Parallel Algorithm Design with Coarse-Grained Synchronization

Vijaya Ramachandran*

Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
vlr@cs.utexas.edu

Abstract. We describe the Queuing Shared-Memory (QSM) and Bulk-Synchronous Parallel (BSP) models of parallel computation. The former is shared-memory and the latter is distributed-memory. Both models use the 'bulk-synchronous' paradigm introduced by the BSP model. We describe the relationship of these two models to each other and to the 'LogP' model, and give an overview of algorithmic results on these models.

Keywords: Parallel algorithms; general-purpose parallel computation models; bulk-synchronous models.

1 Introduction

An important goal in parallel processing is the development of general-purpose parallel models and algorithms. However, this task has not been an easy one. The challenge here has been to find the right balance between simplicity, accuracy and broad applicability.

Most of the early work on parallel algorithm design has been on the simple and influential *Parallel Random Access Machine (PRAM)* model (see e.g., [29]). Most of the basic results on parallelism in algorithms for various fundamental problems were developed on this simple shared-memory model. However, this model ignores completely the latency and bandwidth limitations of real parallel machines, and it makes the unrealistic assumption of unit-time global synchronization after every fine-grained parallel step in the computation. Hence algorithms developed using the PRAM model typically do not map well to real machines. In view of this, the design of general-purpose models of parallel computation has been an important topic of study in recent years [3,5,6,7,10,13,15, 18,25,30,32,35,36,39,44]. However, due to the diversity of architectures among parallel machines, this has also proved to be a very challenging task. The challenge here has been to find a model that is general enough to encompass the

* This work was supported in part by NSF Grant CCR-9988160 and Texas Advanced Research Program Grant 3658-0029-1999.

V.N. Alexandrov et al. (Eds.): ICCS 2001, LNCS 2074, pp. 619–627, 2001.

wide variety of parallel machines available, while retaining enough of the essential features of these diverse machines in order to serve as a reasonably faithful model of them.

In this paper we describe the approach taken by the *Queuing Shared-Memory (QSM)* model [22] and the *Bulk-Synchronous Parallel (BSP)* model [46]. The former is shared-memory while the latter is distributed-memory. The two models are distinguished by their use of *bulk synchronization* (which was proposed in [46]).

Bulk-synchronization moves away from the costly overhead of the highly synchronous PRAM models on one hand, and also away from the completely asynchronous nature of actual machines, which makes the design of correct algorithms highly nontrivial. The QSM and BSP are algorithmic/programming models that provide coarse-grained synchronization in a manner that facilitates the design of correct algorithms that achieve good performance.

In the following sections, we define the QSM and BSP models, and provide an overview of known relationships between the models (and between both models and the *LogP* model [14]) as well as some algorithmic results on these models.

2 Definitions of Models

In the following we review the definitions of the BSP [46], LogP [14], and QSM [22] models. These models attempt to capture the key features of real machines while retaining a reasonably high-level programming abstraction. Of these models, the QSM is the simplest because it has only 2 parameters and is shared-memory, which is generally more convenient than message passing for developing parallel algorithms. On the other hand the LogP is more of a performance evaluation model than a model for parallel algorithm design, but we include it here since it is quite similar to the BSP model.

BSP Model. The Bulk-Synchronous Parallel (BSP) model [46,47] consists of p processor/memory components that communicate by sending point-to-point messages. The interconnection network supporting this communication is characterized by a bandwidth parameter g and a latency parameter L. A BSP computation consists of a sequence of "supersteps" separated by global synchronizations. Within each superstep, each processor can send and receive messages and perform local computation, subject to the constraint that messages are sent based only on the state of the processor at the start of the superstep.

Let w be the maximum amount of local work performed by andy processor in a given superstep and let h be the maximum number of messages sent or received by any processor in the superstep; the BSP is said to route an *h-relation* in this superstep. The *cost*, T, of the superstep is defined to be $T = \max(w, g \cdot h, L)$. The time taken by a BSP algorithm is the sum of the costs of the individual supersteps in the algorithm.

The (d, x)-BSP [11] is similar to the BSP, but it also attempts to model memory bank contention and delay in shared-memory systems. The (d, x)-BSP

is parameterized by five parameters, p, g, L, d and x, where p, g and L are as in the original BSP model, the *delay* d is the 'gap' parameter at the memory banks, and the *expansion* x reflects the number of memory banks per processor (i.e., there are $x \cdot p$ memory banks).

The computation of a (d, x)-BSP proceeds in supersteps similar to the BSP. In a given superstep, let h_s be the maximum number of read/write requests made by any processor, and let h_r be the maximum number of read/write requests to any memory bank. Then the cost of the superstep is $\max(w, g \cdot h_s, d \cdot h_r, L)$, where w is as in the BSP.

The original BSP can be viewed as a (d, x)-BSP with $d = g$ and $x = 1$.

LogP Model. The LogP model [14] consists of p processor/memory components communicating through point-to-point messages, and has the following parameters: the *latency* l, which is the time taken by the network to transmit a message from one to processor to another; an *overhead* o, which is the time spend by a processor to transfer a message to or from the network interface, during which time it cannot perform any other operation; the *gap* g, where a processor can send or receive a message no faster than once every g units of time; and a *capacity constraint:* whereby a receiving processor can have no more than $\lceil l/g \rceil$ messages in transit to it. If the number of messages in transit to a destination processor π is $\lceil l/g \rceil$ then a processor that needs to send a message to π *stalls*, and does not perform any operation until the message can be sent. The *nonstalling LogP* is the LogP model in which it is not allowed to have more than $\lceil l/g \rceil$ messages in transit to any processor.

QSM and s-QSM models. The Queuing Shared Memory (QSM) model [22] consists of a number of identical processors, each with its own private memory, that communicate by reading and writing shared memory. Processors execute a sequence of synchronized phases, each consisting of an arbitrary interleaving of shared memory reads, shared memory writes, and local computation. The value returned by a shared-memory read can be used only in a subsequent phase. Concurrent reads or writes (but not both) to the same shared-memory location are permitted in a phase. In the case of multiple writers to a location x, an arbitrary write to x succeeds in writing the value present in x at the end of the phase. The *maximum contention* of a QSM phase is the maximum number of processors reading or writing any given memory location. A phase with no reads or writes is defined to have maximum contention one.

Consider a QSM phase with maximum contention κ. Let m_{op} be the maximum number of local operations performed by any processor in this phase, and let m_{rw} be the maximum number of read/write requests issued by any processor. Then the *time cost* for the phase is $\max(m_{op}, g \cdot m_{rw}, \kappa)$. The *time* of a QSM algorithm is the sum of the time costs for its phases. The *work* of a QSM algorithm is its processor-time product. Since the QSM model does not have a latency parameter, the effect of latency can be incorporated into the performance analysis by counting the number of phases in a QSM algorithm with the goal to minimizing that number.

The s-QSM (*Symmetric QSM*) is a QSM in which the time cost for a phase is $\max(m_{op}, g \cdot m_{rw}, g \cdot \kappa)$, i.e., the gap parameter is applied to the accesses at memory as well as to memory requests issued at processors.

The (g, d)-QSM is the most general version of the QSM. In the (g, d)-QSM, the time cost of a phase is $\max(m_{op}, g \cdot m_{rw}, d \cdot \kappa)$, i.e., the gap parameter g is applied to the memory requests issued at processors, and a different gap parameter d is applied to accesses at memory. Note that the QSM is a $(g, 1)$-QSM and an s-QSM is a (g, g)-QSM.

The special case of QSM and s-QSM where the gap parameter g equals 1, is the QRQW PRAM [19], a precursor to the QSM.

3 Relationships between Models

Table 1 presents recent research results on *work-preserving* emulations between QSM, BSP and LogP models [22,42,43].

An emulation of one model on another is work-preserving if the processor-time bound on the emulating machine is the same as that on the machine being emulated, to within a constant factor. The ratio of the running time on the emulating machine to the running time on the emulated machine is the *slowdown* of the emulation. Typically, the emulating machine has a smaller number of processors and takes proportionately longer to execute. For instance, consider the entry in Table 1 for the emulation of s-QSM on BSP. It states that there is a randomized work-preserving emulation of s-QSM on BSP with a slowdown of $O(L/g + \log p)$. This means that, given a p-processor s-QSM algorithm that runs in time t (and hence with work $w = p \cdot t$), the emulation algorithm will map the p-processor s-QSM algorithm on to a p'-processor BSP, for any $p' \leq p/((L/g) + \log p)$, to run on the BSP in time $t' = O(t \cdot (p/p'))$ w.h.p. in p.

Table 1. All results are randomized and hold w.h.p. except those marked as 'det.', which are deterministic emulations. These results are reported in [22,42,43]. Results are also available for the (d, x)-BSP and (g, d)-QSM.

Slowdown of Work-Preserving Emulations between Parallel Models				
Emulated Models (p procs)	Emulating Models			
	BSP	LogP (stalling)	s-QSM	QSM
BSP		$\log^4 p \ + \ (L/g) \ \log^2 \ p$	$\lceil \frac{g \log p}{L} \rceil$	$\lceil \frac{g \log p}{L} \rceil$
LogP (non-stalling)	L/l (det)[1]	1 (det.)	$\lceil \frac{g \ \log \ p}{l} \rceil$	$\lceil \frac{g \log \ p}{l} \rceil$
s-QSM	$(L/g) + \log p$	$\log^4 p \ + \ (l/g) \ \log^2 \ p$		1 (det.)
QSM	$(L/g) + g \log p$	$\log^4 p + (l/g) \log^2 \ p + g \cdot \log \ p$	g (det.)	

[1] This result is presented in [9] but it is stated there erroneously that it holds for stalling LogP programs.

All emulations results listed in Table 1 are work-preserving, and the one mis-match is between stalling and non-stalling LogP, and here we do not know

how to provide a work-preserving emulation with small slow-down of a stalling LogP on any of the other models. (Note that earlier it was stated erroneously in Bilardi et al. [9] that LogP is essentially equivalent to BSP, but this was refuted in Ramachandran et al. [43]).

This collection of work-preserving emulations with small slowdown between the three models – BSP, LogP and QSM — suggests that these three models are essentially interchangeable (except for the stalling versus nonstalling issue for LogP) in terms of the relevance of algorithm design and analysis on these models to real parallel machines.

4 Algorithms

QSM Algorithmic results. Efficient QSM algorithms for several basic problems follow from the following observations [22]. (An 'EREW' PRAM is the PRAM model in which each shared-memory read or write has at most one access to each memory location. A 'QRQW' PRAM [19] is the QSM model in which the gap parameter has value 1.)

1. *(Self-simulation)* A QSM algorithm that runs in time t using p processors can be made to run on a p'-processor QSM, where $p' < p$, in time $O(t \cdot p/p')$, i.e., while performing the same amount of work.
2. *(EREW and QRQW algorithms on QSM)*
   (a) An EREW or QRQW PRAM algorithm that runs in time t with p processors is a QSM algorithm that runs in time at most $t \cdot g$ with p processors.
   (b) An EREW or QRQW PRAM algorithm in the work-time framework that runs in time t while performing work w implies a QSM algorithm that runs in time at most $t \cdot g$ with w/t processors.
3. *(Simple lower bounds for QSM)* Consider a QSM with gap parameter g.
   (a) Any algorithm in which n distinct items need to be read from or written into global memory must perform work $\Omega(n \cdot g)$.
   (b) Any algorithm that needs to perform a read or write on n distinct global memory locations must perform work $\Omega(n \cdot g)$.

There is a large collection of logarithmic time, linear work EREW and QRQW PRAM algorithms available in the literature. By the second observation mentioned above these algorithms map on to the QSM with the time and work both increased by a factor of g. By the third observation above the resulting QSM algorithms are work-optimal (to within a constant factor). More generally, by working with the QSM model we can leverage on the extensive algorithmic results compiled for the PRAM model.

Some QSM algorithmic results for sorting and list ranking that focus on reducing the number of phases are given in [43]. Related work on minimizing the number of supersteps on a BSP using the notion of *rounds* is reported in [23] for sorting, and in [12] for graph problems. Several lower bounds for the number of phases needed for basic problems are presented in [34]. Some of these lower bounds are given in Table 2. The 'linear approximate compaction' problem

mentioned in Table 2 is a useful subroutine for load-balancing; a simple algorithm for this problem on the QRQW PRAM (and hence the QSM) that improves on the obvious logarithmic time algorithm is given in [20].

Table 2. Lower bounds for s-QSM [34]. ([34] also presents lower bounds for these problems for QSM and BSP.)

problem (n=size of input)	**Deterministic time l.b.**	**Randomized time l.b.**	**# of phases w/ p procs. and $O(n)$ work/phase**
Lin. approx. compaction	$\Omega(g\sqrt{\frac{\log n}{\log\log n}})$	$\Omega(g \log\log n)$	$\Omega(\sqrt{\frac{\log n}{\log(n/p)}})$
OR	$\Omega(\frac{g \log n}{\log\log n})$	$\Omega(g \log^* n)$	$\Omega(\frac{\log n}{\log(n/p)})^\dagger$
Prefix sums, sorting	$\Omega(g \log n)^\dagger$	$\Omega(\frac{g \log n}{\log\log n})$	$\Omega(\frac{\log n}{\log(n/p)})^\dagger$

†This bound is tight since there is an algorithm that achieves this bound.

Some experimental results are presented in [24]. In this paper, the QSM algorithms for prefix sums, list ranking and sorting given in [43] were examined experimentally to evaluate the trade-offs made by the simplicity of the QSM model. The results in [24] indicate that analysis under the QSM model yields quite accurate results for reasonable input sizes.

BSP Algorithmic Results. By the emulation results of the previous section, any QSM algorithm can be mapped on to the BSP to run with equal efficiency and only a small slow-down (with high probability). Thus, all of the results obtained for QSM are effective algorithms for the BSP as well. Additionally, there has been a considerable amount of work on algorithms design for the BSP model. For instance, sorting and related problems are considered in [17, 23], list and graph problems are considered in [12], matrix multiplication and linear algebra problems are considered in [17,46,37], algorithms for dynamic data structures are considered in [8], to cite just a few.

LogP Algorithmic Results. Some basic algorithms for LogP are given in [14]. LogP algorithms for summing and broadcasting are analyzed in great detail in [28]. Several empirical results on performance evaluation of sorting and other algorithms are reported in the literature. However, there are not many other results on design and analysis of LogP algorithms. This appears to be due to the asynchronous nature of the model, and the capacity constraint requirement, which is quite stringent, especially in the nonstalling LogP.

References

1. M. Adler, J. Byer, R. M. Karp, Scheduling Parallel Communication: The h-relation Problem. In *Proc. MFCS*, 1995.
2. M. Adler, P.B. Gibbons, Y. Matias, and V. Ramachandran, Modeling parallel bandwidth: Local vs. global restrictions, In *Proc. 9th ACM Symp. on Parallel Algorithms and Architectures*, 94–105, June 1997.
3. A. Aggarwal, A.K. Chandra, and M. Snir, Communication complexity of PRAMs, *Theoretical Computer Science*, 71(1):3–28, 1990.

4. A. Alexandrov, M.F. Ionescu, K.E. Schauser, and C. Sheiman, LogGP: Incorporating long messages into the LogP model — one step closer towards a realistic model for parallel computation, In *Proc. 7th ACM Symp. on Parallel Algorithms and Architectures*, 95–105, July 1995.
5. B. Alpern, L. Carter, and E. Feig, Uniform memory hierarchies, In *Proc. 31st IEEE Symp. on Foundations of Computer Science*, 600–608, October 1990.
6. Y. Aumann and M.O. Rabin, Clock construction in fully asynchronous parallel systems and PRAM simulation, In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, 147–156, October 1992.
7. A. Bar-Noy and S. Kipnis, Designing broadcasting algorithms in the postal model for message-passing systems, In *Proc. 4th ACM Symp. on Parallel Algorithms and Architectures*, 13–22, 1992.
8. A. Baumker and W. Dittrich, Fully dynamic search trees for an extension of the BSP model, In *Proc. 8th ACM Symp. on Parallel Algorithms and Architectures*, 233–242, June 1996.
9. G. Bilardi, K. T. Herley, A. Pietracaprina, G. Pucci, P. Spirakis. BSP vs LogP. In *Proc. ACM SPAA*, pp. 25–32, 1996.
10. G.E. Blelloch, *Vector Models for Data-Parallel Computing*, The MIT Press, Cambridge, MA, 1990.
11. G.E. Blelloch, P.B. Gibbons, Y. Matias, and M. Zagha, Accounting for memory bank contention and delay in high-bandwidth multiprocessors, In *Proc. 7th ACM Symp. on Parallel Algorithms and Architectures*, 84–94, July 1995.
12. E. Caceres, F. Dehne, A. Ferreira, P. Flocchini, I. Rieping, A. Roncato, N. Santoro, and S. W. Song. Efficient parallel graph algorithms for coarse grained multicomputers and BSP. In *Proc. ICALP*, LNCS 1256, pp. 390-400, 1997.
13. R. Cole and O. Zajicek, The APRAM: Incorporating asynchrony into the PRAM model, In *Proc. 1st ACM Symp. on Parallel Algorithms and Architectures*, 169–178, June 1989.
14. D. Culler, R. Karp, D. 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 Principles and Practices of Parallel Programming*, 1–12, May 1993.
15. C. Dwork, M. Herlihy, and O. Waarts, Contention in shared memory algorithms, In *Proc. 25th ACM Symp. on Theory of Computing*, 174–183, May 1993.
16. S. Fortune and J. Wyllie, Parallelism in random access machines, In *Proc. 10th ACM Symp. on Theory of Computing*, 114–118, May 1978.
17. A.V. Gerbessiotis and L. Valiant, Direct bulk-synchronous parallel algorithms, *Journal of Parallel and Distributed Computing*, 22:251–267, 1994.
18. P.B. Gibbons, A more practical PRAM model, In *Proc. 1st ACM Symp. on Parallel Algorithms and Architectures*, 158–168, June 1989.
19. P.B. Gibbons, Y. Matias, and V. Ramachandran, The Queue-Read Queue-Write PRAM model: Accounting for contention in parallel algorithms, *SIAM Journal on Computing*, vol. 28:733-769, 1999.
20. P.B. Gibbons, Y. Matias, and V. Ramachandran, Efficient low-contention parallel algorithms, *Journal of Computer and System Sciences*, 53(3):417–442, 1996.
21. P.B. Gibbons, Y. Matias, and V. Ramachandran, The Queue-Read Queue-Write Asynchronous PRAM model, *Theoretical Computer Science: Special Issue on Parallel Processing*, vol. 196, 1998, pp. 3-29.

22. P.B. Gibbons, Y. Matias, and V. Ramachandran, Can a shared-memory model serve as a bridging model for parallel computation? *Theory of Computing Systems* Special Issue on *SPAA'97*, 32:327-359, 1999.
23. M. Goodrich, Communication-Efficient Parallel Sorting. In *Proc. STOC*, pp. 247–256, 1996.
24. B. Grayson, M. Dahlin, V. Ramachandran, Experimental evaluation of QSM, a simple shared-memory model. In *Proc. IPPS/SPDP*, 1999.
25. T. Heywood and S. Ranka, A practical hierarchical model of parallel computation: I. The model, *Journal of Parallel and Distributed Computing*, 16:212–232, 1992.
26. B. H. H. Juurlink and H.A.G. Wijshoff, A quantitative comparison of parallel computation models, In *Proc. 8th ACM Symp. on Parallel Algorithms and Architectures*, pp. 13–24, 1996.
27. B.H.H. Juurlink and H.A.G. Wijshoff, The E-BSP Model: Incorporating general locality and unbalanced communication into the BSP Model, In *Proc. Euro-Par'96*, 339–347, August 1996.
28. R. 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, June-July 1993.
29. R.M. Karp and V. Ramachandran, Parallel algorithms for shared-memory machines, In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A*, 869–941. Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 1990.
30. Z.M. Kedem, K.V. Palem, M.O. Rabin, and A. Raghunathan, Efficient program transformations for resilient parallel computation via randomization, In *Proc. 24th ACM Symp. on Theory of Computing*, 306–317, May 1992.
31. K. Kennedy, A research agenda for high performance computing software, In *Developing a Computer Science Agenda for High-Performance Computing*, 106–109. ACM Press, 1994.
32. P. Liu, W. Aiello, and S. Bhatt, An atomic model for message-passing, In *Proc. 5th ACM Symp. on Parallel Algorithms and Architectures*, 154–163, June-July 1993.
33. P.D. MacKenzie and V. Ramachandran, ERCW PRAMs and optical communication, *Theoretical Computer Science: Special Issue on Parallel Processing*, vol. 196, 153–180, 1998.
34. P.D. MacKenzie and V. Ramachandran, Computational bounds for fundamental problems on general-purpose parallel models, In *ACM SPAA*, 1998, pp. 152-163.
35. B.M. Maggs, L.R. Matheson, and R.E. Tarjan, Models of parallel computation: A survey and synthesis, In *Proc. 28th Hawaii International Conf. on System Sciences*, II: 61–70, January 1995.
36. Y. Mansour, N. Nisan, and U. Vishkin, Trade-offs between communication throughput and parallel time, In *Proc. 26th ACM Symp. on Theory of Computing*, 372–381, 1994.
37. W.F. McColl, A BSP realization of Strassen's algorithm, Technical report, Oxford University Computing Laboratory, May 1995.
38. K. Mehlhorn and U. Vishkin, Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories, *Acta Informatica*, 21:339–374, 1984.
39. N. Nishimura, Asynchronous shared memory parallel computation, In *Proc. 2nd ACM Symp. on Parallel Algorithms and Architectures*, 76–84, July 1990.

40. S. Pettie, V. Ramachandran, A time-work optimal parallel algorithm for minimum spanning forest. In Proc. Approx-Random'99, August 1999.
41. C. K. Poon, V. Ramachandran, A randomized linear work EREW PRAM algorithm to find a minimum spanning forest. In Proc. 8th Intl. Symp. on Algorithms and Computation (ISAAC '97), Springer-Verlag LNCS vol. 1530, 1997, pp. 212-222.
42. V. Ramachandran, A general purpose shared-memory model for parallel computation, invited paper in *Algorithms for Parallel Processing*, Volume 105, IMA Volumes in Mathematics and its Applications, Springer-Verlag, pp. 1-17, 1999.
43. V. Ramachandran, B. Grayson, M. Dahlin, Emulations between QSM, BSP and LogP: A framework for general-purpose parallel algorithm design. In *ACM-SIAM SODA'99*, 1999.
44. A.G. Ranade, *Fluent parallel computation*, PhD thesis, Department of Computer Science, Yale University, New Haven, CT, May 1989.
45. L. Snyder, Type architecture, shared memory and the corollary of modest potential, *Annual Review of CS*, I:289–317, 1986.
46. L.G. Valiant, A bridging model for parallel computation, *Communications of the ACM*, 33(8):103–111, 1990.
47. L.G. Valiant, General purpose parallel architectures, In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A*, 943–972. Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 1990.
48. H.A.G. Wijshoff and B.H.H. Juurlink, A quantitative comparison of parallel computation models, In *Proc. 8th ACM Symp. on Parallel Algorithms and Architectures*, 13–24, June 1996.