Communicating Process Architectures 2004
Ian East, Jeremy Martin, Peter Welch, David Duce, and Mark Green (Eds.)
IOS Press, 2004

Dynamic BSP: Towards a Flexible Approach to Parallel Computing over the Grid

Jeremy M. R. MARTIN
Oxagen Limited, 91 Milton Park, Abingdon, Oxon OX14 4RY, UK
Alexander V. TISKIN
Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK

Abstract. The Bulk Synchronous model of parallel programming has proved to be a successful paradigm for developing portable, scalable, high performance software. Originally developed for use with traditional supercomputers, it was later applied to networks of workstations. Following the emergence of grid computing, new programming models are needed to exploit its potential. We consider the main issues relating to adapting BSP for this purpose, and propose a new model *Dynamic BSP*, which brings together many elements from previous work in order to deal with quality-of-service and heterogeneity issues. Our approach uses a task-farmed implementation of supersteps.

1 Introduction

The BSP parallel computation model [1, 2] is very simple. It assumes that there is a set of p processors, each capable of performing s operations per second. The processors are connected by a communication fabric, which can communicate one data item to or from every processor in the time it takes each processor to perform g floating-point operations. It can also perform a global handshake synchronisation of all the processors in the time it takes each processor to perform l floating-point operations.

The BSP programming model is also very simple. Execution of a BSP program is divided into *supersteps*, each separated by global synchronisations. A superstep consists of each processor doing some calculation on local data *and/or* communicating some data by direct memory transfer to other processors. The global synchonisation event guarantees that all communication of data has completed before the commencement of the next superstep.

Perhaps the most useful feature of BSP is the ability to construct *cost functions* of the parameters (s, p, l, g) in order to predict the performance and scalability of parallel algorithms across different hardware platforms. This can be done prior to implementation. Tables of approximate values for these parameters are available for a wide range of machines [3, 2]. There have been a number of successful case-studies of using BSP in practice (see for instance [4]).

BSP was originally intended for use within a reliable, homogeneous, dedicated parallel computing environment, rather than with the unpredictable and variable resources that are associated with grid computing [5]. However, it is such an attractive model for programmers, that it is surely worthy of consideration as to whether it can be adapted for use in the grid environment.

2 Previous Work

There are three main areas where the grid deviates from the BSP model.

1. *Processor heterogeneity*: variation between grid nodes of available computation power, either due to architectural differences, or due to time-dependent resource sharing issues. If a BSP program were run on a heterogeneous cluster, the progress of the overall computation would be constrained by the rate of the slowest processor. In some cases, we could consider getting around this problem by using sophisticated domain decomposition techniques to achieve better load balancing. However, this would make both programming and cost modelling substantially harder, which is against the spirit of the original BSP philosophy.

2. *Network heterogeneity*: significant variation of communication performance between nodes. Previous work has suggested that progress of a BSP program is usually constrained by the slowest communication link in the network [6].

3. *Reliability and availability*: processors may fail intermittently or be withdrawn unexpectedly by the service provider. This may lead to a variation in processor count during execution.

Let us consider existing published work in this field and assess where it has attempted to deal with the above issues.

Vasilev [7] has developed the BSPGRID model for grid-based parallel algorithms, by extending the BSPRAM model of Tiskin [8]. A BSPGRID computer is a collection of processor-memory units, a shared memory considered to be of unlimited capacity (which is likely to be implemented as a collection of disk units), and a global synchronisation mechanism. Unlike the standard BSP model, there is no persistence of data at processor nodes between supersteps — the contents of all local memories are discarded at the end of each superstep.

The amount of memory at each processor is considered to be limited, so for large problem sizes where the total amount of available memory at the processor nodes is insufficient, the concept of virtual processors is used. This means that each physical processor may be required to perform the work of multiple virtual processors sequentially in a particular superstep. The issue of processor reliability and availability is addressed by allowing the number of available physical processors to vary between supersteps. There is a recovery protocol for the case when processors may fail unexpectedly during a superstep: an additional synchronisation barrier is introduced, and the work of failed processors is rescheduled after the barrier.

A centralised global shared memory would lead to a communication bottleneck at the master processor, therefore any implementation of BSPGRID is likely to implement virtual shared memory distributed over the grid. Such an implementation would need to be made easy to use and fault-tolerant.

BSPGRID has a cost model consisting of the following parameters:

- M — the amount of memory per processor in words;
- g — the cost of shared memory accesses per word;
- l — the cost of synchronisation;
- N — the problem size in words.

The model is used to predict two cost functions for an algorithm: time and work. The time cost of an algorithm is the optimal execution time that could be achieved if enough real processors are applied to the problem, whereas the work cost is the processor-time product of the algorithm. The model does not consider the issues of network and processor heterogeneity, apart from variation in the amount of available memory at each node. A static and uniform allocation of virtual processors to physical processors is performed at each superstep.

Work by Goldschleger et al. [9, 10] describes development of a grid middleware infrastructure InteGrade, and implementation of BSP using that system. It is particularly focussed on provision of a virtual BSP computer to a user by allocation of idle resources within an organisation. There is no treatment of heterogeneity, but it is reported that work is in progress to support fault tolerance via a checkpoint and recovery protocol. The system is based on the Oxford BSP toolkit [11], and is claimed by the authors to be the first grid implementation of BSP.

Mattsson and Kessler [12] describe implementation of a BSP-based virtual shared memory programming language for grid computing. It supports a hierarchical extension to the BSP paradigm with localised supersteps. Other hierarchical extensions of BSP and a discussion of the localised approach can be found in various sources in the literature, including earlier work by the present authors [6, 13].

Nibhanupudi and Szymanski [14] have developed a fault-tolerant version of BSP, which works by running multiple redundant peers for each BSP processor. The peers are able to take over whenever the original process is assumed to have failed. They have developed a complex arbitration protocol to manage this, since it is hard to detect that a process has failed and is not merely slow.

The Satin system by van Nieuwpoort et al. [15] allows dynamic processes in the form of "pure" (side-effect free) function calls, scheduled by work stealing. Our proposed approach is more flexible, allowing more general process types and scheduling strategies. Tiskin [13] proposed a mechanism for dynamic process management in the BSP model, using both SPMD parallelism and dynamic processes. Our approach is conceptually simpler and easier to program, since it involves only one of these parallelism types (dynamic processes).

Rosenberg, Adler and Gong [16, 17] have investigated in depth the matter of optimally scheduling a bag of similar tasks to a heterogeneous network of processors. They have compared mathematically the predicted performance of using a FIFO communication protocol with that of a BSP communication protocol, in order to distribute tasks and gather results. This work is highly relevant to optimisation of the implementation of a superstep within a model such as BSPGRID.

3 Towards a New Approach: Dynamic BSP

Although there has clearly been substantial progress concerning grid implementation of BSP, we have seen that there is no single approach which would address all of our major concerns. Here we present a significant modification to the BSPGRID approach, which will enable us to address the heterogeneity issues, as well as fault-tolerance. It will also offer us a more flexible programming model, with the ability to spawn additional processes within supersteps as and when required.

There has been considerable success in utilising the internet to solve embarrassingly parallel problems using task-farms, for instance by the application of screen savers performing drug-protein docking simulations on vast numbers of personal computers [18]. Task-farming has also been proposed as a general programming paradigm for grid computing, e.g. in [19].

The essence of our new approach is to use the task-farm model to implement BSP supersteps, where the individual tasks correspond to virtual processors (see Figure 1). A task-

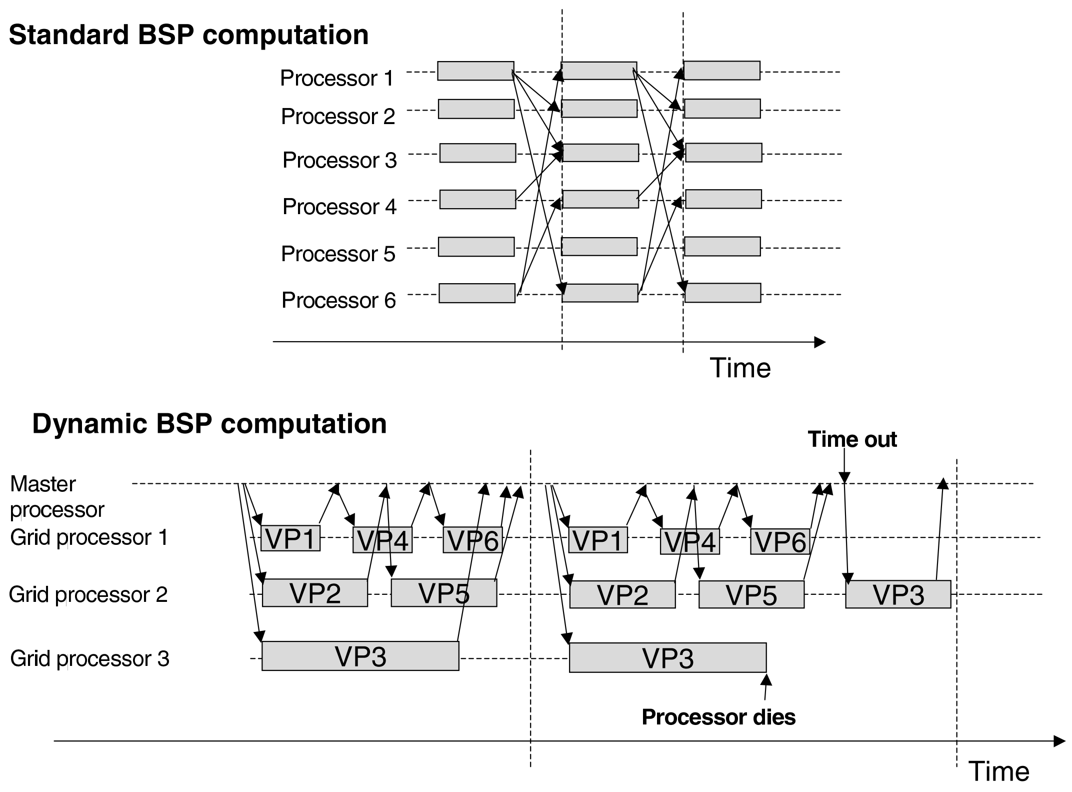

Figure 1: Standard BSP compared with Dynamic BSP

farm implementation of BSP has been suggested previously by Sarmenta [20], whose paper recognises that this approach suffers from the data bottleneck, unless the computation is embarrassingly parallel. We propose a mechanism of avoiding the data bottleneck.

Our model consists of:

- a master processor (task server);
- worker processors; and
- a data server (which can either be implemented as distributed shared memory or remote/external memory).

In each superstep there is a bag of virtual processors to be run on a pool of available physical processors. The computation and communication performance of each processor is considered to be variable, but we do assume a certain minimum level of available memory at each node, as with BSPGRID.

The master processor is responsible for task scheduling, memory management, and resource management. At the beginning of each superstep, a virtual processor number is distributed to each physical processor, which then has the responsibility to retrieve local data from the data server, perform the required computations, write back the modified data, and then inform the master processor that it has finished the task. The master processor maintains a queue of pending virtual processors and dynamically assigns them to waiting physical processors. As soon as the all the virtual processors have been executed on a particular superstep, the global shared memory is restored to a consistent state and the next superstep commences.

Using the task farm approach, the problem of heterogeneity across the grid can be concealed by choosing the number of virtual processors to far exceed the number of physical processors. This approach is sometimes known as *parallel slackness*, and complements explicit heterogeneous extensions to the BSP model, such as the ones by Williams and Parsons [21], and Morin [22].

3.1 Fault Tolerance

If a physical processor fails to complete its task within a reasonable time, then it is considered to have died, and its work is reallocated to another physical processor *within the same superstep*. Also the master process is at liberty to seek additional resources at any point to expand the processor pool. This approach to fault-tolerance is likely to be less computationally expensive than the generic Unix process migration mechanism proposed by Hill et al. [23].

3.2 Creation of Child Processes

Dynamic BSP allows the number of virtual processors to vary not only between supersteps but also *during* supersteps. Hence we may allow them to spawn other virtual processors (which would be useful for example to implement divide-and-conquer algorithms). Since the master processor still has to keep control, a virtual processor has to send a message to the master to spawn one or more children, and is then descheduled. The master will reschedule the requesting processor once all its children have terminated. There can be no data redistribution within supersteps, therefore the new virtual processor can only see a snapshot of the global data as it was at the beginning of the superstep, together with local state inherited from its parent. A traditional superstep would only allow one level of spawning, but within a task farm implementation arbitrarily many levels of descendant processors can be spawned, as long as data redistribution is not required.

The master processor generates tasks (virtual processors) and can either execute them (if they are small enough), or pass them on to workers. Workers can spawn new tasks, which must be registered with the master. Tasks may contain remote data references, and can transmit data to and from the data server. Additionally, sometimes it may be convenient that the data server can perform by request simple data-parallel computations without passing the data to the master or workers.

3.3 Memory Bottleneck

The main barrier to scalability with both BSPGRID and Dynamic BSP is the implementation of the global shared memory. To avoid a bottleneck, the data would need be distributed and treated similarly to BSPRAM model. Virtual processors would be decoupled from their data, but they would also need a mechanism of knowing where their data is (potentially distributed across several physical processors), and be able to access it bypassing the master processor. There would also need to be a separate fault-tolerance mechanism for the data, requiring replication and/or check-pointing such as implemented by Nibhanupudi and Szymanski [14].

3.4 Cost Model

The standard cost model for BSP would appear to be suitable for dynamic BSP, despite the fact that the g and l parameters might very well vary significantly between grid nodes. Using the task-farm approach, together with use of parallel slackness, would make it reasonable to use measured values for g and l (suitably averaged) to predict cost.

3.5 An Example: Strassen's Algorithm

McColl [24] (see also [25]) proposed a synchronisation-efficient BSP Strassen matrix multiplication algorithm, which generates block multiplication subtasks recursively in a data-parallel fashion. When the number of tasks becomes sufficiently high (equal to the number of physical processors, or more if one needs parallel slackness), matrix data are redistributed and the computation is completed in task-parallel fashion.

In our model, the master will generate the first “root” task, which will request the data server to do data-parallel work (without communication), and then spawn some children tasks. The children will do the same recursively. All this can be done in the master processor, since the tasks do not need to download data from the data server, hence their cost is at this point negligible. When the number of spawned tasks is large enough, they are distributed across the workers, and enter the task-parallel phase of the computation (i.e. download the matrix data, synchronise, compute block products, upload them back to the data server, and synchronise again). As soon as the task-parallel part of the computation is finished, the workers can send the tasks back to the master. The children tasks now start to terminate, and the parents resume and combine the childrens’ results by issuing data-parallel computation requests to the data server. Upon termination of the root task, the data server contains the final output.

4 Conclusions and Future Work

We have investigated the potential issues in implementation of the BSP model over the grid. We have reviewed some useful existing work in this context, and also proposed a new dynamic model. Our model builds on Vasilev’s BSPGRID model, and retains some of its key elements: a dynamic task pool and a virtual shared memory. In contrast with BSPGRID, our model utilises the task-farm approach to implement supersteps, and allows the tasks to spawn an arbitrary number of subtasks within a superstep. We have also introduced into the model some capability of data-parallel computation, which can be performed by the tasks remotely via the data server. The flexibility of this approach has been demonstrated by the Strassen matrix multiplication example. Since computational tasks are decoupled from data, both BSPGRID and our approach require an efficient implementation of distributed/remore shared memory for the data server.

The next logical step is implementing our model and testing it on a real-life system. In addition to that, here are some issues that would be worthy of further consideration.

Improved fault-tolerance. Instead of timeouts, it might be possible to use more sensitive mechanisms, such as ping clients.

Security. Commercial enterprises have to be extremely careful about data security, and this tends to be the main barrier to the uptake of grid computing in industry. A BSP implementation offering data encryption could help to solve this problem.

Persistent data. Many computationally intensive algorithms in use today require local access to substantial databases, e.g. the Blast program for biological sequence similarity analysis [26]. We need to consider extending our protocols to allow external resources to be requested with special attributes such as this.

Economics. In real-life grid applications, there is likely to be a tradeoff between quality of service and price. Commercial service providers, such as [27], offer use of dedicated reliable homogeneous virtual clusters for a competitive fee, whereas it is also possible to harness idle cycles on workstations within an organisation at very little cost. Work [28] contains an analysis of some of the economic factors involved in running a supercomputing service. It could be possible to take these factors into account by making financial cost an integral part of the cost model. Vasilev’s BSPGRID makes the first step in this direction: the time cost of a computation is appropriate for evaluation of an algorithm implemented using inexpensive idle cycles, whereas the work cost is more relevant to time hired on commercial processor warehouses.

Acknowledgements

We thank the anonymous referees for helpful comments.

References

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

[2] R. H. Bisseling. *Parallel Scientific Computation: A structured approach using BSP and MPI*. Oxford University Press, 2004.

[3] BSP machine parameters. `http://www.bsp-worldwide.org/implmnts/oxtool/params.html`.

[4] J. M. R. Martin and Y. Huddart. Parallel algorithms for deadlock and livelock analysis of concurrent systems. In *Proceedings of Communicating Process Architectures*, pages 1–14. IOS Press, 2000.

[5] I. Foster and C. Kesselman. *The Grid: Blueprint for a new computing infrastructure*. Morgan Kaufmann, second edition, 2004.

[6] J. M. R. Martin and A. V. Tiskin. BSP modelling of two-tiered parallel architectures. In B. M. Cook, editor, *Proceedings of WoTUG*, volume 57 of *Concurrent Systems Engineering Series*, pages 47–55, 1999.

[7] V. Vasilev. BSPGRID: Variable resources parallel computation and multiprogrammed parallelism. *Parallel Processing Letters*, 13(3):329–340, 2003.

[8] A. Tiskin. The bulk-synchronous parallel random access machine. *Theoretical Computer Science*, 196(1–2):109–130, April 1998.

[9] A. Goldchleger, C. A. Queiroz, F. Kon, and A. Goldman. Running highly-coupled parallel applications in a computational grid. In *Proceedings of Brazilian Symposium on Computer Networks*, 2004.

[10] A. Goldchleger, F. Kon, A. Goldman, M Finger, and C. C. Bezerra. InteGrade: object-oriented Grid middleware leveraging idle computing power of desktop machines. *Concurrency and Computation: Practice and Experience*, 16:449–454, 2004.

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

[12] H. Mattsson and C. W. Kessler. Towards a virtual shared memory programming environment for grids. In *Proceedings of PARA*, Lecture Notes in Computer Science. Springer-Verlag, 2004. To appear.

[13] A. Tiskin. A new way to divide and conquer. *Parallel Processing Letters*, 11(4):409–422, 2001.

[14] M. Nibhanupudi and B. Szymanski. Runtime support for virtual BSP computer. In *Proceedings of IIPS/SPDP*, volume 1388 of *Lecture Notes in Computer Science*, pages 147–158. Springer-Verlag, 1996.

[15] R. V. van Nieuwpoort, J. Maassen, G. Wrzesinska, T. Kielmann, and H. E. Bal. Satin: Simple and efficient Java-based Grid programming. *Journal of Parallel and Distributed Computing Practices*. To appear.

[16] A. L. Rosenberg. To BSP or not to BSP in heterogeneous NOWs. In *Proceedings of Workshop on Advances in Parallel and Distributed Computation Models*, 2003.

[17] M. Adler, Ying Gong, and A. L. Rosenberg. Optimal sharing of bags of tasks in heterogeneous clusters. In *Proceedings of ACM SPAA*, pages 1–10, 2003.

[18] E. K. Davies, M. Glick, K. N. Harrison, and W. G. Richards. Pattern recognition and massively distributed computing. *Journal of Computational Chemistry*, 23(16):1544–1550, 2002.

[19] J.-P. Goux, S. Kulkarni, M. Yoder, and J. Linderoth. Master-worker: An enabling framework for applications on the computational grid. *Cluster Computing*, 4:63–70, 2001.

[20] L. F. G. Sarmenta. An adaptive, fault-tolerant implementation of BSP for Java-based volunteer computing systems. In *Proceedings of IPPS Workshop on Java for Parallel and Distributed Computing*, volume 1586 of *Lecture Notes in Computer Science*, pages 763–780. Springer-Verlag, 1999.

[21] T. L. Williams and R. J. Parsons. The heterogeneous bulk synchronous parallel model. In J. Rolim et al., editors, *Proceedings of IPDPS Workshops*, volume 1800 of *Lecture Notes in Computer Science*, pages 102–108. Springer-Verlag, 2000.

[22] P. Morin. Coarse grained parallel computing on heterogeneous systems. In *Proceedings of ACM SAC*, pages 628–634, 2000.

[23] J. M. D. Hill, S. R. Donaldson, and T. Lanfear. Process migration and fault tolerance of BSPlib programs running on a network of workstations. In D. Pritchard and J. Reeve, editors, *Proceedings of Euro-Par*, volume 1470 of *Lecture Notes in Computer Science*, pages 80–91. Springer-Verlag, 1998.

[24] W. F. McColl. A BSP realisation of Strassen's algorithm. In M. Kara et al., editors, *Abstract Machine Models for Parallel and Distributed Computing*, pages 43–46. IOS Press, 1996.

[25] W. F. McColl and A. Tiskin. Memory-efficient matrix multiplication in the BSP model. *Algorithmica*, 24(3/4):287–297, 1999.

[26] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[27] Sychron Inc. `http://www.sychron.com`.

[28] J. M. R. Martin K. M. Measures and R. C. F. McLatchie. Supercomputing resource management — experience with the SGI Cray Origin 2000. In *Proceedings of WoTUG*. IOS Press, 1999.