A Parallel Multigrid Skeleton Using BSP

Femi O. Osoba and Fethi A. Rabhi

Department of Computer Science, University of Hull,
Hull HU6 7RX
{B.O.Osoba,F.A.Rabhi}@dcs.hull.ac.uk

Abstract. Skeletons offer the opportunity to improve parallel software development by providing a template-based approach to program design. However, due to the large number of architectural models available and the lack of adequate performance prediction models, such templates have to be optimised for each architecture separately. This paper proposes a programming environment based on the Bulk Synchronous Parallel(BSP) model for multigrid methods, where key implementation decisions are made according to a cost model.

1 Introduction

Programming environments and CASE tools are increasingly becoming popular for developing parallel software. Skeleton-based programming environments offer known advantages of intellectual abstraction and development of structured programs [1,9]. Most skeleton-based systems are implemented at a low level using functional languages [3,12] or imperative languages with parallel constructs e.g. PVM/C, dataparallel languages etc. [1,9]. However such models suffer from having no simple cost calculus, thereby hindering the functionality of the skeleton. To generate the required target code for different architectures, such skeleton-based systems have to be optimsed for each architecture. This paper describes a skeleton-based programming environment that is implemented at the low-level with a programming model that posseses a cost calculus, namely the Bulk Synchronous Parallelism (BSP) model.

The next section presents our case study skeleton and Section 3 presents an overview of the proposed programming environment. The next sections present an overview of the user interface, the specification analysis phase and the code generation process. Finally the last section presents some conclusions and future directions of our work.

2 A Case Study : Multigrid Methods

Most scientific computing problems represent a physical system by a mathematical model equation. Presently we are considering a model differential equation of the form $\nabla.K\nabla u = f$ in the 2D space. To make it suitable for a computer solution, the continuous physical domain of the system being modelled is discretised and this leads to a set of linear equations of the form $Ax = b$ with

n unknowns, which needs to be solved for. A particular class of methods for solving such equations is known as *multigrid(MG) methods*. Such methods have been selected as a case study on the basis of their usefulness in an important application area (CFD) and their similarity with SIT algorithms which were the focus of earlier work [9, 12].

The principle behind a multigrid method is to use a relaxation method (e.g. Gauss Seidel or Jacobi) on a sequence of m discretisation grids $G^1 \ldots G^m$ (G^1 represents the finest grid and G^m the coarsest). By employing several levels of discretisation, the multigrid method is able to accelerate the convergence rate of the relaxation method on the finest grid. One iteration of the MG method from finest to coarsest grid and back is called a *cycle*. During a cycle, the different stages in an MG algorithm are *relaxation* which involves the use of an iterative method on the current grid level, *restriction* which involves the transfer of the residual from one grid to the next coarser grid, *coarsest grid solution* which provides an exact solution on the coarsest grid and *interpolation* which is concerned with the transfer of the solution obtained from a coarser grid to the next finer grid. Additional details on multigrid methods can be found in [2].

3 Overview of a Skeleton-Based Programming Environment

The overall structure of the system is very similar to other skeleton-based programming environments [9] as figure 1 illustrates. The primary aim is separating the role of the user and the system developer. The user decides on how the multigrid method should be used and what parameters are to determine the optimal performance irrespective of the underlying architecture, while the system developer addresses issues concerned with generating efficient code and determining suitable or recommended parameters for different architectures based on performance prediction and cost modelling. The main components of the system as shown in figure 1 are described in turn in the next sections.

3.1 User Interface

The role of the User Interface is to allow the user to enter and edit the parameters of the multigrid algorithm. The User Interface also displays the results of the computation which consist of a visual representation of the solution. Figure 2 shows three sets of parameters: *multigrid parameters*, *physical parameters* and *implementation parameters*.

Multigrid parameters The multigrid parameters include the number of grids, the relaxation method, the stencil (e.g. 4 or 8 points) and the cycling strategy (V or W cycles). Another parameter is the number of relaxation iterations performed at various grid levels. Some parameters (e.g. the relaxation method) are expected to be changed after the Specification Analyser has been invoked. When selecting values, the user primarily attempts to reduce the overall execution time by minimizing the number of cycles required to converge to the solution.

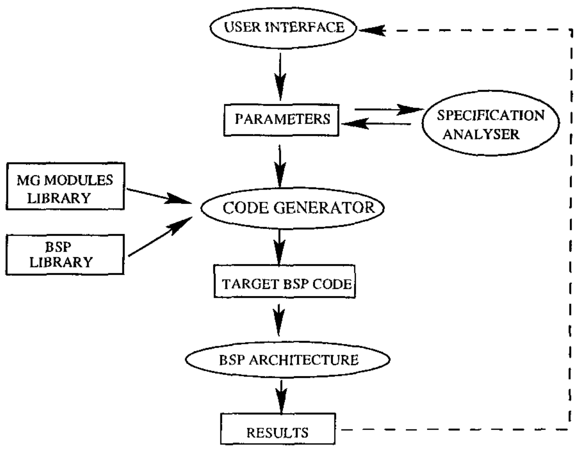
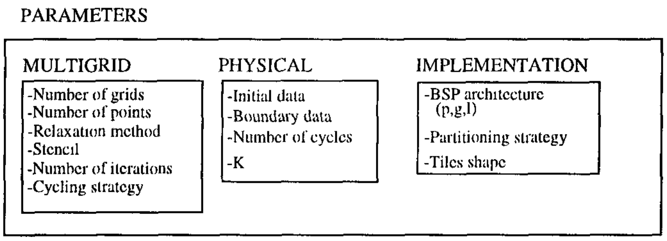


Fig. 1. Overall description of the system



PARAMETERS

MULTIGRID	PHYSICAL	IMPLEMENTATION
-Number of grids	-Initial data	-BSP architecture (p,g,l)
-Number of points	-Boundary data	-Partitioning strategy
-Relaxation method	-Number of cycles	-Tiles shape
-Stencil	-K	
-Number of iterations		
-Cycling strategy		

Fig. 2. User Interface parameters

Physical parameters These parameters are dependent on the physical attributes of the problem e.g. the constant K describes the anisotropy in the problem.

Implementation parameters These parameters represent the characteristics of the parallel implementation. First, the BSP architecture is specified as a triplet (p, g, l) [6]. Another parameter represents the partitioning strategy, which is generally based on grid partitioning techniques. Finally, the tiles shape parameter refers to the shape of sub-partitions (e.g. square or rectangular tiles).

3.2 Specification Analyser

The role of the Specification Analyser is to predict the optimal value of some of the parameters given a minimal subset of values provided by the user. These predictions are made based on estimates on the computation and communication requirements for parallel execution. We base our domain partitioning theoretical cost model on a simple analytic model presented in [11]. Because BSP removes the notion of network topology [6], the goal is to build solutions that are optimal with respect to total computation, total communication, and total number of *supersteps*[6]. For our system this is represented by a parameter set. Each set represents one variation of the algorithm. A parameter set consists of Number of Cycles(Cy), Relaxation Method(RM), Stencil type(St), and Tileshape(Ts). Designing a particular program then becomes a matter of choosing a parameter set that is optimal for the range of machine sizes envisaged for the application. Our BSP theoretical cost model for a standard MG algorithm for a regular grid is of the form:

$$Cy[C.(n^2/p) + g.(n(1/t_r + 1/t_c).M)] + l.S \tag{1}$$

where C is the total number of computation steps, M is the total amount of communication, p is the number of processors (and also the number of tiles), t_r is the number of tiles in a row, t_c the number of tiles in a column, and S is the total number of supersteps. C is a function of (RM, St, n^2, p), M is a function of (Ts, RM, St) and S is a function of (RM, Ts). So for a given number of cycles (Cy), the analyser produces a parameter set (Cy, RM, St, Ts) which results in the lowest execution time for the particular architecture.

3.3 The Code Generator

The role of the Code Generator is to produce BSP code according to the current values of the parameters. The Code Generator uses a set of reusable library modules which form the *MG Module Library* and the BSP library (BSPlib) [4] for handling communication and synchronisation. The MG modules are coded in C and implement each of the phases identified in Section 2.

For further details on the multigrid programming environment, the reader is referred to the Ph.D thesis in [8].

4 Conclusion and Future Work

This paper described a skeleton-based programming environment for multigrid methods which generates BSP code. The user is offered an interface that abstracts from the underlying hardware thus ensures portability and intellectual abstraction. By combining the skeletons idea with BSP, accurate analysis and predictions of the performance of the code on architectures can be made. Since the system is aware of the global properties of the underlying BSP architecture, the system can decide which aspects of the MG algorithm may need to be changed to provide optimal performance and then performs the code generation.

Future work will refine the performance prediction model according to practical results, develop a systematic framework for automatic code generation and increase the parameter set to cater for more realistic problems. Finally, we will also investigate a suitable notation for specifying and modifying parameters and evaluate the proposed system with real users who work with such methods.

5 Acknowledgements

We would like to thank members of the BSP team at Oxford University Computing Laboratory U.K, for their continuous support on the use of the BSP library, and also for making the SP-2 available to us.

References

1. G.H. Botorog and H. Kuchen, Algorithmic skeletons for adaptive multigrid methods, In *Proceedings of Irregular '95*, LNCS 980, Springer Verlag , 1995, pp. 27-41.
2. J.H. Bramble, *Multigrid methods*, Pitman Research Notes in Mathematics, Series 294, Longman Scientific & Technical, 1993.
3. J. Darlington, A. Field, P. Harrison,P. Kelly, D. Sharp, Q. Wu, R. While, *Parallel Programming using Skeleton Functions*, In *Proceedings of PARLE '93*, LNCS, Munich, Springer Verlag, June 1993.
4. M. W. Goudreau, J.M.D. Hill, K. Lang, W.F. McColl, S. B. Rao, D.C. Stefanescu, T. Suel, and T. Tsantilas, A proposal for the BSP worldwide standard library, July 1996, available on WWW `http://www.bsp-worldwide.org/`.
5. O.A. McBryan, P.O. Frederickson, J. Linden, A. Schuller, K. Solchenbach, K. Stuben, C.A. Thole, and U. Trottenberg, Multigrid methods on parallel computers - a survey of recent developments, *IMPACT Comput. Sci. Eng.*, vol, 3, pp. 1-75, 1991.
6. W.F. McColl, Bulk synchronous parallel computing, In *Abstract Machine Models for Highly Parallel Computers*, J.R. Davy and P.M. Dew (eds), Oxford University Press, 1995, pp. 41-63.
7. M. Nibhanupudi, C. Norton, and B. Szymanski, Plasma Simulation On Networks of Workstations using the Bulk Synchronous Parallel mode, in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, Athens, GA, November 1995.
8. B.O. Osoba, *Design of a Parallel Multigrid Skeleton-Based System using BSP*, Ph.D thesis in preparation in the Department of Computer Science, University of Hull, 1998.
9. P.J. Parsons and F.A. Rabhi, Generating parallel programs from paradigm-based specifications, to appear in the *Journal of Systems Architectures*, 1998.
10. F.A. Rabhi, A Parallel Programming Methodology Based on Paradigms, In *Transputer and Occam Developments*, P. Nixon (Ed.), IOS Press, 1995, pp. 239-252.
11. P. Ramanathan and S. Chalasani, Parallel multigrid algorithms on CM-5. In *IEE Proc. Computers and Digital Techniques*, vol. 142, no 3, May 1995.
12. J. Schwarz and F.A. Rabhi, A skeleton-based implementation of iterative transformation algorithms using functional languages, In *Abstract Machine Models for Parallel and Distributed Computing*, M. Kara *et al.* (eds), IOS Press, 1996,