Scheduling Parallel Processors: Structural Properties and Optimal Policies*

Cheng-Shang Chang[†], Randolph Nelson[‡], and David D. Yao[§]

(December, 1994; Revision: April, 1995)

Abstract

Consider a set of parallel processors operating in a distributed fashion, which prohibits task migration among processors. A given set of jobs, each of which consists of a number of tasks, is to be processed by the parallel processors. Finding an optimal schedule in this context is a difficult combinatorial problem in general. Our focus here is on identifying key properties of the problem so as to provide insight to the structure of optimal policies. We demonstrate that these properties are all rooted in the *subadditivity, submodularity, convexity*, and *Schur convexity* of two operators, max and plus, which relate task times to the flow time (expected job completion time). We show that in general the optimal policy has a threshold structure and a sequential "tail": there exists a threshold, such that once the number of jobs already scheduled exceeds this threshold, all remaining jobs must be scheduled sequentially, i.e., each job with the entirety of its tasks is assigned to one processor only. In the special case of two processors, we further develop a recursive algorithm that generates the complete optimal schedule. For the case of three or more processors, we focus on a class of policies that we call "fully parallel or fully sequential" (FPFS), and identify optimal policies within this class.

Key Words: optimal scheduling, parallel processors, max-plus, threshold structure.

*This research was undertaken when Chang and Nelson were affiliated with and Yao an academic visitor at IBM Research Division, T.J. Watson Research Center, Yorktown Heights, New York.

[†]Department of Electrical Engineering, National Tsing Hua University, Hsin Chu 30043, Taiwan; cschang@ee.nthu.edu.tw.

[‡]OTA Limited Partnership, One Manhanttanville Rd, Purchase, NY 10577; rnelson@ox.com.

[§]IEOR Department, Columbia University, New York, NY 10027-6699; yao@ieor.columbia.edu. Supported in part by NSF Grants DDM-91-08540 and MSS-92-16490.

1 Introduction

We study the optimal scheduling of n jobs, each with a given number of tasks, on a set of parallel processors that operate in a distributed fashion, i.e., with no exchange of tasks among the processors. The task processing times are independent and identically distributed, and a job is completed only when all of its tasks are completed. The processors are all identical; in particular, they all have the same processing rate. The optimal schedule assigns the tasks to the processors, so as to minimize the sum of the expected completion times over all jobs, or the *flow time.*

In assuming that tasks, once scheduled on a processor, will be executed on that processor, our model is motivated by parallel processing systems that have non-uniform-access shared-memory or distributed memory. (Examples of such systems include BBN's butterfly machine, IBM's RP3 and Illinois' Cedar.) In such systems, access to non-local memory is expensive and usually tasks are not allowed to move across processors. Even if such access is not unduly expensive, one might still organize a system in this fashion, just to reduce scheduling complexity associated with dynamic task migration.

Executing jobs in parallel across multiple processors has the benefits of speeding up job execution time, at the price of incurring overheads associated with task synchronization and contention for shared resources such as memory and interconnection bandwidth, In contrast, sequential processing avoids some of the overheads, but does not offer the benefit of speedups associated with parallel execution. Clearly, however, if overheads increase sufficiently the benefits of parallel execution can decrease to the point where parallel execution ceases to offer any advantage over sequential processing. A main objective of our paper is hence to study the tradeoff between parallel and sequential executions, in particular, to investigate when parallel execution ceases to be beneficial.

As we shall demonstrate, in general the problem of finding the optimal scheduling of parallel processors as posed earlier is an extremely difficult combinatorial optimization problem. On the other hand, the problem has a very rich structure. This is largely due to the fact that the flow time has a simple representation that involves only two operators, max and plus (+): the latter cumulates the task times, while the former decides the completion of a whole job – when all tasks in the job are completed. We focus on exploiting four specific properties of these operators: *subadditivity, submodularity, convexity,* and *Schur convexity*, and show that these lead to the revelation of many interesting and deep properties of the structure of the optimal policies. Therefore, although we may not always be able to pin down exactly the optimal schedule, we know much about its qualitative structure.

In terms of related literature, the recent books in the area of discrete event systems, [1] and [2], as well as the numerous references there, present many deep results associated with the max-plus operators. In particular, [2] reveals that in a very general setting max-plus is a consequence of a special *antimatroid* structure. Implications of max-plus, on the other hand, include an adaptation of subadditive ergodic theory ([3], also see [4]), and the numerous applications of submodularity in monotone optimal control ([2], Chapter 6). Whereas our model here is more limited in the sense of lacking any dynamics, we exploit not only subadditivity and submodularity but also convexity and Schur convexity of max-plus. Other related works include [5] and [6], which also exploit the max-plus structure in a variety of applications, and [7, 8, 9], which study convexity and Schur convexity in a stochastic setting.

Our work also complements [10, 11], which study *individually* optimal policies, i.e., policies that allocate tasks of each arriving job on the processors so as to minimize the expected response time of the job. In contrast, here we focus on *socially* (or *globally*) optimal policies, which minimize the total expected response time of *all* jobs processed by the system. (Hence, a socially optimal policy might, for instance, sacrifice the performance of some jobs if so doing achieves an overall benefit.) Other related works that consider the effect of randomness on job synchronization include [12, 13, 14, 15, 16, 17].

It is important to note that the scheduling problem we study here differs quite significantly from the models in the scheduling literature (e.g., [8, 18] and the numerous references there). In traditional scheduling, there is no such feature as multiple tasks of a job that can be distributed over different processors; instead, each job in its entirety has to be scheduled on a single processor. That is, in our terminology, all jobs are scheduled sequentially; there is no notion of a parallel schedule. Therefore, the kind of schedules suitable for parallel processors must be fundamentally different from those in the scheduling literature. (This said, at the level of mathematical structure, the commonality between our problem and those in traditional scheduling is, in fact, quite substantial, as evident from comparing this study with [8].)

As pointed out earlier, finding the optimal schedule for parallel processors is in general a very hard combinatorial problem. Our results below, however, will support the general conclusion that the class of *fully parallel or fully sequential* (FPFS) policies will perform very well in general. Specifically, these policies first schedule each job by spreading it tasks evenly over the processors (i.e., fully parallel), until the number of jobs so scheduled has reached a threshold, and then the remaining jobs are all scheduled in a fully sequential fashion, i.e., each job with the entirety of its tasks is assigned to one processor only. Furthermore, the threshold value is in general increasing in the number of processors and decreasing in the variation of the task times. In other words, parallelism favors massiveness (large number of processors) and determinism (constant task times).

To support this general thesis, we first show (Theorem 3.3) that in general the optimal flow time can be bounded from both above and below, respectively by a fully sequential policy and by a fully parallel policy with deterministic task times; and the ratio between the upper and lower bounds decreases and approaches unity as the number of jobs grows, but increases as the number of processors grows. We further show (Theorem 5.3) that in general the optimal policy has a "sequential tail": there exists a threshold value, beyond which all jobs have to be scheduled in a fully sequential fashion. However, we do not know in general, how to optimally schedule those intial jobs that lead up to the threshold.

To close this gap, we proceed in two ways. First, we focus on the special case of two processors, for which we generate the complete optimal schedule (Theorem 6.3). Second, for the general case of three and more processors, we restrict the policies to the FPFS class, and find the optimal threshold value (Theorem 7.4).

The rest of the paper is organized as follows. We present a precise problem statement in Section 2, followed by listing the four key properties of max-plus in Lemma 2.1. In Section 3, we establish the upper and lower bounds on the optimal flow time mentioned above. In Section 4, we reveal the many basic properties of the optimal schedule, which eventually lead to, in Section 5, the threshold structure — the "sequential tail" mentioned earlier. In Section 6, we focus on the special case of two processors and present a recursive algorithm that generates the complete optimal schedule. For the more general case of three or more processors, we study in Section 7 the optimality of a more restricted class, the class of FPFS policies. Concluding remarks are summarized in Section 8.

An abridged predecessor of this paper, [19], was first presented at *Performance'93* in Rome, Italy. For completeness as well as for ease of reading, we have included here (as part of Sections 2 through 6) most of the materials in [19].

2 Model Description and Preliminary Properties

We consider a distributed parallel processing system consisting of P single-server queues. Each server corresponds to a processor, and the queue to local memory where the processor stores work. There are n jobs to be processed, all available at time zero. Job j is comprised of $b_j \geq 1$ tasks, $j = 1, \ldots, n$, where throughout the paper we assume that $1 \leq b_1 \leq b_2 \leq \cdots \leq b_n = b_{max}$. The tasks have iid (independent and identically distributed) processing times. Let the random variable X denote the generic task time, and assume $EX = 1$. In addition, let $S(m)$ denote the sum of m iid replicas of X, and by convention set $S(0) = 0$. Hence $S(m)$ is the time required to process m tasks on a single processor.

A *scheduling policy* (or simply, a *schedule* or a *policy*), denoted by ψ, is an assignment of the tasks of the n jobs onto the P processors. Let n_p^ψ denote the total number of tasks allocated to processor p under policy ψ. Note that

$$\sum_{p=1}^{P} n_p^\psi = \sum_{j=1}^{n} b_j, \tag{1}$$

independent of any policy ψ. Let $L_{p,j}^\psi$ denote the last position of job j's tasks that are scheduled by ψ on processor p; and set $L_{p,j}^\psi = 0$ if no task of job j is scheduled on processor p.

The *flow time*, i.e., the sum of expected completion times of all n jobs under policy ψ, denoted T^ψ, is then

$$T^\psi = \sum_{j=1}^{n} E \max_p \{S(L_{p,j}^\psi)\}. \tag{2}$$

Our objective is to find a policy that minimizes the above flow time, a policy we denote by π.

We next introduce some terminologies that will be used in the later sections. A *split* job is one whose tasks are scheduled on two or more processors; a *non-split* or *sequential* job is one whose tasks are scheduled contiguously on a single processor. A *fully sequential* policy is one that assigns the jobs, each in its entirety, to the processors in a round-robin fashion. Hence, all the jobs are non-split jobs. A *fully parallel* policy, on the other hand, assigns in a round-robin fashion the *tasks* (of each job, one job at a time) on the processors; hence every job is a split job. When we refer to the positions in the queue at each processor, "front" means closer to the processor (and hence the task there will be processed first). Finally, it would be helpful to keep in mind two simple facts: (i) it never pays to leave any empty position in any queue; (ii) reducing the job size, i.e., the number of tasks, always helps reducing the flow time.

Now, following the definition of $S(m)$ above, let $S'(m_1)$ and $S''(m_2)$ denote two *independent* random variables, which are summations of, respectively, m_1 and m_2 iid replicas of X. Define a function:

$$\alpha(m_1, m_2) \equiv E \max(S'(m_1), S''(m_2)).$$

The four inequalities listed below play a key role in establishing the various properties of optimal schedules.

Lemma 2.1 *For all non-negative integers m_1, m_2, c_1 and c_2:*
(i) (subadditivity) $\quad \alpha(m_1 + c_1, m_2 + c_2) \leq \alpha(m_1, m_2) + \alpha(c_1, c_2)$;
(ii) (submodularity) $\quad \alpha(m_1 + c_1, m_2 + c_2) + \alpha(m_1, m_2) \leq \alpha(m_1 + c_1, m_2) + \alpha(m_1, m_2 + c_2)$;
(iii) (convexity: Jensen) $\quad \alpha(m_1 + c, m_2 + c) \geq \alpha(m_1, m_2) + c$;
(iv) (Schur convexity: majorization) $\quad \alpha(m_1, m_2) \geq \alpha(\lceil (m_1 + m_2)/2 \rceil, \lfloor m_1 + m_2/2 \rfloor)$.

The first inequality follows from the subadditivity of max, and the second follows from submodularity. Refer to, e.g., [2], Chapters 6 and 7 for these properties. In particular, $f(x, y)$ is submodular, if for any $x_1 \leq x_2$ and $y_1 \leq y_2$,

$$f(x_1, y_1) + f(x_2, y_2) \leq f(x_1, y_2) + f(x_2, y_1).$$

The third inequality is a direct application of Jensen's inequality, making use of the convexity of max. The fourth follows from the Schur convexity of max, which preserves the majorization ordering; refer to [20]. (Roughly speaking, a Schur convex function becomes smaller when its components are more even in values, or more "balanced".) Note that all these properties are trivially satisfied by plus (+), a linear operator. Also note that all four inequalities hold for more than two variables.

Letting $m_1 = m_2 = m$ and $c = 0$ in Lemma 2.1 (iii) yields $\alpha(m, m) \geq m$. This special case can be strengthened as follows.

Lemma 2.2 *If the task time is* non-deterministic, *i.e.,* $P(X \neq EX) > 0$, *then there exists a constant* $\gamma > 0$ *such that*

$$\liminf_{m \to \infty} \frac{\alpha(m, m) - m}{\sqrt{m}} \geq \gamma.$$

Proof.

Let $\sigma^2 = E(X - EX)^2$ be the variance of X. Then $\sigma > 0$, since $P(X \neq EX) > 0$. First, suppose $\sigma < \infty$. Note that

$$\alpha(m, m) - m = E \max(S'(m), S''(m)) - ES''(m) = E(S'(m) - S''(m))^+, \tag{3}$$

where x^+ denotes $\max(0, x)$. Applying Markov's inequality yields

$$E(S'(m) - S''(m))^+ \geq xP((S'(m) - S''(m))^+ > x) = xP(S'(m) - S''(m) > x)$$

for all $x \geq 0$. Letting $x = \sqrt{2m}\sigma y$ for some $y > 0$, we have

$$E(S'(m) - S''(m))^+ \geq \sqrt{2m}\sigma y P(\frac{S'(m) - S''(m)}{\sigma\sqrt{2m}} > y). \tag{4}$$

The central limit theorem then leads to

$$\lim_{m\to\infty} P(\frac{S'(m) - S''(m)}{\sigma\sqrt{2m}} > y) = \int_y^\infty \frac{1}{\sqrt{2\pi}} e^{-u^2/2} du \equiv \bar{\Phi}(y).$$

This, in conjunction with (3) and (4), yields

$$\liminf_{m\to\infty} \frac{\alpha(m,m) - m}{\sqrt{m}} \geq \sqrt{2}\sigma y \bar{\Phi}(y) > 0.$$

Choosing $\gamma = \sqrt{2}\sigma y \bar{\Phi}(y)$ completes the argument for the case $\sigma < \infty$.

Next, consider the case $\sigma = \infty$. Let $\tilde{X} = \min(X, k)$, and let $\tilde{\sigma}^2$ be its variance. Choose k large enough so that $P(\tilde{X} \neq E\tilde{X}) > 0$. Thus, $0 < \tilde{\sigma} \leq k < \infty$. ¿From the result for the case $\sigma < \infty$, it suffices to show

$$E(S'(m) - S''(m))^+ \geq E(\tilde{S}'(m) - \tilde{S}''(m))^+ \tag{5}$$

where $\tilde{S}'(m)$ and $\tilde{S}''(m)$ are sums of m iid replicas of $\tilde{X}$. For any convex function ϕ, and any real x_1, x_2 and k, we have

$$\phi(\min(x_1, k) - \min(x_2, k)) + \phi(\min(x_2, k) - \min(x_1, k)) \leq \phi(x_1 - x_2) + \phi(x_2 - x_1).$$

Thus, replacing x_1 and x_2 by two iid random variables, X_1 and X_2, and taking expectations on both sides, we have

$$E\phi(\min(X_1, k) - \min(X_2, k)) \leq E\phi(X_1 - X_2). \tag{6}$$

Since $\phi(x) = [x]^+$ is a convex function, the inequality in (5) holds. ■

Lemma 2.3 *If the task time has a finite variance σ^2, then*

$$\frac{\alpha(m,m) - m}{\sqrt{m}} \leq \sigma\sqrt{2}.$$

Proof. Since

$$\alpha(m,m) - m = E(S'(m) - S''(m))^+, \tag{7}$$

applying Jensen's inequality yields

$$E(S'(m) - S''(m))^+ \leq (E((S'(m) - S''(m))^+)^2)^{1/2} \leq (E(S'(m) - S''(m))^2)^{1/2} = \sigma\sqrt{2m}. \tag{8}$$

■

Corollary 2.4 *If $P(X \neq EX) > 0$, then for any $b > 0$, $m^* = \inf\{m : \alpha(m,m) > m + b\} < \infty$. Moreover, for all $m \geq m^*$, $\alpha(m,m) > m + b$.*

Proof. That $m^* < \infty$ follows from Lemma 2.2. Also, we have from Jensen's inequality in Lemma 2.1 that

$$\alpha(m,m) \geq \alpha(m^*,m^*) + m - m^* > m^* + b + m - m^* = m + b,$$

for all $m \geq m^*$. ■

3 Bounds on Optimal Policies

We start with some simple facts on the fully sequential and fully parallel policies introduced at the end of the last section. We shall use the superscripts s and p to refer to these two policies.

Lemma 3.1 *Suppose there are $n = mP + r$ jobs, where $m \geq 1$ is an integer multiplier and $0 \leq r < P$. Then*

$$T^s = \sum_{\ell=1}^{m} \sum_{j=1}^{n-(\ell-1)P} b_j + \sum_{j=1}^{r} b_j.$$

Proof. First observe that under the fully sequential policy, the flow time is solely determined by the *mean* task time, $EX \equiv 1$. In other words, to derive T^s, we can view each task as requiring deterministically one unit of time.

Now, the policy schedules the n jobs on the P processors in a round-robin fashion. Suppose it schedules job j_1 first on processor 1, then job j_2 on processor 2, and so forth. (The job scheduled last is job j_n, on processor r.) Consider the P jobs that are scheduled last. Clearly, the sum of their completion times is $\sum_{i=1}^{n} b_{j_i} = \sum_{j=1}^{n} b_j$; [cf. (1)]. We can then "peel off" these P jobs from the schedule and apply the same argument to get the sum of the completion times of the next P jobs: $\sum_{i=1}^{n-P} b_{j_i}$. Continue in this fashion, the m the set of P jobs have a sum of completion times $\sum_{i=1}^{n-(m-1)P} b_{j_i}$. Finally, the remaining r jobs have a sum of completion times $\sum_{i=1}^{r} b_{j_i}$.

Since we want to minimize all of the above sums, the best sequence is the natural order, i.e., $j_i = i$, which minimizes each and every sum above. (Recall $b_1 \leq b_2 \leq \ldots \leq b_n$.) The desired result then follows. ■

Lemma 3.2 *When the task time is deterministic, $X \equiv 1$, the fully parallel policy is optimal, with flow time*

$$T^p = \sum_{j=1}^{n} \lceil \eta_j / P \rceil, \quad \text{where} \quad \eta_j \equiv \sum_{i=1}^{j} b_i.$$

Proof. The fully parallel policy assigns in a round-robin fashion the tasks of each job, one job at a time, on the processors. Suppose, as in the proof of Lemma 3.1, the job sequence is $(j_1, ...j_i, ...j_n)$. Then, clearly job j_i is completed when all $\eta_{j_i} \equiv \sum_{k=1}^{i} b_{j_k}$ tasks are completed. With P processors, this requires $\lceil \eta_{j_i} / P \rceil$ units of time. Hence, to minimize the flow time, the optimal sequence is again the natural order, $j_i = i$ for all $i = 1, ..., n$, same as in Lemma 3.1.

Finally, note that the minimum amount of time, for any schedule, to complete η_j tasks on P processors is $\lceil \eta_j / P \rceil$, hence the optimality of the fully parallel policy. ■

Theorem 3.3 *The flow time under the optimal schedule π of $n = mP + r$ jobs satisfies the following bounds:*

$$\sum_{\ell=1}^{n} \lceil \sum_{j=1}^{\ell} b_j / P \rceil \leq T^\pi \leq \sum_{\ell=1}^{m} \sum_{j=1}^{n-(\ell-1)P} b_j + \sum_{j=1}^{r} b_j.$$

Proof. First note that the upper bound is simply the flow time under the fully sequential policy (Lemma 3.1), hence it follows trivially from the optimality of π. For the lower bound, make use of Jensen's inequality: $E\phi(X) \geq \phi(EX)$, for any convex function ϕ. (Note that the flow time is a convex function; in particular, max is a convex function; cf. (2) and Lemma 2.1 (iii).) We have $T^\pi \geq T_1^\pi$, with T_1^π denoting the flow time of n jobs with deterministic, unit task time ($X \equiv 1$), under the same policy π. But when task times are deterministic, the fully parallel policy is optimal, hence the lower bound (cf. Lemma 3.2). ■

The ratio of the upper bound to the lower bound in Theorem 3.3 is very close to one in most cases. To see this, consider $n = mP$. The upper and lower bounds in Theorem 3.3 can be rewritten, with some algebra, as follows (since the upper and lower bounds correspond, respectively, to fully sequential and fully parallel policies, we use the superscripts s and p to denote them):

$$T^s = \sum_{\ell=0}^{m-1} \sum_{j=1}^{P} (m-\ell) b_{\ell P+j}, \quad T^p = T^s - \Delta,$$

where

$$\Delta \equiv \sum_{\ell=0}^{m-1} \sum_{j=1}^{P} \frac{j-1}{P} b_{\ell P+j}.$$

Hence, we have

$$\frac{\Delta}{T^s} \le \frac{b_{max}}{b_1} \frac{\sum_{\ell=0}^{m-1} \sum_{j=1}^{P} (j-1)/P}{\sum_{\ell=0}^{m-1} \sum_{j=1}^{P} (m-\ell)} \to 0, \qquad \text{as } m \to \infty,$$

provided $b_{\max} < \infty$. This implies $T^s/T^p \to 1$.

When the jobs have the same size, $b_j = b, j = 1, \ldots, n$, with $n = mP$, and $b = \ell P$, then the bounds in Theorem 3.3 specialize to

$$\frac{1}{2} n(n+1)\ell \le T^\pi \le \frac{1}{2} n(n+P)\ell,$$

and

$$\frac{T^s}{T^p} = 1 + \frac{P-1}{n+1}. \tag{9}$$

The above results suggest that the benefit of executing jobs on distributed processors becomes minimal when the number of jobs is large, in which case, the fully sequential policy has a flow time that is approximately equal to that of the optimal policy. On the other hand, (9) shows that the benefit of the optimal policy is more pronounced with a large number of processors.

4 Basic Structure of Optimal Policies

We summarize below the basic properties that must be satisfied by a class of optimal schedules: if any of these properties is violated, then an optimal schedule can be altered to satisfy these properties without increasing the flow time. Collectively these properties identify a class of schedules that are candidates for optimality. In other words, in the search for optimal schedules, it suffices to consider only schedules in this class.

Proposition 4.1 *The tasks of each job scheduled on the same processor must be adjacent.*

Proof. Suppose in a schedule there are two tasks of a given job (say j) that are not adjacent on a processor. We show it can be improved by having the two tasks adjacent to each other. Refer to these as i and i', with i positioned in the front (i.e., first to be processed). Move i back to be adjacent with (i.e., right before) i', and move all the other tasks in between i and i' up one position. (See Figure 1.) This will not delay the completion time of job j; also, it can only improve the completion time of any other job. ∎

In Section 3 we have seen that for the fully sequential and the fully parallel policies, the best order to schedule the jobs is to follow the natural order, $(1, 2, ..., n)$, i.e., in increasing order of the job size (recall $b_1 \leq b_2 \leq ... \leq b_n$). Under a general policy, the tasks of each job can be split on any number of processors (the tasks on each processor have to be adjacent though, following Proposition 4.1). However, a general policy would still schedule the jobs in a certain order, say j_1 followed by j_2 ... followed by j_n, denoted $(j_1, j_2, \ldots, j_n)$. That is, unless the policy schedules *all* tasks of job j_i before it schedules any task of job j_{i+1}, it can be improved upon. This is precisely stated in the next proposition. But first recall that $L^{\pi}_{p,j}$ denotes the last position of job j's tasks that are scheduled on processor p under a schedule π (Section 2). It plays a key role in evaluating the completion time of the job (on processor p).

Proposition 4.2 *There exists an ordering of the n jobs, $(j_1, \ldots, j_n)$, such that for any pair of jobs j_i and j_k, if j_i precedes j_k (in this ordering), then on any processor p, $L^{\pi}_{p,j_i} \leq L^{\pi}_{p,j_k}$, provided $L^{\pi}_{p,j_k} > 0$.*

Proof. By contradiction. Suppose the proposition is not true. Then it follows that there exists m, for $m = 2, 3, \ldots, P$, split jobs, say jobs $1, \ldots, m$, scheduled on m processors, say processors $1, \ldots, m$, so that $L^{\pi}_{1,1} < L^{\pi}_{1,2}, L^{\pi}_{2,2} < L^{\pi}_{2,3}, \ldots, L^{\pi}_{m-1,m-1} < L^{\pi}_{m-1,m}$ and $L^{\pi}_{m,m} < L^{\pi}_{m,1}$. We call this a *cycle* of length m. Suppose, without loss of generality, that $n_1 = \min_{i \in \{1,\ldots,m\}} n_i$, where n_i is the number of tasks of job i on processor i and define $n_{\min} \equiv n_1$. We can break this cycle without increasing the total flow time by performing a *circular shift operation.* This operation corresponds to a modified schedule where $n_{\min}$ tasks are moved from processor i to processor $i-1$ in a circular fashion (i.e., tasks are moved from processor 1 to processor m).

Specifically, we move all $n_{\min}$ tasks of job 1 scheduled on processor 1 to processor m in positions $L^{\pi}_{m,m}, L^{\pi}_{m,m}-1, \ldots, L^{\pi}_{m,m}-n_{\min}-1$. The $n_{\min}$ tasks of job m that are displaced by this move, are then moved to processor $m-1$ in positions $L^{\pi}_{m-1,m-1}, L^{\pi}_{m-1,m-1}-1, \ldots, L^{\pi}_{m-1,m-1}-n_{\min}-1$. The $n_{\min}$ tasks of job $m-1$ thus displaced are moved to processor $m-2$ in a similar manner as above. Continuing on in this fashion finally results in moving the $n_{\min}$ displaced tasks of job 2 initially scheduled on processor 2 to positions $L^{\pi}_{1,1}, L^{\pi}_{1,1}-1, \ldots, L^{\pi}_{1,1}-n_{\min}-1$ of processor 1. These tasks thus fill the vacancies left by the movement of the original $n_{\min}$ tasks of job 1.

After performing this circular shift is clear that job 1 is no longer scheduled on processor 1 and, furthermore, we claim that the completion time of job 1 is not increased. This follow from the fact that the tasks of job 1 have been moved to processor m in positions less than those of it's siblings that are already scheduled on this processor (the tasks that end at position $L^{\pi}_{m,1}$). Job i, for $i = 2, 3, \ldots, m$, on the other hand, is scheduled on processors i and $i-1$ in the modified schedule. To see that the completion time for job i is not increased by the shift observe that on processor $i-1$ the shifted tasks of job i are placed before those that are already scheduled on the processor, i.e., they are in positions less than $L^{\pi}_{i-1,i}$. The position of the last task of job i on processor i is not, however, increased by the shift operation. Hence the completion time of job i cannot increase.

The circular shift operation eliminates this cycle from the modified schedule without increasing the flow time. By repeating the argument for any cycles that still remain in the schedule we eventually are left with an optimal schedule that does not contain any cycles. The proposition is proved by observing that this final modified schedule has a flow time that is not greater than the original schedule. ■

Proposition 4.3 *The number of tasks scheduled on the processors satisfies*

$$\left| n^{\pi}_{p} - n^{\pi}_{p'} \right| \leq b_{max}, \quad 1 \leq p, p' \leq P.$$

Proof. Suppose for two processors $p \neq p'$, $n^{\psi}_{p} - n^{\psi}_{p'} > b_{max}$. Let j be the last job on processor p. Move all its tasks (the number of which is at most b_{max}) to the end of the queue on processor p'. This will not affect the completion time of any other job, and will only improve j's completion time. Continuing in this fashion, we can bring the difference in the number of tasks scheduled on the P processors to below the job size b_{max}. ■

Consider the case where $b_j = b$, $j = 1, \ldots, n$. ¿From the following relations,

$$P \min_p \{n_p^\pi\} - (P-1)b \leq nb = \sum_{p=1}^{P} n_p^\pi \leq P \max_p \{n_p^\pi\} + (P-1)b,$$

along with Proposition 4.3, we have

Corollary 4.4 *If $b_j = b$, $j = 1, \ldots, n$, then the number of tasks assigned to any processor p, $p = 1, \ldots, P$, satisfies*

$$\frac{(n-P+1)b}{P} \leq n_p^\pi \leq \frac{(n+P-1)b}{P}.$$

The next result, Theorem 4.5 below, establishes a "subset" property: If an optimal policy, for $n+P$ jobs (P being the total number of processors), schedules the last P jobs sequentially, one on each processor, then what remains after "peeling off" these last P jobs is an optimal schedule for the remaining n jobs.

Theorem 4.5 *Suppose the optimal policy π, for $n+P$ jobs, schedules the last P jobs sequentially, one for each processor, then it also schedules the remaining n jobs optimally. Furthermore, the last P sequential jobs are necessarily the largest ones (in terms of job size) among the $n+P$ jobs.*

Proof. It is clear that the sum of the last P sequential jobs' expected completion times is equal to [cf. (1)]

$$\sum_{p=1}^{P} n_p^\pi = \sum_{j=1}^{n+P} b_j,$$

and this sum is independent of the schedule and the sizes of the initial n jobs. Hence, we can write the flow time under π (for $n+P$ jobs) as follows

$$T^\pi(n+P) = T^{\pi'}(n) + \sum_{j=1}^{n+P} b_j,$$

where $T^{\pi'}(n)$ denotes the flow time of the other n jobs under π, in particular, π' denotes the way π schedules its initial n jobs. Hence, for π to be optimal, π' must be optimal in scheduling the initial n jobs, which also have to be those with the smallest sizes since reducing job size cannot increase the flow time. ■

5 The Asymptotic Sequential Property

Here we assume that all jobs have the same size, $b_j = b, j = 1, \ldots, n$. In this case we show in Theorem 5.3 that provided the number of jobs is sufficiently large, the optimal policy will eventually schedules jobs sequentially. A threshold value is also explicitly given in Theorem 5.3 such that all the jobs that exceed the threshold should all be processed sequentially. To start with, we need two lemmas.

When all jobs have the same size, they are no longer distinguishable. Although the ordering result in Proposition 4.2 still applies, i.e., one would still exhaust all the tasks of one job before scheduling the next job, its interpretation becomes more subtle. We will use the following *labelling* scheme. Call a job *terminal* if there are no other job's tasks scheduled behind its tasks on any processor. (Note this notion is well defined under Proposition 4.2.) More precisely, j^t is a terminal job if $L^{\psi}_{p,j^t} \geq L^{\psi}_{p,j}$, for any $j \neq j^t$ and $L^{\psi}_{p,j} > 0$ (break ties arbitrarily). Among all the terminal jobs, pick the one with the largest $\max_p L^{\psi}_{p,j^t}$ value, termed the *last* terminal job, and label it n (given a total of n jobs). In view of Proposition 4.2, we can "peel off" all the tasks of this last terminal job from the schedule without affecting any other job. Using the same argument, we can then identify another last terminal job among the remaining jobs and label it $n-1$. In this way all n jobs are labeled.

Lemma 5.1 *Suppose in an optimal policy π, the last k jobs (i.e., those with labels $n, n-1, n-k+1$), for $k = 2, \ldots, P$, are non-split (i.e., sequential) jobs. Then no processor is assigned more than one of these k jobs.*

Proof. Suppose, without loss of generality, that the last k jobs are sequentially scheduled on processors $p_1, \ldots, p_m$, with $m < k$, and hence suppose p_1 has two of these k sequential jobs, say jobs j' and j'', at the end. We will show that either the last sequential job j'' can be moved to the other processors p_j, $j \geq m+1$, or a contradiction will be reached.

Suppose the end jobs on processors $p_j, j \geq m+1$, have labels $j_1, \ldots, j_s$, for some s. Note these labels are all smaller than $n-k+1$. First, we argue that the jobs $j_1, \ldots, j_s$, cannot have any task scheduled on processors $p_1, \ldots, p_m$. For suppose job j_1 is the last job on processor p_{m+1} and it has k tasks on processor p_m, say. Now, move these k tasks to the end of processor p_{m+1}. The cost reduction for the last sequential job on p_m is k. On the other hand, the cost increase of job j_1 is at most k, following the subadditive inequality in Lemma 2.1.

Therefore, following Proposition 4.2, there must exist at least one *terminal* job among jobs $j_1, \ldots, j_s$. Without loss of generality, we assume j_1 is a terminal job on p_{m+1}. If $n^{\pi}_{p_{m+1}} \leq n^{\pi}_{p_1} - b$, we can move the last sequential job j'' on p_1 to p_{m+1} without increasing the overall flow time. If $n^\pi_{p_{m+1}} > n^\pi_{p_1} - b$, then the terminal job j_1 on p_{m+1} must have a larger label than the job j' on p_1, which is a contradiction. ■

Lemma 5.2 *Let*

$$m^* = \inf\{m : \alpha(m,m) > m + b - 1\}. \tag{10}$$

If $(n-P+1)b/P \geq m^*$, then the optimal policy π (for n jobs) schedules the last P jobs sequentially: one for each processor. More generally, the optimal policy schedules the last $(\ell+1)P$ jobs sequentially: in a round-robin fashion among the P processors, where

$$\ell = \lfloor \frac{1}{b}(\frac{(n-P+1)b}{P} - m^*) \rfloor. \tag{11}$$

Proof. Consider the last job under schedule $\pi(n)$, i.e., the job with label n. Suppose its tasks are scheduled on more than one processor and, without loss of generality, that there are scheduled tasks from job n on processors p_1 and p_2 and furthermore that $n^\pi_{p_1} \leq n^\pi_{p_2}$. If we move all n's other tasks to the end of p_1, we will have another policy, which differs from π only in the assignment of tasks of this terminal job. Hence, we must have

$$E \max\{S^1(n^\pi_{p_1}), S^2(n^\pi_{p_2})\} \leq n^\pi_{p_1} + (b-1),$$

where the left hand side is a lower bound on the completion time of the terminal job under π, while the right hand side is the upper bound on the completion time of the terminal job under the new policy. This implies, given $n^\pi_{p_1} \leq n^\pi_{p_2}$,

$$\alpha(n^\pi_{p_1}, n^\pi_{p_1}) = E \max\{S^1(n^\pi_{p_1}), S^2(n^\pi_{p_1})\} \leq n^\pi_{p_1} + (b-1).$$

But this violates $\alpha(m,m) > m+b-1$ for all $m \geq n^\pi_{p_1}$ (see Corollary 2.4), since $(n-P+1)b/P \leq n^\pi_{p_1}$, following Corollary 4.4. Therefore, job n must be sequentially scheduled. The above argument also applies to all the other jobs with labels $n-1, \ldots, n-P+1$.

For the generalization part, remove the last P jobs, then Theorem 4.5 guarantees what remains is still optimal. Hence, the problem reduces to one with $n-P$ jobs. Apply the above argument repeatedly until $(n-P+1)b/P \geq m^*$ is violated. ■

The above lemma leads immediately to the following

Theorem 5.3 *There exists an optimal policy π that sequentially schedules all jobs with labels greater than or equal to a threshold value $n^* = (m^*/b+1)P-1$, where m^* is defined in Lemma 5.2.*

Intuitively, it is conceivable that as task times become less variable the threshold n^* should increase, i.e., more jobs will be scheduled in parallel, and this is indeed what one observes in practice. To quantify this behavior, consider tasks times that follow a k-stage Erlang distribution with unit mean. Note the variance of task times is given by $1/k$. Below let $\alpha_k(m,m)$ denote the α value under Erlang-k task times. We have

$$\alpha_k(m,m) = \frac{\alpha_1(km,km)}{k}. \tag{12}$$

and

$$\alpha_1(m_1,m_2) = \frac{1}{2} + \frac{1}{2}\alpha_1(m_1-1,m_2) + \frac{1}{2}\alpha_1(m_1,m_2-1), \tag{13}$$

for $m_1 > 0$ and $m_2 > 0$, with the boundary conditions: $\alpha(m,0) = \alpha(0,m) = m$ for all m. To solve the recursion, we consider a two-dimensional random walk on the non-negative integer lattice. Let $N(m_1,m_2)$ be the hitting time of the random walk at (0,0), starting from (m_1,m_2). Then, we have

$$\alpha_1(m_1,m_2) = \frac{1}{2}EN(m_1,m_2) \tag{14}$$

since each step takes $1/2$ unit of time on average. With this interpretation, we can derive

$$EN(m+1,m+1) - EN(m,m) = 2 + \frac{2m!}{m!m!}(\frac{1}{2})^{2m}, \tag{15}$$

which, in conjunction with (14), leads to

$$\alpha_1(m+1,m+1) = \alpha_1(m,m) + 1 + (\frac{1}{2})^{2m+1}\frac{2m!}{m!m!}. \tag{16}$$

Applying Stirling's formula to (16), we have, for large m,

$$\alpha_1(m,m) \approx m + \sum_{j=0}^{m-1}\frac{1}{2}\frac{1}{\sqrt{\pi j}} \approx m + \frac{1}{\sqrt{\pi}}\sqrt{m}. \tag{17}$$

Using the above in (10) and solving for m, we have

$$m^* \approx k\pi(b-1)^2,$$

and hence, following Theorem 5.3,

$$n^* \approx P[\frac{k\pi(b-1)^2}{b}+1]-1. \tag{18}$$

That is, n^* increases linearly in k (i.e., inversely proportional to the variance) and roughly linearly in b and linearly in P as well. This supports the intuition that as a general rule less variability favors parallel processing. The Stirling approximation involved in estimating n^* above works very well. For instance, we have checked (18) for values of $k \leq 10$ and $b \leq 10$, and found that it is remarkably accurate.

6 Optimal Policy for Two Processors

In view of Theorem 5.3, we know how to optimally schedule the jobs after a threshold value n^*. However, any further characterization on the optimal schedule of the initial jobs whose labels are less than the threshold appears to be extremely difficult. In this section we consider the special case of $P = 2$, for which we can establish a complete characterization of the optimal policy. In addition, here we can allow the job sizes to be different.

Proposition 6.1 *Under the optimal policy π, the number of non-split jobs scheduled after the last split job (if any) must be the same for both processors.*

Proof. Let j denote the last split job (if any). Let m_1 and m_2 be the number of split jobs after job j on processor 1 and 2 respectively. Suppose $m_1 > m_2$; we will contradict the optimality of π. Suppose there are $n_{1,j}$ tasks of job j are scheduled on processor 1. Move these tasks to processor 2, and let them join the other tasks of job j. The net increase in j's completion time is $L^\pi_{2,j} + n_{1,j} - \alpha(L^\pi_{1,j}, L^\pi_{2,j})$. On the other hand, the completion times of the m_1 and m_2 sequential jobs that follow j have a net increase of $(m_2 - m_1)n_{1,j}$ units. Since $\alpha(L^\pi_{1,j}, L^\pi_{2,j}) \geq L^\pi_{2,j}$ and $m_1 > m_2$, we have

$$L^\pi_{2,j} + n_{1,j} - \alpha(L^\pi_{1,j}, L^\pi_{2,j}) + (m_2 - m_1)n_{1,j} \leq 0.$$

That is, the changes result in an improvement, contradicting the optimality of π. ■

Next, we present an algorithm that recursively carries out the optimal schedule. Let π_i denote an optimal policy for jobs $\{1, \ldots, i\}$, $i = 1, \ldots, n$ and recall that $b_1 \leq b_2 \ldots \leq b_n$. By definition we set $T^{\pi_0} \equiv 0$. We first present a lower bound for the optimal flow time.

Lemma 6.2 *For $P = 2$, the optimal flow time satisfies the following recursion:*

$$T^{\pi_n} \geq \min\{T^{\pi_{n-1}} + \beta_n, T^{\pi_{n-2}} + \eta_n\}, \quad n \geq 2. \tag{19}$$

where

$$\beta_n \equiv \alpha(\lceil \sum_{j=1}^{n} b_j/2 \rceil, \lfloor \sum_{j=1}^{n} b_j/2 \rfloor), \quad \eta_n \equiv \sum_{j=1}^{n} b_j.$$

Proof. Note that in the two processor case, either (i) the last job is a non-split job, in which case, following Proposition 6.1, the last job on the other processor must also be a non-split job, or (ii) the last job is a split job with tasks on both processors.

In case (i), the sum of the completion times of the last two sequential jobs is equal to η_n, independent of the schedule of the other $n-1$ jobs. Hence, for any policy ψ that schedules the n jobs, we have,

$$T^{\psi} \geq T^{\pi_{n-2}} + \eta_n,$$

since the right hand side is the optimal schedule in case (i).

In case (ii), applying the majorization inequality in Lemma 2.1, we know the completion time of the last job cannot be shorter than β_n. In addition, no schedule for $n-1$ jobs has a smaller flow time than π_{n-1}. Hence,

$$T^{\psi} \geq T^{\pi_{n-1}} + \beta_n. \tag{20}$$

(Note, we are *not* claiming here that the right hand side above corresponds to a feasible schedule for n jobs: such a claim is only needed in Theorem 6.3 below.) Combining the above, we have

$$T^{\pi_n} \geq \min\{T^{\pi_{n-1}} + \beta_n, T^{\pi_{n-2}} + \eta_n\}, \quad n \geq 2.$$

■

In the next theorem we show that the inequality in (19) actually holds as an equality, by arguing that $T^{\pi_{n-1}} + \beta_n$ of (20) does correspond to a feasible schedule of n jobs. We do this by developing an algorithm that recursively constructs the optimal policy.

Theorem 6.3 *The following algorithm constructs an optimal policy $\pi \equiv \pi_n$:*

1. *π_1 assigns $\lceil b_1/2 \rceil$ tasks on processor 1 and $\lfloor b_1/2 \rfloor$ tasks on processor 2.*

2. *π_i, $i = 2, \ldots, n$, is constructed recursively as follows:*

If $T^{\pi_{i-1}} + \beta_i < T^{\pi_{i-2}} + \eta_i$, then π_i follows π_{i-1} in scheduling jobs $1, \ldots, i-1$, while splits the b_i tasks of job i on both processors, so that each processor will have a total of $\lceil \sum_{j=1}^{i} b_j/2 \rceil$ and $\lfloor \sum_{j=1}^{i} b_j/2 \rfloor$ tasks, respectively. In this case, $|n_1^{\pi} - n_2^{\pi}| \leq 1$.

Otherwise π_i follows π_{i-2} in scheduling jobs $1,\ldots,i-2$, and schedules job $i-1$ sequentially on the processor that has a smaller number of tasks and job i on the other processor. In this case, $|n_1^\pi - n_2^\pi| \leq \max_{j\leq i}(b_j - b_{j-1})$.

Furthermore the optimal policy satisfies

$$T^{\pi_n} = \min\{T^{\pi_{n-1}} + \beta_n, T^{\pi_{n-2}} + \eta_n\}, \quad n \geq 2; \quad T^{\pi_1} = \beta_1. \tag{21}$$

Proof. We use induction on i. The statement about $i = 1$ is trivially true. Suppose the construction has been carried out to π_{i-1}; consider the construction of π_i. Let $d_i \equiv |n_1^{\pi_i} - n_2^{\pi_i}|$. In the first case, since following the induction hypothesis,

$$d_{i-1} \leq \max_{j\leq i-1}(b_j - b_{j-1}) < \max_{j\leq i-1} b_j \leq b_{i-1},$$

and since $b_{i-1} \leq b_i$, the b_i tasks of job i can be split on the two processors to achieve the desired balance in the number of tasks. Hence, $d_i \leq 1$. In the second case, following the specified task assignment and the induction hypothesis, we have

$$d_i = |d_{i-2} + b_{i-1} - b_i| \leq \max\{d_{i-2}, b_i - b_{i-1}\} \leq \max_{j\leq i}(b_j - b_{j-1}).$$

Therefore, in both cases the construction is feasible. In particular, $T^{\pi_{i-1}} + \beta_i$ [cf. (19)] indeed corresponds to a feasible schedule (for i jobs). Hence, (for $n = i$) the inequality in (19) becomes an equality, which is (21). This concludes the induction, and hence the proof. ∎

To illustrate the recursive algorithm in Theorem 6.3, consider the scheduling of $n = 5$ jobs with exponential task times (unit mean). The job sizes are $2, 3, 4, 5$ and 8. Figure 1 illustrates the steps in the recursion. Observe here the interplay of parallel and sequential scheduling as they vary with the number of jobs scheduled. For instance, when there are two jobs, both are scheduled fully parallel. The addition of a third job changes job 2 to a sequential job. Adding a fourth job turns job 2 back to parallel.

When the jobs have the same size, there exists a threshold n^*, in the spirit of Theorem 5.3. However, for the two processor case, we can also show that (a) the threshold structure exists for any given n, i.e., it need not be an asymptotic result; (b) the threshold value can be evaluated more explicitly.

Theorem 6.4 *Suppose $b_j = b$ for all $j = 1, \ldots, n$. For the optimal policy π, there exists a threshold value $1 \leq n^* \leq n$, such that the initial n^* jobs are fully parallel, and the remaining $n - n^*$ (necessarily an even number) jobs are fully sequential, where*

$$n^* = \max\{i \leq n :\ \beta_{i-1} + \beta_i \leq ib;\ n - i \text{ is an even number}\}. \tag{22}$$

Proof. First, we show that if π_i schedules job i sequentially, then π_{i+1} also schedules job $i+1$ sequentially. ¿From Theorem 6.3, set $b_j = b, j = 1, \ldots, n$. Then, for all $j \geq 2$,

$$T^{\pi_j} = \min\{T^{\pi_{j-1}} + \beta_j,\ T^{\pi_{j-2}} + jb\}. \tag{23}$$

Suppose that for some i, $T^{\pi_i} = T^{\pi_{i-2}} + ib$. Then,

$$\begin{aligned} T^{\pi_{i-1}} + (i+1)b &\leq T^{\pi_{i-2}} + \beta_{i-1} + (i+1)b \\ &\leq T^{\pi_{i-2}} + ib + \beta_{i+1} = T^{\pi_i} + \beta_{i+1}, \end{aligned}$$

where the first inequality follows from (23), the second from Jensen's inequality in Lemma 2.1. Thus, $T^{\pi_{i+1}} = T^{\pi_{i-1}} + (i+1)b$; hence π_{i+1} schedules job $i+1$ sequentially.

Let i^* be the largest i such that

$$T^{\pi_i} = T^{\pi_{i-1}} + \beta_i. \tag{24}$$

Thus,

$$T^{\pi_{i-1}} + \beta_i \leq T^{\pi_{i-2}} + ib \tag{25}$$

holds for all $i \leq i^*$. In conjunction with (24), the inequality in (25) is equivalent to $\beta_{i-1} + \beta_i \leq ib$. Since the optimal policy π_i in the algorithm of Theorem 6.3 only depends on π_{i-1} and π_{i-2}, an inductive argument on i shows that π_i schedules (i) the initial $i^* - 1$ jobs with a fully parallel policy, and the remaining $i - i^* + 1$ jobs with a fully sequential policy if $i - i^*$ is odd, and (ii) the initial i^* jobs with a fully parallel policy, and the remaining $i - i^*$ jobs with a fully sequential policy if $i - i^*$ is even. Hence, the specification of n^* follows. ■

In the case of Erlang-k task times (cf. Section 4), we can apply (17) to (22) to obtain

$$n^* \approx \frac{\pi k b}{8}.$$

It is interesting to observe that the threshold is essentially a constant multiplier of the ratio of job size (b) to variance ($1/k$). This supports the intuition that large job size or low variance favors parallel processing.

The next result gives a case for which a fully parallel policy is optimal in the two-processor case (i.e., the threshold $n^* = n$).

Proposition 6.5 *Suppose the job sizes, $b_i, i = 1, \ldots, n$, are all even numbers. Then a* fully parallel *policy is optimal, if the standard deviation of the task times, σ satisfies*

$$\sigma \leq \min_{1 \leq i \leq n} \{\sqrt{\eta_i} - \sqrt{\eta_{i-1}}\}/2 ,$$

or equivalently, $b_i \geq 4\sigma^2 + 4\sigma\sqrt{\eta_{i-1}}$, for all $i = 1, \ldots, n$.

Proof. Similar to the proof of Theorem 6.4, the sufficient condition for π_i to be a fully parallel policy is $\beta_{i-1} + \beta_i \leq \eta_i$. Note when the job sizes are all even numbers, we have $\beta_j = \alpha(\frac{1}{2}\eta_j, \frac{1}{2}\eta_j)$ for all j. ¿From Lemma 2.3 we have

$$\alpha(\frac{1}{2}\eta_j, \frac{1}{2}\eta_j) \leq \sigma\sqrt{\eta_j} + \frac{1}{2}\eta_j.$$

Hence, it suffices to have

$$\sigma(\sqrt{\eta_{i-1}} + \sqrt{\eta_i}) \leq \frac{1}{2}(\eta_i - \eta_{i-1}) = \frac{1}{2}b_i.$$

A simple calculation then yields the two desired relations (between the variance and the job sizes). ■

7 Optimal FPFS Policies

We now turn to the case of three or more processors. In the last section, via the example illustrated in Figure 1, we have seen that even in two processors, there is a need to alternate between parallel and sequential scheduling of jobs as the number of jobs increases. Such alternations are quite common in general, and they shed light on the complexity associated with determining the optimal policy for more than two processors. Finding the optimal policy requires evaluating numerous combinatoric inequalities, and there appears to be no general theory. The main difficulty is that the initial part of the optimal policy becomes extremely complicated. (The asymptotic "tail" part of the optimal policy, as we know from Section 4, consists of only sequential jobs.)

As an example consider scheduling 3 jobs each with 3 tasks on 3 processors. There are at least five different configurations that satisfy Proposition 4.1 and Proposition 4.2:

C1:

$$\begin{array}{cc|ccc|}
 & & \multicolumn{3}{c}{\text{Jobs}} \\
 & & 1 & 2 & 3 \\ \cline{3-5}
 & 1 & 1 & 1 & 1 \\
\text{Proc} & 2 & 1 & 1 & 1 \\
 & 3 & 1 & 1 & 1 \\ \cline{3-5}
\end{array}$$

C2:

$$\begin{array}{cc|ccc|}
 & & \multicolumn{3}{c}{\text{Jobs}} \\
 & & 1 & 2 & 3 \\ \cline{3-5}
 & 1 & 3 & 0 & 0 \\
\text{Proc} & 2 & 0 & 3 & 0 \\
 & 3 & 0 & 0 & 3 \\ \cline{3-5}
\end{array}$$

C3:

$$\begin{array}{cc|ccc|}
 & & \multicolumn{3}{c}{\text{Jobs}} \\
 & & 1 & 2 & 3 \\ \cline{3-5}
 & 1 & 1 & 2 & 0 \\
\text{Proc} & 2 & 1 & 1 & 1 \\
 & 3 & 1 & 0 & 2 \\ \cline{3-5}
\end{array}$$

C4:

$$\begin{array}{cc|ccc|}
 & & \multicolumn{3}{c}{\text{Jobs}} \\
 & & 1 & 2 & 3 \\ \cline{3-5}
 & 1 & 3 & 0 & 0 \\
\text{Proc} & 2 & 0 & 1 & 2 \\
 & 3 & 0 & 2 & 1 \\ \cline{3-5}
\end{array}$$

C5:

$$\begin{array}{cc|ccc|}
 & & \multicolumn{3}{c}{\text{Jobs}} \\
 & & 1 & 2 & 3 \\ \cline{3-5}
 & 1 & 0 & 2 & 1 \\
\text{Proc} & 2 & 1 & 1 & 1 \\
 & 3 & 2 & 0 & 1 \\ \cline{3-5}
\end{array}$$

The (i, j) th entry of the above "magic squares" gives the number of tasks of job j that are scheduled on processor i. Consider Erlang-k task times. It is easy to compute flow times. For instance, the flow time for **C3** is

$$T^{C3} = \alpha_k(1, 1, 1) + \alpha_k(2, 3) + \alpha_k(3, 3).$$

Hence we have the following results:

	$k = 1$	$k = 2$	$k = \infty$
C1	9.54	8.53	6*
C2	9*	9	9
C3	9.20	8.52*	8
C4	9.18	8.80	8
C5	9.49	8.73	8

where a $*$ indicates the optimum for the corresponding k value.

¿From this and other examples (not reported here), we observe that the optimal policy depends heavily on the distribution of task times. Generally it appears necessary to compute numerically the flow times of all schedules that satisfy the properties in Section 4 before the optimal schedule can be identified. This quickly leads to a combinatorial growth of the problem complexity. This motivates us to focus on a restricted class of policies that schedule the tasks of a job either on all processors (fully parallel) or on one processor (fully sequential). These will be termed "Fully Parallel or Fully Sequential policies" (FPFS).

We will further assume that all jobs have the same number of tasks that is equal to the number of processors, i.e., $b_i = b = P$, $i = 1, 2, \ldots, n$. Below we show that the optimal FPFS policy has a threshold structure that is similar to the optimal policy in the two-processor case, namely, the initial jobs are parallel jobs, until a threshold value is reached, then the remaining jobs are all sequential. An algorithm that constructs an optimal FPFS policy is developed. For Erlang task times, we further characterize this threshold value by bounds and approximations.

Let π_n^f denote the optimal policy among the class of FPFS policies when there are n jobs (The subscript n will be omitted if the argument is for any arbitrary n). Since we assume that $b_i = b = P$, $i = 1, 2, \ldots, n$, then for any FPFS policy the difference between the number of tasks scheduled on two different processors must be an integer multiple of P. As in the discussion of the general case, we can restrict our search of optimal FPFS policies to a class of policies that satisfy a set of properties. An optimal FPFS schedule that violates these properties can be modified to satisfy these properties without increasing the flow time. These properties are analogous to those previously established for the general case and are given by:

1. *The difference in the number of tasks scheduled on each processor under π^f must be either 0 or P.*

   The proof of this is analgous to the argument in Proposition 4.3. Here, because we are dealing with FPFS policies, we can eliminate intermediate values between 0 and P.

2. *There is an ordering of parallel jobs on the processors, i.e., all the tasks of a parallel job must be scheduled either completely before or completely after all the tasks of another parallel job.*

   This property of FPFS schedules follows from the submodularity property in Lemma 2.1(ii) (see [20] Proposition 6.E.1 for details). Suppose that there is no ordering and denote by $L^{\psi}_{p,j'}$ and $L^{\psi}_{p,j''}$ $p = 1, \ldots, P$, the positions of the tasks of jobs j' and j'' on processor p. The total cost for these jobs is given by

   $$\alpha(L^{\psi}_{1,j'}, \ldots, L^{\psi}_{P,j'}) + \alpha(L^{\psi}_{1,j''}, \ldots, L^{\psi}_{P,j''}). \tag{26}$$

   If jobs j' and j'' are ordered on the processors then the cost for these two jobs is given by

   $$\begin{aligned} & \alpha\left(\min\left\{L^{\psi}_{1,j'}, L^{\psi}_{1,j''}\right\}, \ldots, \min\left\{L^{\psi}_{P,j'}, L^{\psi}_{P,j''}\right\}\right) \\ + \quad & \alpha\left(\max\left\{L^{\psi}_{1,j'}, L^{\psi}_{1,j''}\right\}, \ldots, \max\left\{L^{\psi}_{P,j'}, L^{\psi}_{P,j''}\right\}\right). \end{aligned} \tag{27}$$

   The submodularity of α, however implies that (27) is less than or equal to (26) and thus the parallel jobs can be ordered on the processors without increasing their completion times.

Hence, we can restrict our investigation of optimal FPFS polices to those that satisfy the above properties. In following lemma, we show that the optimal FPFS policy is of the threshold type.

Lemma 7.1 *Under π_n^f, there exists a threshold value n^*, such that the initial n^* jobs are fully parallel, and the remaining $n - n^*$ jobs are fully sequential.*

Proof. We prove this by contraction. Suppose there is a parallel job that is scheduled after a sequential job. Since there is an ordering of the parallel jobs, let job j' be the first parallel job that is scheduled after a sequential job and job j'' be the last parallel job that is scheduled before job j'. Since there is no sequential job before j'', $L^{\psi}_{p,j''} = \ell$, $p = 1, \ldots, P$, for some constant ℓ. We will show that a new policy that schedules all the tasks of job j' to the positions $\ell + 1$ and all the sequential jobs between jobs j'' and j' one position back, would result in a smaller cost. The cost increase for the sequential jobs between these two jobs is $\sum_{p=1}^{P} (L^{\psi}_{p,j'} - 1 - \ell)/P$ (note that this value is always an integer since ψ is FPFS). The cost reduction of job j' is

$$\alpha(L^{\psi}_{1,j'}, \ldots, L^{\psi}_{P,j'}) - \alpha(\ell + 1, \ldots, \ell + 1). \tag{28}$$

It then follows from the majorization inequality and Jensen's inequality in Lemma 2.1 that

$$\begin{aligned}
& \alpha(L^{\psi}_{1,j'}, \ldots, L^{\psi}_{P,j'}) \\
\geq \; & \alpha(\frac{1}{P}\sum_{p=1}^{P} L^{\psi}_{p,j'}, \ldots, \frac{1}{P}\sum_{p=1}^{P} L^{\psi}_{p,j'}) \\
\geq \; & \alpha(\ell + 1, \ldots, \ell + 1) + \frac{1}{P}\sum_{p=1}^{P} L^{\psi}_{p,j'} - 1 - \ell. \qquad (29)
\end{aligned}$$

Thus, the sequential jobs are scheduled after parallel jobs under π^f. ■

The problem left is to find the threshold n^*. Define

$$\alpha_P(j) \equiv \alpha\,\underbrace{(j, j, \ldots, j)}_{P}.$$

Let $F(n, s)$ be the cost when s sequential jobs are scheduled after $n - s$ parallel jobs. Thus, the objective is to find $s^*(n)$ such that $F(n, s^*(n)) = \min_{0 \leq s \leq n} F(n, s)$.

Lemma 7.2 *The function $F(n, s)$ is submodular [cf. Lemma 2.1 (ii)], and $s^*(n)$ is increasing in n.*

Proof. Note that $F(n+1,s) - F(n,s) = \alpha_P(n+1-s) + s$ since there is one more parallel job in $F(n+1,s)$ and all the s sequential jobs in $F(n+1,s)$ are scheduled one position after those in $F(n,s)$. It then follows from Jensen's inequality that

$$\begin{aligned} & F(n+1,s+1) - F(n,s+1) - (F(n+1,s) - F(n,s)) \\ = \; & 1 - \alpha_P(n+1-s) + \alpha_P(n-s) \leq 0. \end{aligned}$$

This concludes the argument for submodularity. To show that $s^*(n)$ is increasing observe from the submodularity of $F(n,s)$ that

$$F(n+1,s) - F(n+1,s^*(n)) \geq F(n,s) - F(n,s^*(n)) \geq 0$$

for all $s < s^*(n)$. ■

Note that the increasing property of the minimizer (e.g., $s^*(n)$) of a submodular function (e.g. $F(n,s)$) is known to hold in a more general setting; refer to Topkis [21].

Lemma 7.2 has simplified the amount of computation requires to determine the threshold. In the following lemma, we will further show that the threshold is periodic for large n. Moreover, the number of sequential jobs is increased by at most P when the total number of jobs is increased by 1.

Lemma 7.3 **(i)** *If $s^*(n) \geq P$, then $s^*(n+P) = s^*(n) + P$.*

(ii) $s^*(n+1) - s^*(n) \leq P$.

Proof. (i) From Lemma 7.2 and by assumption it follows that $s^*(n+P) \geq s^*(n) \geq P$. The extension of the subset property in Theorem 4.5 to that of FPFS schedules implies that $s^*(n+P) = s^*(n) + P$.

(ii) If $s^*(n+1) > s^*(n) + P$, then it follows from (i) that $s^*(n+1) = s^*(n+1-P) + P$. This implies that $s^*(n+1-P) > s^*(n)$ which contradicts Lemma 7.2. ■

Theorem 7.4 *The following algorithm constructs an optimal FPFS policy π^f.*

1. $s^*(1) = 0$.

2. *If $s^*(n) < P$, then $s^*(n+1)$ is the argument of $F(n+1,s)$ such that*

$$F(n+1, s^*(n+1)) = \min_{s^*(n) \le s \le \min(n+1, s^*(n)+P)} F(n+1, s).$$

Otherwise $s^*(n+1) = s^*(n+1-P) + P$.

Note that for $s \le P$, $F(n,s) = \sum_{j=1}^{n-s} \alpha_P(j) + s(n-s+P)$, where the first and second terms are the cost of parallel and sequential jobs respectively. For exponential service times, $\alpha_P(j)$ can be obtained by the recursive equations as in (12) and (13). A more efficient algorithm than the recursive equations is proposed in the Appendix.

As outlined in Theorem 7.4, the optimal FPFS policy is completely determined once $s^*(n) = P$ for some n is reached. Thus, it is of interest to find $n^f = \inf\{n : s^*(n) = P\}$. The condition is equivalent to

$$\begin{aligned} n^f &= \inf\{n : \sum_{j=1}^{n-s} \alpha_P(j) + s(n-s+P) \ge \sum_{j=1}^{n-s} \alpha_P(j) + nP, s = 0, \ldots, P-1\} \\ &= \inf\{n : \sum_{j=n-P+1}^{n-s} \alpha_P(j) \ge (n-s)(P-s), s = 0, \ldots, P-1\} \end{aligned} \tag{30}$$

Analogous to the asymptotic sequential property in Theorem 5.3, $n^f < \infty$ if the task times are non-deterministic. In the following theorem, we show a lower bound for n^f for Erlang task times.

Theorem 7.5 *If the task time X follows the Erlang-k distribution, then*

$$n^f \ge \max_{0 \le s \le P-1} \frac{P+s-1}{2} + \frac{k}{4\log P}\left(\frac{P-s-1}{2} - \frac{\log P}{k}\right)^2 \equiv \tilde{n}. \tag{31}$$

Proof. Note that $\alpha_P(j)$ can be bounded as follows: for $\theta > 0$,

$$\begin{aligned} \alpha_P(j) &= E\max(S^1(j), \ldots, S^P(j)) \\ &\le \frac{1}{\theta} \log E \exp\Big(\theta \max(S^1(j), \ldots, S^P(j))\Big) \\ &= \frac{1}{\theta} \log E \max\Big(\exp(\theta S^1(j)), \ldots, \exp(\theta S^P(j))\Big). \\ &\le \frac{1}{\theta} \log \sum_{i=1}^{P} E e^{\theta S^i(j)} = \frac{1}{\theta}\log P + \frac{j}{\theta}\log\phi(\theta), \end{aligned}$$

where the first inequality is due to Jensen's inequality (log is concave), the second inequality is due to $\max(x_1, x_2) \le x_1 + x_2$ for $x_1, x_2 \ge 0$, and $\phi(\theta) = E\exp(\theta X)$ is the moment generating function of X. Thus,

$$\sum_{j=n-P+1}^{n-s} \alpha_P(j) \leq \frac{P-s}{\theta} \log P + \frac{P-s}{\theta} \log \phi(\theta)(n - \frac{P+s}{2} + \frac{1}{2}). \tag{32}$$

When X follows the Erlang-k distribution, $\phi(\theta) = (\frac{1}{1-\theta/k})^k$. It then follows from the inequality $\log(1+x) \leq x$ that

$$\frac{1}{\theta} \log \phi(\theta) \leq \frac{1}{1-\theta/k}. \tag{33}$$

Inserting (33) into (32) and optimizing the right hand side of (32) with respect to θ, we have

$$\sum_{j=n-P+1}^{n-s} \alpha_P(j) \leq (P-s)(\sqrt{\frac{\log P}{k}} + \sqrt{n - \frac{P+s}{2} + \frac{1}{2}})^2$$

Replacing the right hand side above in (30) and solving n for each fixed s yields $\tilde{n}$ in (31). ∎

We note that one can also use the bounding technique for $\alpha_P(j)$ in Theorem 7.5 to improve the upper bound in Theorem 3.3 for Erlang-k task times. For instance, one can show that

$$\alpha_P(j) \leq (\sqrt{\frac{\log P}{k}} + \sqrt{j})^2. \tag{34}$$

Let $\tilde{n}'$ be the smallest integer such that $\tilde{n}' \geq \tilde{n}$ and $n - \tilde{n}'$ is an integer multiple of P. Then

$$T_n^\pi \leq F(n, n-\tilde{n}') \leq \sum_{j=1}^{\tilde{n}'} (\sqrt{\frac{\log P}{k}} + \sqrt{j})^2 + (n-\tilde{n}')(\frac{n+\tilde{n}'+P}{2}).$$

Note that an approximation for $\alpha_P(j)$ could be obtained by using the central limit theorem. Recall that $\alpha_P(j) = E\max(S^1(j), \ldots, S^P(j))$. When j is large, $(S^i(j)-j)/\sigma\sqrt{j}$, for $i = 1, ..., P$, can be approximated by a standard normal variate, provided the task time X has a positive, finite variance σ^2. Thus, $(\alpha_P(j)-j)/\sigma\sqrt{j}$ is the expectation of the maximum of P independent standard normal variates, and hence (cf. [22])

$$\alpha_P(j) \approx j + \sigma\sqrt{2j \log P}. \tag{35}$$

Since $\sigma^2 = 1/k$ for Erlang-k task times, the approximation for $\alpha_P(j) - j$ in (35) is approximately one half of the upper bound in (34).

8 Concluding Remarks

The main findings of the paper can be summarized as follows:

(i) In genearal (Theorem 3.3), there exist upper and lower bounds on the optimal flow time, which correspond, respectively, to the fully sequential schedule and to the fully parallel schedule with constant task times. Furthermore, the ratio of the upper bound to the lower bound approaches unity as the number of jobs increases.

(ii) For equal-size jobs, the optimal schedule has a sequential tail (Theorem 5.3): all jobs beyond a certain threshold value has to be scheduled in a fully sequential fashion. Furthermore, the threshold value is usually increasing in the number of processors and decreasing in the variance of task times.

(iii) For the case of two processors, the optimal schedule is completely generated by a recursive algorithm (Theorem 6.3).

(iv) For the case of three or more processors and equal size jobs, the optimal schedule in the FPFS class is characterized by a single threshold value (Theorem 7.4). The quality of such a schedule in general (i.e., outside of the FPFS class) is supported by the bounds in (i) and the asymptotic sequential property in (ii).

In several ways, our results have brought out the importance of *sequential* scheduling in the distributed parallel processing setting; for instance, refer to the role of sequential policies in (i), (ii) and (iv) above. These results suggest that unless there is a large number of processors and/or the task times are nearly deterministic, the synchronization overhead of parallel processing will make it ineffective as compared against sequential processing.

Underlying most of the structural results of the optimal schedule are four key properties of the max and plus operators: subadditivity, submodularity, convexity, and Schur convexity (cf. Lemma 2.1). This suggests that our approach may extend to objectives that are more general than the flow time. The essential requirement is that the objective function preserves those properties of the max and plus operators.

In the most general setting, $P \geq 3$ processors and different job sizes, finding the optimal schedule becomes an extremely difficult combinatorial problem. For instance, we do not even know whether it is always optimal to schedule smaller jobs first (as in the case of two processors, as well as in fully sequential and fully parallel policies with deterministic task times). However, our study supports the thesis that a policy that will do very well is one that schedules jobs in increasing order of their sizes, with fully parallel jobs up to a threshold, and then switches to fully sequential jobs. Furthermore, our results have provided qualitative characterization and, in some cases, constructive ways for identifying the threshold value.

9 Appendix: An Algorithm for $\alpha_P(m)$

Here we present an algorithm to compute $\alpha_P(m)$ for exponential service times. For Erlang service times, we can use the transformation in (12). Solving a similar recursion as (13), Durivault and Nelson [23] derive the following expression

$$\alpha_P(m) = \sum_{k=1}^{P} \frac{(-1)^{k+1}}{k} \binom{P}{k} \left[\sum_{i_1=0}^{m-1} \cdots \sum_{i_k}^{m-1} \binom{i_1 + \cdots + i_k}{i_1, \ldots, i_k} \frac{1}{k^{i_1+\cdots+i_k}} \right]. \tag{36}$$

Rewrite the above as

$$\alpha_P(m) = \sum_{k=1}^{P} \frac{(-1)^{k+1}}{k} \binom{P}{k} \left[\sum_{\ell=0}^{k(m-1)} \frac{f_k^m(\ell)}{k^\ell} \right], \tag{37}$$

where

$$f_k^m(\ell) = \sum_{\substack{i_1 + \cdots + i_k = \ell \\ i_1, \ldots, i_k \leq m-1}} \binom{\ell}{i_1, \ldots, i_k}. \tag{38}$$

Note that $f_k^m(\ell)$ has the following combinatorial interpretation: it is the number of ways to put ℓ distinguishable balls into k distinguishable cells, with each cell having at most $m-1$ balls. Two recursions can be derived for $f_k^m(\ell)$:

$$f_k^m(\ell) = \sum_{j=0}^{m-1} \binom{\ell}{j} f_{k-1}^m(\ell - j); \tag{39}$$

$$f_k^m(\ell + 1) = k[f_k^m(\ell) - \binom{\ell}{m-1} f_{k-1}^m(\ell - m + 1)]. \tag{40}$$

To understand (39), we place $j \leq m-1$ balls in cell k. There are $\binom{\ell}{j}$ ways to choose the j balls and for each choice there are $f_{k-1}^m(\ell - j)$ ways to put the remaining $\ell - j$ balls into the remaining $k-1$ cells. To understand (40), pick one ball, say ball ℓ. It can be put into any one of the k cells. Once the ball is in a particular cell, the number of ways to put the other ℓ balls into the k cells follows (39), with the exception that since the cell that already has a ball it can only accommodate at most $m-2$ balls. These arguments thus yield the quantity in squared brackets.

For computational purposes further simplification is helpful. Defining $g_k^m(\ell) = f_k^m(\ell)/k^\ell$ and substituting this into (39) and (40) yields

$$g_k^m(\ell) = \sum_{j=0}^{m-1} \binom{\ell}{j} \frac{(k-1)^{\ell-j}}{k^\ell} g_{k-1}^m(\ell-j); \tag{41}$$

$$g_k^m(\ell+1) = g_k^m(\ell) - \binom{\ell}{m-1} \frac{(k-1)^{\ell-m+1}}{k^\ell} g_{k-1}^m(\ell-m+1). \tag{42}$$

Since for $\ell \le m-1$, $f_k^m(\ell) = k^\ell$, and hence $g_k^m(\ell) = 1$, we can now rewrite (37) as

$$\alpha_P(m) = \sum_{k=1}^{P} \frac{(-1)^{k+1}}{k} \binom{P}{k} \left[n + \sum_{\ell=m}^{k(m-1)} g_k^m(\ell) \right]. \tag{43}$$

Next, consider the general case $\boldsymbol{n} = (n_j)_{k=1}^P$. Write $\boldsymbol{n}(j) = (n_1, \ldots, n_k)$, for $k = 1, \ldots, P$. Then (38) and (39) become

$$f^{\boldsymbol{n}(k)}(\ell) = \sum_{\substack{i_1 + \cdots + i_k = \ell \\ i_1 \le n_1 - 1, \ldots, i_k \le n_k - 1}} \binom{\ell}{i_1, \ldots, i_k}, \tag{44}$$

$$f^{\boldsymbol{n}(k)}(\ell) = \sum_{j=0}^{n_k-1} \binom{\ell}{j} f_{k-1}^{\boldsymbol{n}(k-1)}(\ell-j). \tag{45}$$

(Note, in this case, the recursion in (40) does not apply.) Accordingly, (41) becomes

$$g_k^{\boldsymbol{n}(k)}(\ell) = \sum_{j=0}^{n_k-1} \binom{\ell}{j} \frac{(k-1)^{\ell-j}}{k^\ell} g_{k-1}^{\boldsymbol{n}(k-1)}(\ell-j). \tag{46}$$

Finally, without loss of generality, assume $n_1 = \min_k\{n_k\}$; then (43) becomes

$$\alpha(P;\boldsymbol{n}) = \sum_{k=1}^{P} \frac{(-1)^{k+1}}{k} \binom{P}{k} \left[n_1 + \sum_{\ell=n_1}^{n_1+\cdots+n_k-k} g_k^{\boldsymbol{n}(k)}(\ell) \right]. \tag{47}$$

Using the recursion in (40) and (43), the computation effort of our algorithm is $O(mP^2)$ and the storage requirement is $O(P^2)$.

References

[1] F. Baccelli, G. Cohen, G.J. Olsder and J.-P. Quadrat, *Synchronization and Linearity*, Wiley, Chichester, 1992.

[2] P. Glasserman and D.D. Yao, *Monotone Structure in Discrete-Event Systems*, Wiley, New York, 1994.

[3] J.F.C. Kingman, "Subadditive ergodic Theory," *Ann. Probability*, **1** (1973), pp. 883-909.

[4] J.E. Conhen, "Subadditivity, generalized products of random matrices and Operations Research," *SIAM Review*, **30** (1988), pp. 69-86.

[5] C.S. Chang, "On the exponentiality of stochastic linear systems under the max-plus algebra," preprint.

[6] C.S. Chang and R. Nelson, "Bounds on the speedup and efficiency of partial synchronization in parallel processing systems," *J. Assoc. Comp. Mach.*, to appear.

[7] C.S. Chang, "A new ordering for stochastic majorization: theory and application," *Adv. Appl Prob.*, **24** (1992), pp. 604-634.

[8] C.S. Chang and D.D. Yao, "Rearrangement, majorization and stochastic scheduling," *Math. of Operations Res.* **18** (1993), pp. 658-684.

[9] C.S. Chang, J.G. Shanthikumar and D.D. Yao, "Stochastic convexity and stochastic majorization," in *Stochastic Modeling and Analysis of Manufacturing Systems,* D.D. Yao (ed.), Springer-Verlag, New York, 1994, pp. 189-231.

[10] A.M. Makowski and R. Nelson, "Optimal scheduling for a distributed parallel processing model," *IBM Research Report* **RC–17449**, 1991.

[11] A.M. Makowski and R. Nelson, "Distributed parallelism considered harmful," *IBM Research Report* RC 17448, 12/12/91, submitted to *IEEE Transactions on Distributed and Parallel Systems.*

[12] F. Baccelli and A. Makowski, "Queueing models for systems with synchronization constraints," *Proceedings of the IEEE* **77** (1989), pp. 138–161.

[13] F. Baccelli, A.M. Makowski and A. Shwartz, "The Fork–Join queue and related systems with synchronization constraints: Stochastic ordering and computable bounds," *Advances in Applied Probability* **21** (1989), pp. 629–660.

[14] F. Baccelli, W.A. Massey and D. Towsley, "Acyclic Fork–Join queueing networks," *J. Assoc. Comp. Mach.* **36** (1989), pp. 615–642.

[15] F. Baccelli and Z. Liu, "On the execution of parallel programs on multiprocessor systems: A queueing theory approach," *J. Assoc. Comp. Mach.* **37** (1990), pp. 371–413.

[16] R. Nelson, D. Towsley and A.N. Tantawi, "Performance analysis of parallel processing systems," *IEEE Transactions on Software Engineering* **SE–14** (1988), pp. 532–540.

[17] R. Nelson and A.N. Tantawi, "Approximate analysis of Fork/Join synchronization in parallel queues," *IEEE Transactions on Computers* **C–37** (1988), pp. 739–743.

[18] M.L. Pinedo, *Scheduling: Theory, Algorithms and Systems,* Prentice-Hall, 1994.

[19] C.S. Chang, R. Nelson and D.D. Yao, "Optimal Task Scheduling on Distributed Parallel Processors," *Performance Evaluation,* **20** (1994), pp. 207-221.

[20] A.W. Marshall and I. Olkin, *Inequalities: Theory of Majorization and Its Applications*, Academic Press, New York (NY), 1979.

[21] D.M. Topkis, "Minimizing a Submodular Function on a Lattice," *Operations Research*, **26** (1978), pp. 305–321.

[22] H.A. David, *Order Statistics*, J. Wiley & Sons, New York (NY), 1981.

[23] L. Durivault and R. Nelson, "An Expression for the Maximum of Erlang Random Variables," *IBM Research Report RC 16751*, 1991.