Parallel Models and Job Characterization for System Scheduling*

X. Deng[1], H. Ip[1], K. Law[1], J. Li[2], W. Zheng[3], and S. Zhu[1]

[1] Department of Computer Science, City University of Hong Kong
Tat Chee Avenue, Kowloon, Hong Kong, P.R. China
{csdeng, cship, cskckl, cszhusf}@cityu.edu.hk

[2] Department of Mathematics, Yunnan University
Kunming 650091, P.R. China
jianping@public.km.yn.cn

[3] Department of Computer Science, Tsinghua University
Beijing 100084, P.R. China
zwm-dcs@tsinghua.edu.cn

Abstract

In this work, we study job characterization in multi-programmed multiprocessor system by taking into consideration the parallel job models. We first introduce an example to illustrate the issues involved. Then we focus on two popular system scheduling policies: round-robin for single processor systems, and equi-partition for multiprocessor systems. We analytically study effect of job parallelization on the overall performance of the system, and also present simulation results. Through these studies, we discuss scheduling policies of parallel jobs in relation to the ratio of number of jobs and the number of processors in the multiprocessor system, and propose new paradigm for parallel algorithm designs.

1 Introduction

In contrary to single processor computers, the multiprocessor system environment creates many complications in design and analysis of architecture and algorithms. Different types of parallel architecture resulted in architecture-dependent algorithm designs in the early stages of parallel computing (and still do.) Later, several parallel models (e.g., [13, 18, 2]) emerged that proposed architecture independent parallel computing, and hence made it possible to have a general purpose parallel computer environment. In a general purpose parallel computer system, users submit jobs and the system schedule their execution to make full use of the system's computing power. Even though users may have a good estimation of the processing time of their submitted jobs, it is often not be taken into consideration by the system scheduler for

*Research is partially supported by a CERG grant (CityU 1074/00E) of Hong Kong RGC and a research grant of City U of Hong Kong (7001215).

various reasons: e.g., user programs may contain bugs that defy users' estimation; users may lie to gain advantage according to scheduling policies; and the resulted scheduler may cause extra overhead if all job information is taken into consideration. In fact, recent studies on parallel computer systems have favored simple policies (e.g., equi-partition and dynamic equi-partition [15, 19, 9, 10, 6, 8]) that originated from the Round-Robin policy in single processor systems. For such policies, no specific job information is required. We should consider such schedule policies in our discussion. In particular, we view a sequential algorithm for a problem as generating a single process, and parallel algorithm for a problem as generating k processes (k is determined by the particular algorithm and no more than the number of processors). Each process is assigned with an equal processing power when severed by the system as in the Round-Robin policy.

Despite the fact that it may be difficult to collect, or utilize all job information in a general purpose parallel system, there is still the possibility that some simple characterization of parallel jobs can be exploited to improve system performance [14, 3, 1, 11, 12]. To one extreme, some studies considered utilizing all job information in parallelization of jobs [13, 16, 17]. To understand job characterization in multiprocessor system, one must look into the type of jobs are presented to such systems, i.e., to study parallel algorithms developed to attack problems that demand high performance computing. While design of parallel algorithms is often dependent on the parallel models that they are going to be applied, the measurement of speedup is common to most of the models. The theoretical model of PRAMs, for example, call a parallel algorithm to be optimal if the product of the execution time T_p and the number of processors P is of the same magnitude as the best sequential time T_1: $P \times T_p = O(T_1)$ (we should call the ratio $\frac{P \times T_p}{T_1)}$ the inefficiency.) In more recent models advocating architecture independent parallel algorithm designs by reducing communication cost ([18, 2, 5, 4]), extra works are often done to distribute data to be processed by individual processes (each executed in one single processor), with interleaved communication phases to coordinate and synchronize the processes.

While system scheduling calls for efficient utilization of computing resources, parallelism achieves speedup at the cost of introducing inefficiency (see, e.g., [7]). A comprehensive study of parallel system scheduling therefore should take into the consideration of the trade-off between the benefit of speedup and the cost of inefficiency when introducing parallelism for task processing. We focus on this factor of inefficiency in our study of scheduling policies. In particular, we are interested in decisions to execute a task with a parallel algorithm or a sequential algorithm in a general purpose parallel computer system environment, with n jobs and P processors. Very often, for the same parallel algorithm (may work for different number of processors), the inefficiency resulted from communication delays would be the same, no matter what is the number of processes the task is divided into. For example, the inefficiency that is resulted from communication delays (for the parallel system communication), the number of communication rounds (as advocated in architecture independent parallel algorithm designs), and the extra work necessary for efficient synchronization of separated computing processes are usually the same no matter how many processes are used, especially for new parallel algorithm design paradigms mentioned above. In design of algorithm for coarse grained multicomputers [4], for example, algorithms often require $O(\frac{T_1}{p})$ local computation time (the constant hidden in the big-O is independent of both T_1 and p). When the communication costs are sufficiently small (as a major requirement in design of coarse grained parallel algorithms), the inefficiency ratio is mainly determined by this constant in the local computation time, and thus independent of both T_1 and p.

We exploit the associated inefficiency for each job in the design of our scheduling policy. When inefficiency is bigger than one, it may not be helpful to execute jobs with parallel algorithms, especially when the number of jobs is relatively large. However, when the number of jobs is reduced to sufficiently small than the number of processors of the system, it may become necessary to execute these remaining jobs in parallel mode. However, we may have to start these jobs from the beginning since we may not know which are the jobs that we would like to execute and they may have been executed for some time in the system, under Round-Robin policy. This introduces a further inefficiency into the system performance. It is thus desirable to have jobs that could be execute in sequential mode first and then continue with parallel mode for the remaining part at any point in time. This poses a new challenge for parallel algorithm designs. We should not deal with the new challenge here but study the benefit of the existence of such algorithms.

Section 2 analyzes a special case of two jobs and three processors to illustrate the issues that might arise from our consideration. We consider the inefficiencies from generating two or three processes for each job and obtain analytic results. In Section 3, we study a general case of n jobs and p processors. We deal with the problem with a comparison of two policies: parallel processing and sequential processing. We aim at obtaining a policy that decides to introduce parallelism for job processing or to use sequential algorithms. In Section 4, we consider the more general case, where there are more jobs than threshold of introducing parallelism. In this situation, we consider the benefit of introducing parallelism, after many of the jobs finished execution, to the last few jobs remained uncompleted. Two models are considered: for one, only the remained portions are parallelized, and for another, jobs have to restart from the beginning in the parallel execution mode. The former is obviously more desirable but it would put harder requirement on parallel algorithm designs (and one that has not been noticed before to our best knowledge.) In Section 5, we conclude with discussion on the parallel algorithm design issues arisen from our study.

2 An Illustrative Example

In this section, we use our devised parallel algorithm to execute a task with two jobs $\mathcal{J}_1$, $\mathcal{J}_2$ on three identical parallel processors $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$. Here, two jobs $\mathcal{J}_1$, $\mathcal{J}_2$ require respectively x_1, x_2 units of unknown processing time, but we shall know their exact values at the final execution.

Our devised parallel algorithm is as follows: For any job $\mathcal{J}_i$, we consider the two possibilities to process it, i.e., either $\mathcal{J}_i$ is parallelized in average into two small jobs requiring $\frac{\beta_2}{2} \cdot x_i$ units of processing time or it is serialized, then each part (including parallelized ones) will be parallelized to process one unit of processing time per each time on one of three processors. We may always assume $\beta_2 \leq 2$ below.

By using our devised algorithm, we get a first follwoing result of total completion time involving in either parallelizing one or two jobs in average into small parts or serializing two jobs, whose proof can be found in the full version.

Theorem 1 Suppose three identical parallel processors $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$ and two jobs $\mathcal{J}_1$, $\mathcal{J}_2$ which require x_1, x_2 units of unknown processing time, respectively.

(i) If $1 \leq \beta_2 \leq \frac{6}{5}$, then we get the (determined) minimum total completion time by parallelizing each of two jobs $\mathcal{J}_1$, $\mathcal{J}_2$ in average into two small parts than any other ways;

(ii) If $\frac{6}{5} \leq \beta_2 \leq 2$ and it is assumed that two processing times x_1 and x_2 are both randomized variables with uniformly identical distribution, then the expected value of total completion time by parallelizing both jobs $\mathcal{J}_1$, $\mathcal{J}_2$ in average into two small parts is better than any other ways if $\frac{6}{5} < \beta_2 \leq \frac{18}{11}$ and the expected value of total completion time by only parallelizing one of two jobs in average into two small parts is better than any other ways if $\frac{18}{11} < \beta_2 \leq 2$.

In addition, we consider another possibility to process $\mathcal{J}_i$, i.e., $\mathcal{J}_i$ is parallelized in average into three small jobs requiring $\frac{\beta_3}{3} \cdot x_i$ units of processing time, where $\beta_3 \leq 3$. We can get following result whose proof can be found in the full version.
Theorem 2 Suppose three identical parallel processors $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$ and two jobs $\mathcal{J}_1$, $\mathcal{J}_2$ which require at random x_1, x_2 units of processing time, respectively. The three variables Φ, Ψ and Υ are defined above. Then

(i) If $1 \leq \beta_2 \leq \frac{18}{13}$ and $\beta_3 \leq \frac{11}{10}\beta_2$, the minimum expected value of completion time of two jobs is obtained by parallelizing each of two jobs in average into three small parts with $\frac{\beta_3}{3}x_1$, $\frac{\beta_3}{3}x_2$ units of processing time, respectively;

(ii) If $1 \leq \beta_2 \leq \frac{18}{13}$ and $\frac{11}{10}\beta_2 \leq \beta_3 \leq 3$, the minimum expected value of completion time of two jobs is obtained by parallelizing each of two jobs in average into *two* small parts with $\frac{\beta_2}{2}x_1$, $\frac{\beta_2}{2}x_2$ units of processing time, respectively;

(iii) If $\frac{18}{13} \leq \beta_2 \leq 2$ and $\beta_3 \leq \frac{9}{20}\beta_2 + \frac{1}{10}$, the minimum expected value of completion time of two jobs is obtained by parallelizing each of two jobs in average into three small parts with $\frac{\beta_3}{3}x_1$, $\frac{\beta_3}{3}x_2$ units of processing time, respectively;

(vi) If $\frac{18}{13} \leq \beta_2 \leq 2$ and $\frac{9}{20}\beta_2 + \frac{1}{10} \leq \beta_3 \leq 3$, the minimum expected value of completion time of two jobs is obtained by parallelizing only one of two jobs in average into two small parts with $\frac{\beta_2}{2}x_1$ (or $\frac{\beta_2}{2}x_2$) units of processing time, respectively.

3 Decision to Process Job in Parallel or in Sequent

In this section, suppose that there are p identical parallel processors $\mathcal{M}_1$, $\mathcal{M}_2$, ..., $\mathcal{M}_p$ and n jobs $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_n$. We shall execute these n jobs on p identical parallel processors by following devised algorithm: For any job $\mathcal{J}_i$, we consider the two possibilities to process it, i.e., either $\mathcal{J}_i$ is parallelized in average into p small parts requiring $\frac{\alpha_i}{p} \cdot x_i$ units of processing time or it is serialized, then each part (including parallelized one) will be parallelized to process one unit of processing time per each time on one of p processors, where $\alpha_i \geq 1$ is called an expansion coefficient of job $\mathcal{J}_i$. If α_i is smaller, it is better to parallelize job $\mathcal{J}_i$ in average into p small parts requiring $\frac{\alpha_i}{p} \cdot x_i$ units of processing time, otherwise it is serialized. Roughly, if α_i satisfies $\alpha_i \leq \frac{p}{n}$ for each job $\mathcal{J}_i$, then it is better to parallelize each job in average into p small parts such that the completion time of these n jobs is better than that of serializing jobs (see Theorem 4); otherwise, we can parallelize partial jobs in average into p small parts such that the completion time of these n jobs is better than that of serializing jobs (see next section).

Firstly, if each of n jobs is serialized, we can get the total completion time of these n jobs to process these n jobs on p identical parallel processors as follows, whose proof can be found in the full version of this paper.

Theorem 3 Suppose that we process n jobs $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_n$ on p identical parallel processors $\mathcal{M}_1$, $\mathcal{M}_2$, ..., $\mathcal{M}_p$ by assigning an unit of processing time of each job to some processor per each time (i.e., each job is processed on at most one processor per each time), where n jobs require x_1, x_2, ..., x_n units of unknown processing times, respectively. Then

(i) If $n \le p$, then the total completion time of these n jobs is $\sum_{i=1}^{n} x_i$;

(ii) If $p \le n$, then the total completion time of these n jobs is $\sum_{i=1}^{n-p} \frac{2n-2i+1}{p} x_{j_i} + \sum_{i=n-p+1}^{n} x_{j_i}$, where $x_{j_1} \le x_{j_2} \le \cdots \le x_{j_n}$. In particular, its total completion time is at most $\frac{n+p}{p} \sum_{i=1}^{n-p} x_{j_i} + \sum_{i=n-p+1}^{n} x_{j_i}$.

Secondly, if each of n jobs is parallelized into p small parts, we can get the total completion time of these n jobs as follows.

Theorem 4 Suppose that we parallelize each of n jobs $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_n$ in average into p small parts, each of which requires $\frac{\alpha_i}{p} \cdot x_i$ units of processing time, then assign each of which on p identical parallel processors $\mathcal{M}_1$, $\mathcal{M}_2$, ..., $\mathcal{M}_p$ by an unit per each time, where n jobs require x_1, x_2, ..., x_n units of unknown processing times, respectively. So the total completion time of these n jobs is as follows: $\frac{1}{p} \sum_{i=1}^{n} (2n - 2i + 1)\alpha_{j_i} x_{j_i} - \frac{n(n-1)}{2}$, where $\{j_1, j_2, \ldots, j_n\} = \{1, 2, \ldots, n\}$ satisfies $\alpha_{j_1} x_{j_1} \le \alpha_{j_2} x_{j_2} \le \cdots \le \alpha_{j_n} x_{j_n}$. In particular, the total completion time of these n jobs is at most $\frac{n}{p} \sum_{i=1}^{n} \alpha_i x_i - \frac{n(n-1)}{2}$.

Proof. Since each job $\mathcal{J}_{j_i}$ is parallelized in average into p small jobs, each of which requires $\frac{\alpha_{j_i}}{p} \cdot x_{j_i}$ units of unknown processing time, we can suppose that the completion time of job $\mathcal{J}_{j_i}$ starting at the end of job $\mathcal{J}_{j_{i-1}}$ to the completion of job $\mathcal{J}_{j_i}$ is t_i for each $i \in \{1, 2, \ldots, n\}$ (here we put $x_{j_0} = 0$ and $\alpha_{j_0} = 0$ for convenience), then we get $t_1 = (\frac{\alpha_{j_1} x_{j_1}}{p} - 1)n + 1$ and $t_i = \frac{\alpha_{j_i} x_{j_i} - \alpha_{j_{i-1}} x_{j_{i-1}}}{p} \cdot (n - i + 1) + 1$ if $2 \le i \le n$. So the total completion time of n jobs $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_n$ is as follows

$$\mathit{total} = \sum_{j=1}^{n} \sum_{i=1}^{j} t_i = \frac{1}{p} \sum_{i=1}^{n} (2n - 2i + 1)\alpha_{j_i} x_{j_i} - \frac{n(n-1)}{2}.$$

Especially, by the fact $\frac{\alpha_{j_1} x_{j_1}}{p} \le \frac{\alpha_{j_2} x_{j_2}}{p} \le \cdots \le \frac{\alpha_{j_n} x_{j_n}}{p}$, we easily get $\sum_{i=1}^{n} (2n - 2i + 1)\alpha_{j_i} x_{j_i} \le \sum_{i=1}^{n} n\alpha_{j_i} x_{j_i} = n \sum_{i=1}^{n} \alpha_i x_i$. So we get $\mathit{total} \le \frac{n}{p} \sum_{i=1}^{n} \alpha_i x_i - \frac{n(n-1)}{2}$, i.e., the total completion time is at most $\frac{n}{p} \sum_{i=1}^{n} \alpha_i x_i - \frac{n(n-1)}{2}$. □

For the case $\alpha_i = \alpha$ for each integer i, we get the following direct consequence as follows: the total completion time of these n jobs is $\frac{\alpha}{p} \sum_{i=1}^{n} (2n - 2i + 1) x_{j_i} - \frac{n(n-1)}{2}$, which is at most $\frac{n\alpha}{p} \sum_{i=1}^{n} x_i - \frac{n(n-1)}{2}$, where $x_{j_1} \le x_{j_2} \le \cdots \le x_{j_n}$.

Here, we give some remarks: (i). For the case $n \le p$ and $1 \le \alpha \le \frac{p}{n}$, we get $\mathit{total} < \sum_{i=1}^{n} x_i$, this means that it is better to parallelize each job in average into p small parts in this case such that the total completion times of these n jobs is smaller than that whose jobs are all serialized; (ii). In the preceding proof, we assume indeed that our devised parallel algorithm is processed on the order of $J_{j_1}, J_{j_2}, \ldots, J_{j_n}$, but when the devised parallel algorithm will be processed on any order of these n jobs, then we can get the total completion time of these n jobs is at most $\frac{1}{p}\sum_{i=1}^{n}(2n-2i+1)\alpha_{j_i}x_{j_i}$, this means that the absolute deviation of our devised parallel algorithm is at most $\frac{n(n-1)}{2}$.

As an experimental result shown in Figure 1, when we assign n jobs $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_n$ to satisfy the uniform distributions, it is given out the ratio of $\sum_{i=1}^{n} x_i$ divided by $\frac{\alpha}{p}\sum_{i=1}^{n}(2n-2i+1)x_{j_i} - \frac{n(n-1)}{2}$ at the some different values of α and the number of jobs n. It is easy to see that the total completion time of parallelizing each of these n jobs in average into p small parts is less than that of serializing these n jobs if the ratio is greater than one. In addition, when we also assign n jobs $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_n$ to satisfy the negative exponent distribution or Poisson distribution respectively, we shall obtain the similar figure as Figure 1.

4 The Benefit of Parallelizing Partial Jobs

In this section, suppose that we have p identical parallel processors $\mathcal{M}_1$, $\mathcal{M}_2$, ..., $\mathcal{M}_p$ and n jobs $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_n$, where $n \leq p$. Considering differently from parallelizing all jobs, we shall pay our attention to some possibilities of parallelizing partial jobs in average into p small parts. Roughly, by using the devised parallel algorithm in the proceeding section, we consider some possibilities of parallelizing partial jobs in average into p small parts in order to get the better total completion time than that of serializing these jobs. We consider two possibilities as follows: For some fixed integer s $(0 \leq s \leq n)$, at the end of execution of the s shortest jobs, we have got the left $n-s$ jobs and we can parallelize each left job in average into p small parts. The first method is to parallelize each left job in average into p small parts, each final part of which requires $\frac{\alpha}{p}\cdot(x_i - x_s + 1)$ units of unknown processing time (here $x_s \geq 1$) , and the second one is to do so with each final part requiring $\frac{\alpha}{p}\cdot x_i$ units of processing time, where the parallelized job $\mathcal{J}_i$ requires x_i units of unknown processing time. We hope to obtain the better integer s such that the final total completion time of n jobs is minimum.

Theorem 5 Suppose that we have p identical parallel processors $\mathcal{M}_1$, $\mathcal{M}_2$, ..., $\mathcal{M}_p$ and n jobs $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_n$ which require respectively x_1, x_2, ..., x_n units of unknown processing time on some processors, where $n \leq p$ and $x_1 \leq x_2 \leq \cdots \leq x_n$. Let s be a fixed integer satisfying $0 \leq s \leq n$.

(i) At the end of s jobs $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_s$ executed and $x_s \geq 1$, suppose that each left job $\mathcal{J}_j$ requiring $x_j - (x_s - 1)$ units of unknown processing time is parallelized in average into p small jobs, each of which requires $\frac{\alpha}{p}\cdot(x_j - x_s + 1)$ units of processing time, then the total completion time of these n jobs $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_n$ is

$$\sum_{i=1}^{s} x_i + (n - s - \frac{\alpha(n-s)^2}{p})x_s + \frac{\alpha}{p}\sum_{j=s+1}^{n}(2n-2j+1)x_j + f(n,s),$$

where $f(n,s) = \frac{\alpha(n-s)^2}{p} - \frac{(n-s)(n-s-1)}{2}$;

(ii) At the end of s jobs $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_s$ executed (putting $x_0 = 0$ for convenience), suppose that each left job $\mathcal{J}_j$ requiring x_j units of unknown processing time is parallelized in average into p small jobs, each of which requires $\frac{\alpha}{p} \cdot x_j$ units of unknown processing time, i.e., we restart to process the $n-s$ jobs left by parallelizing each in average into p small parts, then the total completion time of n jobs $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_n$ is

$$\sum_{i=1}^{s} x_i + (n-s)x_s + \frac{\alpha}{p} \sum_{j=s+1}^{n} (2n-2j+1)x_j - \frac{(n-s)(n-s-1)}{2}.$$

Proof. (i). We can get the total completion time of these n jobs in the two following stages:

Stage 1. At the end of the s executed jobs $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_s$, we get the total completion time of these s jobs as follows $total_1 = \sum_{i=1}^{s} x_i$.

Stage 2. At the end of the s executed jobs $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_s$ (here $x_s \geq 1$), we know that each left job $\mathcal{J}_j$ which requires $x_j - (x_s - 1)$ units of unknown processing time is parallelized in average into p small jobs, each of which requires $\frac{\alpha}{p} \cdot (x_j - x_s + 1)$ units of processing time. Put t_i to represent the completion time of job $\mathcal{J}_i$ starting at the end of job $\mathcal{J}_{i-1}$ to the completion of job $\mathcal{J}_i$ for each $i \in \{s+1, s+2, \ldots, n\}$ (note $t_s = x_s$ for convenience), by using similar arguments in the proof of preceding theorem, we easily get $t_{s+1} = (\frac{\alpha(x_{s+1}-x_s+1)}{p} - 1)(n-s) + 1$ and $t_j = \frac{\alpha(x_j - x_{j-1})}{p} \cdot p(n-j+1) + 1$ if $s+2 \leq i \leq n$. Then we get the total completion time of $\mathcal{J}_{s+1}$, $\mathcal{J}_{s+2}$, ..., $\mathcal{J}_n$ as follows $total_2 = \sum_{i=s+1}^{n} \sum_{j=s}^{i} t_j$.

So the total completion time of n jobs $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_n$ is as follows

$$\begin{aligned} total &= \sum_{i=1}^{s} x_i + \sum_{i=s+1}^{n} \sum_{j=s}^{i} t_j = \sum_{i=1}^{s} x_i + (n-s)t_s + \sum_{j=s+1}^{n} (n-j+1)t_j \\ &= \sum_{i=1}^{s} x_i + (n - s - \frac{\alpha(n-s)^2}{p})x_s + \frac{\alpha}{p} \sum_{i=s+1}^{n} (2n-2i+1)x_i \\ &\quad + \frac{\alpha(n-s)^2}{p} - \frac{(n-s)(n-s-1)}{2} \end{aligned}$$

(ii). With the similar arguments in (i), we only consider the following stage: at the end of s executed jobs $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_s$, each left job $\mathcal{J}_j$ which also requires x_j units of processing time is parallelized in average into p small jobs and restart to be processed, each of which requires $\frac{\alpha}{p} \cdot x_j$ units of processing time. Put t_i to represent the completion time of job $\mathcal{J}_i$ starting at the end of job $\mathcal{J}_{i-1}$ to the completion of job $\mathcal{J}_i$ for each $i \in \{s+1, s+2, \ldots, n\}$ (note $t_s = x_s$ for convenience), we get $t_{s+1} = (\frac{\alpha x_{s+1}}{p} - 1)(n-s) + 1$ and $t_j = \frac{\alpha(x_j - x_{j-1})}{p} \cdot (n-j+1) + 1$ if $s+2 \leq i \leq n$. Then we get the total completion time of $\mathcal{J}_{s+1}$, $\mathcal{J}_{s+2}$, ..., $\mathcal{J}_n$ as follows $total_2 = \sum_{i=s+1}^{n} \sum_{j=s}^{i} t_j$.

So the total completion time of n jobs $\mathcal{J}_1$, $\mathcal{J}_2$, ..., $\mathcal{J}_n$ is as follows

$$total = \sum_{i=1}^{s} x_i + (n-s)x_s + \frac{\alpha}{p} \sum_{i=s+1}^{n} (2n-2i+1)x_i - \frac{(n-s)(n-s-1)}{2}$$

□

We give some remark: In the proof of Theorem 5, we assume indeed that our devised parallel algorithm is processed on the order of $J_1, J_2, \ldots, J_n$, but when the devised parallel algorithm will be processed on any order of these n jobs, then we can get the total completion time of these n jobs is at most $\sum_{i=1}^{s} x_i + (n - s - \frac{\alpha(n-s)^2}{p})x_s + \frac{\alpha}{p}\sum_{j=s+1}^{n}(2n - 2j + 1)x_j$ in the first method (or $\sum_{i=1}^{s} x_i + (n - s)x_s + \frac{\alpha}{p}\sum_{j=s+1}^{n}(2n - 2j + 1)x_j$ in the second method, respectively), this means that the absolute deviation of our devised parallel algorithm is at most the absolute value of $\frac{\alpha(n-s)^2}{p} - \frac{(n-s)(n-s-1)}{2}$ in the first method (or $\frac{(n-s)(n-s-1)}{2}$ in the second method, respectively).

In the Figures 2 and 3, we give out some experimental results of the ratios of $\sum_{i=1}^{n} x_i$ divided by the total completion time of the two models stated in Theorem 5 when we assign n jobs $\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_n$ to satisfy the uniform distributions. It is shown that the first model that the remained portions are parallelized is better than the second model that restarts from the beginning in the parallel execution mode. On the other hand, it would be better not to parallelize these n jobs if the value of α is increased. In addition, when we also assign n jobs $\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_n$ to satisfy the negative exponent distribution or Poisson distribution respectively, we shall obtain the similar figures as Figures 2 and 3.

5 Remarks and Conclusion

In this work, we study the tradeoff of parallelism and efficiency in terms of minimizing the mean completion time of jobs for jobs with known inefficiency but unknown execution time. Our characterization of parallel jobs are based observation on parallel algorithms designs. Our results show that best performance would be achieved if jobs can be executed in single process mode first and then continue in parallel mode without creating much extra inefficiency. This may lead to interesting questions and new models for design of parallel algorithms.

We are particularly interesting to find parallel algorithms that allow for sequential execution with time $T_1 + o(T_1)$ or parallel execution with time $T_p + o(T_p)$ for problems with best known sequential algorithm of time T_1 and best known parallel algorithm of time T_p with p processors. Most interesting solutions would have sequential time τ_1 for duration in sequential mode and parallel time τ_p for duration in parallel mode such that the remaining portion of the sequential execution mode, $T_1 - \tau_1$ is as efficiently parallelized as the original problem. That is, $\frac{p\tau_p}{T_1-\tau_1}$ is roughly the same as $\frac{pT_p}{T_1}$.

References

[1] T. Brecht, and K. Guha, *Using Parallel Program Characteristics in Dynamic Processor Allocation Policies*, to appear in Performance Evaluation.

[2] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eicken, *LogP: Towards a Realistic Model of Parallel Computation*, Proceedings of Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Diego, May, 1993, pp. 1–12.

[3] S. Chiang, R. Mansharamani, and M. Vernon, *Use of Application Characteristics and Limited Preemption for Run-to-Completion Parallel Processor Scheduling Policies*, Proceedings of the 1994 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, 1994, pp. 33–44.

[4] F. Dehne, X. Deng, P. Dymond, A. Fabri, and A. Khokhar, *A Randomized Parallel 3D Convex Hull Algorithm for Coarse Grained Multicomputers*, Theory of Computing Systems, Vol. 30, 1997, pp. 547-558, a special issue for selected papers presented at the 7th ACM Symposium on Parallel Algorithms and Architectures, 1995, Santa Barbara.

[5] F. Dehne, A. Fabri, and A. Rau-Chaplin, *Scalable parallel computational geometry for coarse grained multicomputers*, International Journal on Computational Geometry, Vol. 6, No. 3, 1996, pp. 379 - 400.

[6] Xiaotie Deng, Nian Gu, Tim Brecht, KaiCheng Lu, *Preemptive Scheduling of Parallel Jobs on Multiprocessors*, SIAM J. Comput. 30, (2000), pp.145-160.

[7] D. Eager, J. Zahorjan, and E. Lazowska, *Speedup Versus Efficiency in Parallel Systems*, IEEE Trans. on Computers, Vol. 38., No. 3, (1989), pp. 408–423.

[8] Jeff Edmonds, Donald D. Chinn,Tim Brecht, X. Deng, *Non-clairvoyant Multiprocessor Scheduling of Jobs with Changing Execution Characteristics* , Proceedings of the 22nd ACM Symposium on Theory of Computing, May, 1997, pp. 120–129.

[9] R. Mansharamani and M. Vernon, *Qualitative Behavior of the EQS Parallel Processor Allocation Policy*, Technical Report CS TR 1192, Computer Sciences Department, University of Wisconsin, Madison, Madison, WI, November, 1993.

[10] R. Motwani, S. Phillips, and E. Torng *Non-Clairvoyant Scheduling*, Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms, Austin, Texas, January, 1993, pp. 422–431.

[11] T. Nguyen and R. Vaswani and J. Zahorjan, *Using Runtime Measured Workload Characteristics in Parallel Processor Scheduling*, Proceedings of the IPPS'96 Workshop on Job Scheduling Strategies for Parallel Processor Scheduling, Honolulu, HI, 1996, pp. 93–104.

[12] E. Parsons, and K. Sevcik, *Multiprocessor Scheduling for High-Variability Service Time Distributions*, Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, 949: edited by G. Feitelson and L. Rudolph, Springer-Verlag, 1995, pp. 127–145.

[13] C. Papadimitriou, and M. Yannakakis, *Towards an Architecture-Independent Analysis of Parallel Algorithms*, Proceedings of the 20th ACM Symposium on Theory of Computing, 1988, pp. 510–513.

[14] K. Sevcik, *Characterizations of Parallelism in Applications and their use in Scheduling*, Proceedings of the 1989 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, May, 1989, pp. 171–180.

[15] A. Tucker and A. Gupta, *Process Control and Scheduling Issues for Multiprogrammed Shared-Memory Multiprocessors*, Proceedings of the Twelfth ACM Symposium on Operating Systems Principles, 1989, pp. 159–166.

[16] J. Turek, W. Ludwig, J. L. Wolf, L. Fleischer, P. Tiwari, J. Glasgow, U. Schwiegelshohn, P. Yu, *Scheduling Parallelizable Tasks to Minimize Average Response Time*, Proceedings of the 6th Annual Symposium on Parallel Algorithms and Architectures, June, 1994, pp. 200–209.

[17] J. Turek, U. Schwiegelshohn, J. Wolf, P. Yu, *Scheduling Parallel Tasks to Minimize Average Response Time*, Proceedings of the 5th SIAM Symposium on Discrete Algorithms, 1994, pp. 112–121.

[18] L. Valiant, *A Bridging Model for Parallel Computation*, CACM , Vol. 33, No. 8, (1990), pp. 103–111.

[19] J. Zahorjan and C. McCann, *Processor Scheduling in Shared Memory Multiprocessors*, Proceedings of the 1990 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, Boulder, CO, May, 1990, pp. 214–225.