we have $u_q = a_q$. Since $a_q = \min\{p_q, d_q\}$ by definition, on a satisfied quantum, we have $a_q = d_q \le p_q$. Thus, we have $u_q \le p_q$. Since we cannot guarantee that $u_q = p_q$, we assume pessimistically that quantum q is deductible.

Efficient and deprived: As before, on an efficient quantum q, we have $u_q = a_q$. On a deprived quantum, we have by definition that $a_q < d_q$, and since $a_q = \min\{p_q, d_q\}$, we have $a_q = p_q$. Thus, we have $u_q = a_q = p_q$, and quantum q is accounted.

Time Analysis

We prove the completion time bound of A-GREEDY by bounding the number of deductible and accounted quanta separately. We use a potential function argument to prove that the number of deductible quanta is at most $2T_\infty + \log_\rho P + 1$. We then show that the number of accounted quanta is at most T_1/P_A, where T_1 is the total work and $\widetilde{P}$ and P_A is the mean availability on accounted quanta. Thus, the total time to complete the job is at most $T_1/P_A + 2T_\infty + \log_\rho P + 1$, which is the sum of the number of accounted and deductible quanta, since each quantum consists of a single time step. Finally, we show that $P_A \ge \widetilde{P}$, where $\widetilde{P}$ is the $(2T_\infty + L\log_\rho P + 1)$-trimmed availability, which yields the desired result.

Our analysis uses a characterization of greedy scheduling based on whether the job uses all its allotted processors on a given step. We define a step to be ***complete*** if the job uses all the allotted processors in the step and ***incomplete*** if the job does not use all the available processors. In the special case of A-GREEDY with unit quanta, an inefficient quantum consists of a single incomplete step and an efficient quantum consists of a single complete step. The following lemma from the literature [7, 10, 19] shows that whenever a greedy scheduler (including A-GREEDY) schedules an incomplete step, the job makes progress on its critical path.

LEMMA 1. *Any greedy scheduler reduces the length of a job's remaining critical path by* 1 *after every incomplete step.* □

We next bound the maximum desire during the course of the computation.

LEMMA 2. *Suppose that* A-GREEDY *schedules a job on a machine with P processors. If ρ is* A-GREEDY*'s responsiveness parameter, then for every quantum q, the job's desire satisfies $d_q \le \rho P$.*

Proof. We use induction on the number of quanta. The base case $d_1 = 1$ holds trivially. If a given quantum $q-1$ was inefficient, the desire d_q decreases, and thus $d_q < d_{q-1} \le \rho P$ by induction. If quantum $q-1$ was efficient and satisfied, then $d_q = \rho d_{q-1} = \rho a_{q-1} \le \rho P$. If quantum $q-1$ was efficient and deprived, then $d_q = d_{q-1} \le \rho P$ by induction. □

The deductible quanta for A-GREEDY are either inefficient or efficient and satisfied. The next lemma bounds their number.

LEMMA 3. *Suppose that* A-GREEDY *schedules a job with critical-path length T_∞ on a machine with P processors. If ρ is* A-GREEDY*'s responsiveness parameter, $\delta = 1$ is its utilization parameter, and $L = 1$ is the quantum length, then the schedule produces at most $2T_\infty + \log_\rho P + 1$ deductible quanta.*

Proof. We use a potential-function argument based on the job's desire d_q before quantum q. Define the potential before quantum q to be

$$\Phi(q) = 2T_\infty^q - \log_\rho d_q \ ,$$

where T_∞^q denotes the length of the remaining critical path before quantum q is executed, that is, the length of the longest path in the unexecuted dag. The initial potential is

$$\begin{aligned}\Phi(1) &= 2T_\infty - \log_\rho d_1 \\ &= 2T_\infty \ ,\end{aligned}$$

since the desire in the first quantum is $d_1 = 1$. If the job executes for Q quanta, the final potential is

$$\begin{aligned}\Phi(Q+1) &= 2T_\infty^{Q+1} - \log_\rho d_{Q+1} \\ &\ge 0 - \log_\rho(\rho P) \\ &= -\log_\rho P - 1 \ ,\end{aligned}$$

by Lemma 2. Since the potential starts at $2T_\infty$ and is at least $-\log_\rho P - 1$ at the end of the computation, the total decrease of the potential is $\Phi(1) - \Phi(Q+1) \le 2T_\infty + \log_\rho P + 1$.

We now compute the decrease in potential during each quantum based on the three-way classification. Each case will use the fact that the decrease in potential during any quantum q is

$$\begin{aligned}\Delta\Phi &= \Phi(q) - \Phi(q+1) \\ &= (2T_\infty^q - \log_\rho d_q) - (2T_\infty^{q+1} - \log_\rho d_{q+1}) \\ &= 2(T_\infty^q - T_\infty^{q+1}) - (\log_\rho d_q - \log_\rho d_{q+1}) \ .\end{aligned}$$

Inefficient: An inefficient quantum q consists of a single incomplete step. After an incomplete step, the length of the remaining critical path reduces by 1 (Lemma 1). Moreover, we have $d_{q+1} = d_q/\rho$, since A-GREEDY reduces the desire after an inefficient quantum. Thus, the decrease in potential after an inefficient quantum is

$$\begin{aligned}\Delta\Phi &= 2(T_\infty^q - T_\infty^{q+1}) - (\log_\rho d_q - \log_\rho d_{q+1}) \\ &= 2\left(T_\infty^q - (T_\infty^q - 1)\right) - \left(\log_\rho d_q - \log_\rho(d_q/\rho)\right) \\ &= 2(1) - (1) \\ &= 1 \ .\end{aligned}$$

Efficient and satisfied: A-GREEDY increases the desire after every efficient and satisfied quantum ($d_{q+1} = \rho d_q$). The remaining critical-path length never increases. Thus, the decrease in potential is

$$\begin{aligned}\Delta\Phi &= 2(T_\infty^q - T_\infty^{q+1}) - (\log_\rho d_q - \log_\rho d_{q+1}) \\ &\ge 0 - \left(\log_\rho d_q - \log_\rho(\rho d_q)\right) \\ &= 1 \ .\end{aligned}$$

Efficient and deprived: After efficient and satisfied quanta, A-GREEDY maintains the previous desire ($d_{q+1} = d_q$), and, as before, the critical-path length never increases. Thus, the decrease in potential is

$$\begin{aligned}\Delta\Phi &= 2(T_\infty^q - T_\infty^{q+1}) - (\log_\rho d_q - \log_\rho d_{q+1}) \\ &\ge 0 \ .\end{aligned}$$

Thus, the potential never increases, and it decreases by at least 1 after every deductible quantum. Thus, the number of deductible quanta is at most $2T_\infty + \log_\rho P + 1$, the total decrease in potential. □

We now bound the number of accounted quanta.

LEMMA 4. *Suppose that* A-GREEDY *schedules a job with work T_1. If $\delta = 1$ is* A-GREEDY*'s utilization parameter and $L = 1$ is the quantum length, then the schedule produces at most T_1/P_A accounted quanta, where P_A is the mean availability on accounted quanta.*

Proof. Let A be the set of accounted quanta, and D be the set of deductible quanta. The mean availability on accounted quanta is $P_A = (1/|A|)\sum_{q\in A} p_q$. The total number of tasks executed over the course of the computation is $T_1 = \sum_{q\in A\cup D} u_q$, since each of the T_1 tasks is executed exactly once in either an accounted or

a deductible quantum. Since accounted quanta are those for which $u_q = p_q$, we have

$$\begin{aligned} T_1 &= \sum_{q \in A \cup D} u_q \\ &\geq \sum_{q \in A} u_q \\ &= \sum_{q \in A} p_q \\ &= |A| P_A \end{aligned}$$

Thus, the number of accounted quanta is $|A| \leq T_1/P_A$. □

We can now bound the completion time of a job scheduled by A-GREEDY with unit quanta.

THEOREM 5. *Suppose that* A-GREEDY *schedules a job with work T_1 and critical-path length T_∞ on a machine with P processors. If ρ is* A-GREEDY*'s responsiveness parameter, $\delta = 1$ is its utilization parameter, and $L = 1$ is the quantum length, then* A-GREEDY *completes the job in*

$$T \leq T_1/\widetilde{P} + 2T_\infty + \log_\rho P + 1$$

time steps, where $\widetilde{P}$ is the $(2T_\infty + \log_\rho P + 1)$-trimmed availability.

Proof. The proof is a trim analysis. Let A be the set of accounted quanta, and D be the set of deductible quanta. Lemma 3 shows that there are $|D| \leq 2T_\infty + L \lg P + 1$ deductible time steps, since each quantum consists of a single time step. We have $P_A \geq \widetilde{P}$, since the mean availability on the accounted time steps (we trim the $|D|$ deductible steps) must be at least the $(2T_\infty + L \lg P + 1)$-trimmed availability (we trim the $2T_\infty + L \lg P + 1$ steps that have the highest availability). From Lemma 4, the number of accounted quanta is $|A| \leq T_1/P_A \leq T_1/\widetilde{P}$, and since $T = L(|A| + |D|)$, the desired time bound follows. □

Waste Analysis

We now prove the waste bound for A-GREEDY with unit quanta. Let $w_q = a_q - u_q$ be the waste of quantum q. In efficient quanta, the usage is $u_q = a_q$, and the waste is $w_q = 0$. Therefore, the job wastes processor cycles only on inefficient quanta. The next theorem shows that the waste on inefficient quanta can be amortized against the work done on efficient quanta.

THEOREM 6. *Suppose that* A-GREEDY *schedules a job with work T_1 on a machine. If ρ is* A-GREEDY*'s responsiveness parameter, $\delta = 1$ is its utilization parameter, and $L = 1$ is the quantum length, then* A-GREEDY *wastes at most ρT_1 processor cycles in the course of its computation.*

Proof. We use a potential-function argument based on the job's desire d_q before quantum q. Define the potential $\Psi(q)$ before quantum q as

$$\Psi(q) = \rho T_1^q + \frac{\rho}{\rho - 1} d_q \,,$$

where T_1^q is the total number of unexecuted tasks in the computation before quantum q. Thus, the initial potential is

$$\begin{aligned} \Psi(1) &= \rho T_1^1 + \frac{\rho}{\rho - 1} d_1 \\ &= \rho T_1 + \rho/(\rho - 1) \,, \end{aligned}$$

since $d_1 = 1$. If the job executes for Q quanta, the final potential is

$$\begin{aligned} \Psi(Q+1) &= \rho T_1^{Q+1} + \frac{\rho}{\rho - 1} d_{Q+1} \\ &\geq 0 + \rho/(\rho - 1) \,, \\ &= \rho/(\rho - 1) \,, \end{aligned}$$

since the desire d_q of any quantum q is at least 1. Therefore the total decrease in potential is $\Psi(1) - \Psi(Q+1) \leq \rho T_1$.

Based on the three-way classification, we shall show that if the waste on quantum q is $w_q = a_q - u_q$, then the potential decreases by at least w_q during the quantum. Each way will use the fact that the decrease in potential during any quantum q is

$$\begin{aligned} \Delta\Psi_q &= \Psi(q) - \Psi(q+1) \\ &= \left(\rho T_1^q + \frac{\rho}{\rho - 1} d_q\right) - \left(\rho T_1^{q+1} + \frac{\rho}{\rho - 1} d_{q+1}\right) \\ &= \rho(T_1^q - T_1^{q+1}) + \frac{\rho}{\rho - 1}(d_q - d_{q+1}) \,. \end{aligned}$$

Inefficient: For any quantum q, $w_q < a_q$, which is to say, the number of processor cycles wasted is less than the total number of processor cycles allotted. Since the allotment is $a_q \leq d_q$, we have $w_q < d_q$. After an inefficient quantum q, A-GREEDY reduces the desire to be $d_{q+1} = d_q/\rho$. Thus, the decrease in potential is

$$\begin{aligned} \Delta\Psi_q &= \rho(T_1^q - T_1^{q+1}) + \frac{\rho}{\rho - 1}(d_q - d_{q+1}) \\ &> \frac{\rho}{\rho - 1}(d_q - d_q/\rho) \\ &= d_q \\ &> w_q \,. \end{aligned}$$

Efficient and satisfied: Since no processor cycles are wasted on any efficient quantum q, we have $w_q = 0$ and the remaining work reduces by $u_q = a_q$. On an efficient and satisfied quantum q, the allotment is the same as the desire ($a_q = d_q$) and A-GREEDY increases the desire ($d_{q+1} = \rho d_q$) after the quantum. Thus, the decrease in potential is

$$\begin{aligned} \Delta\Psi_q &= \rho(T_1^q - T_1^{q+1}) + \frac{\rho}{\rho - 1}(d_q - d_{q+1}) \\ &= \rho a_q + \frac{\rho}{\rho - 1}(d_q - \rho d_q) \\ &= \rho d_q - \rho d_q \\ &= 0 \\ &= w_q \,. \end{aligned}$$

Efficient and deprived: On any efficient quantum q, we have $w_q = 0$ and the amount of remaining work reduces by $u_q = a_q$. Since the quantum q is efficient and deprived, we have $d_{q+1} = d_q$, because A-GREEDY maintains the previous desire. Therefore, the decrease in potential is

$$\begin{aligned} \Delta\Psi &= \rho(T_1^q - T_1^{q+1}) + \frac{\rho}{\rho - 1}(d_q - d_{q+1}) \\ &= \rho a_q + 0 \\ &> 0 \\ &= w_q \,. \end{aligned}$$

In all three cases, if the job wastes w_q processors in quantum q, the potential decreases by at least w_q. Consequently, the total waste during the course of the computation is at most ρT_1, the total decrease in potential. □

4. Trim Analysis of the General Case

We now use a trim analysis to analyze the general case of A-GREEDY when each scheduling quantum has L time steps, δ is the utilization parameter, ρ is the responsiveness parameter, and P is the number of processors in the machine. For a job with work T_1 and critical-path length T_∞, A-GREEDY achieves the following bounds on running time and waste, where $\widetilde{P}$ is the $(2T_\infty/(1-\delta) + L \log_\rho P + L)$-

trimmed availability:

$$\begin{aligned} T &\leq \frac{T_1}{\delta \widetilde{P}} + \frac{2T_\infty}{1-\delta} + L \log_\rho P + L \,, \\ W &\leq \frac{1+\rho-\delta}{\delta} T_1 \,. \end{aligned}$$

As in Section 3, we label each quantum as either accounted or deductible. Recall that a quantum q of length L and processor availability p_q has a total of Lp_q processor cycles available. Accounted quanta are those for which $u_q \geq \delta L p_q$, that is, the job uses at least a δ fraction of all available processor cycles. The deductible quanta are those for which $u_q < L\delta p_q$. By the same logic as in Section 3, inefficient quanta or efficient and satisfied quanta are labeled deductible. Efficient and deprived quanta, on the other hand, are labeled accounted.

Time Analysis

We bound the accounted and deductible quanta separately. We first show how inefficient quanta affect the remaining critical path of the job.

LEMMA 7. A-GREEDY *reduces the length of a job's remaining critical path by at least* $(1-\delta)L$ *after every inefficient quantum, where* δ *is* A-GREEDY*'s utilization parameter and* L *is the quantum length.*

Proof. The total number of tasks completed in an inefficient quantum q is less than $\delta L a_q$. Therefore, there can be at most δL complete steps in an inefficient quantum, since on a complete step, the job uses all the allotted processors, completing a_q tasks. Since there are L time steps in a quantum, there are at least $(1-\delta)L$ incomplete steps. Thus, the critical path reduces by at least $(1-\delta)L$, since Lemma 1 shows that every incomplete step reduces the critical path by 1. □

The next lemma bounds the number of deductible quanta.

LEMMA 8. *Suppose that* A-GREEDY *schedules a job with critical-path length* T_∞ *on a machine with* P *processors. If* ρ *is* A-GREEDY*'s responsiveness parameter,* δ *is its utilization parameter, and* L *is the quantum length, then the schedule produces at most* $2T_\infty/(1-\delta)L + \log_\rho P + 1$ *deductible quanta.*

Proof. We use a potential-function argument as in Lemma 3. Define the potential before quantum q as

$$\Phi(q) = 2T_\infty^q/(1-\delta)L - \log_\rho d_q \,,$$

where T_∞^q is the remaining critical path before quantum q. If the job completes in Q quanta, the total decrease in potential is

$$\begin{aligned} \Phi_1 - \Phi_{Q+1} &= \frac{2T_\infty - 0}{(1-\delta)L} - (\log_\rho 1 - \log_\rho d_{Q+1}) \\ &\leq \frac{2T_\infty}{(1-\delta)L} + \log_\rho P + 1 \,, \end{aligned}$$

since $d_{Q+1} \leq \rho P$ by Lemma 2.

We can compute the decrease in potential during each quantum based on the three-way classification. By arguments similar to those in Lemma 3, we can show that the potential decreases by at least 1 after every deductible quantum and that it never increases. Therefore, the total number of deductible quanta is at most $2T_\infty/((1-\delta)L) + \log_\rho P + 1$, the total decrease in potential. □

We now bound the number of accounted quanta.

LEMMA 9. *Suppose that* A-GREEDY *schedules a job with work* T_1*. If* δ *is* A-GREEDY*'s utilization parameter and* L *is the quantum length, then the schedule produces at most* $T_1/\delta L P_A$ *accounted quanta, where* P_A *is the mean availability on accounted quanta.*

Proof. Let A be the set of accounted quanta, and let D be the set of deductible quanta. The mean availability on accounted quanta is $P_A = (1/|A|)\sum_{q\in A} p_q$. The total number of tasks executed in the course of the computation is $T_1 = \sum_{q\in A\cup D} u_q$. Since the accounted quanta are those for which $u_q \geq \delta L p_q$, we have

$$\begin{aligned} T_1 &= \sum_{q\in A\cup D} u_q \\ &\geq \sum_{q\in A} u_q \\ &\geq \sum_{q\in A} \delta L p_q \\ &= \delta L |A| P_A \,. \end{aligned}$$

Therefore, the total number of accounted quanta is at most $|A| \leq T_1/\delta L P_A$. □

The next theorem provides the time bound for A-GREEDY.

THEOREM 10. *Suppose that* A-GREEDY *schedules a job with work* T_1 *and critical-path length* T_∞ *on a machine with* P *processors. If* ρ *is* A-GREEDY*'s responsiveness parameter,* δ *is its utilization parameter, and* L *is the quantum length, then* A-GREEDY *completes the job in*

$$T \leq T_1/\delta\widetilde{P} + 2T_\infty/(1-\delta) + L\log_\rho P + L$$

time steps, where $\widetilde{P}$ *is the* $(2T_\infty/(1-\delta) + L\log_\rho P + L)$*-trimmed availability.*

Proof. The proof is a trim analysis. Let A be the set of accounted quanta, and D be the set of deductible quanta. Lemma 8 shows that there are $|D| \leq 2T_\infty/(1-\delta)L + \log_\rho P + 1$ deductible quanta, and hence at most $L|D| = 2T_\infty/(1-\delta) + L\log_\rho P + L$ time steps belong to deductible quanta. We have that $P_A \geq \widetilde{P}$, since the mean availability on the accounted time steps (we trim the $L|D|$ deductible steps) must be at least the $(2T_\infty/(1-\delta) + L\log_\rho P + L)$-trimmed availability (we trim the $2T_\infty/(1-\delta) + L\log_\rho P + L$ steps that have the highest availability). From Lemma 9, the number of accounted quanta is $|A| \leq T_1/\delta P_A \leq T_1/\delta\widetilde{P}$, and since $T = L(|A| + |D|)$, the desired time bound follows. □

Waste Analysis

We now prove the waste bound for A-GREEDY.

THEOREM 11. *Suppose that* A-GREEDY *schedules a job with work* T_1 *on a machine. If* ρ *is* A-GREEDY*'s responsiveness parameter,* δ *is its utilization parameter, and* L *is the quantum length, then* A-GREEDY *wastes at most* $(1+\rho-\delta)T_1/\delta$ *processor cycles in the course of its computation.*

Proof. We prove the bound using a potential-function argument similar to the one presented in Theorem 6. In this case the potential before quantum q is defined as

$$\Psi(q) = \frac{1+\rho-\delta}{\delta} T_1^q + \frac{\rho}{\rho-1} L d_q \,,$$

where T_1^q is the total number of unexecuted tasks in the computation before quantum q and d_q is the desire for quantum q. By arguments similar to those presented in Theorem 6, one can show that if the job wastes w_q processors in quantum q, then the potential decreases by at least w_q in quantum q. If the computation completes in Q quanta, the total decrease in potential in the course of the computation is

$$\Psi(1) - \Psi(Q+1) = \frac{1+\rho-\delta}{\delta}(T_1 - T_{Q+1})$$

$$+\frac{\rho}{\rho-1}L(d_1 - d_{Q+1})$$
$$\leq \frac{1+\rho-\delta}{\delta}T_1 ,$$

since $d_1 = 1$ and $d_{Q+1} \geq 1$. Therefore, the total waste is at most $(1+\rho-\delta)T_1/\delta$, the total decrease in potential.

□

We can decompose the bounds of Theorems 10 and 11 into separate bounds for accounted and deductible quanta.

COROLLARY 12. *Suppose that* A-GREEDY *schedules a job with work T_1 and critical-path length T_∞ on a machine with P processors, and suppose that ρ is* A-GREEDY*'s responsiveness parameter, δ is its utilization parameter, and L is the quantum length. Let T_a and T_d be the number of time steps in accounted and deductible quanta, respectively, and let W_a and W_d be the waste on accounted and deductible quanta, respectively. Then,* A-GREEDY *achieves the following bounds:*

$$\begin{aligned} T_a &\leq (1/\delta)T_1/\widetilde{P} , \\ T_d &\leq (2\min\{L, 1/(1-\delta)\})T_\infty + L\log_\rho P + L , \\ W_a &\leq (1/\delta - 1)T_1 , \\ W_d &\leq (\rho/\delta)T_1 . \end{aligned}$$

□

As can be seen from these inequalities, the bounds for accounted quanta are stronger than those for deductible quanta. The reason is that the job scheduler in our model is adversarial. In practice, however, it seems unlikely that the job scheduler would actually act as an adversary. Thus, A-GREEDY's behavior on the deductible quanta is likely to be much better than these worst-case bounds predict. Moreover, since the adversary's bad behavior is limited to relatively few deductible quanta, we conjecture that in practice the overall time and waste of a real scheduler based on A-GREEDY more closely follows the bounds for accounted quanta.

5. Adaptive Data-Parallel Scheduling

In this section, we discuss a practical application of A-GREEDY to schedule programs written in data-parallel languages, such as High Performance Fortran (HPF) [26], *Lisp [35], C* [45], NESL [4, 6], and ZPL [16]. Indeed, data-parallel job scheduling algorithms in the literature often model a job as a dag of tasks [3, 6, 24, 33, 51]. Of particular interest is the work by Blelloch and his coauthors [3, 6, 43] which provides various nonadaptive task schedulers for a generalized class of data-parallel jobs, called ***nested*** data-parallel jobs. Specifically, their "prioritized" task schedulers are provably efficient with respect to both time and space. This section applies the desire-estimation strategy of A-GREEDY to data-parallel scheduling. In particular, A-GREEDY can be combined with prioritized task schedulers to produce adaptive task schedulers that are provably efficient with respect to time, space, and waste.

Data-parallel languages present the abstraction of operations on vectors (or matrices), rather than on single scalar values. The total number of vector operations corresponds to the critical-path length of the computation, and the total number of scalar operations corresponds to the work. The time to perform a vector operation on a given number of processors may vary, because vectors may have different lengths from operation to operation. The time may also vary due to vector operations requiring different amounts of work. For example, one vector operation might be an element-wise addition operation, taking time proportional to the length of the vectors, and another vector operation might be an outer-product, taking time proportional to the product of vector lengths.

In an multiprogramming setting, several data-parallel programs might share a single parallel machine, and the job scheduler changes the allotment of processors to various jobs based on their parallelism feedback and its administrative policy. A typical data-parallel task scheduler might map the individual scalar operations to the allotted processors before each vector operation, perhaps using central control. The instantaneous parallelism of the job is simply the work in the next vector operation, which is typically known to the task scheduler, because it knows the vector lengths. If the task scheduler can communicate with the job scheduler before every vector operation, then using instantaneous parallelism as feedback works fine. This strategy may induce high scheduling overheads, however, since it may not be possible to amortize communication with the job scheduler over a single vector operation. Since the task scheduler only knows the parallelism of the next vector operation, not of subsequent ones, if the task scheduler executes multiple vector operations in a single scheduling quantum, an A-GREEDY-like adaptive strategy for parallelism feedback should outperform a strategy based on instantaneous parallelism.

Blelloch, Gibbons, and Matias's prioritized task scheduler [3], called PDF, provides good bounds for data-parallel scheduling. The machine model used in this work is a synchronous P-processor EREW-PRAM [32] augmented with a "scan" primitive [28, 29]. Suppose that a job has T_1 work and a critical-path length of T_∞, and suppose that executing the job in a serial, depth-first fashion uses S_1 space. Then, PDF completes the job in at most $O(T_1/P + T_\infty)$ time steps and requires less than $S_1 + O(PT_\infty)$ space. Blelloch and Greiner [6] extend the PDF algorithms to schedule programs written in the nested data-parallel language NESL with only a small increase in the running time and space.

Combining PDF with A-GREEDY produces an algorithm A-PDF that can schedule data-parallel jobs efficiently in an adaptive setting. A-PDF uses the parallelism feedback mechanisms of A-GREEDY to interact with the job scheduler. At the beginning of each quantum q, A-PDF calculates the desire d_q based on the three-way classification of the previous quantum and reports the desire to the job scheduler according to the algorithm in Figure 1. The job scheduler allots $a_q = \min(p_q, d_q)$ processors to the job. Then, A-PDF uses the prioritized depth-first-like techniques described in [3] for task synchronization and execution on a_q processors in the quantum q. For each time step in quantum q, if there are more than a_q processors in the ready pool, the a_q ready tasks with highest priorities are scheduled. Otherwise, all the ready tasks in the pool are scheduled.

The next theorem — which can be proved in a straightforward fashion by combining our analysis of A-GREEDY with that of [3] — bounds the time, space, and waste of A-PDF.

THEOREM 13. *Suppose that* A-PDF *schedules a job with work T_1 and critical path T_∞ on a P-processor EREW-PRAM augmented with a scan primitive, where L is the quantum size, $\widetilde{P}$ is the $O(T_\infty + L\lg P)$-trimmed availability, and S_1 is the space taken for a serial schedule. Then,* A-PDF *completes the job in T steps, takes S space, and wastes W processor cycles, where*

$$\begin{aligned} T &= O(T_1/\widetilde{P} + T_\infty + L\lg P) \\ S &\leq S_1 + O(PT_\infty) , \\ W &= O(T_1) . \end{aligned}$$

□

Narlikar and Blelloch [43] present an asynchronous algorithm which can be used to schedule data-parallel jobs. Their algorithm obtains the bounds $T = O(T_1/P + T_\infty \lg P)$ and $S = S_1 + O(PT_\infty \lg P)$. A-GREEDY can be combined with this scheduler

as well, producing the following bounds:

$$\begin{aligned} T &= O(T_1/\widetilde{P} + T_\infty \lg P + L \lg P) \,, \\ S &= S_1 + O(PT_\infty \lg P) \,, \\ W &= O(T_1) \,, \end{aligned}$$

where $\widetilde{P}$ is the $O(T_\infty \lg P + L \lg P)$-trimmed availability.

6. Related Work

This section discusses related work on adaptive scheduling of multitasked jobs. Work in this area has generally centered on job schedulers, some of which use "dynamic equipartitioning" as a strategy for allotting processors to jobs. Work on adaptive task scheduling has generally either not used parallelism feedback or been studied only empirically.

Adaptive job schedulers have been studied empirically [36, 37, 39, 42, 53, 55] and theoretically [2, 17, 22, 23, 31, 41]. McCann, Vaswani, and Zahorjan [39] studied many different job schedulers and evaluated them on a set of benchmarks. They also introduced the notion of dynamic equipartitioning, which gives each job a fair allotment of processors, while allowing processors that cannot be used by a job to be reallocated to other jobs. Their studies indicate that dynamic equipartitioning may be an effective strategy for adaptive job scheduling. Gu [31] proved that dynamic equipartitioning with instantaneous parallelism feedback is 4-competitive with respect to makespan for batched jobs with multiple phases, where the parallelism of the job remains constant during the phase and the phases are relatively long compared with the length of a scheduling quantum. Deng and Dymond [17] proved a similar result for mean response time for multiphase jobs regardless of their arrival times. Song [49] proves that a randomized distributed strategy can implement dynamic equipartitioning.

Adaptive task scheduling without parallelism feedback has been studied in the context of multithreading, primarily by Blumofe and his coauthors [1, 11, 13]. In this work, the task scheduler schedules threads using a "work-stealing" [10, 40] strategy, but it does not provide the feedback about the job's parallelism to the job scheduler. The work in [11,13] addresses networks of workstations where processors may fail or join and leave a computation while the job is running, showing that work-stealing provides a good foundation for adaptive task scheduling. In theoretical work, Arora, Blumofe, and Plaxton [1] exhibit a work-stealing task scheduler that provably completes a job in $O(T_1/\overline{P} + PT_\infty/\overline{P})$ expected time, where $\overline{P}$ is the average number of processor allotted to the job by the job scheduler. Although they provide no bounds on waste, one can prove that their algorithm may waste $\Omega(T_1 + PT_\infty)$ processor cycles in an adversarial setting.

Adaptive task scheduling without parallelism feedback has also been studied empirically in the context of data-parallel languages [20, 21]. This work focuses on compiler and runtime support for environments where the number of processors changes while the program executes.

Adaptive task scheduling with parallelism feedback has been studied empirically in [48, 49, 52]. These researchers use a job's history of processor utilization to provide feedback to dynamic-equipartitioning job schedulers. These studies use different strategies for parallelism feedback, and all report better system performance with parallelism feedback than without, but it is not apparent which strategy is superior.

7. Conclusion

We conclude with a brief discussion of trim analysis and some observations that suggest directions for future work on task scheduling with parallelism feedback.

In this paper, we introduced trim analysis as a means of limiting a powerful adversary, which enabling us to analyze an adaptive scheduler with parallelism feedback. The idea of ignoring a few outliers while calculating averages is often used in statistics to ignore anomalous data points. Teachers sometimes ignore the lowest score when computing a student's grade, and in the Olympic Games, the lowest and the highest scores are sometimes ignored when judges average competitors' scores. In theoretical computer science, however, when an adversary is too powerful, researchers have generally tended to resort to making statistical assumptions about the input to render the analysis tractable. Unfortunately, statistical assumptions may not be valid in a particular real-world setting. We are optimistic that trim analysis will find wider application beyond the scheduling problems studied here to situations where statistical methods have heretofore been employed.

A-GREEDY's desire-estimation strategy is robust to incomplete information, which suggests that it should work well in the context of work-stealing schedulers [1, 10, 40]. Work-stealing is a practical and provably efficient method for scheduling dynamic multithreaded computations [8, 12, 27, 34, 44]. Since work-stealing schedulers employ distributed control, the task scheduler has no direct information about the instantaneous parallelism of the job. A-GREEDY does not require perfect, up-to-date information to compute its parallelism feedback, however, and hence it is robust to latency in the gathering of utilization data from processors. We are currently studying how the strategy of providing parallelism feedback based on the history of utilization can be applied to work-stealing schedulers.

Dynamic equipartitioning [17,31,39,55] appears to be an effective way for job schedulers to allocate processors to jobs. If task schedulers provide parallelism feedback using instantaneous parallelism and parallelism rarely changes during a scheduling quantum, a dynamic-equipartitioning job scheduler can optimize global properties like makespan and average completion time [17,31]. For many practical situations, however, a job's parallelism does change quickly and often, making it difficult to obtain perfect information about parallelism. We conjecture that by coupling an A-GREEDY-like task scheduler with a dynamic-equipartitioning job scheduler, provably good global properties can be obtained.

We have analyzed A-GREEDY using an adversarial model for the job scheduler. In practice, however, one would not expect the job scheduler to behave diabolically. Thus, observed bounds on waste and completion time may actually be smaller than the theoretical bounds we have proved. In particular, in this paper we proved that the waste is at most a constant factor of the work. We have begun empirical studies, which although preliminary, seem to indicate that the observed constant is actually quite a bit smaller than the theoretical bound indicates. In this paper, we also proved that A-GREEDY achieves a linear speedup over a trimmed availability, but it seems that it should actually achieve a linear speedup over average availability in practice. The reason is that the availability in deductible steps should rarely be orders of magnitude higher than other steps, because the job scheduler is not a true adversary. We are currently engaged in empirical studies and in implementing a practical scheduler, both of which should shed light how A-GREEDY performs in the real world.

Acknowledgments

We thank Jim Sukha, Bradley Kuszmaul, and Gideon Stupp of MIT CSAIL for their help in designing and analyzing an early variant of A-GREEDY. Siddhartha Sen, also of MIT CSAIL, contributed to an implementation of an adaptive scheduler for Cilk, which inspired this research.

References

[1] Nimar S. Arora, Robert. D. Blumofe, and C. Greg Plaxton. Thread scheduling for multiprogrammed multiprocessors. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 119–129, 1998.

[2] Nikhil Bansal, Kedar Dhamdhere, Jochen Konemann, and Amitabh Sinha. Non-clairvoyant scheduling for minimizing mean slowdown. *Algorithmica*, 40(4):305–318, 2004.

[3] Guy Blelloch, Phil Gibbons, and Yossi Matias. Provably efficient scheduling for languages with fine-grained parallelism. *Journal of the ACM*, 46(2):281–321, 1999.

[4] Guy E. Blelloch, Siddhartha Chatterjee, Jonathan C. Hardwick, Jay Sipelstein, and Marco Zagha. Implementation of a portable nested data-parallel language. *Journal of Parallel and Distributed Computing*, 21(1):4–14, 1994.

[5] Guy E. Blelloch, Phillip B. Gibbons, and Yossi Matias. Provably efficient scheduling for languages with fine-grained parallelism. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 1–12, 1995.

[6] Guy E. Blelloch and John Greiner. A provable time and space efficient implementation of NESL. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming*, pages 213–225, 1996.

[7] Robert D. Blumofe. *Executing multithreaded programs efficiently*. PhD thesis, Massachusetts Institute of Technology, 1995.

[8] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. Cilk: An efficient multithreaded runtime system. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 207–216, 1995.

[9] Robert D. Blumofe and Charles E. Leiserson. Space-efficient scheduling of multithreaded computations. *SIAM Journal on Computing*, 27(1):202–229, February 1998.

[10] Robert D. Blumofe and Charles E. Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM*, 46(5):720–748, 1999.

[11] Robert D. Blumofe and Philip A. Lisiecki. Adaptive and reliable parallel computing on networks of workstations. In *Proceedings of the USENIX Annual Technical Symposion*, pages 133–147, 1997.

[12] Robert D. Blumofe and Dionisios Papadopoulos. Hood: a user-level threads library for multiprogrammed multiprocessors. Technical report, University of Texas at Austin, 1999.

[13] Robert D. Blumofe and David S. Park. Scheduling large-scale parallel computations on networks of workstations. In *Proceedings of the IEEE International Symposium on High-Performance Distributed Computing*, pages 96–105, 1994.

[14] R. P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM*, pages 201–206, 1974.

[15] John L. Bruno, Jr. Edward G. Coffman, and Ravi Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17(7):382–387, 1974.

[16] Bradford L. Chamberlain, Sung-Eun Choi, E Christopher Lewis, Calvin Lin, Lawrence Snyder, and W. Derrick Weathersby. ZPL: a machine independent programming language for parallel computers. *IEEE Transactions on Software Engineering*, 26(3):197–211, 2000.

[17] Xiaotie Deng and Patrick Dymond. On multiprocessor system scheduling. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 82–88, 1996.

[18] Xiaotie Deng, Nian Gu, Tim Brecht, and KaiCheng Lu. Preemptive scheduling of parallel jobs on multiprocessors. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 159–167. Society for Industrial and Applied Mathematics, 1996.

[19] Derek L. Eager, John Zahorjan, and Edward D. Lozowska. Speedup versus efficiency in parallel systems. *IEEE Transactions on Computers*, 38(3):408–423, 1989.

[20] Guy Edjlali, Gagan Agrawal, Alan Sussman, Jim Humphries, and Joel Saltz. Compiler and runtime support for programming in adaptive parallel environments. Technical Report Technical Report CS-TR-3510, University of Maryland, 1995.

[21] Guy Edjlali, Gagan Agrawal, Alan Sussman, and Joel Saltz. Data parallel programming in an adaptive environment. Technical Report Technical Report CS-TR-3350, University of Maryland, 1994.

[22] Jeff Edmonds. Scheduling in the dark. In *Proceedings of the ACM Symposium on the Theory of Computing*, pages 179–188, 1999.

[23] Jeff Edmonds, Donald D. Chinn, Timothy Brecht, and Xiaotie Deng. Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics. *Journal of Scheduling*, 6(3):231–250, 2003.

[24] Zhixi Fang, Peiyi Tang, Pen-Chung Yew, and Chuan-Qi Zhu. Dynamic processor self-scheduling for general parallel nested loops. *IEEE Transactions on Computers*, 39(7):919–929, 1990.

[25] Dror G. Feitelson. Job scheduling in multiprogrammed parallel systems (extended version). Technical report, IBM Research Report RC 19790 (87657) 2nd Revision, 1997.

[26] High Performance Fortran Forum. High Performance Fortran language specification version 1.0. Technical report, Rice University, 1993.

[27] Matteo Frigo, Charles E. Leiserson, and Keith H. Randall. The implementation of the Cilk-5 multithreaded language. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 212–223, 1998.

[28] Joseph Gil and Yossi Matias. Fast and efficient simulations among CRCW PRAMs. *Journal on Parallel and Distribributed Computing*, 23(2):135–148, 1994.

[29] Allan Gottlieb, Boris D. Lubachevsky, and Larry Rudolph. Basic techniques for the efficient coordination of very large numbers of cooperating sequential processors. *Programming Languages and Systems*, 5(2):164–189, 1983.

[30] R. L. Graham. Bounds on multiprocessing anomalies. *SIAM Journal on Applied Mathematics*, pages 17(2):416–429, 1969.

[31] Nian Gu. Competitive analysis of dynamic processor allocation strategies. Master's thesis, York University, 1995.

[32] Tim J. Harris. A survey of PRAM simulation techniques. *ACM Computing Survey*, 26(2):187–206, 1994.

[33] S. F. Hummel and E. Schonberg. Low-overhead scheduling of nested parallelism. *IBM Journal of Research and Development*, 35(5-6):743–765, 1991.

[34] David A. Kranz, Robert H. Halstead, Jr., and Eric Mohr. Mul-T: a high-performance parallel Lisp. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 81–90, 1989.

[35] C. Lasser. *The Essential *Lisp Manual*. Thinking Machines Corporation, 1986.

[36] Scott T. Leutenegger and Mary K. Vernon. The performance of multiprogrammed multiprocessor scheduling policies. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 226–236, 1990.

[37] Shikharesh Majumdar, Derek L. Eager, and Richard B. Bunt. Scheduling in multiprogrammed parallel systems. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 104–113, 1988.

[38] Xavier Martorell, Julita Corbalán, Dimitrios S. Nikolopoulos, Nacho Navarro, Eleftherios D. Polychronopoulos, Theodore S. Papatheodorou, and Jesús Labarta. A tool to schedule parallel applications on multiprocessors: the NANOS CPU manager. In Dror G. Feitelson and Larry Rudolph, editors, *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 87–112, 2000.

[39] Cathy McCann, Raj Vaswani, and John Zahorjan. A dynamic processor allocation policy for multiprogrammed shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 11(2):146–178, 1993.

[40] Eric Mohr, David A. Kranz, and Jr. Robert H. Halstead. Lazy task creation: a technique for increasing the granularity of parallel programs. In *Proceedings of the ACM Conference on Lisp and Functional Programming*, pages 185–197, 1990.

[41] Rajeev Motwani, Steven Phillips, and Eric Torng. Non-clairvoyant scheduling. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 422–431, 1993.

[42] V. K. Naik, M. S. Squillante, and S. K. Setia. Performance analysis of job scheduling policies in parallel supercomputing environments. In *Proceedings of Supercomputing*, pages 824–833, 1993.

[43] Girija J. Narlikar and Guy E. Blelloch. Space-efficient scheduling of nested parallelism. *ACM Transactions on Programming Languages and Systems*, 21(1):138–173, 1999.

[44] Liang Peng, Weng Fai Wong, Ming Dong Feng, and Chung Kwong Yuen. SilkRoad: a multithreaded runtime system with software distributed shared memory for SMP clusters. In *IEEE International Conferrence on Cluster Computing*, pages 243–249, 2000.

[45] J. R. Rose and G. L. Steele Jr. C*: An extended C language for data parallel programming. In *Proceedings of Supercomputing*, pages 2–16, 1987.

[46] Emilia Rosti, Evgenia Smirni, Lawrence W. Dowdy, Giuseppe Serazzi, and Brian M. Carlson. Robust partitioning schemes of multiprocessor systems. *Performance Evaluation*, 19(2-3):141–165, 1994.

[47] Emilia Rosti, Evgenia Smirni, Giuseppe Serazzi, and Lawrence W. Dowdy. Analysis of non-work-conserving processor partitioning policies. In *Proceedings of the International Parallel Processing Symposium*, pages 165–181, 1995.

[48] Siddhartha Sen. Dynamic processor allocation for adaptively parallel jobs. Master's thesis, Massachusetts Institute of Technology, 2004.

[49] B. Song. Scheduling adaptively parallel jobs. Master's thesis, Massachusetts Institute of Technology, 1998.

[50] Mark S. Squillante. On the benefits and limitations of dynamic partitioning in parallel computer systems. In *Proceedings of the International Parallel Processing Symposium*, pages 219–238, 1995.

[51] Dan Suciu and Val Tannen. Efficient compilation of high-level data parallel algorithms. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 57–66, 1994.

[52] Kaushik Guha Timothy B. Brecht. Using parallel program characteristics in dynamic processor allocation policies. *Performance Evaluation*, 27-28:519–539, 1996.

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

[54] John Turek, Walter Ludwig, Joel L. Wolf, Lisa Fleischer, Prasoon Tiwari, Jason Glasgow, Uwe Schwiegelshohn, and Philip S. Yu. Scheduling parallelizable tasks to minimize average response time. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 200–209, 1994.

[55] K. K. Yue and D. J. Lilja. Implementing a dynamic processor allocation policy for multiprogrammed parallel applications in the solaris (tm) operating system. *Concurrency and Computation-Practice and Experience*, 13(6):449–464, 2001.