Programming Research Group

THE THEORY, PRACTICE, AND A TOOL FOR BSP PERFORMANCE PREDICTION APPLIED TO A CFD APPLICATION

Jonathan M. D. Hill
Paul I. Crumpton
David A. Burgess

PRG-TR-4-1996

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

The theory, practice, and a tool for BSP performance prediction applied to a CFD application

Jonathan M. D. Hill   Paul I. Crumpton   David A. Burgess

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 with a theoretical cost model; (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 an IBM SP2 and SGI PowerChallenge. We show how the 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. Furthermore, the resulting performance is often only known once this task has been completed. This is unsatisfactory considering the implementation effort. In this paper the Bulk Synchronous Parallel (BSP) [14, 10] approach to parallel computing is introduced and the ability to accurately predict the performance of BSP applications is discussed in detail.

To illustrate the BSP approach, a state-of-the-art computational fluid dynamics application is considered which models the flow 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 advantage of this implementation of the BSP model in comparison to conventional message-passing systems are: (1) the model eliminates the possibility of deadlock from parallel code; (2) one-sided communication based upon remote memory access is used to exchange data among processors. This style of communication removes the intellectual burden of matching send-receive pairs; (3) unlike either PVM [7] or MPI [11] the library is small with only eight subroutines; (4) it has a straightforward and accurate cost model.

This approach to parallel programming is applicable to all kinds of parallel architecture: distributed memory architectures, shared memory multiprocessors, and networks of workstations. It provides a consistent and portable framework for parallel software development.

2 The BSP model

The bulk synchronous parallel (BSP) computational model views a parallel machine as a set of processor-memory pairs, a global communications network, and mechanism for synchronising all processors. The essence of the BSP approach to parallel programming is the notion of the superstep in which communication and synchronisation are completely decoupled.

A BSP calculation consists of a sequence of supersteps. Each superstep can be decomposed into three phases: (1) each processor-memory pair can perform a number of computations on data held locally at the start of superstep; (2) the processors can remotely *get* data from or *put* data into other processor's memories; (3) these communications occur *en masse* at the end of the superstep and are followed by a barrier synchronisation of all processors. The barrier ensures that a global view of a BSP computation can be kept in check at regular intervals within a program.

An important feature of the model is the ability to easily assess the cost of parallel algorithms. To ensure cost analysis is simple, the model assumes that processor locality is irrelevant when costing communication. The rationale for this simplification is that if the communication patterns within an algorithm utilise communication hardware in a machine-specific way, then the goal of portable parallel programming will be compromised.

3 BSP cost modeling

The essence of BSP cost modeling is that the cost of a series of supersteps executed one after the other is simply the sum of the costs of each separate superstep. The computations within a superstep can be decomposed into three distinct phases: local computation, communication, and synchronisation. Therefore the cost of p processes executing a single superstep is the sum of (1) the computational cost of the process that takes the longest time to perform it's local sequential computation; (2) the communication cost of the global exchange of data (this is taken to be proportional to the largest amount of data either entering or leaving a *single* process); (3) the synchronisation cost. It is therefore realistic to cost entire BSP algorithms in terms of formulae that have the structure:

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

where γ is the number of supersteps performed by an algorithm; α is the sum of the *costs attributed* to the local computations in each of the γ supersteps; and β is the sum of all the costs attributed to communication for each superstep. These terms are *architecture-independent costs.* The BSP parameters s, l, and g are constants that capture the performance of the parallel machine:

s is the rate of computation of a process in flops (i.e., the number of floating point operations not flop per second). It is used to calibrate l and g.

l is the synchronisation latency cost in units of s. It is the minimum amount of time for all processors to barrier synchronise. For the purposes of the cost model, l is assumed to be independent of number of processors. The validity of this assumption is discussed in section 3.2.

g is the number of flops required for all processors to simultaneously communicate a single word.

The use of the parameters s and l is relatively intuitive. For example, if n supersteps are executed one after the other, and each of p processes performs f floating point operations on local data, then the cost of the supersteps in flops is $f + nl$. We use flops as our unit of cost so that algorithms can be costed in an architecture independent way—albeit a crude measure. This does not mean that we count the number of flops performed by a program. The interpretation of the cost is that $f + nl$ floating point operations *could have been performed by a single processor* in the same time as to execute all the supersteps. The interpretation of g is not so simple, and is discussed in more detail in the next section.

3.1 The computation to communication ratio g

In traditional message passing systems such as PVM [7] and MPI [11] the cost of communication has to be considered in terms of sender-receiver pairs. In contrast, the *bulk* aspect of 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 modelled by analysing the process with the largest amount of data entering or leaving itself. If h 32-bit 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, the simplest form of 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 h-relations* (see section 3.2).

Machine	s	p	l		g [32-bit word]		$N_{\frac{1}{2}}$
	flops		flops	μs	flops	μs	words
SGI PowerChallenge	55 million	1	136	2.5	0.4	0.007	64
		2	627	11.4	7.6	0.14	8
		3	940	17.1	7.3	0.13	9
		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	0.3	6
		8	8340	320.8	7.8	0.3	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

The values of the BSP machine parameters considered in this paper are shown in Table 1.

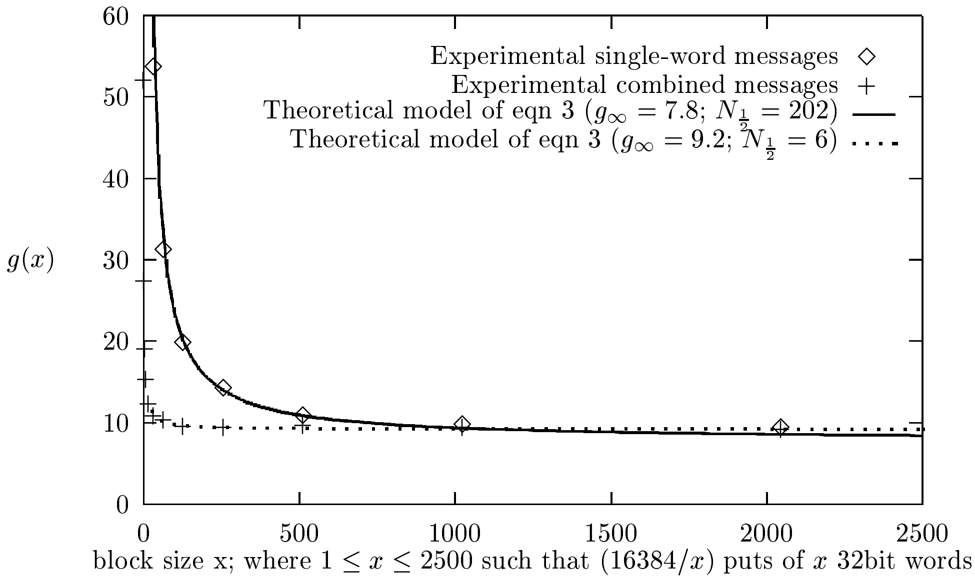


Figure 1: Asymptotic communication bandwidth for an 8 processor IBM SP2 using the switch

The BSP machine parameters l and g are constant for a scalable parallel machine. However, no assumptions about the scalability of real parallel machines are made, so we give parameter entries for different numbers of processors. For the switching network of the IBM SP2 [1], g remains fixed as the number of processors p is scaled. However, if the SP2 communications hardware is changed to ethernet, then the machine is not scalable. As a rough guide to the relative efficiency of communication, the cost of g for $p = 1$ represents the memory speed of a machine. Comparing this value with g for $p > 1$ gives a ratio of inter-processor communication to memory speed, i.e., a factor of six[1] for the IBM SP2.

3.2 Is bulk communication too simple a model?

One obvious concern with the prior definition of g was the rather woolly idea of a *suitably large h-relation*. The BSP model makes no distinction between the costs of one process sending h messages of size one, or a single message of size h. Although both communications have the same BSP cost as they both realise a h-relation with cost hg, it seems extremely difficult to believe that their communication time will be the same on a real parallel machine.

Miller [13] refines the BSP cost model by using Hockney's [9] model for including the effect of message granularity on communication cost. In the refined model, g is defined as a function $g(x)$, where x is the message size, and g_∞ is the asymptotic communication cost for very large messages (g reported in Table 1 is g_∞).

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

$N_{\frac{1}{2}}$ is the size of message that produces half the optimal bandwidth of the machine, i.e,

[1] i.e., *g for 8 processors/g for 1 processor = 7.8/1.3*

$g(N_{\frac{1}{2}}) = 2g_\infty$. Equation 3 therefore accounts for the extra startup latency costs associated with lots of small messages. As the message size x increases above $N_{\frac{1}{2}}$ the effect of latency becomes negligible. Figure 1 shows the experimental results for $g(x)$ when differing block sizes are communicated on an eight processor SP2. The curve labelled "single word messages" in the figure shows the different values of $g(x)$ when each *put* communication is implemented as a separate communication within a superstep. The value for $N_{\frac{1}{2}}$ is determined by equation 3 by fitting a curve to the experimental data. The actual communication cost is $1 \times g(1) = 202g_\infty$ instead of the 1-relation value of g_∞. Such a large factor cannot be ignored when trying to produce an accurate cost model.

Although it seems that $N_{\frac{1}{2}}$ cannot be ignored, its incorporation into the BSP cost model undermines the bulk properties assumed when costing h-relations. The reason for this is that costing would become as difficult as in message passing systems as each of the separate costs for all message sizes occurring in a superstep would have to be taken into account. In fact, we believe this is one of the failings of the LogP model [6] (which is very similar to the BSP model) as it includes an extra parameter o that models message size.

To reconcile the perceived gulf between theory and practice our approach has been to ensure that our implementation of the BSP programming library provides an accurate simulation of the *standard* cost model in which the cost of a h-relation is hg. This is possible by delaying any communications until the end of a superstep, so that multiple communications to the same process can be coalesced into a single message. As the semantics of the BSP model insist that communications only take effect at the end of a superstep, this combining scheme is semantically safe.

The lower curve in figure 1 shows the effect of the combining optimisation. As expected it reduces the effective communication startup latency $N_{\frac{1}{2}}$. More importantly it ensures that the mix of message sizes communicated in a superstep can be ignored when costing algorithms. Only the accumulated size of messages is important—this is exactly what is required when costing communications in terms of h-relations!

To ensure that the standard model holds a minimum size of h-relation must be satisfied. This value is Valiant's [14] h_0 parameter. $N_{\frac{1}{2}}$ is easy to calculate empirically so we define h_0 to be the size of an h-relation such that the experimental cost is within $y\%$ accuracy of $h_0 g_\infty$. i.e.,

$$\begin{aligned}\left(\frac{100+y}{100}\right) h_0 g_\infty &= h_0 g(h_0) = \left(\frac{N_{\frac{1}{2}}}{h_0} + 1\right) h_0 g_\infty \\ h_0 &= \frac{100 N_{\frac{1}{2}}}{y} \qquad (4)\end{aligned}$$

To be within 10% accuracy on the SP2 with switch, h-relations have to be greater than 0.2Kbytes—a figure that should be easily attainable if BSP communications can really be termed *bulk*.

In the remainder of the paper we use the standard model for costing h-relations when we develop the theoretical model of the CFD application. However, to make our profiling tool more accurate, the cost $hg(h)$ is used when costing supersteps—we believe this approach would normally be too cumbersome when costing algorithms by hand.

Also, as alluded to earlier, the cost model assumes l and g are independent of p. However, as is clearly shown in Table 1, this doesn't always hold. Thus when costing algorithms by hand, we assume l and g are fixed. In contrast, the profiling tool uses the specific value of l and g_∞ (used in equation 3) for the size of machine p used when profiling.

4 A toolset for BSP programming

The Oxford BSP toolset provides a parallel communication library based around a SPMD model of computation. The toolset builds upon the ideas in Miller's BSP library [12] and provides the basis for a proposed world-wide standard for a BSP programming library [8]. A major strength of the toolset is its simplicity with only eight operations required for a C or FORTRAN programmer.

Creating BSP processes Processes are created by the procedures `begin_bsp` and `end_bsp` which bracket a piece of code to be run in a SPMD manner among a number of processors.

Superstep synchronisation The library decouples the local computation phase of a superstep, from the global exchange of data and synchronisation phase of a superstep. This is achieved by using the the library operations `begin_superstep` and `end_superstep` to bracket the part of a superstep where communication is performed. Any *put* and *get* communications have to occur between these two delimiters. At the end of a superstep, marked by the closing `end_superstep`, all processors barrier synchronise. The local computation phase of a superstep is implicitly defined as the sequence of operations between a call to `end_superstep` and the next `begin_superstep` encountered during the program execution. Only after `end_superstep` are the communications that occurred within the superstep guaranteed to have taken place.

Bulk synchronous remote memory access The library uses one-sided Direct Remote Memory Access (DRMA) communication facilities to perform data-communication. This is similar to the method of communicating in the Cray Shmem library on the T3D. In this mode of operation, the local address space of each process can be manipulated by other processes by using either `bsp_get` or `bsp_put`. The operation `bsp_put` stores locally held data into the local memory of a target process, without the active participation of the target process. The operation `bsp_get` reaches into the local memory of another process to copy data values held there into a data structure in its own local memory. Unlike data-communication based upon message passing, communications within the superstep framework can never dead-lock. This is a major benefit of the BSP approach!

4.1 A tool for visualising communication patterns

Figure 2 shows a *communication* profile for four different supersteps that realise a 500-relation on an 8 processor SP2.

superstep 1 each of 8 processes *put* 500 integers into their right neighbour in a cyclic manner.

superstep 2 process i (where $0 \leq i \leq 4$) *put* 500 integers into process $i + 4$.

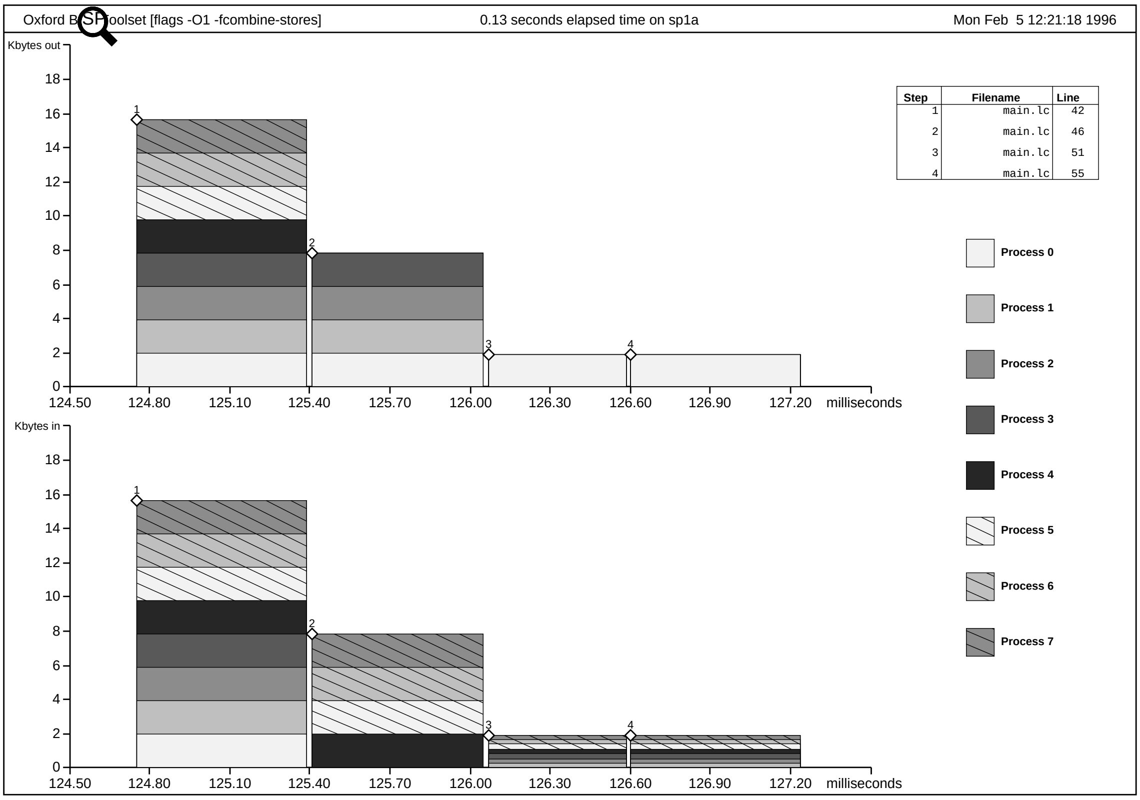


Figure 2: Four 500-relation supersteps

superstep 3 every process *gets* $500/7 = 71$ integers from process zero. The communication pattern can be considered as a broadcast from process zero, where $p-1$ messages will leave that process (we do not count the internal memory copy of the broadcast data on process zero when costing the h-relation).

superstep 4 a broadcast implemented as process zero *put*ting data into every other process.

The top and bottom graphs in figure 2 show the number of bytes leaving and entering each process on the y-axis, with respect to time on the x-axis. Each pair of columns that are vertically aligned represents the number of bytes leaving and entering a process during the communication phase of a superstep (i.e., the time taken in the block of code between `begin_superstep` and `end_superstep`). The label found at the top left-hand corner of each column can be used in conjunction with the legend at the top right-hand side of the profile to identify the superstep being executed. The white space between the columns represents the elapsed time of the process that takes the longest time to perform it's local computation phase of a superstep. In this case it is negligible.

Each column in Figure 2 consists of a series of bands where the height of a band represents the amount of data communicated by the process identified by the band's shade. The sum of all the bands (i.e., the height of the column) represents the total communication across all processors for a superstep. The width represents the total time spent in *both* communication and bulk synchronisation (i.e., $l + hg$); where the duration of the superstep is defined as the time between the last process to pass through the `begin_superstep` and the first process to exit `end_superstep`. As can be seen in figure 2, the width of each column is approximately the same. This is expected as each superstep realises a 500-relation. However, the differing heights of each column reflect the difference in aggregated communication for each superstep. For example, in the first superstep, as one message leaves and enters each process then the communication realises a 500-relation. However, the total amount of data transferred between all processors is $ph = 8 \times 500 \times 4 = 15.6 Kbytes$.

In contrast to the large column produced by the first superstep, only 2Kbytes of data is exchanged between all processors in the last superstep. In this situation the h-relation is captured by the fact that 500 words leave process zero.

Comparing the theoretical cost of the superstep with the experimental cost is favourable. As each superstep realises a 500-relation, it's cost in seconds is $(500g + l)/s$. The overhead of the profiling system is an additional 20% to the synchronisation time. Therefore using the BSP parameters in Table 1 gives an expected runtime of $534\mu sec$. This compares with actual runtimes of $636\mu sec$ achieved in supersteps 1,2 and 4; and $516\mu sec$ for superstep 3.

The role of this output from the profiling tool is to expose two pieces of information: (1) the size of a h-relation is identified by the thickest band in either the "in" or "out" columns; (2) the communication pattern.

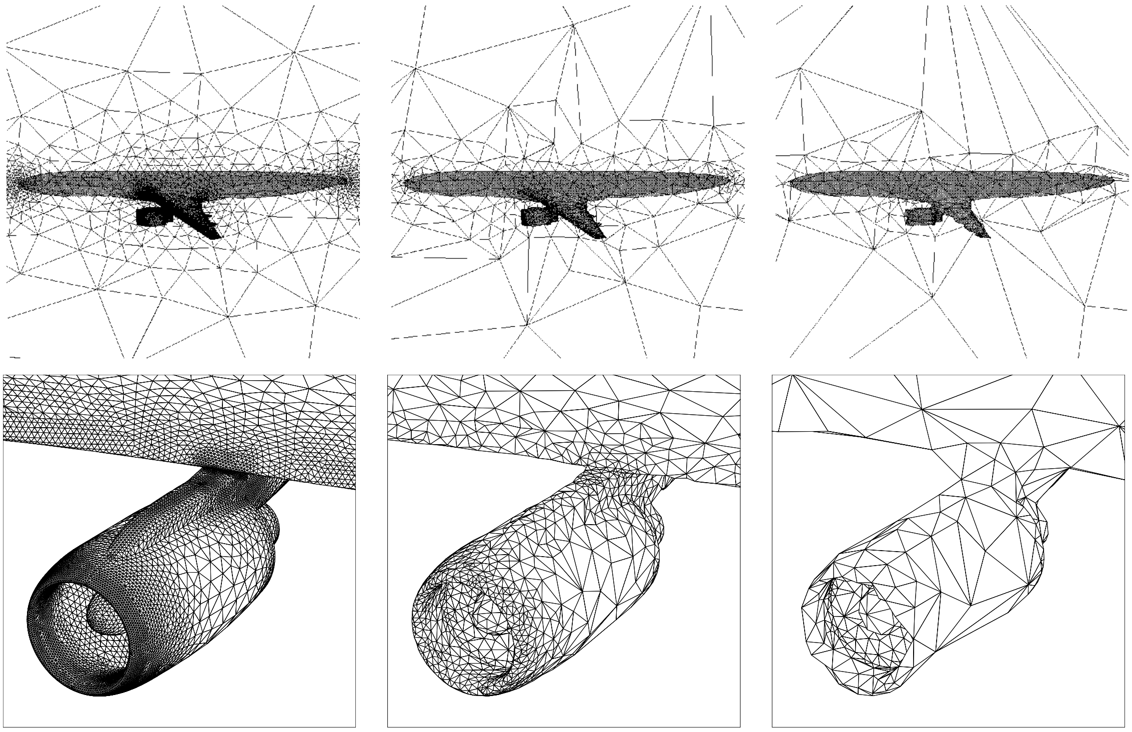
5 Multigrid CFD application

It is impossible to choose one representative application, or even a group of applications, to demonstrate the effectiveness of the BSP approach. Here, a single application is considered, this models the inviscid flow over an aircraft [3] on an unstructured tetrahedral mesh using an edge collapsing multigrid technique [4]. This application is chosen because:

- it represents a complex algorithm mapped onto a parallel machine that is not trivially parallelised,
- it produces a varied pattern of communication to test the validity of the BSP cost model,
- it is an industrially relevant application, not a model problem.

The following text introduces the underlying concepts behind the algorithm and derives a simple cost model. This is by no means meant as a comprehensive discussion of how a multigrid method can be mapped onto a parallel machine, but is an example of how to construct such a model, and show the practical use of the model.

The underlying method uses a 3D unstructured tetrahedral grid that tessellates the domain around an aircraft. The surface triangulations are shown in Figure 3. The two main components of the algorithm are a "smoother" and a sequence of successively coarser grids as shown in Figure 3. The smoother is an explicit iterative process that converges to a fixed point. The main computational cost is a loop over the tetrahedra, which consists of: (1) gathering information from the four nodes of the tetrahedra; (2) performing a large quantity of arithmetic; (3) scattering information from the tetrahedra to the nodes. This "smoothing" iteration can be iterated many thousands of times on a given fine mesh in order to converge to the solution. However, this is inefficient. In order to accelerate the convergence we exploit the fact that this "smoothing" iteration damps all high frequency errors very quickly, thus leaving only low frequency errors that can be represented on a coarser grid. Applying this



Grid 1 Grid 2 Grid 3

Figure 3: Sequence of grids for aircraft configuration

idea recursively leads to the multigrid method, were all modes are damped. For certain classes of problems multigrid is an optimal method (the work required to solve a problem of size N cells is $\mathcal{O}(\mathcal{N})$). There are many different strategies for cycling through a given grid sequence; the one considered here is the so called W-cycle, which is schematically represented in Figure 4. Cycling through multiple grids, each of which is stored as a list of indirect addresses, significantly complicates the algorithm in comparison to performing smoothing iterations on the finest mesh. However, the numerical efficiency benefits more than compensate the added implementation complexity.

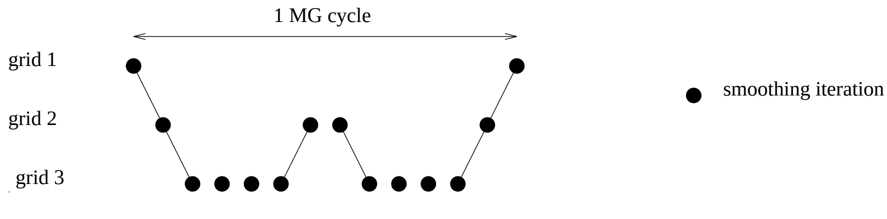
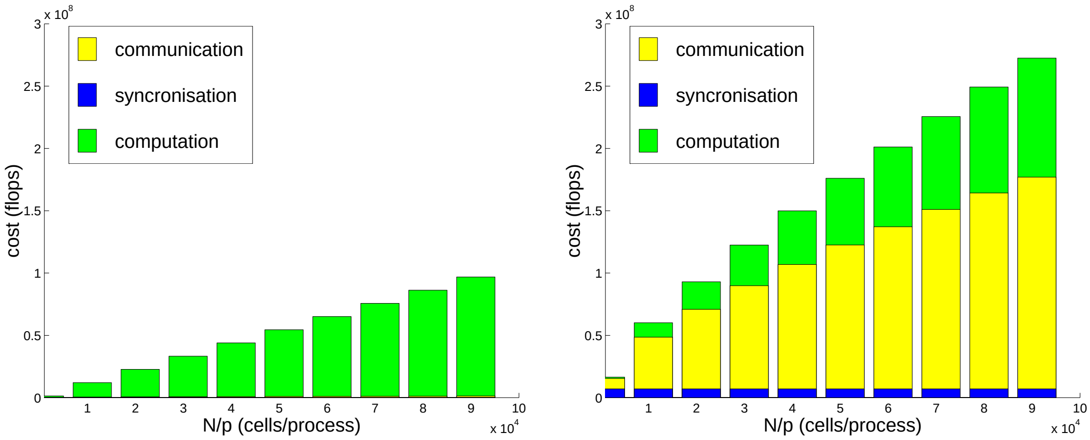


Figure 4: Schematic of a W-cycle

In this study the computational cost of the smoother (in flops) is assumed to be proportional to the number of cells N within a particular tetrahedral grid, i.e., the cost is $C_1 N$, where C_1 is the flop count per cell for a smoothing iteration. Thus the computational cost of smoothing on the finest mesh within a W-cycle is $2C_1 N$ (for the smoothing at the beginning and end of the cycle, see Figure 4), and the cost of smoothing the first coarse grid (grid 2) is $4C_1Nr$ (where r is the reduction fraction describing the fewer cells on the coarser grid). Thus on a sequential machine the computational cost can be described,

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



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

Figure 5: Cost model predictions using equation 5

To execute this in parallel the grids are partitioned and placed on the processors. Consequently each processor "owns" a piece of the domain on each grid, and is responsible for communicating partition surface information with neighbours to ensure the parallel execution is consistent with sequential execution. Lets assume that for p processors each process receives (N/p) cells on the finest grid and each partition has an exterior surface of $(N/p)^\gamma$ cells, where $\gamma < 1$. This is proportional to the amount of data exchange required for parallel execution. The parallel smoothing cost can thus be expressed

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

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 needed per smoothing iteration. The cost of a W-cycle can be then written,

$$\textit{parallel W cycle cost} = \sum_{i=1}^{N_{grids}} 2^i \left[C_1\left(\frac{Nr^{i-1}}{p}\right) + C_2\left(\frac{Nr^{i-1}}{p}\right)^\gamma g + C_3 l\right] \qquad (5)$$

It is worth mentioning that for good parallel efficiency the local computation in a superstep must dominate. i.e., $C_1(\frac{Nr^{i-1}}{p})/C_2 g(\frac{Nr^{i-1}}{p}) \gg 1$. However, on the coarser grids this ratio is scaled by $r^{1-\gamma}$ (typically $r = 1/8$, $\gamma = 2/3$, so $r^{1-\gamma} = 1/2$). Consequently the coarse grids smoothing iteration may become communication bound. This difficulty is compounded by the choice of the W-cycle, where many smoothing iterations are performed on the coarsest grid, see Figure 4. Equation 5 represents a rather simplistic model, and was derived by way of example rather than a definitive statement about how multigrid will execute on a parallel machine. However, it has all the leading order cost terms and so will be asymptotically correct. It is worth reiterating the assumptions made in this model:

1. The tetrahedra have been partitioning perfectly on all the grids. This is certainly not true in reality as coarse grid partitions are "inherited" from fine grid partitions [5]. The resulting load imbalance could easily be accounted for in the model.

2. The computational cost of the transfer operations within the multigrid cycle have been ignored. This again could be accounted for but is thought to be negligible. Observations from using the BSP tool confirm this.

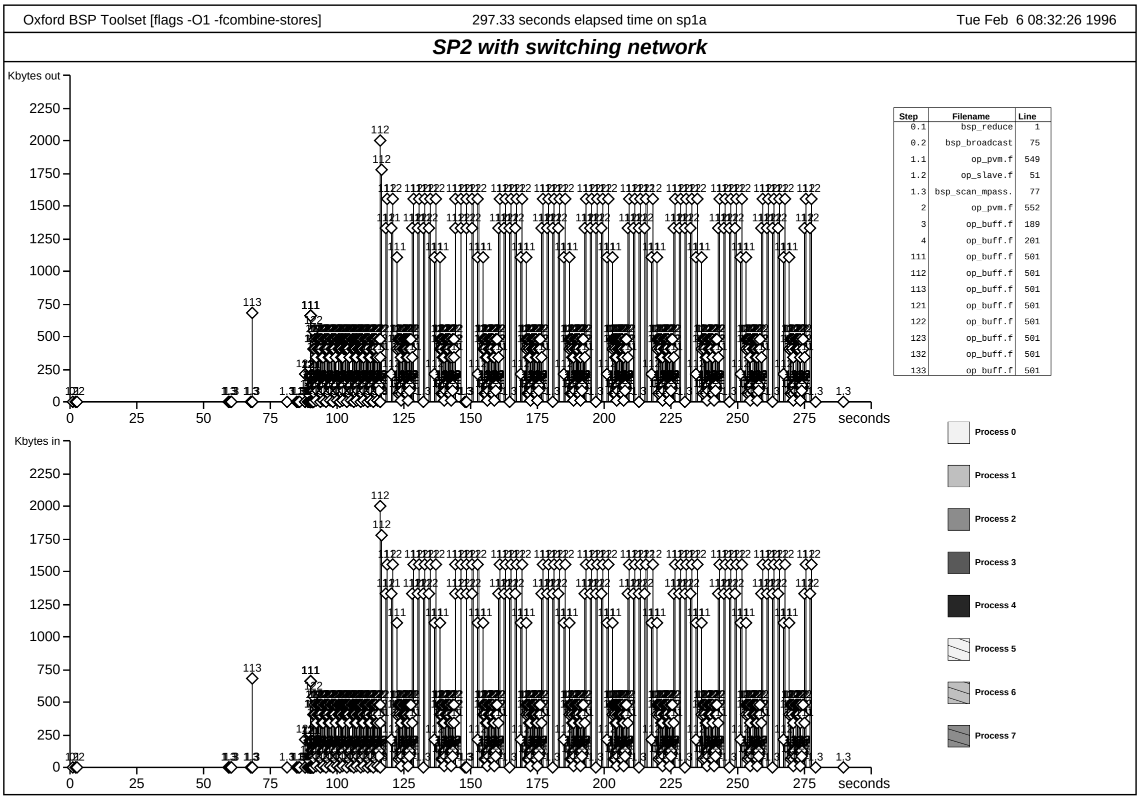
3. Some technical aspects of the multigrid scheme have been ignored. They could easily be incorporated, but are beyond the scope of this paper.

4. Only the computational cost over cells have been accounted for. However, the execution of boundary conditions, especially at the engine inlet (see figure 3), can be expensive.

Because of the above we *expect the cost model to be optimistic*, but asymptotically correct. It is worth noting that converting the sequential multigrid cost model to a BSP parallel cost model is a straightforward task. The following paragraphs will hopefully demonstrate the useful information that can be extracted from such a model. In general the accuracy of the model depends on how much information about the particular application is known. For instance, if a sequential code already exists then by profiling the code on a given machine, and identifying the CPU hot spots a very accurate sequential cost model can be produced. If data partitioning information is also known (there are a plethora of these methods publicly available), then the parallel BSP cost model can be refined. The model introduced here is rather crude, but still gives good qualitative information.

The parameters of the model are now chosen to approximate a multigrid cycle on the sequence of grids shown in Figure 3. These are $N = 7.5 \times 10^5$, $r = 1/8$, $N_{grids} = 3$, and for the smoother the constants are estimated to be $C_1 = 400$, $C_2 = 15$, $C_3 = 2$. In this study, execution is contrasted between the IBM SP2 configured to use either ethernet or a high-performance switch. Since the values of l and g for these machine configurations are not independent of p, we take the worst case values for the largest machine configurations, i.e., switch l, g=(8340,8), ethernet l, g=(223120,1645).

Figure 5 plots N/p against cost with switch and ethernet respectively. The blocks within the figure show accumulated synchronisation, communication and computation costs. If a high parallel efficiency is required then the computational cost must dominate. The right hand-side of these plots represent the current problem on eight processors. i.e., $N/p = 9.3 \times 10^4$. Clearly eight processors will be ineffective if ethernet communication is used, with most of the time spent in communication. In comparison the synchronisation and communication costs for the SP2 using the switch are negligible and are not visible in the graph. Using ethernet we estimate that $N/p \approx 10^6$, i.e., 10^6 fine grid tetrahedra need to be placed on each processor for effective parallel execution, whereas for the switch communication only 10^2 will be needed per processor. This would be invaluable information if a prospective user looking to purchase a parallel machine, who presumably would have some idea of the problem size.



Step	Filename	Line
0.1	bsp_reduce	1
0.2	bsp_broadcast	75
1.1	op_pvm.f	549
1.2	op_slave.f	51
1.3	bsp_scan_mpass.	77
2	op_pvm.f	552
3	op_buff.f	189
4	op_buff.f	201
111	op_buff.f	501
112	op_buff.f	501
113	op_buff.f	501
121	op_buff.f	501
122	op_buff.f	501
123	op_buff.f	501
132	op_buff.f	501
133	op_buff.f	501

Figure 6: CFD application on a 8 processor SP2 (using switch)

In 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 (i.e, what happens as p becomes large) can also be studied using this technique. This approach is by no means revolutionary, however, the simplicity, portability and accuracy of the model with BSP programs gives an application programmer a significant advantage over message passing.

The implementation of this application was carried out in FORTRAN 77 using the OPlus library [2], which uses the BSP toolset for communication. The OPlus library removes the parallelisation burden from the application programmer by handling data partitioning, all I/O and the organisation of the data transfers.

Figure 6 is the output from the BSP profiling tool executing 10 W-cycles for the aircraft problem on an 8 node SP2 using the switch communication. The white space at the beginning of the plot accounts for the time taken in I/O, data partitioning and distribution. In practice 100 iterations would be required for numerical convergence of the algorithm, and thus the startup costs are acceptable. We only profile 10 iterations as the pictures become rather repetitive otherwise. A clear pattern of communication is evident. The actual pattern can only be assessed by zooming into a W-cycle shown in Figure 7. It is worth noticing the close resemblance between this profile and the schematic of the W-cycle Figure 4, with large computational gaps between large messages on the fine mesh, and short computational gaps and short messages on the coarse grids. It is clear the computation dominates the overall time, as the BSP model predicted for the SP2 using the switch as shown in figure 5(a).

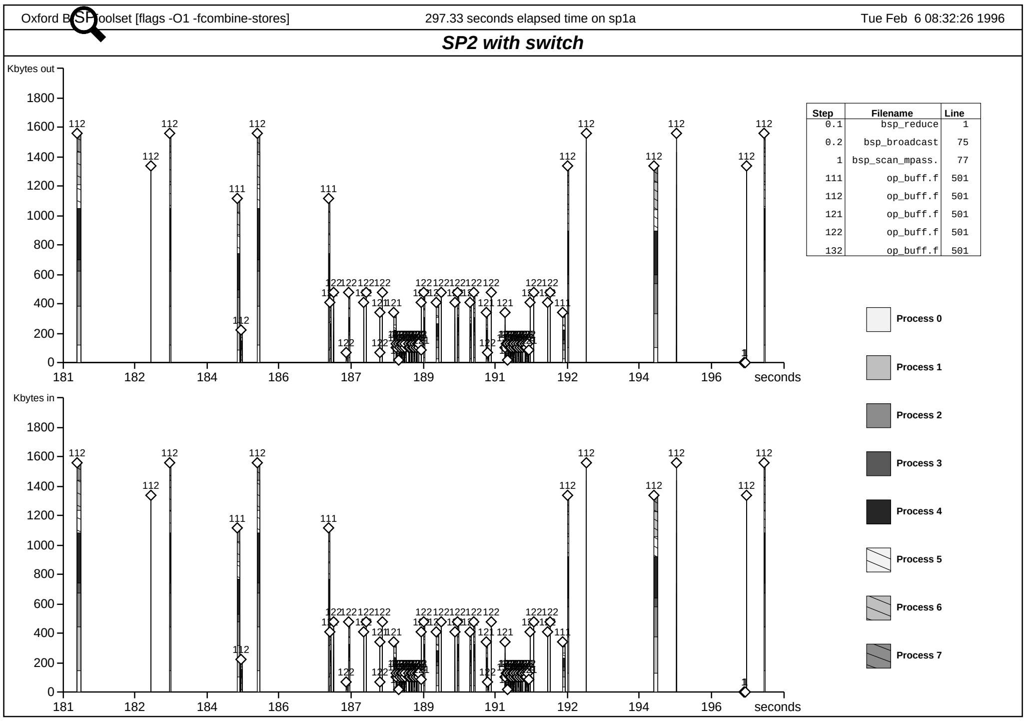


Figure 7: Zooming into a "W-cycle" (using switch)

The objective of BSP programming is to tune for good performance on a hypothetical BSP computer. Thus enabling predictive performance across a wide variety of machines. Figure 8 shows a *prediction* profile. The idea is for each superstep, as the size of the h-relation can be calculated from the profiled data, then instead of the width of the column being the actual time taken to perform the communication and synchronisation, the width is changed to be proportional to $l + hg$. This predicted performance, which would be obtained if the bulk parameters were exactly matched, superstep by superstep, by the performance of the real system, gives the profile for the calculation as it would run on a hypothetical BSP machine with the same BSP parameters as the SP2. Very wide discrepancies between these plots would indicate a problem worthy of further investigation but the main objective would be to tune the calculation to secure good performance on the BSP machine. In figure 8 the predicted cost is compared against the actual cost of the W-cycle. The match between practice and prediction is close, although there a few discrepancies where some supersteps take longer than expected.

A major advantage of the tool is to predict the performance of the code on other parallel machines. This is simply achieved by plugging in the BSP parameters for the machine we wish to predict the performance of. Figure 9 shows the result of this exercise when the profile data generated from the run of the code using the SP2 and switch, is used to predict the performance of the code if ethernet where used. The top graph shows the actual cost on the SP2 with switch, the lower graph predicted cost on a SP2 with ethernet. Normally the reverse process would be performed when the cost of a code running on-top of a network of workstations is used to predict the cost of the code on a *real* parallel machine. The tool predicts that the W-cycle will take $26625 - 23275 = 3350 Mflops$, which is equivalent to 128

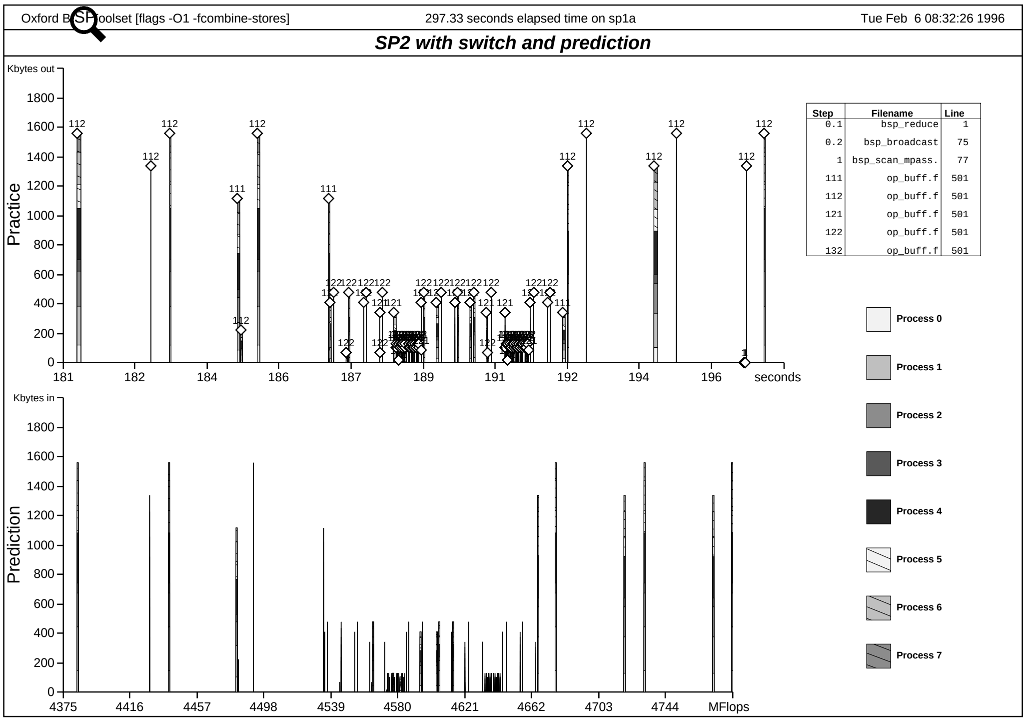


Figure 8: Comparing the actual cost of a "W-cycle" with the predicted cost

seconds on the SP2. If all our claims of automatic predictability sound too good to be true, figure 10 shows the actual cost of 101 seconds for the W-cycle using ethernet communication. The theoretical model shown in figure 5 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. In contrast, the tool makes fewer assumptions as it bases it prediction on actual communication patterns realised during a trace run. The tool predicts that ethernet will be a factor seven (i.e, $128/17 = 7.5$) slower than communication over the switch. The actual experimental ratio is $101/17 = 5.9$.

In the predicted plot in Figure 9, because of the slow communication, the amount of data entering each processor is clearly visable. It is reassuring to note that the total amount of data is evenly spread among the eight processors, indicating that the data-partioning of grids is adequate for this problem.

Similar plots have been collated on the SGI PowerChallenge, which because l and g are even lower than the SP2 with switch communication, show computation dominance for this application. These are not included as they do not further exemplify the BSP approach, other than the machine architecture is very different, but the same code runs on it efficiently.

6 Conclusions

In a perfect world programmers would develop cost models for applications before coding commences. Unfortunately in the real world, people rarely do such things. At best the

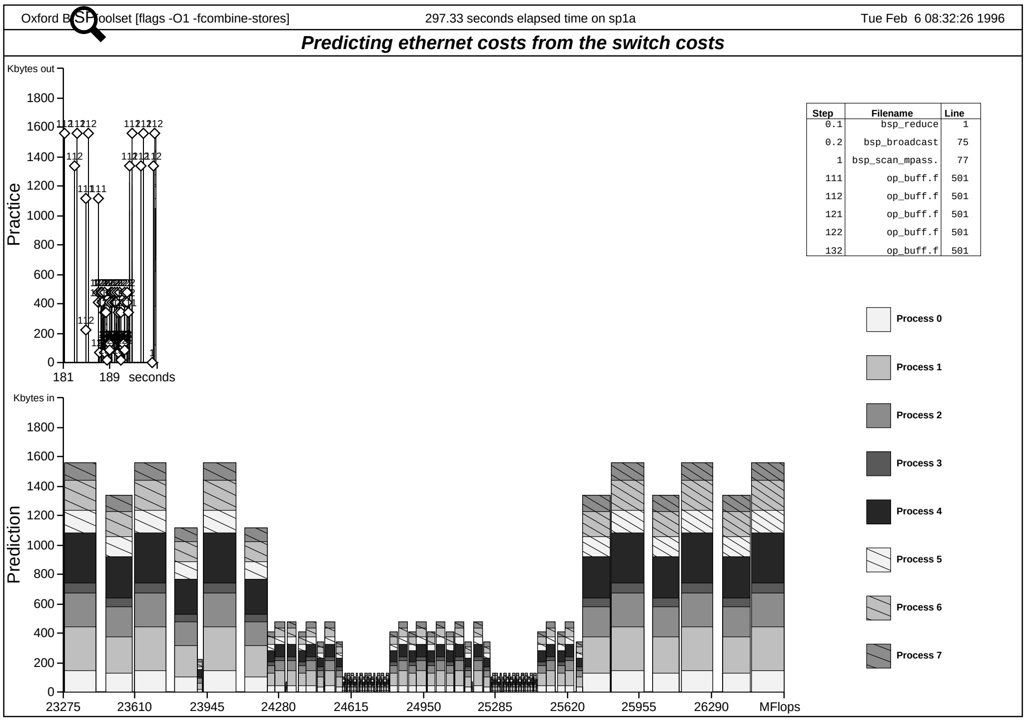


Figure 9: Fortune telling: predicting the cost on ethernet from data gathered on the switch

asymptotic complexity of a program may be considered before coding.

The application we have described was initially developed using the PVM system. With hindsight we realise that the performance of that implementation, on a wide variety of parallel machines, has overly expensive communication costs. The BSP implementation and cost model for the application were developed later. This gave *significantly* faster performance than the PVM implementation. Moreover the BSP implementation has a simple cost model, is deadlock-free, and is as portable as the PVM system. In fact, the communication is so efficient that it is almost negligible in this realistic CFD application. The problem is now compute bound and needs to be run on a larger number or faster processors.

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. Usually assumptions are made to simplify costing algorithms. The profiling tool makes no assumptions. The tool can effectively predict the cost of the algorithm on any parallel machine. Unfortunately, the tool doesn't have the facility to predict the cost on differing numbers of processors. Pen and paper cost modeling still has a place in BSP cost analysis—albeit smaller now!

Acknowledgements

This work was performed within Oxford Parallel with financial support from Rolls–Royce plc and EPSRC. We gratefully acknowledge the use of the unstructured grid generator of Jaime Peraire and Joaquim Peiro. We would also like to thank Bill McColl, David Skillicorn, and Bob McLatchie for their advice and encouragement.

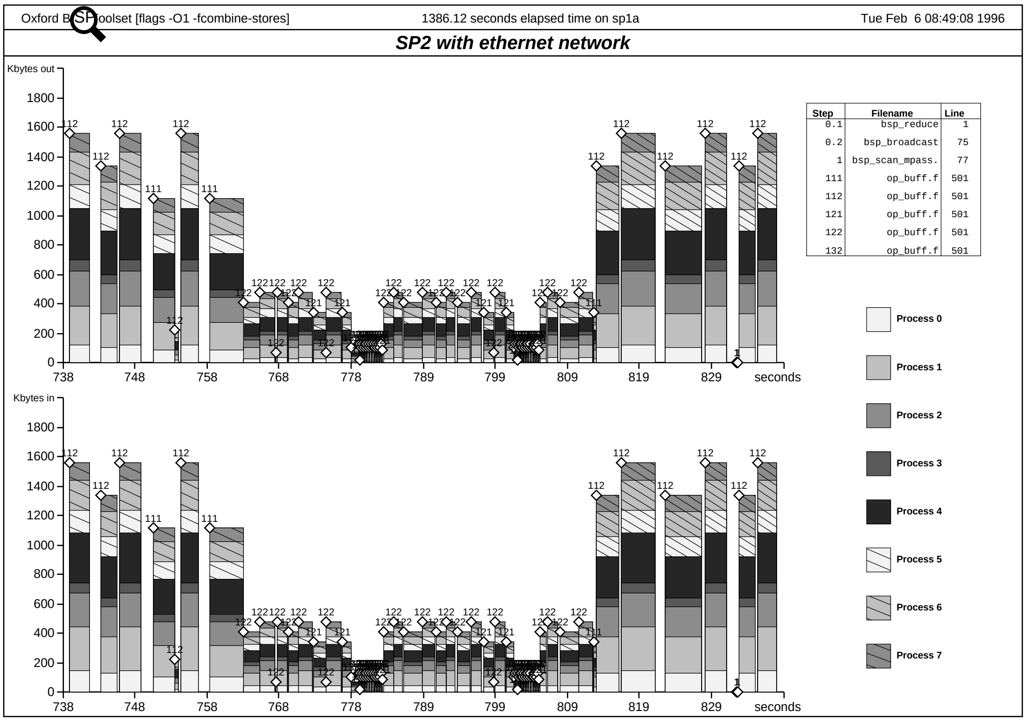


Figure 10: Observed performance of a "W-cycle" over ethernet

References

[1] T. Agerwala, J. L. Martin, J. H. Mirze, D. C. Sadler, D. M. Dias, and M. Snir. SP2 system architecture. *IBM Systems Journal*, 34(2):152–184, 1995.

[2] D. Burgess, P. Crumpton, and M. Giles. A parallel framework for unstructured grid solvers. In S. Wagner, E. Hirschel, J. Périaux, and R. Piva, editors, *Computational Fluid Dynamics '94. Proceedings of the Second Europ ean Computational Fluid Dynamics Conference 5-8 September 1994 Stuttga rt, Germany*, pages 391–396. John Wiley & Sons, 1994.

[3] P. Crumpton and M. Giles. OPlus programmer's guide, rev. 1.0. Oxford University Computing Laboratory, 1993.

[4] P. Crumpton and M. Giles. Implicit time accurate solutions on unstructured dynamic grids. AIAA Paper 95-1671, 1995.

[5] P. Crumpton and R. Haimes. Parallel visualisation of unstructured grids. In S. Taylor, A. Ecer, J. Periaux, and N. Satofuka, editors, *Proceedings of Parallel CFD'95, Pasadena, CA, USA 26–29 June*, 1995.

[6] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. v. Eicken. Logp: Towards a realistic model of parallel computation. In *Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Diego, CA, May 1993.

[7] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM 3 Users Guide and Reference manual.* Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, May 94.

[8] J. M. D. Hill and W. F. McColl. An initial proposal for the BSP Worldwide standard library. Technical report, Oxford University Computing Laboratory, January 1996.

[9] R. W. Hockney. Performance parameters and benchmarking of supercomputers. *Parallel Computing*, 17:1111–1130, 1991.

[10] W. F. McColl. Scalable computing. In J. van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, number 1000 in LNCS, pages 46–61. Springer-Verlag, 1995.

[11] Message Passing Interface Forum. MPI: A message passing interface. In *Proc. Supercomputing '93*, pages 878–883. IEEE Computer Society, 1993.

[12] R. Miller. A library for Bulk Synchronous Parallel programming. In *Proceedings of the BCS Parallel Processing Specialist Group workshop on General Purpose Parallel Computing*, pages 100–108, December 1993.

[13] R. Miller. *Two approaches to architecture-independent parallel computation.* D.Phil thesis, Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, Michaelmas Term 1994.

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