In *Proc. DARPA Sortware Technology Conference 1992*, pp. 184-188.

Ultrafast Parallel Algorithms and Reconfigurable Meshes

Quentin F. Stout

EECS Department
University of Michigan
Ann Arbor, MI 48109-2122 USA
qstout@eecs.umich.edu

1 Introduction

This research is concerned with the development of very fast parallel algorithms, ones faster than those available through normal programming techniques or standard parallel computers. Algorithms have been developed for problems in geometry, graph theory, arithmetic, sorting, and image processing. The computing models that these algorithms have been developed for are concurrent read concurrent write parallel random access machines (*CRCW PRAMs*), and reconfigurable meshes (*rmeshes*, defined below).

For CRCW PRAMS, our work has shown that by combining randomization with the use of some extra memory, one can solve some problems far faster than they can be solved if only randomization is used. We have developed ultrafast algorithms for several problems, where by *ultrafast algorithm* we mean a parallel algorithm with an input of size n which uses at most a linear number of processors and finishes in poly-loglog time, i.e., in time $O((\log\log n)^k)$ time for some $k > 0$. We have also proven the optimality of some ultrafast algorithms.

For rmeshes, our work has shown that they can deterministically solve many problems faster than regular mesh-connected computers, and in some cases can solve problems even faster than a CRCW PRAM. For example, it is possible to sort in constant time [6]. While various notions of reconfigurability have been around for some time, rmeshes are a class of machines which have only recently been studied and built, and which show great promise for a variety of tasks [7, 8].

This work is primarily being carried out by the author, who is the Principal Investigator, and Philip MacKenzie, a doctoral student finishing his thesis in this area [9]. Recently two beginning graduate students, Douglas Van Wieren and Eric Hao, have started work on this project.

2 Randomization and Extra Space

Randomization is an important technique for parallel and distributed computing. For example, the randomized backoff strategy in ethernet systems allows fast, simple, decentralized deadlock avoidance. Similarly, simple randomized mapping strategies can be used to achieve approximate

load balance in a dynamic parallel or distributed system, using only local information instead of resorting to collecting more costly global balance information.

In the field of parallel algorithms, especially those for PRAMs, randomization has been used to provide faster expected time solutions to many problems. For example, to find the minimum of n values using n processors in a CRCW PRAM takes $\Theta(\log\log n)$ worst-case time [14], but it can be completed in constant expected time [11]. However, for some problems randomization by itself does not achieve the maximal possible speedup. For example, for n processors to sort n numbers into an array of size n takes $\Omega(\log n/\log\log n)$ expected time, even if the processors use randomization and it is known that the numbers were chosen uniformly and independently from the unit interval. This result comes from a very general lower bound on the time required to determine parity [1], and holds even if a polynomial number of processors are utilized.

2.1 Ultrafast Algorithms

Our work has shown that lower bounds, such as that for parity, can sometimes be circumvented by combining randomization with the use of extra space. To illustrate, suppose we allow the sort to yield an array of $2n$ positions, where half of the enties will be blank and the nonblank entries are in sorted order. We call this a *padded sort*, and have shown that it can be completed in $\Theta(\log\log n/\log\log\log n)$ time [11]. This time can be achieved using only $n\log\log\log n/\log\log n$ processors, i.e., it can be achieved with linear speedup over the optimal linear expected time serial algorithm.

Padded sort is not quite as useful as a standard sort, since one does not know exactly where the blanks will occur. Thus it cannot be used to solve parity, which is why it can avoid the lower bound for parity. Nonetheless, for several problems it provides just enough information to help solve the problem quickly. For example, given n planar points chosen uniformly and independently from the unit square, in $\Theta(\log\log n/\log\log\log n)$ expected time one can find the nearest neighbor of each point, construct the Voronoi diagram of the set, and determine the relative neighborhood graph [11]. In earlier work, similar use of randomization had shown that the extreme points of their convex hull, and the maximal points, can all be determined in constant expected time [13].

2.2 Lower Bounds

In conjunction with designing ultrafast algorithms, we are also interested in understanding why they cannot be still faster. Unfortunately the same power which permits ultrafast algorithms makes it difficult to prove nontrivial lower bounds. One area where we have succeeded is in approximate load balancing on CRCW PRAMs. In this problem, suppose there is a constant $C \geq 2$, and n processors with tasks, where the total number of tasks is at most n. The goal is to redistribute the tasks so that no processor has more than C. This approximate load balancing is a critical step in many randomized ultrafast algorithms, and it had been shown that it can be accomplished in $\Theta(\log^* n)$ expected time [4]. ($\log^* n$ is a function which grows extremely slowly, being 5 or less for all numbers less than $2^{65,536}$). Note that achieving exact load balance takes $\Omega(\log n/\log\log n)$ expected time, but the extra measure of balance is not needed in many situations. This is especially true in ultrafast algorithms, where the rest of the algorithm can be completed faster than exact load balance.

P. MacKenzie [10] has now shown that any algorithm for approximate load balancing must take $\Omega(\log^* n)$ expected time, no matter how much memory is available. Thus the previous algorithm is optimal. This is the first nonconstant lower bound for such problems on the CRCW PRAM, and it applies even if the concurrent write operation is the strong "priority write", even though the algorithm only needs weaker versions of concurrent write.

2.3 Future Ultrafast Work

In yet unwritten work, MacKenzie has extended his lower bound proof to show that any algorithm for finding a hamiltonian cycle in a random graph must take at least $\Omega(\log^* n)$ expected time, and he has also found an algorithm achieving this time. (An earlier ultrafast algorithm for this problem, taking $\Theta((\log \log n)^2)$ expected time, appeared in [3]). We believe that there should be several other problems which can be similarly solved. We also hope that this lower bound proof technique can be extended to other problems which are known to be solvable in $\Theta(\log^* n)$ time, including hashing, generation of random permutations, integer chain sorting, and some PRAM simulations [4].

Unfortunately we still do not know if our padded sort algorithm is optimal, and we are trying to find a lower bound for it. Similarly we are looking at other problems with ultrafast solutions, such as pattern matching [15] or the geometry problems mentioned above, trying to establish lower bounds for them, or at least establish that they can be solved no faster than some more fundamental problem such as padded sort.

3 Reconfigurable Meshes

To construct an $n \times n$ rmesh, imagine a grid of n horizontal and n vertical wires. At each intersection there is a processor attached, controlling switches on each of the incoming wires. The switch settings determine connected regions of the rmesh, called *buses*, over which information can be broadcast. We assume that only one value at a time can be broadcast on any bus, and in constant time it reaches all processors attached to the bus.

Implementations of rmeshes include the Hughes/University of Massachusetts Image Understanding Architecture and the IBM Polymorphic Torus [8]. These are fine-grained SIMD machines with many processors per chip. The Hughes/U. Mass. rmesh is used for the lower and intermediate levels of image understanding, and the Polymorphic Torus is used for VLSI design.

3.1 Rmesh Algorithms

Rmeshes achieve dramatic results by tightly coupling together the actions of calculation, changing switch setings, and communcation. For example, to determine the maximum of n numbers stored one per diagonal entry, first each processor sets its switches so that the buses form rows. Then each diagonal processor broadcasts its value. Then the switches are set to form column buses, and another broadcast occurs. If a processor hears a second value which is smaller than the first value, then the second value is not the maximum, and so any such processor broadcasts "No" on its column bus. Only one column bus will not have such a broadcast (assuing values are distinct. Simple changes handle the case of duplicate values). Then the switches are set so that all processors

are in a single bus, and the one diagonal processor which didn't receive a "No" broadcasts the answer to the entire mesh.

More interesting algorithms come from applications such as image processing, where images are stored one pixel per processor. Each processor can compare its pixel to those of its neighbors to decide if they are parts of the same object, and then disconnect from neighbors that differ. Now the communication buses of the rmesh match the objects of the image, and operations such as object labeling, determining the size of objects, etc., can be carried on concurrently on each object, typically finishing in logarithmic or polylogarithmic time [12].

Similarly, for graph algorithms, if the graph is stored as an adjacency matrix then each row or column is a 1-dimensional rmesh devoted to a single vertex. For arithmetic algorithms, where numbers are stored one bit per processor, the rightmost column may be devoted to all lowest-order bits. Or, in one of the significant early rmesh algorithms, there may be one bit per column, and the sum of these (and all prefix partial sums) can be determined in constant time [12]. (Note that the parity lower bound implies that this problem must take $\Omega(\log n / \log\log n)$ expected time on a PRAM). For VLSI routing, regions may be mapped onto blocks of processors, and as wires are being created in the VLSI model they may correspond directly to buses.

3.2 Current and Future Rmesh Work

Recent work includes algorithms for various graph and image problems, algorithms for arithmetic operations, algorithms to sort n numbers in constant time on an $n \times n$ rmesh (obtained nearly simultaneously by at least 4 groups), and an algorithm for determining the median of n^2 numbers in $\Theta(\log n)$ time [5]. Work is being started on extending the model to reconfigurable hypercubes, to studying how well an rmesh of one dimension or shape can simulate one of a different dimension or shape, and in combining PRAM and rmesh concepts into a single machine.

There is also some research being conducted on determining lower bounds. In some cases optimality is easy to prove through cutset arguments, which are prevelant in proofs of lower bounds for VLSI and distributed memory computing. These come from considering how much information must flow through a set of wires, and are applicable to rmeshes when there is a lot of data which must be moved a considerable distance. Unfortunately they are often too weak when less data must be moved, which is precisely the goal of many rmesh algorithms. One systematic area where they fail is when there are many more processors than data.

One problem for which we have been able to achieve a new lower bound proof is that of finding the maximum of n^2 numbers in an $n \times n$ rmesh. Earlier work [12] had shown that this could be determined in $\Theta(\log\log n)$ time by a recursive divide-and-conquer approach. Recently we have shown that a PRAM lower bound argument from [2] can be carried over to the rmesh to show that this algorithm is optimal. This proof is based on very generic arguments, without requiring that the algorithm be comparison-based, and applies to rmeshes of all dimensions.

References

[1] P. Beame and J. Hastad, "Optimal bounds for decision problems on the CRCW PRAM", *J. ACM* 36 (1989), pp. 643-670.

[2] F.E. Fich, F.M. auf der Heide, P. Ragde, and A. Wigderson, "One, two, three ...infinity: Lower bounds for parallel computation", *Proc. 17 Symp. Theory Comput.* (1985), pp. 48-58.

[3] A.M. Frieze, "Parallel algorithms for finding hamilton cycles in random graphs", *Info. Proc. Let.* 25 (1987), pp. 111-117.

[4] J. Gil, Y. Matias, and U. Vishkin, "Towards a theory of nearly constant time parallel algorithms", *Proc. 32 Symp. on Theory of Comput.* (1990), pp. 244-253.

[5] E. Hao, P. MacKenzie, and Q.F. Stout, "Selection on the reconfigurable mesh", submitted.

[6] J-w. Jang and V.K. Prassana, "An optimal sorting algorithm on reconfigurable mesh", IRIS Tech. Rep. 277, Univ. Southern Calif., 1991.

[7] H. Li and Q.F. Stout, "Reconfigurable SIMD parallel processors", *Proc. of the IEEE* 79 (1991), pp. 429-443.

[8] H. Li and Q.F. Stout, editors, *Reconfigurable Massively Parallel Computers*, Prentice-Hall, 1991.

[9] P. MacKenzie, *Parallel Algorithms with Ultra-Fast Expected Times*, Ph.D. thesis, University of Michigan, 1992 (in preparation).

[10] P. MacKenzie, "Load balancing requires $\Omega(\log^* n)$ expected time", *Proc. 3 Symp. on Disc. Alg.* (1992), pp. 94-99.

[11] P. MacKenzie and Q.F. Stout, "Ultra-fast expected time parallel algorithms", *Proc. 2 Symp. on Disc. Alg.* (1991), pp. 414-424.

[12] R. Miller, V.K. Prasanna-Kumar, D. Reisis, and Q.F. Stout, "Parallel computations on reconfigurable meshes", *IEEE Trans. Comp.*, 1992 (to appear).

[13] Q.F. Stout, "Constant-time algorithms on PRAMs", *Proc. 1988 Int'l. Conf. Parallel Proc.*, pp. 104-107.

[14] L. Valiant, "Parallelism in comparison problems", *SIAM J. Comp.* 4 (1975), pp. 348-355.

[15] U. Vishkin, "Deterministic sampling - a new technique for fast pattern matching", *Proc. 22 Symp. Found. Comp. Sci.* (1990), pp. 170-180.