

# Local Synchronization with Batching

Preliminary Version

Julia Lipman  
*jlipman@eecs.umich.edu*

Quentin F. Stout  
*qstout@eecs.umich.edu*

Computer Science and Engineering  
University of Michigan

## Abstract

We examine the reduction in efficiency in a system where each processor has a sequence of tasks and must synchronize with other processors after each task is done. This occurs, for example, in many stepwise simulations. In OpenMP, the only explicit synchronization is the barrier, which causes all processors to wait for the slowest one. This is problematic when the tasks take variable amounts of time. The same inefficiency occurs when collective communication is used in MPI. However, use of send/receive commands in MPI generally reduces the synchronization to only that which is necessary, increasing efficiency. Here we look at the use of batching, whereby each processor completes several tasks before synchronizing, as yet another way to improve efficiency. We provide analytic and simulation results showing the results of batching with both barrier and send/receive synchronization models, where the task times can assume a variety of probability distributions. For the geometric and exponential distributions and cyclic dependencies, by using local synchronization and batching one can achieve any desired efficiency with a batching size independent of the number of processors. In contrast, for global synchronization and a fixed batch size, whenever the distribution has infinite support the efficiency will go to 0 as the number of processors tends to infinity.

## 1 Introduction

Synchronization is often necessary in parallel computing, but it can create delays whenever a processor is idle, waiting for information to arrive. This is especially true for barrier, or *global*, synchronization, in which every processor must synchronize with every other processor. For example, Tabe *et al.* [9] observed that barrier synchronization on the IBM SP2 caused performance degradation due to random operating system interrupts with unexpectedly long tails, even though every processor was completing tasks that should have taken the same amount of time. Nonetheless, barriers are the only form of synchronization explicitly

supplied in OpenMP, and they occur whenever collective communication operations are used in MPI.

However, in applications such as time-stepping algorithms for solving partial differential equations, a processor updating values at a site need only wait for the previous values from neighboring sites, a form of *local* synchronization. In a typical MPI send/receive program for such calculations, one region of the simulation may be a few time steps ahead of another without violating computational dependencies. Local synchronization is especially useful when the calculations can take varying amounts of time (perhaps due to the interrupts mentioned above) and delays can occur randomly at any processor or task (such as was observed in the operating system interrupts mentioned above). In this case, delays in one region may be smoothed out before they cause significant slowdown for the entire system.

Another way to reduce the effect of synchronization is to batch tasks, with each processor solving several before synchronizing. For example, this is often used in branch-and-bound calculations, where occasionally global bounds need to be determined. More generally, it occurs in many distributed manager-worker programs in which work is occasionally rebalanced, or in simulations where the workload shifts from one region to another. For this latter situation typically the shift involves many nearby sites and persistent imbalance for a number of cycles, in which case the imbalance does not smooth itself out. However, it may be more efficient to wait many cycles before taking a global picture of the situation, rather than doing so after every cycle.

In [5], we showed that local synchronization provided a significant advantage over barrier synchronization, and derived some bounds and asymptotic limits for synchronization times on various graphs with different probability distributions on the tasks. Here we extend our results to a model that includes batching, in which processors synchronize not after every single task, but after some number of tasks; this number is known as the *batch size*. We show exact asymptotic bounds for the case where tasks are exponentially or geometrically distributed and the synchronization graph is a directed cycle. We find a constant upper bound for barrier synchronization of normal tasks under batching. Finally, we present simulation results that show that introducing even a small amount of batching provides greater efficiency improvements than changing the entire graph from, say a 3-dimensional grid to an undirected cycle, to reduce synchronization constraints.

## 2 The Model

Synchronization can be modeled as a directed or undirected graph. We model the system as processors  $1, \dots, n$  connected according to some digraph  $G$ , all trying to complete a succession of tasks that are independent and identically distributed with probability distribution  $f$ . The nodes of  $G$  correspond to the processors, and there is an edge from processor  $p$  to processor  $q$  if  $q$  depends on  $p$ , as described below. The processors will complete  $B$  tasks at a time, the batch size, before synchronizing.

The system starts out with no processor having completed a task. A processor is able to begin the  $i$ th task if it has completed its  $(i - 1)$ st and all of its neighbors (that is, all processors from which it has an incoming edge) in the graph have completed at least  $(i - B + 1)$  tasks. By *level  $i$*  we mean the set consisting of the  $i$ th task over all processors.

Let  $\mathcal{T}_{G,f,B}(i)$  denote the expected time per level when all processors have completed the  $i$ th task. The (asymptotic) *expected time per level* of  $G$ ,  $\mathcal{T}_{G,f,B}^*$ , is  $\lim_{i \rightarrow \infty} \mathcal{T}_{G,f,B}(i)$ .

When  $G$  is infinite we modify  $\mathcal{T}_{G,f,B}(i)$  to mean the expected time at any given vertex, rather than the expected time per level in the entire graph. In a finite graph the asymptotic values are the same, but in an infinite graph, for any task distribution times with unbounded support, the expected time per level in the entire graph is infinite for all  $i$ .

### 3 Analytic Results for Local Synchronization with Geometric and Exponential Distributions

In [5], we derived an exact limit for synchronization time on a directed cycle with geometrically and exponentially distributed tasks. Here we extend these ideas to synchronization with batch size  $B$ . Throughout this section,  $G$  will be the directed cycle.

#### 3.1 The geometric distribution

Let the distribution  $f$  be the geometric distribution. We now derive an exact limit for  $\mathcal{T}_{G,f,B}^*$ .

The actions of the processors can be modeled as a discrete-time Markov chain where the states are ordered  $n$ -tuples of integers  $(p_0, \dots, p_{n-1})$  such that  $p_i$  corresponds to the number of tasks completed by the  $i$ th processor. So, in a three-processor system, if the first processor has completed the first task and the second two have not completed any, the state would be  $(1, 0, 0)$  (which we will abbreviate as 100). For the directed cycle, the states will consist of  $n$ -tuples where for all  $i$ ,  $p_i \leq p_{i+1} + B$  (and  $p_{n-1} \leq p_0 + B$ ).

When every processor has completed at least one task, the state is *rezeroed*: each number is decremented. The rate of rezeroing is what we want to measure; the expected amount of time between rezeroing operations is the expected time between when every processor in the system has completed at least  $i$  tasks and when every processor in the system has completed  $i + 1$  tasks. One way to represent this operation is to include states with no zeroes whose incoming edges are states with zeroes: the state 112 would have a single outgoing edge with probability 1 going to 001. However, rezeroing is considered to be instantaneous, and such an edge would imply that a time step had taken place. To avoid this problem, such a state can be collapsed with its succeeding state. The state 012, then, instead of having an outgoing edge going to 112, would instead have one going directly to 001; the rezeroing is implicitly understood. We call any processor that is not waiting for its neighbor to start the next task a *free* processor, and any processor that could have finished a task at the most recent time step a *last-minute processor*.

Calculating the steady-state probabilities of this Markov chain is much easier with the following lemma, which shows that the chain is *balanced*, i.e., that each of its states has the same number of incoming edges as it has outgoing edges.

**Lemma 3.1.** *The Markov chain for the directed cycle with geometrically distributed tasks and batching size  $B$  is a balanced digraph.*

*Proof.* We need to have some way to count the number of outgoing and incoming edges of a state  $S = (p_0, \dots, p_{n-1})$ . Note that a processor  $i$  is free if and only if  $p_i < p_{i-1} + B$ . If there

are  $k$  free processors, then there are  $2^k$  possible next states and  $2^k$  outgoing edges, since any subset of the free processors could finish their tasks by the next time step. These edges also have equal probability, since the coin is assumed to be fair.

Counting incoming edges is similar. Any  $i$  for which  $p_i > p_{i+1} - B$  is last-minute. To see this, observe that if  $p_i = r > 0$  at time  $t$ , then  $p_i$  could have been  $r - 1$  at time  $t - 1$ . If  $p_i = 0$ , then, clearly, it could not have been  $-1$  at the previous time step. But consider the state  $S' = (p_0 + 1, \dots, p_{n-1} + 1)$ . Such a state would be one of the states with no zeroes that we removed from the chain. We can see here that the  $i$ th processor is last-minute. According to our model, all of the edges that would have been  $S'$ 's incoming edges will go to  $S$ . So  $p_i$  could have been incremented at the last time step, with rezeroing possibly taking place. If  $p_i \leq p_{i+1} - B$ , then  $i$  could not have been incremented at the last time step, because that would imply that  $p_{i+1}$  was more than  $B$  greater than  $p_i$  at some time. If there are  $\ell$  last-minute processors, then there are  $2^\ell$  possible previous states, and  $2^\ell$  incoming edges with equal probability.

We would like to show that  $\ell = k$  in any possible state. To see this, observe that  $p_i \leq p_{i-1}$  if and only if  $p_{i-1} \geq p_i$ . That is, there is a one-to-one correspondence between free processors and last-minute processors. So each state has the same number of incoming edges as outgoing edges.  $\square$

**Theorem 3.1.** *In the Markov chain for the geometric distribution with  $n$  processors and batch size  $B$ , there are  $\binom{n}{k} \binom{Bn-1}{k-1}$  states with  $k$  free processors.*

*Proof.* Stanley [8] proves that the Narayana numbers  $N(n, k) = \frac{1}{n} \binom{n}{k} \binom{n}{k-1}$  enumerate sequences of length  $2n$  in which 1 appears  $n$  times and  $-1$  appears  $n$  times such that each partial sum is nonnegative and there are exactly  $k$  instances of a 1 immediately followed by a  $-1$ . A variation on his proof yields a proof of this lemma.

Stanley's construction begins with two compositions,  $A : a_1 + \dots + a_k = n + 1$  and  $B : b_1 + \dots + b_k = n$ . There are  $\binom{n}{k-1} \binom{n-1}{k-1}$  ordered pairs  $(A, B)$ . From each ordered pair, he creates a circular sequence with a block of  $a_1$  1's, then a block of  $b_1$   $-1$ 's, then a block of  $a_2$  1's, and so on. He shows that each such circular sequence could have been created from exactly  $k$  composition pairs, and that there is only one way to get a linear sequence from it that starts with a 1 and has all nonnegative partial sums when this 1 is removed, so there are  $\frac{1}{k} \binom{n}{k-1} \binom{n-1}{k-1} = \frac{1}{n} \binom{n}{k} \binom{n}{k-1}$  such sequences.

Our proof is similar, except instead of circular sequences consisting of  $n+1$  occurrences of 1 and  $n$  occurrences of  $-1$ , we consider those with  $n+1$  occurrences of  $B$  and  $Bn$  occurrences of  $-1$ . A state of length  $n$  with  $k$  free processors is a sequence  $(p_0, \dots, p_{n-1})$  of nonnegative integers that satisfies the following three conditions:

1. It contains at least one 0,
2. For all  $i$ ,  $p_{i+1 \pmod n} \leq p_i + B$
3. There are exactly  $k$  positions  $i$  at which  $p_{i+1 \pmod n} < p_i + B$ .

Such a state can be obtained from a circular sequence  $S$  of the form we have described as follows: break  $S$  into a linear sequence that starts with at least two  $B$ 's, and remove the

first of these; call this new linear sequence  $S'$ . We create a new sequence  $T$  from  $S'$ : At the first element (which must be a  $B$ ), write a 0. At each subsequent  $B$ , write the partial sum at that point (take  $-1$ 's into account in the calculation of the partial sums, but write partial sums only at 1's, not at  $-1$ 's). Some of the terms in  $T$  may be negative. Now increase each term in  $T$  by the same amount such that the least term is 0. The sequence we construct from  $T$  in this way is a state of length  $n$  with  $k$  free processors.

There are  $n - k + 1$  points at which to break  $S$  into a linear sequence such that it starts with at least two  $B$ 's, since there are  $n + 1$   $B$ 's and  $k$  of them are immediately followed by a  $-1$ . Now we must deal with two cases: the case where breaking the circular sequence at each of these points results in a unique circular sequence and the case where it does not.

First we consider the case where each legal point to break the circular sequence  $S$  results in a different state, so any particular circular sequence yields  $n - k + 1$  states. Then every one of the  $Bn + n + 1$  cyclic permutations of  $S$  is unique. We can see that every length- $n$ ,  $k$ -free-processor state can be made into a circular sequence of the form in Stanley's construction: Go through the sequence, at each  $p_i$  writing  $p_i - p_{i+1 \pmod n} + 1$   $-1$ 's followed by a  $B$ . Finish by writing an extra 1. This will result in a circular sequence containing  $n + 1$   $B$ 's (one for each entry in the state and one extra) and  $n - 1$ 's, since it ends up in the same place it started. Since there are  $k$  free processors, there are  $k$  positions at which  $p_{i+1 \pmod n} < p_i + B$ , so there will be  $k$  groups of 1's. So the circular sequence created is one of the form that we have described.

We have shown that there are  $n - k + 1$  times as many states of length  $n$  with  $k$  free processors as there are such circular sequences. Since every circular sequence  $S$  we consider in this case has every cyclic permutation unique, there are  $k$  ordered composition pairs from which  $S$  could have been obtained. So each such composition pair yields  $\frac{n-k+1}{k}$  states.

Now we consider the case where every cyclic permutation of  $S$  is not unique. Then there are  $\frac{Bn+n+1}{m}$  unique cyclic permutations of  $S$ , where  $m$  is some integer that divides  $Bn + n + 1$ . This means that there are  $\frac{n-k+1}{m}$  places at which to break  $S$  to create a unique state. But there are also only  $\frac{k}{m}$  places at which to break the sequence to find a unique composition pair from which it could have been obtained. So each such composition pair yields  $\frac{m(n-k+1)}{mk} = \frac{n-k+1}{k}$  states — exactly the same as in the case where every cyclic permutation of  $S$  is unique.

So, since there are  $\binom{n}{k-1} \binom{Bn-1}{k-1}$  composition pairs of the form we have described, the total number of states of length  $n$  with  $k$  free processors in a system with batch size  $B$  is:

$$\frac{n - k + 1}{k} \binom{n}{k - 1} \binom{Bn - 1}{k - 1} = \binom{n}{k} \binom{Bn - 1}{k - 1}.$$

□

We will now specialize to the geometric distribution with  $p = 0.5$ . For this distribution, the expected time of a task, which is also the expected time per level of a graph with no edges, is 2.

**Theorem 3.2.** *Each processor in a directed cycle with geometrically distributed tasks with  $p = 0.5$  is free for a fraction of the time that approaches  $B + 1 - \sqrt{B^2 + 1}$  as the number of processors in the cycle goes to infinity.*

*Proof.* A state with  $k$  free processors has outdegree  $2^k$  in the Markov chain of the directed cycle with geometric tasks. As we have shown, the Markov chain is a balanced digraph for  $p = 0.5$ , which means that the steady-state probability of a state is proportional to its (out)degree.

We can find the expected number of free processors at any given time by summing, over all states, the number of free processors multiplied by the outdegree, which using what we have shown in Theorem 3.1, is  $\sum_{k=1}^n k2^k \binom{n}{k} \binom{Bn-1}{k-1}$ , and normalizing by the total outdegree of all states  $\sum_{k=1}^n 2^k \binom{n}{k} \binom{Bn-1}{k-1}$ . So we want to find the limit of the quotient

$$\frac{\sum_{k=1}^n k2^k \binom{n}{k} \binom{Bn-1}{k-1}}{\sum_{k=1}^n 2^k \binom{n}{k} \binom{Bn-1}{k-1}}$$

as  $n$  goes to infinity.

In order to estimate these sums, we find the  $k$  at which the greatest term occurs in each sum. In each case, the terms in the sum are unimodal, so we look for the  $k$  at which the ratio between the  $(k+1)$ th and  $k$ th term is closest to 1. In  $\sum_{k=1}^n k2^k \binom{n}{k} \binom{Bn-1}{k-1}$ , we have the equation  $k2^k \binom{n}{k} \binom{Bn-1}{k-1} = (k+1)2^{k+1} \binom{n}{k+1} \binom{Bn-1}{k}$ . Solving the resulting quadratic for  $k$  results in  $1 + (B+1 - \sqrt{B^2+1})n$ .

Doing the same for the sum in the denominator,  $\sum_{k=1}^n 2^k \binom{n}{k} \binom{Bn-1}{k-1}$ , results in

$$k = 1.5 + n + Bn - \frac{1}{2}\sqrt{1 + 4n + 4Bn + 4n^2 + 4B^2 + 4B^2n^2}$$

which approaches  $(B+1 - \sqrt{B^2+1})n$  as  $n$  goes to infinity. So the largest term of each sum occurs at a  $k$  that approaches  $(B+1 - \sqrt{B^2+1})n$ .

The terms of both sums drop off exponentially above and below the greatest term. So the ratio of the two sums evaluated at  $k = (B+1 - \sqrt{B^2+1})n$  approaches  $k = (B+1 - \sqrt{B^2+1})n$  as  $n$  goes to infinity.

Since a fraction of processors that approaches  $B+1 - \sqrt{B^2+1}$  of the total is expected to be free at any given time, by symmetry, any given processor will be free for a proportion of the time that approaches  $B+1 - \sqrt{B^2+1}$ .  $\square$

**Corollary 3.2.1.** *When  $G$  is a directed cycle and  $f$  is the geometric distribution with  $p = 0.5$ , for a batch size  $B$ ,  $\mathcal{T}^*$  approaches  $\frac{B+1+\sqrt{B^2+1}}{B} = 2 + \frac{1}{B} + \Theta\left(\frac{1}{B^2}\right)$  as the number of processors in the cycle goes to infinity. Thus to achieve efficiency  $c < 1$  one need only choose  $B \approx \frac{c}{2(1-c)}$ .*

*Proof.* Each processor is free for a fraction of the time that approaches  $B+1 - \sqrt{B^2+1}$ . Whenever a processor is free, it is actively engaged in working on a task; that is, waiting for a coin to come up heads. The expected amount of time for a fair coin to come up heads is 2, so  $\mathcal{T}^*$  is  $\frac{2}{B+1-\sqrt{B^2+1}} = \frac{B+1+\sqrt{B^2+1}}{B}$ .  $\square$

Note that if  $B$  is any increasing function of  $n$ , then the efficiency converges to 1 as  $n \rightarrow \infty$ .

## 3.2 The exponential distribution

We use the techniques described above to derive  $\mathcal{T}_{G,f,B}^*$  when  $f$  is the exponential distribution with  $\lambda = 2$  (so that the expected time per task is 2, as for our geometric distribution.) Here

we model the system as a discrete-time Markov chain whose states are exactly the same as the states of the chain we constructed for the geometric distribution, but where only one processor can finish a task at a particular time step. At every time step, each of the processors in the state  $(p_0, \dots, p_{n-1})$  has an equal chance of being “chosen.” If the  $p_i$  chosen is free, it completes its task; if not, the system remains in the same state.

**Lemma 3.2.** *The Markov chain for the directed cycle with exponentially distributed tasks and batching size  $B$  is a balanced digraph in which every state has indegree and outdegree  $n$ .*

*Proof.* In lemma 3.1, we showed that every state in the Markov chain for the geometric distribution has the same number of free processors as last-minute processors. Since the states are exactly the same in this chain, that still holds. However, only one processor can act at a time, making the outdegree  $n$  (if the processor chosen is free, the outgoing edge goes to a different state; if not, it is a self-loop.) Every state also has indegree  $n$ ; consider a state with  $k$  free processors. Then it also has  $k$  last-minute processors and therefore  $k$  incoming edges from other states. It also has  $n - k$  non-free processors, resulting in  $n - k$  incoming self-loops, for a total of  $n - k + k = n$  incoming edges.  $\square$

This means that every state in the Markov chain has the same steady-state probability. (A version of this without considering batching was shown in [7].) So in order to calculate the expected number of free processors in the steady state, we need only to take the average of all states, weighted by free processors:

$$\frac{\sum_{k=1}^n k \binom{n}{k} \binom{Bn-1}{k-1}}{\sum_{k=1}^n \binom{n}{k} \binom{Bn-1}{k-1}} = \frac{Bn^2}{n + Bn - 1} \approx \frac{B}{B + 1}n$$

Following the same analyses as for the geometric distribution, we have

**Theorem 3.3.** *When  $G$  is the directed cycle of  $n$  processors and  $f$  is the exponential distribution with mean 2, then when batches of size  $B$  are used,  $\mathcal{T}^*$  converges to  $2 + 2/B$  as  $n \rightarrow \infty$ . Thus one will achieve efficiency  $c < 1$  when  $B \approx \frac{c}{1-c}$ .*

As in the model with geometrically distributed tasks, when  $B$  is any increasing function of  $n$ ,  $\mathcal{T}_{G,f,B}^*$  approaches the value for a graph with no synchronization as  $n \rightarrow \infty$ .

## 4 Barrier Synchronization of Normally Distributed Tasks with Batching

We have seen how batching improves efficiency for local synchronization, but what about barrier synchronization? Here we consider the case where  $f$  is a normal distribution with parameters  $\mu$  and  $\sigma^2$  and  $G$  is a complete graph. Again,  $B$  is the batch size.

**Theorem 4.1.** *If the task-completion times are identical and normally distributed with parameters  $\mu$  and  $\sigma^2$  and the synchronization graph is complete (i.e. barrier synchronization), then a batch size of  $B$  results in an expected time per level that approaches*

$$\mu + \sqrt{\frac{2 \log(n) - \frac{1}{2} \log(B) - \log(\sqrt{\frac{\pi}{2}} \sigma)}{B}}$$

*Proof.* Since our individual task-completion times are normally distributed, sums of  $B$  of them are normally distributed as well, with parameters  $B\mu$  and  $B\sigma^2$ . The expected time per level in this complete graph is the expected maximum of  $n$  sums of  $B$  task times, divided by  $B$ .

By convexity arguments [3], an upper bound on the expected value of the greatest of  $m$  normally distributed random variables with distribution function  $F$  is  $F^{-1}(1 - \frac{1}{2m})$  — that is, the point  $x$  at which the area under the tail of the density function  $f$  to the right of  $x$  is  $\frac{1}{2m}$ . A well-known upper bound for the area of the tail of a normal distribution to the right of  $x$  is  $f(x)$ , in our case,  $\frac{1}{\sqrt{2B\pi\sigma}} e^{-\frac{(x-B\mu)^2}{2B\sigma^2}}$ .

Solving the equation

$$\frac{1}{\sqrt{2\pi B\sigma}} e^{-\frac{(x-B\mu)^2}{2B\sigma^2}} = \frac{1}{2n}$$

for  $x$  results in a time per batch of

$$x = B \left( \mu + \sqrt{\frac{2 \log(n) - \frac{1}{2} \log(B) - \log(\sqrt{\frac{\pi}{2}}\sigma)}{B}} \right)$$

□

When  $G$  is a complete graph and  $f$  is the normal distribution, with no batching,  $\mathcal{T}_{G,f,1}^*$  grows unboundedly with the number of processors as  $\Theta(\sqrt{\log(n)})$ . But when we introduce batching, this is immediately cut by a multiplicative factor of  $\sqrt{B}$ . So we have

**Corollary 4.1.1.** *If  $B$  increases with  $n$  as  $\Omega(\log n)$ , then  $\mathcal{T}^*$  is bounded by a constant as  $n \rightarrow \infty$ .*

## 5 Simulation Results

To extend the analytic results to additional synchronization graphs, we implemented a simulation of  $n$  processors with undirected cycle, 2-dimensional toroidal grid, and 3-dimensional toroidal grid local synchronization.

### 5.1 The exponential distribution

We found that even a small amount of batching greatly increases efficiency. In an undirected cycle with 484 processors, the average time per level was 4.76 (efficiency 0.42), but with a batching size of 3, it was 2.86 (efficiency 0.70). In a 2-dimensional grid with 484 processors, the average time per level was 6.26 (efficiency 0.32) with no batching, but decreased to 3.25 (efficiency 0.62) with a batch size of just 3. Similarly, in a 3-dimensional grid with 729 processors, going from no batching to a batch size of 3 decreases the time per level from 7.20 to 3.50.

In fact, a small amount of batching had more of an effect on the efficiency than did reducing the dimension (and thus dependencies) of the entire graph. Comparing Figure 1

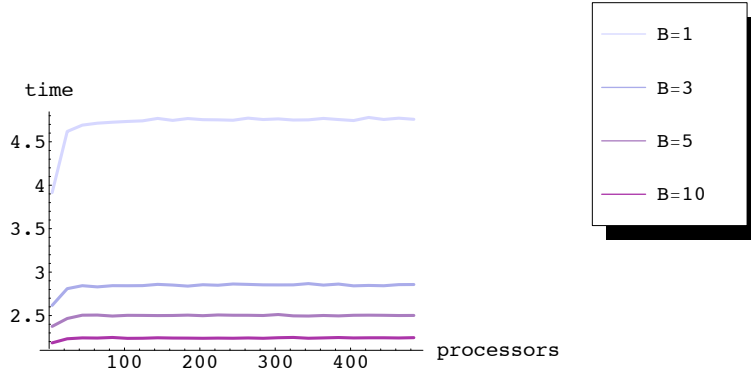


Figure 1: Asymptotic time per level versus  $n$  on an undirected cycle, exponentially distributed task times with  $\lambda = 0.5$  and batch size  $B$ .

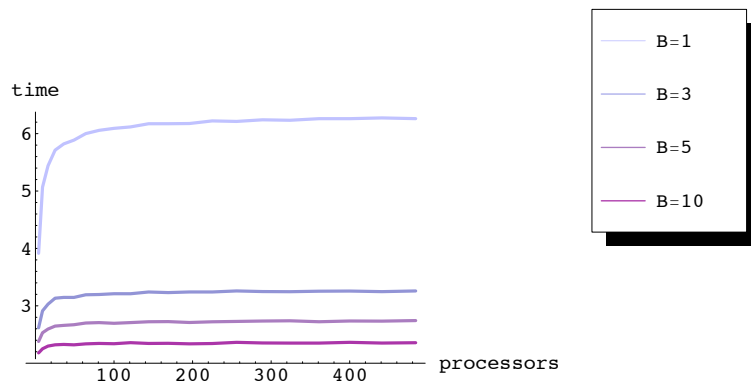


Figure 2: Asymptotic time per level versus  $n$  on a 2D grid, exponentially distributed task times with  $\lambda = 0.5$  and batch size  $B$ .

and Figure 3, we see that the efficiency on a 3-dimensional grid with a batch size 3 is much better than that on an undirected cycle with no batching, even though the 3-dimensional grid is much more densely connected.

## 5.2 Pareto distribution

One of the challenging aspects of some computing tasks is that they can have heavy-tailed distributions, typically power laws (Pareto distributions) with a pdf of the form  $x^{-\lambda}$ . For example, file sizes and network traffic have such distributions [1, 6, 4]. For power laws, the probability of sampling a large value is greatly higher than it is for the other distributions considered, and thus one would expect that reducing synchronization is even more important. Our simulations in Figures 4, 5 and 6 are for  $\lambda = 2$ , which also has a mean of 2. We found that batching provides improvements similar to those that it provides for exponentially-distributed tasks, except that the time per level with Pareto-distributed tasks does not appear to be converging to any constant value.

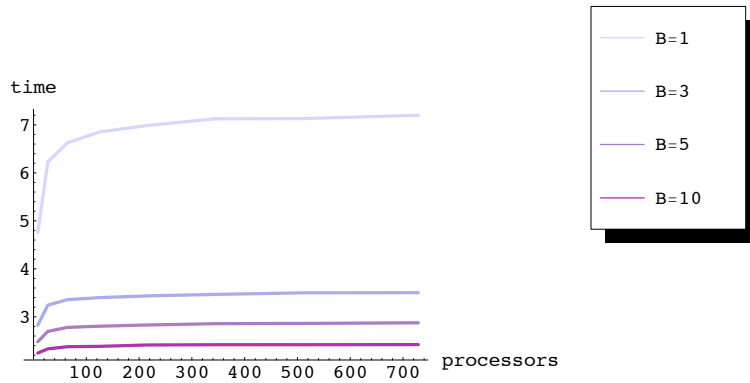


Figure 3: Asymptotic time per level versus  $n$  on a 3D grid, exponentially distributed task times with  $\lambda = 0.5$  and batch size  $B$ .

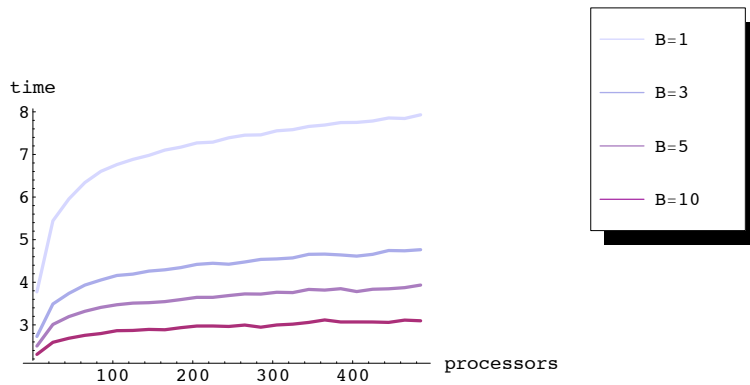


Figure 4: Asymptotic time per level versus  $n$  on an undirected cycle, Pareto distributed task times with  $\lambda = 2$  and batch size  $B$ .

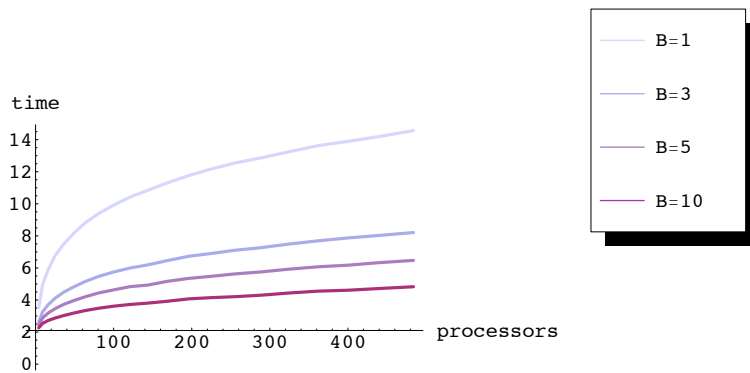


Figure 5: Asymptotic time per level versus  $n$  on a 2D grid, Pareto distributed task times with  $\lambda = 2$  and batch size  $B$ .

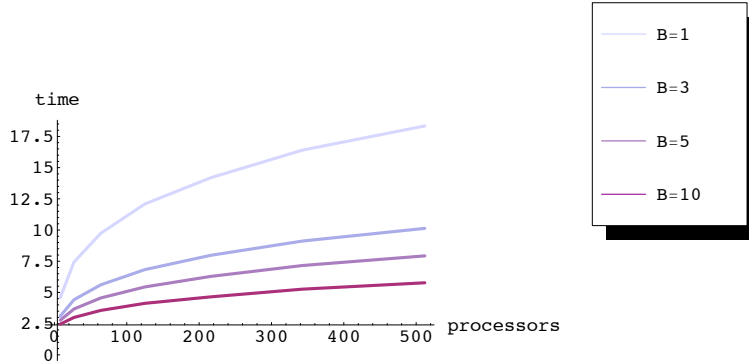


Figure 6: Asymptotic time per level versus  $n$  on a 3D grid, Pareto distributed task times with  $\lambda = 2$  and batch size  $B$ .

### 5.3 Barrier synchronization

It is easy to show that if a distribution has infinite support, as is the case with all of those considered here, then as  $n$  goes to infinity, the time per level for barrier synchronization also goes to infinity. In Section 4 we showed that barrier synchronization with the normal distribution has a time per level that grows as  $\sqrt{\log n}$ , and that this can be reduced to a constant by choosing batch sizes that grow as  $\Omega(\log n)$ . For a Pareto distribution with parameter  $\lambda$ , the time per level grows as  $n^{-\lambda}$ , while for the exponential and geometric distributions it grows as  $\log(n)$ .

In Figure 7 we show the batch size needed to achieve the given efficiency for a geometric distribution with  $p = 0.5$ . While it is easy to show that the batch sizes must grow as  $\Omega(\log n)$  to achieve constant efficiency, we do not yet have an exact bound.

## 6 Final Remarks

Note: this is a preliminary version of the paper — the conference version will have several improvements.

This paper is part of an analysis of the interaction of randomness in task times, synchronization, and batching. The random task times are often due to forces beyond one’s control, such as external interrupts or inputs which have random size or computational requirements. When the randomness is significant, and barrier synchronization is used, then the combined effect can dramatically lower performance. On the other hand, sometimes one has control over the synchronization used, or the amount of batching, and can use this to reduce the impact of the random task times.

As we have shown, when one uses the least amount of synchronization possible, then for some basic communication graphs the behavior is significantly better than if the simpler barrier synchronization is used. For example, even though the time per level of the exponential distribution grows as  $\log(n)$  for global synchronization, for each degree  $d$  it is bounded by a constant for all graphs of degree no greater than  $d$  [2]. For the directed cycle we computed exact bounds, and showed that one could use batching to achieve any desired

Eff.	number of processors									
	2	4	8	16	32	64	128	256	512	1024
0.8	3	10	19	30	41	53	66	78	91	103
0.9	13	46	88	137	189	243	299	355	412	468

Global synchronization, geometric distribution with  $p = 0.5$ .

Figure 7: Batch size needed to achieve given efficiency.

level of efficiency, with a batch size that is independent of the size of the cycle. As the figures in Section 5 show, even a little batching can have a dramatic effect on smoothing out randomness in the task times. Using local synchronization instead of global (barrier) synchronization, coupled with batching, can greatly improve efficiency.

We believe that similar results will hold for all graphs of bounded degree and a wide range of task time distributions. While any distribution with infinite support must have global synchronization task times that go to infinity as the number of processors goes to infinity, for many useful distributions far different behavior will occur if only local synchronization is used, coupled with batching. In particular, since exponential distributions have a heavier tail than the normal distribution, the batching result for directed cycles applies to normal distributions as well.

However, one problematic class of distributions are the power-law, or Pareto, distributions. These have very heavy tails, so that occasionally tasks take a very long time. For example, the fact that power law distributions have been observed in file sizes on the web makes it possible that the file-processing tasks will take widely varying times [1, 6, 4]. While there is some controversy over whether a power law or lognormal model is the best model of file sizes, both have quite heavy tails. We do not believe that fixed size batching will produce fixed efficiency even for the directed cycle, though we don't yet have a proof of this.

## References

- [1] P. Barford, A. Bestavros, A. Bradley, and M. Crovella. Changes in web client access patterns: characteristics and caching implications. *World Wide Web*, 2:15–28, 1999.
- [2] C. S. Chang and R. Nelson. Bounds on the speedup and efficiency of partial synchronization in parallel processing systems. *Journal of the ACM*, 42(1):204–231, 1995.
- [3] H. A. David. *Order Statistics*. Wiley, 1970.
- [4] A. Downey. The structural cause of file size distributions. In *Proc. 2001 SIGMETRICS*, pages 328–329, 2001.
- [5] J. Lipman and Q. Stout. A performance analysis of local synchronization. In *Proceedings of the 18th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 254–260, July 2006.

- [6] M. Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics*, 1:226–251, 2003.
- [7] S. Rajsbaum and M. Sidi. On the performance of synchronized programs in distributed networks with random processing times and transmission delays. *IEEE Transactions on Parallel and Distributed Systems*, 5(9):939–950, 1994.
- [8] R. Stanley. *Enumerative Combinatorics*, volume 2. Cambridge University Press, 1999.
- [9] T. Tabe, J. Hardwick, and Q. Stout. Statistical analysis of communication on the IBM SP2. *Computing Science and Statistics*, 27:347–351, 1995.