A Semi-dynamic Multiprocessor Scheduling Algorithm with an Asymptotically Optimal Competitive Ratio

Satoshi Fujita*

Department of Information Engineering
Graduate School of Engineering, Hiroshima University
Higashi-Hiroshima, 739-8527, Japan

Abstract. In this paper, we consider the problem of assigning a set of n independent tasks onto a set of m identical processors in such a way that the overall execution time is minimized provided that the precise task execution times are not known a priori. In the following, we first provide a theoretical analysis of several conventional scheduling policies in terms of the worst case slowdown compared with the outcome of an optimal scheduling policy. It is shown that the best known algorithm in the literature achieves a worst case competitive ratio of $1+1/f(n)$ where $f(n) = O(n^{2/3})$ for any fixed m, that approaches to one by increasing n to the infinity. We then propose a new scheme that achieves a better worst case ratio of $1 + 1/g(n)$ where $g(n) = \Theta(n/\log n)$ for any fixed m, that approaches to one more quickly than the other schemes.

1 Introduction

In this paper, we consider the problem of assigning a set of n independent tasks onto a set of m identical processors in such a way that the overall execution time of the tasks will be minimized. It is widely accepted that, in the multiprocessor scheduling problem, both dynamic and static scheduling policies have their own advantages and disadvantages; for example, under dynamic policies, each task assignment incurs (non-negligible) overhead that is mainly due to communication, synchronization, and the manipulation of date structures, and under static policies, unpredictable faults and the delay of task executions will significantly degrade the performance of the scheduled parallel programs.

The basic idea of our proposed method is to adopt the notion of clustering in a "balanced" manner in terms of the worst case slowdown compared with the outcome of an optimal scheduling policy; i.e., we first partition the given set of independent tasks into several clusters, and apply static and dynamic schedulings to them in a mixed manner, in such a way that the worst case competitive ratio will be minimized. Note that this method is a generalization of two extremal cases in the sense that the case in which all tasks are contained in a single cluster

* This research was partially supported by the Ministry of Education, Culture, Sports, Science and Technology of Japan (# 13680417).

B. Monien and R. Feldmann (Eds.): Euro-Par 2002, LNCS 2400, pp. 240–247.
© Springer-Verlag Berlin Heidelberg 2002

corresponds to a static policy and the case in which each cluster contains exactly one task corresponds to a dynamic policy. In the following, we first provide a theoretical analysis of several scheduling policies proposed in the literature; it is shown that the best known algorithm in the literature achieves a worst case competitive ratio of $1 + 1/f(n)$ where $f(n) = O(n^{2/3})$ for any fixed m, that approaches to one by increasing n to the infinity. We then propose a new scheme that achieves a better worst case ratio of $1 + 1/g(n)$ where $g(n) = \Theta(n/\log n)$ for any fixed m, that approaches to one more quickly than the other schemes.

The remainder of this paper is organized as follows. In Section 2, we formally define the problem and the model. A formal definition of the competitive ratio, that is used as the measure of goodness of scheduling policies, will also be given. In Section 3, we derive upper and lower bounds on the competitive ratio for several conventional algorithms. In Section 4, we propose a new scheduling policy that achieves a better competitive ratio than conventional ones.

2 Preliminaries

2.1 Model

Let S be a set of n independent tasks, and $P = \{p_1, p_2, \ldots, p_m\}$ be a set of identical processors connected by a complete network. The execution time of a task u, denoted by $\tau(u)$, is a real satisfying $\alpha_u \leq \tau(u) \leq \beta_u$ for predetermined boundaries α_u and β_u, where the precise value of $\tau(u)$ can be known only when the execution of the task completes. Let $\alpha \stackrel{\text{def}}{=} \min_{u \in S} \alpha_u$ and $\beta \stackrel{\text{def}}{=} \max_{u \in S} \beta_u$. A **scheduling** of task u is a process that determines: 1) the processor on which the task is executed, and 2) the (immediate) predecessor of the task among those tasks assigned to the same processor[1]. Scheduling of a task can be conducted in either static or dynamic manner. In a static scheduling, each task can start its execution immediately after the completion of the predecessor task, although in a dynamic scheduling, each task execution incurs a scheduling overhead ϵ before starting, the value of which depends on the configuration of the system and the sizes of S and P.

A **scheduling policy** $\mathcal{A}$ for S is a collection of schedulings for all tasks in S. A scheduling policy $\mathcal{A}$ is said to be "static" if all schedulings in $\mathcal{A}$ are static, and is said to be "dynamic" if all schedulings in $\mathcal{A}$ are dynamic. A scheduling policy that is neither static nor dynamic will be referred to as a semi-dynamic policy.

In this paper, we measure the goodness of scheduling policies in terms of the worst case slowdown of the resultant schedule compared with the outcome of an optimal off-line algorithm, where term "off-line" means that it knows precise value of $\tau(u)$'s before conducting a scheduling; i.e., an off-line algorithm can generate an optimal static scheduling with overhead zero, although in order to

[1] Note that in the above definition, a scheduling does not fix the start time of each task; it is because we are considering cases in which the execution time of each task can change dynamically depending on the runtime environment.

obtain an optimal solution, it must solve the set partition problem that is well known to be NP-complete [1].

Let $\mathcal{A}(S, m, \tau)$ denote the length of a schedule generated by scheduling policy $\mathcal{A}$, which assigns tasks in S onto a set of m processors under an on-line selection τ of execution times for all $u \in S$. Let OPT denote an optimal off-line scheduling policy. Then, the (worst case) competitive ratio of $\mathcal{A}$ is defined as

$$r(\mathcal{A}, m, n) \stackrel{\text{def}}{=} \sup_{|S|=n,\tau} \frac{\mathcal{A}(S, m, \tau)}{OPT(S, m, \tau)}.$$

Note that by definition, $r(\mathcal{A}, m, n) \geq 1$ for any $\mathcal{A}$, $n \geq 1$, and $m \geq 2$. In the following, an asymptotic competitive ratio is also used, that is defined as follows:

$$r(\mathcal{A}, m) \stackrel{\text{def}}{=} \sup_{n \geq 1} r(\mathcal{A}, m, n).$$

2.2 Related Work

In the past two decades, several semi-dynamic scheduling algorithms have been proposed in the literature. Their main application is the parallelization of nested loops, and those semi-dynamic algorithms are commonly referred to as "chunk" scheduling schemes.

In the chunk self-scheduling policy (CSS, for short), a collection of tasks is divided into several chunks (clusters) with an equal size K, and those chunks are assigned to processors in a greedy manner [3] (note that an instance with $K = 1$ corresponds to a dynamic scheduling policy). CSS with chunk size K is often referred to as $CSS(K)$, and in [3], the goodness of $CSS(K)$ is theoretically analyzed under the assumption that the execution time of each task (i.e., an iteration of a loop) is an independent and identically distributed (i.i.d) random variable with an exponential distribution.

Polychronopoulos and Kuck proposed a more sophisticated scheduling policy called guided self-scheduling (GSS, for short) [4]. This policy is based on the intuition such that in an early stage of assignment, the size of each cluster can be larger than those used in later stages; i.e., the size of clusters can follow a decreasing sequence such as geometrically decreasing sequences. More concretely, in the i^{th} assignment, GSS assigns a cluster of size R_i/m to an idle processor, where R_i is the number of remaining loops at that time; e.g., R_1 is initialized to n, and R_2 is calculated as $R_1 - R_1/m = n(1 - 1/m)$. That is, under GSS, the cluster size geometrically decreases as

$$n/m,\ n/m(1-1/m),\ n/m(1-1/m)^2, \ldots.$$

Factoring scheduling proposed in [2] is an extension of GSS and CSS in the sense that a "part" of remaining loops is equally divided among the available processors; Hence, by using a parameter ℓ, that is a function of several parameters such as the mean execution time of a task and its deviation, the decreasing sequence of the cluster size is represented as $\underbrace{(n/m)\ell, \ldots, (n/m)\ell}_{m}, \underbrace{(n/m)\ell^2, \ldots, (n/m)\ell^2}_{m}, \ldots$.

Trapezoid self-scheduling (TSS, for short) proposed in [5] is another extension of GSS; in the scheme, the size of clusters decreases linearly instead of exponentially, and the sizes of maximum and minimum clusters can be specified as a part of the policy. (Note that since the total number of tasks is fixed to n, those two parameters completely define a decreasing sequence.) In [5], it is claimed that TSS is more practical than GSS in the sense that it does not require a complicated calculation for determining the size of the next cluster.

3 Analysis of Conventional Algorithms

This section gives an analysis of conventional algorithms described in the last section in terms of the competitive ratio.

3.1 Elementary Bounds

Recall that $\alpha \stackrel{\text{def}}{=} \min_{u \in S} \alpha_u$ and $\beta \stackrel{\text{def}}{=} \max_{u \in S} \beta_u$. The competitive ratio of any static and dynamic policies is bounded as in the following two lemmas (proofs are omitted in this extended abstract).

Lemma 1 (Static). *For any static policy $\mathcal{A}$ and for any $m \geq 2$, $r(\mathcal{A}, m) \geq 1 + \frac{\beta-\alpha}{\alpha+\beta/(m-1)}$, and the bound is tight in the sense that there is an instance that achieves it.*

Lemma 2 (Dynamic). *1) For any dynamic policy $\mathcal{A}$ and for any $m, n \geq 2$, $r(\mathcal{A}, m, n) \geq 1 + \epsilon/\alpha$, and 2) for any $2 \leq m \leq n$, there is a dynamic policy $\mathcal{A}^*$ such that $r(\mathcal{A}^*, m, n) \leq 1 + \frac{\epsilon}{\alpha} + \left(\frac{\beta+\epsilon}{\alpha}\right)\left(\frac{m}{n}\right)$.*

The goodness of chunk self-scheduling (CSS) in terms of the competitive ratio could be evaluated as follows.

Theorem 1 (CSS). *$r(CSS, m, n)$ is at least $1 + \left(\frac{2\sqrt{\beta\epsilon}}{\alpha}\right)\sqrt{\frac{m}{n}} + \frac{(\beta+\epsilon)m}{\alpha n}$, which is achieved when the cluster size is selected as $K = \left\lceil \sqrt{n\epsilon/m\beta} \right\rceil$.*

Since the largest cluster size in GSS is n/m, by using a similar argument to Lemma 1, we have the following theorem.

Corollary 1 (GSS). *$r(GSS, m, n) \geq 1 + \frac{\beta-\alpha}{\alpha+\frac{\beta}{m-1}}$.*

A similar claim holds for factoring method, since it does not take into account two boundaries α and β to determine parameter ℓ; i.e., for large β such that $\beta(n/m)\ell > \{n - (n/m)\ell\}\alpha$, we cannot give a good competitive ratio that approaches to one.

3.2 Clustering Based on Linearly Decreasing Sequence

Let Δ be a positive integer that is given as a parameter. Consider a sequence of integers $s_1, s_2, \ldots$, defined as follows: $s_i = s_1 - \Delta(i-1)$ for $i = 1, 2, \ldots$. Let k be an integer such that $\sum_{i=1}^{k-1} s_i < n \leq \sum_{i=1}^{k} s_i$. Trapezoid self-scheduling (TSS) is based on a sequence of k clusters $S_1, S_2, \ldots, S_k$, such that the sizes of the first $k-1$ clusters are $s_1, s_2, \ldots, s_{k-1}$, respectively, and that of the last cluster is $n - \sum_{i=1}^{k-1} s_i$. (A discussion for rational Δ's is complicated since it depends on the selection of m and n; hence we leave the analysis for rational Δ's as a future problem.)

In this subsection, we prove the following theorem.

Theorem 2. *$r(TSS, m, n) \geq 1 + 1/f(n)$ where $f(n) = O(n^{2/3})$ for fixed m.*

Proof. If $k \leq m$, then the same bound with Lemma 1 holds since in such cases, $|S_1| > n/m$ must hold. So, we can assume that $k > m$, without loss of generality. Let t be a non-negative integer satisfying the following inequalities:

$$(t+1)m \ < \ k \ \leq \ (t+2)m.$$

In the following, we consider the following three cases separately in this order; i.e., when t is an even greater than or equal to 2 (Case 1), when t is odd (Case 2), and when $t = 0$ (Case 3).

Case 1: For even $t \geq 2$, we may consider the following assignment τ of execution times to each task:

- if $|S_{tm+1}| \geq 2|S_{(t+1)m+1}|$ then the $(tm+1)$st cluster S_{tm+1} consists of tasks with execution time β, and the other clusters consist of tasks with execution time α, and
- if $|S_{tm+1}| < 2|S_{(t+1)m+1}|$ then the $(tm+m+1)$st cluster $S_{(t+1)m+1}$ consists of tasks with execution time β, and the other clusters consist of tasks with execution time α.

Since S contains at most $|S_{tm+1}|$ tasks with execution time β and all of the other tasks have execution time α, the schedule length of an optimal (off-line) algorithm is at most

$$OPT = \frac{n\alpha + (\beta - \alpha)|S_{tm+1}|}{m} + \beta \tag{1}$$

where the first term corresponds to the minimum completion time among m processors and the second term corresponds to the maximum difference of the completion times. On the other hand, for given τ, the length of a schedule generated by TSS is at least

$$TSS = \frac{n\alpha}{m} + t\epsilon + \frac{(\beta - \alpha)|S_{tm+1}|}{2} \tag{2}$$

where the first term corresponds to an optimal execution time of tasks provided that the execution time of each task is α, the second term corresponds to the

total overhead (per processor) incurred by the dynamic assignment of tasks, and the third term corresponds to the minimum difference of the completion times between the longest one and the others, provided that the execution time of tasks of one cluster (i.e., S_{tm+1} or S_{tm+m+1}) becomes β from α. Note that under TSS, clusters are assigned to m processors in such a way that all processors complete their $(2i)$th cluster simultanesously for each $1 \leq i \leq t/2$, and either S_{tm+1} or S_{tm+m+1} will be selected as a cluster consisting of longer tasks. Note also that by the rule of selection, at least $|S_{tm+1}|/2$ tasks contribute to the increase of the schedule length, according to the change of execution time from α to β.

Hence the ratio is at least

$$\begin{aligned} r(GSS, m, n) &= \frac{n\alpha/m + t\epsilon + (\beta - \alpha)|S_{tm+1}|/2}{n\alpha/m + (\beta - \alpha)|S_{tm+1}|/m + \beta - \alpha} \\ &= 1 + \frac{t\epsilon m + (\beta - \alpha)(|S_{tm+1}|m/2 - |S_{tm+1}| - m)}{n\alpha + (\beta - \alpha)(|S_{tm+1}| + m)} \\ &\geq 1 + \frac{\epsilon k - (\beta - \alpha + \epsilon)m + (\beta - \alpha)|S_{tm+1}|(m/2 - 1)}{n\alpha + (\beta - \alpha)(|S_{tm+1}| + m)} \end{aligned}$$

where the last inequality is due to $tm < k - m$.

Now consider the following sequence of clusters $S'_1, S'_2, \ldots, S'_k$ such that $|S'_i| = |S'_1| - \Delta'(i-1)$ for some Δ', $|S'_k| = 1$, and $\sum_{i=1}^{k} |S'_i| = n$. It is obvious that $|S_i| \geq |S'_i|$ for $k/2 \leq i \leq k$, and $tm + 1 \geq k/2$ holds since $t \geq 2$; i.e., $|S_{tm+1}| \geq |S'_{tm+1}|$. On the other hand, since $|S'_1| = 2n/k$ and $tm + 1 - k > m$, we can conclude that $|S_{tm+1}| \geq 2nm/k^2$. By substituing this inequality to the above formula, we have

$$\begin{aligned} r(TSS, m, n) &\geq 1 + \frac{\epsilon k - (\beta - \alpha + \epsilon)m + (\beta - \alpha)\frac{2nm}{k^2}(m/2 - 1)}{n\alpha + (\beta - \alpha)(|S_{tm+1}| + m)} \\ &= 1 + \frac{\epsilon k - (\beta - \alpha + \epsilon)m + (\beta - \alpha)\frac{2nm}{k^2}(m/2 - 1)}{\beta n + (\beta - \alpha)m}, \end{aligned}$$

where the right hand side takes a minimum value when $\epsilon k = (\beta - \alpha)\frac{2nm}{k^2}(m/2 - 1)$, i.e., when $k \simeq \sqrt[3]{\frac{(\beta - \alpha)nm}{\epsilon}}$. Hence by letting $k = \Theta(\sqrt[3]{n})$, we have $r(TSS, m, n) \geq 1 + 1/f(n)$ where $f(n) = O(n^{2/3})$ for any fixed m.

Case 2: For odd $t \geq 1$, we may consider the following assignment τ of execution times to each task:

- if $|S_{tm+1}| \geq 2|S_{(t+1)m+1}|\alpha/(\beta - \alpha)$ then the $(tm+1)$st cluster S_{tm+1} consists of tasks with execution time β, and the other clusters consist of tasks with execution time α.
- if $|S_{tm+1}| < 2|S_{(t+1)m+1}|\alpha/(\beta - \alpha)$ then the $(tm+m+1)$st cluster $S_{(t+1)m+1}$ consists of tasks with execution time β, and the other clusters consist of tasks with execution time α.

For such τ, an upper bound on the schedule length of an optimal (off-line) algorithm is given as in Equation (1), and the length of a schedule generated by

TSS can be represented in a similar form to Equation (2), where the last term should be replaced by

$$(\beta - \alpha)\,|S_{tm+1}| - \alpha|S_{tm+m+1}| \geq \frac{(\beta - \alpha)|S_{tm+1}|}{2}$$

when S_{tm+1} is selected, and by

$$(\beta - \alpha)|S_{tm+m+1}| > \frac{(\beta - \alpha)^2|S_{tm+1}|}{2\alpha}$$

when $S_{(t+1)m+1}$ is selected. Note that in both cases, a similar argument to Case 1 can be applied.

Case 3: When $t = 0$, we may use τ such that either S_1 or S_{m+1} is selected as a cluster with longer tasks as in Case 1, and for such τ, a similar argument to Lemma 1 holds. Q.E.D.

4 Proposed Method

In this section, we propose a new semi-dynamic scheduling policy that exhibits a better worst case performance than the other policies proposed in the literature. Our goal is to prove the following theorem.

Theorem 3. *There exists a semi-dynamic policy $\mathcal{A}$ such that $r(\mathcal{A}, m, n) = 1 + 1/g(n)$ where $g(n) = \Theta(n/\log n)$ for any fixed m.*

In order to clarify the explanation, we first consider the case of $m = 2$. Consider the following (monotonically decreasing) sequence of integers $s_0, s_1, s_2, \ldots$:

$$s_i \stackrel{\text{def}}{=} \begin{cases} n & \text{if } i = 0 \\ \left\lceil \left(\frac{\beta}{\alpha+\beta}\right) s_{i-1} \right\rceil & \text{if } i \geq 1. \end{cases}$$

Let k be the smallest integer satisfying $s_k \leq 2 + \frac{\beta}{\alpha}$. Note that such a k always exists, since $s_i \geq s_{i-1}$ for any $i \geq 1$, and if $s_{k'} = s_{k'-1}$ for some k', then $k' > k$ must hold (i.e., $s_1, s_2, \ldots, s_k$ is a strictly decreasing sequence). In fact, $s_{k'} = s_{k'-1}$ implies $\left(\frac{\beta}{\alpha+\beta}\right) s_{k'-1} > s_{k'-1} - 1$; i.e., $s_{k'-1} < 1 + \frac{\beta}{\alpha}$ $(< 2 + \frac{\beta}{\alpha})$.

By using a (finite) sequence $s_0, s_1, \ldots, s_k$, we define a partition of S, i.e., $\{S_1, S_2, \ldots, S_k, S_{k+1}\}$, as follows:

$$|S_i| \stackrel{\text{def}}{=} \begin{cases} s_{i-1} - s_i & \text{if } i = 1, 2, \ldots, k \text{ and} \\ s_k & \text{if } i = k + 1. \end{cases}$$

By the above definition, we have

$$\sum_{u \in S_i} \tau(u) \leq \sum_{v \in S_{i+1} \cup \ldots \cup S_k} \tau(v)$$

for any i and τ, provided that it holds $\alpha \leq \tau(u) \leq \beta$ for any $u \in S$. Hence, by assigning clusters $S_1, S_2, \ldots, S_{k+1}$ to processors in this order, we can bound the difference of completion times of two processors by at most $\beta|S_k| + \epsilon$; i.e., we can bound the competitive ratio as

$$r(\mathcal{A}, 2) \leq \frac{(X + (k+1)\epsilon)/2 + \beta|S_{k+1}| + \epsilon}{X/2} \;\leq\; 1 + \frac{k\epsilon + 2\beta|S_{k+1}| + 3\epsilon}{n\alpha} \quad (3)$$

Since we have known that $|S_{k+1}| \leq 2 + \frac{\beta}{\alpha}$, the proof for $m = 2$ completes by proving the following lemma.

Lemma 3.

$$k \leq \frac{\log_2 n}{\log_2(1 + \alpha/\beta)}.$$

Proof. Let a be a constant smaller than 1. Let $f_a(x) \stackrel{\text{def}}{=} \lceil ax \rceil$, and let us denote $f_a^i(x) \stackrel{\text{def}}{=} f_a(f_a^{i-1}(x))$, for convenience. Then, by a simple calculation, we have

$$f_a^i(x) \leq a^i \times x + a^{i-1} + a^{i-2} + \cdots + 1 \;\leq\; a^i \times x + \frac{1}{1-a}.$$

Hence, when $a = \left(\frac{\beta}{\alpha+\beta}\right)$ and $i = \log_{(1+\alpha/\beta)} n$, since $a^i = 1/n$, we have

$$f_a^i(n) \leq \frac{1}{n} \times n + \frac{1}{1 - \left(\frac{\beta}{\alpha+\beta}\right)} \;=\; 2 + \frac{\beta}{\alpha}$$

Hence, the lemma follows. Q.E.D.

We can extend the above idea to general m, as follows: Given sequence of clusters $S_1, S_2, \ldots, S_{k+1}$, we can define a sequence of $(k+1)m$ clusters by partitioning each cluster into m (sub)clusters equally (recall that this is a basic idea that is used in the factoring method). By using a similar argument to above, we can complete the proof of Theorem 3.

References

1. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide for the Theory of NP-Completeness.* Freeman, San Francisco, CA, 1979.
2. S. F. Hummel, E. Schonberg, and L. E. Flynn. Factoring, a method for scheduling parallel loops. *Communications of the ACM*, 35(8):90–101, August 1992.
3. C. P. Kruscal and A. Weiss. Allocationg independent subtasks on parallel processors. *IEEE Trans. Software Eng.*, SE-11(10):1001–1016, October 1985.
4. C. Polychronopoulos and D. Kuck. Guided self-scheduling: A practical self-scheduling scheme for parallel supercomputers. *IEEE Trans. Comput.*, C-36(12):1425–1439, December 1987.
5. T. H. Tzen and L. M. Ni. Trapezoid self-scheduling: A practical scheduling scheme for parallel compilers. *IEEE Trans. Parallel and Distributed Systems*, 4(1):87–98, January 1993.