High Performance Computing for Numerical Applications

K. A. Hawick and D. J. Wallace *

December 1992

Abstract

There is an increasing demand for advanced numerical algorithms for computational fluid dynamics studies of the complex flows found in whole systems modelling. The cost of computation requires that these algorithms be implemented efficiently on massively parallel computer systems. There is therefore an interesting interplay between the numerical algorithms and the parallel computing implementation that is necessary to obtain the most accurate solutions as cheaply as possible and in as short an absolute time as possible. We note some of the advanced techniques involving adaptive and dynamic meshing for finite difference, element and volume techniques, multigrid methods and a range of implicit and explicit solvers for numerical modelling in general and computational fluid dynamics in particular. The key issues for implementing these algorithms on massively parallel computer systems are described. A brief overview of the appropriate parallel programming paradigms is presented. Some of these issues are illustrated by activities of Edinburgh Parallel Computing Centre (EPCC) in the field of high performance numerical computing.

*Edinburgh Parallel Computing Centre, James Clerk Maxwell Building, Mayfield Road, Edinburgh EH9 3JZ

1 Introduction

Edinburgh Parallel Computing Centre has a number of application-related activities in parallel computing which seek to enable technology transfer to UK industry and to accelerate the exploitation of high performance parallel computing.

These activities are in the context of a scenario for the high performance computing system environment of the future, which, for the purposes of this talk, starts with engineering workstations, which themselves will soon all be (transparently) parallel, as their silicon components will contain multiprocessors. They will normally be connected in clusters, which will be used as parallel systems for suitably coarse-grained applications, with limited communication requirements. For heavier use, a number of servers (compute, data-base, visualisation etc) will be provided on the local cluster: all will be moderately parallel. Finally, the most demanding applications will access remote supercomputer systems through a high-speed network; the top end of the range will be massively parallel systems. Multi-processor parallelism and networking will therefore be pervasive. The issues for industrial, business and scientific competitiveness are firstly to be on this technology chain and secondly to rise as far up it as the particular application can go.

2 Parallel Software Issues

There are at present a plethora of different parallel and high performance architectures available for numerical applications codes. There are unfortunately many different classification schemes available too. We might however choose to describe three major architecture classes with the emphasis on how they are programmed from an applications perspective. These are: Distributed memory SIMD; Distributed Memory MIMD and (Virtual) Shared Memory systems. SIMD stands for Single Instruction-stream Multiple Data-stream and MIMD stands for Multiple Instruction-stream Multiple Data-stream [1].

2.1 Systems and Environments

Nearly all highly parallel systems currently on the market have physical memory arranged in distributed form — that is to say processors in the system have their own physically separate portion of the memory. The distinction is only important from the point of view of the programmer in terms of how the memory space looks to him/her. Some systems such as the KSR machine [2] support a virtual addressing space that hides the distributed nature of the user's data structures. This makes such systems superficially easier to programme although the loss of user control often means that any parallelism inherent in the user code is not fully exploited.

Distributed Memory SIMD systems such as the Connection Machine CM-200 [3] allow the applications programmer explicit control over the placement of user data in the processors' memory either through FORTRAN90-like array syntax or even more explicitly through the use of compiler directives. Typically such a system will have a very large number of processors, each with its own local memory and interconnected with a fixed

topology network. User data can be laid out in a number of different mappings chosen by the applications programmer as appropriate for the algorithms involved. For example, solving a partial differential equation with model fields on a regular grid naturally suggests partitioning the data geometrically so that a given processor will be able to carry out the local computations on its own part of the field data. This is the "data-parallel" paradigm. For this to be effective on current systems, the computations should be 'local' (and preferably not too high-dimensional), in order to avoid incuring excessive communication costs inherent in accessing another processor's data. The processors typically do not each have a full copy of the applications code; at most they have small subsections of code broadcast to them which they act on synchronously.

One of the inhibiting factors preventing the effective exploitation of parallel systems is that most compilers for such systems (and also for many serial machines) support various *manufacturer-specific* compiler directives and the resulting loss of portability has tended to lock optimised applications codes onto one system. Recent attempts to remedy this situation, at least for scientific codes, have centred around the proposed High Performance FORTRAN Draft [4].

The other approach to parallel systems exploitation involves the explicit message passing programming paradigm, usually found on Distributed Memory MIMD systems. For such systems, the applications programmer takes full account of the fact that the machine has a number of different and autonomous processors. Machines such as the Computing Surface series from Meiko [5], and now Performance Computing Industries, can be programmed by placing an independent applications code on each processor in a system which can have interconnection topologies chosen by the programmer at run time. These independent codes might be completely distinct codes or more often they will be copies of the same image, which will behave differently at run time according to conditional statements built in by the applications programmer. This programming model is known as Single Program Multiple Data (SPMD). Data flow between individual processors is controlled by the applications programmer through the use of explicit messages sent and received by the different copies of the code on different processors. It is perfectly feasible to construct pde solvers and so forth using this programming paradigm and high performance can be achieved although at some cost to the code portability.

One of the main factors perceived to inhibit this means of exploiting parallel systems concerns the lack of standardisation of the syntax for the explicit message passing calls within the applications code. We believe this argument is somewhat spurious. There are indeed a number of different message passing systems available at the time of writing, many are proprietary but some are now more widely available on many parallel systems [6, 7, 8]. However, it is our direct experience from porting applications code to run on parallel systems that by far the bulk of the programming design and coding effort goes into porting a *serial* code to a *parallel* message passing system. Once such a code is running, the changes from one message passing system to another involve relatively minor changes in syntax.

2.2 Tools

An appealing route to exploiting parallel systems involves the use of automatic tools to convert serial code into code suitable for a parallel system. Many parallel systems

available today do come with some elementary tools to aid code conversion, usually to a manufacturer-specific intermediate format. Typically such tools involve a code pre-processor which carries out some data layout and dependency analysis and will output FORTRAN with (machine specific) embedded compiler directives for example. There are two problems with systems available at present. Firstly, the analysis that such tools are capable of at present is still elementary and it is nearly always necessary for the applications programmer to provide extra direction. This requires the programmer to know something about the parallel machine and furthermore to express this in some way to the compiler. Secondly, most tools to date have been very specific to one parallel system and have output very non-portable code. This may allow efficient implementation on the particular system concerned, but inevitably once the applications programmer has also added his own directives and has ceased to work on the original serial version of the code, there are serious portability problems.

Tools such as Forge [9] do provide some basic conversion of serial code to message passing form. Similarly tools such as VAST-90 [10] will convert serial (FORTRAN-77) code to explicit data-parallel form. Most virtual shared memory systems available today come with tools such as autotaskers which attempt to divide some of the computational load of an applications code across their processors [11]. To our knowledge, however, there is no such tool available at present that is independent of a particular manufacturer's system.

The tools essentially carry out the "grunt" work but are still incapable of understanding a full production industrial code in the way that a trained parallel programmer can. In fact it appears to be generally agreed that there is insufficient information in a typical serial code for a compiler, or a tool based on compiler-technology, to replace the programmer on such a task if code is written in the langauges that are in common use for scientific and engineering work at present.

2.3 Utilities and Libraries

There have been a number of attempts to conceal the lack of standardisation in parallel software by abstracting the problems further towards the applications algorithms and code itself. Some of the best known packages to aid in the exploitation of vector architectures have been the Basic Linear Algebra Subprograms (BLAS) [12] and the libraries of routines constructed on top of them such as LINPACK [13] and more recently LAPACK [14]. These projects have been successful in allowing applications programmers to exploit the *vector* parallelism in a standard and portable way. What is desperately needed is some means to construct similar packages to exploit *massively parallel* systems in a portable (non-vendor specific) way.

In the PUL component of its Key Technology Programmes, EPCC is working to provide support for applications codes in the form of a set of utilities which exemplify some of the different programming paradigms for parallel systems [15]. Current packages include support for task farms, regular domain decomposition and parallel file access. Packages for scattered spatial decomposition and irregular domain decomposition are currently under development.

Other institutions have proposed similar libraries to insulate the applications codes from

machine specific details and from differing syntactical expressions of parallelism. See for example [16, 17].

A weakness of such packages in the past has become apparent when trying to specify the required interface to the applications code. In LAPACK/LINPACK the problem manifests itself in the need to have many different routines that carry out the same operation on different data types (eg REAL, DOUBLE, COMPLEX). A more serious manifestation becomes apparent when dealing with large industrial codes. Such code will typically not consist of a single algorithm or operation. For example a computational fluid dynamics or weather modelling code will not solely consist of a partial differential equation(pde) solver. It will typically have other operations ongoing on its data sets. These might be some filtering operations such as an FFT on the pde model fields, or some analysis of the model data or perhaps some global operations such as might be involved in finding an average model field value. This causes problems because the data mapping onto processors for one algorithm may be entirely inappropriate for another and somehow optimisations for all the operations in a production code must be compatible. See section 4.1 below.

The problem can be addressed by providing data remapping routines that are independent of the numerical routines. This can prove ineffective however, if the long range communications costs of doing a remap are high, particularly if the numerical algorithms are heavily intertwined. A particular example of this is described for a weather model [18] in which the analysis scheme takes place inside the same loop as evolution of the model weather fields. A good (regular) data mapping for the weather dynamics is not naturally compatible with the (scattered) decomposition scheme which is optimal for the analysis.

EPCC strategy to address such problems is the design of the PUL utilities to support remapping of data within the same applications code [15].

2.4 Numerical Algorithm Issues

Traditional solver algorithms for partial differential equations such as Jacobi are relatively straightforward to implement on parallel systems. The Jacobi algorithm involves only local computations [19] and consequently is well suited to a simple data decomposition scheme. More modern solvers may require the data to be accessed in a more complex fashion that does not map so easily onto regular processor topologies. Modern techniques such as multigrid and multilevel methods are becoming increasingly important for modelling whole systems in mechanics and engineering applications. The numerical stability and accuracy requirements call for methods such as these, but they are certainly more challenging to implement efficiently on parallel systems. The multigrid method [20] involves maintaining a number of "coarsened" model meshes as well as the original mesh and a number of interpolations between grids are interspersed with the main calculations. Apart from the reduction in the degree of potential parallelism, these "W-cycles" are less easy to implement than straightforward methods since it is no longer obvious on which processors' local memory the model data should reside nor which processor should be doing which calculation.

The applications programmer may then have a design choice of using a high performance

numerical algorithm that may be inefficient to implement on a parallel system, or an unsophisticated algorithm that can be implemented very efficiently. In terms of job throughput to a desired precision, and in a given wall-clock time, the latter can often win. See section 4.4 below.

For engineering problems a number of highly sophisticated techniques such as adaptive and dynamic gridding are often necessary not only to economise on the number of grid points but sometimes just to obtain any stable solution at all. It is a considerable challenge to modern programming methodology to map an adaptive or dynamic set of mesh nodes of a finite volume or element problem to a fixed processor topology. It is necessary to rely on software to address the locality issues and to balance the computational load across the processors in a parallel system. Algorithms such as recursive bisection with quad and octal trees are being developed for this purpose [21]. See section 4.2 below.

3 Edinburgh Parallel Computing Centre

Work on applications of parallel computing at Edinburgh dates back to 1980, when members of the Physics Department began exploiting the ICL DAP at Queen Mary College, London. These activities culminated in the establishment in 1990 of EPCC as an Interdisciplinary Research Centre in the Faculty of Science and Engineering. The Centre has grown to a professional staff of more than 40, with activities spanning applications, the Key Technology Programmes and service provision. Much of the work is in collaboration with industry and a significant effort (in University terms) is targeted to marketing.

The initial growth of the activity was based on the procurement of parallel systems driven by the physicists, and hosted in the University Computing Services. The general service provision they provided built up the user base and expertise in a wide range of departments. The net effect has been a positive feedback loop of broadening interest and utilisation. The major systems presently include:

- The Edinburgh Concurrent Supercomputer, a Meiko Computing Surface with around 400 nodes (Inmos T800 transputers) and 1.8 Gbytes of memory; this MIMD system provides a national multi-user resource with an active user community of around 200, claiming around 1.5M processor hours per year in recent years.
- A 4K-processor AMT DAP 608.
- A 64-node i860 Computing Surface with 5 Gflops peak performance and 1 Gbyte of memory; this system is hosted at Edinburgh primarily on behalf of two UK "Grand Challenge" collaborations, 65% in Quantum chromodynamics (QCD) and 25% in material structure calculations using the Car-Parrinello method. The first sustains up to 1.5 Gflops, the second around 5 times (single processor) Cray XMP; together these collaborations have had access to a dedicated resource broadly equivalent in real throughput to a Cray YMP8. In recognition of the work of the groups, Meiko have recently installed a further 16 node system with 512 Mbytes of memory.
- A 16K processor CM-200 was installed by Thinking Machines in September 1991; it has a peak of 8 Gflops, 0.5 Gbytes of memory and a 10 Gbyte data vault. Utilisation has risen to around 70-80% over the last several months.

- A network of more than 50 workstations and servers for EPCC development work.

All systems are run under SunOS, with a common file store and are fully networked for national use. Enquiries for registration should be addressed to EPCCsupport@uk.ac.ed.

The major programmes in the Centre currently include:

Applications undertaken largely with industrial collaboration, funded by direct contract, or within the DTI/SERC Parallel Applications Programme, or, most recently, through the LINK scheme;

Key Technology Programmes, currently in message passing (CHIMP), parallel utility libraries (PUL) and networked visualisation (NEVIS). This work is aimed at bridging the gap between research and applications, and draws on common requirements across a range of applications. It is funded by a combination of industrial and public agency sources;

Education, training and awareness covering a range of activities, from Industrial Affiliation, short courses, the Summer Scholarship Programme and the TRACS programme, in which EPCC will be supported as a Large Scale Facility by the CEC, under Human Capital and Mobility.

Further details of these and other activities are contained in the quarterly Newsletter and Annual Report available from EPCC at the University of Edinburgh. Literature is also available electronically from the EPCC anonymous ftp server. Connect to epcc.ed.ac.uk (129.215.56.29), logging in as *anonymous* using your username as a password.

4 Numerical simulation projects ongoing in EPCC

EPCC is currently undertaking twelve applications projects, most of which have some numerical component. The following outlines four of these which involve particularly heavy numerical simulation.

4.1 Computational Fluid Dynamics for Aerospace Industry

EPCC has an ongoing collaborative project with Rolls Royce plc to investigate production computational fluid dynamics (CFD) codes that are used for example to model turbofans in engine systems. One of the codes being studied is several tens of thousands of lines of vectorised FORTRAN and presents some interesting problems for a massively parallel implementation. It is unfortunately the case that well written code which has been optimised both for computational performance and to manage memory in an economical manner may hide some of the inherent parallelism in such a way that parallel compilers can not presently detect it. It therefore becomes the task of the parallel programmer to employ his knowledge of the applications problem and all its symmetries and in-built structure to make decisions as to how the data, memory and computations should be partitioned. In this particular case we are able to employ a regular data decompositional scheme for the main fluid dynamics code, although some additional highly optimised communications routines are necessary for the turbulence model. That the

data structures are regular is not obvious from the code itself, but is known from the science of the modelling. Consequently, humans can see it, compilers presently can not.

4.2 Computational Fluid Dynamics for the Nuclear Industry

Another EPCC project with AEA Technology, Dounreay involves a CFD code for whole systems modelling. This code is nearly 100,000 lines of FORTRAN and had already been ported to a message passing environment by AEA. The port had been done in terms of a proprietary message passing system and our first task was to port it to the CHIMP message passing system which has been implemented on a wide base of parallel systems[6]. The code itself involves many options for different solution methods such as multigrid on highly irregular meshes. The load balancing for a static and ultimately for an adaptive and dynamic decomposition is therefore an interesting software challenge. Some of the work on adaptive and dynamic meshing is described in [21].

4.3 Weather and Climate Modelling

A problem related to CFD is that of numerical weather prediction (NWP) and climate modelling. We have an ongoing collaboration with the UK Meteorological Office to investigate their Unified Model for NWP and Climate Modelling. This is a code of approximately 150,000 lines of well optimised vector code and has a number of different algorithms that operate on the same model data fields. The CFD component solves the Navier-Stokes equations on a regular mesh and is relatively straightforward to decompose onto processors and memory, but the analysis stage at each time step involves interpolating real observational data at arbitrary positions onto the model grid. Our parallel processing strategy for this involved a scattered spatial decomposition of observational data and is described in [18].

4.4 Oil Reservoir Modelling

Another area of engineering simulation that is akin to CFD concerns the modelling of oil and gas reservoir systems. EPCC has ongoing commercial projects to study parallel methods for these. The equations are set up using an an empirical law know as D'Arcy's law rather than from the Navier-Stokes equations but the numerical methodology concerned is very similar to that for CFD. Our current strategies are to find automatic methods for converting code written for shared memory systems to code appropriate for distributed memory systems, either through explicit message passing calls or through HPF style directives. The solvers typically used in such codes to date are often highly optimised for serial, vector or shared memory systems with few processors. Finding solver algorithms which permit a scalable parallel implementation is important.

5 Concluding Remarks

The evidence today is that, for a wide range of applications, scalable performance to the highest absolute levels will be obtained only on physically distributed memory systems.

Machines supporting explicit message passing and HPF style specification of parallelism are exploitable for scientific and engineering applications. Virtual shared memory systems, while arguably easier to use at an elementary level, pass the problems of parallelism onto the compiler writers and are still very much in their infancy.

In terms of the systems software available to the applications programmer, notwithstanding the fascination of parallel computing systems *per se*, it is important to remember that most programmers use them not because they want to but because they have to, in order to obtain the price/performance they need. For this reason it is of the utmost importance that a means be found to express parallelism in a standard syntax. The proposed HPF draft is one move in this direction.

Finally, although the problems of parallel computing are by no means solved and applications programing of such machines not yet trivial, it is worth stressing that the rewards in scientific computation are already being realised, and the potential rewards in engineering, industrial and commercial applications are considerable.

6 Acknowledgements

Edinburgh Parallel Computing Centre is supported by major grants and contracts from the Advisory Board for the Research Councils, the Department of Trade and Industry, the Information Systems Committee of the Universities Funding Council, the Science and Engineering Research Council, Scottish Enterprise Software Group, and industry. It is a pleasure to thank colleagues in EPCC for their contribution to the projects reported here.

References

[1] Architecture of high performance computers, R.N.Ibbett and N.P.Topham. Pub. Macmillan, 1989.

[2] KSR1 Overview, Kendall Square Research, 1992.

[3] Connection Machine CM-200 Series, Technical Summary, Thinking Machines Corporation, 1991.

[4] *Draft* High Performance FORTRAN Language Specification, High Performance FORTRAN Forum, November 1992.

[5] Computing Surface User Guide, Meiko Ltd, 1990.

[6] Common High Level Interface to Message Passing (CHIMP), *various user guides and interface specifications*, Edinburgh Parallel Computing Centre, 1992.

[7] A Users' Guide to PVM Parallel Virtual Machine, Adam Beguelin, Jack Dongarra, Al Geist, Robert Manchek and Vaidy Sunderam, Oak Ridge National Laboratory, ORNL/TM-11826, 1991.

[8] PARMACS, The Argonne/GMD Macros in FORTRAN for Portable Parallel Programing Using the Message Passing Programming Model, PALLAS GmbH, 1991.

[9] FORGE90 User's Guide, Applied Parallel Research, Inc, 1992.

[10] VAST-90, User's Guide, Pacific-Sierra Research Corporation, 1992.

[11] CRAY 2 User Guide, CRAY Research (IK) Ltd, 1990.

[12] Basic Linear Algebra Subprograms for FORTRAN usage, C.L.Lawson, R.J.Hanson, D.Kincaid and F.T.Krogh, ACM Trans.Math.Soft.,5(1979),pp308-323.

[13] LINPACK Users' Guide, J.J.Dongarra, J.R.Bunch, C.B.Moler, G.W.Stewart, SIAM, 1979.

[14] LAPACK Users' Guide, E.Anderson, Z.Bai, C.Bischof, J.Demmel, J.Dongarra, J. Du Croz, A.Greenbaum, S.Hammarling, A.McKenney, S.Ostrouchov and D.Sorensen, Siam, 1992.

[15] Parallel Utility Library (PUL), *various user guides and interface specifications*, Edinburgh Parallel Computing Centre, 1992.

[16] PARLANCE Numerical Algorithms in a Virtual Shared Data Space, R.J.Allen, Daresbury Laboratory, 1992.

[17] The GMD Communications Library for Grid-Oriented Problems. R.Hempel and H.Ritsdorf, GMD Arbeitspapier No 589.

[18] Parallelisation of the Unified Model Data Assimilation Scheme, Kenneth A. Hawick, R. Stuart Bell, Alan Dickinson, Patrick D. Surry and Brian J.N. Wylie, Proc. 5th ECMWF Workshop on Use of Parallel Processors in Meteorology, November 1992.

[19] Numerical Recipes, The Art of Scientific Computing, William H. Press, Brian P. Flannery, Saul A. Teukolsky and William T. Vetterling, Pub. Cambridge, 1989.

[20] Multilevel Projection Methods for Partial Differential Equations, Stephen F. McCormick, SIAM, 1992 and references therein.

[21] Distributed Irregular Meshes in Computational Fluid Dynamics, K.D.Murphy, EPCC-SS92-21, and A Navier Stokes Equations Solver for Viscous Incompressible Fluid, Sigitas Keras, EPCC Summer Student Programme Technical Reports, 1992.