The Heterogeneous Bulk Synchronous Parallel Model

Tiffani L. Williams and Rebecca J. Parsons

School of Computer Science
University of Central Florida
Orlando, FL 32816-2362
{williams,rebecca}@cs.ucf.edu

Abstract. Trends in parallel computing indicate that heterogeneous parallel computing will be one of the most widespread platforms for computation-intensive applications. A heterogeneous computing environment offers considerably more computational power at a lower cost than a parallel computer. We propose the Heterogeneous Bulk Synchronous Parallel (HBSP) model, which is based on the BSP model of parallel computation, as a framework for developing applications for heterogeneous parallel environments. HBSP enhances the applicability of the BSP model by incorporating parameters that reflect the relative speeds of the heterogeneous computing components. Moreover, we demonstrate the utility of the model by developing parallel algorithms for heterogeneous systems.

1 Introduction

Parallel computers have made an impact on the performance of large-scale scientific and engineering applications such as weather forecasting, earthquake prediction, and seismic data analysis. However, special-purpose massively parallel machines have proven to be expensive to build, difficult to use, and have lagged in performance by taking insufficient advantage of improving technologies. Heterogeneous computing [8, 14] is a cost-effective approach that avoids these disadvantages. A heterogeneous computing environment can represent a diverse suite of architecture types such as Pentium PCs, shared-memory multiprocessors, and high-performance workstations. Unlike parallel computing, such an approach will leverage technologies that have demonstrated sustained success, including: computer networks; microprocessor technology; and shared-memory platforms.

We propose a framework for the development of parallel applications for heterogeneous platforms. Our model is called Heterogeneous Bulk Synchronous Parallel (HBSP), which is an extension to the BSP model of parallel computation [17]. BSP provides guidance on designing applications for good performance on homogeneous parallel machines. Experiments [5] indicate that the model also accurately predicts parallel program performance on a wide range of parallel machines. HBSP enhances the applicability of the BSP model by incorporating parameters that reflect the relative speeds of the heterogeneous computing components.

Our starting point for the development of algorithms for HBSP are efficient BSP or HCGM [10, 11] applications. Specifically, we develop three HBSP algorithms—prefix sums, matrix multiplication, and randomized sample sort—that distribute the computational load according to processor speed without sacrificing performance. In fact, the cost model indicates that wall clock performance is increased in many cases. Furthermore, these algorithms can execute unchanged on both heterogeneous and homogeneous platforms.

The rest of the paper proceeds as follows. Section 2 reviews related work. Section 3 describes the HBSP model. Section 4 presents a sampling of algorithms for HBSP. Concluding remarks and future directions are given in Section 5.

2 Related Work

The theoretical foundations of the BSP model were presented in a series of papers by Valiant [15, 16, 17, 18, 19], which describe the model, how BSP computers can be programmed either in direct mode or in automatic mode (PRAM simulations), and how to construct efficient BSP computers. Other work presents theoretical results, empirical results, or experimental parameterization of BSP programs [1, 2, 3, 4, 5, 21]. Many alternative models of parallel computation have been proposed in the literature—a good survey on this topic are papers by Maggs, Matheson, and Tarjan [9] and Skillicorn and Talia [13].

Several models exist to support heterogeneous parallel computation. However, they are either primarily of theoretical interest or are basically languages/runtime systems without a solid theoretical foundation. For an overview of these approaches, we refer the reader to the surveys by Siegel *et al.* [12] and Weems *et al.* [20]. One notable exception is the the Heterogeneous Coarse-Grained Multicomputer (HCGM) model, developed by Morin [10, 11]. HBSP and HCGM are similar in structure and philosophy. The main difference is that HGCM is not intended to be an accurate predictor of execution times whereas HBSP attempts to provide the developer with predictable algorithmic performance.

3 Heterogeneous BSP

The Heterogeneous Bulk Synchronous Parallel (HBSP) model is a generalization of the BSP model [17] of parallel computation. The BSP model is a useful guide for parallel system development. However, it is inappropriate for heterogeneous parallel systems since it assumes all components have equal computation and communication abilities. The goal of HBSP is to provide a framework that makes parallel computing a viable option for heterogeneous systems. HBSP enhances the applicability of BSP by incorporating parameters that reflect the relative speeds of the heterogeneous computing components.

An HBSP computer is characterized by the following parameters:

- the number of processor-memory components p labeled $P_0, ..., P_{p-1}$;
- the *gap* g_j for $j \in [0..p-1]$, a bandwidth indicator that reflects the speed with which processor j can inject packets into the network;
- the *latency* L, which is the minimum duration of a superstep, and which reflects the latency to send a packet through the network as well as the overhead to perform a barrier synchronization;
- processor parameters c_j for $j \in [0..p-1]$, which indicates the speed of processor j relative to the *slowest* processor, and
- the total speed of the heterogeneous configuration $c = \sum_{i=0}^{p-1} c_i$.

For notational convenience, $P_f(P_s)$ represents the fastest (slowest) processor. The communication time and the computation speed of the fastest (slowest) processor are $g_f(g_s)$ and $c_f(c_s)$, respectively. We assume that c_s is normalized to 1. If $c_i = j$, then P_i is j times faster than P_s.

Computation consists of a sequence of supersteps. During a superstep, each processor performs asynchronously some combination of local computation, message transmissions, and message arrivals. A message sent in one superstep is guaranteed to be available to the destination processor at the beginning of the next superstep. Each superstep is followed by a global synchronization of all the processors.

Execution time of an HBSP computer is as follows. Each processor, P_j, can perform $w_{i,j}$ units of work in $\frac{w_{i,j}}{c_j}$ time units during superstep i. Let $w_i = \max(\frac{w_{i,j}}{c_j})$ represent the largest amount of local computation performed by any processor during superstep i. Let $h_{i,j}$ be the largest number of packets sent or received by processor j in superstep i. Thus, the execution time of superstep i is:

$$w_i + \max_{j \in [0..p-1]} \{g_j \cdot h_{i,j}\} + L \qquad (1)$$

The overall execution time is the sum of the superstep execution times.

The HBSP model leverages existing BSP research. The more complex cost model does not change the basic programming methodology, which relies on the superstep concept. Furthermore, when $c_j = 1$ and $g_j = g_k$, where $0 \leq j, k < p$, HBSP is equivalent to BSP.

4 HBSP Algorithms

This section provides a sampling of applications for the HBSP model based on those proposed by Morin for the HCGM model [10, 11]. Our algorithms, which include prefix sums, matrix multiplication, and randomized sample sort, illustrate the power and elegance of the HBSP model. In each of the applications, the input size is partitioned according to a processor's speed. If c_i is the speed of processor P_i, then P_i holds $\frac{c_i}{c}n$ input elements. When discussing the performance of the algorithms, we will often make use of a coarse-grained assumption, $p \ll n$, i.e., the size of the problem is significantly larger than the number of processors. Our interpretation of "significantly larger" is $p \leq \frac{n}{p}$.

4.1 Prefix Sums

Given a sequence of n numbers $\{x_0, x_1, ..., x_{n-1}\}$, it is required to compute their prefix sums $s_j = x_0 + x_1 + ... + x_j$, for all j, $0 \leq j \leq n-1$. Under HBSP, each processor locally computes its prefix sums and sends the total sum to P_f. Next, P_f computes the prefix sums of this sequence and sends the $(i-1)$st element of the prefix to P_i. Lastly, P_i adds this value to each element of the prefix sums computed in the first step to obtain the prefix sums of the overall result. The prefix sums algorithm is shown below.

```
1. Each processor locally computes the prefix sums of its $\frac{c_i}{c}n$ input elements.
2. Each processor, $P_i$, sends the total sum of its input elements to $P_f$.
3. $P_f$ computes the prefix sums of the $p$ elements received in Step 2.
4. For $1 \leq i \leq p-1, P_f$ sends the $(i-1)$st element computed in Step 3 to $P_i$.
5. Each processor computes its final portion of the prefix sums by adding the
   value received in Step 4 to each of the values computed in Step 1.
```

Analysis. In Step 1 and Step 5, each processor P_i does $O(\frac{c_i}{c}n)$ work and this can be done in $O(\frac{n}{c})$ time. Steps 2 and 4 require a communication time of $\max\{g_s \cdot 1, g_f \cdot p\}$. Step 3 takes $O(\frac{p}{c_f})$ computation time. Since $c_f \geq \frac{c}{p}$ and $p \leq \frac{n}{p}$, $O(\frac{p}{c_f}) \leq O(\frac{n}{c})$. Thus, the algorithm takes time

$$O(\frac{n}{c}) + 2 \cdot \max\{g_s \cdot 1, g_f \cdot p\} + 3L. \quad (2)$$

If $g_s \leq pg_f$, the communication time is $2pg_f$, otherwise it's $2g_s$.

4.2 Matrix Multiplication

Matrix multiplication is perhaps one of the most common operations used in large-scale scientific computing. Given two $n \times n$ matrices A and B, we define the matrix $C = A \times B$ as $C_{i,j} = \sum_{k=0}^{n-1} A_{i,k} \times B_{k,j}$. We assume that matrix A is partitioned among the processors so that each processor, P_i, holds $\frac{c_i}{c}n$ rows of A and $\frac{n}{p}$ columns of B. At the completion of the computation, P_i will hold $\frac{c_i}{c}n$ rows of C. We denote the parts of A, B, and C held by P_i as A_i, B_i, and C_i, respectively. The matrix multiplication algorithm consists of circulating the columns of B among the processors. When P_i receives column j of B, it can compute column j of C_i. Once P_i has seen all columns of B, it will have computed all of C_i. The matrix multiplication algorithm is given below.

```
1. repeat $p$ times.
2.     $P_i$ computes $C_i = A_i \times B_i$.
3.     $P_i$ sends $B_i$ to $P_{(i+1) mod\ p}$.
4. end repeat
```

Analysis Step 3 requires P_i to perform $O(\frac{c_i}{c}n \cdot \frac{n}{p} \cdot n) = O(\frac{n^3 c_i}{cp})$ amount of work. Over p rounds, the total computation time is $O(\frac{n^3}{c})$. During Step 4, each processor sends and receives $\frac{n}{p}$ columns of matrix B. Therefore, the total time of HBSP matrix multiplication is

$$O(\frac{n^3}{c}) + g_s n^2 + pL. \tag{3}$$

4.3 Randomized Sample Sort

One approach for parallel sorting that is suitable for heterogeneous computing is randomized sample sort. It is based on the selection of a set of $p-1$ "splitters" from a set of input keys. In particular, we seek splitters that will divide the input keys into approximately equal-sized buckets. The standard approach is to randomly select pr sample keys from the input set, where r is called the oversampling ratio. The keys are sorted and the keys with ranks $r, 2r, ..., (p-1)r$ are selected as splitters. By choosing a large enough oversampling ratio, it can be shown with high probability that no bucket will contain many more keys than the average [7]. Once processors gain knowledge of the splitters, their keys are partitioned into the appropriate bucket. Afterwards, processor i locally sorts all the keys in bucket i.

When adapting this algorithm to the HBSP model, we change the way in which the splitters are chosen. To balance the work according to the processor speeds $c_0, ..., c_{p-1}$, it is necessary that $O(\frac{c_i}{c}n)$ keys fall between s_i and s_{i+1}. This leads to the following algorithm.

1. Each processor randomly selects a set of r sample keys from its $\frac{c_i}{c}n$ input keys.
2. Each processor, P_i, sends its sample keys to P_f.
3. P_f sorts the pr sample keys. Denote these keys by $sample_0, ..., sample_{pr-1}$ where $sample_i$ is the sample key with rank i in the sorted order. P_f defines $p-1$ splitters, $s_0, ..., s_{p-2}$, where $s_i = sample_{\lceil (\sum_{j=0}^{i} \frac{c_j}{c})pr \rceil}$.
4. P_f broadcasts the $p-1$ splitters to each of the processors.
5. All keys assigned to the ith bucket are sent to the ith processor.
6. Each processor sorts its bucket.

Analysis In Step 1, each processor performs $O(r) \leq O(n)$ amount of work. This requires $O(\frac{n}{c_s})$ time. Step 2 requires a communication time of $\max\{g_s \cdot r, g_f \cdot pr\}$. To sort the pr sample keys, P_f does $O(pr \lg pr) \leq O(n \lg n)$ amount of work. This can be done in $O(\frac{n}{c_f} \lg n)$ time. Broadcasting the $p-1$ splitters requires $\max\{g_s \cdot (p-1), g_f \cdot p(p-1)\}$ communication time. Since each processor is expected to receive approximately $\frac{c_i}{c}n$ keys [11], Step 5 uses $O(\frac{n}{c})$ computation time and $\max\{g_i \cdot \frac{c_i}{c}n\}$ communication time, where $i \in [0..p-1]$. Once each processor receives their keys, sorting them requires $O(\frac{n}{c}\lg n)$ time. Thus, the total time is

$$O\left(\frac{n}{c_f}\lg n\right) + X(r + (p-1)) + \max_{i\in[0..p-1]}\left\{g_i\frac{c_i}{c}n\right\} + 4L, \text{where} \qquad (4)$$

$$X = \begin{cases} pg_f \text{ if } g_s \le pg_f \\ g_s \quad \text{otherwise.} \end{cases}$$

5 Conclusions and Future Directions

The HBSP model provides a framework for the development of parallel applications for heterogeneous platforms. HBSP enhances the applicability of BSP by incorporating parameters that reflect the relative speeds of the heterogeneous computing components. Although the HBSP model is somewhat more complex than BSP, it captures the most important aspects of heterogeneous systems. Existing BSP and HCGM algorithms provide the foundation for the HBSP algorithms presented here. These algorithms suggest that improved performance under HBSP results from utilizing the processor speeds of the underlying system. However, experimental evidence is needed to corroborate this claim.

We plan to extend this work in several directions. First, a library based on BSP*lib* (a small, standardized library of BSP functions) [6] will provide the foundation for HBSP programming. Experiments will be conducted to test the effectiveness of the model on a network of heterogeneous workstations. These experiments will test the predictability, scalability, and efficiency of applications written under HBSP. Currently, the HBSP model only addresses a heterogeneous collection of uniprocessor machines. We are investigating variants to the model to address multiprocessor systems.

In conclusion, the goal of HBSP is to offer a framework that makes parallel computing a viable option for a wide range of tasks. We seek to demonstrate that it can provide a simple programming approach, portable and efficient application code, predictable execution, and scalable performance.

References

[1] R. H. Bisseling. Sparse matrix computations on bulk synchronous parallel computers. In *Proceedings of the International Conference on Industrial and Applied Mathematics*, Hamburg, July 1995.

[2] R. H. Bisseling and W. F. McColl. Scientific computing on bulk synchronous parallel architectures. In B. Pehrson and I. Simon, editors, *Proceedings of the 13th IFIP World Computer Congress*, volume 1, pages 509–514. Elsevier, 1994.

[3] A. V. Gerbessiotis and C. J. Siniolakis. Deterministic sorting and randomized mean finding on the BSP model. In *Eighth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 223–232, June 1996.

[4] A. V. Gerbessiotis and L. G. Valiant. Direct bulk-synchronous parallel algorithms. *Journal of Parallel and Distributed Computing*, 22(2):251–267, August 1994.

[5] M. W. Goudreau, K. Lang, S. Rao, T. Suel, and T. Tsantilas. Towards efficiency and portability: Programming with the BSP model. In *Eighth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 1–12, June 1996.

[6] J. M. D. Hill, B. McColl, D. C. Stefanescu, M. W. Goudreau, K. Lang, S. B. Rao, T. Suel, T. Tsantilas, and R. Bisseling. BSPlib: The BSP programming library. *Parallel Computing*, 24(14):1947–1980, 1998.

[7] J. Huang and Y. Chow. Parallel sorting and data partitioning by sampling. In *IEEE Computer Society's Seventh International Computer Software & Applications Conference (COMPSAC'83)*, pages 627–631, November 1983.

[8] A. Khokhar, V. Prasanna, M. Shaaban, and C. Wang. Heterogeneous computing: Challenges and opportunities. *Computer*, 26(6):18–27, June 1993.

[9] B. M. Maggs, L. R. Matheson, and R. E. Tarjan. Models of parallel computation: A survey and synthesis. In *Proceedings of the 28th Hawaii International Conference on System Sciences*, volume 2, pages 61–70. IEEE Press, January 1995.

[10] P. Morin. Coarse-grained parallel computing on heterogeneous systems. In *Proceedings of the 1998 ACM Symposium on Applied Computing*, pages 629–634, 1998.

[11] P. Morin. Two topics in applied algorithmics. Master's thesis, Carleton University, 1998.

[12] H. J. Siegel, H. G. Dietz, and J. K. Antonio. Software support for heterogeneous computing. In A. B. Tucker, editor, *The Computer Science and Engineering Handbook*, pages 1886—1909. CRC Press, 1997.

[13] D. B. Skillicorn and D. Talia. Models and languages for parallel computation. *ACM Computing Surveys*, 30(2):123–169, June 1998.

[14] L. Smarr and C. E. Catlett. Metacomputing. *Communications of the ACM*, 35(6):45–52, June 1992.

[15] L. G. Valiant. Optimally universal parallel computers. *Philosophical Transactions of the Royal Society of London*, A 326:373–376, 1988.

[16] L. G. Valiant. Bulk-synchronous parallel computers. In M. Reeve and S. E. Zenith, editors, *Parallel Processing and Artificial Intelligence*, pages 15–22. John Wiley & Sons, Chichester, 1989.

[17] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[18] L. G. Valiant. General purpose parallel architectures. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A: Algorithms and Complexity, chapter 18, pages 943–971. MIT Press, Cambridge, MA, 1990.

[19] L. G. Valiant. Why BSP computers? In *Proceedings of the 7th International Parallel Processing Symposium*, pages 2–5. IEEE Press, April 1993.

[20] C. C. Weems, G. E. Weaver, and S. G. Dropsho. Linguistic support for heterogeneous parallel processing: A survey and an approach. In *Proceedings of the Heterogeneous Computing Workshop*, pages 81–88, 1994.

[21] T. L. Williams and M. W. Goudreau. An experimental evaluation of BSP sorting algorithms. In *Proceedings of the 10th IASTED International Conference on Parallel and Distributed Computing Systems*, pages 115–118, October 1998.