Effects Of Asynchronism On The Convergence Rate Of A Class Of Iterations

Aydin Uresin and Michel Dubois

CENG Technical Report 95-05

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213)740-4475

March 1995

EFFECTS OF ASYNCHRONISM ON THE CONVERGENCE RATE OF A CLASS OF ITERATIONS

Aydin Üresin* and Michel Dubois

*Istanbul Technical University
Department of Electrical Engineering
Ayazaga Kampusu, 80626
Istanbul, Turkey
eeuresin%tritu.bitnet@vm.usc.edu

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-2562
(213)740-4475
dubois@paris.usc.edu

Abstract

In multiprocessor systems, iterative algorithms can be implemented synchronously or asynchronously. The choice depends mostly on performance. Unfortunately, few guidelines exist to make that decision. In this paper, we compare the execution times of an asynchronous iterative algorithm and of its synchronous counterpart for a class of asynchronous iterations which includes iterations with monotone mappings. Synchronization overhead and communication times are neglected in order to focus on the effect of asynchronism on the convergence rate.

Under some assumptions, we derive an analytical model. In this model, Q tasks with identical and independently distributed execution time distributions execute on P processors. Using simulations as well as analytical models, we show the effects of execution time fluctuations, of the number of processors and tasks, of the scheduling policy and of the amount of coupling among iterate components. The models show that the asynchronous iteration may be up to twice as slow as the synchronous one. This worst case is achieved when the processing time fluctuations are very small.

Keywords: iterative algorithms, asynchronous algorithms, multiprocessors, synchronization, performance analysis.

1. INTRODUCTION

The goal of parallel processing is to reduce execution time by running concurrent and cooperating processes on multiple processors. Parallel processes communicate and synchronize by accesses to shared writable data or by message-passing mechanisms. The way in which these communication and synchronization mechanisms are supported has great impact on system performance. In this paper we focus on iterative algorithms, in which an operator is applied to a set of data repetitively until convergence. In a multiprocessor, consecutive iterations are separated by a *barrier synchronization.* Barrier synchronization is a common synchronization mechanism often supported by the hardware and which defines a point in the execution of a parallel program such that no process can pass the barrier unless all other processes have done so [1]. Barriers are not needed in asynchronous implementations of iterative algorithms and therefore an asynchronous iteration has the potential of being tolerant to very large communication and synchronization latencies. Unfortunately, the convergence rate of an asynchronous iteration may be less than that of its synchronous counterpart because processors never wait for fresh values of iterate components. The goal of this paper is to evaluate through simulations and analytical models the effects of asynchronism on the convergence rate of iterative algorithms for an important class of iterations including iterations with monotone mappings.

Most iterative algorithms, both for numerical and non-numerical data, can be described by an operator F repetitively applied to some data x of dimension $n > 1$ starting with an initial value $x(0)$. Such algorithms are represented as follows:

$$x_i(k+1) = F_i(x(k)), \qquad k = 0, 1,... \tag{1}$$

for all $i = 1, 2,..., n$, and where $x(k) = (x_1(k), x_2(k), ..., x_n(k))$ is the value of $x = (x_1, x_2, ..., x_n)$ at the k-th iteration. In a multiprocessor different components can be computed in parallel by different processors in each iteration. The strict application of (1) results in a *synchronous* iterative algorithm (*synchronous iteration*), in which each processor has to wait for all the new values of all components before starting the next iteration. By contrast, in *asynchro-*

nous iterative algorithms (*asynchronous iterations*), processors are not required to wait for the new values produced by other processors. Rather the components are updated at arbitrary time instances and a new value of the i-th component is produced at some time instance t according to:

$$x_i(t) = F_i(u^i(t))$$

where the values of the components of $u^i(t)$ were generated before t. $u^i(t)$ is called *the input at t for the i-th component*. $x(t)$ denotes the value of x at time t. For notational convenience, we assume that the computations start at $t = 0$ and that the initial data $x(0)$ are generated at $t = -\infty$. Notice that this definition covers a large class of algorithms, including synchronous iterations.

Without any restriction on the timing of computations in an asynchronous iteration, it is not possible to guarantee its convergence. The minimum requirements for the convergence of an asynchronous iteration are as follows.

1. For each time instance t of the execution of an asynchronous iteration, there exists an instance $t' > t$ such that all the components are updated between t and t' (*Progress of updates*).

2. For each time instance t of the execution of an asynchronous iteration, let $\tau(t)$ be the largest time instance such that all the updates after t never use input components generated prior to $\tau(t)$. Then the sequence $\tau(t)$ must tend to infinity as t goes to infinity (*Progress of inputs*).

Asynchronous iterations satisfying these conditions are called *totally asynchronous* [5]. Such asynchronous iterations only exclude starving computations.

There has been a considerable amount of work in the literature on the convergence conditions for totally asynchronous iterations [2, 4, 5, 6, 7, 9, 13, 14, 15, 18, 20, 21, 22, 23, 25, 26, 27]. The most general result can be stated as follows [5, 7, 27].

Proposition 1. *Let* $\{X(k)\}$ *be a sequence of sets such that*

- $X(k) = X_1(k) \times X_2(k) \times \ldots \times X_n(k)$, *for all k.*
- $\{\xi\} \subseteq \ldots \subset X(k+1) \subset X(k) \subset \ldots \subset X(0) \subseteq X$ *for all k, and furthermore all sequences* $\{z(k)\}$ *such that* $z(k) \in X(k)$ *for all k converge to* ξ, *where* ξ *is the unique fixed point of F in* $X(0)$.
- $F(x) \in X(k+1)$, *for all k and* $x \in X(k)$.

Then, all asynchronous iterations corresponding to F and starting with some $x(0)$ *in* $X(0)$ *converge to* ξ.

We will not give a complete proof here, but for later reference we will state the key intermediate result as a lemma. Let $\{\varphi(k)\}$ be a non-decreasing sequence of time instances starting with zero such that

- All the components are updated within the time interval $(\varphi(k), \varphi(k+1)]$.[1]
- All the updates after $\varphi(k)$ use input components generated after $\varphi(k-1)$.

Also let $\varphi_i(k)$ be the first time within the interval $(\varphi(k), \varphi(k+1)]$ at which the i-th component is updated. Then the following lemma holds.

Lemma 1. *Under the conditions of Proposition* 1

$$t \geq \varphi_i(k) \Rightarrow x_i(t) \in X_i(k)$$

This lemma can be proven by mathematical induction. For more details, the reader should refer to [27]. Since all the components are updated within the interval $(\varphi(k), \varphi(k+1)]$ $x(t)$ indefinitely enters $X(k+1)$ after $\varphi(k+1)$, i.e.,

$$t \geq \varphi(k) \Rightarrow x(t) \in X(k) \qquad \text{for all } k > 0 \tag{2}$$

1. $(a, b]$ denotes the set of real numbers $\{x \mid a < x \leq b\}$

Any sequence $\{(\varphi(k), \varphi(k+1)]\}$ of time intervals satisfying the above property (2) is called a *pseudo-cycle sequence.*

This paper addresses the problem of the efficiency of an asynchronous iteration with respect to its synchronous counterpart. In particular, we would like to know the performance effect obtained by simply removing the synchronization points at the end of each iteration of a synchronous iterative algorithm. Few authors have tackled this performance problem by using various models. In [6] and in [7, section 6.3.5], two important classes of iteration operators, namely monotonic and contracting operators, were considered in the context of message-passing systems and under the assumption that the messages in the system are received in the same order as they are sent. The analysis is applied to iterations with various degrees of coupling among iterate components. It is concluded that an asynchronous iteration cannot be slower than its synchronous counterpart, provided that the computation of each component takes one time unit and that the synchronous version suffers delays of at least one time unit between iterations due to the communication of recent data. It is not clear how the results of [6] can be applied when the computation times are not constant. In [12], which deals with shared-memory multiprocessors, the convergence issues are ignored but conflicts for memory and for critical sections are modeled. In [25], it is shown that, under a set of conditions, asynchronous distributed algorithms have good communication complexity in the sense that the message traffic they generate is not excessive.

There are many factors affecting the efficiency of multiprocessor algorithms and it is impossible to include them all in the same model. For example, in shared-memory systems, the performance of iterative algorithms is affected by the following factors: overhead caused by the execution of synchronization primitives, processor allocation and scheduling overhead, software lock-out on critical sections, memory access conflicts, and timing of shared memory updates (i.e., some efficiency is lost because of explicit waits in the synchronous case and of implicit delays in utilizing the most recent data in the asynchronous case.) In this paper, we focus on this latter effect. We introduce a model to evaluate the effects of variable computation times on the conver-

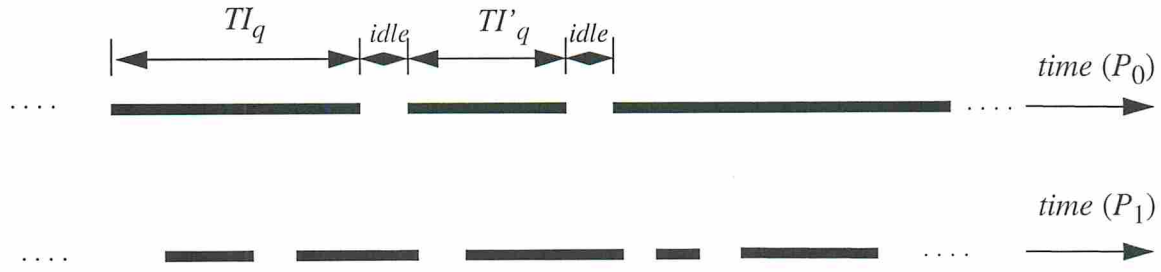
gence rate of asynchronous iterations.

The basic model of Section 2 restricts the class of operators to which the results of this paper are applicable. Sections 3 and 4 refine the model of Section 2 for the cases of strongly coupled and of partially coupled computations, and present the results of simulations and of an analytical model based on tasks with independent and identical distribution of their execution times. Finally the contributions of the paper are highlighted in Section 5.

2. GENERAL MODEL

In this section, we describe the general computational model of this paper. The notations and assumptions stated here will hold throughout the paper without further mentioning explicitly

Figure 1. An asynchronous iteration.



2.1. Architecture Model

We consider a multiprocessor system with P processors. The set of processors in the system is $\mathcal{P}$= $\{P_0, P_1,..., P_{P-1}\}$. The components of x are partitioned into Q subsets ($Q \geq P$). Each partition forms a *task* which is an indivisible unit of execution: from its beginning until its end, a task is executed by the same processor without interruption. The set of tasks is $\mathcal{T}$= $\{T_0, T_1,..., T_{Q-1}\}$. Each task is executed infinitely many times. All the activities regarding the update of each component in a partition, including fetching the current value of the shared data, computing the components and updating the global store are performed in the associated task. A time interval that covers all these activities of a task T_q is called a *task interval* of T_q, and is denoted TI_q. An asynchronous iterative algorithm (asynchronous iteration) corresponds to an allocation of each task to infinitely many time slots of available processors (Figure 1). We make the following assumption.

Assumption 1.

- *The length of each task interval is finite and in a finite period of time all the components are updated.*
- *The processors never stay idle except at the synchronization points between the iterations. The execution time of the synchronization primitives and the overhead due to other type of synchronizations (if any) such as critical sections are ignored.*
- *The task intervals of the same task never overlap in time.*

Although no particular architecture is specified, this model is more useful for shared memory systems. The first condition is required to enforce total asynchronism. The second condition eliminates the kind of overhead which is associated with the synchronous iteration. The major restriction is the third (non-redundancy) condition. Although the results are derived under this condition, they are still good approximations when a small amount of redundancy is allowed.

2.2. Restriction on the Iteration Operator

Let us now further restrict F in the following manner.

Assumption 2. *Let $R(k)$ be defined as $R(k) = X(k) - X(k+1)$, for all k. Then,*

$$x \in R(k) \Rightarrow F(x) \in R(k+1) \quad \textit{for all } k.$$

This restriction simply states that the application of the operator to some data in $R(k)$ moves the data to $X(k+1)$, but not any further, i.e., not to $X(k+2)$. If we define the *age* $A(x)$ of x as the largest integer that satisfies $x \in X(A(x))$, then it can be restated as $A(F(x)) = A(x) + 1$.

The main problem addressed in this paper is the comparison between the convergence rate of an asynchronous iteration $\mathcal{A}$ corresponding to F and starting with $x(0) \in R(0)$ and the convergence rate of its synchronous version starting with the same initial vector. The synchronous version can be represented by $y(k) = F(y(k-1))$ for all k, where $y(k)$ is the value of the data at the end

of the k-th iteration. Given Assumption 2, the iterates in the synchronous algorithm enter $R(k)$, at the end of the k-th iteration, for all k.

The execution time of an iteration is the time it takes for the iterates to indefinitely enter a certain domain $X(M)$. From the definition of pseudo-cycle sequence, the execution time is no more than M pseudo-cycles in the asynchronous case and, from Assumption 2, it is exactly M iterations in the synchronous case. Therefore the ratio between the asynchronous and synchronous execution times is upper-bounded by the ratio between the average pseudo-cycle time and the average synchronous iteration time. **This result, derived from Assumption 2, is key to all evaluations in this paper**. Without this assumption on F the results would not always hold because, in general, the synchronous version might "luckily" hit the solution in an arbitrarily small number of iterations.

The class of iterative algorithms satisfying Assumption 2 includes iterations with monotone mappings. A monotone mapping f is such that if $x \leq y$ then $f(x) \leq f(y)$. Important examples of algorithms with monotone mappings are: Linear iterations involving non-negative matrices, the Bellman-Ford algorithm for the shortest path problem, the successive approximation algorithm for infinite horizon dynamic programming, and dual relaxation algorithms for linear and non-linear network flow problems [6, 7].

Let $\{\varphi_{min}(k)\}$ be the minimum pseudo-cycle sequence corresponding to an asynchronous iteration $\mathcal{A}$. In other words, for all pseudo-cycle sequences and for all k, $\varphi_{min}(k) \leq \varphi(k)$. Also let ϕ_{min} and I denote the average pseudo-cycle time of $\{\varphi_{min}(k)\}$ and the average iteration time of the synchronous version, respectively. These averages are taken over all the pseudo-cycles (or iteration) of one execution. The ratio between the asynchronous and synchronous versions of the same algorithm is called the *slowdown factor* and is denoted by S. Then,

$$S \leq \frac{M \times \phi_{min}}{M \times I} .$$

The value of ϕ_{min} depends on the coupling among iterate components. In the following section, we analyze the case of strong coupling.

3. STRONGLY COUPLED ITERATION OPERATORS

For a given iteration operator F, the worst we can expect is that, in each pseudo-cycle, each task needs the outcome of all the tasks executed in the previous pseudo-cycle in order to make any progress towards the solution. This is the situation where the "coupling" among the tasks is the strongest possible.

3.1. Description of the Model

We define *strong coupling* by the following Assumptions 3 and 4. The first one restricts the computational model and the second one describes the condition on the iteration operator F.

Assumption 3. *The components computed in each task interval TI_q are released only at the end of TI_q. Furthermore, the values of input components used for these computations are the ones available at the start of TI_q.*

The *age* $A_i(x_i)$ of component x_i is defined as the largest integer such that $x_i \in X_i(A_i(x_i))$ and the *age* $A(x)$ of x is the $Min_j\{A_j(x_j)\}$. When a task is allocated to multiple components, the age of the task is also defined as the age of its components.

Assumption 4. *For all x and i the iteration operator F satisfies*

$$A_i(F_i(x)) = \min_j \{A_j(x_j)\} + 1 = A(x) + 1$$

For strongly coupled iterations, we define the following sequence $\{\varphi'(k)\}$.

Definition 1. *$\{\varphi'(k)\}$ is the increasing sequence of time instances starting with zero such that for all k, the time interval $(\varphi'(k), \varphi'(k+1)]$ is the smallest interval that covers at least one task interval of each task.*

Let ϕ' be the average pseudo-cycle length of $\{\varphi'(k)\}$. The following proposition shows that under the above assumptions, ϕ' is the exact estimate of ϕ_{min}.

Proposition 2. $t < \varphi'(k) \Rightarrow x(t) \notin X(k)$, *for all* $k > 0$.

Proof. We can prove this claim by induction. Since there exists a component x_i which is not updated before $\varphi'(1)$, the age of x_i and therefore the age of x is 0 prior to $\varphi'(1)$. This proves the proposition for k=1. Suppose that it also holds for k=1,2,...,l. Consider the first task interval TI_q on a processor P_p covered by the l+1-th pseudo-cycle of $\{\varphi'(k)\}$. Because of Lemma 1 each component x_i updated in TI_q indefinitely enters $X_i(l+1)$ right after TI_q. If components x_i updated in TI_q are also updated right before TI_q within the l+1-th pseudo-cycle, the input for this update is generated earlier than $\varphi'(l)$; therefore it is not in $X(l)$ and the updated component x_i at this instance cannot be in $X_i(l+1)$. This means that x does not enter $X(l+1)$ before the processors execute at least one instance of each task in the l-th pseudo-cycle. This proves the proposition for $k=l+1$ and the claim follows. ❑

Since the iterates indefinitely enter $X(k)$ after $\varphi'(k)$ and since $\varphi'(k)$ is the earliest such instance, an asynchronous iteration under the assumption of this section takes exactly M pseudo-cycles of $\{\varphi'(k)\}$ and the slowdown factor S can be written as $S = \frac{\phi_{min}}{I} = \frac{\phi'}{I}$.

Figure 2(a) shows an iteration of a synchronous algorithm for 5 tasks and 3 processors. A pseudo-cycle of a corresponding asynchronous algorithm is in Figure 2(b). We observe that in the synchronous case all the processors restart right after the previous iteration, whereas in the asynchronous case only one processor restarts right after the previous pseudo-cycle (P_0 in the figure). From Proposition 2, the tasks starting before the k-th pseudo-cycle and completing within the k-th pseudo-cycle do not increment the ages of their components and they must be considered "wasted."

We can identify three parts in the total processor time $\phi(k)\bullet$P spent in the k-th pseudo-cycle of an asynchronous iteration, as follows.

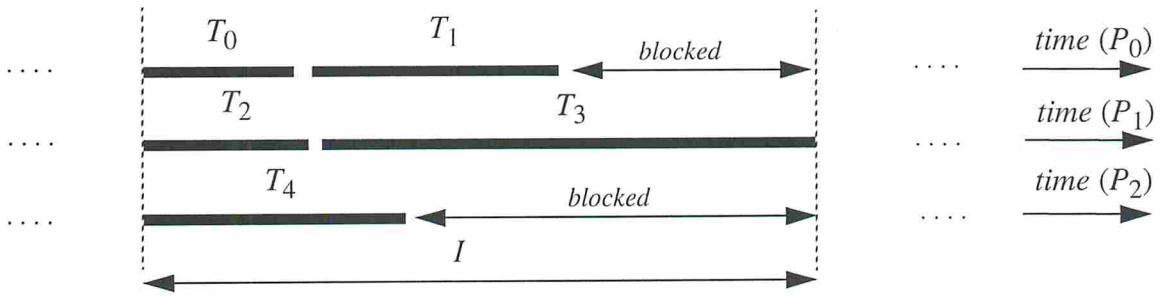
- Part 1: $\phi_1(k)$•P. Let $d_p(k)$ be the time from the beginning of a pseudo-cycle k until the first processor initiation on P_p. Then $\phi_1(k)$•P is defined as the sum of all $d_p(k)$'s. (For example, in Figure 2(b), it is equal to $d_1(k) + d_2(k)$). Note that there are up to $(P-1)$ nonzero $d_p(k)$'s.

- Part 2: $\phi_2(k)$•P. This is the total "useful" work in the pseudo-cycle including the first full execution of each task. (For example, in Figure 2(b), it is the total processor time covered by T_0 through T_4).

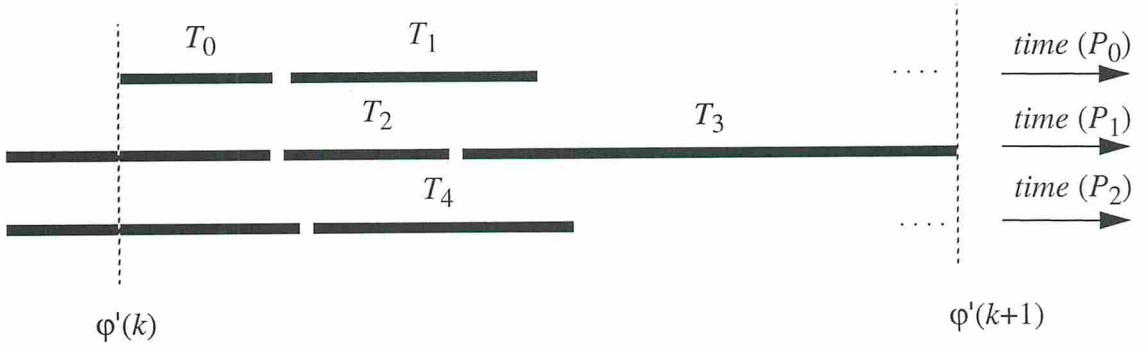
- Part 3: $\phi_3(k)$•P. This is defined as the sum of all $e_p(k)$'s, where $e_p(k)$ is the time wasted by P_p after the completion of its useful work in the k-th pseudo-cycle.

In the rest of the paper, ϕ_1, ϕ_2, ϕ_3, d_p, and e_p denote the average quantities taken over k. Each of these three parts of the pseudo-cycle must be estimated, either analytically or by simulations.

Figure 2. Iteration time and pseudo-cycle time



(a) Synchronous Iteration



(b) Asynchronous Iteration

3.2. Probabilistic Analysis

In this section, we use the approach in [19] to derive estimates of ϕ', I and S. For the analytical derivations, the following restriction on the distribution of the task intervals must hold and will be assumed throughout the section.

Assumption 5. *The task interval lengths are independent and identically distributed (i.i.d.) random variables drawn from the same distribution function with increasing failure rate (IFR) with mean* μ *and variance* σ^2.

A distribution function $G(x)$ is said to be IFR if $G(0) = 0$ (i.e., it is the distribution function of a positive random variable) and if for all $z_0 > 0$

$$\frac{1 - G(x + x_0)}{1 - G(x)} \text{ is monotone decreasing in } z.$$

When G has a density g then this is equivalent to

$$\frac{g(x)}{1 - G(x)} \text{ is monotone increasing in } z.$$

IFR distributions include: Exponential, Gamma with $\mu/\sigma \geq 1$, Weibull, Truncated Normal (i.e., Normal constrained to be positive), Uniform on the interval $(0, A)$ for any $A>0$, and Constant $= A$ for any $A>0$.

In practice a task interval is made of a succession of operations. The length of the task interval is the sum of the (random) times taken by each individual operation. In computations that are not data dependent, such as iterative solutions of linear systems of equations, the distribution of the task interval lengths tends to a truncated normal distribution as a consequence of the central limit theorem. (We can consider each task interval as a batch of small tasks as in [19].) When the fluctuations are dependent on the data, Assumption 5 may be difficult to justify. It may not even be appropriate to assume that task interval lengths are random.

3.2.1. Synchronous Case

When Q=P, the average synchronous iteration time is the mean of the largest order statistics among P random variables [10]. In general, let us denote the expected value of the maximum of P independent variates (the P-th order statistic) each with cumulative distribution function $G(x)$ by $X^{(1)}{}_{P.P}$ (the superscript will be clarified later). Then,

$$X^{(1)}{}_{P.P} = E\left(Max\left(x_1, x_2, \ldots, x_P\right)\right)$$

Assuming to simplify that $G(x)$ is continuous (the argument also hold in the case of a discrete distribution), the cumulative distribution function of the P-th order statistic is

$$F_P(t) = Prob\{all\ x_i \le t\} = G^P(t)$$

and since x is nonnegative,

$$X^{(1)}{}_{P.P} = \int_0^\infty t dG(t) = \int_0^\infty \left[1 - G^P(t)\right] dt$$

Consider a family of distributions $G(t)$ with mean μ and variance σ^2 such that $G(t) = G^o\left(\frac{t-\mu}{\sigma}\right)$. Then,

$$X^{(1)}{}_{P.P} = \int_0^\infty \left[1 - G^{oP}\left(\frac{t-\mu}{\sigma}\right)\right] dt = \mu + \sigma \int_0^\infty \left[1 - G^{oP}(y)\right] dy \text{ and}$$

$$X^{(1)}{}_{P.P} = \mu + \sigma \times O_{P.P} \tag{3}$$

where $O_{P.P}$ is the P-th order statistics [10] among P samples drawn from the interval length distribution with mean 0 and variance 1.

For a **uniform** distribution, $O_{P.P} = \sqrt{3} \times \frac{P-1}{P+1} \approx \sqrt{3}$ for large values of P. For an **exponential** distribution with a positive offset, $O_{P.P} = \sum_{1 < k \le P} \frac{1}{k} \approx \log P - 0.42$ for large values of P.

For a **normal** distribution, no analytical formula exists but $O_{P.P}$'s can be found in tables or by recurrence relations [10]. Some bounds exist; in particular, for very large values of P, $O_{P.P} \approx \sqrt{2\log P}$.

Since the normal distribution can take negative values we need to truncate the distribution at 0 in order to avoid generating negative task interval lengths. Unfortunately, the family of distributions generated by truncating a normal distribution with different values of μ and σ^2 does not satisfy the condition leading to (3). $O_{P.P}$ or $X^{(1)}{}_{P.P}$ must be estimated with a simple simulation program repetitively generating P samples from the distribution and taking the average of the maximum values.

When $Q>P$, the exact value of the average synchronous iteration time cannot be obtained analytically in general. We use a similar approach as in [19] to obtain an approximation. We divide the iteration time into two parts: T_1 and T_2. T_1 is the time between the start of the iteration and the start of the last task interval in that iteration and T_2 is the rest of the iteration time until the end of the last task interval. Since all processors are busy until T_1, since the distribution is IFR, and since the total work in each iteration must be less than the work done up to T_1 plus $P\mu$ we have that $Q\mu \le P\mu + PE(T_1)$ and therefore $\frac{Q-P}{P}\mu \le E(T_1)$. On the other hand, $E(T_2) \le X^{(1)}{}_{P.P}$. We approximate the expected value of the synchronous iteration time by

$$E(I) \approx \frac{Q-P}{P}\mu + X^{(1)}{}_{P.P} \tag{4}$$

Given a distribution of the interval lengths, (4) is correct for the case $Q=P$, but is an approximation otherwise [11, 24]. The first term of (4) is a low estimate of the time until the start of the last task, and the second term is a high estimate of the time between the start of the last task and the completion of the last task. Depending on which term is dominant (4) may therefore underestimate or overestimate the time taken by a synchronous iteration. We have run a large number of simulations and have stressed the model with very large values of P, Q and c_v, the coefficient of variation[2]. Some of these results are reported in the appendix and in the figures, (4) is always

within 10% of the simulated synchronous iteration times.

3.2.2. Asynchronous Case

Since different scheduling of the tasks yields different asynchronous outcome, we restrict the scheduling in the analytic model and in most simulations, as spelled out by Assumption 6.

Assumption 6. *The scheduling policy satisfies the following requirement. When a processor becomes free at any time instance* τ *in a pseudo-cycle k, it always selects for the next execution a task* T_q, *which has not yet started in the k-*th *pseudo-cycle, unless all the tasks have been started.*

This is a *fairness* condition imposed on the scheduling policy. We call this scheduling policy *age scheduling*. Age scheduling can be implemented by time-stamping each task descriptor in the scheduling queue with the time when the latest execution of the task was started and by assigning a higher priority level to the tasks with lower timestamps. This scheduling strategy ensures that each execution contributes to increasing the age of the iterate vector. It optimizes the speed of the asynchronous iteration and is very easy to enforce in a practical situation.

Under age scheduling, a pseudo-cycle is the time it takes to complete Q tasks. Remember that $d_p(k)$ is the time from the beginning of a pseudo-cycle k until the next task interval on P_p. The IFR condition assures that d_p is stochastically bounded by the average time it takes to complete a whole task [19]. Since there are up to $(P-1)$ $d_p(k)$'s for each k, an upper bound on the overhead at the beginning of a pseudo-cycle is given by $(P-1)$ task executions. Therefore, a pseudo-cycle takes no more than $(Q+P-1)$ task executions. Using the same decomposition into T_1 and T_2 as above for the synchronous case, we obtain

$$E(\phi') \approx \frac{P-1}{P}\mu + \frac{Q}{P}\mu + (X^{(1)}{}_{P:P} - \mu) = \frac{Q-1}{P}\mu + X^{(1)}{}_{P:P} \qquad (5)$$

This formula is an approximation for any value of P and Q. The decomposition in three terms

2. The coefficient of variation is the ratio between the standard deviation and the mean. We denote it by c_v. Therefore, $c_v = \sigma/\mu$.

reflects the contributions of ϕ_1, ϕ_2, and ϕ_3. The first term is the overhead due to the use of out-dated information in component updates, the second term is the useful work, and the third term is caused by fluctuations in the computation times.

Although approximate, (5) yields pseudo-cycle times that are mostly well within 10% of the simulated times, except when Q=P and when the coefficient of variation of the interval length distribution is very high. The reason is that the task interval starting the pseudo-cycle can be very short so that the processor allocated to it does useless work until T_1 (which is the time when the last useful task interval starts in the pseudo cycle). Therefore the estimate for $E(T_1)$ is too low and the error becomes larger than 10%, increasing with the coefficient of variation and the number of processors.

When Q=P and the coefficient of variation is high, we use a different approximation than (5), as follows. The useful work in each pseudo-cycle is the task of the processor starting the pseudo-cycle plus one task from every other processor which must first complete its task started before the beginning of the pseudo-cycle. Because of the IFR condition, the time taken by each processor to complete its useful work in the pseudo-cycle is upper bounded by the time taken by two consecutive task intervals. Therefore:

$$E(\phi') \leq X^{(2)}_{P,P} \tag{6}$$

where the superscript (2) indicates that the order statistic is for the variate equal to the sum of two random samples drawn from the distribution of task interval lengths.

3.2.3. Slowdown Factor

Combining (4) and (5) we obtain

$$S \approx 1 + \frac{(P-1)\mu}{(Q-P)\mu + PX^{(1)}_{P,P}} \leq 1 + \frac{P-1}{Q} \tag{7}$$

This approximation on the slowdown factor is always greater than 1 but less than 2. It decreases as σ increases and its maximum value of $1 + (P-1)/Q$ is reached for small values of the fluctuation, so that $P\mu \gg PX^{(1)}{}_{P.P}$. For very large fluctuations, the slowdown factor is close to 1.

The upper bound in (7) is an upper bound on an approximation. In fact, for $Q>P$, it is not possible to prove that the upper bound on the slowdown is 2, because of the lack of a good lower bound for the synchronous iteration time. However, for $Q=P$ (an important case in practice), we know the exact value of the synchronous iteration time. From (6), we then conclude:

$$S \leq \frac{X^{(2)}{}_{P.P}}{X^{(1)}{}_{P.P}} \leq 2 \tag{8}$$

That $X^{(2)}{}_{P.P} \leq 2X^{(1)}{}_{P.P}$ can be easily understood from the following argument. The simulation to obtain $X^{(1)}{}_{P.P}$ draws consecutive batches of P samples and selects the maximum value in each batch. The simulation to obtain $X^{(2)}{}_{P.P}$ draws two batches of P samples from the same distribution and adds them pair-wise. Clearly, the sum of the maxima in two consecutive batches of P samples cannot be less than the maximum of the values obtained by adding the samples in the same batches pair-wise.

3.3. Simulations

We have run simulations to verify the accuracy of (4), (5), (6), (7) and (8). The simulations were run for three stochastic distributions of the task interval lengths. The first distribution is a truncated normal: we draw a sample from a normal with mean 1 and variance σ; if the sample is less than zero, we set it to zero. To stress the model, we have used very high coefficients of variation of the original normal distributions, from 0.01 to 100. The second distribution is a uniform distribution over [0, 2) with no offset. The third distribution is an exponential with no offset and with mean equal to 1. The number of processors was varied from P=2 to P=2,048.

The simulations use Assumptions 3 and 4 (strong coupling). For $Q=P$, the tasks are allo-

cated statically to processors. For $Q>P$, the tasks are allocated dynamically, according to age scheduling. Every time a processor is released, the set of tasks is scanned to find the current value of *age* $A(x)$ (the minimum value of all the tasks' ages). One of the tasks with the lowest age value is allocated to the processor. When the task interval is completed on the processor the age of the task is set to its age at the start of the interval plus one. We run the simulations for 1,000 consecutive iterations (synchronous case) and for 1,000 pseudo-cycles (asynchronous case). Because of the stochastic behavior, both estimates may fluctuate by 1 or 2%, depending on the sequence of random numbers. These simulations are very computation intensive (of the order of Q^2) so that it is impossible to run simulations with very large number of processors and tasks.

We have also derived the analytical models. When $Q>P$, we always use the models of (4), (5) and (7); $X^{(1)}_{P,P}$ is found by the formulas for the exponential and uniform distributions whereas for the truncated normal distributions we derive the order statistic through simulation. We also use the same model for $Q=P$ and for the truncated normal distributions with low coefficients of variation (c_v=0.01, 0.1, or 0.3). For $Q=P$ we use the models of (4), (6) and (8) for the uniform, exponential, and truncated normal distributions with high coefficient of variation (c_v=1, 5, 10, or 100). In these models, $X^{(2)}_{P,P}$ is obtained by taking the average of the maxima in 1,000 batches of P samples; each sample is the sum of two independent samples drawn from the interval length distribution (The complexity of this computation of the order of Q.)

The Appendix shows the comparison between model and simulation across a large number of possible systems. As can be seen the error on the synchronous iteration time and on the asynchronous pseudo cycle is never higher than 11%. However, the relative error on S may be much more than 10% because it is the ratio of two estimated values. In particular, this occurs for $Q=P$ and c_v= 5, 10, or 100, where the error on S reaches 15%. These coefficients of variations are very high though and the values are included to show the performance of the model in extreme cases.

Figure 3. Pseudo-cycle time for P=64 and truncated normal distribution

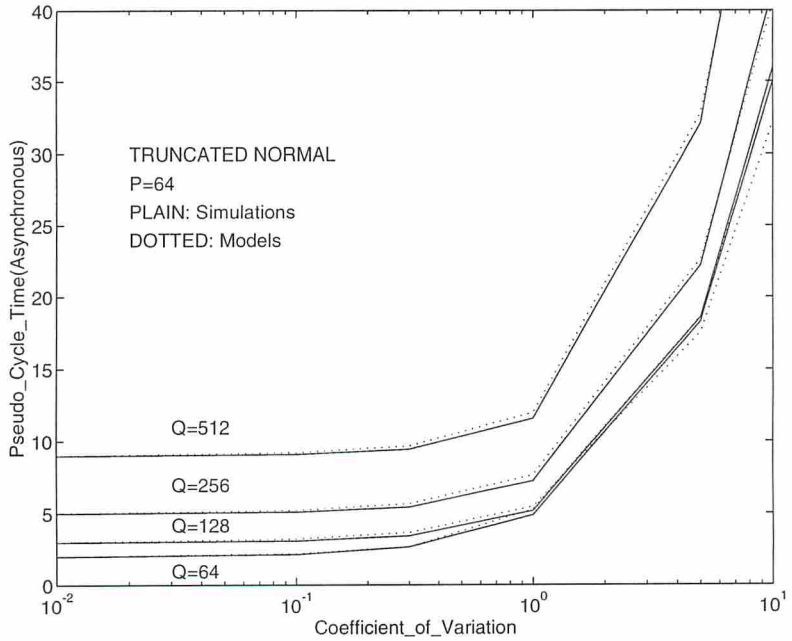
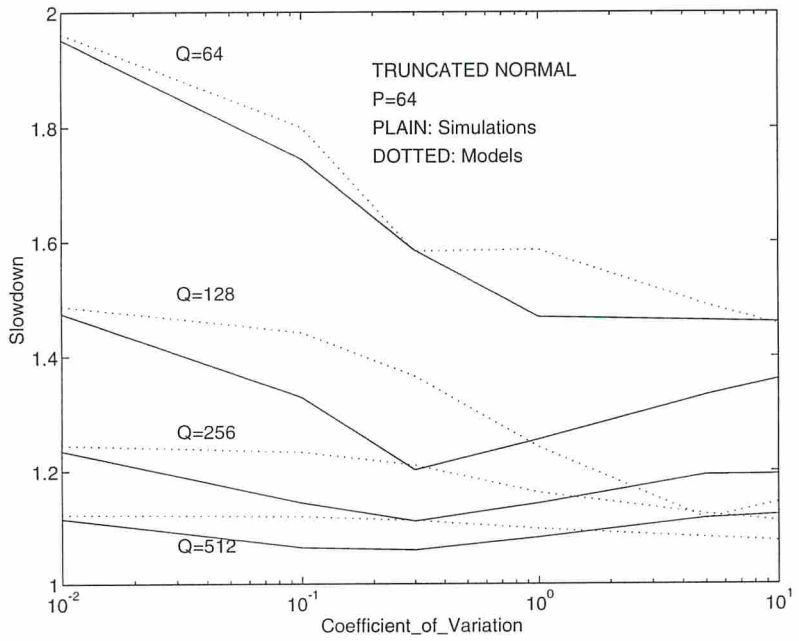


Figure 3 shows that the estimated pseudo-cycle time predicted by the model is very close to the ones obtained by simulation and Figure 4 displays the slowdown factor for the same cases as in Figure 3.

Figure 4. Slowdown factor for P=64 and normal distribution



The simulation results agree with the conjecture that the slowdown is upper-bounded by 1 + P/Q. The model is not as good as for the pseudo-cycle times because the error of the slowdown

factor sometimes accumulates the error made in the estimation of both the synchronous iteration time and the asynchronous pseudo-cycle time when $Q>P$. Large errors (15%) are observed for Q=128 and $c_v>1$. Such large coefficients of variation (which imply that the fluctuation can be more than three times the mean) are unlikely in practice. The upper bound on the slowdown factor decreases with the number of tasks and with the coefficient of variation. As Q/P increases, the behaviors of the synchronous and asynchronous iterations tend to converge and the slowdown factor goes down to 1. The worst-case value for the slowdown factor is close to 2, for $P=Q$ and $c_v\approx 0$. This is because the pseudo-cycle time is very sensitive to small fluctuations of the task interval lengths whereas the synchronous iteration time is not.

Figure 5. Slowdown factor as a function of P (Q=P)

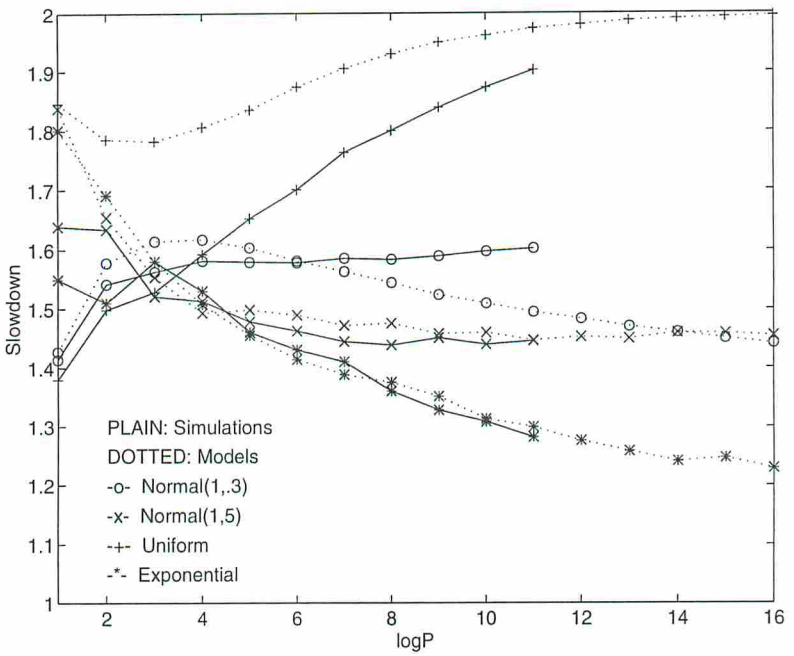


Figure 5 shows the slowdown factor as a function of the number of processors and for different distributions for the case of $Q=P$. The simulations (up to 2K processors) are compared to the models (up to 64K processors). We see that the model does not always agree with the simulations for small numbers of processors. The model is worse for the uniform distribution and best for the exponential distribution. Significant differences in the trends of the curves exist among the three distributions, which indicates that a single model based on a distribution-free parameter such as the coefficient of variation cannot explain all behaviors. All curves tend towards a con-

stant less than two as the number of processors increases.

Figure 6. Synchronous iteration times as a function of P (Q=2P)

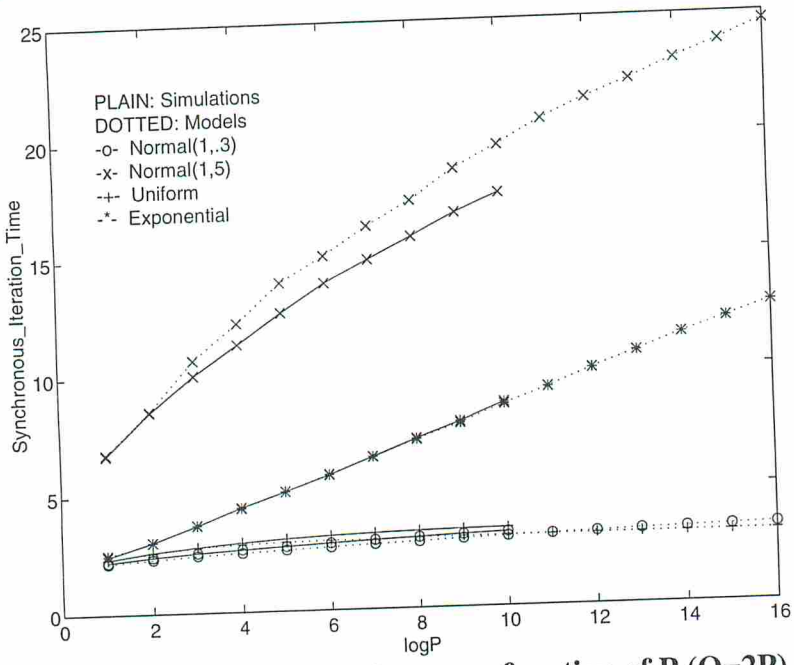
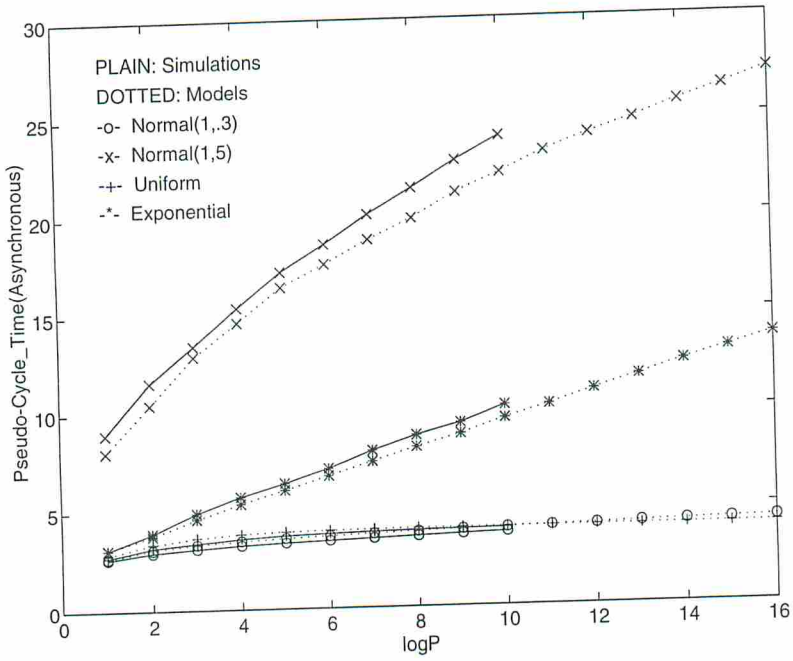


Figure 7. Pseudo-cycle times as a function of P (Q=2P)



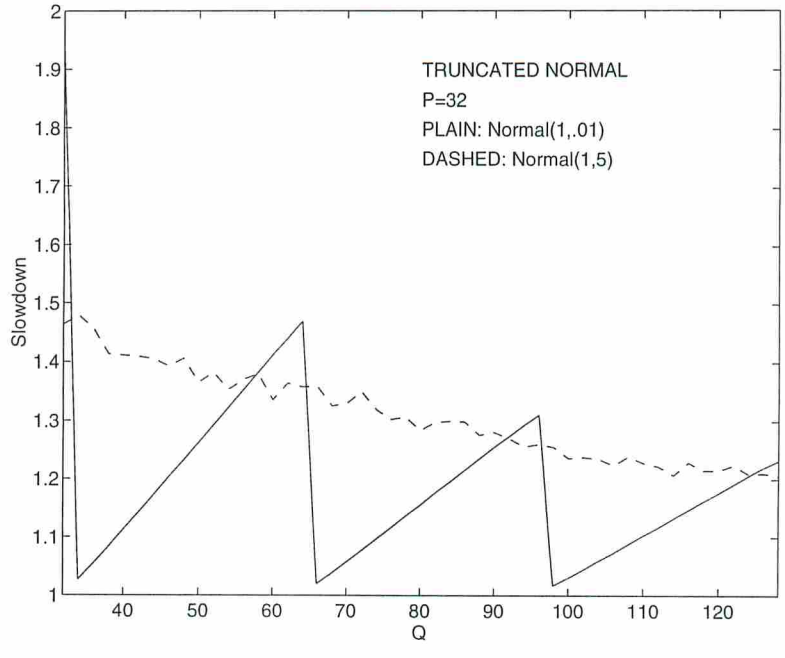
As shown in Figures 6 and 7, the synchronous iteration time and the pseudo-cycle time are predicted very accurately by the models for the case where $Q=2P$. From these figures, we see, that for distributions with lower coefficients of variations (such as the uniform and the truncated normal with $c_v = .3$), the fluctuation has very little effect on the synchronous iteration time and on

the pseudo-cycle times. For distributions with large coefficients of variation (such as the exponential and the truncated normal with c_V= 5) the effect of the fluctuation dominates. The computation times in the synchronous and asynchronous cases increase at a logarithmic rate with *P*. However, the slowdown factors (not shown) remain roughly constant as a function of *P*, within a band between 1.15 and 1.35. Because the range is so narrow, the models are not very successful at capturing the small differences between the slowdown factors.

3.4. Effect of the Number of Tasks

As the number of tasks increases for a given number of processors, the difference between the time taken by the synchronous and asynchronous iterations narrows. The variations in the slowdown factor as a function of *Q* varies markedly with the coefficient of variation of the task interval lengths.

Figure 8. Slowdown factor as a function of Q (P=32)



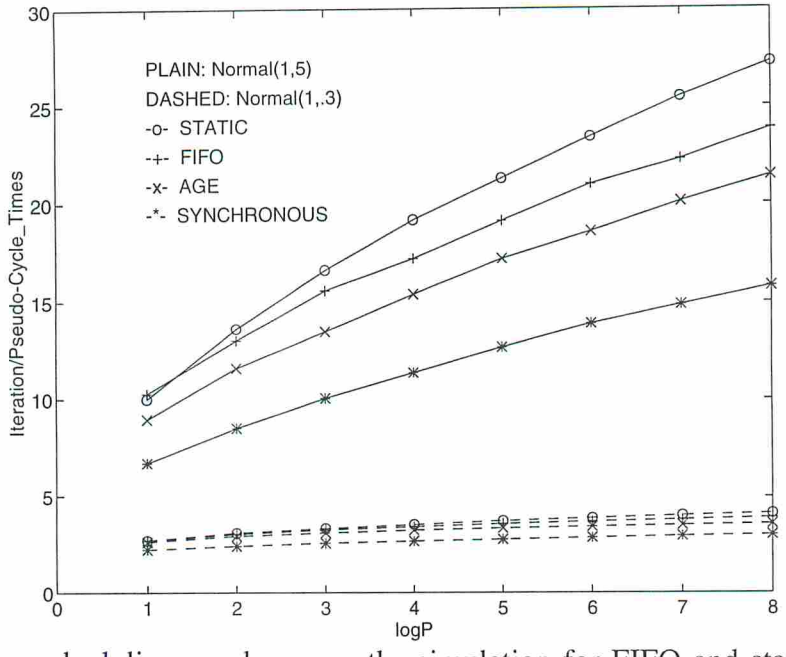
In Figure 8, integer values of *Q/P* correspond to good load balancing, in which case the efficiency of the synchronous iteration is best. This explains the peaks at integer values of *Q/P* in Figure 8, for the case of a coefficient of variation equal to 0.01. When the coefficient of variation is large however, the peaks disappear because the effects of the variations in task interval lengths

dominate.

3.5. Effect of the Task Scheduling Policy

In all the simulations above, the next task to run is the one which currently does not run and has the lowest age, according to Assumption 6 (age scheduling). In practice, the task scheduling algorithm may be different. It is not difficult to come up with scheduling strategies which could considerably slow down or even stall completely the convergence of the asynchronous iteration.

Figure 9. Effects of Scheduling



Besides age scheduling, we have run the simulation for FIFO and static policies. In the FIFO policy, there is a global queue of tasks. When a processor is available, it selects the task from the head of the queue and it removes it from the queue. When the task interval is complete the task descriptor is added to the tail of the queue. The FIFO policy is different from age scheduling because the priorities in the scheduling queue are based on the time when tasks join the queue. In the static scheduling policy, each processor is assigned Q/P tasks and executes them in turn. This policy has the lowest runtime scheduling overhead. As shown in Figure 9, differences in scheduling have little effect on the convergence rate of the asynchronous algorithm when the coefficient of variation of the execution times is low. However, for large coefficients of varia-

tions, the differences among scheduling strategies become significant.

3.6. Global and Local Computations

The definition of $\{\varphi'(k)\}$ assumes that the computed values in each task are released at the end of the task (Assumption 3). This implies that the task intervals falling on the pseudo-cycle boundaries are wasted. In many cases, however, it is possible to rearrange the computations so that the critical components that will be used by other tasks are computed and released as soon as possible, before the components that will not be used by other tasks. This reduces the number of wasted tasks.

Example 1. Consider the discretized approximation of the Laplace equation

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

Discretization yields a rectilinear grid with boundary conditions. Each point of the grid is updated successively using the following iteration formula:

$$u_{i,j} = \frac{1}{4} \times [u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}]$$

The next value of each point is computed by taking the average of its four neighboring points. We can partition the grid into rectangular regions and order the components such that each task first computes and updates the boundary points of its corresponding region (global computation phase), and then processes the interior points (local computation phase). ❑

Since the local computation phase of a task does not interact with other tasks, it should be allowed to overlap with the previous or the next pseudo-cycles. We define the following sequence $\{\varphi''(k)\}$ such that all the components reach the solution no later that $\varphi''(M)$.

Definition 2. $\{\varphi''(k)\}$ *is an increasing sequence of time instances such that for all k,* the interval $(\varphi''(k), \varphi''(k+1)]$ *covers at least one* global *computation phase of each task.*

In other words, local computation phases can be considered as idle times in which no significant global work is done. Obviously, $\varphi''(k) \leq \varphi'(k)$ for all k. It is also clear that for a given task

scheduling and task interval lengths, reducing the size of the global computation phases can only reduce $\varphi''(k)$ and therefore the execution time.

4. PARTIALLY COUPLED ITERATION OPERATORS

The execution times of asynchronous algorithms estimated so far are at least as large as those of their synchronous counterparts. The reason is that we have assumed that the execution of a task depends critically on the outcome of *all* the tasks executed in the previous pseudo-cycle, in order to make any progress towards the solution. In general however this is very pessimistic because there are rarely dependencies between all pairs of tasks.

4.1. General Model

Let $S_q(k)$ be the set of tasks on which the computation of T_q depends in the k-th pseudo-cycle in order to increment its age (including T_q itself).

Assumption 7. *For all x and i the iteration operator F satisfies*

$$A_i(F_i(x(k))) = \min_{j \in S_i(k)} \{A_j(x_j(k))\} + 1$$

Taking this limited coupling into account, we can slightly modify the definition of the pseudo-cycle sequence to obtain lower upper bounds.

Definition 3. *For all q,* $\{\varphi^{(q)}(k)\}$ *is defined as a non-decreasing sequence of time instances, starting with the initial time, such that the interval* $(\beta^{(q)}(k), \varphi^{(q)}(k+1)]$ *covers at least one global computation phase of the task* T_q, *where*

$$\beta^{(q)}(k) = \max_{j \in S_q(k)} \{\varphi^{(j)}(k)\}$$

After $\varphi^{(q)}(M)$, the iterates updated by task T_q indefinitely enter domain $X(M)$, whereas the algorithm with barrier synchronization takes M iterations to enter domain $X(M)$. The proof is

omitted since it is very similar to the proof of Lemma 1.

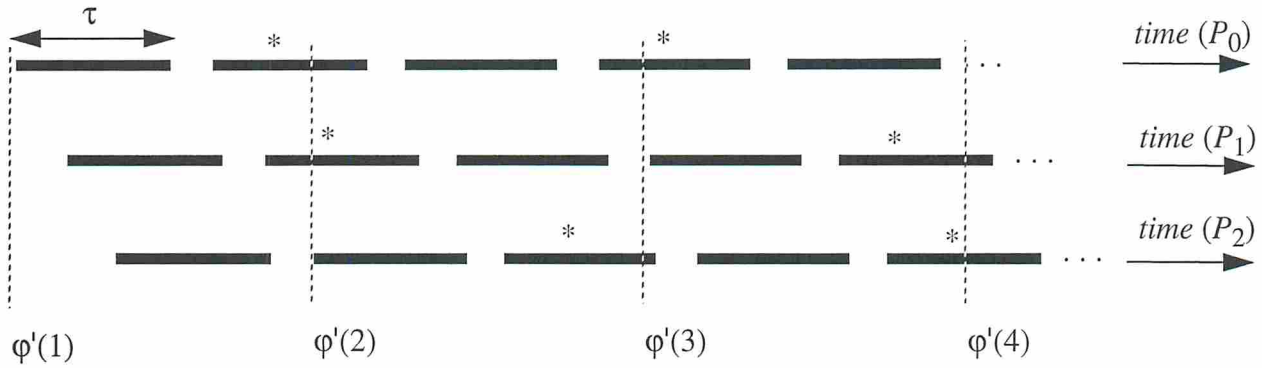
If the time between $\beta^{(q)}(k)$ and the beginning of the first execution of T_q after $\beta^{(q)}(k)$ is denoted by $r^{(q)}(k)$ and if $g^{(q)}(k)$ is the duration of this first execution of T_q, then

$$\varphi^{(q)}(k+1) = \underset{j \in S_q(k)}{Max} \{\varphi^{(j)}(k)\} + r^{(q)}(k) + g^{(q)}(k)$$

and the total execution time will be $T = \underset{q}{Max} \{\varphi^{(q)}(M)\}$.

When task interval lengths fluctuate, the convergence rate of the asynchronous iteration will be very high provided the average distance $r^{(q)}(k)$ is kept large so that task intervals of tasks in $S_q(k)$ have very small probability to overlap with the following task interval of task T_q. This is the case, for example, when $Q>P$.

Figure 10. Example of partial coupling



Example 2. Consider the computation in Figure 10. There are three data iterates and each iterate x_i is statically assigned to P_i. Each task interval length is constant and equal to τ. Assuming the worst-case scenario, F_0 cannot advance the iterate vector towards convergence without a recent value of x_2, and the second task interval of the first component (x_0) does not contribute to the pseudo-cycle from φ'(1) to φ'(2). This is because the definition of φ'(k) assumes the worst case in which F_0 cannot advance the iterate towards convergence without a recent value of x_2. In this way, 2 out of 5 global computation intervals of each component are wasted and the average pseudo-cycle time is (5/3)τ. The wasted tasks are marked by * in the figure. Now, suppose that the dependencies among the components are restricted as follows.

$x_0 = F_0(x_0)$
$x_1 = F_1(x_0, x_1)$
$x_2 = F_2(x_0, x_1, x_2)$

Since F_0 only depends on itself, the computation of x_0 in a pseudo-cycle may overlap with the computation of x_2 in the previous pseudo-cycle. As a result, every computation of x_0 makes progress towards the solution. Similarly, every computation of x_1 and x_2 also makes progress. Consequently, the average minimum pseudo-cycle time is upper bounded by τ. ❑

Example 1 suggests that the sequence specified in Definition 3 sometimes yields a much smaller bound on the execution time than $\{\varphi'(k)\}$, and it can be concluded that, when the coupling is weak, the pseudo-cycle time of an asynchronous implementation may be much less than the iteration time of its synchronous counterpart.

4.2. Self-Coupled Iteration Operators

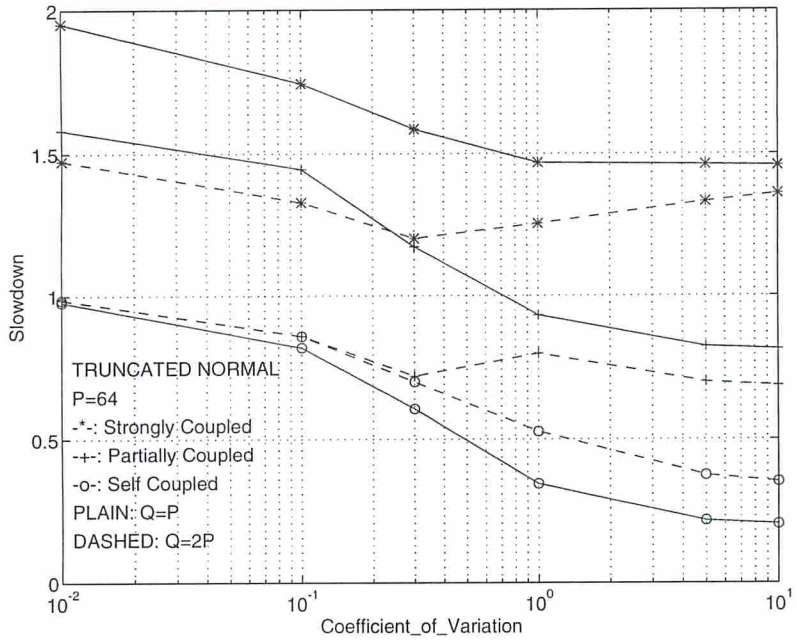
An optimistic approach to estimate the execution time of an asynchronous iteration is to assume that every task execution makes progress towards the solution. This approach was taken in [3], [10], and [11] and is equivalent to assuming that each task is only coupled with itself, i.e., $S^{(q)}(k) = \{q\}$, for all q and k. Let M be the total number of iterations to reach the solution. Then, an asynchronous execution contains exactly QxM task intervals. Using the same approach as for the derivations of (5) and (6)

$$M \times E(\phi) \approx \frac{M \times Q - 1}{P} \times \mu + X^{(1)}_{P:P} \approx \frac{M \times Q}{P} \times \mu.$$

Figure 11 compares the slowdown factor for strongly, partially and self coupled iterations using simulations. In all cases we assume that iterates are read at the start of a task interval and are released at the end of a task interval. For the self-coupled iterations, the age of a task is always incremented by one at the end of each task interval. For the partially-coupled iteration, we assume that each task T_q depends on components produced by its two neighbors, $T_{(q-1)}$ and $T_{(q+1)}$. Therefore, the age of the iterates updated by T_q is computed at the beginning of the task interval as the minimum of the ages of T_q, $T_{(q-1)}$ and $T_{(q+1)}$ and is incremented at the end of the task

interval. We always schedule first one of the tasks with the lowest age (age scheduling).

Figure 11. Comparison of iterations with various degree of coupling (P=64)



As can be seen from the figure, when $Q=P$, the maximum slowdown of the partially coupled iteration is lower than one for $c_v > 0.7$. When $Q=2P$, the maximum slowdown of the partially coupled iteration is less than one, for all coefficients of variation, and for $c_v < 0.3$., it is as good as for the self coupled iteration. Of course, the self-coupled iteration is always the most efficient and its slowdown is always less than one.

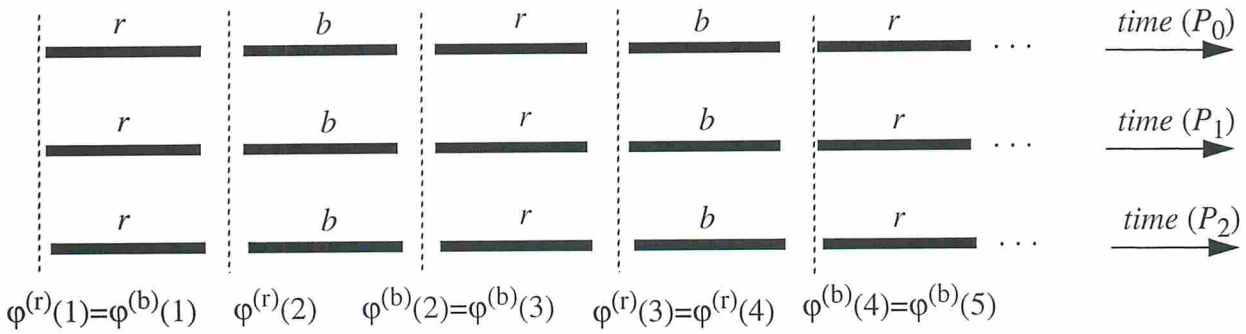
4.3. Colorable Iteration Operators

Definition 3 does not rule out the possibility of pseudo-cycles of length zero. We illustrate this possibility by a popular scheme to order component updates, called red-black ordering, in the following example.

Example 3. In red-black ordering, each component has a color, red or black, and the computation of the red (black) components depends only on the values of the black (red) components. This computational scheme is shown in Figure 12 for 3 processors and 6 tasks. Three of the tasks update red components and the other three tasks update black components; they are labelled by r or b (respectively) in the figure. All task intervals are constant and equal to τ. $\varphi^{(r)}(k)$ ($\varphi^{(b)}(k)$) is equal to $\varphi^{(q)}(k)$ given in Definition 2 for red (black) tasks. The first phase computes the red com-

ponents and makes one unit of progress. The second phase updates the black components using the latest value of the red components and therefore also makes one unit of progress. Effectively, one unit of progress is made in each time period of τ, which means that the average pseudo-cycle time is τ, half the time given by the analysis in Section 3. (According to the definition of $\{\varphi'(k)\}$ a pseudo-cycle should cover all the tasks and in this example should have a length of 2τ.) ❑

Figure 12. Red-black ordering



We can generalize the example as follows. Let $\mathcal{S} = \{\mathcal{T}_0, \mathcal{T}_1, ..., \mathcal{T}_{R-1}\}$ be a partition of the set of all tasks $\mathcal{T}$ such that the tasks in $\mathcal{T}_i$ are only coupled with the tasks in $\mathcal{T}_{(i-1) mod\ R}$, for all $0 \leq i \leq R-1$. Iteration operators for which the components can be partitioned in this fashion are called *colorable* and the tasks in $\mathcal{T}_C$ have color C. In the case of colorable operators, it is sufficient that the k-th pseudo-cycle covers the tasks with color $k\ mod\ R$. For such pseudo-cycle sequences all tasks with color $k\ mod\ R$ make at least k units of progress in k pseudo-cycles. This partitioning of the set of tasks into R subsets in effect reduces the number of tasks a pseudo-cycle has to cover. An average pseudo-cycle has to cover Q/R tasks instead of Q.

4.4. Discussion

In the case where the sets $S^{(q)}(k)$'s are constant in the whole data domain and are detectable by simple dependency analysis, it is not necessary to use a global barrier synchronization in the synchronous algorithm. Furthermore, if the sets $S^{(q)}(k)$ are small, it is unfair to compare an asynchronous iteration to its synchronous version with global barrier synchronization. Rather, in the synchronous computation, a task T_q should only wait for the completion of the tasks in the set $S^{(q)}(k)$ in order to start its next execution, as advocated in [16]. One problem with this approach is

the complexity of synchronizing and scheduling tasks to take advantage of this opportunity.

Another problem arises when the sets $S^{(q)}(k)$'s are small but dynamically changing during the course of the computation. For example, in the consistent labeling problem, dependencies among components are data dependent [17, 26]. In such a situation, the sets $S^{(q)}(k)$ are not known a priori and it is not possible to use the restricted form of synchronization.

We can conclude that the elimination of synchronization points may improve the performance of iterative algorithms significantly when the sets $S^{(q)}(k)$ are small. Unfortunately, we do not yet know how to measure and estimate the sets $S^{(q)}(k)$ in all cases. The worst case is strong coupling. The best case is self coupling. A particular iteration will fall in between these two extremes. As shown in Figure 11 the slowdown factor of all partially coupled asynchronous iterations with an operator satisfying Assumption 2 and under age scheduling is in the band delimited by the curves for the simulations under strong and self coupling.

5. CONCLUSION

We now summarize the results derived in this paper. It should be emphasized that we have made a basic assumption on the iteration operator F, since it is not possible to obtain results on the convergence rate of neither asynchronous nor synchronous executions for arbitrary iteration operators. The class of iteration operators considered in this paper includes monotonic operators.

Our computational model allows for the evaluation of asynchronous algorithms in which Q, the number of tasks, may be greater or equal to P, the number of processors. When $Q=P$, the allocation of tasks to processors is static and each processor executes a different task. When $Q>P$ the scheduling of tasks is dynamic and it is possible to design scheduling strategies that will make the convergence of the asynchronous algorithm arbitrarily slow. In our study we have concentrated on the scheduling strategy that maximizes the convergence rate of the asynchronous implementation and which we have called *age scheduling*.

We have developed a simulation approach to evaluate the convergence speed of the asyn-

chronous iteration, under various conditions of coupling among iterate components, from strong coupling to self coupling. These two types of coupling yield an upper and a lower bounds (respectively) on the time taken by an asynchronous iteration. Under strong coupling, the asynchronous iteration cannot converge faster than its synchronous counterpart whereas under self coupling it always does. When the coupling is intermediate between self and strong, the convergence rate of the asynchronous iteration may be better, depending on the size of the fluctuations of the task execution times.

Simulations are quite complex and limit the size of the multiprocessor we can evaluate. To provide insight into the simulation data and to extrapolate the simulation results, we have also developed some analytical models which assume that task execution times are random and independent and drawn from the same IFR distribution. Under these conditions we proved that the asynchronous iteration under strong coupling and with age scheduling is at most twice slower than its synchronous counterpart when $Q=P$. When $Q>P$, we could not prove the same property. However, an approximate model validated in the paper by extensive simulations shows that the asynchronous algorithm can be up to $1 + P/Q$ times slower than its synchronous version. This upper bound is reached when the fluctuations of the task execution times are very small. Therefore, in the important case where each processor is statically allocated to one task, and where all tasks are roughly equal in size (good static load balance) it is very difficult to obtain a good speedup through asynchronism.

The simulations covered uniform, exponential and truncated normal distributions for a wide range of coefficient of variation and of numbers of processors. When P increases we have observed that the ratio of the convergence rates of the asynchronous and synchronous algorithms under strong coupling converges to a constant between 1 and 2 which varies with the execution time distribution. This is a very interesting observation: As P increases, the overhead of synchronization is likely to dominate and, provided the ratio of the convergence rates does not increase, there should be a system size for which the asynchronous algorithm is better than the synchronous

one, even under worst-case conditions.

An asynchronous iteration executes faster by delaying the fetching of input data for component updates and by releasing the updated data earlier. As a consequence, we can design more efficient algorithms by ordering the component updates in a task, such that the components needed by other tasks are updated earlier than the ones that are only used locally.

When the dependency among tasks is weak and predictable, we do not need barrier synchronization to implement a synchronous iteration. In this case, only the dependent tasks need to be synchronized. A synchronous implementation with this type of synchronization seems to be a better choice than the asynchronous version. However, when the coupling is not predictable, barrier synchronization is unavoidable in a synchronous implementation. Therefore, the weak and unpredictable coupling exploits the performance advantage of an asynchronous implementation the most, which executes faster with decreasing coupling.

We have surveyed the (abundant) literature on computational experience with asynchronous iterations. The results of these papers are very difficult to relate to our models because they give total execution times, including all effects and do not contain the kind of information --such as statistics on the execution times of the tasks-- needed to make a valid comparison. Moreover, these papers give little insight to explain the observations. One possible comparison is to consider the number of tasks executed in the two versions of the same algorithm. Experiments where the asynchronous algorithm was reported to converge in less task executions than its synchronous counterpart are very rare, which tend to show that the strong coupling assumption may be more applicable than would seem at first. A notable exception was found in [10]. In this paper spectacular speedups --which cannot be explained by synchronization overheads alone-- are reported for some (non-monotonic) operators, even when coupling among iterates is strong. We are currently investigating these cases more closely.

The type of information needed to understand the fundamentals behind the convergence rate of asynchronous algorithms is impossible to obtain from an implementation on an actual

machine. Besides models such as the one developed in this paper, we believe that a better way to understand asynchronism in algorithms is to run the algorithms on architectural simulators, in which we can easily observe all timings. The understanding provided by such simulations would allow one to make a decision as to the best synchronization scheme to use in a particular situation. This is one path that our future research in this field will take.

6. REFERENCES

[1] Axelrod, T.S.,"Effects of synchronization barriers on multiprocessor performance," *Parallel Computing*, 3:129-140, 1986.

[2] Baudet, G.M., "Asynchronous iterative methods for multiprocessors," *Journal of the Association for Computing Machinery (JACM)*, 25(2), pp. 226-244, April 1978.

[3] Brochard, L., Prost, J.-P., and Fauire, F.,"Synchronization and load unbalance effects of parallel iterative algorithms," *Proc. of the Int. Conf. on Parallel Processing (ICPP)*, pp. 153-160, August 1989.

[4] Bertsekas, D.P., "Distributed dynamic Programming, *IEEE Transaction on Automatic Control*, AC-27, pp. 610-616, 1982.

[5] Bertsekas, D.P., "Distributed asynchronous computation of fixed points," *Mathematical programming,* 27, pp. 107-120, 1983.

[6] Bertsekas, D.P., and Tsitsiklis, J.N., "Convergence rate and termination of asynchronous iterative algorithms," *Proc. of the Int. Conf. on Supercomputing*, pp. 461-470, June 1989.

[7] Bertsekas, D.P., and Tsitsiklis, J.N. "Parallel and Distributed Computation," Prentice-Hall, 1989.

[8] Bull, J.M. and Freeman, T.L. "Numerical performance of an asynchronous Jacobi iteration." *Second Joint International Conference on Vector and Parallel Processing, CONPAR92-VAPP-V*, Springer-Verlag, pp. 361-366, 1992.

[9] Chazan, D., and Miranker, W., "Chaotic relaxation," *Liner Algebra and its Applications*, Vol. 2, pp. 199-222, 1969.

[10] David, H.A., "Order Statistics," John Wiley and Sons, 1970.

[11] Dubois, M., and Briggs, F.A., "Performance of synchronized iterative processes in multiprocessor systems," *IEEE Trans. of Software Eng.,* Vol 8(4), pp. 419-431, July 1982.

[12] Dubois, M., and Briggs, F.A., "The runtime efficiency of parallel asynchronous algorithms, *IEEE Trans. on Computers*, Vol. 40(11), pp. 1260-1266, November 1991.

[13] El Baz, D., "M-functions and parallel asynchronous algorithms," *SIAM Journal on Numerical Analysis*, 27, pp. 136-140.

[14] El Tarazi, M.N., "Some convergence results for asynchronous algorithms," *Numerisch Mathematik*, 39, pp.325-340, 1982.

[15] Frommer, A., "On asynchronous iterations in partially ordered spaces," *Numer. Funct. Anal. and Optimiz.*, 12(3&4), pp. 315-325, 1991.

[16] Greenbaum, A., "Synchronization costs on multiprocessors." *Parallel Computing*, Vol. 10, pp. 3-14, 1989.

[17] Haralick, R.M., and Shapiro, L.G., "The consistent labeling problem: Part I," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 1(2), pp. 173-184, April 1979.

[18] Kung, H.T., "Synchronized and asynchronous algorithms for multiprocessors," in J.F. Traub, ed., Algorithm and Complexity: New Directions and Recent Results, Academic Press, New York, 1976.

[19] Kruskal, C.P. and Weiss, A, "Allocating independent subtasks on parallel processors," *Proc. of the Int. Conf. on Parallel Processing (ICPP)*, pp. 236-240, August 1984.

[20] Miellou, J.C., "Algorithmes de relaxation chaotiques a retards," R.A.I.R.O., R_1, pp. 55-82, 1975.

[21] Miellou, J.C., "Iterations chaotiques a retards, etude de la convergence dans le cas d'espaces partiellement ordonnes," C.R.A.S. Paris, 280, pp.233-236, 1975.

[22] Miellou, J.C., "Asynchronous iterations and order intervals," in Parallel Algorithms and Architectures, M. Cosnard ed., North Holland, pp. 85-96, 1986.

[23] Miellou, J.C., and Spiteri, P.,"Un critere de convergence pour des methodes generales de point fixe," R.A.I.R.O. MMAN, 19, pp. 645-669, 1985.

[24] Robinson, J.T., "Some analysis techniques for asynchronous multiprocessor algorithms," *IEEE Trans. on Software Eng.*, Vol. 5(1), pp. 24-31, January 1979.

[25] Tsitsiklis, N,T. and Stamoulis, G.D., "On the average communication complexity of asynchronous distributed algorithms," Technical Report, Laboratory for Information and Decision Systems, No. LIDS-P-1986, September 1990.

[26] Üresin, A., and Dubois, M. "Sufficient conditions for the convergence of asynchronous iterations," *Parallel Computing*, Vol. 10, pp. 83-92, 1989.

[27] Üresin, A., and Dubois, M., Parallel asynchronous algorithms for discrete data," *Journal of the Association for Computing Machinery (JACM)*, Vol. 37(3), pp. 588-606, 1990.

APPENDIX

Table 1: Simulation vs Model for a Truncated Normal Distribution, P=64

Q=	c_v=	I(simul)	I(mod)	ε(I)	ϕ'(simul)	ϕ'(mod)	ε(ϕ')	S(simul)	S(mod)	ε(S)
64	0.01	1.023	1.023	0.0	1.997	2.007	-0.5	1.952	1.962	-0.5
64	0.1	1.231	1.231	0.0	2.147	2.215	-3.2	1.744	1.800	-3.2
64	0.3	1.689	1.689	0.0	2.676	2.673	0.1	1.585	1.583	0.1
64	1.0	3.326	3.326	0.0	4.884	5.275	-8.0	1.468	1.586	-8.0
64	5.0	12.504	12.504	0.0	18.285	18.622	-1.8	1.462	1.489	-1.8
64	10.0	24.039	24.039	0.0	35.070	34.988	0.2	1.459	1.455	0.2
64	100.0	233.88	233.88	0.0	334.42	332.57	0.6	1.430	1.422	0.6
128	0.01	2.031	2.023	0.4	2.993	3.007	-0.5	1.474	1.487	-0.9
128	0.1	2.329	2.231	4.2	3.090	3.215	-4.1	1.329	1.441	-8.4
128	0.3	2.853	2.689	5.8	3.433	3.673	-7.0	1.202	1.366	-13.6
128	1.0	4.152	4.409	-6.2	5.184	5.476	-5.6	1.255	1.242	1.1
128	5.0	13.937	15.046	-8.0	18.582	17.549	5.6	1.333	1.166	12.5
128	10.0	26.476	28.195	-6.5	35.986	32.286	10.3	1.361	1.145	15.8
128	100.0	253.75	274.34	-8.1	345.29	314.18	9.0	1.340	1.145	14.5
256	0.01	4.043	4.023	0.5	4.992	5.007	-0.3	1.235	1.245	-0.8
256	0.1	4.457	4.231	5.1	5.097	5.215	-2.3	1.144	1.233	-7.8
256	0.3	4.886	4.689	4.0	5.434	5.673	-4.4	1.112	1.210	-8.8
256	1.0	6.341	6.576	-3.7	7.248	7.642	-5.4	1.143	1.162	-1.7
256	5.0	18.605	20.131	-8.2	22.200	22.634	-2.0	1.193	1.124	5.8
256	10.0	34.793	36.507	-4.9	41.536	40.598	2.3	1.194	1.112	6.8
256	100.0	320.88	355.28	-10.7	391.18	395.13	-1.0	1.219	1.112	8.8
512	0.01	8.058	8.023	0.4	8.991	9.007	-0.2	1.116	1.123	-0.6
512	0.1	8.541	8.231	3.6	9.093	9.215	-1.3	1.065	1.120	-5.2
512	0.3	8.894	8.689	2.3	9.437	9.673	-2.5	1.061	1.113	-4.9
512	1.0	10.670	10.909	-2.1	11.569	11.975	-3.5	1.083	1.098	-1.3
512	5.0	28.762	30.301	-5.4	32.115	32.803	-2.1	1.117	1.083	3.0
512	10.0	52.665	53.130	-0.9	59.137	57.221	3.2	1.123	1.077	4.1
512	100.0	485.65	517.15	-6.5	548.21	556.98	-1.6	1.129	1.077	4.6

Table 2: Simulation vs Model for a Truncated Normal Distribution, P=128

Q=	C_V=	I(simul)	I(mod)	ε(I)	ϕ'(simul)	ϕ'(mod)	ε(ϕ')	S(simul)	S(mod)	ε(S)
128	0.01	1.025	1.025	0.0	2.000	2.017	-0.9	1.950	1.968	-0.9
128	0.1	1.252	1.252	0.0	2.196	2.244	-2.2	1.754	1.793	-2.2
128	0.3	1.763	1.763	0.0	2.793	2.755	1.4	1.585	1.563	1.4
128	1.0	3.555	3.555	0.0	5.212	5.635	-8.1	1.466	1.585	-8.1
128	5.0	13.704	13.704	0.0	19.785	20.162	-1.9	1.444	1.471	-1.9
128	10.0	26.487	26.487	0.0	38.386	38.694	-0.8	1.449	1.461	-0.8
128	100.0	255.11	255.11	0.0	367.25	366.73	0.1	1.440	1.438	0.1
256	0.01	2.034	2.025	0.4	3.000	3.018	-0.6	1.474	1.490	-1.0
256	0.1	2.361	2.252	4.6	3.128	3.244	-3.7	1.325	1.441	-8.7
256	0.3	2.935	2.765	5.8	3.519	3.757	-6.7	1.199	1.359	-13.3
256	1.0	4.367	4.625	-5.9	5.457	5.700	-4.4	1.249	1.232	1.4
256	5.0	14.935	16.279	-9.0	20.095	18.802	6.4	1.345	1.155	14.2
256	10.0	28.533	30.614	-7.3	38.591	34.737	10.0	1.353	1.135	16.1
256	100.0	275.29	296.32	-7.6	375.13	336.47	10.3	1.363	1.363	16.7
512	0.01	4.047	4.025	0.5	4.999	5.018	-0.4	1.235	1.246	-0.9
512	0.1	4.498	4.252	5.5	5.131	5.244	-2.2	1.141	1.233	-8.1
512	0.3	4.976	4.765	4.3	5.525	5.757	-4.2	1.110	1.208	-8.8
512	1.0	6.590	6.791	-3.1	7.463	7.866	-5.4	1.133	1.158	-2.3
512	5.0	19.625	21.364	-8.9	23.268	23.887	-2.7	1.186	1.118	5.7
512	10.0	36.296	38.926	-7.2	43.756	43.049	1.6	1.206	1.106	8.3
512	100.0	342.75	377.26	-10.1	415.24	417.41	-0.5	1.212	1.106	8.7

Table 3: Simulation vs Model for a Truncated Normal Distribution, P=256

Q=	c_v=	I(simul)	I(mod)	ε(I)	ϕ'(simul)	ϕ'(mod)	ε(ϕ')	S(simul)	S(mod)	ε(S)
256	0.01	1.027	1.027	0.0	2.004	2.024	-1.0	1.951	1.969	-1.0
256	0.1	1.276	1.276	0.0	2.244	2.272	-1.2	1.759	1.781	-1.2
256	0.3	1.833	1.833	0.0	2.901	2.829	2.5	1.583	1.544	2.5
256	1.0	3.759	3.759	0.0	5.535	5.978	-8.0	1.473	1.590	-8.0
256	5.0	14.785	14.785	0.0	21.263	21.813	-2.6	1.438	1.475	-2.6
256	10.0	28.729	28.729	0.0	40.966	41.247	-0.7	1.426	1.436	-0.7
256	100.0	275.01	275.01	0.0	393.27	398.48	-1.3	1.430	1.449	-1.3
512	0.01	2.038	2.027	0.5	3.001	3.024	-0.7	1.473	1.491	-1.3
512	0.1	2.394	2.276	4.9	3.157	3.272	-3.6	1.319	1.438	-9.0
512	0.3	3.014	2.821	6.4	3.595	3.817	-6.2	1.193	1.353	-13.5
512	1.0	4.616	4.862	-5.3	5.734	5.941	-3.6	1.242	1.222	1.6
512	5.0	15.895	17.368	-9.3	21.415	19.901	7.1	1.347	1.146	15.0
512	10.0	30.525	32.717	-7.2	41.100	36.856	10.3	1.346	1.127	16.3
512	100.0	296.69	317.60	-7.1	400.68	357.91	10.7	1.355	1.127	16.8

Table 4: Simulation vs Model for a Uniform Distribution

P=	Q=	b	I(simul)	I(mod)	ε(I)	ϕ'(simul)	ϕ'(mod)	ε(ϕ')	S(simul)	S(mod)	ε(S)
64	64	2.0	1.970	1.970	0.0	3.349	3.690	-10.2	1.700	1.874	-10.2
64	128	2.0	3.171	2.968	6.4	3.786	3.952	-4.4	1.194	1.332	-11.6
64	256	2.0	5.112	4.968	2.8	5.786	5.952	-2.9	1.132	1.198	-5.9
64	512	2.0	9.102	8.968	1.5	9.788	9.952	-1.7	1.075	1.110	-3.2
128	128	2.0	1.984	1.984	0.0	3.499	3.780	-8.0	1.763	1.905	-8.0
128	256	2.0	3.219	2.984	7.3	3.849	3.976	-3.3	1.196	1.333	-11.4
128	512	2.0	5.183	4.984	3.8	5.851	5.976	-2.1	1.129	1.199	-6.2
256	256	2.0	1.992	1.992	0.0	3.586	3.844	-7.2	1.800	1.930	-7.2
256	512	2.0	3.270	2.992	8.4	3.894	3.988	-2.4	1.192	1.333	-11.9

Table 5: Simulation vs Model for an Exponential Distribution

P=	Q=	λ	I(simul)	I(mod)	ε(I)	ϕ'(simul)	ϕ'(mod)	ε(ϕ')	S(simul)	S(mod)	ε(S)
64	64	1.0	4.699	4.699	0.0	6.784	6.776	0.1	1.444	1.442	0.1
64	128	1.0	5.755	5.699	1.0	7.093	6.684	5.8	1.232	1.173	4.8
64	256	1.0	7.670	7.699	-0.4	8.748	8.684	0.7	1.141	1.128	1.1
64	512	1.0	11.734	11.699	0.3	12.776	12.684	0.7	1.089	1.084	0.4
128	128	1.0	5.368	5.368	0.0	7.630	7.611	0.3	1.421	1.418	0.3
128	256	1.0	6.435	6.380	0.8	7.969	7.372	7.5	1.238	1.156	6.7
128	512	1.0	8.365	8.380	-0.2	9.435	9.372	0.7	1.128	1.118	0.8
256	256	1.0	6.107	6.107	0.0	8.373	8.414	-0.5	1.371	1.378	-0.5
256	512	1.0	7.145	7.111	0.5	8.721	8.107	7.0	1.221	1.140	6.6

Formulas:

Truncated Normal

$$E(I) = \frac{Q-P}{P} \times m + 1 + \sigma \times 2.54631 \text{ and } E(\phi') = \frac{Q-1}{P} \times m + 1 + \sigma \times 2.3118 \text{ for P=64.}$$

$$E(I) = \frac{Q-P}{P} \times m + 1 + \sigma \times 2.54631 \text{ and } E(\phi') = \frac{Q-1}{P} \times m + 1 + \sigma \times 2.54631 \text{ for P=128}$$

$$E(I) = \frac{Q-P}{P} \times m + 1 + \sigma \times 2.54631 \text{ and } E(\phi') = \frac{Q-1}{P} \times m + 1 + \sigma \times 2.7597 \text{ for P=256}$$

In all three cases, m=1.083357 for c_v=1, m=2.54241 for c_v=5, m=4.51587 for c_v=10, m=40.46744 for c_v=100.

Uniform on [0, 2):

$$E(I) = \frac{Q}{P} + \frac{P-1}{P+1} \text{ and } E(\phi') = \frac{P+Q-1}{P} + \frac{P-1}{P+1}$$

Exponential with λ =1:

$$E(I) = \frac{Q}{P} + \log P - 0.42 \text{ and } E(\phi') = \frac{P+Q-1}{P} + \log P - 0.42$$