

A Performance Analysis of Local Synchronization

Julia Lipman
Computer Science and Engineering
University of Michigan
2260 Hayward
Ann Arbor, MI 48109
jlipman@eecs.umich.edu

Quentin F. Stout
Computer Science and Engineering
University of Michigan
2260 Hayward
Ann Arbor, MI 48109
qstout@eecs.umich.edu

ABSTRACT

Synchronization is often necessary in parallel computing, but it can create delays whenever the receiving processor is idle, waiting for the information to arrive. This is especially true for barrier, or global, synchronization, in which every processor must synchronize with every other processor. Nonetheless, barriers are the only form of synchronization explicitly supplied in MPI and OpenMP.

Many applications do not actually require global synchronization; local synchronization, in which a processor synchronizes only with those processors from which it has an incoming edge in some directed graph, is often adequate. However, the behavior of a system under local synchronization is more difficult to analyze, since processors do not all start tasks at the same time.

In this paper, we show that if the synchronization graph is a directed cycle and the task times are geometrically distributed with $p = 0.5$, the time it takes for a processor to complete a task, including synchronization time, approaches an exact limit of $2 + \sqrt{2}$ as the number of processors in the cycle approaches infinity. Under global synchronization, however, the time is unbounded, increasing logarithmically with the number of processors. Similar results also apply for $p \neq 0.5$.

We give a new proof of the constant upper bounds that apply when tasks are normally distributed and the synchronization graph is any graph of bounded degree. We also prove that for some power-law distributions on the tasks, there is no constant upper bound as the number of processors increases, even for the directed cycle. Finally, we show that constant upper bounds apply for some cases of a different synchronization model in which a processor waits for only a subset of its neighbors.

Categories and Subject Descriptors

F.2 [Theory of Computation]: Analysis of Algorithms and Problem Complexity

General Terms

Theory, Performance

Keywords

Synchronization, performance analysis, geometric distribution, heavy-tailed distribution, stochastic task times

1. INTRODUCTION

The problem of synchronization is a very general one that appears in many contexts. It applies to any situation in which some kind of distributed agents or processors can communicate information about their individual states to each other and a receiving processor cannot proceed until this information is received. Synchronization is often necessary in parallel computing, but it can create delays whenever the receiving processor is idle, waiting for the 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.* [13] 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 MPI and OpenMP.

Synchronization requirements can be modeled as a directed graph. Barrier synchronization corresponds to a complete graph, but often the required synchronization is a far sparser graph. As an important example: for most explicit 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. In this case, local, or neighbor, synchronization can be used, where a processor waits only on its neighbors in the synchronization graph.

In this paper we prove an exact limit of expected performance as the number of processors approaches infinity in the case where task times are geometrically distributed and neighbor synchronization in a directed cycle is used. We also give a new proof of the constant bound for normally distributed tasks shown by Chang and Nelson [1]. Finally, we show that for certain power-law distributions, the time

between tasks for a processor goes to infinity as the number of processors increases, even for neighbor synchronization, as long as the graph is strongly connected.

We model the system as processors p_1, \dots, p_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 system starts out with no processor having completed a task. A processor begins the i th task as soon as it and all of its neighbors (that is, all processors from which it has an incoming edge) in the graph have completed the $(i - 1)$ st. Let $\mathcal{T}_{G,f}(i)$ denote the expected time per level when all processors have completed the i th task. We define the (asymptotic) *expected time between levels* of G , $\mathcal{T}_{G,f}^*$, to be $\lim_{i \rightarrow \infty} \mathcal{T}_{G,f}(i)$.

When G is infinite we modify $\mathcal{T}_{G,f}(i)$ 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 .

2. PREVIOUS WORK

Chang and Nelson [1] proved some bounds on the performance of a parallel system under local synchronization. Using a branching process method of Kingman [7], they showed that if the task time distribution has all finite moments and the synchronization graph is of bounded degree, the expected time for a processor to complete a task, including synchronization time, is bounded above by a constant as the number of processors approaches infinity, although this constant does not generally have a closed form. “The difficulty of the analysis lies in the fact that the times processors start their i th iteration differ, which is not the case for barrier synchronization. This makes an exact analysis intractable . . .” they wrote.

Rajsbaum and Sidi [10] show that, among all “new better than used in expectation” (NBUE) distributions for f , the exponential distribution produces the worst performance. They also perform an exact analysis of the cases where G is a directed cycle or a complete graph, and f is the exponential distribution, using both combinatorial and queueing-theoretic methods. Finally, they study the behavior of a system with one slow “bottleneck” processor.

Legedza and Wehl [8] studied the effect of replacing barrier synchronization with local synchronization in sequential simulators of parallel systems. Their experiments showed that replacing barriers with local synchronization improved program performance by 24 percent for some applications. Han *et al.* [5] proposed a compiler optimization to replace many instances of barrier synchronization with local synchronization.

Malony *et al.* [9] implemented a stochastic technique for predicting the performance scalability of parallel programs which transformed the graph of the original program into a “scaled” model or models. But for a neighbor-synchronized program, they conclude, “If we were to represent each task and dependency . . . in the scaled model, the graph size and complexity would be unmanageable.” Their solution is to

transform the original graph of the neighbor-synchronized program into one with barriers to create an upper bound, and into one with no synchronization at all to create a lower bound, or to implement barriers at some points and no synchronization at others.

One widely used method for predicting the performance of parallel programs with stochastic task duration times is to create a dependency graph of the tasks in the program and then study a similar series-parallel graph, which is much easier to analyze. The algorithm that Escribano and van Gemund [4] propose for converting non-series-parallel graphs into series-parallel graphs results in particularly large upper bounds for neighbor-synchronized graphs; in fact, it simply replaces local synchronization with barriers, as Malony’s method does. Escribano and van Gemund [3] researched the extent to which adding dependencies to a task graph in order to make it series-parallel affects its expected completion time. They conjectured that any graph representing a parallel computation has an equivalent series-parallel graph which takes no longer than twice that of the original graph. However, Salamon [11] writes, “These [neighbor-synchronized] task graphs are known to provide counterexamples to [Escribano’s conjecture] . . . Benchmark structures that occur frequently in practice, such as neighbor synchronization, need to be analyzed further.”

3. ANALYTIC RESULTS FOR THE GEOMETRIC DISTRIBUTION

Here we consider the geometric distribution, where each task can be thought of as flipping a coin until it comes up heads. The memoryless properties of this distribution make it a natural choice for applying analytical methods, but traditional closed-queue methods do not apply like they do to the exponential distribution that Rajsbaum and Sidi studied.

For a complete graph of n processors, the expected time between levels is simply the expected time it takes for n coins all to come up heads; that is, the expected maximum of n geometric random variables. This expected maximum grows with n as $\log_2 n$. For the remainder of this paper, unless otherwise explicitly stated, we assume that the coin is fair.

We consider the local synchronization case where G is a directed cycle; each processor must wait only for itself and its left neighbor. Certainly, the expected time between levels in this case can grow no faster with n than that for the complete graph. We show that it is in fact bounded by a constant as n goes to infinity.

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 , $p_i \leq p_{i+1} + 1$ (and $p_{n-1} \leq p_0 + 1$).

When every processor has completed at least one task, the

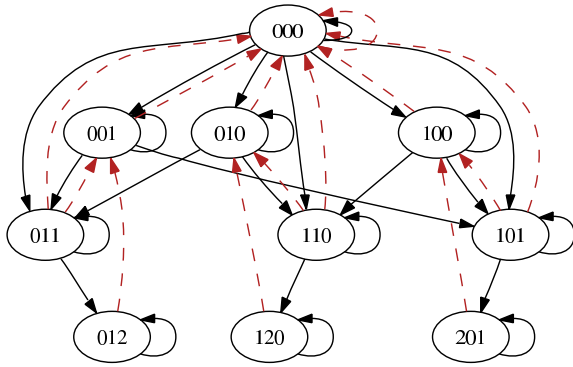


Figure 1: Markov chain for the directed cycle on 3 processors.

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 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 $p_i \leq p_{i+1}$. If there are f free processors, then there are 2^f possible next states and 2^f 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 $p_i \geq p_{i+1}$ 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 $p_i < p_{i+1}$, then i could not have been incremented at the last time step, because that would imply that p_{i+1} was more than 1 greater than p_i at some time. If there are l last-minute processors, then then there are 2^l possible previous states and 2^l incoming edges with equal probability.

We would like to show that $l = f$ 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

Since the Markov chain of states is a balanced digraph, the steady-state probability of a state is proportional to its degree, so we can calculate the steady-state probabilities of a state simply by observing which of its processors are free. By looking at the numbers of free processors of all the states in the chain, we can calculate how many processors are expected to be free at a given time, and then how much of the time any processor is expected to be free.

LEMMA 3.2. *There are $\binom{n}{k} \binom{n-1}{k-1}$ states of length n with k free processors.*

PROOF. Stanley [12] proves that the Narayana numbers $N(n, k) = \frac{1}{n} \binom{n}{k} \binom{n-1}{k-1}$ 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 $\frac{1}{k} \binom{n}{k-1} \binom{n-1}{k-1} = \frac{1}{n} \binom{n}{k} \binom{n-1}{k-1}$ such sequences.

Our proof is similar. A state of length n with k free processors is a sequence p_0, \dots, p_{n-1} of nonnegative integers containing at least one 0 that satisfies the following two conditions: For all i , $p_{i+1 \pmod n} \leq p_i + 1$, and there are exactly k positions i at which $p_{i+1 \pmod n} \leq p_i$. Such a state can be obtained from a circular sequence S of the form in Stanley's construction as follows: break S into a linear sequence that starts with at least two 1'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 1), write a 0. At each subsequent 1, 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 1's, since there are $n + 1$ 1's and k of them are immediately followed by -1 's. Breaking it at each of these points results in a different beginning state for the same reason that S could have been created from exactly k composition pairs: n and $n + 1$ are relatively prime. We can also see that every length- n , k -free-processor beginning 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 1. Finish by writing an extra 1. This will result in a circular sequence containing $n + 1$ 1'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 $p_i - p_{i+1 \pmod n} + 1 > 0$, so there will be k groups of 1's. So the circular sequence created is one of the form in Stanley's construction.

We have shown that there are $n - k + 1$ times as many beginning states of length n with k free processors as there are such circular sequences:

$$(n - k + 1) \frac{1}{k} \binom{n}{k-1} \binom{n-1}{k-1} = \binom{n}{k} \binom{n-1}{k-1}. \quad \square$$

THEOREM 3.1. *Each processor in a directed cycle with geometrically distributed tasks with $p = 0.5$ is free for a fraction of the time that approaches $2 - \sqrt{2} \approx 0.58$ 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 Lemma 3.2 is $\sum_{k=1}^n k 2^k \binom{n}{k} \binom{n-1}{k-1}$, and normalizing by the total outdegree of all states ($\sum_{k=1}^n 2^k \binom{n}{k} \binom{n-1}{k-1}$). So we want to find the limit of the quotient

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

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 $\sum_{k=1}^n k 2^k \binom{n}{k} \binom{n-1}{k-1}$, we have the equation $k 2^k \binom{n}{k} \binom{n-1}{k-1} = (k + 1) 2^{k+1} \binom{n}{k+1} \binom{n-1}{k}$. Solving the resulting quadratic for k results in $(2 - \sqrt{2})n \approx 0.58n$. Doing the same for the sum

in the denominator, $\sum_{k=1}^n 2^k \binom{n}{k} \binom{n-1}{k-1}$, results in a value of k that approaches $(2 - \sqrt{2})n$ as n goes to infinity. So the largest term of both sums occurs at $k = (2 - \sqrt{2})n$.

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

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

COROLLARY 3.1.1. *When G is a directed cycle and f is the geometric distribution with $p = 0.5$, $\mathcal{T}_{G,f}^*$ approaches $2 + \sqrt{2} \approx 3.42$ as the number of processors in the cycle goes to infinity.*

PROOF. Each processor is free for a fraction of the time that approaches $2 - \sqrt{2}$. 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 $\mathcal{T}_{G,f}^*$ is $\frac{2}{2 - \sqrt{2}} = 2 + \sqrt{2}$. \square

It should be noted that for the directed cycle with exponentially distributed tasks having expectation 2, $\mathcal{T}_{G,f}^*$ approaches 4, even though the exponential distribution is the continuous analogue of the geometric distribution.

So regardless of how many processors are connected in the directed cycle, the time between levels will always approach a value that is bounded above by $2 + \sqrt{2}$. The improvement over the complete graph holds even for very small graphs. It can be shown by numerical calculation that for geometrically distributed tasks with $p = 0.5$, the time between levels in a complete graph of 4 processors is greater than 3.5, which already exceeds our upper bound for a directed cycle of any number of processors. These results also hold for the infinite directed line.

The result for $p = 0.5$ can be used to give bounds for other values of p . For any graph G , if \mathcal{T}_{G,f^*} is the asymptotic time per level using probability distribution f , then $\alpha \mathcal{T}_{G,f^*} + \beta$ is the asymptotic time using distribution $f'(\alpha x + \beta) = f(x)$, and if the cdf of f'' is less than the cdf of f then $T(f'') > T(f)$. Using this, one can show that if $p = 0.25$, then the asymptotic time per level is between $2(2 + \sqrt{2}) - 1$ and $3(2 + \sqrt{2}) + 2$. Similar bounds can be found for all values of p .

More generally, this approach yields the following:

COROLLARY 3.1.2. *For any probability distribution function f defined on the positive reals, with $f(x) = O(C^x)$ for some $C < 1$, the expected time per level in a directed cycle of n processors with task-times given by f is bounded by a constant independent of n .*

Many common distributions, such as normal and Poisson, obey this condition.

4. AN UPPER BOUND FOR NORMALLY DISTRIBUTED TASKS

Now we consider the case in which tasks are each normally distributed with mean μ and variance σ^2 . Chang and Nelson's bound for the time per level works out to $\mu + \sqrt{2 \ln d} \sigma$, where d is the degree of a processor in the synchronization graph. Their proof uses a supermartingale method. Here we prove the same bound using a simpler convexity argument.

We introduce some new notation: let $S_i(j)$ be the time when the i th processor starts the j th task, $F_i(j)$ the time when the i th processor finishes the j th task and $T_i(j)$ the time it takes, not including synchronization delay, for the i th processor to do the j th task (the *task-completion time*.) From the point of view of a particular processor i , the average time between levels after r tasks is $\frac{S_i(r)}{r}$.

Let us consider neighbor synchronization on a directed cycle. We want to calculate $S_i(r)$. We know that $S_i(0)$ is 0. In a directed cycle, processor i waits for only itself and processor $i - 1$. So $S_i(1) = \max(F_i(0), F_{i-1}(0))$, which is the same as $\max(T_i(0), T_{i-1}(0))$, since each processor started the 0th task at time 0. Similarly, we can calculate $S_i(2)$ as

$$\begin{aligned} S_i(2) &= \max(F_i(1), F_{i-1}(1)) \\ &= \max(S_i(1) + T_i(1), S_{i-1}(1) + T_{i+1}(1)) \\ &= \max(\max[T_i(0), T_{i-1}(0)] + T_i(1), \\ &\quad \max[T_{i-1}(0), T_{i-2}(0)] + T_{i-1}(1)) \end{aligned}$$

That is, the time at which processor i starts the third task is the maximum of 4 possible sums of 2 task-completion times each: $T_i(1) + T_i(2)$, $T_{i+1}(1) + T_i(2)$, $T_{i-1}(1) + T_{i-1}(2)$ and $T_{i-2}(1) + T_{i-1}(2)$. Notice that the $T_i(j)$ s are independent, so these are sums of independent random variables. The sums themselves are not independent of each other, but as they are sums with common terms, their maximum is upper bounded by independent sums of independent normal random variables.

Since the i th processor starts the j th task either when it completes the $(j - 1)$ th or when the $(i - 1)$ th processor completes the $(j - 1)$ th, whichever comes last, we can see that number of possible sums to describe the starting time for the j th task is twice the number for the $(j - 1)$ th. So the starting time for a processor to begin the j th task is the maximum of 2^j sums of j task-completion times. By a similar argument, if we have any synchronization graph of maximum degree d , the starting time for a processor to begin the j th task is upper-bounded by the maximum of d^j sums of j task-completion times.

When the task-completion times are normally distributed, this argument gives an upper bound for the expected time between levels in a graph with degree not exceeding d that does not depend at all on the number of processors in the graph or the amount of tasks completed and increases with d as $\Theta(\sqrt{\log d})$.

THEOREM 4.1. *If the task-completion times are identical and normally distributed with parameters μ and σ^2 , and the synchronization graph has maximum indegree d , then the*

expected value for the average time between levels is bounded above by $\mu + \sqrt{2 \ln d} \sigma$.

PROOF. Since our individual task-completion times are normally distributed, sums of r of them are normally distributed as well, with parameters $r\mu$ and $r\sigma^2$. By convexity arguments [2], 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, $\frac{1}{\sqrt{2r\pi}\sigma} e^{-\frac{(x-r\mu)^2}{2r\sigma^2}}$. Since we have d^r of these sums, if we set $f(x)$ equal to $\frac{1}{2d^r}$ and solve the resulting quadratic for x , we obtain an upper bound on the expected time for a processor to start the r th level: $r\mu + r\sigma\sqrt{2 \ln d}$. Dividing by r gives the expected time between levels. \square

Note that the above applies to expander graphs and infinite, as well as finite, graphs. Also note that it is significantly stronger than the application of Corollary 3.1.2.

5. A LOWER BOUND FOR PARETO-DISTRIBUTED TASKS

The Pareto, or power law, distribution is used as a model for many network and computer tasks. Here we show that when tasks are Pareto-distributed, there are graphs of bounded degree for which there is no constant upper bound on the time between levels as the number of processors increases.

We say a processor i *depends on* a task-completion time $T_j(s)$ to start task r if $T_j(s)$ appears in any of the possible sums that could determine when processor i starts task r . For example, if the graph is a directed cycle, i depends on times $T_i(r - 1)$ and $T_{i-1}(r - 1)$ from the last task, $T_i(r - 2)$, $T_{i-1}(r - 2)$ and $T_{i-2}(r - 2)$ from the task before that one, and so on. So it depends on a total of $2 + 3 + \dots + r = \Theta(r^2)$ task-completion times, provided that $r < n$, and $\Theta(\min(r^2, rn))$ in general. If the graph is a D -dimensional grid, directed or undirected, a processor depends on $\Theta(r^{D+1})$ task-completion times to start the r th task, when $n > r$. So even though a processor in an undirected cycle and one in a directed square grid is each directly connected to 3 processors (including itself), a processor in the cycle depends on $\Theta(r^2)$ task-completion times to start the n th task, while the one in the square grid depends on $\Theta(r^3)$. Each processor in a binary tree is also directly connected to 3 processors, but it depends on $\Theta(2^r)$ task completion times to start the r th task. We call the number of task-completion times on which a processor in a graph depends to start the r th task the *dependency function* of that graph.

A lower bound for the time when a processor begins task r is the single worst task-completion time on which the processor depends, as illustrated in Figure 2. So if a processor depends on $\Omega(g(r))$ task-completion times to start the r th task and the expected maximum of k task-completion times is $\Omega(h(k))$, then the expected time for a processor to begin task r is bounded below by $\Omega(h(g(r)))$ and the expected time between levels, $\mathcal{T}_{G,f}(r)$, is bounded below by $\Omega(\frac{1}{r}h(g(r)))$.

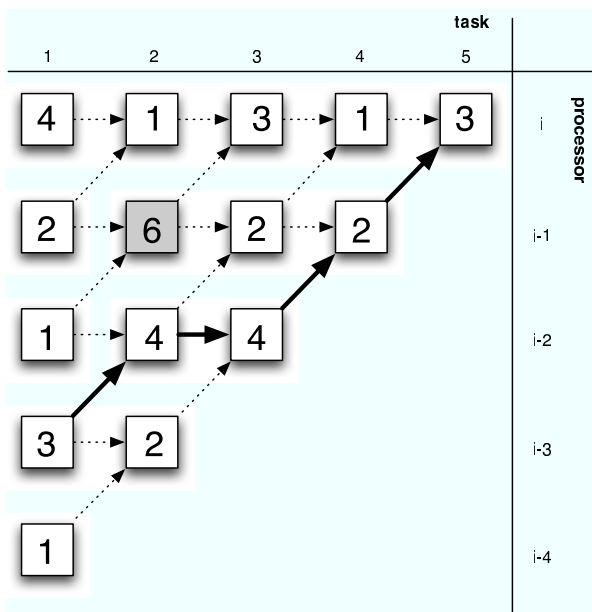


Figure 2: Example times for processors to complete tasks. The longest task time is not on the longest path, but it is a lower bound for the sum along the longest path.

Since graphs with the same bounded degree can have different dependency functions, as described above, this bound depends on the shape of the graph as well as its degree, unlike the upper bound for normally distributed tasks.

For a given class of graphs \mathcal{G} , such as directed cycles, and a given task-completion time distribution f , let $\mathcal{T}_{G,f}(i)$ be the value of $\mathcal{T}_{G,f}(i)$, where G is the element of \mathcal{G} of size n , and let $\mathcal{T}_{n,f}^* = \mathcal{T}_{G,f}^*$. If there is more than one G of size n then take the maximum over all such graphs. Theorem 3.1, for example, states that for the class of directed cycles with geometric task-distribution times, where $p = 0.5$, $\mathcal{T}_{n,f}^*$ is uniformly bounded by $2 + \sqrt{2}$.

For any graph G , $\mathcal{T}_{G,f}(r)$ is an increasing function of r , and hence a lower bound on $\mathcal{T}_G(r)$ for some r is a lower bound on $\mathcal{T}_{G,f}^*$. Thus if we have a class of graphs \mathcal{G} for which $\mathcal{G}_{n,f}(r)$ is an unbounded function of r under the assumption that $n > r$, then there is no finite upper bound on the expected time between levels, $\mathcal{T}_{n,f}^*$, as $n \rightarrow \infty$.

For example, let the task-completion times be Pareto, or power-law, distributed with parameter λ ; that is, the cumulative distribution function is $F(x) = 1 - x^{-\lambda}$. The expected value of the maximum of k random variables distributed such is $k! \frac{\Gamma(1-\frac{1}{\lambda})}{\Gamma(k+1-\frac{1}{\lambda})}$ [6]. Using Stirling's approximation for the gamma function, it can be seen that this expression is $\Theta(k^{\frac{1}{\lambda}})$. So $\mathcal{T}_{n,f}(r) = \Theta(\frac{1}{r}(r^d)^{\frac{1}{\lambda}})$, where d is the degree of the synchronization graph. This is an increasing function of r whenever $\lambda < d$.

If $\lambda < 2$ and the processors are connected in a directed cycle, which means that each processor depends on $\Theta(r^2)$ task-completion times to start the r th task, for $n > r$, $\mathcal{T}_{n,f}(r) =$

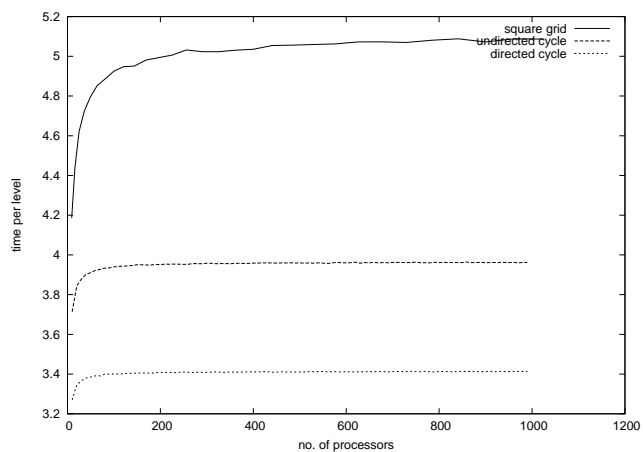


Figure 3: Asymptotic time per level versus n , geometrically distributed task times with $p=0.5$.

$\Theta(\frac{1}{r}(r^2)^{\frac{1}{\lambda}})$, or $\Theta(r^{\frac{2}{\lambda}-1})$, which increases unboundedly with r . This means that there is no uniform upper bound on the expected time between levels for directed cycles with Pareto-distributed tasks with $\lambda < 2$. Furthermore, since the directed cycle provides the best times among all strongly connected graphs, this bound holds for any strongly connected graph.

6. SIMULATION RESULTS

We ran simulations to model three different kinds of local synchronization graphs: the directed cycle, the undirected cycle and the square grid (actually a torus). For every graph, the task time distribution was geometric with $p = 0.5$. Each simulation ran for 100,000 time steps. As n increases, the time between levels in the directed cycle quickly approaches what we have shown to be its limit, $2 + \sqrt{2}$. The time between levels in the undirected cycle appears to display similar behavior, increasing rapidly and then leveling out around 3.96. The time between levels in the square grid has a graph that looks much like those for the other two graphs, as shown in Figure 3.

7. THE FIRST-NEIGHBOR MODEL

Another type of synchronization can be described by the first-neighbor model. In this model, a processor does not wait for all a of its neighbors in the synchronization graph to finish their $(i-1)$ st iterations before starting its i th; instead, it waits for only b of them to finish, where $b < a$, and then proceeds.

Here we consider the case where n processors are connected in a complete graph and each processor starts the i th iteration when itself and $\ell(n)$ of its $n-1$ neighbors have completed the $(i-1)$ st. Adapting the method of Chang and Nelson, we can show that if $\ell(n) = O(\sqrt{n})$ and f has all finite moments, then $\mathcal{T}_{G,f}^*$ is bounded above by a constant as n goes to infinity.

THEOREM 7.1. *In the first-neighbor model as described above where $\ell(n) = O(\sqrt{n})$ and f has all finite moments,*

the constant upper bound of Chang and Nelson for a directed cycle applies.

PROOF. Chang and Nelson define a *loop* as a graph in which the node (i, j) corresponds to the j th processor on its i th iteration. Node (i, j) has an incoming edge from $(i-1, k)$ if processor j 's i th iteration requires data from processor k 's $(i-1)$ st.

In the loops they study, each node has a number of incoming edges that is bounded by a constant as the number of processors n goes to infinity. In our case, this is not true; a node can have as many as \sqrt{n} incoming edges. However, the *average* number of incoming edges over all n nodes corresponding to a particular iteration is bounded by a constant: Let $\ell(n) = \sqrt{n}$. The first processor to complete the i th iteration must wait for \sqrt{n} processors (itself and $\sqrt{n} - 1$ neighbors) to complete theirs before proceeding. The second must wait for $\sqrt{n} - 1$ processors, and so on. Any processor that is the $(n - \sqrt{n})$ th or above to finish waits for only one processor, itself.

So the average incoming degree for a node in a particular iteration is

$$\frac{1}{n} (\sqrt{n} + (\sqrt{n} - 1) + (\sqrt{n} - 2) + \dots + 1(n - \sqrt{n})) \approx 1.5$$

This is, of course, less than the number of dependencies for each under directed-cycle synchronization, 2. Furthermore, the expected time for a node along a path in the loop induced by this synchronization is less than or equal to the unconditional expectation of the task distribution, since processors that finish tasks earlier are included along more paths than those that finish later. Using these two facts, we can apply their supermartingale that sums along all possible paths and show that their bound applies here (for more details, see [1].) \square

8. CONCLUSIONS AND FUTURE WORK

We have shown that the expected time for a processor completing geometrically distributed tasks under directed-cycle local synchronization approaches a limit of $2 + \sqrt{2}$ as the number of processors n goes to infinity. This is in contrast to the $\Theta(\log n)$ behavior when global synchronization is used. We have also given a new proof of the constant upper bound for the expected time of a processor completing normally-distributed tasks under any kind of bounded-degree synchronization. However, we have shown that when the task times are distributed as certain heavy-tailed random variables, the time between tasks can be unbounded with the number of processors.

Finally, we have investigated a different synchronization model, where processors wait only for their fastest neighbors in the synchronization graph, and shown that constant bounds apply to some of its variants.

We would also like to investigate models in which a processor must synchronize with only a subset of those adjacent to it, such as the first-neighbor model. In one such variant, a processor chooses a subset of neighbors with which

to synchronize randomly at the beginning of each task. In not yet published research, we show that the lower bound we have described here for Pareto-distributed tasks under directed-cycle neighbor synchronization also applies when the processors are connected in an undirected cycle and each decides randomly before beginning a task whether to synchronize with its left or right neighbor. Yet another natural generalization is to include delays on the edges.

9. REFERENCES

- [1] 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.
- [2] H. A. David. *Order Statistics*. Wiley, 1970.
- [3] A. G. Escribano, V. C. Payo, and A. van Gemund. On the loss of parallelism by imposing synchronization structure. In *Proceedings 1st EURO-PDS Int'l Conference on Parallel and Distributed Systems*, pages 251–256, June 1997.
- [4] A. G. Escribano and A. van Gemund. An algorithm for transforming NSP to SP graphs. Technical report, Technical University, Delft, The Netherlands, 1996.
- [5] H. Han, C. Tseng, and P. Keleher. Eliminating barrier synchronization for compiler-parallelized codes on software DSMs. *International Journal of Parallel Programming*, 26(5):591–612, 1998.
- [6] J. S. Huang. A note on order statistics from the Pareto distribution. *Scandinavian Actuarial Journal*, 3:881–889, 1975.
- [7] J. F. C. Kingman. The first-birth problem for an age-dependent branching process. *Annals of Probability*, 3(5):790–801, 1975.
- [8] U. Legedza and W. Wehl. Reducing synchronization overhead in parallel simulation. In *Workshop on Parallel and Distributed Simulation*, pages 86–95, 1996.
- [9] A. Malony, V. Mertsiotakis, and A. Quick. Stochastic modeling of scaled parallel programs. In *Proceedings of the International Conference on Parallel and Distributed Systems*, pages 274–279, December 1994.
- [10] 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.
- [11] A. Salamon. Task graph performance bounds through comparison methods. Technical report, Department of Computer Science, University of the Witwatersrand, Johannesburg, South Africa, Jan. 2001. MSc Dissertation (141 pages).
- [12] R. Stanley. *Enumerative Combinatorics*, volume 2. Cambridge University Press, 1999.
- [13] T. Tabe, J. Hardwick, and Q. Stout. Statistical analysis of communication on the IBM SP2. *Computing Science and Statistics*, 27:347–351, 1995.