[PF77] Gary L. Peterson and Michael J. Fischer. Economical solutions for the Critical Section Problem in a distributed system. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, pages 91–97, May 1977.

[PU87] Christos H. Papadimitriou and Jeffrey D. Ullman. A communication-time trade-off. *SIAM Journal on Computing*, 16(4):639–646, August 1987.

[PY88] Christos H. Papadimitriou and Mihalis Yannakakis. Towards an architecture-independent analysis of parallel algorithms. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 510–513, 1988.

[Wyl79] James C. Wyllie. *The Complexity of Parallel Computation.* PhD thesis, Cornell University, August 1979. Technical report number TR 79-387, Department of Computer Science.

[CZ91a] Richard Cole and Ofer Zajicek. The APRAM — The rounds complexity measure and the explicit costs of synchronization. Courant Institute Technical Report No. 539, 1991.

[CZ91b] Richard Cole and Ofer Zajicek. Asynchronous graph connectivity in O($\log n$) rounds. Courant Institute Technical Report No. 546, 1991.

[Gib89] Phillip B. Gibbons. Towards better shared memory programming models. In *Proceedings of 1st ACM Symposium on Parallel Algorithms and Architectures*, pages 158–168, 1989.

[HR89] Torben Hagerup and Christine Rub. A guided tour of Chernoff bounds. Universitat des Saarlandes, Computer Science Technical Report No. A88/03, 1989.

[KRS88a] Clyde P. Kruskal, Larry Rudolph, and Marc Snir. A complexity theory of efficient parallel algorithms. In *Proceedings of the 15th International Colloquium on Automata, Languages and Programming*, pages 333–346. Springer-Verlag, July 1988.

[KRS88b] Clyde P. Kruskal, Larry Rudolph, and Marc Snir. A complexity theory of efficient parallel algorithms. Technical Report RC 13572, International Business Machines, 1988.

[Lub88] Michael Luby. On the parallel complexity of symmetric connection networks. Technical Report No. 214/88, University of Toronto, Department of Computer Science, 1988.

[LF81] Nancy A. Lynch and Michael J. Fischer. On describing the behavior and implementation of distributed systems. *Theoretical Computer Science*, 13:17–43, 1981.

[MPS89] Charles Martel, Arvin Park, and Ramesh Subramonian. Optimal asynchronous algorithms for shared memory parallel computers. Computer Science and Engineering CSE-89-8, University of California at Davis, July 1989.

[MS90] Charles Martel and Ramesh Subramonian. Asynchronous pram algorithms for list ranking and transitive closure. UCDavis, manuscript, January 1990.

[MSP90] Charles Martel, Ramesh Subramonian, and Arvin Park. Asynchronous PRAMs are (almost) as good as synchronous PRAMs. In *Proceedings of the 31st Annual Symposium on the Foundations of Computer Science*, 1990.

[Nis90] Naomi Nishimura. Asynchronous shared memory parallel computation. In *Proceedings of 2nd ACM Symposium on Parallel Algorithms and Architectures*, pages 76–84, 1990.

We have shown that for any constant, c, if $1/(\log n)^c \leq p \leq 1/3^{2.5}$ and $pk \leq 1/18$ then $E_p(RD) = \Theta(\log n + k \log\log n)$ rounds, where RD stands for the recursive doubling algorithm. On the other hand, we have shown that if pk is a constant, $pk < 1$, the expected rounds complexity of the summation algorithm, $E_p(Sum) = \Theta(rk \log n)$.

For example, when $k = \log n / \log\log n$ and when $p = \frac{l}{k}$ for some constant l, $E_p(RD) = \Theta(\log n)$ while $E_p(Sum) = \Theta(\log^2 n / \log\log n)$.

References

[AC88] Alok Aggarwal and Ashok K. Chandra. Communication complexity of PRAMs. In *Proceedings of the 15th International Colloquium on Automata, Languages and Programming*, pages 1–17. Springer-Verlag, July 1988.

[ACS89] Alok Aggarwal, Ashok K. Chandra, and Marc Snir. On communication latency in PRAM computations. In *Proceedings of 1st ACM Symposium on Parallel Algorithms and Architectures*, pages 11–21, 1989.

[AW91] Richard J. Anderson and Heather Woll. Wait-free parallel algorithms for the Union-Find problem. In *Proceedings of the 23rd ACM Symposium on Theory of Computing*, pages 370–380, 1991.

[AFL83] Eshrat Arjomandi, Michael J. Fischer, and Nancy A. Lynch. Efficiency of synchronous versus asynchronous distributed systems. *Journal of the ACM*, 30(3):449–456, July 1983.

[AG87] Baruch Awerbuch and Robert G. Gallager. A new distributed algorithm to find breadth first search trees. *IEEE Transactions on Information Theory*, IT-33(3):315–322, May 1987.

[Awe85] Baruch Awerbuch. Complexity of Network Synchronization. *Journal of the ACM*, 32(4):804–823, October 1985.

[Awe87] Baruch Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 230–240, May 1987.

[CZ89] Richard Cole and Ofer Zajicek. The APRAM: Incorporating asynchrony into the PRAM model. In *Proceedings of 1st ACM Symposium on Parallel Algorithms and Architectures*, pages 169–178, 1989.

[CZ90] Richard Cole and Ofer Zajicek. The expected advantage of asynchrony. In *Proceedings of 2nd ACM Symposium on Parallel Algorithms and Architectures*, pages 85–94, 1990.

Let $s = 3\log n/(\log\log n)^2$ and consider the list as if it is divided into n/s segments of s contiguous items. Let μ be a constant to be specified later. We call a segment *superslow* if the first $\mu \log\log n$ events of each of the processes associated with the segment were slow. If after $k\mu \log\log n$ rounds the first element, x, in a superslow segment, S, points to an element, y, inside S, y is at distance exactly $2^{\mu \log\log n} = \log^{\mu} n$ from x. If $\mu \leq \frac{1}{3}$ it is straightforward to show that for any $n \geq 4$, $\log^{\mu} n < s$. Consequently, the first element of S points to an element strictly inside S and the algorithm could not have terminated.

Consider a segment, S, of length s. Let P_s be the probability that S is superslow. Then

$$P_s = p^{s\mu \log\log n} \geq (\log n)^{-3\alpha\mu \log n/\log\log n} = n^{-3\alpha\mu}.$$

There are $n/s = n(\log\log n)^2/(3\log n)$ such segments. Let Q_s be the probability that at least one such segment is superslow. Then the probability that no segment is superslow, $1 - Q_s$, is

$$\begin{aligned} 1 - Q_s &= (1 - P_s)^{n/s} \\ &\leq (1 - n^{-3\alpha\mu})^{\frac{n(\log\log n)^2}{3\log n}} \\ &= \left((1 - n^{-3\alpha\mu})^{n^{3\alpha\mu}}\right)^{n^{1-3\alpha\mu}(\log\log n)^2/(3\log n)} \\ &\quad \text{Assuming } 3\alpha\mu \leq \tfrac{1}{2}, \text{ then for } n \geq 16 \\ &\leq \left(1 - n^{-3\alpha\mu}\right)^{n^{3\alpha\mu}} \leq \frac{1}{e}. \end{aligned}$$

We have shown:

Theorem 4.4 *For any number, α, $\alpha \geq 1$, for any $\mu \leq \frac{1}{3}$ such that $\alpha\mu \leq \frac{1}{6}$, and for any $n \geq 16$ if $p \geq (\log n)^{-\alpha}$ the recursive doubling algorithm requires at least $k\mu \log\log n$ rounds with probability at least $1 - \frac{1}{e}$.*

Corollary 4.4 *For any number, α, $\alpha \geq 1$, for any number, μ, $\mu \leq \frac{1}{3}$ such that $\alpha\mu \leq \frac{1}{6}$, and for $n \geq 16$ if $p \geq (\log n)^{-\alpha}$ then $E_p(A) \geq \frac{1}{2}(\log n + (1 - \frac{1}{e})k\mu \log\log n) \geq \frac{(e-1)\mu}{2e}(\log n + k\log\log n)$ rounds.*

Theorem 4.5 *For any constant, $c \geq 1$, if $1/(\log n)^c \leq p \leq 1/3^{2.5}$ and $pk \leq 1/18$ then $E_p(A) = \Theta(\log n + k\log\log n)$ rounds.*

Summary of the Results Under the Bounded Delays Model. Under the bounded delays model the list based recursive doubling algorithm performs better than the tree based summation algorithm under certain conditions.

Each node of the execution tree at level t will be eliminated with probability q_1. If we choose $c = 2$, the probability that at least $1/2$ the nodes at level t are trimmed is at most $(2/3)^{2\log_{3/2} n} = 1/n^2$. Therefore, for any element, x, any round, t, and for $M = \log_{2/3} n$, the probability that the tree rooted at $\langle x, t + 2M\rangle$ does not have M nodes at level t is at most $1/n^2$. We have shown:

Lemma 4.4 *For $\beta = 11/10$, for any element, x, and for any number, t, $t \geq 2\lceil \log_\beta n \rceil$, the probability that after $t + k(1 + \log\log_{3/2} n)$ rounds x has fewer than n leaf descendants is at most $\frac{3}{n^2} + \mathrm{O}(1/n^{2\beta})$.*

Proof of Theorem 4.3. Consider the root of the untrimmed computation DAG. For any point, w of the computation DAG, and any level, t, let $Desc(w, t)$ be the set comprising the level t descendants of w. By the above arguments, for any level, t, for any number, $m \geq 2$, and for any node, x, $|Desc(\langle x, t + k \log m\rangle, t)| \geq m$.

Applying Lemma 4.4 to each member, y, of $Desc(\langle x, t + k\log m\rangle, t)$ we get that for some constant, h, if $t \geq h(\log n + k \log\log n)$ the probability that y does not have n leaf descendants is at most $p_1 = 3/n^2 + \mathrm{O}(1/n^{2\beta})$. The number of leaf descendants of $\langle x, t + k\log m\rangle$ is the sum over all members, y, of $Desc(\langle x, t + k \log m\rangle, t)$ of the number of leaf descendants of y. In particular, if any such y has n leaf descendants then $\langle x, t + k\log m\rangle$ has at least n leaf descendants. It follows that for some constant, h_1, if x is at level $h_1(\log n + k(\log m + \log\log n))$ the probability that x does not have at least n leaf descendants is at most $(p_1)^m$. The input list has n elements, therefore, for any number, m, $m \geq 2$, the recursive doubling algorithm terminates in $h_1(\log n + k(\log m + \log\log n))$ rounds with probability at least $1 - \mathrm{O}(n \cdot 1/n^{2m}) = 1 - \mathrm{O}(1/n^{2m-1})$. □

Lower Bounds. Next, we show that this bound is tight by showing that for any constant, $\alpha \geq 1$, if $p \geq (\log n)^{-\alpha}$ then for some constant, h', the computation has length at least $r \cdot \max\{h' k \log\log n, \log n\}$ rounds, with high probability.

Consider the uniform adversary, $\mathcal{A}$: each process is given one pseudo-event in each round and to each event $\mathcal{A}$ associates elimination probability p and delay k. Consider any random computation, C. For any process, P, and any event, e, of process P, the distance between P and its parent after round t is at most 2^t; thus, $|C| \geq r \log n$.

For each event, e, of C, we say that e is slow if the corresponding pseudo-event tossed *slow* (and as a consequence the next $k - 1$ pseudo-events of the same process were deleted from the sequence of pseudo-events).

We analyze the algorithm using the productive probability as before. However, we have to augment the argument to take into account nodes which have no children. Suppose the execution tree has a level with M nodes, M to be chosen later. When M is sufficiently large the probability that the level is productive (i.e., the probability that the next level down has a constant factor more nodes than the current level) is very large. We underestimate the termination probability by computing the probability that all sufficiently large levels of the tree are productive.

For any number, l, we call a level with l nodes *nonproductive* if either it has at least $l/5$ nodes with no children or it has at least $l/2$ nodes with 1 child. It follows that if a level is productive it has at least $\frac{11}{10}l$ children.

For any number, M, if $q_0 \leq 1/3^5$ then the probability that a level with at least M nodes has at least $M/5$ nodes with no children is at most $(2/3)^M$. Similarly, if $q_1 \leq 1/9$ then the probability that a level with at least M nodes has at least $M/2$ nodes with 1 child is at most $(2/3)^M$. Thus, the probability that a level with at least M nodes is not productive is at most $2(\frac{2}{3})^M$. Setting $M = 2\log_{3/2} n$, we get:

Lemma 4.3 *Given $p \leq 2/27$ and $pk \leq 1/18$, for $\beta = 11/10$ and for $t = \lceil \log_\beta n \rceil$, if $|C^t| \geq 2\log_{3/2} n$ then $Pr[|C^0| < n] \leq \frac{2}{n^2} + O(\frac{1}{n^{2\beta}})$.*

Proof. Assume that $|C^t|$ has at least $2\log_{3/2} n$ nodes. $q_0 \leq p^2 k \leq 1/3^5$. Also, $q_1 \leq 2pk \leq 1/9$. Starting from level t down, if the first l levels are productive then level $t-l$ has at least $\beta^l 2\log_{3/2} n$ elements. Therefore, the probability that some level of the tree, below level t, is non-productive is at most

$$Pr \quad \leq \quad \sum_{0 \leq i < t} 2\left(\frac{2}{3}\right)^{2\beta^i \log_{3/2} n} = \sum_{0 \leq i < t} 2n^{-2\beta^i} = \frac{2}{n^2} + O(n^{-2\beta}).$$

If all the levels are productive, the tree has at least n leaves. □

The analysis is concluded by showing that some level of the tree has at least $M = 2\log_{3/2} n$ nodes. Consider the computation as if it is proceeding in phases, each comprising k rounds and consider the untrimmed computation DAG and its associated execution tree. Each process is guaranteed to execute at least one operation in each phase. Therefore, each node at level $r + k$ has at least 2 descendants at level r. Let $f(n) = \log(cM)$, for some c, $c \geq 2$. It follows that for any element, x, and any level, t, the point $\langle x, t + k \cdot f(n)\rangle$ has at least cM descendants at level t. We selectively trim the computation DAG eliminating all long edges which cross a level, t', $t' \leq t$. This may eliminate some nodes from level t which were descendants of $\langle x, t + k \cdot f(n)\rangle$.

Theorem 4.2 *For any number, $c < 1$, if $p(k-1) \geq \frac{c}{1-c}$, the summation algorithm requires at least $r \log n(1 + \frac{c}{16}(k-1))$ rounds with probability at least $1 - 1/n^{cr\sqrt{n}/(32 \ln 2)}$.*

In particular, if the number c is a constant, we get:

Corollary 4.2 *For any constant, c, $c < 1$, if $p(k-1) \geq \frac{c}{1-c}$, the expected complexity of the summation algorithm is $\Theta(kr \log n)$ rounds.*

4.2 Recursive Doubling

Now, we consider the recursive doubling algorithm under the bounded delays measure with parameters p and k. Under the synchronous variation the algorithm proceeds like the summation algorithm in $r \log n$ levels, for some constant r. Therefore, when $p \geq 1/n^{1/2}$ the expected rounds complexity for the synchronous variation is $\Theta(rk \log n)$.

The main result of this section is stated in the following theorem:

Theorem 4.3 *If $p \leq 1/3^{2.5}$ and $pk < 1/18$, then for any number, $c > 2$, the algorithm terminates in $O(\log n + k(\log c + \log\log n))$ rounds with probability at least $1 - 1/n^{2c-1}$.*

Corollary 4.3 *If $p \leq 1/3^{2.5}$ and $pk < 1/18$, the expected number of rounds the algorithm executes is $O(\log n + k \log\log n)$.*

The analysis of the recursive doubling algorithm under the bounded delay model is similar to the analysis given for the unbounded delays model, but more involved. The mapping of the computation to an execution tree is the key to making the analysis tractable. This allows complicated cases due to slow processes to be circumvented by pruning the tree; this overestimates the number of rounds required, but only by constant factors.

We start, as before, with the computation DAG. The long delays appear in the graph as long vertical edges. We underestimate the progress made by a node by ignoring the progress it made before a long delay. We achieve this by trimming the DAG, eliminating all the long vertical edges and all cross edges whose head falls on a long vertical edge.

We define the execution tree as before; however, the tree may contain nodes with no children. A node will have no children if after a long delay it executed a doubling operation, jumping over a node which was in the middle of a long delay; this happens with probability $q_0 \leq p^2 k$. A node will have 1 child in one of two cases: If after a long delay it jumped over a "normal" node or if it is a normal node which jumped over a node which is in the middle of a long delay. Each of these cases happens with probability at most pk; so, the probability, q_1, that a node has 1 child is at most $2pk$. Otherwise, the node has two children; this happens with probability $q_2 = 1 - (q_0 + q_1)$.

Proof. It follows from Lemma 4.1 that the probability that a process did not have an event in the round after its child was validated is $\frac{p(k-1)}{1+p(k-1)}$. Given that the process was waiting, following the validation of its child it is equally likely to wait any number of rounds between 1 and $k-1$. Therefore, the probability that it had to wait at least $\lfloor (k-1)/2 \rfloor$ rounds is

$$\frac{p(\lceil (k-1)/2 \rceil)}{1+p(k-1)}.$$

as desired. $\square$

Let $m = \frac{1}{2} r \log n$ and consider all the nodes of the tree at height m. We compute the probability that at least one of these nodes is α-late, for some number α. Note that there are $\sqrt{n}$ nodes at height m so there are $\sqrt{n}$ node disjoint paths from the leaves to nodes at height m. Consider one such path, P, and let S be the number of slow nodes on P. We use the Chernoff bounds [HR89]: for any number, ϵ, $0 < \epsilon < 1$, the probability that P has fewer than $(1-\epsilon)mq_s$ slow nodes is

$$Pr[S \leq (1-\epsilon)mq_s] \leq e^{-\frac{1}{2}\epsilon^2 mq_s}$$

When $\epsilon = \frac{1}{2}$,

$$Pr\left[S \leq \frac{1}{2}mq_s\right] \leq e^{-\frac{1}{8}mq_s} \leq e^{-\frac{1}{16}m\frac{p(k-1)}{1+p(k-1)}}.$$

It then follows that

$$Pr\left[S \leq \frac{m}{4}\frac{p(k-1)}{(1+p(k-1))}\right] \leq Pr\left[S \leq \frac{1}{2}mq_s\right] \leq e^{-\frac{1}{16}m\frac{p(k-1)}{1+p(k-1)}}.$$

Let c be any number, $c < 1$, and assume that $p(k-1) \geq \frac{c}{1-c}$. Then $\frac{p(k-1)}{1+p(k-1)} \geq c$. It follows that

$$\begin{aligned} Pr\left[S \leq \frac{cm}{4}\right] &\leq Pr\left[S \leq \frac{mp(k-1)}{4(1+p(k-1))}\right] \\ &\leq e^{-\frac{1}{16}mc} \leq e^{-\frac{1}{32}cr\log n} \\ &\leq e^{-\frac{1}{32}cr\frac{\ln n}{\ln 2}} \leq n^{-\frac{cr}{32\ln 2}}. \end{aligned}$$

In other words, for each node, e, at level m, the probability that e is not $\frac{c}{4}$-late is at most $n^{-\frac{cr}{32\ln 2}}$. The tree has $\sqrt{n}$ nodes at level m. Thus, the probability that none of them is $\frac{c}{4}$-slow is at most $n^{-\frac{cr\sqrt{n}}{32\ln 2}}$.

Thus, for some path, its computation requires at least $r\log n + \frac{mc(k-1)}{8} = r\log n(1 + \frac{c}{16}(k-1))$ rounds with probability at least $1 - 1/n^{cr\sqrt{n}/(32\ln 2)}$. We have shown:

Lemma 4.1 *For any round, r, and for any process, l, the probability, $Q_l(r)$, that process l has an event in round r is*

$$Q_l(r) = \frac{1}{1+p(k-1)}.$$

Proof. Consider a process, l. The proof is by induction on the round number, r. We start with the induction step: If $r \geq k$ then l has an event in round r in one of two cases.

1. If l has an event in round $r-1$ then l has an event in round r if and only if it tossed "normal" at round $r-1$.

2. If l does not have an event in round $r-1$ then l has an event in round r if and only if process l has an event at round $r-k$ and it tossed "delayed" at that round.

In other words, if $r \geq k$ then

$$Q_l(r) = p \cdot Q_l(r-k) + (1-p) \cdot Q_l(r-1).$$

The proof is concluded by showing that the lemma holds for all r, $0 \leq r \leq k-1$.

Assume that $r \leq k-1$. Then the only way in which process l can have an event in round r is if all the pseudo-events of process l with round number less than r which were not eliminated by the Init step tossed "normal." The probability that no pseudo-events were eliminated in the Init step is $\frac{1}{1+p(k-1)}$ by design. If the Init step eliminates pseudo-events, then for any number, i, $1 \leq i \leq k-1$, the Init step eliminates exactly i pseudo-events with probability $\frac{p}{1+p(k-1)}$. Therefore, if $r \leq k-1$

$$\begin{aligned}
Q_l(r) &= \frac{1}{1+p(k-1)} \cdot (1-p)^r + \frac{p}{1+p(k-1)} \sum_{1 \leq i \leq r} (1-p)^{r-i} \\
&= \frac{1}{1+p(k-1)} \left((1-p)^r + p \sum_{0 \leq i \leq r-1} (1-p)^i \right) \\
&= \frac{1}{1+p(k-1)} \left((1-p)^r + p \frac{1-(1-p)^r}{p} \right) \\
&= \frac{1}{1+p(k-1)}.
\end{aligned}$$

This concludes the proof of the lemma. □

Let q_s be the probability that a node in the tree is slow. Then

Lemma 4.2 *For any node, e, in the summation tree, the probability, q_s, that e is slow is*

$$q_s = \frac{p(\lceil (k-1)/2 \rceil)}{1+p(k-1)} \geq \frac{1}{2} \frac{p(k-1)}{(1+p(k-1))}.$$

The probability, $P_m(\alpha, l)$, that there is a path with at least αl slow nodes is then at most

$$P_m(\alpha, l) \leq n \cdot \binom{r \log n}{\alpha l} q^{\alpha l} \leq n^{r+1} q^{\alpha l}.$$

Thus, whenever $q \leq n^{-\frac{r+1}{l}}$, $P_m(\alpha, l) \leq n^{(r+1)(1-\alpha)}$.

Theorem 4.1 *For any two numbers, l, and α, $l, \alpha \geq 1$, if $pk \leq n^{-\frac{r+1}{l}}$, the probability that the length of the computation exceeds $r \log n + k\alpha l$ is at most $1/n^{(r+1)(\alpha-1)}$.*

Proof. Suppose every path has at most αl slow nodes. Then the computation completes in at most $r \log n + k \cdot \alpha l$ rounds. □

Corollary 4.1 *For any number, l, $l \geq 1$, if $pk \leq n^{-\frac{r+1}{l}}$, the expected length of a computation of the summation algorithm is at most $r \log n + 2kl + \mathrm{O}(kl/n^{r+1})$.*

Proof. The expected length of a computation, $E_p(A)$, is

$$\begin{aligned} E_p(A) &= \sum_{j \geq 0} NT_p(A, j) \leq r \log n + kl + \sum_{j > 0} NT_p(A, r \log n + kl + j) \\ &\leq r \log n + kl + \sum_{j \geq 0} kl \cdot NT_p(A, r \log n + kl(1+j)) \\ &\leq r \log n + kl(1 + \sum_{j \geq 0} 1/n^{j(r+1)}) \\ &\leq r \log n + 2kl + \mathrm{O}(\frac{kl}{n^{r+1}}). \end{aligned}$$

□

Lower Bounds. Now, we consider the lower bound on the complexity of the parallel summation algorithm under the bounded delays model. Consider a single path P, from a leaf to the root. We call a node v on the path *slow* if when v's child w is validated, it takes at least $\lfloor (k-1)/2 \rfloor + 1$ rounds before v completes the execution of its next event. For any number, $\alpha \leq 1$, and for any node, e, we say that e is α-late if there is a path P from a leaf to e which contains at least $\alpha height(e)$ slow nodes, where $height(e)$ denotes the height of node e in the tree.

Consider the uniform adversary, $\mathcal{A}$: each process is given one pseudo-event in each round and to each pseudo-event $\mathcal{A}$ associates a probability p of elimination and a delay k. We show that for some constant, α, the root of the tree is α-late with high probability. We start with some preliminary results.

For any round, r, and for any process, l, let $Q_l(r)$ be the probability that process l has an event at round r. We show:

The expected number of rounds to complete the whole computation is:

$$\begin{aligned}
E &= r \sum_{i=0}^{\log n - 1} E_{2^i} \\
&= r \sum_{i=0}^{\log n - 1} [k - (k-1)(1-p)^{2^i}] \\
&= rk \log n - r(k-1) \left[\sum_{i=0}^{\log \frac{1}{p} - 1} (1-p)^{2^i} + \sum_{i=\log \frac{1}{p}}^{\log n - 1} (1-p)^{2^i} \right] \\
&\geq rk \log n - r(k-1) \left[\log \frac{1}{p} + \sum_{i=0}^{\log n - 1 - \log \frac{1}{p}} (e^{-2^i}) \right] \\
&\geq rk(\log n - \log \frac{1}{p}) + r \log \frac{1}{p} - r(k-1) \frac{1}{e-1} \\
&\qquad \text{When } p = n^{-\epsilon} \text{ for } \epsilon < 1 \\
&= \Omega(k(1-\epsilon)\log n)
\end{aligned}$$

When $\epsilon \leq \frac{1}{2}$, it is easy to check that the underestimate is only by a constant factor. Thus, if $p \geq 1/n^{1/2}$ the expected rounds complexity for the synchronous variation is $\Theta(k \log n)$.

Now, consider the asynchronous case. An argument identical to that given for the unbounded delays scenario shows that it suffices to determine the complexity of computing n paths of length $r \log n$, where there is a separate process for each node; each process performs events with the goal of validating its node, v; if v's child is valid, then v can be validated; otherwise its process executes a null statement. The analysis is complicated by the fact that when a node is validated its parent's process may be in the middle of a long delay. It is convenient, where no confusion will result, to refer to a node and its process interchangeably.

Consider all the nodes on a path P. We call a node *slow* if its process is in the middle of a delay when its child is validated. Let q be the probability that a node is slow. Then $q \leq pk$.

The algorithm has $\log n$ levels of superevents, each comprising r events. For any two numbers, α and l, $\alpha \geq 1$ and $l \geq 1$, the probability, $P(\alpha, l)$, that a path has at least αl slow nodes is

$$P(\alpha, l) \leq \binom{r \log n}{\alpha l} q^{\alpha l}.$$

a coin which has probability $p_1 = \frac{p_0(d_0-1)}{1+p_0(d_0-1)}$ of showing *delayed* and a probability of $1 - p_1 = \frac{1}{1+p_0(d_0-1)}$ of showing *normal.* If the coin shows delayed, pick a number, m, between 1 and $d_0 - 1$ uniformly at random and eliminate the first m pseudo-events of process id.

Iter: Let e_i denote the ith pseudo-event of process id. Let p_i and d_i be respectively the probability and delay associated with e_i by the adversary. If e_i has not been eliminated (it can be eliminated either in an application of step Init or in an earlier application of this step) toss a coin which has probability p_i of showing *delayed* and probability $1 - p_i$ of showing *normal.* In either case pseudo-event i remains in the sequence. However, if the coin shows delayed, the $d(i) - 1$ pseudo-events of process id immediately following pseudo-event i are eliminated. This corresponds to a long duration for the event of process id following event e_i.

The remaining pseudo-events are events and comprise the computation.

As before, the probability of a computation C is the probability that it arises in the above elimination game. The nontermination probability and the expected length of a computation are defined as before. We require the adversaries to be *conforming*, that is, for any round, r, and for any process, id, we require that the probability that process id starts a delay at round r is at most p.

The bounded delays model is not as robust as the unbounded delays model. The first property, monotonicity, holds, but the other two do not. Therefore, we must analyze the algorithms using events and rounds rather than superevents and superrounds. The complexity bounds we obtain here are not obviously tight, so we provide lower bounds on the performance of the algorithms as well.

4.1 Summation Algorithm

Now, we consider the summation algorithm under the bounded delays measure with parameters p and k. We start by analyzing the synchronous variation.

Consider the computation of time step i. If none of the operations forming time step i are delayed, time step i takes one round, otherwise, it takes k rounds. Therefore, the number of rounds to complete a step with m processes has the following distribution:

$$P(m) = \begin{cases} 1 & \text{with probability } (1-p)^m \\ k & \text{with probability } 1-(1-p)^m \end{cases}$$

The expected number of rounds it takes to complete a step involving m nodes is $E_m = k - (k-1)(1-p)^m$. Each level of the computation comprises some r events, r a constant.

Now, assume that the root is at level t and that $t \geq 2\log_{3/2} n$. Let $Q(t,l)$ denote the probability that at least l levels of a computation tree with t levels are not productive.

$$\begin{aligned}
\Pr\left[|C^0| < n \middle| t \geq 2\log_{3/2} n\right] &\leq Q(t, t-\log_{3/2} n | t \geq 2\log_{3/2} n) \\
&\leq \binom{t}{t-\log_{3/2} n} q^{t-\log_{3/2} n} \\
&\leq 2^t q^{t-\log_{3/2} n} \leq (4q)^{\frac{t}{2}}.
\end{aligned}$$

On restricting $p \leq \frac{1}{9}$, we get that $4q \leq 4p \leq \frac{4}{9}$ and thus

$$\Pr\left[|C^0| < n \middle| t \geq 2\log_{3/2} n\right] \leq \left(\frac{2}{3}\right)^t$$

□

4 The Bounded Delays Measure

The bounded delays measure has two parameters: p, the probability of being delayed, and k, the length of the longest duration. We have the following scenario in mind. In a multiuser parallel environment the operating system may need to preempt a process for a relatively long time but not long enough to justify reallocating the work it was performing. In a well designed system we expect such interference to be infrequent. We capture such behavior in the bounded delays model with parameters p and k: the probability that the duration is larger than 1 is at most p and the duration of an event is bounded by k.

As for the unbounded delays measure, a probabilistic computation for the bounded delays measure is obtained with the aid of an adversary. The adversary creates an interleaving of the pseudo-events as before, by attaching a distinct real time to each pseudo-event; also, it associates a probability of elimination with each pseudo-event, as well as an integer future delay. The future delay is bounded by k.

Given an adversary, $\mathcal{A}$, in order to obtain a computation we start with an infinite sequence, $S = e_0, e_1, \ldots$, of pseudo-events, where each process has at least one pseudo-event in each round (a unit of time). The computation is obtained in two steps: For each process, the first step (Init) defines the delay before the first event of the process. The second step (Iter) defines the durations of all other events. The need for the first step will become clear in the lower bound discussion of Section 4.1 (see Lemma 4.1 (p. 20)).

Init: Let *id* be any process. Let e_0 be the first pseudo-event for process *id*. Let p_0 and d_0 be respectively the probability and delay associated with e_0 by the adversary. Toss

Corollary 3.6 *For any $l \geq 1$, if $p \leq (\frac{1}{9})^{1/l}$, the expected length of a computation of the recursive doubling algorithm is at most $2l \log_{3/2} n + o(1)$.*

Proof. The expected length of a computation of the recursive doubling algorithm is:

$$\begin{aligned}
\sum_j NT_p(A,j) &= \sum_{j<2l\log_{3/2} n} NT_p(A,j) + \sum_{j \geq 2l \log_{3/2} n} NT_p(A,j) \\
&\leq 2l \log_{3/2} n + \sum_{j \geq 2l\log_{3/2} n} n \left(\frac{2}{3}\right)^{\frac{j}{l}} \\
&= 2l \log_{3/2} n + o(1).
\end{aligned}$$

□

Proof of Theorem 3.2. Let C be any computation and let x be any element in the list. Consider the computation tree (of C) relative to x. For any i and l, let $Y(i,l)$ denote the number of children of $\langle i,l \rangle$ in the computation tree. The probability that a process does not execute an event in a given round is independent of all other events (by the measure assumption). Therefore, the variables $Y(\cdot,\cdot)$ treated as random variables, are independent and identically distributed: For any i and l, $Y(i,l)=1$ with probability p and $Y(i,l) \geq 2$ with probability $1-p$.

Consider the tree from the root down. Let β, $0 < \beta < 1$, be a constant and for any number, t, let C^t denote the elements at level t of the tree. We call level t of the tree *productive* if at least $\beta|C^t|$ of its elements have two children. Note that if C^t is productive then $|C^{t-1}| \geq (1+\beta)|C^t|$. There is one element at level k; therefore, if at least $\lceil \log_{(1+\beta)} n \rceil$ levels of the tree are productive then the tree has at least n leaves.

Productive Probability. Recall p, the probability that a node has only one child. Let t be any number. Define q_l to be the probability that level t is not productive given that it has l nodes. Then

$$q_l = \Pr\left[|C^{t-1}| < (1+\beta)\cdot l \,\middle|\, |C^t| = l\right] \leq \binom{l}{\lfloor l-\beta l+1 \rfloor} p^{\lfloor l-\beta l+1 \rfloor} \leq 2^l p^{(1-\beta)l} = (2p^{1-\beta})^l.$$

If $p < (\frac{1}{2})^{\frac{1}{1-\beta}}$, the probability of a level being unproductive shrinks exponentially with the number of elements in the level. For $\beta = \frac{1}{2}$ and $p \leq \frac{1}{8}$, $q_1 = p$, $q_2 = p^2 < p$, $q_3 \leq 3p^2 < p$, $q_4 \leq 4p^3 < p$ and $q_5 \leq 10p^3 < p$. Setting $\beta = \frac{1}{2}$ and restricting $p \leq \frac{1}{8} < (\frac{1}{2})^{\frac{1}{1-\beta}} = \frac{1}{4}$, the probability, q, that a level is not productive is then

$$q \leq \max_{l \geq 1}\{q_l\} \leq \max\left\{p, \max_{l \geq 6}\{(2p^{1-\beta})^l\}\right\} \leq \max\{p, (2p^{1-\beta})^6\} = p.$$

explicitly by the analysis but, unless e is the first event of the process, it is some time between $t(e)$ and the end of the previous event executed by $P(e)$. Let e be any event, let v be the element associated with e, and let w be the parent of v at time $t(e^-)$. Add a directed edge to the graph from the point $\langle v, t(e) \rangle$ to the point $\langle w, t^-(e) \rangle$.

For any element, x, define the *execution tree of C relative to x* as follows. The root of the tree is the point $\langle x, |C| \rangle$, where $|C|$ represents the length of the computation. For any point, $\langle e, r \rangle$, the children of $\langle e, r \rangle$ are those points at level $r-1$ which are on a path from $\langle e, r \rangle$. In other words, for any integer, r, if $r = 0$ then $\langle e, r \rangle$ is a leaf; otherwise, the children of $\langle e, r \rangle$ are $\{ \langle j, r-1 \rangle \mid \langle j, r-1 \rangle \text{ is on a path from } \langle e, r \rangle \}$.

The execution tree has the following structure: Let r be any round number. A point $\langle i, r \rangle$ has only one child if and only if process i did not execute an event at round r; this can happen with probability at most p. Otherwise, $\langle i, r \rangle$ has at least two children. For any i and r, let X_i^r be the number of leaf descendants of $\langle i, r \rangle$ in the computation tree. Then

Lemma 3.6 *For any i and l, $X_i^l \leq D(i, p_l(i))$.*

Proof. Follows by induction from the definition. □

The analysis is reduced to finding a number k and showing that, with high probability, for any element, x, $\langle x, k \rangle$ has at least n leaf descendants in the computation tree relative to x. In fact,

Theorem 3.2 *Given $p \leq \frac{1}{9}$, and for any element, x, if $|C| = t \geq 2\lceil \log_{3/2} n \rceil$ then $Pr[X_x^t < n] \leq (\frac{2}{3})^t$.*

Theorem 3.2 states that, for any given element, x, after $t > 2\log_{3/2} n$ rounds the probability that x does not point to the end of the list is at most $(\frac{2}{3})^t$. The array has n elements, therefore, the probability that after t rounds there is an element, x, for which $p(x)$ is not the head of the list is at most $n(\frac{2}{3})^t$.

Theorem 3.3 *Given $p \leq \frac{1}{9}$ and for any computation of the recursive doubling algorithm, after $t \geq 2\lceil \log_{3/2} n \rceil$ rounds every element in the list points to the end of the list with probability at least $n(\frac{2}{3})^t$.*

This can be generalized for larger values of p using Lemma 3.2 yielding:

Corollary 3.5 *For any $l \geq 1$, if $p \leq (\frac{1}{9})^{1/l}$, after $t \geq 2l\lceil \log_{3/2} n \rceil$ rounds every element in the list points to the end of the list with probability at least $n(\frac{2}{3})^{\frac{t}{l}}$.*

And finally,

3.5 Recursive Doubling

Unlike the summation algorithm, which was performed along an implicit binary tree, in the recursive doubling algorithm each process proceeds independently of the other processes. Without loss of generality we assume that the input comprises a single linked list of elements.

Under the synchronous variation the recursive doubling algorithm behaves very similarly to the summation algorithm. It proceeds in $\lceil \log n \rceil$ levels. A computation associated with level i does not proceed before all the computations of level $i-1$ have terminated. The expected complexity of the algorithm is then $\Theta(\log^2 n/\log(1/p))$. For $p=\frac{1}{4}$, for instance, this is $\Theta(\log^2 n)$.

We turn to the asynchronous case. Consider any probabilistic computation, C, of the recursive doubling algorithm. We show that, with high probability, the algorithm terminates in $O(\log n)$ rounds. Recall that each process is associated with one element of the list. For any element, v, any round, r, and any event, e, we say that the event *e is seen by v at round r* if the process associated with v executed event e at round r.

The algorithm can be viewed as running on an infinite list consisting of the first $n-1$ elements followed by the head of the list repeated indefinitely. Recall that for any pair of elements, u and v, $D(u,v)$ is the *distance from u to v* and is defined to be the number of links between u and v in the input list. Let V be the set comprising the first $n-1$ elements of the list and let $p_k(v)$ be the successor of v after the kth round. The algorithm terminates when the first $n-1$ elements of the list point to elements at distance at least $n-1$. Therefore, we wish to compute the probability

$$\Pr\left[\min_{v\in V}\{D(v,p_k(v))\} < n-1\right].$$

Note that regardless of the events seen by v at round $k+1$, $D(v,p_{k+1}(v)) \geq D(v,p_k(v))$.

Recall that each iteration of the algorithm comprises two operations: a *read operation* in which a node, v, reads the current parent of its parent, and an *update operation* in which the node replaces its current parent by the new value it read.

We analyze the algorithm with the aid of a *computation DAG* defined as follows. The processes (or list elements) are listed along the x axis and the virtual time along the y axis. There is a vertical line going through each element point on the x axis; this represents the element's history; the vertical lines are assumed to be directed towards the x axis (i.e., towards the past).

For any event, e, let $t(e)$ be the virtual time assigned to e and let $P(e)$ be the process which executed e. $t(e)$ represents the virtual time at which superevent e was completed. Let $t^{-}(e)$ be the (virtual) time when e executed the read operation. This time is not given clause of the if statement is executed and the event is called a *waiting* event. Otherwise, an event validates a node and is called the *validating* event.

In order to simplify the analysis we consider an auxiliary graph comprising n paths, one path corresponding to each path in the implicit tree from a leaf to the root. Given a computation on the tree, we define an event to occur at a node v on the path if an event occurs at the corresponding node in the tree (in general, an event will occur at nodes on several paths simultaneously). The event on the path is validating if its only child is valid (initially the leaves are the only valid nodes). An easy induction shows that the computation at a node v in the tree is validated exactly when all the corresponding nodes on the paths are validated. Now, we can overestimate the non-termination probability for the algorithm by multiplying by n the non-termination probability for a single path. (A very similar argument is due to Luby [Lub88].)

So the probability that some path requires at least $c{\cdot}k$ rounds to be completely validated, assuming $c > 1$ and $k \geq \log n$, is given by:

$$\begin{aligned} n\binom{c \cdot k}{ck - \log n} p^{(ck-\log n)} &\leq 2^{ck+\log n} p^{(c-1)k} \leq 2^{(c+1)k} p^{(c-1)k} \\ &= \left(2^{c+1} p^{c-1}\right)^k \\ &\leq 1/2^k \end{aligned}$$

$$\text{assuming that } p \leq 2^{-\frac{c+2}{c-1}}.$$

Taking $c = 4$ we have shown:

Theorem 3.1 *For any $j \geq \log n$, if $p \leq \frac{1}{4}$, the summation algorithm terminates in at most $4j$ rounds with probability at least $1 - \frac{1}{2^j}$.*

This can be generalized for larger values of p using Lemma 3.2 yielding:

Corollary 3.3 *For any $j \geq \log n$, and any $l \geq 1$, if $p \leq (\frac{1}{4})^{1/l}$, the summation algorithm terminates in at most $4jl$ rounds with probability at least $1 - \frac{1}{2^j}$.*

For example, when $p = \frac{1}{2}$, the algorithm terminates in at most $8j \log n$ rounds with probability at least $1 - \frac{1}{n^j}$.

Corollary 3.4 *For any number, $l \geq 1$, if $p \leq (\frac{1}{4})^{1/l}$, the expected number of rounds the summation algorithm executes is at most $4l \log n + o(1)$.*

events.

Lemma 3.4 *For any algorithm, A, any probability, p, and any number, c,*

$$\begin{aligned} E_{p^c}(A) &= \sum_i \frac{\log N_i}{\log(1/p^c)} = \frac{1}{c} \cdot \sum_i \frac{\log N_i}{\log(1/p)} \\ &= \frac{1}{c} \cdot E_p(A). \end{aligned}$$

Let A^r be the algorithm A defined in terms of superevents, each superevent comprising exactly r events. Then

Lemma 3.5 *For any algorithm, A, and any probability, p, $E_p(A) = r \cdot E_p^r(A^r)$.*

Proof. Follows from the additivity of expectations. □

3.4 Summation Algorithm

The summation algorithm iterates the following superevent comprising a condition test plus a possible execution of the if statement.

```
Algorithm for process $i$:
    1   while ($V_i$ is not valid) do
    2       if $R_i$ and $L_i$ are valid then
    3           set $V_i := L_i + R_i$
    4           set the tag of $V_i$ to valid
    5       end if
    6   end while
```

The summation algorithm comprises $\log n$ levels of superevents; henceforth we call superevents events.

Under the synchronous variation, none of the processes at level $i+1$ can proceed before all the processes at level i finish their computation. Furthermore, there are $n/2^i$ processes at level i (the leaves are considered to be at level 0). Let i be given and let P be a process associated with an internal node at level i. Let N_i be the expected number of rounds it takes to compute the ith level of the computation tree. Since all the probabilities are assumed to be independent, if there are m nodes at level i, the expected number of rounds required by the level is $\Theta(\log m/\log(1/p))$. The expected number of rounds for the entire computation is the sum over all i of the expected number of rounds for level i; this is $\Theta(\log^2 n/\log(1/p))$. When $p = \frac{1}{4}$, for instance, this becomes $\Theta(\log^2 n)$.

Let us consider the asynchronous case. The events of the computation are of two types. If at least one of the children of the node associated with the event is not valid, the else

Corollary 3.2 *For any algorithm, A, any probability, $0 \leq p \leq 1$, and any integer, $r \geq 1$, let $q = 1-(1-p)^{2r-1}$. Then,*

1. $E_p(A^r) \leq E_p(A) \leq (2r-1)E_q(A^r)$.
2. *For any* $t \geq 1$, $NT_p(A^r,(2r-1)t) \leq NT_p(A,(2r-1)t) \leq NT_q(A^r,t)$.

Proof. The inequalities on the left are trivial lower bounds. □

3.3 The Synchronous Variation

Recall that the purpose of the probabilistic complexity measure is to assess the reduction in the implicit costs for synchronization of the asynchronous algorithms compared with their synchronous counterparts. To this end, we introduce a modified synchronous model, whose performance is compared to that of the asynchronous model; we call this model the *synchronous variation.* In the synchronous variation, in addition to being synchronous, a computation is subject to the same delay pattern as an asynchronous computation. However, in the modified model no cost is charged for explicit synchronization, and thus, the difference in complexity between an algorithm in the synchronous variation and in the corresponding asynchronous variation provides some indication of the reduction in the implicit costs of synchronization yielded by the asynchronous model.

More formally, consider an algorithm, A, and consider any computation, C_A, of A. A synchronous computation of A under the unbounded delays measure with parameter p is obtained as follows. A pseudo-computation is created with exactly one pseudo-event per process for each round. Then the adversary attaches a probability $p(e)$ to each pseudo-event e. As before, $p(e)$ is the probability that e is eliminated; the requirement is that $p(e) \leq p$. Again, the surviving events are the events of the computation. Now, an event may be a busy-wait, required to await the (implicit) synchronization which ends each step of the synchronous computation.

Consider an algorithm, A, with N processes. In each time step of the synchronous computation, each process must execute exactly one step. Therefore, the expected number of rounds comprising a single step of the computation is the maximum of N independent random variables. For the unbounded delays measure with parameter p, the expected duration of each step is $\log_{1/p} N = \log N/\log(1/p)$. For any time step, t, let N_t be the number of processes at that time step. Then, the expected number of rounds required for time step, t, is $\log N_t/\log(1/p)$.

Now, we give two simulations for the synchronous variation that allow us to ignore constants in the probabilities and to analyze algorithms in terms of superevents rather than

p. Then

$$Pr_{\mathcal{A}'}(id, r) = \prod_{i=0}^{c-1} Pr_{\mathcal{A}}(id, rc + i) \leq \prod_{i=0}^{c-1} p = p^c = q.$$

Therefore, $\mathcal{A}'$ is a conforming adversary for algorithm A under the unbounded delays measure with parameter q. For adversary $\mathcal{A}'$, algorithm A has expected rounds complexity $\lceil E_p/c \rceil$; the first result follows. Furthermore, for any number, t, if the probability that A does not terminate in t rounds under adversary $\mathcal{A}$ is p_1, then the probability that A does not terminate in $\lfloor t/c \rfloor$ rounds under adversary $\mathcal{A}'$ is at least p_1; the second result follows. $\square$

Corollary 3.1 *For any algorithm, A, and any integer, $l \geq 1$,*

1. $E_{p^l}(A) \leq E_p(A) \leq l \cdot E_{p^l}(A)$.
2. *For any* $t \geq 1$, $NT_{p^l}(A, l \cdot t) \leq NT_p(A, l \cdot t) \leq NT_{p^l}(A, t)$.

Proof. The inequalities on the left follow from Lemma 3.1, the ones on the right from Lemma 3.2. $\square$

Recall that A^r denotes algorithm A specified in terms of superevents, where each superevent comprises at most r events.

Lemma 3.3 (Scalability) *For any algorithm, A, for any number, r, and any two probabilities, p and q, if $q = 1 - (1-p)^{(2r-1)}$ then*

1. $E_p(A) \leq (2r-1)E_q(A^r)$.
2. *For any number* $t \geq 1$, $NT_p(A, (2r-1)t) \leq NT_q(A^r, t)$.

Proof. Let $\mathcal{A}$ be any conforming adversary for A under the unbounded delays measure with parameter p. Define an adversary, $\mathcal{A}'$ as follows: $\mathcal{A}'$ defines the same interleaving as $\mathcal{A}$; $\mathcal{A}'$ associates the same probability as $\mathcal{A}$ to each pseudo-event; but where $\mathcal{A}$ associates time $t(e)$ to pseudo-event e, $\mathcal{A}'$ associates time $\lfloor t(e)/(2r-1) \rfloor$.

For any number, s, consider the segment of $2r-1$ rounds of $\mathcal{A}$ starting from round $s(2r-1)$ and ending with round $(s+1)(2r-1)-1$. The probability that this segment contains at least one event from each of the $2r-1$ rounds is at least $(1-p)^{(2r-1)}$. Therefore, the probability that round s of $\mathcal{A}'$ contains at least $2r-1$ events is at least $(1-p)^{(2r-1)} = 1-q$. Thus, $\mathcal{A}'$ is a conforming adversary of A^r under the unbounded delays measure with parameter q.

For adversary $\mathcal{A}'$, algorithm A has expected rounds complexity $\lceil E_p/(2r-1) \rceil$; the first result follows. Furthermore, for any number, t, if the probability that A does not terminate in t rounds under adversary $\mathcal{A}$ is p_1, then the probability that A^r does not terminate in $\lfloor t/(2r-1) \rfloor$ rounds under adversary $\mathcal{A}'$ is at least p_1; the second result follows. $\square$

3.2 Robustness Criteria

In order to obtain a robust measure we require it to observe three properties. The first is *monotonicity*: intuitively, we require that when the speed of the processes is increased the complexity does not increase. Translated to our measure, we require that when the probability that a round does not contain an event of a given process is decreased, the complexity does not increase.

The second property is *scalability*. Scalability refers to the property that the complexity of the algorithm analyzed in terms of superevents is within a constant factor of the complexity of the algorithm analyzed in terms of events.

The third property is *convertibility*. It defines the relationship between results obtained under the unbounded delays measure with different parameters.

In the next three lemmas we show that the unbounded delays measure observes all three properties.

Lemma 3.1 (Monotonicity) *For any algorithm, A, and any two probabilities, p and q, if $p \leq q$ then*

1. $E_p(A) \leq E_q(A)$.
2. *For any number, t, $NT_p(A,t) \leq NT_q(A,t)$.*

Proof. Any conforming adversary for algorithm A under the unbounded delays measure with parameter p is also a conforming adversary for algorithm A under the unbounded delays measure with parameter q. □

Lemma 3.2 (Convertibility) *For any algorithm, A, for any two probabilities, p and q, and for any number, c, if $q = p^c$ then*

1. $E_p \leq c \cdot E_q(A)$.
2. *For any number, t, $NT_p(A,tc) \leq NT_q(A,t)$.*

Proof. Let $\mathcal{A}$ be any conforming adversary for A under the unbounded delays measure with parameter p. Define an adversary, $\mathcal{A}'$ as follows: $\mathcal{A}'$ defines the same interleaving as $\mathcal{A}$; $\mathcal{A}'$ associates the same probability as $\mathcal{A}$ to each pseudo-event; but where $\mathcal{A}$ associates time $t(e)$ to pseudo-event e, $\mathcal{A}'$ associates time $\lfloor t(e)/c \rfloor$.

For any process identifier, id, and for any round, r, let $Pr_{\mathcal{A}}(id,r)$ denote the probability that process id will not have an event in round r given adversary $\mathcal{A}$. Note that $Pr_{\mathcal{A}}(id,r) \leq$

the coin shows dummy, e_i is eliminated. The remaining pseudo-events are the events; they form the computation. Each successive event of a process indicates when the next step of its program is to be performed.

For each integer, r, *round* r of the computation is the subsequence containing all the events with time field $\lfloor t \rfloor = r$.

Let C be any computation obtained as above. Define $Pr_{\mathcal{A}}(C)$, the *probability of obtaining C given the adversary $\mathcal{A}$*, to be the combined probability of all the choices which led to the computation. The adversary is called *conforming* if for each round, r, and each process, id, the probability that round r does not have a tuple with identifier id is at most p. We consider only conforming adversaries.

Notation. For any algorithm, A, let $NT_p(A, k)$ (resp. $E_p(A)$) be the nontermination probability (resp. the expected complexity) of A under the unbounded delays measure with parameter p.

3.1 Superevents

It is often more convenient to describe algorithms in terms of higher level constructs called *superevents.* For any algorithm, A, and any number, r, let A^r denote algorithm A specified in terms of superevents, where each superevent comprises at most r events. A superevent, e, is said to have executed within superround t, for some superround number t, if all the events comprising the superevent are within the superround.

As before, we obtain a computation of A^r with the aid of an adversary. Although the algorithm is specified in terms of superevents, a computation is defined as a sequence of events. The usefulness of the definitions will become clear when we analyze the two algorithms in later sections. The adversary for A and for A^r differ only in the definition of conformity. An adversary for A is a conforming adversary if the probability that a round does not contain at least one event is at most p. An adversary for A^r is said to be conforming if the probability that a round (or in this case a superround) does not contain a superevent is at most p.

As each superevent comprises at most r events, a round which contains at least $2r - 1$ events of a given process contains at least one superevent of that process. An adversary of A^r is *conforming* if for every round, t, and every process, id, the probability that round t contains at least $2r - 1$ events is at least $1 - p$.

may create events with duration greater than 1.

Now, we define two yardsticks: For any number, k, and any adversary, $\mathcal{A}$, the *nontermination probability of algorithm A relative to adversary $\mathcal{A}$*, $NT_{\mathcal{A}}(A,k)$ is the probability that a computation of A has length at least k. The expected length of a computation relative to adversary $\mathcal{A}$, $E_{\mathcal{A}}(A)$, is then

$$E_{\mathcal{A}}(A) = \sum_{j\geq 0} NT_{\mathcal{A}}(A,j).$$

Given any algorithm, A, and any number, k, the nontermination probability of A, $NT(A,k)$ is the maximum over all possible adversaries, $\mathcal{A}$, of the nontermination probability of A relative to $\mathcal{A}$. In other words:

$$NT(A,k) = \max_{\mathcal{A}} NT_{\mathcal{A}}(A,k).$$

The expected complexity of A is the maximum over all adversaries, $\mathcal{A}$, of the expected complexity of A relative to $\mathcal{A}$. In other words,

$$E(A) = \max_{\mathcal{A}} E_{\mathcal{A}}(A).$$

3 The Unbounded Delays Measure

The unbounded delays measure has a single parameter, p. Given a round, r, and a process, id, the parameter p roughly corresponds to the probability that process id does not execute an event in round r.

We have the following scenario in mind. Typically, the number of processes used by the program substantially exceeds the number of available processors, requiring each processor to execute many processes concurrently. The load of the different processors may not be balanced; even if the processes are initially distributed equally among all the processors, in a dynamic computation it may be difficult to maintain a balance among the processors. Furthermore, some processes may be suspended, waiting on an external event such as an I/O operation or an interrupt.

As described earlier, the computations are obtained with the aid of an adversary. The adversary determines an interleaving of objects called pseudo-events. A subsequence of the pseudo-events forms the computation, a sequence of events. The adversary associates a distinct time with each pseudo-event. For each process, its sequence of pseudo-events have strictly increasing times. Pseudo events are eliminated as follows. The adversary associates a probability p_i with pseudo-event e_i. To determine if e_i is eliminated, the following coin is tossed: it has probability p_i of showing *dummy* and probability $1-p_i$ of showing *real*. If *virtual clock*. This approach was introduced in [PF77] and used in [AFL83,LF81,KRS88b] and is common in the area of distributed computing (see [Awe87], [Awe85,AG87]). Consider a computation, C. A *virtual clock of* C is an assignment of unique virtual times to the events of C; the times assigned are a non-decreasing function of the event number.

The virtual clock is meant to correspond to the "real" time at which the operations occurred in one possible execution of the algorithm, called a computation. The time difference between two consecutive events of a process is called the *duration* of the later event. The length of a computation is the time assigned to the last event in the computation.

The rounds complexity of an algorithm is the length of a computation maximized over all possible computations; i.e., maximized over all possible interleavings of the events of the processes. In the *rounds complexity measure* of [CZ91a], we assumed that the duration of events was at most one. In effect, in the rounds complexity measure the slowest process defined a unit of time (a round); the complexity is expressed in rounds. This normalizes the complexity relative to the speed of the slowest process. Therefore, it is inadequate for measuring the implicit costs of synchronization; to capture these, we allow the duration to assume values larger than one.

There appear to be several ways to introduce large durations. We model the duration as a random variable as follows. We define the variable speed APRAM. Intuitively, a computation of the variabe speed APRAM is divided into rounds. But, unlike the rounds complexity we do not require each process to execute at least one event in each round. Rather, for each process and for each round the process has a certain probability of executing an event in that round. We then consider all possible computations and associate with each computation a probability.

More foramlly, the computations of this model are specified with the help of an adversary $\mathcal{A}$. First, a *pseudo-computation* $\tilde{C}$ is created. It is an interleaving of *pseudo-events*, the interleaving being specified by $\mathcal{A}$: $\mathcal{A}$ labels each pseudo-event with a distinct time, a real number. In addition, $\mathcal{A}$ is allowed to eliminate pseudo-events by means of a probabilistic game; the exact rules of the game depend on the complexity measure being considered. The pseudo-events remaining after this elimination form the events of the computation. Note that one adversary may create many computations; each computation C has an associated probability $Pr_{\mathcal{A}}(C)$, namely the probability that it is created by the elimination game. The requirement on pseudo-events is that their duration be at most one (the duration of a pseudo-event e, associate with process *id*, is the time from the pseudo-event of *id* preceding e, if any, to the time of e; if there is no preceding pseudo-event, the duration is just the time associated with e). The duration of events is defined analogously. The elimination game it appears necessary to design them in an asynchronous environment; this is the focus of our work.

Now, we briefly discuss some of the work concerned with uniform environments. [PU87], [PY88,AC88] show that the communication among the processes can be reduced by designing algorithms which take advantage of temporal locality of access; they assumed that each (global) memory access has a fixed cost associated with it. In a related paper [ACS89], Aggarwal et al. argue that typically it takes a substantial time to get the first word from global memory, but after that, subsequent words can be obtained quite rapidly. They introduce the BPRAM model which allows block transfers from shared memory to local memory. They show that the complexity of algorithms can be reduced significantly by taking advantage of spatial locality of reference. This approach implicitly assumes that the (virtual) machine is uniform, comprising a collection of similar processes. In addition to latency to memory, Gibbons [Gib89] studied the cost of process synchronization by considering an asynchronous model. Although he assumed that the machine is asynchronous, his analysis in effect assumes that the processes are roughly similar in speed. More precisely, he required that read and writes be separated by global synchronization; this results in a uniform environment from the process perspective.

2 The APRAM model

The RAM is a very popular model in sequential computation and algorithm design. Therefore, it is not surprising that a generalization of the RAM, the PRAM, became a popular model in the field of parallel computation. We use another generalization of the RAM model, called the APRAM, suggested by Kruskal et al. [KRS88a]; a formal definition of the APRAM model is given in [CZ91a]. It includes the standard PRAM as a submodel. The APRAM can be viewed as a collection of processes which share memory. Each process executes a sequence of basic atomic operations called *events*. An event can either perform a computation locally (by changing the state of the process) or access the shared memory; accessing the memory has a unit cost associated with it. An APRAM computation is a serialization of the events executed by all the processes.

The parallel runtime complexity associated with the PRAM model describes the complexity of PRAM algorithms as a function of the global clock provided by that model. The APRAM model does not have such a clock. Consequently, it is necessary to define complexity measures to replace the running time complexity. In [CZ91a] we defined one such complexity called the rounds complexity: it replaces the global clock used by the PRAM model by a the moment it suffices to remark that they facilitate algorithm analysis. We show that the unbounded delays measure satisfies all three properties. This is not true of the bounded delays measure, which only satisfies monotonicity; even so, we are able to analyze some intersting algorithms with the latter measure.

We analyze two algorithms under the new model: a parallel summation algorithm along an implicit tree and a recursive doubling algorithm. Both are fundamental algorithms and appear as subroutines in many parallel algorithms. We obtain two main sets of results. In Section 3, we show that under the unbounded delays measure, in asynchronous environments, both the APRAM summation algorithm and the APRAM recursive doubling algorithm perform significantly better than their synchronous counterparts, in a sense made precise later. Next, in Section 4, we obtain similar results in the bounded delays measure. These analyses provide an unexpected separation result: in the bounded delays measure, the recursive doubling algorithm performs better than the tree based summation algorithm in some asynchronous environments. In Section 2, the APRAM model and the associated complexity measures are described.

At this point it may be helpful to comment on our contribution. The algorithms we analyze are simple and known (or only minor modifications of known algorithms). The analyses are fairly straightforward probabilistic arguments (the main non-obvious element arising in the analysis of the recursive doubling algorithm, where our analysis is time-reversed). The main contribution of the paper lies in the results themselves, which demonstrate by example the advantage to be gained from avoiding the implicit costs of synchronization, at least in certain asynchronous environments.

1.1 Related Work

The work of Nishimura [Nis90] is motivated by similar considerations to our work. She analyzes the performance of a pointer jumping algorithm in her model and proves several simulation results. More recently, Anderson and Woll described a wait-free algorithm for the Union-Find problem [AW91]; the advantage of a wait-free algorithm is that no slow or failing process can prevent progress indefinitely.

Another approach to the problem of asynchrony is to seek to efficiently compile PRAM algorithms to operate in asynchronous environments; this approach is followed by Martel, Park and Subramonian [MPS89,MSP90]. They give efficient randomized simulations of arbitrary PRAM algorithms on an asynchronous model, given certain architectural assumptions (e.g. the availability of a compare&swap instruction). It is not clear whether similar deterministic compilers exist; in the absence of such compilers, to obtain deterministic algorithms from the user. Our work has proceeded in two directions. In [CZ89,CZ91a,CZ91b] we studied the cost of achieving synchronization, which we called the *explicit costs of synchronization.* This includes the cost of executing extra code that must be added to the algorithm in order to synchronize. A PRAM algorithm, for example, assumes that all processes synchronize before every step. By considering the explicit costs of synchronization we can design algorithms which require less synchronization. It is anticipated that such algorithms will perform better in asynchronous settings.

In this paper, we investigate the effects of nonuniformity of the environments on the complexity of algorithms. For asynchronous environments may well produce such nonuniformity: different processes may proceed at different speeds and even the same process may proceed at different speeds at different times. Even if the underlying hardware is synchronous and uniform, the processes may appear to be proceeding at different and varying speeds. This may be due, for instance, to uneven loads among the physical processors or to processes executing instructions that have different execution times.

To determine the possible gains note that the most rigid execution of an algorithm is a lock step execution in which a process is not allowed to execute the ith step before all processes finish step $i-1$. We compare asynchronous executions of an algorithm to the lock step execution, ignoring the explicit costs of the synchronization needed to achieve lock step execution. We call the added complexity resulting from lock step execution the *implicit costs of synchronization.* We do not define the notion of explicit and implicit costs formally, for we do not think we have enough experience to justify definitive definitions. In general, we do not expect an algorithm to perform better when process speeds are non-uniform. What we seek are algorithms whose performance degrades only modestly; we say these algorithms are *resilient* to changes in the environment. We anticipate resilient algorithms will perform significantly better that their synchronous counterparts in asynchronous environments.

[CZ91a] defines the APRAM model formally; its use was first suggested in [KRS88a]. The *rounds complexity* measure was used to measure the explicit costs of synchronization. In this paper we introduce the *variable speed APRAM*, an APRAM model in which the speed of processes may vary. We define two complexity measures called the *unbounded delays* measure and the *bounded delays* measure. They model the non-uniformity in the environment by defining the duration of an operation using a random variable which may take values larger than 1. The implicit costs of synchronization are inferred by observing the effect of varying the durations on the complexity of the algorithm. In addition, we identify several characteristics which are desirable in order to have robust measures. These are called *monotonicity, scalability* and *convertibility.* We discuss these properties later. For

The Expected Advantage of Asynchrony[*]

Richard Cole
New York University

Ofer Zajicek[†]
Transarc Corporation

Abstract

This paper studies the implicit costs of synchronization and the advantage that may be gained by avoiding synchronization in asynchronous environments. An asynchronous generalization of the PRAM model called the APRAM model is used and appropriate complexity measures are defined. The advantage asynchrony provides is illustrated by analyzing two algorithms: a parallel summation algorithm which proceeds along an implicit complete binary tree and a recursive doubling algorithm which proceeds along a linked list.

1 Introduction

This paper continues our study of the effect of process asynchrony on parallel algorithm design. As is well known, the main effort in parallel algorithm design has employed the PRAM model. This model hides many of the implementation issues, allowing the algorithm designer to focus first and foremost on the structure of the computational problem at hand — synchronization is one of these hidden issues.

This paper is part of a broader research effort which has sought to take into account some of the implementation issues hidden by the PRAM model. Broadly speaking, two major approaches have been followed. One body of research is concerned with asynchrony and the resulting non-uniform environment in which processes operate[1] [CZ89,Nis90,MPS89], [MSP90,AW91]. The other body of research has considered the effect of issues such as latency to memory, but assumes a uniform environment for the processes [PU87,PY88,AC88], [ACS89,Gib89].

The PRAM is a synchronous model and thus it strips away problems of synchronization. However, the implicit synchronization provided by the model hides the synchronization costs

[*]The work was supported in part by NSF grants CCR-8902221 and CCR-8906949, and by a John Simon Guggenheim Memorial Foundation Fellowship.

[†]The work of this author was performed mainly while he was at New York University.

[1]We distinguish between processes and processors in order to emphasize that the APRAM is not a machine model but rather a programming model; it is the task of a compiler to implement the programming model on actual machines. The term processor will be used to refer to this component of a machine.