Parallel Algorithms Column 1: Models of Computation

Michael T. Goodrich*
Department of Computer Science
Johns Hopkins University
Baltimore, MD 21218
goodrich@cs.jhu.edu

1 Introduction

If you are the type that always orders vanilla in the ice cream shop, then you should probably stay away from parallel algorithm design, for you may never get past the fact that any given parallel algorithm may be designed for any one of several different computational models. If, on the other hand, you are the type who thinks there are still not enough channels on cable TV, then you might think the number of parallel models is woefully small. If you fall into neither camp, which, according to the Chernoff bounds [22], you probably should, then you may find this first *SIGACT News* column on parallel algorithm design of some use. For here I review a "classic" model for parallel computation and I survey some interesting work on alternative models for parallel computation.

2 A Classic Model for Parallel Computation: The PRAM

The shared-memory model, or parallel random access machine (PRAM), is classic in parallel algorithm design. In this model processors share a common memory space, which they access in a synchronous (lock-step) fashion. This model is further subdivided by assumptions placed on how concurrent accesses to the same memory locations are to be handled, with the most popular conventions being exclusive-read/exclusive-write (EREW), concurrent-read/exclusive-write (CREW), and concurrent-read/concurrent-write (CRCW). The CRCW PRAM is further subdivided by the mechanism used to handle concurrent writes, with the most popular methods being *common*, where all processors writing to a location must all be writing the same value, and *arbitrary*, where memory contention is handled arbitrarily (which, of course, is different than resolving the conflict randomly). The primary advantage of the PRAM model is that it is simple to design algorithms for it; it is the natural extension of the well-accepted RAM model. Typically, the first parallel algorithms solving a particular problem are designed for the PRAM model, and it is often used to demonstrate the "parallelizability" of such a problem. Several good general publications addressing techniques for designing algorithms for this model have been written, including the book edited by Reif [23] and those written by JáJá [24], Gibbons and Rytter [18], Akl [6], and Akl and Lyons [7], as well as the book chapter by Karp and Ramachandran [27]. The primary disadvantage of the PRAM model is that it doesn't realistically capture all the intricacies of parallel computation as we understand it today.

*The author's research is supported by the National Science Foundation under Grants IRI-9116843 and CCR-9300079.

3 More Realistic Parallel Computation Models

For the remainder of this column I would therefore like to provide an overview properties of real parallel architectures that are not explicitly modeled by the PRAM, giving references to work that more closely models each such property. This is not intended to be a comprehensive survey, however, so please do not be offended if I have omitted some important recent work in this area. Nevertheless, I trust that I have provided enough references so that the dedicated reader can "come up to speed" after a few pointer hops in the literature.

3.1 Network Topology

One of the well-known intricacies of parallel computation ignored by the PRAM model is the method for interconnecting the processors and their common memory space. In the *network model* this interconnection between processors is described explicitly. In it each processor is allocated some amount of local memory (as in the RAM model) and then the processors are interconnected in a specified communication network. The network model is then subdivided by assumptions about the topology of this network. For example, this network might be a binary tree, a mesh, a hypercube, or an expander graph. Published network algorithms tend to be more complicated than their PRAM counterparts, since data movement through the network must be described in detail, but the algorithm is usually more efficient than what can be derived by a straightforward simulation of the PRAM algorithm. The published literature on such simulation results, as well as a host of specific network algorithms, is quite rich, and several such methods can be found in the recent comprehensive book by Leighton [30], as well as the somewhat older book by Ullman [42].

3.2 Asynchrony

Another fundamental issue that the PRAM models (and standard network models) do not specifically address is how to implement the synchronous parallel clocks driving the different processors. Existing parallel machines may be implemented synchronously by having all the processors simultaneously perform the same program, whose instructions are broadcast to them by a synchronized communication network, or the processors may be allowed to perform different computations in an asynchronous fashion. Simulating a PRAM algorithm in this latter framework is therefore complicated by this lack of synchrony.

Fortunately, this issue has not been lost on the parallel algorithms community, as several researchers have described asynchronous versions of the PRAM model, and have given several methods for solving specific problems, such as list ranking and parallel prefix sums, and for simulating synchronous PRAM algorithms that possess "limited synchrony," on asynchronous PRAMs [8, 10, 9, 11, 12, 19, 29, 32, 33, 36, 41]. Typically, the asynchronous PRAM is defined so that time is partitioned into "slots" in which processors may either "sit out" or may perform some type of atomic read and write operation. Computations are usually analyzed in terms of an expected work bound, which is either taken over all possible interleavings of atomic operations or over all possible random choices made by the asynchronous machine. For a survey of such work, please see the Ph.D. thesis of Subramonian [40].

3.3 Memory contention

Another important issue that is arguably not fully modeled by the standard PRAM models is memory contention. The performance of most existing parallel machines significantly degrades if there are

a lot of simultaneous accesses to the same memory cell or to the same module of memory cells. Typically, such requests get queued up and processed in a (slow) sequential fashion. Fortunately, this issue is addressed in recent papers by Dwork, Herlihy, and Waarts [17], who define an asynchronous parallel model that serves simultaneous memory accesses in a queued fashion, and by Gibbons, Matias, and Ramachandran [20], who define synchronous and asynchronous versions of a model they call the QRQW PRAM. In this model k simultaneous accesses to the same memory cell require $f(k)$ time to process, for some function f (usually $f(k) = k$). This model more-accurately accounts for the way real parallel machines handle memory contention than the existing PRAM models, which either assume that $f(k) = \infty$ for $k > 1$ (as in the EREW PRAM model) or assume that $f(k) = 1$ (as in the CRCW PRAM model). Interestingly, Gibbons et al. show that a randomized QRQW PRAM is strictly stronger than a randomized EREW PRAM.

A related model is the Distributed Memory Machine (DMM) or Module Parallel Computer (MPC), where memory is partitioned into modules, one per processor, such that each module can accept only one access at a time. Access to a module is made through an *access window*, which is a register that can be read from or written to by the module as well as the processors in the DMM. As with the PRAM, the DMM is subclassified by the mechanism used to resolve access conflicts to access windows, with EREW and CRCW schemes being common. This model more realistically captures the fact that most memory modules have many more memory cells than input/output pins with which to access those cells. The main computational issues addressed for DMMs, then, is how they can simulate PRAMs, typically through the use of data replication and/or hashing [16, 25, 26, 31, 34, 35, 39]. For example, Karp, Luby, and Meyer auf der Heide [26] show how to simulate an $O(n \log\log n)$-processor CRCW PRAM on a randomized n-processor CRCW DMM with delay $O(\log\log n \log^* n)$. For more information on such results, please see the survey paper by Meyer auf der Heide [35].

3.4 Latency

Another issue that is not specifically addressed by the PRAM is *latency*, the time that elapses between the issuance of a memory access request and the receipt of the response to that request at the processor that issued it. Of course, if all accesses have the same latency and accesses cannot be pipelined, then one can simply multiply the running time of a PRAM algorithm by some latency delay factor in order to derive a more realistic running time. But if either of these conditions fails to hold, then one may wish to explicitly model the latency properties. For example, parallel architectures that use caches, parallel disks, or other hierarchical memories are instances when these conditions fail. There may be a considerable delay for accessing a specific memory cell, c, but accessing the memory cells neighboring c may subsequently become much faster. Several parallel hierarchical memory models have been introduced so as to model this phenomenon, and several researchers have developed efficient algorithms for solving fundamental problems, such as routing and sorting, in these models [1, 2, 3, 4, 5, 13, 14, 21, 37, 38, 45]. For more information on these algorithms and models, please see the survey paper by Vitter [44], as well as some of the more recent papers just referenced.

4 Conclusion

In broad brush strokes I have reviewed a classic parallel computational model—the PRAM—and I have tried to outline recent work on methods for modeling aspects of parallel architectures that are not captured by the PRAM model—network topology, asynchrony, memory contention, and latency. I would be remiss, however, if I did not mention two recent models, Valiant's *Bulk-Synchronous*

Parallel (BSP) model [43] and the *LogP* model of Culler *et al.* [15, 28], which try to capture the difficulties presented by all of these issues (in a topology-independent fashion). The BSP and LogP models both explicitly account for the latency in memory accesses, as well as the potential for pipelining such accesses (with the LogP model being a bit more general). They both deal with memory contention, and they both allow for asynchrony (with the LogP model being a bit less restrictive). It is more difficult to design optimal algorithms for these models, but, then again, they are dealing "head on" with fundamental difficulties of parallel computation.

Thus, we now have what can be viewed as a complete spectrum of computational models for parallel computation, which ranges from the simple, yet unrealistic, PRAM model to the more-complex, but fairly realistic, BSP and LogP models. One can therefore choose to concentrate on the study of the inherent parallelism in a problem via a PRAM algorithm or one can address the issues in parallel computation are not fully captured by such a PRAM algorithm. Current research in parallel algorithm design seems to be well-balanced in this regard, as demonstrated by the papers that appeared in this year's (SIGACT co-sponsored) ACM Symposium on Parallel Algorithms and Architectures (SPAA). According to my classification, there are a dozen papers describing PRAM algorithms, a dozen describing network algorithms or protocols, and at least another dozen directed at methods for dealing with asynchrony, memory contention, and/or latency.

References

[1] A. Aggarwal, B. Alpern, A. K. Chandra, and M. Snir, "A model for hierarchical memory," in *Proc. ACM Symp. on Theory of Computing*, 305–314, 1987.

[2] A. Aggarwal, A. K. Chandra, and M. Snir, "On communication latency in PRAM computation," in *Proc. (1st) ACM Symp. on Parallel Algorithms and Architectures*, 11–21, 1989.

[3] A. Aggarwal, A. K. Chandra, and M. Snir, "On communication latency in PRAM computation," in *Proc. (1st) ACM Symp. on Parallel Algorithms and Architectures*, 11–21, 1989.

[4] A. Aggarwal and C. G. Plaxton, "Optimal parallel sorting in multi-level storage," in *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, 1994.

[5] A. Aggarwal and J. S. Vitter, "The input/output complexity of sorting and related problems," *Comm. ACM*, **31**, 1116–1127, 1987.

[6] S. G. Akl, *The Design and Analysis of Parallel Algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1989.

[7] S. G. Akl and K. A. Lyons, *Parallel Computational Geometry*, Prentice-Hall, 1993.

[8] R. J. Anderson, "Primitives for asynchronous list compression," in *Proc. 4th ACM Symp. on Parallel Algorithms and Architectures*, 199–208, 1992.

[9] R. J. Anderson and H. Woll, "Wait-free parallel algorithms for the union-find problem," in *Proc. 23rd ACM Symp. on Theory of Computing*, 370–380, 1991.

[10] J. Aspnes and M. Herlihy, "Wait-free data structures in the asynchronous PRAM model," in *Proc. 2nd ACM Symp. on Parallel Algorithms and Architectures*, 340–349, 1990.

[11] Y. Aumann, Z. M. Kedem, K. V. Palem, and M. O. Rabin, "Highly efficient asynchronous execution of large-grained parallel programs," in *Proc. 34th IEEE Symp. on Foundations of Computer Science*, 1993.

[12] R. Cole and O. Zajicek, "The expected advantage of asynchrony," in *Proc. 2nd ACM Symp. on Parallel Algorithms and Architectures*, 85–94, 1990.

[13] T. H. Cormen, "Asymptotically tight bounds for performing BMMC permutations on parallel disk systems," in *Proc. 5th ACM Symp. on Parallel Algorithms and Architectures*, 130–139, 1993.

[14] T. H. Cormen, "Fast permuting on disk arrays," *J. Par. and Dist. Comput.*, **17**, 41–57, 1993.

[15] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a realistic model of parallel computation," in *Proc. 4th ACM SIGPLAN Symp. on Princ. and Practice of Parallel Programming*, 1993.

[16] M. Dietzfelbinger and F. Meyer auf der Heide, "Simple, efficient shared memory simulations," in *Proc. 5th ACM Symp. on Parallel Algorithms and Architectures*, 110–119, 1993.

[17] C. Dwork, M. Herlihy, and O. Waarts, "Contention in shared memory algorithms," in *Proc. 25th ACM Symp. on Theory of Computing*, 174–183, 1993.

[18] A. Gibbons and W. Rytter, *Efficient Parallel Algorithms*, Cambridge University Press, New York, 1988.

[19] P. B. Gibbons, "A more practical PRAM model," in *Proc. (1st) ACM Symp. on Parallel Algorithms and Architectures*, 158–168, 1989.

[20] P. B. Gibbons, Y. Matias, and V. Ramachandran, "The QRQW PRAM: Accounting for contention in parallel algorithms," in *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, 1994.

[21] M. T. Goodrich, J. J. Tsay, D. E. Vengroff, and J. S. Vitter, "External-memory computational geometry," in *Proc. 34th IEEE Symp. on Foundations of Computer Science*, 1993.

[22] T. Hagerup and C. Rüb, "A guided tour of Chernoff bounds," *Information Processing Letters*, **33**, 305–308, 1989/90.

[23] ed. J. H. Reif, *Synthesis of Parallel Algorithms*, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1993.

[24] J. JJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, Reading, Mass., 1992.

[25] A. R. Karlin and E. Upfal, "Parallel hashing: An efficient implementation of shared memory," *J. ACM*, **35**(4), 876–892, 1988.

[26] R. Karp, M. Luby, and F. Meyer auf der Heide, "Efficient pram simulation on distributed machines," in *Proc. 24th ACM Symp. on Theory of Computing*, 318–326, 1992.

[27] R. M. Karp and V. Ramachandran, "Parallel algorithms for shared memory machines," in *Handbook of Theoretical Computer Science*, J. van Leeuwen, editor, Elsevier/The MIT Press, Amsterdam, 869–941, 1990.

[28] R. M. Karp, A. Sahay, E. Santos, and K. E. Schauser, "Optimal broadcast and summation in the LogP model," in *Proc. 5th ACM Symp. on Parallel Algorithms and Architectures*, 142–153, 1993.

[29] C. Kruskal, L. Rudolph, and M. Snir, "A complexity theory of efficient parallel algorithms," *Theoretical Computer Science*, **71**, 95–132, 1990.

[30] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann, San Mateo, CA, 1992.

[31] F. Luccio, A. Pietracaprina, and G. Pucci, "A new scheme for the deterministic simulation of prams in vlsi," *Algorithmica*, **5**, 529–544, 1990.

[32] C. Martel, A. Park, and R. Subramonian, "Work-optimal asynchronous algorithms for shared memory parallel computers," *SIAM J. Comput.*, **21**(6), 1070–1099, 1992.

[33] C. Martel, R. Subramonian, and A. Park, "Asynchronous PRAMs are (almost) as good as synchronous PRAMs," in *Proc. 30th IEEE Symp. on Foundations of Computer Science*, 590–599, 1990.

[34] K. Mehlhorn and U. Vishkin, "Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories," *Acta Informatica*, **9**(1), 29–59, 1984.

[35] F. Meyer auf der Heide, "Hashing stratefies for simulating shared memory on distributed memory machines," in *Parallel Architectures and Their Efficient Use: First Heinz Nixdorf Symposium, Lecture Notes in Computer Science*, vol. 678, Springer-Verlag, 1993.

[36] N. Nishimura, "Asynchronous shared memory parallel computation," in *Proc. 2nd ACM Symp. on Parallel Algorithms and Architectures*, 76–84, 1990.

[37] M. H. Nodine and J. S. Vitter, "Large-scale sorting in parallel memories," in *Proc. 3rd ACM Symp. on Parallel Algorithms and Architectures*, 29–39, 1991.

[38] M. H. Nodine and J. S. Vitter, "Deterministic distribution sort in shared and distributed memory multiprocessors," in *Proc. 5th ACM Symp. on Parallel Algorithms and Architectures*, 120–129, 1993.

[39] A. G. Ranade, "How to emulate shared memory," *J. on Computers and System Sci.*, **42**, 307–326, 1991.

[40] R. Subramonian, *Asynchronous Algorithms for Shared Memory Parallel Computers*, Ph.D. thesis, UC, Davis, 1991.

[41] R. Subramonian, "Designing synchronous algorithms for asynchronous processors," in *Proc. 4th ACM Symp. on Parallel Algorithms and Architectures*, 189–198, 1992.

[42] J. D. Ullman, *Computational Aspects of VLSI*, Morgan Kaufmann, San Mateo, CA, 1992.

[43] L. G. Valiant, "A bridging model for parallel computation," *Comm. ACM*, **33**, 103–111, 1990.

[44] J. S. Vitter, "Efficient memory access in large-scale computation," in *Proc. 1991 Symposium on Theoretical Aspects of Computer Science (STACS)*, Lecture Notes in Computer Science, Springer-Verlag, 1991.

[45] J. S. Vitter and E. A. M. Shriver, "Optimal disk I/O with parallel block transfer," in *Proc. 22nd ACM Symp. on Theory of Computing*, 159–169, 1990.