Competitive Online Scheduling of Perfectly Malleable Jobs with Setup Times

Jessen T. Havill[*] and Weizhen Mao[†]

Abstract

We study how to efficiently schedule online perfectly malleable parallel jobs with arbitrary arrival times on $m \geq 2$ processors. We take into account both the linear speedup of such jobs and their setup time, i.e., the time to create, dispatch, and destroy multiple processes. Specifically, we define the execution time of a job with length p_j running on k_j processors to be $p_j/k_j + (k_j - 1)c$, where $c > 0$ is a constant setup time associated with each processor that is used to parallelize the computation. This formulation accurately models data parallelism in scientific computations and realistically asserts a relationship between job length and the maximum useful degree of parallelism. When the goal is to minimize makespan, we show that the online algorithm that simply assigns k_j so that the execution time of each job is minimized and starts jobs as early as possible has competitive ratio $4(m-1)/m$ for even $m \geq 2$ and $4m/(m+1)$ for odd $m \geq 3$. This algorithm is much simpler than previous offline algorithms for scheduling malleable jobs that require more than a constant number of passes through the job list.

Keywords: scheduling, parallel jobs, setup times, online algorithms.

1 Introduction

Malleable parallel jobs can distribute their workload among any number of available processors in a parallel computer in order to decrease their execution time. In contrast, *nonmalleable* parallel jobs must use a fixed number of processors. The ideal execution time of a malleable parallel job with length p is p/k if it utilizes k processors. However, inherently serial code and parallel processing overhead (from process management, shared memory access and contention, communication, and/or synchronization) often prevent actual execution times from achieving this ideal. It is natural to consider this extra time as a type of *setup time*, a term commonly used by the scheduling community [1, 2]. We will derive an execution time function that takes both speedup (i.e., p/k) and setup time into account for a particular class of malleable jobs, and study how to efficiently schedule these jobs online.

Formally, we have available $m \geq 2$ identical processors and we are given n parallel jobs $J_1, J_2, \ldots, J_n$ to schedule, each with an *a priori* length $p_j > 0$ and arrival time $a_j \geq 0$. (Assume that the jobs are indexed by nondecreasing arrival times, i.e., for $1 \leq i < j \leq n$, $a_i \leq a_j$.) For each job J_j, a scheduling algorithm must assign both $k_j \in \{1, 2, \ldots, m\}$ processors to execute the job and a start time $s_j \geq a_j$ that does not conflict with previously scheduled jobs. A job must be executed simultaneously on all assigned processors and may not be preempted. The execution time of the job, t_j, depends both on the job and on k_j, and will be defined shortly. We will denote the work done by job J_j as $w_j = k_j t_j$. (This concept will be used extensively in our proofs later.) We wish to minimize the makespan of the schedule, defined to be $C = \max_j C_j$, where $C_j = s_j + t_j$ is the completion time of job J_j. We choose the makespan in this study since it measures the utilization of resources and is most often used in past research. For instances with arbitrary arrivals, other optimization criteria such as $\max_j(C_j - a_j)$ and $\sum_j(C_j - a_j)$ are also important topics of ongoing research.

[*]Dept. of Mathematics and Computer Science, Denison University, Granville, OH 43023, USA, havill@denison.edu, (740) 587-6582 (phone), (740) 587-5749 (fax)

[†]Dept. of Computer Science, The College of William and Mary, Williamsburg, VA 23187–8795, USA, wm@cs.wm.edu, (757) 221-3472 (phone), (757) 221-1717 (fax)

The problem of scheduling malleable jobs is strongly NP-hard [6]. Several papers [3, 22, 18] have addressed approximation algorithms for the special *monotonic* case in which the execution time function monotonically decreases as the number of processors increase. Other papers [16, 4, 14] have presented approximation results for the case without this restriction. All of these approximation algorithms require $\omega(1)$[1] passes through the job list to guarantee their performance bounds and are relatively complex. We are thus motivated to study a realistic special case of the malleable job scheduling problem for which a simple *online* scheduling algorithm works well. In particular, we study the problem of scheduling jobs that are *perfectly malleable*, that is, the work of the job can be divided evenly among available processors, resulting in an ideal linear speedup. However, executing such a job on multiple processors requires the operating system to create, dispatch, and eventually destroy multiple processes, so we must take this setup time into account as well. Therefore, we define the *execution time* of job J_j to be

$$t_j = p_j/k_j + (k_j - 1)c \tag{1}$$

where $c > 0$ is the setup time required to manage a single process. We use $(k_j - 1)c$ instead of $k_j c$ so that $k_j = 1 \Rightarrow t_j = p_j$ to remain consistent with classical (non-parallel) job scheduling. Notice that if we do not take into account setup times ($c = 0$), the problem is solved optimally by simply assigning every job to m processors. Other papers have studied the problem of scheduling perfectly malleable jobs without setup times in the presence of dependencies [23] and unknown processing times [21].

In general, the setup time can be any reasonable function (e.g., logarithmic, linear, quadratic) of k_j, or even just a constant additive term [5], but we select a linear function and constant c to reflect the typical implementation of jobs exhibiting data parallelism in shared memory architectures [15]. Data parallelism techniques are very common and can be applied to sorting and searching, matrix multiplication, and other vector and polynomial computations. Such programs are typically composed of forall loops like the following:

```
forall $i \leftarrow 1$ to $N$ do
    $A[i] \leftarrow f(A[i])$;
```

In this case, $p_j = Nt$, where t is the time required for each iteration of the loop. At runtime, the system decides to use k_j processors for the computation and assigns each processor to execute N/k_j iterations of the loop. Typically, the "master" process created when the main program begins is counted as one of the k_j processes, so only $k_j - 1$ new child processes are created (by the master process) beyond what would be required on a single processor. So the running time of the job will be approximately $t_j = p_j/k_j + (k_j - 1)c$. (The assignment of work to processes could be accomplished in $O(\log k_j)$ time, but the creation of processes is still likely to be sequential.) We also point out that p_j is relatively easy to approximate in advance in these kinds of computations, since one need only approximate the time required by the function f.

The same execution time function (1) can represent more general distributed computations which utilize a constant number of message passing "rounds". In each round of message passing, a process sends a message to at most $k_j - 1$ other processes. Then p_j/k_j represents the parallel computation time and $(k_j - 1)c$ represents the time needed to complete c/t rounds of message passing, where t is the time required to send a message. In more general cases where the number of message passing rounds varies greatly among jobs, a better model would have $t_j = p_j/k_j + (k_j - 1)c_j$, where c_j/t is the number of message passing rounds required for job J_j. This latter formula is consistent with one proposed by Sevcik ([20], p. 120).

[1] $\omega(g(n)) = \{f(n) : \forall \text{ constant } c > 0,\ \exists \text{ constant } n_0 > 0 \text{ such that } 0 \le cg(n) < f(n)\ \forall\ n \ge n_0\}$.

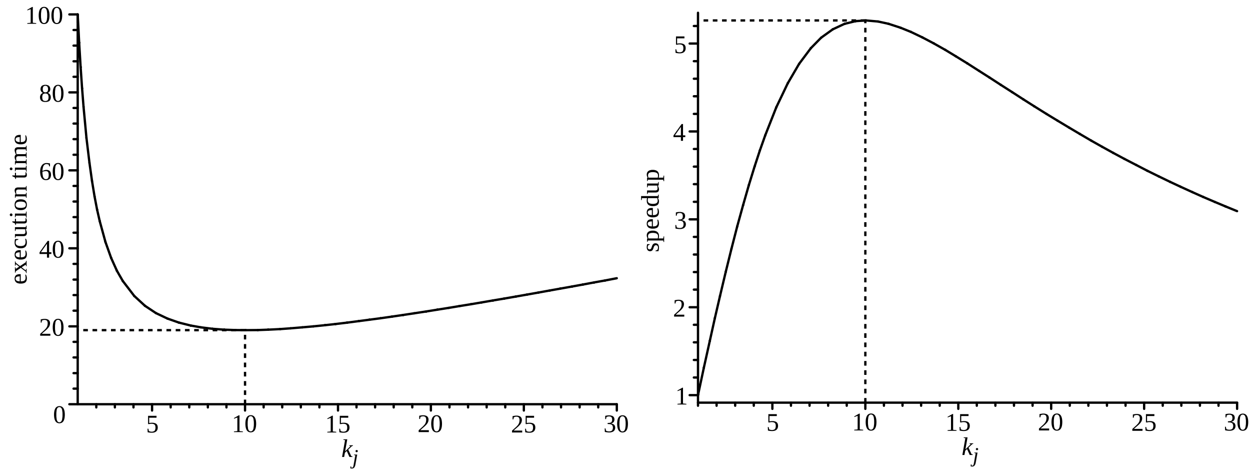

Figure 1: Execution time (t_j) and speedup (p_j/t_j) of a job with $p_j = 100$ as a function of k_j ($c = 1$). Execution time is minimized, and hence speedup is maximized, when $k_j = \sqrt{p_j/c} = 10$.

Our execution time formula (1) is plotted in Figure 1. Notice that the execution time quickly decreases until it reaches a minimum at $k_j = \sqrt{p_j/c}$, and then slowly increases. (Alternatively, the speedup, defined to be p_j/t_j, is maximized at the same point.) Therefore, every job has a maximum useful degree of parallelism that depends on its length, which differentiates this model from the monotonic model discussed earlier. Put another way, this problem formulation makes the realistic assumption that the execution time of smaller jobs is likely to be dominated by the setup time on a smaller number of processors than larger jobs.

We study a simple online algorithm called *Shortest Execution Time* (or SET). For each job J_j, the algorithm computes k_j so that it minimizes the execution time function $T_j(k) = p_j/k + (k-1)c$ and then schedules the job on k_j processors as early as possible. Notice that, by definition, the following must be true:

Fact 1 *The online algorithm SET assigns* $k_j \in \{1, 2, \ldots, m-1\}$ *processors to jobs with length* $p_j \in (k_j(k_j-1)c,\, k_j(k_j+1)c]$ *and* $k_j = m$ *processors to jobs with length* $p_j > m(m-1)c$.

Proof This follows directly since the choice of k_j minimizes the execution time of job J_j:

$$\frac{p_j}{k_j} + (k_j - 1)c \le \frac{p_j}{k_j+1} + k_j c \;\Rightarrow\; p_j \le k_j(k_j+1)c$$

and

$$\frac{p_j}{k_j} + (k_j - 1)c < \frac{p_j}{k_j-1} + (k_j-2)c \;\Rightarrow\; p_j > k_j(k_j-1)c\,.$$

□

The details of the algorithm are described in Figure 2. First, in lines 4–7, SET computes the value of k_j that minimizes the job's execution time. Next, in lines 8–9, SET sets s_j to be the earliest possible starting time on k_j processors. Finally, in lines 10–13, SET schedules the job on the k_j most loaded processors that are available at time s_j. (Note that line 11 is not required for the analysis that follows; it is simply the most logical choice.)

Recall that past algorithms require $\omega(1)$ passes through the job list. However, SET is online in the classical list scheduling [11] sense, i.e., it chooses, for each job J_j in a list of jobs, a value of k_j and a start time $s_j \ge a_j$ before jobs $J_{j+1}, J_{j+2}, \ldots, J_n$ are known. Simple algorithms like this have been identified as having the most promise in real systems [8]. In addition, SET is more suitable

```
1.  for each processor $i \leftarrow 1, 2, \ldots, m$ do
2.      $\text{load}_i \leftarrow 0$
3.  for each job $j \leftarrow 1, 2, \ldots, n$ do
4.      $k \leftarrow \min\left\{m, \sqrt{p_j/c}\right\}$
5.      if $k = m$ then $k_j \leftarrow m$
6.      else if $p_j/\lfloor k \rfloor + (\lfloor k \rfloor - 1)c \le p_j/\lceil k \rceil + (\lceil k \rceil - 1)c$ then $k_j \leftarrow \lfloor k \rfloor$
7.      else $k_j \leftarrow \lceil k \rceil$
8.      $s'_j \leftarrow \text{load}_i$, where $i$ is the processor with the $k_j$-th smallest load
9.      $s_j \leftarrow \max\{s'_j, a_j\}$
10.     $L \leftarrow \{i : i \in \{1, 2, \ldots, m\}, \text{load}_i \le s_j\}$
11.     $L' \leftarrow \{\text{the } k_j \text{ most loaded processors in } L\}$
12.     schedule $j$ at time $s_j$ on the processors in $L'$
13.     for each $i \in L'$ do $\text{load}_i \leftarrow s_j + p_j/k_j + (k_j - 1)c$
```

Figure 2: The detailed SET algorithm.

m	2	3	4	5	6	7	8	16	32	64	128	1024
$\mathcal{R}$[SET]	2	3	3	$3.\bar{3}$	$3.\bar{3}$	3.5	3.5	3.75	3.875	3.938	3.969	3.996

Table 1: Approximate values of $\mathcal{R}$[SET] for representative values of m.

for practical situations since jobs in modern systems are usually submitted over time, rather than in a batch.

We say that an online algorithm A is *r-competitive* if, for all instances I, $C(I) \le r \cdot C^*(I) + b$, where $C(I)$ and $C^*(I)$ are the makespan of the schedule constructed by A for instance I and the optimal makespan for instance I, respectively, and b is constant with respect to the job sequence (but may depend on m, which is constant for a given parallel system). The competitive ratio[2] $\mathcal{R}[A]$ of A is the infimum over all such values r.

After discussing related research in more detail in Section 2, we show in Sections 3–5 that the competitive ratio of SET, $\mathcal{R}$[SET], is $4(m-1)/m$ for even $m \ge 2$ and $4m/(m+1)$ for odd $m \ge 3$. These values range from 2 for $m = 2$ to 4 for arbitrarily large m, as shown in Table 1. In Section 6, we prove some lower bounds on the competitive ratio of any online algorithm. In Section 7, we conclude and discuss topics for future research.

2 Related Research

Du and Leung [6] showed that scheduling malleable jobs is strongly NP-hard. Several authors have studied offline approximation algorithms for related malleable job scheduling problems. Belkhale and Banerjee [3] presented an algorithm that has approximation ratio $2m/(m+1)$ if the execution time of the jobs is defined to decrease monotonically with the number of processors and the work is nondecreasing with respect to the number of processors (note that neither are true in our model). Turek, Wolf, and Yu [22] showed how to transform a nonmalleable job scheduling algorithm into an algorithm for scheduling malleable jobs with monotonically decreasing execution time functions that has the same approximation ratio and a multiplicative $O(mn)$ increase in time complexity.

[2]We follow convention by using *competitive ratio* for online (approximation) algorithms and *approximation ratio* for (offline) approximation algorithms, although the terms have similar definitions.

This technique yields an algorithm with approximation ratio 2 based on the nonmalleable scheduling algorithm of Garey and Graham [11]. Later, Mounie, Rapine, and Trystram [18] presented an algorithm for the monotonic problem with approximation ratio $\sqrt{3}$. Ludwig and Tiwari [16] improved upon the result of Turek, Wolf, and Yu [22] by showing how to accomplish the same transformation with only an additive $O(mn)$ increase in time complexity, and without any assumptions about the execution time or work functions. Blazewicz, et al. [4] offered another algorithm for the same problem with the same approximation ratio. Jansen and Porkolab [14] proposed an approximation scheme for the general problem with time complexity that is linear in n but exponential in m. Jansen [13] later presented an asymptotic fully polynomial time approximation scheme.

Previous papers by the authors studied the model in which $t_j = p_j/k_j + k_j c$ and all $a_j = 0$. Mao, Chen, and Watson [17] showed that the competitive ratio of SET is 2 when $m = 2$ and presented simulation results showing that the algorithm performs well on uniformly random job sequences for a number of different values of m and c. Havill [12] showed that an algorithm very similar to SET has competitive ratio at most 4 for any m. This paper improves upon that result by proving a tight bound for all m rather than just arbitrarily large m.

Wang and Cheng [23] studied the related problem with a linear execution time function, job dependencies, and a maximum degree of parallelism for each job. They showed that the competitive ratio of the online Earliest Completion Time algorithm is between $5/2$ and $3 - 2/m$.

Others have studied the problem with a linear execution time function and maximum degrees of parallelism, but where the lengths of the jobs are unknown until they complete. (See Sgall [21] for a survey.) Notably, Feldmann, Sgall, and Teng [10] showed that the algorithm that simply assigns the maximum allowable number of processors to a job when that many processors are available has competitive ratio 2. Feldmann, Kao, Sgall, and Teng [9] and Rapine, Scherson, and Trystram [19] studied related problems with dependencies and preemption, respectively.

Setup time has also been extensively studied in the context of sequential job scheduling [1, 2]. In these problems, setup time is typically related to job (or batch) specific preparation that is necessary for the execution of a job. In our problem, setup time is only related to overhead arising from parallel processing.

As discussed previously, in contrast to these results, we are interested in online algorithms for a problem with a particular nonlinear execution time function.

3 Lower Bounds for SET

In this section, we give lower bounds on the competitive ratio of SET. We consider the cases where m is even and m is odd separately.

Theorem 3.1 *$\mathcal{R}[SET] \geq 4(m-1)/m$ when $m \geq 2$ is even.*

Proof Consider an instance consisting of $n = 2\alpha m$ jobs, where α is a positive integer. We define $p_j = (m/2)(m/2-1)c + \epsilon$, for odd j and $p_j = (m/2)(m/2+1)c + \epsilon$ for even j, where $\epsilon > 0$ is arbitrarily small. For all j, $a_j = 0$. SET will assign each of the odd-indexed jobs to $m/2$ processors and each of the even-indexed jobs to $m/2+1$ processors. Clearly, the algorithm will schedule the jobs in sequential order. Therefore, the makespan of the schedule on this sequence, as $\epsilon \to 0$, is

$$\begin{aligned} C &= \sum_j \left(\frac{p_j}{k_j} + (k_j - 1)c\right) \\ &= \frac{n}{2}\left(\left(\frac{(m/2)(m/2-1)c}{m/2} + \left(\frac{m}{2} - 1\right)c\right) + \left(\frac{(m/2)(m/2+1)c}{m/2+1} + \frac{mc}{2}\right)\right) \end{aligned}$$

$$= n(m-1)c\ .$$

On the other hand, an offline algorithm can construct a better schedule by assigning each job to one processor and executing all the odd-indexed jobs followed by the even-indexed jobs. In this case, the optimal makespan, as $\epsilon \to 0$, is

$$C^* \le \frac{1}{m}\sum_j p_j \le \frac{n}{2m}\left(\left(\frac{m}{2}\right)\left(\frac{m}{2}-1\right)c + \left(\frac{m}{2}\right)\left(\frac{m}{2}+1\right)c\right) = \frac{nmc}{4}\ .$$

Thus, when m is even, the competitive ratio of SET is at least $4(m-1)/m$. □

Theorem 3.2 *$\mathcal{R}[SET] \ge 4m/(m+1)$ when $m \ge 3$ is odd.*

Proof Consider an instance consisting of $n = \alpha m$ jobs, each with length $((m+1)/2)((m-1)/2)c + \epsilon$, for arbitrarily small $\epsilon > 0$, where α is a positive integer. For all j, $a_j = 0$. SET will assign each of these jobs to $(m+1)/2$ processors. Clearly, the algorithm will schedule the jobs in sequential order. Therefore, the makespan of the schedule on this sequence, as $\epsilon \to 0$, is

$$C = \sum_j \left(\frac{p_j}{k_j} + (k_j - 1)c\right) = \sum_j \left(\frac{((m+1)/2)((m-1)/2)c}{(m+1)/2} + \frac{(m-1)c}{2}\right) = n(m-1)c\ .$$

On the other hand, an offline algorithm can construct a better schedule by assigning each job to one processor. In this case, the optimal makespan, as $\epsilon \to 0$, is

$$C^* \le \frac{1}{m}\sum_j p_j = \frac{1}{m}\sum_j \left(\frac{m+1}{2}\right)\left(\frac{m-1}{2}\right)c = \frac{n(m^2-1)c}{4m}\ .$$

Thus, when m is odd, the competitive ratio of SET is at least $4m/(m+1)$. □

4 Upper Bounds for SET When Arrival Times Are 0

In this section, we prove matching upper bounds for the case in which all jobs arrive at time 0 but are scheduled in the given order. In the next section, we show how the proof can be modified to handle arbitrary online arrivals.

We partition a SET schedule into K *blocks*. Each block B_i, $i = 0, 1, \ldots, K-1$, is defined to be a maximal time interval $[b_i, b_{i+1})$ such that *neither* of the following events occur in the open time interval (b_i, b_{i+1}), for any job J_j:

1. $s_j \in (b_i, b_{i+1})$ (a job begins execution)
2. $C_j \in (b_i, b_{i+1})$ and $k_j \ge \lfloor (m+3)/2 \rfloor$ (a job executing on $\lfloor (m+3)/2 \rfloor$ or more processors ends)

In other words, each block B_i starts when one or more jobs begin execution or a job assigned to at least $\lfloor (m+3)/2 \rfloor$ processors ends execution (at time b_i), and ends when one of these two events happens next (at time b_{i+1}) or the schedule ends. The first block has $b_0 = 0$ and the last block has $b_K = C$. We visualize a schedule as a $C \times m$ rectangle with the horizontal dimension representing time and the vertical dimension representing processors. A job is then a rectangle (perhaps divided into strips) with total area $t_j \times k_j$ and a block B_i is a $(b_{i+1} - b_i) \times m$ rectangle.

We will classify the blocks in a schedule into three different types. By inspection, it will be easy to see that the three types cover all possible blocks. The first type of block, which we call a Type I block, is one in which at least $\lceil (m+1)/2 \rceil$ processors are busy at all times during the time

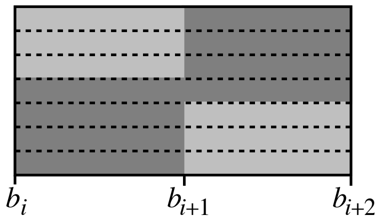

Figure 3: One possibility for two consecutive Type I blocks ($m = 7$). The darker gray represents the area during which processors are busy in each of the two blocks. The lighter gray represents the area where other jobs could also be executing.

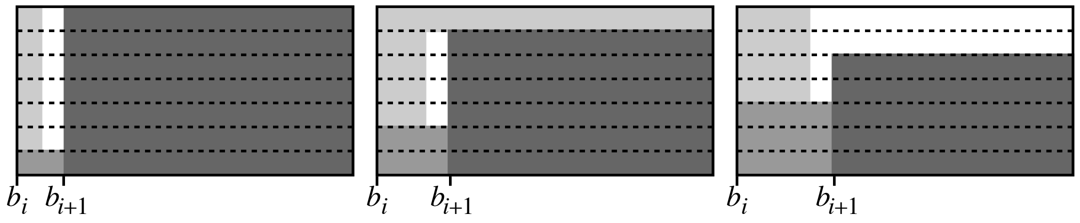

Figure 4: Possible Type IIa blocks ($m = 7$). The medium gray represents the work on the d processors defined in the text. The dark gray represents job J_l. The light gray represents the area where other jobs could also be executing. If a Type IIb block exists, it looks like this as well, but lacks the following large job J_l.

interval $[b_i, b_{i+1})$. See Figure 3 for a graphical depiction of consecutive Type I blocks. The second type of block, which we call a Type II block, is one in which fewer than $\lceil (m+1)/2 \rceil$ (or at most $\lceil (m-1)/2 \rceil$) processors are busy at all times during the interval $[b_i, b_{i+1})$. Note that, since no jobs start in the interval (b_i, b_{i+1}), if there are $d \leq \lceil (m-1)/2 \rceil$ processors busy at all times during the interval, there must be $m - d \geq \lfloor (m+1)/2 \rfloor$ processors idle at the end of the interval. We further classify Type II blocks into two subtypes. A Type II block B_i is of the first subtype, which we call Type IIa, when it is not the last in the schedule (i.e., when $b_{i+1} < C$). A Type IIa block must be followed by a job J_l running on $k_l \geq m - d + 1 \geq \lfloor (m+3)/2 \rfloor$ processors. The job J_l is contained within one or more consecutive Type I blocks. See Figure 4 for a graphical depiction of possible Type IIa blocks. The second subtype of Type II, which we call Type IIb, is a Type II block B_i that is the last block in the schedule (i.e., $b_{i+1} = b_K = C$). Clearly, there is at most one Type IIb block in any schedule.

For each block B_i in the schedule, we define its work area (or simply work) W_i and its idle area I_i. The work area is the total area in the block in which jobs are executing. The idle area is the complement: the total area in which processors are idle. Therefore, for a particular block B_i, $W_i + I_i = m \times (b_{i+1} - b_i)$. As mentioned earlier, the work performed by a particular job J_j in the SET schedule is denoted

$$w_j = k_j t_j = p_j + k_j(k_j - 1)c \, . \tag{2}$$

Similarly in an optimal schedule, if k_j^* is the number of processors assigned to J_j, the work performed by job J_j is denoted

$$w_j^* = p_j + k_j^*(k_j^* - 1)c \geq p_j \, .$$

In addition, we let

$$W = \sum_j w_j = \sum_j (p_j + k_j(k_j - 1)c)$$

and

$$W^* = \sum_j w_j^* = \sum_j (p_j + k_j^*(k_j^* - 1)c) \geq \sum_j p_j$$

be the total work in the SET and optimal schedules, respectively. Also, we let I and I^* be the total idle areas in the SET and optimal schedules, respectively. Then the makespan of the SET schedule is $C = (W + I)/m$ and the optimal makespan is $C^* = (W^* + I^*)/m$. Using this notation, we note a simple lower bound on C^*:

Lemma 4.1 $C^* \geq W^*/m \geq \sum_j p_j/m$.

Proof Since $I^* \geq 0$ and by the definition of W^*, $C^* = (W^* + I^*)/m \geq W^*/m \geq \sum_j p_j/m$. □

4.1 Upper Bound for Odd $m \geq 3$

We will explicitly prove the upper bound for odd $m \geq 3$. In the next subsection, we will explain how to modify the proof for even $m \geq 2$.

Theorem 4.1 *$\mathcal{R}[SET] \leq 4m/(m + 1)$ for odd $m \geq 3$ when all jobs arrive at time 0.*

Proof We claim that the result follows if

$$I \leq \frac{m-1}{m+1} W + \frac{2m}{m+1} \sum_j (p_j - k_j(k_j - 1)c) + mb \;, \tag{3}$$

where b is a term that is independent of the job sequence and the schedule constructed, but may be a function of m. This claim is true because

$$\begin{aligned}
C &= \frac{1}{m}(W + I) \\
&\leq \frac{2}{m+1} W + \frac{2}{m+1} \sum_j (p_j - k_j(k_j - 1)c) + b && \text{(by (3))} \\
&= \frac{2}{m+1} \sum_j (p_j + k_j(k_j - 1)c) + \frac{2}{m+1} \sum_j (p_j - k_j(k_j - 1)c) + b && \text{(by (2))} \\
&= \frac{4}{m+1} \sum_j p_j + b \\
&\leq \left(\frac{4m}{m+1}\right) C^* + b && \text{(by Lemma 4.1)} \;.
\end{aligned}$$

For block B_i, let F_i denote the set of indices of jobs executing during (b_i, b_{i+1}). For each $j \in F_i$, let α_{ji}, where $0 < \alpha_{ji} \leq 1$, denote the fraction of the work w_j of job J_j executed in block B_i. If $j \notin F_i$, then $a_{ji} = 0$. For any j, $\sum_i \alpha_{ji} = 1$. For an arbitrary schedule constructed by SET, we will prove the following version of inequality (3) for sets of blocks $\{B_i, B_{i+1}, \ldots, B_q\}$, where $q \geq i$:

$$\sum_{r=i}^{q} I_r \leq \sum_{r=i}^{q} \left(\frac{m-1}{m+1} W_r + \frac{2m}{m+1} \sum_{j \in F_r} \alpha_{jr}(p_j - k_j(k_j - 1)c) \right) + mb \;,$$

where the additive mb term can only appear in the inequality for the Type IIb block (if it exists). Since the sets we consider will constitute a partition of the schedule, we will then be able to sum these values to arrive at (3).

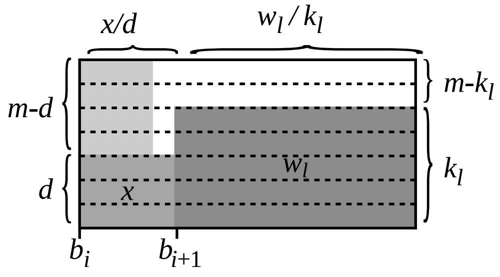

Figure 5: Variables in the analysis of a Type IIa block, applied to the third block in Figure 4.

We first consider Type IIa blocks B_i. Recall that a Type IIa block must be followed by a job J_l with $s_l = b_{i+1}$ and $k_l \geq (m+3)/2$. We will analyze a Type IIa block together with the consecutive Type I blocks $B_{i+1}, \ldots, B_q$ that together contain job J_l. Also recall that d processors must be busy at all times in the interval $[b_i, b_{i+1})$, where $m - k_l + 1 \leq d \leq (m-1)/2$. Let $x = d \times (b_{i+1} - b_i)$ denote the work performed on these d processors in the interval (b_i, b_{i+1}) and let F_i' denote the set of indices of jobs executing on these d processors in the interval (b_i, b_{i+1}). Notice that $F_i' \subseteq F_i$ and, since no job starts in (b_i, b_{i+1}), $\sum_{j \in F_i'} k_j = d$. Now we consider the relationship between $\sum_{r=i}^q I_r$ and $\sum_{r=i}^q W_r$.

$$\begin{aligned}
\sum_{r=i}^{q} I_r &\leq \frac{m-d}{d} x + \frac{m-k_l}{k_l} w_l \qquad \text{(See Figure 5.)} \\
&= \frac{m-1}{m+1}(x + w_l) + \left(\frac{m-d}{d} - \frac{m-1}{m+1}\right) x + \left(\frac{m-k_l}{k_l} - \frac{m-1}{m+1}\right) w_l \\
&\leq \frac{m-1}{m+1}\left(W_i + \sum_{r=i+1}^{q} W_r\right) + \left(\frac{m}{d} - \frac{2m}{m+1}\right) \sum_{j \in F_i'} \alpha_{ji} \left(p_j + k_j(k_j-1)c\right) \\
&\quad + \left(\frac{m}{k_l} - \frac{2m}{m+1}\right)\left(p_l + k_l(k_l-1)c\right) \qquad \text{(by (2))} \\
&= \frac{m-1}{m+1} \sum_{r=i}^{q} W_r + \frac{2m}{m+1}\left(\sum_{j \in F_i'} \alpha_{ji}(p_j - k_j(k_j-1)c) + (p_l - k_l(k_l-1)c)\right) \\
&\quad + \left(\frac{m}{d} - \frac{4m}{m+1}\right) \sum_{j \in F_i'} \alpha_{ji} p_j + \frac{m}{d} \sum_{j \in F_i'} \alpha_{ji} k_j (k_j - 1)c + \left(\frac{m}{k_l} - \frac{4m}{m+1}\right) p_l + m(k_l - 1)c \\
&\leq \sum_{r=i}^{q}\left(\frac{m-1}{m+1} W_r + \frac{2m}{m+1} \sum_{j \in F_r} \alpha_{jr}(p_j - k_j(k_j-1)c)\right) + \left(\frac{m}{d} - \frac{4m}{m+1}\right) \sum_{j \in F_i'} \alpha_{ji} p_j \\
&\quad + \frac{m}{d} \sum_{j \in F_i'} \alpha_{ji} k_j (k_j-1)c + \left(\frac{m}{k_l} - \frac{4m}{m+1}\right) p_l + m(k_l-1)c\,. \qquad (4)
\end{aligned}$$

We can bound the last two terms of (4) as follows:

$$\begin{aligned}
&\left(\frac{m}{k_l} - \frac{4m}{m+1}\right) p_l + m(k_l-1)c \\
&\qquad\qquad < \left(\frac{m}{k_l} - \frac{4m}{m+1}\right) k_l(k_l-1)c + m(k_l-1)c
\end{aligned}$$

$$
\begin{aligned}
&\quad \text{(since } m/k_l - 4m/(m+1) < 0 \text{ and by Fact 1)} \\
&= \frac{-2m(k_l-1)(2k_l-m-1)c}{m+1} \\
&\le \frac{-2m(m-d)(m-2d+1)c}{m+1} \quad \text{(since } k_l \ge m-d+1 > (m+1)/2\text{)} . \qquad (5)
\end{aligned}
$$

To bound the second and third terms of (4), we consider two cases. (Lemmas A.1–A.4 can be found in Appendix A.)

Case 1: $1 \le d \le (m+1)/4$. In this case, $m/d - 4m/(m+1) \ge 0$. Therefore,

$$
\begin{aligned}
&\left(\frac{m}{d} - \frac{4m}{m+1}\right) \sum_{j \in F_i'} \alpha_{ji} p_j + \frac{m}{d} \sum_{j \in F_i'} \alpha_{ji} k_j (k_j - 1) c \\
&\le \left(\frac{m}{d} - \frac{4m}{m+1}\right) \sum_{j \in F_i'} p_j + \frac{m}{d} \sum_{j \in F_i'} k_j (k_j - 1) c \\
&\le \left(\frac{m}{d} - \frac{4m}{m+1}\right) \sum_{j \in F_i'} k_j (k_j + 1) c + \frac{m}{d} \sum_{j \in F_i'} k_j (k_j - 1) c \quad \text{(by Fact 1)} \\
&\le \left(\frac{m}{d} - \frac{4m}{m+1}\right) d(d+1)c + \frac{m}{d} d(d-1)c \quad \text{(by Lemma A.1)} \\
&= \frac{2md(m-2d-1)c}{m+1} . \qquad (6)
\end{aligned}
$$

Substituting (5) and (6) into (4), we conclude that

$$
\begin{aligned}
\sum_{r=i}^{q} I_r &\le \sum_{r=i}^{q} \left(\frac{m-1}{m+1} W_r + \frac{2m}{m+1} \sum_{j \in F_r} \alpha_{jr} (p_j - k_j(k_j-1)c) \right) - \frac{2m(m^2+m-4md+4d^2)c}{m+1} \\
&< \sum_{r=i}^{q} \left(\frac{m-1}{m+1} W_r + \frac{2m}{m+1} \sum_{j \in F_r} \alpha_{jr} (p_j - k_j(k_j-1)c) \right) \quad \text{(by Lemma A.2)} . \qquad (7)
\end{aligned}
$$

Case 2: $(m+1)/4 < d \le (m-1)/2$. In this case, $m/d - 4m/(m+1) < 0$. Therefore,

$$
\begin{aligned}
&\left(\frac{m}{d} - \frac{4m}{m+1}\right) \sum_{j \in F_i'} \alpha_{ji} p_j + \frac{m}{d} \sum_{j \in F_i'} \alpha_{ji} k_j (k_j - 1) c \\
&< \left(\frac{2m}{d} - \frac{4m}{m+1}\right) \sum_{j \in F_i'} \alpha_{ji} k_j (k_j - 1) c \quad \text{(by Fact 1)} \\
&\le \left(\frac{2m}{d} - \frac{4m}{m+1}\right) \sum_{j \in F_i'} k_j (k_j - 1) c \quad \text{(since } 2m/d - 4m/(m+1) \ge 0\text{)} \\
&\le \left(\frac{2m}{d} - \frac{4m}{m+1}\right) d(d-1)c \quad \text{(since } 2m/d - 4m/(m+1) \ge 0 \text{ and by Lemma A.1)} \\
&= \frac{2m(d-1)(m-2d+1)c}{m+1} . \qquad (8)
\end{aligned}
$$

Substituting (5) and (8) into (4), we conclude that

$$
\sum_{r=i}^{q} I_r \le \sum_{r=i}^{q} \left(\frac{m-1}{m+1} W_r + \frac{2m}{m+1} \sum_{j \in F_r} \alpha_{jr} (p_j - k_j(k_j-1)c) \right) - \frac{2m(m-2d+1)^2 c}{m+1}
$$

$$\leq \sum_{r=i}^{q} \left(\frac{m-1}{m+1} W_r + \frac{2m}{m+1} \sum_{j \in F_r} \alpha_{jr}(p_j - k_j(k_j-1)c) \right) . \tag{9}$$

We now consider the Type I blocks B_i not analyzed with a Type IIa block above. Recall that, in a Type I block, there are $d \geq (m+1)/2$ processors busy at all times during the interval $[b_i, b_{i+1})$. Thus, we know that $W_i \geq (b_{i+1} - b_i)(m+1)/2$ and $I_i \leq (b_{i+1} - b_i)(m-1)/2$. Therefore,

$$I_i \leq \frac{m-1}{m+1} W_i \leq \frac{m-1}{m+1} W_i + \frac{2m}{m+1} \sum_{j \in F_i} \alpha_{ji}(p_j - k_j(k_j-1)c) , \tag{10}$$

since $\alpha_{ji} > 0$ for all $j \in F_i$ and, by Fact 1, $p_j > k_j(k_j-1)c$ for all $j \in F_i$.

Finally, we consider the single Type IIb block B_i (if it exists). B_i is the last block and contains jobs that keep $d \leq (m-1)/2$ processors busy for the entire interval. As with a Type IIa block, let $x = d \times (b_{i+1} - b_i)$ denote the work performed on these d processors in the interval (b_i, b_{i+1}) and let F'_i denote the set of indices of jobs executing on these d processors in the interval (b_i, b_{i+1}). Notice that $F'_i \subseteq F_i$ and, since no job starts in (b_i, b_{i+1}), $\sum_{j \in F'_i} k_j = d$.

$$\begin{aligned}
I_i &\leq \frac{m-d}{d} x \\
&= \frac{m-1}{m+1} x + \left(\frac{m-d}{d} - \frac{m-1}{m+1} \right) x \\
&\leq \frac{m-1}{m+1} W_i + \left(\frac{m}{d} - \frac{2m}{m+1} \right) \sum_{j \in F'_i} \alpha_{ji}\,(p_j + k_j(k_j-1)c) \quad \text{(by (2))} \\
&= \frac{m-1}{m+1} W_i + \frac{2m}{m+1} \sum_{j \in F'_i} \alpha_{ji}(p_j - k_j(k_j-1)c) \\
&\quad + \left(\frac{m}{d} - \frac{4m}{m+1} \right) \sum_{j \in F'_i} \alpha_{ji} p_j + \frac{m}{d} \sum_{j \in F'_i} \alpha_{ji} k_j(k_j-1)c \\
&\leq \frac{m-1}{m+1} W_i + \frac{2m}{m+1} \sum_{j \in F_i} \alpha_{ji}(p_j - k_j(k_j-1)c) \\
&\quad + \left(\frac{m}{d} - \frac{4m}{m+1} \right) \sum_{j \in F'_i} \alpha_{ji} p_j + \frac{m}{d} \sum_{j \in F'_i} \alpha_{ji} k_j(k_j-1)c . \qquad (11)
\end{aligned}$$

We now consider the same two cases from the analysis of Type IIa blocks.
Case 1: $1 \leq d \leq (m+1)/4$. In this case, we can substitute (6) into (11) to conclude that

$$\begin{aligned}
I_i &\leq \frac{m-1}{m+1} W_i + \frac{2m}{m+1} \sum_{j \in F_i} \alpha_{ji}(p_j - k_j(k_j-1)c) + \frac{2md(m-2d-1)c}{m+1} \\
&\leq \frac{m-1}{m+1} W_i + \frac{2m}{m+1} \sum_{j \in F_i} \alpha_{ji}(p_j - k_j(k_j-1)c) + \frac{m(m-1)^2 c}{4(m+1)} \quad \text{(by Lemma A.3)} . \qquad (12)
\end{aligned}$$

Case 2: $(m+1)/4 < d \leq (m-1)/2$. In this case, we can substitute (8) into (11) to conclude that

$$\begin{aligned}
I_i &\leq \frac{m-1}{m+1} W_i + \frac{2m}{m+1} \sum_{j \in F_i} \alpha_{ji}(p_j - k_j(k_j-1)c) + \frac{2m(d-1)(m-2d+1)c}{m+1} \\
&\leq \frac{m-1}{m+1} W_i + \frac{2m}{m+1} \sum_{j \in F_i} \alpha_{ji}(p_j - k_j(k_j-1)c) + \frac{m(m-1)^2 c}{4(m+1)} \quad \text{(by Lemma A.4)} . \qquad (13)
\end{aligned}$$

In conclusion, we sum over all blocks (using (7), (9), (10), (12), and (13)), to get

$$\begin{aligned} I &= \sum_i I_i \\ &\le \frac{m-1}{m+1}\sum_i W_i + \frac{2m}{m+1}\sum_i \sum_{j\in F_i} \alpha_{ji}(p_j - k_j(k_j-1)c) + mb \\ &\le \frac{m-1}{m+1} W + \frac{2m}{m+1}\sum_j (p_j - k_j(k_j-1)c) + mb \,, \end{aligned}$$

where $b = (m-1)^2 c/(4(m+1))$. □

4.2 Upper Bound for Even $m \ge 2$

For the case where m is even, we have the following theorem:

Theorem 4.2 *$\mathcal{R}[SET] \le 4(m-1)/m$ for even $m \ge 2$ when all jobs arrive at time 0.*

The proof for the even case follows almost exactly the proof for the odd case, so we will not repeat it here. In most places, all that is necessary is to replace m with $m-1$ in the text. In place of (3), the goal is to show:

$$I \le \frac{m-2}{m} W + \frac{2(m-1)}{m} \sum_j (p_j - k_j(k_j-1)c) + mb \,. \tag{14}$$

5 Upper Bounds for SET When Arrival Times Are Arbitrary

Previously, we assumed that all jobs arrived at time 0. However, our results still hold when each job J_j can have an arbitrary arrival time $a_j \ge 0$. As noted in the introduction, we assume that $i < j \Rightarrow a_i \le a_j$.

Let $W^*_{i,j}$ denote the optimal work for jobs $J_i, J_{i+1}, \ldots, J_j$. First, we notice that, in the case of arbitrary arrival times, the statement of Lemma 4.1 becomes

$$C^* \ge \max_j \left\{ a_j + \frac{W^*_{j,n}}{m} \right\} \,. \tag{15}$$

Theorem 5.1 $\mathcal{R}[SET] \le \begin{cases} 4(m-1)/m\,, & m \ge 2 \textit{ is even} \\ 4m/(m+1)\,, & m \ge 3 \textit{ is odd} \end{cases}$ *when arrival times are arbitrary.*

Proof To simplify notation, let

$$r = \begin{cases} 4(m-1)/m\,, & m \ge 2 \text{ is even} \\ 4m/(m+1)\,, & m \ge 3 \text{ is odd} \end{cases} .$$

The proofs of Theorems 4.1 and 4.2 hinge on the fact that a Type IIa block B_i must be followed immediately by a job J_l with $s_l = b_{i+1}$ and $k_l \ge m - d + 1$. However, when arrival times can be arbitrary, it is possible that a Type IIa block is *not* followed immediately by such a job. Therefore, we must modify the analysis of Type IIa blocks to show that the results still hold with arbitrary arrival times. The analyses of Type I and Type IIb blocks are unchanged by the introduction of arbitrary arrival times.

First, we notice that the given proofs of Theorems 4.1 and 4.2 hold for arbitrary online arrivals if all Type IIa blocks B_i have $d \ge 1$ processors busy at all times in the interval $[b_i, b_{i+1})$, and are

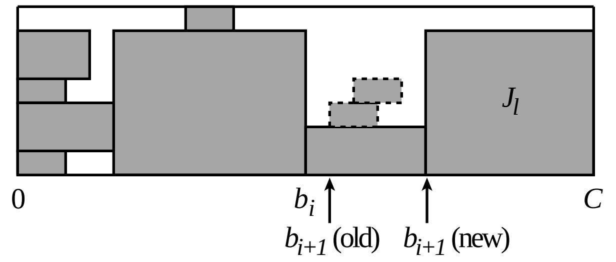

Figure 6: Modifying a Type IIa block B_i in the proof of Theorem 5.1.

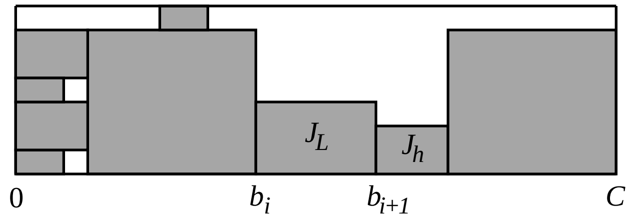

Figure 7: Case 1 in the proof of Theorem 5.1.

immediately followed by a job J_l with $s_l = b_{i+1}$ and $k_l \geq m - d + 1$. More generally, we can make the proofs hold if at least one of the job(s) executing on the d processors that are busy at all times in any Type IIa block B_i is immediately followed by a job J_l with $k_l \geq m - d + 1$, but J_l is in some block B_ι, where $\iota \geq i + 1$. (The jobs previously said to be executing on these d processors in Block B_i are actually contained in Type IIa blocks $B_i, B_{i+1}, \ldots, B_{\iota-1}$.) In each of these cases, we can simply ignore any job(s) J_j with $b_{i+1} \leq s_j = a_j < s_l$, reset $b_{i+1} = s_l$, and modify the indices of successive blocks accordingly. Then the proof holds for the modified partition. See Figure 6 for an illustration of this modification.

Therefore, consider an online schedule containing a Type IIa block B_i in which the job(s) executing on the d processors that are busy at all times in the block (and consecutive Type IIa blocks $B_i, B_{i+1}, \ldots, B_{\iota-1}$) are not immediately followed by a job J_l with $k_l \geq m - d + 1$ in block B_ι. In particular, consider the last such Type IIa block B_i. Let J_h be a job with $k_h \leq m - d$, where $d \geq 0$ is the number of processors that are busy at all times in B_i, and $s_h = a_h = b_{i+1}$. Also, let J_L be a job with $C_L = \max_{1 \leq j < h}\{C_j\}$, and let $W_{i,j}$ and $I_{i,j}$ denote the work and idle area, respectively, of an online schedule for just jobs $J_i, J_{i+1}, \ldots, J_j$. We consider two cases:
Case 1: $C_L \leq a_h$. (See Figure 7.) In this case, all jobs J_j with $j \geq h$ are scheduled after all jobs J_j with $j < h$. Therefore,

$$\begin{aligned}
C &= a_h + \frac{W_{h,n} + I_{h,n}}{m} \\
&\leq a_h + r\left(\frac{W^*_{h,n}}{m}\right) + b && \text{(from the proofs of Theorems 4.1 and 4.2)} \\
&< r\left(a_h + \frac{W^*_{h,n}}{m}\right) + b \\
&\leq r \max_j \left\{a_j + \frac{W^*_{j,n}}{m}\right\} + b \\
&\leq r\,C^* + b && \text{(by (15))}\ .
\end{aligned}$$

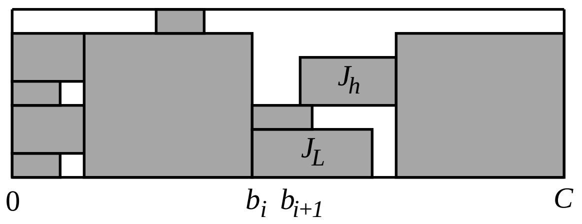

Figure 8: Case 2 in the proof of Theorem 5.1.

Case 2: $C_L > a_h$. (See Figure 8.) We first notice that there must not be any job J_j, where $j < h$, with $C_j > a_h$ and $k_j \geq m-d+1$. Otherwise, this job would immediately follow the job(s) executing on the d processors that are busy at all times in B_i, and therefore satisfy the definition of J_l given above. Thus, in particular, $k_L \leq m - d$. Also, since there were at least $m - d$ processors idle when J_h arrived and $a_L \leq a_h$, $s_L \leq a_h$. Now consider a modified SET schedule in which jobs J_j, where $h \leq j \leq n$, are scheduled to start at or after time C_L. Since the makespan of this modified schedule can be no shorter than the makespan of the original SET schedule, we can bound the makespan of the original SET schedule as follows:

$$\begin{aligned}
C &\leq C_L + \frac{W_{h,n} + I_{h,n}}{m} \\
&\leq C_L + r\left(\frac{W^*_{h,n}}{m}\right) + b && \text{(from the proofs of Theorems 4.1 and 4.2)} \\
&< (C_L - a_h) + r\left(a_h + \frac{W^*_{h,n}}{m}\right) + b \\
&\leq t_L + r\,C^* + b && \text{(since } C_L = s_L + t_L \text{ and } s_L \leq a_h \text{ and by (15))} \\
&\leq 2k_L c + r\,C^* + b && \text{(by Fact 1 since } k_L < m\text{)} \\
&\leq r\,C^* + (b + (m+1)c) && \text{(since } k_L \leq m - d \leq (m+1)/2\text{)}\ .
\end{aligned}$$

□

6 General Lower Bounds

We do not know of any existing general lower bounds specifically for parallel job scheduling, although we show below how lower bound instances for classical job scheduling [7] can be adapted to apply to our problem if no additive constant is allowed in the definition of competitive ratio (i.e., a *strong* competitive ratio). We then show that, if an additive constant proportional to m is allowed, the competitive ratio of any online algorithm is still greater than 1.

We first show that the strong competitive ratio of any online algorithm is at least 3/2 for $m \geq 2$ and at least 5/3 for $m \geq 3$.

Theorem 6.1 *The strong competitive ratio of any online algorithm is at least* $3/2$ *for* $m \geq 2$.

Proof We frame the proof as a contest between an adversary, which issues the job sequence, and an arbitrary online algorithm A. We will show that the adversary can force A to schedule its jobs so that its makespan at least 3/2 times the optimal makespan. The adversary first issues up to m jobs $p_1 = p_2 = \cdots p_m = c$. If A assigns one or more of these m jobs to two or more processors, or two or more jobs to the same processor, the adversary stops because A has incurred makespan at least $3/2$ times the optimal makespan of c. Otherwise, the adversary issues $p_{m+1} = 2c$. Since this job will require at least $2c$ units of time to execute, regardless of the number of processors assigned to it, A has incurred makespan at least $3c$. On the other hand, the optimal makespan is $2c$. Therefore, A has competitive ratio at least $3/2$. $\square$

Similarly, we can adapt the classical scheduling lower bound for $m \geq 3$ [7].

Theorem 6.2 *The strong competitive ratio of any online algorithm is at least* $5/3$ *for* $m \geq 3$.

Proof Let A be an arbitrary online algorithm. An adversary begins by issuing m jobs $p_1 = p_2 = \cdots = p_m = c/3$. If A assigns one or more of these jobs to two or more processors, or if it assigns two or more jobs to the same processor, the adversary stops because A has incurred makespan at least twice the optimal makespan of $c/3$. Otherwise, the adversary issues m additional jobs $p_{m+1} = p_{m+2} = \cdots = p_{2m} = c$. If A assigns any of these latter m jobs to two or more processors, then at least two of them must be assigned to the same processor. In this case, or if A otherwise assigns two or more to the same processor, the adversary stops, and A's makespan will be at least $7c/3$. Since the optimal makespan at this point is $4c/3$, A's competitive ratio is at least $7/4$. Otherwise, the adversary issues a final job $p_{2m+1} = 2c$. Since this last job will require at least $2c$ units to execute regardless of the number of processors assigned to it, A's makepan will be at least $10c/3$. On the other hand, the optimal makespan is $2c$, so A has competitive ratio at least $5/3$. $\square$

We now consider the case in which a constant proportional to m is allowed.

Theorem 6.3 *If an additive constant proportional to* m *is allowed in the definition of competitive ratio, then the competitive ratio of any online algorithm is still strictly greater than* 1 *for any* $m \geq 2$.

Proof We frame the proof as a contest between an adversary, which issues the job sequence, and an arbitrary online algorithm A. We will show that the adversary can force A to schedule its jobs so that the difference between its makespan and the optimal makespan is arbitrarily large, thus yielding a competitive ratio greater than 1. The adversary will issue jobs in phases. In each phase, the adversary begins by issuing m jobs $p_1 = p_2 = \cdots p_m = c$. If A assigns one or more of these m jobs to two or more processors, or two or more jobs to the same processor, A has used at least $2c$ units of time to schedule the m jobs. Otherwise, A has used only c units of time to schedule the jobs, so the adversary issues another job $p_{m+1} = 2c$. Since this job will require at least $2c$ units of time to execute, regardless of the number of processors assigned to it, A has now used at least $3c$ units of time to schedule the $m + 1$ jobs. Now, to round out this phase, the adversary issues a job with length Pc, where Pc is arbitrarily large (but finite). If A assigns this job to m processors, then A requires at least $2c + Pc/m + (m-1)c$ units of time in a phase with m small jobs and at least $3c + Pc/m + (m-1)c$ units of time in a phase with $m+1$ small jobs. On the other hand, if A assigns the large job to fewer than m processors, then A needs at least $2c + Pc/(m-1) + (m-2)c$ units of time in a phase with m small jobs and at least $3c + Pc/(m-1) + (m-2)c$ units of time in a phase with $m+1$ small jobs. The adversary will repeatedly issue new phases of jobs, as described above, until A assigns the large job to less than m processors.

Let f denote the number of phases and let a denote the number of phases, except the last, with m small jobs. First, suppose f is finite and, without loss of generality, that the last phase contains m small jobs. Then A's makespan is at least

$$\begin{aligned} C &\geq a\left(2c + \frac{Pc}{m} + (m-1)c\right) + (f-a-1)\left(3c + \frac{Pc}{m} + (m-1)c\right) + \left(2c + \frac{Pc}{m-1} + (m-2)c\right) \\ &= \frac{((m-1)f+1)Pc}{m(m-1)} + ((m+2)f - a - 2)c \end{aligned}$$

and the optimal makespan is at most

$$C^* \le ac + (f-a-1)(2c) + c + f\left(\frac{Pc}{m} + (m-1)c\right) = \frac{fPc}{m} + ((m+1)f - a - 1)c\ .$$

Therefore, the competitive ratio of A is greater than 1 since $C - C^* = Pc/(m(m-1)) + (f-1)c$ can be arbitrarily large.

On the other hand, if f is infinite, then A's makespan is at least

$$C \ge a\left(2c + \frac{Pc}{m} + (m-1)c\right) + (f-a)\left(3c + \frac{Pc}{m} + (m-1)c\right) = \frac{fPc}{m} + ((m+2)f - a)c$$

and the optimal makespan is at most

$$C^* \le ac + (f-a)(2c) + f\left(\frac{Pc}{m} + (m-1)c\right) = \frac{fPc}{m} + ((m+1)f - a)c\ .$$

Therefore, the competitive ratio of A is greater than 1 since $C - C^* = fc$, which is arbitrarily large. □

7 Conclusions

We studied the problem of scheduling perfectly malleable parallel jobs, in which a job with length p_j that is assigned to k_j processors has execution time $t_j = p_j/k_j + (k_j - 1)c$, where $c > 0$ is a constant setup time associated with each processor beyond the first. For this problem, which models jobs employing data parallelism, we showed that the competitive ratio of a simple online algorithm is $4(m-1)/m$ for even $m \ge 2$ and $4m/(m+1)$ for odd $m \ge 3$.

There are three directions for future research. First, we would like to know whether there is another online algorithm which might beat SET in terms of competitive ratio. A promising candidate is the algorithm that assigns k_j to minimize the completion time of a job instead of its execution time.

We also plan to study parallel job scheduling with other optimization criteria such as maximum or average response time ($C_j - a_j$), as these criteria are well suited for the general online problem with stretched arrival times.

Finally, we are interested in similar scheduling problems that use a different formula for the execution time. For example, as discussed in the introduction, one might define t_j to be $p_j/k_j + (k_j - 1)c_j$ to better reflect more general message passing computations. We point out that the competitive ratio of SET is at least m for this model, even though it is only slightly different from the model we studied in this paper.

Theorem 7.1 *If $t_j = p_j/k_j + (k_j - 1)c_j$, where c_j may be different for each job, then the competitive ratio of SET is at least m.*

Proof Consider an instance consisting of $n = 2\alpha m$ jobs, where α is a positive integer. For odd j, we define $p_j = B$ and $c_j = B/2$. For even j, we define $p_j = 1 + \epsilon$, where $\epsilon > 0$ is arbitrarily small, and $c_j = 1/(m(m-1))$. For all j, $a_j = 0$. SET will assign each of the odd-indexed jobs to one processor and each of the even-indexed jobs to m processors. If backfilling is not allowed, the algorithm will schedule all the jobs at different start times, and thus in a sequential order. Therefore, the makespan of the schedule on this sequence, as $\epsilon \to 0$, is

$$C = \frac{n}{2}\left(B + \frac{1}{m} + (m-1)\frac{1}{m(m-1)}\right) = n\left(\frac{B}{2} + \frac{1}{m}\right)\ .$$

On the other hand, an offline algorithm can construct a better schedule by assigning each job to one processor and executing all the odd-indexed jobs followed by the even-indexed jobs. In this case, the optimal makespan, as $\epsilon \to 0$, is

$$C^* \leq \frac{n}{2m}(B+1) \ .$$

Thus, the competitive ratio of SET is at least $(mB+2)/(B+1) \to m$ for arbitrarily small $\epsilon > 0$ and arbitrarily large $B > 0$. $\Box$

8 Acknowledgments

We would like to thank the anonymous referees for their meticulous readings of our manuscript and their helpful suggestions, which improved the quality of the paper.

References

[1] A. Allahverdi, J. N. D. Gupta, and T. Aldowaisan. A review of scheduling research involving setup considerations. *Omega, The International Journal of Management Science*, 27:219–239, 1999.

[2] A. Allahverdi, C. T. Ng, T. C. E. Cheng, and M. Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, to appear.

[3] K. P. Belkhale and P. Banerjee. An approximate algorithm for the partitionable independent task scheduling problem. In *Proceedings of the IEEE International Parallel Processing Symposium*, pages 72–75, 1990.

[4] J. Blazewicz, M. Machowiak, G. Mounie, and D. Trystram. Approximation algorithms for scheduling independent malleable tasks. In *Proceedings of Euro-Par 2001, LNCS 2150*, pages 191–197, 2001.

[5] L. W. Dowdy. On the partitioning of multiprocessor systems. Technical report, Vanderbilt University, 1988.

[6] J. Du and J. Y.-H. Leung. Complexity of scheduling parallel task systems. *SIAM Journal on Discrete Mathematics*, 2(4):473–487, 1989.

[7] U. Faigle, W. Kern, and G. Turán. On the performance of on-line algorithms for partition problems. *Acta Cybernetica*, 9(2):107–119, 1989.

[8] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong. Theory and practice in parallel job scheduling. In *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pages 1–34. Springer Verlag, 1997.

[9] A. Feldmann, M.-Y. Kao, J. Sgall, and S.-H. Teng. Optimal on-line scheduling of parallel jobs with dependencies. *Journal of Combinatorial Optimization*, 1(4):393–411, 1998.

[10] A. Feldmann, J. Sgall, and S.-H. Teng. Dynamic scheduling on parallel machines. *Theoretical Computer Science*, 130(1):49–72, 1994.

[11] M. Garey and R. Graham. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4(2):187–200, 1975.

[12] J. T. Havill. A competitive online algorithm for a parallel job scheduling problem. In *Proceedings of the 12th IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 611–616, 2000.

[13] K. Jansen. Scheduling malleable parallel tasks: An asymptotic fully polynomial time approximation scheme. *Algorithmica*, 39(1):59–81, 2004.

[14] K. Jansen and L. Porkolab. Linear-time approximation schemes for scheduling malleable parallel tasks. *Algorithmica*, 32(3):507–520, 2002.

[15] B. P. Lester. *The Art of Parallel Programming.* Prentice Hall, 1993.

[16] W. Ludwig and P. Tiwari. Scheduling malleable and nonmalleable parallel tasks. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 167–176, 1994.

[17] W. Mao, J. Chen, and W. Watson III. On-line algorithms for a parallel job scheduling problem. In *Proceedings of the 11th IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 753–757, 1999.

[18] G. Mounie, C. Rapine, and D. Trystram. Efficient approximation algorithms for scheduling malleable tasks. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 23–32, 1999.

[19] C. Rapine, I. D. Scherson, and D. Trystram. On-line scheduling of parallelizable jobs. In *Proceedings of the European Conference on Parallel Computing, LNCS 1470*, pages 322–327, 1998.

[20] K. C. Sevcik. Application scheduling and processor allocation in multiprogrammed parallel processing systems. *Performance Evaluation*, 19:107–140, 1994.

[21] J. Sgall. On-line scheduling of parallel jobs. In *Proceedings of the International Symposium on Mathematical Foundations of Computer Science, LNCS 841*, pages 159–176, 1994.

[22] J. Turek, J. L. Wolf, and P. S. Yu. Approximate algorithms for scheduling parallelizable tasks. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 323–332, 1992.

[23] Q. Wang and K. H. Cheng. A heuristic of scheduling parallel tasks and its analysis. *SIAM Journal on Computing*, 21(2):281–294, 1992.

A Lemmas Referenced in the Proof of Theorem 4.1

Lemma A.1 *If $\sum_{j\in S} k_j = d$, then $\sum_{j\in S} k_j(k_j+1) \le d(d+1)$ and $\sum_{j\in S} k_j(k_j-1) \le d(d-1)$, where $k_j \in \{1,2,\ldots,d\}$ for all $j \in S$.*

Proof Let $n_{i,k}$ be the number of elements in S with $k_j = k$. Then $\sum_{j\in S} k_j$ and $\sum_{j\in S} k_j(k_j+1)$ can be rewritten as $\sum_{k=1}^{d} kn_{i,k}$ and $\sum_{k=1}^{d} k(k+1)n_{i,k}$, respectively. Therefore, $\sum_{j\in S} k_j(k_j+1) = \sum_{k=1}^{d} k(k+1)n_{i,k} \le (d+1)\sum_{k=1}^{d} kn_{i,k} = (d+1)\sum_{j\in F_i'} k_j = d(d+1)$. The proof of the second inequality is similar. □

Lemma A.2 $-2m(m^2+m-4md+4d^2)c/(m+1) < 0$ *for* $m \ge 2$ *and* $1 \le d \le (m+1)/4$.

Proof Let $f(m,d) = -2m(m^2+m-4md+4d^2)c/(m+1)$. The first partial derivative of f with respect to d is $f_d(m,d) = -2m(8d-4m)c/(m+1)$ which gives an extremum at $d = m/2$. The second partial derivative $f_{dd}(m,d) = -16mc/(m+1) < 0$ indicates that the function is concave down. Therefore, $f(m,d) \leq f(m,m/2) = -2m^2c/(m+1) < 0$. $\square$

Lemma A.3 *$2md(m-2d-1)c/(m+1) \leq m(m-1)^2c/(4(m+1))$ for $1 \leq d \leq (m+1)/4$.*

Proof Let $f(m,d) = 2md(m-2d-1)c/(m+1)$. The first partial derivative of f with respect to d is $f_d(m,d) = 2m(m-4d-1)c/(m+1)$ which gives an extremum at $d = (m-1)/4$. The second partial derivative $f_{dd}(m,d) = -8mc/(m+1) < 0$ indicates that the function is concave down. Therefore, $f(m,d) \leq f(m,(m-1)/4) = m(m-1)^2c/(4(m+1))$. $\square$

Lemma A.4 *$2m(d-1)(m-2d+1)c/(m+1) \leq m(m-1)^2c/(4(m+1))$ for $(m+1)/4 < d \leq (m-1)/2$.*

Proof Let $f(m,d) = 2m(d-1)(m-2d+1)c/(m+1)$. The first partial derivative of f with respect to d is $f_d(m,d) = 2m(m-4d+3)c/(m+1)$ which gives an extremum at $d = (m+3)/4$. The second partial derivative $f_{dd}(m,d) = -8mc/(m+1) < 0$ indicates that the function is concave down. Therefore, $f(m,d) \leq f(m,(m+3)/4) = m(m-1)^2c/(4(m+1))$. $\square$