Theory, Practice, and a Tool for BSP Performance Prediction

Jonathan M. D. Hill[1], Paul I. Crumpton[1] and David A. Burgess[2]

[1] Oxford University Computing Laboratory, UK.
[2] SCCM, Stanford University, CA94305, USA.

Abstract. The Bulk Synchronous Parallel (BSP) model provides a theoretical framework to accurately predict the execution time of parallel programs. In this paper we describe a BSP programming library that has been developed and contrast two approaches to analysing performance: (1) a pencil and paper method; (2) a profiling tool that analyses trace information generated during program execution. These approaches are evaluated on an industrial application code that solves fluid dynamics equations around a complex aircraft geometry on IBM SP2 and SGI Power Challenge machines. We show how the profiling tool can be used to explore the communication patterns of the CFD code and accurately predict the performance of the application on *any* parallel machine.

1 Introduction

The efficient implementation of complex algorithms onto parallel machines is an arduous task. The resulting performance is often only known once this task has been completed, which is unsatisfactory considering the implementation effort. In this paper the Bulk Synchronous Parallel (BSP) [6, 8] approach to parallel computing is introduced and the ability to accurately predict the performance of BSP applications is discussed. To illustrate the BSP approach, a state-of-the-art Computational Fluid Dynamics (CFD) application is considered which models the flow of air past an aircraft. A simple cost model is derived and then compared with the actual code by analysing the output from a BSP performance tool. The output from the tool can then be "replayed" to quantitatively predict the performance of the application on *any* parallel architecture.

For the application developer the advantages of a BSP library in comparison to conventional message-passing systems such as PVM and MPI are: (a) the BSP cost model is straightforward and accurate; (b) the library eliminates the possibility of deadlock from parallel code; (c) the communication is based upon one-sided remote memory access; (d) the library optimises global scheduling of communication; (e) the library is smaller and simpler than PVM or MPI. Furthermore, the BSP profiling tool depicts the parallel computation and can be used to predict a code's performance on another machine. The BSP approach to parallel programming is applicable to all parallel architectures: distributed memory architectures, shared memory multiprocessors and networks of workstations. It provides a consistent and portable framework for parallel software development.

The structure of the paper is as follows. First we introduce the BSP computational model in §2. This provides the necessary background to cost model the CFD application in §3. In §4 we use a BSP library and profiling tool to visualise and analyse parallel performance, and conclusions are drawn in §5.

2 The Bulk Synchronous Parallel computational model

A BSP calculation consists of a sequence of *supersteps*. Each superstep can be decomposed into three phases: processor-memory pairs perform a number of computations on data held locally at the start of a superstep; processors communicate data into other processor's memories; all processors barrier synchronise at the end of a superstep. The essence of BSP cost modeling is that the cost of a series of supersteps is simply the sum of the costs of each separate superstep. The cost of p processes executing a single superstep is the sum of: (a) the computational cost of the process that takes the longest time to perform its local sequential computation; (b) the communication cost of the global exchange of data; and (c) the synchronisation cost. It is realistic to cost entire BSP algorithms in terms of formulae with structure:

$$\begin{aligned} \textit{execution cost (in flops)} &= \textit{computation} + \textit{communication} + \textit{synchronisation} \\ &= \alpha + \beta g + \gamma l \qquad (1) \\ \textit{execution time (in secs)} &= (\alpha + \beta g + \gamma l)/s \qquad (2) \end{aligned}$$

where γ is the number of supersteps performed by an algorithm, α is the accumulated cost of the local computations of the γ supersteps and β is the total communication cost. These terms are *application-dependent costs*. The BSP parameters s, l, and g are *architecture-dependent constants* that capture the performance of a parallel machine:

- s is the speed of computation of a process in flops.
- l is the synchronisation latency cost in units of s.
- g is the number of flops/word required for all processors to simultaneously communicate a message.

Flops are used as the unit of cost so that algorithms can be costed in an architecture independent way. The use of the parameters s and l is relatively intuitive but the interpretation of g is not so obvious and is discussed in the next subsections.

2.1 The standard "Pencil and Paper" BSP model of g

In traditional message-passing systems such as PVM and MPI the cost of communication has to be considered by analysing individual sender-receiver pairs. This can be both time consuming and difficult to calculate especially prior to the development of parallel software. In contrast, the BSP model considers all the individual communications that occur within a superstep as a single monolithic unit. The cost of these communications is accurately modeled by analysing the

process with the largest amount of data entering or leaving itself. If h words is the largest accumulated size of all messages either entering or leaving a process within a superstep, and since g is defined to be the number of flops required for all processors to simultaneously communicate a single word, then the communication cost in flops is modeled as hg. Patterns of communication of this form are termed h-relations and form the basis of costing communication in the BSP model. For example, an h-relation with cost hg can be realised by each of p processes having a single message of size h coming in and out of them. In contrast, if p processes communicate data of size h/p into a single process then p messages will enter that process and the communication pattern also realises a $h/p \times p = h$-relation with a cost of hg flops. This method of costing communication is accurate for *suitably large values of* h (see §2.2). The values of the BSP machine parameters considered in this paper are shown in Table 1.

Machine	s	p	l		g [32-bit word]		$N_{\frac{1}{2}}$
	flops		flops	μs	flops/word	μs/word	words
SGI Power Challenge	55 million	1	136	2.5	0.4	0.007	64
		2	627	11.4	7.6	0.14	8
		4	1248	22.7	7.4	0.13	9
IBM SP2 (using switch)	26 million	1	223	8.6	1.3	0.05	7
		2	2386	91.8	7.5	0.29	6
		4	4159	160.0	7.7	0.30	6
		8	8340	320.8	7.8	0.30	6
IBM SP2 (using ethernet)	26 million	1	222	8.5	1.3	0.05	7
		2	20120	773.8	183.5	7.1	3
		4	48476	1864.5	384.1	14.8	5
		8	223120	8581.5	1645.3	63.3	2

Table 1. BSP machine parameters s, l and g for p processors (see in §2.2)

2.2 The refined BSP model of g to account for message latency

In the previous section the definition of g was based on a *suitably large h-relation.* The standard BSP model makes no distinction between the costs of one process sending h messages of size one or a single message of size h. However, on a real parallel machine there is a start-up latency associated with every message so the actual communication cost is dependent on message size. Miller [7] refined the standard BSP cost model using Hockney's model [5] to accurately include the effect of message granularity in the communication cost. In the refined BSP model, g is defined as a function of the message size x:

$$g(x) = \left(\frac{N_{\frac{1}{2}}}{x} + 1\right) g_\infty \tag{3}$$

where g_∞ is the asymptotic communication cost for very large messages (g reported in Table 1 is g_∞) and $N_{\frac{1}{2}}$ is the size of message that produces half the optimal bandwidth of the machine so $g(N_{\frac{1}{2}}) = 2g_\infty$. The incorporation of $N_{\frac{1}{2}}$

into the original BSP cost model of §2.1 undermines the bulk properties assumed when costing h-relations. It's introduction would make cost analysis as complex as in message passing systems as each seperate message size occurring in a superstep would have to be taken into account. One way of overcoming this problem is to provide an implementation of the BSP programming library that coalesces small messages together. The effect of this scheme is that at-most a single message will leave or enter a process during each superstep, and the cost $hg(h)$ can therefore be attributed to the superstep. As a consequence, for simple "pencil and paper" modeling it is sufficient to use the standard BSP cost model. However, the BSP profiling tool described later in this paper uses the refined model $hg(h)$ for accurate cost modeling and performance prediction.

3 Pencil and Paper BSP cost model of CFD application

We demonstrate the effectiveness of the BSP approach with a complex Computational Fluid Dynamics (CFD) application. This application simulates an inviscid flow over an aircraft on an unstructured tetrahedral mesh using an edge collapsing multigrid technique [2].

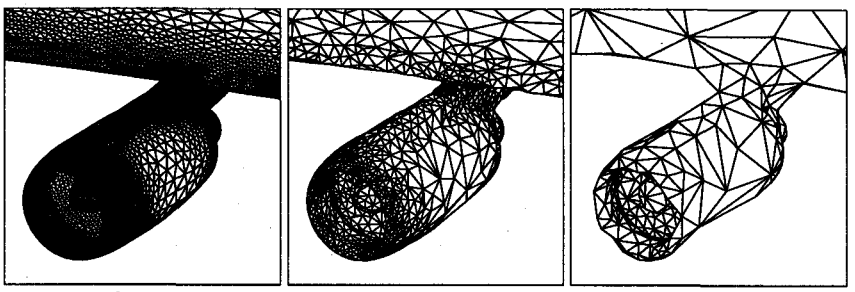


Fig. 1. Sequence of 3 grids for an aircraft configuration.

The underlying CFD method uses a 3D unstructured tetrahedral grid that tessellates the computational domain around an aircraft, see Figure 1. The two main components of the algorithm are a "smoother" and a sequence of successively coarser grids. The smoother is an explicit iterative method that converges to a fixed solution. The main computational cost is two loops over the tetrahedra which consist of: (a) gathering information from the four vertices of each tetrahedron; (b) performing a large quantity of arithmetic; (c) scattering information from the tetrahedra to the vertices. The multigrid method cycles through the grid sequence in a W-shape as will be seen in the profile of Figure 3.

For unstructured grid applications the computational cost of a smoother iteration (in flops) is directly proportional to the number of cells N within a particular tetrahedral grid. Thus the cost is C_1N flops where C_1 is the flop count per cell. Hence the total computational cost of smoothing on the finest mesh at the beginning and end of the multigrid W-cycle is $2C_1N$. The cost of

smoothing the first coarse grid in the W-cycle is $4C_1Nr$, where r $(0 < r < 1)$ is the reduction fraction describing the fewer cells on the coarser grid. Thus on a sequential machine the computational cost is:

$$\textit{sequential computational cost} = \sum_{i=1}^{N_{grids}} 2^i C_1 N r^{i-1}.$$

To execute this in parallel the grids are partitioned and placed on the processors. For p processors each process owns (N/p) cells of the finest grid and each partition has an exterior surface proportional to $(N/p)^\gamma$ cells, where $\gamma < 1$. The parallel smoothing cost in flops of a single grid can thus be expressed as

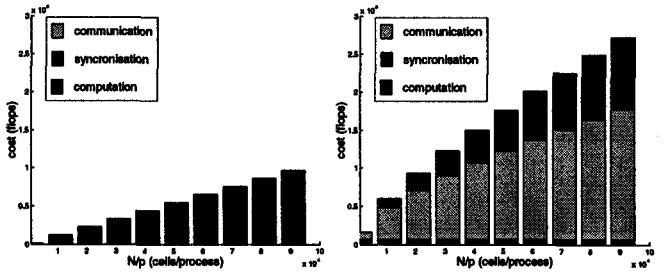
$$\textit{parallel smoothing cost} = C_1 \left(\frac{N}{p}\right) + C_2 \left(\frac{N}{p}\right)^\gamma g + C_3 l$$

where l and g are the BSP parameters, C_2 is the amount of data (in words) that is exchanged between partitions and C_3 is the number of synchronisations per smoothing iteration. The cost of a multigrid W-cycle can be then written as

$$\textit{parallel Wcycle cost} = \sum_{i=1}^{N_{grids}} 2^i \left[C_1 \left(\frac{Nr^{i-1}}{p}\right) + C_2 \left(\frac{Nr^{i-1}}{p}\right)^\gamma g + C_3 l \right] \quad (4)$$

Equation (4) represents a rather simplistic model and only contains the main cost terms of W-cycle multigrid on a parallel machine. The assumptions made in this model are: the tetrahedra are partitioned perfectly; the communication cost of the grid transfer operations is negligible; the execution of boundary conditions is negligible. Because of these assumptions we *expect the cost model to be optimistic.* To improve the accuracy of the model more information about the particular application is required. If a sequential code already exists then by profiling the code on a given machine an accurate computational time can be determined. If data partitioning information is also known then the number of cells per partition and the number of vertices on partition boundaries can be used to accurately determine the computation and communication costs respectively. It is worth noting that converting the sequential multigrid cost model to a BSP parallel cost model is a straightforward task and gives good qualitative information.

In this "paper and pencil" study, execution times are contrasted for the IBM SP2 configured to use either ethernet or a high-performance switch. The grid dependent values are $N = 7.5 \times 10^5$, $r = 1/8$ and $N_{grids} = 3$. The algorithmic constants of the smoother are estimated to be $C_1 = 400$, $C_2 = 15$ and $C_3 = 2$. Figure 2 plots N/p against IBM SP2 cost in flops, using the simplistic cost model in Equation (4). The values of l and g for 8 processors are used so $l = 8340$ and $g = 8$ for switch, and $l = 223120$ and $g = 1645$ for ethernet. The graph clearly portrays that for ethernet communication to be effective with 8 processors at least 10^6 fine grid tetrahedra need to be placed on each processor, whereas for the switch communication only 10^2 are needed per processor. This would be invaluable information for a prospective user looking to purchase a parallel machine, who presumably would have some idea of the target problem size. In



(a) IBM SP2 using the switch (b) IBM SP2 using ethernet

Fig. 2. Cost model predictions of the multigrid CFD code using Equation (4).

addition, the cost model might predict the futility of parallelising a favoured algorithm or parallel strategy, which could prove to be essential. The scalability of an algorithm as p becomes large can also be studied using this technique. This approach is by no means revolutionary, but the simplicity, portability and accuracy of the BSP model gives an application programmer a significant advantage over message-passing modeling.

4 Results from a BSP library and profiling tool

A number of researchers are currently forming a World-Wide Standard BSP Library [3] by synthesising several low-level BSP programming approaches that have been pursued over the last few years [4, 7]. They propose a library called *BSPlib* to provide a parallel communication library based around a SPMD model of computation. The results described in this paper have been obtained using the Oxford BSP toolset implementation of *BSPlib*.

The implementation of the multigrid CFD application was carried out in FORTRAN 77 using the OPlus (Oxford Parallel Library for Unstructured Solvers) high-level programming framework [1]. The OPlus library uses the Oxford BSP toolset for communication. OPlus removes the burdens of parallelisation from the application programmer by handling data partitioning, I/O and the organisation of the data transfers.

Profiling of parallel execution is valuable for evaluating performance and predicting performance on other machines. Both performance evaluation and prediction with message-passing software is difficult. With the BSP approach it is greatly simplified as all the *BSPlib* communication is carried out simultaneously within a BSP superstep. The profiling tool that we have developed graphically exposes three important pieces of information: (a) the elapsed time taken to perform communication; (b) the pattern of communication; (c) the computational elapsed time.

Figure 3 shows a *prediction* profile that compares the actual execution on IBM SP2 (with switch communication) in the top graph with predicted cost in

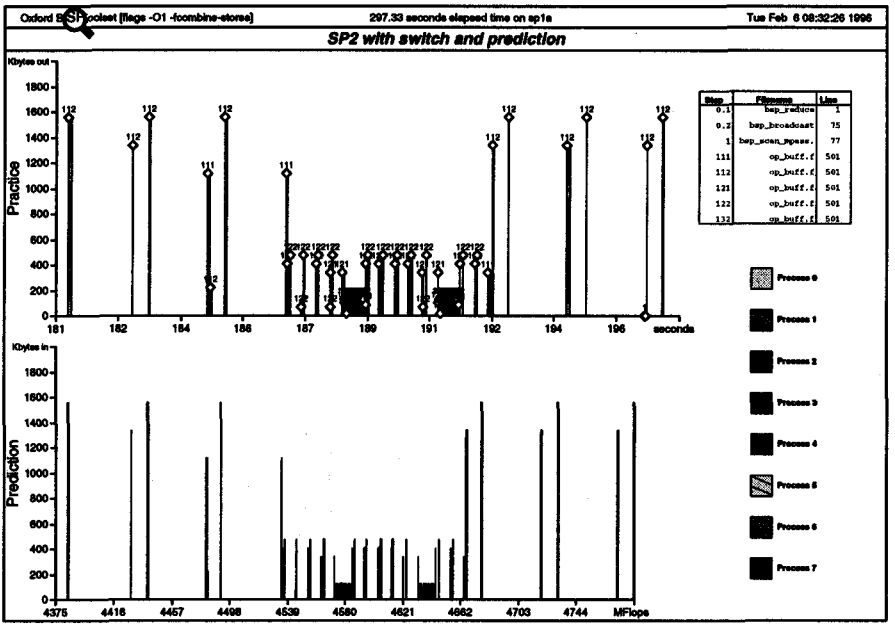


Fig. 3. Comparing actual and predicted cost of W-cycle on IBM SP2 with switch.

the lower. The match between practice and prediction is close, although there are a few discrepancies where some communications take longer than expected.

The salient features of figure 3 are that the white space between the bars represents the elapsed time of the process that takes the longest time to perform its local computation phase of a superstep, whilst the bars in the top graph show the total sizes of all the communications leaving a process during a superstep. The elapsed time taken to perform communication is visualised by the width of the communication bars. The size of the h-relation is identified by the thickest band in the bar, which is easily seen in Figure 4. The label found at the top left-hand corner of each bar is the number of the superstep and in the legend the superstep label identifies the position in the code. It is clear that computation dominates the overall time, as the pencil and paper BSP model predicted for the IBM SP2 using the switch as shown in Figure 2(a).

A major advantage of the Oxford BSP profiling tool is to predict the performance of codes on other parallel machines. This is simply achieved by plugging the BSP parameters of the appropriate machine into the tool. Figure 4 shows the result of this exercise when the profile data generated from the execution of the code using the IBM SP2 with switch is used to predict the performance of the code when ethernet is used. The top graph shows the actual cost on the IBM SP2 with switch and the lower graph predicts the cost of a IBM SP2 with ethernet. Normally the reverse process would be performed when the cost of a code running on a network of workstations is used to predict the cost of the

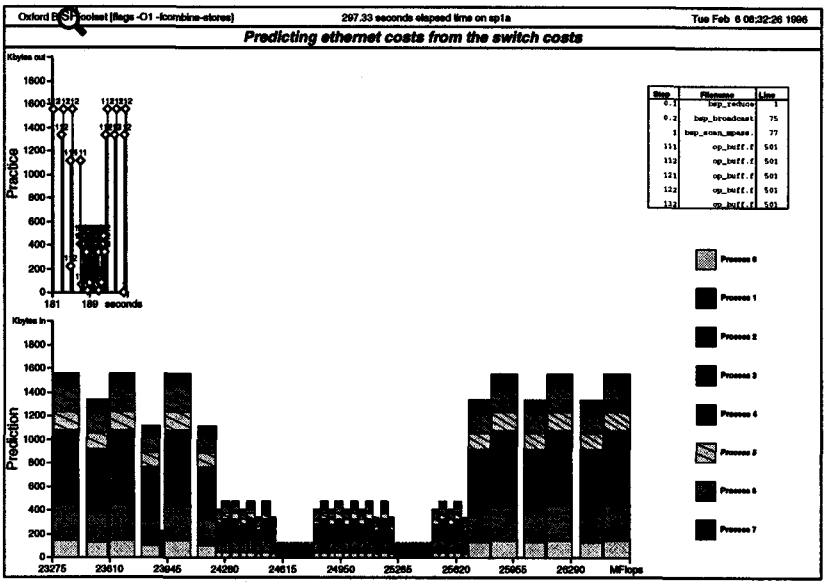


Fig. 4. Predicting the cost on ethernet from data gathered on the switch.

code on a *real* parallel machine. The tool predicts that the W-cycle will take $26625 - 23275 = 3350 Mflops$, which is equivalent to 128 seconds on the SP2 with ethernet. Figure 5 shows the actual cost of 101 seconds for the W-cycle using ethernet communication. The theoretical model shown in figure 2 shows an expected difference between the switch and ethernet as a factor of approximately three at $N/p = 9.3 \times 10^3$. This factor is only an approximation, as many assumptions were made when costing the algorithm by hand. The tool predicts that ethernet will be a factor seven (i.e, $128/17 = 7.5$ as shown in figure 3) slower than switch communication. The actual experimental ratio is $101/17 = 5.9$.

5 Conclusions

The BSP approach has a simple cost model, is deadlock-free and is portable. In general, cost-modeling applications gives a rough ball-park figure of the cost on any parallel machine and configuration size. The role of the profiling tool aids simplistic pencil and paper cost modeling, and it effectively predicts the cost of an algorithm on any parallel machine.

Acknowledgments

This work was performed within Oxford Parallel with financial support from Rolls-Royce plc, EPSRC and NSF grant ECS-9527123. We gratefully acknowledge the use of the unstructured grid generator of Jaime Peraire and Joaquim Peiro. We would also like to thank Bill McColl, Mike Giles, David Skillicorn, and Bob McLatchie.

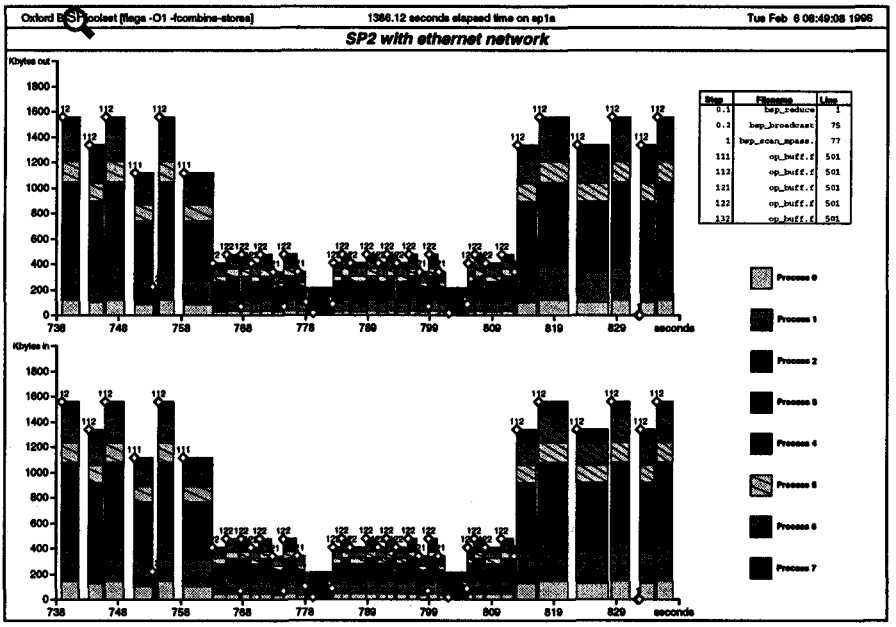


Fig. 5. Observed performance of a "W-cycle" over ethernet.

References

1. D. Burgess, P. Crumpton, and M. Giles. A parallel framework for unstructured grid solvers. In *Computational Fluid Dynamics ECCOMAS'94*, pages 391–396. John Wiley & Sons, 1994.
2. P. Crumpton and M. Giles. Implicit time accurate solutions on unstructured dynamic grids. AIAA Paper 95-1671, 1995.
3. M. W. Goudreau, J. M. D. Hill, K. Lang, B. McColl, S. B. Rao, D. C. Stefanescu, T. Suel, and T. Tsantilas. A proposal for the BSP Worldwide standard library, April 1996. See `www.bsp-worldwide.org` for more details.
4. M. W. Goudreau, K. Lang, S. B. Rao, T. Suel, and T. Tsantilas. Towards efficiency and portability: Programming with the BSP model. In *Proc. 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1996.
5. R. W. Hockney. Performance parameters and benchmarking of supercomputers. *Parallel Computing*, 17:1111–1130, 1991.
6. W. F. McColl. Scalable computing. In J. van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, number 1000 in Lecture notes in Computer Science, pages 46–61. Springer-Verlag, 1995.
7. R. Miller. *Two approaches to architecture-independent parallel computation.* D.Phil thesis, Oxford University, Michaelmas Term 1994.
8. L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.