Minimizing the Flow Time for Parallelizable Task Systems (Preliminary Version)

Jason Glasgow
Technion IIT, Haifa, Israel *

Hadas Shachnai
Technion IIT, Haifa, Israel †

Abstract

Consider the problem of scheduling tasks on a multiprocessor where each task can be parallelized to run simultaneously on several processors. The number of processors assigned to the execution of a task is chosen at runtime. In general, increasing the number of processors assigned to a task will decrease its execution time. Our objective is to schedule the tasks so as to minimize the mean flow time (MFT), also known as the average response time of the system. We show that this problem is NP-hard in the strong sense. For scenarios in which the runtime per task decreases linearly with the number of processors allotted to it, we present the *Sorted Earliest Completion Time* (SECT) heuristic. SECT is shown to achieve a worst case bound of 3 to the MFT of the optimal schedule in time $O(n \log n + nm)$, where m is the number of processors, and n is the number of tasks.

Key words: NP-hard, parallel task system, mean flow time, average response time, non-preemptive scheduling

AMS subject classifications: 68B30, 90B35

*EE Dept., Technion IIT, Email: glasgow@tx.technion.ac.il. This research was supported in part by the Lady Davis Fellowship.

†CS Dept., Technion IIT. Email: hadas@cs.technion.ac.il. Currently at IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598. Email: hadas@watson.ibm.com.

1 Introduction

We study the problem of scheduling a set of tasks on a multiprocessor, which is classic in resource allocation. Typically, scheduling algorithms take as input an arbitrary number of tasks, each with an execution time, and produce as output a mapping of tasks to processors and a list of start times which will minimize either the average response time (mean flow time (MFT)), or the finish time of the entire schedule (makespan). The problem of finding a schedule that minimizes the MFT where each task runs on a single processor, has been widely studied [3, 2, 4].

In recently developed multiprocessor systems [7] parallelizable tasks may be scheduled to run simultaneously on several processors, thus decreasing their runtimes. We study scheduling heuristics for parallelizable tasks, which aim to minimize the MFT of the system. The set of tasks handled by the scheduler may be of the following types:

1. Non-malleable—Some applications are written for a fixed number of processors. In this case the operating system cannot arbitrarily assign a different number of processors to the execution of a task.

2. Malleable linear—Corresponds to applications where the number of processors assigned to a task can be determined by the operating system. In the linear case the runtime of a task decreases linearly with the number of processors assigned to it. This fits the many applications that can be trivially parallelized, such as database searches [12] and circuit simulation using waveform relaxation [6]. In these cases we expect a speedup function that decreases linearly until the problem's maximum level of parallelism is reached. There is no further decrease in the execution time if additional processors are assigned to a task.

3. Malleable non-linear—When the overhead inherent in parallelizing the task on several processors becomes a significant factor, the decrease in runtime is only sub-linear. This fits applications, in which communication costs are significant. In analyzing the non-linear case, we refer to the notion of *work*, defined as the sum of the running times of the processors allocated to a specific task. The *work function* describes the dependence of the work needed to execute a task upon the number of processors allocated to it and includes the overhead that stems from the communication and synchronization between the processors engaged in the execution of a single task. It also includes all other system overhead due to paging, context switching, etc. In most common scenarios the communication and synchronization overhead increases with the number of involved processors and therefore we often refer in the present context to work functions that are *non-decreasing* in the number of processors.

The MFT, which is the average completion time of a task in the system, is typically used as a metric when evaluating schedules in interactive environments where users are waiting for jobs to complete. Minimizing the MFT is equivalent to minimizing the average time that a user waits for results. In this paper we focus on the problem of scheduling parallelizable tasks in order to minimize the MFT.

Although the literature on minimizing the mean flow time for *uniprocessor* tasks is extensive, little attention was given to scheduling *parallelizable* tasks to minimize the flow time. For the case of non-parallelizable tasks scheduled on identical machines, the *Shortest Processing Time* first (SPT) algorithm is known to be optimal with respect to the MFT. We will see in subsequent sections that the principle of scheduling short tasks first carries over to the parallelizable case as well. Bruno et al. present in [1] a polynomial time algorithm that produces the optimal schedule also for the model of unrelated machines. In a seminal paper on minimizing the average response time[9] for parallelizable tasks, Turek et al. introduce the *SMART* algorithm, that provides schedule whose flow time is within a bound of 32 to the optimum for non-malleable tasks. In [10] they present an algorithm for malleable non-linear tasks with a bound of 3 to the optimum. The asymptotic complexity of the algorithm is $O(n^2(n+m))$ where n is the number of tasks, and m the number of processors.

In the present work we consider scenarios where the speedup gained by running a task on several processors is linear in the number of processors. Thus the work function is constant. We present a simple heuristic which we call the Sorted Earliest Completion Time (SECT) algorithm, that achieves a worse case bound of 3 to the optimal MFT with complexity $O(n(m+\log n))$.

The rest of the paper is organized as follows: In Section 2 we show that the problem of scheduling n tasks on m identical processors for non-malleable task systems with the goal of minimizing the mean flow time (or NMS-MFT for short) is NP-hard in the strong sense. In Section 3 we show that the more general problem of scheduling malleable tasks to minimize the MFT is also NP-hard in the strong sense. In Section 4 we present and analyze the performance of the *Sorted Earliest Completion Time* (SECT) algorithm. We show that for linear speedup task systems it produces schedules with a worst case bound of 3 to the optimum in time $O(n \log n + nm)$. We conclude in Section 5 with experimental results and discussion.

2 NMS scheduling for Minimum Flow Time is NP-hard

The NMS-MFT decision problem is described as follows:
Instance: A finite number m of identical processors, a bound $F \in Z^+$ on the flow time, a set of n tasks $T = \{T_1, T_2, \ldots T_n\}$ and for each task T_i, an integer number of required processors $1 \leq \delta_i \leq m$, and an integer runtime $t_i \in Z^+$ when executed on δ_i processors.
Question: Can a valid schedule S be constructed, that specifies a start time s_i for each task, such that the flow time is less than the bound, that is $ft(S) = \sum_{T_i \in T}(s_i + t_i) \leq F$?

A valid schedule is defined as one in which at any point in time, t, at most m processors are busy, i.e.

$$\text{for all } \; t \geq 0 \qquad \sum_{i : s_i \leq t < s_i + t_i} \delta_i \leq m \; .$$

Theorem 1 *The NMS-MFT decision problem is NP-complete in the strong sense.*

Proof: In the following proof we adopt the notation of [5]. We use a transformation from the 3-partition problem:
Instance: A finite set $A = \{a_1, \ldots, a_{3m}\}$ of $3m$ elements, a bound $B \in Z^+$, and a size $s(a_i) \in Z^+$ for each $a_i \in A$, such that each $s(a_i)$ satisfies $B/4 < s(a_i) < B/2$ and $\sum_{a_i \in A} s(a_i) = Bm$.
Question: Can the set A be partitioned into m disjoint sets $S_1, S_2, \ldots, S_m$ such that, for $1 \leq j \leq m$, $\sum_{a_i \in S_j} s(a_i) = B$?
Let Π be the decision problem 3-partition, Π' be the decision problem NMS-MFT, D_Π the domain of Π and $D_{\Pi'}$ the domain of Π'. We prove that NMS-MFT is NP-complete in the strong sense by showing that $\Pi' \in$ NP, and that there exists a pseudo-polynomial transformation $f(\cdot) : D_\Pi \rightarrow D_{\Pi'}$.

Clearly $\Pi' \in$ NP, since a non-deterministic algorithm only needs to produce the start time for each task. Then we can check in polynomial time that this is a valid schedule and that the flow time is less than the bound.

We define the pseudo-polynomial transformation $f(\cdot)$ as follows:

Transformation $f(\cdot)$

Input: an instance $I \in D_\Pi$ of the 3-partition problem.

Output: an instance $I' \in D_{\Pi'}$ of the NMS-MFT decision problem.

1. Let there be m identical processors and $n = 3m + Bm$ tasks.

2. Let the runtimes of the first $3m$ tasks be identical to the sizes of the items from 3-partition, i.e. $t_i = s(a_i)$, $\forall\, 1 \leq i \leq 3m$, such that each task requires a single processor, $\delta_i = 1$, $\forall\, 1 \leq i \leq 3m$. These tasks will be referred to as *uniprocessor* tasks.

3. For the remaining Bm tasks assign full parallelism, $\delta_i = m$, and runtimes of $t_i = B^2 m^3$ $\forall\, 3m + 1 \leq i \leq 3m + Bm$. These will be referred to as *parallelizable* tasks.

4. We choose the bound

$$F = \frac{5}{2}Bm + B^2m + \frac{B^3m^3}{2} + \frac{B^4m^4}{2} .$$

The proof proceeds in two parts:

1. It is easy to show, that for any schedule S in one of the uniprocessor tasks is scheduled after the completion of a parallel task (see Figure 1b) the flow time exceeds the bound, i.e. $ft(S) > F$. To prove this, assume that first the tasks $T_1, \ldots, T_{3m-1}$ are scheduled, followed by one of the tasks $T_i, i > 3m$. After the first parallelizable task is scheduled, the task T_{3m} is scheduled, followed by some assignment of the remaining tasks as in Figure 1b. Without loss of generality choose the first large task to be T_{3m+1}. For any schedule S of this form, a lower bound on the flow time is:

$$\begin{aligned} ft(S) &\geq Bm/4 + Bm/2 + (Bm - t_{3m}) + B^2m^2 + (B^2m^2 + t_{3m}) + \sum_{i=2}^{Bm}(i \cdot t_{3m+i}) \\ &> F . \end{aligned}$$

   $Bm/4$ is the lower bound on the contribution to the flow time due to the first m tasks, $Bm/2$ due to the second m tasks scheduled and $Bm - t_{3m}$ due to the last $m-1$ non-parallelizable tasks. B^2m^2 is the earliest that task T_{3m+1} can finish execution, and $Bm + t_{3m}$ is the earliest that task T_{3m} can finish. Finally, we add the lower bound on the flow time of the Bm parallelizable tasks. Since $ft(S) > F$, we have shown that any schedule in which a uniprocessor task completes execution after a parallelizable task exceeds the upper bound of the optimal schedule, F. Therefore we consider only schedules, in which the $3m$ uniprocessor tasks are scheduled first, 3 tasks to each processor, followed by the remaining Bm tasks, each scheduled on m processors as shown in Figure 1a and 1c.

2. To prove that there is solution to 3-partition if and only if $ft(S) < F$: It is sufficient to show, that in order not to exceed the bound F on the flow time, all $3m$ uniprocessor tasks have to finish execution by time B. Observe, that this is the interpretation of the size F, chosen as the bound: It is the largest flow time of any schedule where the $3m$ uniprocessor tasks are scheduled in three levels and finish simultaneously at time B (over all possible sets of $3m$ tasks where $B/4 < t_i < B/2$). It is easily verified that schedules of the form shown in Figure 1a satisfy the bound F, and correspond to a solution to the original 3-partition problem. To show that only schedules of this form can satisfy the bound, we compute the minimum flow time of any schedule of the tasks generated by the transformation in which one of the uniprocessor tasks completes at time $B + 1$ as in Figure 1c. The minimum flow time for the $3m$ uniprocessor tasks using the bounds on their execution time ($B/4 < t_i < B/2$) is $7Bm/4$, and adding the flow time of the Bm parallel tasks yields a schedule S, with flow time

$$ft(S) \geq \frac{7}{4}Bm + \sum_{i=1}^{Bm}\left(i\frac{t_i}{m} + B + 1\right) = \frac{11}{4}Bm + B^2m + \frac{B^3m^3}{2} + \frac{B^4m^4}{2} > F$$

   which completes the proof. ∎

Corollary 1 *The problem of finding a schedule that minimizes the MFT for non-malleable task system is NP-hard in the strong sense.*

3 Malleable scheduling for minimum flow time is NP-hard

In malleable scheduling each task may be scheduled on a number of processors that is determined at runtime. The amount of time required to execute a task depends on the number of processors that are allocated to the task, as well as the task's speedup function. It is readily seen from the proof of Theorem 1 that minimizing the MFT for malleable task systems with arbitrary speedup functions is also NP-hard in the strong sense: Choose an instance of the problem in which the runtime for any task T_i is

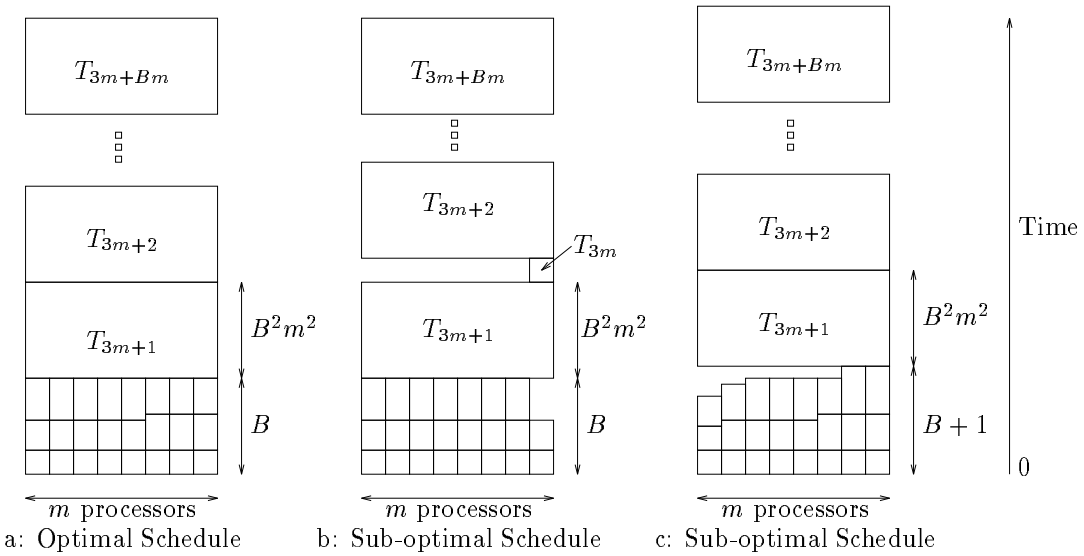


Figure 1: An optimal schedule and two sub-optimal schedules for NMS-MFT. The x-axis represents the m identical processors, and the y-axis represents time. Each rectangle represents a single task, where the width is the number of processors executing the task, and the height is the execution time.

∞ for any number of processors other than δ_i. When T_i is scheduled on δ_i processors, its runtime will be given as in the transformation $f(\cdot)$.

In extending Corollary 1 to malleable task systems, we first define the decision problem for malleable scheduling to minimize the flow time (MS-MFT), in which the tasks have linear speedup:
Instance: A finite number m of identical processors, a bound $F \in Z^+$ on the flow time, a set of n tasks $T = \{T_1, T_2, \ldots T_N\}$ and for each task T_i, an integer runtime $t_i \in Z^+$ when scheduled on a single processor, and an integer maximum number of processors $1 \leq \delta_i \leq m$.
Question: Can a valid schedule be constructed that specifies a start time s_i and a level of parallelism β_i for each task, such that $\beta_i \leq \delta_i$ and the flow time is less than the bound, that is $\sum_{T_i \in T}(s_i + t_i/\beta_i) \leq F$?

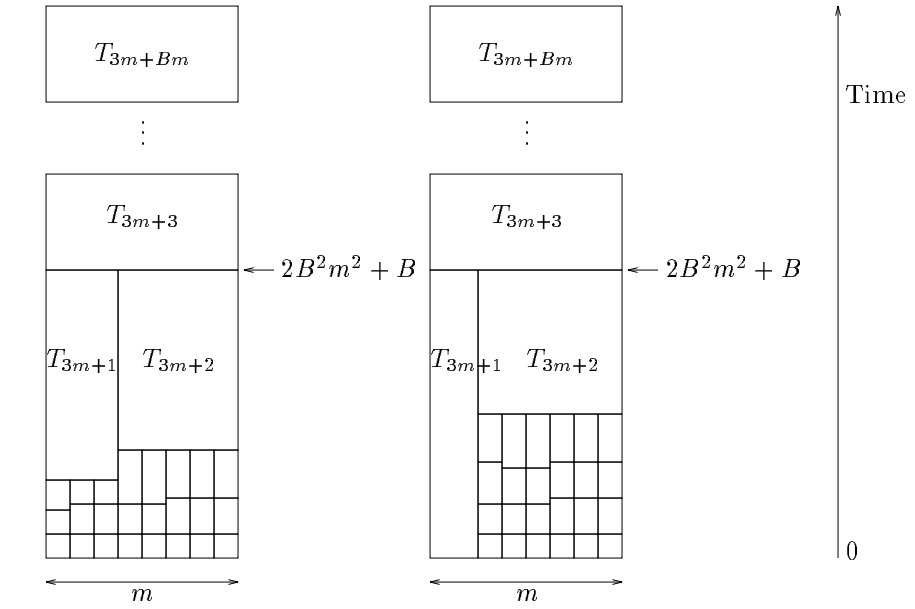
Theorem 2 *The problem of finding a schedule that minimizes the MFT for linear speedup task systems is NP-hard in the strong sense.*

Proof: Basically the proof used for NMS-MFT also applies to MS-MFT, where each task has a limit on its maximum level of parallelism, δ_i, but it can be scheduled on any number p of processors from 1 to δ_i with a runtime t_i/p. We show that this is NP-hard in the strong sense by taking an instance of this problem that corresponds to the instance used in the proof of Theorem 1, with the same bound F.

First note, that parallelizing any of the first $3m$ tasks can only increase the flow time, since the runtime of each task is constant regardless of the number of processors assigned to it. Also, as in the non-malleable case, scheduling any of the uniprocessor tasks after one of the parallelizable tasks completes will produce a schedule whose flow time exceeds the bound F. Thus, we reduce the discussion to schedules in which the uniprocessors tasks are either scheduled first, or in parallel to the long tasks (see Figure 2).

It remains to show, that $ft(S) \leq F$ if and only if the latest completion time of the first $3m$ tasks is B. We show below, that any scenario not satisfying the last condition would yield MFT larger than F:

(a) Assume a schedule S where each of the processors finishes executing its 3 uniprocessor tasks at one of two possible finish times and in which the next two parallelizable tasks finish simultaneously at time $2B^2m^2 + B$ as shown in Figure 2a, i.e., the first $3m + 2$ tasks finish simultaneously. Clearly the flow time of this schedule is less the flow time of a schedule in which the first $3m + 3$ or more tasks finish simultaneously. Also, for the given sizes of the parallelizable tasks, we cannot form



a: sub-optimal schedule b: sub-optimal schedule

Figure 2: Schedule in which all processors finish simultaneously after $3m + 2$ tasks have been executed.

such a schedule with only $3m + 1$ tasks. It is readily seen that

$$ft(S) \geq \sum_{i=3}^{Bm} \left(i \cdot B^2m^2 + B\right) + 2(2B^2m^2 + B) + \frac{7}{4}Bm > F \ .$$

The schedule shown in Figure 2a considers only the situation where tasks T_{3m+1} and T_{3m+2} are scheduled after the completion of the non-parallelizable tasks.

(b) If any of the parallelizable tasks starts at time 0 (as in Figure 2b), it is obvious that the flow time of the schedule will still exceed the bound F, since this will only produce a schedule with a larger flow time than that of Figure 2a.

Following the previous proof for NMS-MFT, the only schedule that will satisfy the bound corresponds to a solution to 3-partition. Thus, we have shown the existence of a schedule, S, that satisfies the bound $ft(S) \leq F$ if and only if there exists a solution to the 3-partition problem. Since the description of the malleable scheduling problem is identical to that of the NMS-MFT, $f(\cdot)$ continues to be a polynomial time transformation. ■

Corollary 2 *MS-MFT is NP-hard in the strong sense for non-linear speedup task systems that have non-decreasing work functions.*

4 Performance of Sorted ECT

In the previous sections we showed that scheduling to minimize the MFT is NP-hard in the strong sense for both malleable and non-malleable task systems. The problem of minimizing the MFT for tasks with arbitrary speedup functions is at least as hard as the problem of scheduling non-malleable tasks. The best known algorithm for this problem obtains a worst case bound of 32 to the optimum[9]. In this section we present an algorithm for scheduling linear speedup tasks which aims to minimize the mean flow time. This algorithm runs in time $O(n \log n + nm)$ and has a worst case performance bound of 3.

We motivate the usage of our algorithm with an example that show that using the original SPT algorithm for parallelizable tasks can produce schedules that are as bad as m times the optimum. In short, there is no constant bound on its worst case behavior. In cases where the number of tasks is

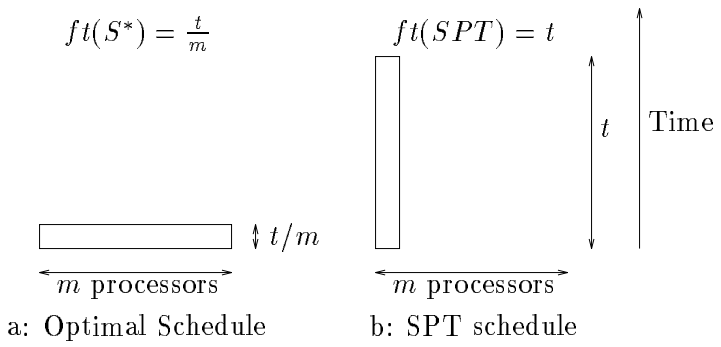


Figure 3: Optimal and SPT schedule for MS-MFT

much larger than the number of processors, SPT may produce schedules where the flow time is close to the optimal, however for $n \ll m$ SPT may obtain a schedule with a flow time m times greater than the optimum. (See Figure 3 for a trivial example.)

As we are developing a worst case bound on the behavior of an algorithm, it is essential that we obtain a lower bound on the flow time of the optimal schedule for any given set of tasks. At this stage we are not interested in how the tasks are actually scheduled on the processors. If follows that the optimal schedule, S^*, can do no better than to execute each task at its maximum level of parallelism,

$$ft(S^*) \geq \sum_{i=1}^{n} t_i/\delta_i \,. \tag{1}$$

This bound corresponds to the case where all the tasks can begin execution simultaneously at time zero and is tight if and only if $\sum_{i=1}^{n} \delta_i \leq m$.

Using an argument based on the total amount of processing that is required by the tasks, we can form another lower bound on the flow time. Assume that the maximum level of parallelism for each task is m, that is it can be scheduled on all the processors. The flow time in this case will be minimized by a schedule S' which assigns m processors to each task and schedules them according to shortest processing time first (SPT). From this we can see that the finish time of the i-th task is the sum of the runtime of the first i tasks, i.e. $f_i(S') = \sum_{j=1}^{i} t_j/m$. Using this we form another lower bound on the flow time:

$$ft(S^*) \geq \sum_{i=1}^{n} f_i(S') = \sum_{i=1}^{n} (n-i+1)\frac{t_i}{m} \,. \tag{2}$$

This bound follows from the fact, that the flow time of this schedule, S' is less than the flow time of any schedule in which the limits on the parallelism for each task is less than m.

For notational convenience we define two variables to represent the above lower bounds. Let $A = \sum_{i=1}^{n} (n-i+1)t_i/m$, and let $H = \sum_{i=1}^{n} t_i/\delta_i$. Clearly, the flow time of the optimal schedule is greater than both the lower bounds, $ft(S^*) \geq \max(A, H)$ and since $\max(a,b) \geq \alpha a + (1-\alpha)b$, $0 \leq \alpha \leq 1$, the optimal flow time is greater than any weighted average of the two bounds.

$$ft(S^*) \geq \alpha A + (1-\alpha)H \qquad\qquad 0 \leq \alpha \leq 1. \tag{3}$$

We now introduce the Sorted Earliest Completion Time (SECT) heuristic. It is essentially the ECT algorithm presented in [11] after the tasks have been sorted in increasing order of work. We show that SECT when used for linear speedup task systems, produces schedules with worst case performance close to the optimal MFT. The motivation for sorting the task in increasing order of runtime is derived from the well known results that SPT produces optimal schedules for non-parallelizable tasks. Considering the special cases in which either all tasks are non-parallelizable, or in which every tasks has the maximum level of parallelism m, it is clear that ECT should schedule the shortest tasks first.

Algorithm 1 (Sorted Earliest Completion Time (SECT))

Input: A set of n tasks with runtimes $t_1, t_2, \ldots, t_n$ and maximum levels of parallelism $\delta_1, \delta_2, \ldots, \delta_n$.

Output: A schedule $SECT$ specifying for each task a start time s_i, and a level of parallelism $1 \leq \beta_i \leq \delta_i$.

Sort the tasks so that $t_1 \leq t_2 \leq \cdots \leq t_n$

for($i = 1$ **to** n)

$finish \leftarrow \infty$

for($p = 1$ **to** $\min(\delta_i, m/2)$)

$s_i \leftarrow$ first time when p processors are available

if $(s_i + t_i/p < finish)$ **then** $\{\ \beta_i \leftarrow p;\ finish \leftarrow s_i + t_i/\beta_i\ \}$

endfor

$s_i \leftarrow$ first time when β_i processors are available

Schedule T_i on β_i processors

endfor

Lemma 1 *Scheduling the set of linear speedup tasks according to the SECT algorithm achieves a flow time, $ft(SECT)$, satisfying*

$$ft(SECT) \leq 2A + H\,.$$

Proof: We first show that the finish time, f_i, for each task satisfies,

$$f_i \leq 2\sum_{j \leq i} t_j/m + t_i/\delta_i\ . \tag{4}$$

Considering the finish time of task T_i, assume that each of the previous tasks, T_j, $j < i$ satisfied the bound, and that they have been scheduled according to the SECT algorithm. This means that from time 0 until T_i is scheduled at least half of the processors are busy. We handle separately two cases.

For the case where $\delta_i \leq m/2$:

It follows that, $s_i(\delta_i)$, the start time of task T_i if it is scheduled on δ_i processors, satisfies

$$s_i(\delta_i) \leq 2\sum_{j<i} t_j/m$$

because at least half the processors are busy. Therefore the finish time, $f_i(\delta_i)$, if T_i is scheduled on δ_i processors satisfies $f_i(\delta_i) \leq 2\sum_{j \leq i} t_j/m + t_i/\delta_i$. Since we only schedule T_i on $\beta_i < \delta_i$ processors if it will decrease the finish time, it follows that $f_i(\beta_i) < f_i(\delta_i)$.

For the case where $\delta_i > m/2$:

Assume that the task T_i is scheduled on $m/2$ processors. It now follows that from the beginning of the schedule until T_i completes, at least half of the processors are being used, and therefore the finish time will satisfy

$$f_i(m/2) \leq 2\sum_{j \leq i} t_j/m \leq 2\sum_{j \leq i} t_j/m + t_i/\delta_i\,.$$

Once again the task is scheduled on $\beta_i \leq m/2$ processors only if this will decrease the finish time. **Note:** Using the arguments given in [11] it is possible to allow tasks to be scheduled on more than $m/2$ processors and still maintain the same bound on the finish time.

We have thus shown that the finish times of all tasks satisfy the bound in equation (4). Summing the finish times of all the tasks yields,

$$ft(SECT) = \sum_{i=1}^{n} f_i(\beta_i) \leq \sum_{i=1}^{n} \left[2 \sum_{j \leq i} t_j/m + t_i/\delta_i \right] \leq 2A + H \ .$$

■

Theorem 3 *The SECT algorithm obtains a bound of 3 to the optimum for linear speedup tasks in time* $O(n \log n + nm)$.

Proof: This follows directly from Lemma 1 and equation (3) when we choose $\alpha = 2/3$. i.e.,

$$ft_{SECT} \leq 3\,[2/3A + 1/3H] \leq 3ft(S^*)\,.$$

The complexity is easily verified to be $O(n \log n)$ to sort the tasks and $O(nm)$ to schedule the tasks. ■

5 Experimental Results

In the previous section we focused on analyzing the worst case performance of the SECT algorithm. In this section we present the average case behavior of this algorithm obtained by simulations. Our results show that the SECT algorithm in general produces schedules close to the optimum, with respect to the MFT.

To test the Sorted Earliest Completion Time (SECT) algorithm we ran experiments to simulate multiprocessors with 16, 32, 64 and 128 processors. We collected results from runs with various numbers of tasks between 4 and 128. In all cases the execution time of the tasks was chosen to be an exponentially distributed random variable with mean 1, and the maximum level of parallelism was a uniformly distributed random variable between 1 and the number of processors. For each given number of tasks, we ran the SECT algorithm on 100 different input sets. For each input set (trial) we calculated the ratio of the flow time of the SECT schedule to the lower bound on flow time of an optimal schedule. In Figure 5 we use box and whisker plots to represent the data. This method of exploratory data analysis is discussed in [8]. Although not as detailed as a plot of the distribution of ratios, it allows us to present 5 important values that summarizes the results. The vertical axis represents the ratio to the lower bound of optimum. Since we can never do better than the optimum the axis begins at 1.0, and because the SECT guarantees a worst case behavior of three, the axis extends to 3.0. For each number of tasks simulated for a given trial (4, 8, 16, 32, 64 and 128) we plot the minimum ratio and the maximum ratio, represented by the two lines (whiskers) extending from box, the average, represented by a large dot, and a box containing 50% of the points centered around the median. Figure 5 shows these points for simulations with 32 processors. The results from the simulation show that the SECT algorithm generally obtains a solution within a factor of 1.35 to the lower bound of the optimum, with the schedules on average being slightly better for trials with more tasks. In no trial did the flow time exceed 2 times the optimal, and thus we conjecture that a tighter bound can be shown for the SECT algorithm. The results for other numbers of processors were similar.

We plotted the results for SECT next to the results of running the classic non-parallel SPT algorithm. The up arrows in the graph of the SPT results represent data points that did not fit on the graph. It is clear from these results that SPT obtains results close to optimal only when $n \gg m$, and even in this situation SECT on the average produces schedules with smaller flow times.

In summary, we have shown that scheduling parallelizable tasks to minimize the mean flow time is NP-hard in the strong sense. We introduced a polynomial time heuristic, SECT, that produces schedules with a worst case performance bound of 3 to the optimum. Simulation show that in general the average case behavior is even closer to the optimum. We leave open the question whether a similar worst case bound may be achieved for *nonlinear* speedup functions.

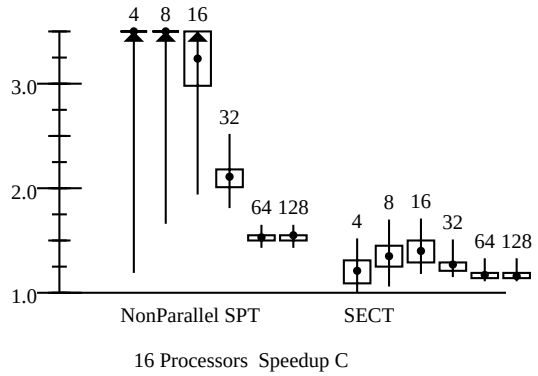


Figure 4: Average Case Performance of SECT algorithm for 16 processors

References

[1] J. Bruno, E .G. Coffman, and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17(7):382–387, July 1974.

[2] J. L. Bruno, E.G. Coffman, R. L. Graham, W. H. Kohler, R. Sethi, K. Steiglitz, and J.D. Ullman. *Computer and Job-Shop Scheduling Theory.* John Wiley & Sons, Inc., 1976.

[3] Richard W. Conway, William L. Maxwell, and Louis W. Miller. *Theory of Scheduling.* Addison-Wesley, Reading, Massachusetts, 1967.

[4] M. A. H. Dempster, J. K. Lenstra, and A. H. G. Rinnooy Kan, editors. *Deterministic and Stochastic Scheduling*, volume 84 of *C — Mathematical and Physical Sciences.* D. Reidel Publishing Company, 1982.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[6] E. Lelarasmee, A. E. Ruehli, and A. L. Sangiovanni-Vincentelli. The waveform relaxation method for time-domain analysis of large scale integrated circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, CAD-1(3):131–145, July 1982.

[7] Marc Snir. Vulcan and SP1—From a research prototype to a product. Personal Communication.

[8] J. W. Tukey. *Exploratory Data Analysis.* Addison-Wesley, 1977.

[9] J. Turek, U. Schwiegelshohn, J. L. Wolf, and P. S. Yu. Scheduling parallel tasks to minimize average response time. To appear in *Proc. of the 5th ACM-SIAM Symposium on Discrete Algorithms*, Arlington, Virgina, Jan. 23–25, 1994.

[10] J. Turek, J. Wolf, L. Fleischer, J. Glasgow, U. Schwiegelshohn, and P. Yu. On malleable response time scheduling. Technical report, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10598, November 1993.

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

[12] J. L. Wolf, J. Turek, M.-S. Chen, and P. S. Yu. The optimal scheduling of multiple queries in a parallel database machine. Technical Report RC 18595 (81362) 12/17/92, IBM Research Division, T.J. Waton Research Center, Yorktown Hieghts, NY 10598, December 1992.