The Parallel Performance of a Tightly Coupled 3D Magnetohydrodynamic Simulation

Lee Margetts *and* **Mike Pettipher**
CSAR, Manchester Computing

ABSTRACT: *In magnetohydrodynamics, two distinct processes, magnetism and fluid flow, are physically coupled. Traditional methods start by decoupling these processes, solving the fluid and electromagnetic problems separately. The physical coupling is achieved later using the results of the fluid calculation in the electromagnetic problem and vice versa. We depart from this approach, solving the full system of equations using a tightly coupled parallel program. The parallel performance is investigated using a 512 processor Origin3800.*

Introduction

In real engineering problems, two or more physical processes are often coupled. A typical example is fluid flow, where the pressure and velocity fields are coupled by the Navier Stokes equations. In magnetohydrodynamics, a further physical process is involved as the fluid is electrically conducting and its behaviour can be modified by an externally applied magnetic field.

A common solution approach is to decouple the physical processes and use separate software packages to solve each physical problem. Although the problem is more manageable, the overall computational effort increases as information is transferred between packages. Often these packages use different computational methods with additional work being carried out interpolating the values of the variables between different computational domains. An example is described by Morandini et al. [1994].

As in the field of computational fluid dynamics, magnetohydrodynamic problems can be solved by a variety of numerical techniques such as finite differences or finite elements. Various formulations are in use that simplify the equations by taking mathematical short cuts. For simple test problems, these methods produce similar answers within reasonable computation times. However, for complicated geometries, these simplified methods may not give the correct answer or give any answer at all.

The closest we can get to the correct solution is to solve the full system of simultaneous equations with no simplifications. This is often referred to as direct numerical simulation or DNS. Being computationally very expensive, DNS is not practical for everyday problems. However, solving large complicated problems using DNS on supercomputers benefits the developers of simplified industrial algorithms by providing accurate solutions for validation work.

In this paper, the physical processes remain fully coupled in terms of the equations, which are solved, in a tightly coupled parallel program using DNS.

Numerical Method

The steady state magnetohydrodynamic (MHD) equations for an electrically conducting viscous incompressible fluid are as follows:

$$u_j \cdot \frac{\partial u_i}{\partial x_j} + \frac{1}{\rho}\frac{\partial p}{\partial x_j} - \frac{B_j}{\mu\sigma}\left(\frac{\partial B_i}{\partial x_j} - \frac{\partial B_j}{\partial x_i}\right) - \frac{\nu}{\rho}\nabla^2 u_i = 0 \qquad (1)$$

$$u_j \cdot \frac{\partial B_i}{\partial x_j} - B_j \cdot \frac{\partial u_i}{\partial x_j} - \frac{1}{\mu\sigma}\frac{\partial^2 B_i}{\partial x^2} = 0 \qquad (2)$$

$$\frac{\partial B_i}{\partial x_j} = 0 \qquad (3)$$

$$\frac{\partial u_i}{\partial x_j} = 0 \qquad (4)$$

The equation relating to the fluid flow (1) and the equation relating to the magnetic field (2) both contain terms in velocity u_i and magnetic field B_i, i.e. they are coupled. Equations (3) and (4) relate to the continuity conditions for the u_i and B_i respectively. The material

properties are as follows ν (viscosity), ρ (density), μ (permittivity) and σ (conductivity).

Approximation by the Galerkin process leads to the equilibrium equation 5:

Navier - Stokes Coupling

$$\begin{bmatrix} C_{11} & C_{12} & 0 & 0 & C_{15} & C_{16} & C_{17} \\ C_{21} & 0 & C_{23} & C_{24} & 0 & 0 & 0 \\ 0 & C_{32} & C_{11} & 0 & C_{35} & C_{36} & C_{37} \\ 0 & C_{42} & 0 & C_{11} & C_{45} & C_{46} & C_{47} \\ C_{51} & 0 & 0 & 0 & C_{55} & 0 & 0 \\ 0 & 0 & C_{51} & 0 & 0 & C_{55} & 0 \\ 0 & 0 & 0 & C_{51} & 0 & 0 & C_{55} \end{bmatrix} \begin{Bmatrix} u \\ p \\ v \\ w \\ B_x \\ B_y \\ B_z \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix} \quad (5)$$

Coupling Magnetic Diffusion

Using 20-node brick elements to solve for the velocity and magnetic fields and 8-node bricks to solve for the pressure leads to a 128 x 128 term element stiffness matrix. The terms that represent each primitive variable (u, p, v, w, etc.) are collected into the submatrices C_{11} etc. The stiffness matrix can be divided into four segments as indicated by the broken lines. In the top left hand corner, we simply have the Navier Stokes terms developed by Taylor [1981]. In the bottom right hand section, the terms describe the diffusion of the magnetic field only. The remaining quadrants describe the coupling between the two physical processes.

Solution by assembly methods requires the construction of a sparse global stiffness matrix that is most simply solved by gaussian elimination. The size of problem that can be solved 'in core' by assembly methods is restricted by memory. 3D problems quickly reach a size where special procedures are required. These invariably rely on 'out of core' storage strategies such as 'paging'. Such input/output strategies, whilst commonly used, should ideally be avoided as they greatly degrade program performance.

In element by element methods, a global stiffness matrix is never assembled and the solution is advanced using an iterative algorithm. Such methods have been shown to lend themselves well to parallelisation (Smith and Pettipher, [1997] and Carey et al.,[1997]).

A serial finite element program was developed to solve both the 3d Navier Stokes and magnetohydrodynamics equations using the iterative solution algorithm BiCGStab(l). This was parallelised using the method outlined by Pettipher & Smith [1997].

The finite elements are divided equally between the processors. Each processor calculates the element stiffness matrix for each of its allocated elements. The solution proceeds with each processor working independently through its set of elements. In both serial and parallel programs, the largest proportion of run time is spent in the matrix-vector multiplication described. The principal difference between the two is that the parallel code expends additional effort in communication and setting up the Message Passing Interface (MPI). Also some processing and storage is duplicated across the processors. However, the parallelisation strategy works well and greatly outweighs these additional costs.

Program performance

Around 99% of the execution time is spent in the iterative solver BiCGStab(l) and the complete solution of the finite element analysis requires hundreds or thousands of iterations, depending on the problem. As exactly the same number of operations are performed in each iteration, the performance of the program can be determined by measuring the performance of one iteration. The performance results presented here were taken over tens of iterations to smooth out any variability which may arise due to system loading.

Most of the computation is carried out in the following 'do loop':

```
gather(x)
   do iel=1,nels-pp
         u=matmul(ke,x)
   end do
scatter(u)
```

The gather operation collects the values required to complete the vector x from neighbouring processors. The matrix vector multiplication proceeds as in the serial program, with each processor working through its own set of elements. The scatter operation ensures that contributions to a shared nodal value (or freedom) are communicated across processors. Later discussions distinguish between the performance of the matrix-vector computation only and the whole kernel, including the gather/scatter communication overhead.

Initial studies, carried out for a Navier-Stokes (NS) problem using a 195MHz Origin 2000, showed that the utilization of the processor dropped from a high of around 20% of peak to 10% of peak when increasing the size of the problem analysed.

The higher value was achieved for a very small problem of around 200 finite elements in size. A simple hand calculation showed that the vectors and stiffness matrices required for the small problem could all fit into the cache memory. A consequence of this is that the processor does not need to retrieve data from main memory during the computation and each time data is required it is available in cache.

Various ways of optimizing the program to overcome the performance drop were investigated. Various compiler optimisations and mathematical libraries were used without success. Matrix-vector computation does not provide enough work to keep the processor busy whilst the memory accesses are being carried out. It is inevitable that at some point, the memory accesses hold up the processor.

Although 10% is quite acceptable, it was of particular concern that in a parallel environment the speed of data

retrieval from main memory can be quite variable, being delayed by other users. It was observed that 10% peak was at the high end of a variable range that depended greatly on the machine loading. As the parallel programs are computationally tightly coupled, a delay to one processor inevitably delays all the other processors in the analysis.

Other than keeping the number of elements within the cache limit, there is not much that can be done to optimise the performance of the program. With respect to the parallel implementation, 210 elements per processor is theoretically the optimum limit. This was not a positive observation considering that one would like to perform analyses with thousands, if not millions of elements. Furthermore, it should be noted that out of the vendor's range of possible configurations, the cache on this machine was of a generous size at 8Mb.

The incomplete parent stiffness matrix

Further thoughts on how to overcome this limit led to the question – do we really need to store all of this information?

The stiffness of a finite element depends on its shape and material properties. For the Navier-Stokes problem, if all the finite elements are of the same shape and have the same material properties, the majority of the stiffness matrix is common to all. Only the submatrix C_{11}, is unique for each finite element. The rest of the stiffness matrix can be considered as being an incomplete parent for all elements with the same geometry and physical properties. With only 8% of the storage originally required, this represents a considerable improvement. The matrix vector multiplication can then be carried out in a number of steps, with the final answer vector being assembled at the end.

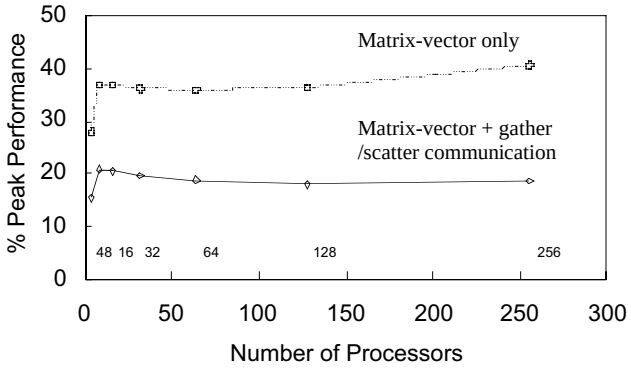

Figure 1 %Peak performance: 6,500,000 NS equations

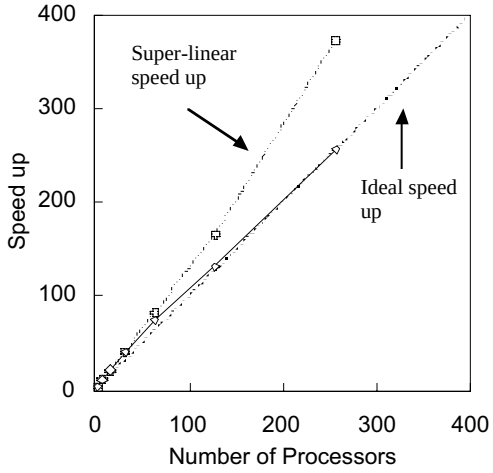

Figure 2 Speed-up: 6,500,000 NS equations

The performance of the program was greatly improved as shown in figure 1. For a problem with 6.5 million equations, a sustained peak performance of around 20% was obtained for up to 256 processors. The matrix-vector computation, excluding the parallel overhead, sustained 40% of peak, representing the upper limit of the performance that could be achieved should the communication overhead be reduced.

The principal drawback of this strategy is that the generality of the finite element approach has been lost - all elements must have the same shape and material property. One of the advantages of finite element analysis is that it can be used to solve problems in arbitrarily shaped domains. Also, techniques such as adaptive mesh refinement, which increase element density in regions of high field gradient, can no longer be applied, although one could have families of finite elements with identical shapes and material properties.

There is an obvious way around this limitation for simple elements. If one was using two dimensional equilateral triangle elements, there exists a parent stiffness matrix that when multiplied by an appropriate scalar value can represent any size equilateral triangle. Arbitrary bricks are difficult to manipulate in this way because complicated functions, taking into account rotation and translation, would have to be applied to a parent brick stiffness matrix to transform it into any arbitrary brick. This information would both be difficult to compute and may result in a similar storage requirement - a matrix for each finite element!

Optimising for cache use – MHD stiffness matrix

Accepting this limitation for the moment and returning to the full MHD element stiffness matrix, it turns out that all of the remaining eleven submatrices, Cij, are unique for each finite element – whether or not they share the same shape and material properties. Storage and therefore cache reuse still remains a problem even if the incomplete parent approach is used. An alternative way to improve cache reuse is to store each submatrix in its own array, store-Cij, and unroll the matrix-vector loop into several partial computations as follows:

The original matrix-vector computation:

```
do iel=1,nels-pp
   u=matmul(ke,x)
end do
```

is replaced by a series of do loops, one for each submatrix:

```
do iel=1,nels-pp
   u'=matmul(C15,x')
end do

do iel=1,nels-pp
   u'=matmul(C55,x')
end do                          etc.
```

where `nels-pp` are the number of finite elements per processor, `ke` is the element stiffness matrix, u' and `x'` are the appropriate parts of `u` and `x` and `C15`, etc are the submatrices.

The performance figures obtained when solving a 4 million MHD problem using an SGI Origin 3800, are shown in figures 3 and 4. In short, the optimisation strategy outlined for the MHD stiffness matrix storage has proven successful.

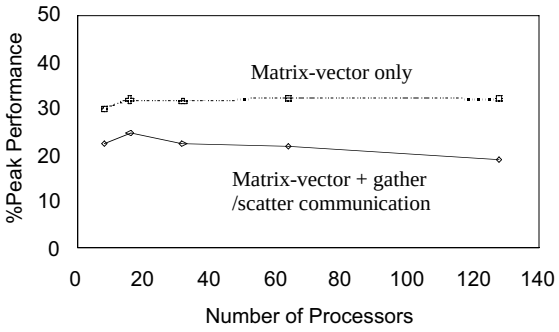

Figure 3 %Peak performance: 4,000,000 MHD equations

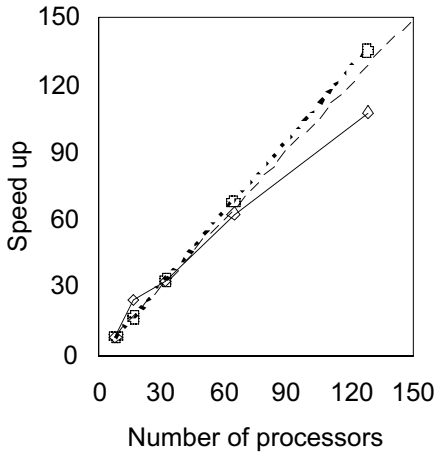

Figure 4 Speed up: 4,000,000 MHD equations

Comparing figures 1 and 3, the performance of the MHD matrix vector computation is ~5-10% lower than that observed for the Navier Stokes problem. In the Navier-Stokes case, there is only one nels_pp loop and this is executed using values stored in the C11 submatrix. As the MHD problem requires 12 separate loops, one for each submatrix, there is an inevitable delay due to main memory access as each submatrix array is loaded into cache.

There is no corresponding performance drop when parallel overhead is included because, between the communication intensive gather and scatter steps, more than twice as much computation is carried out for the MHD case compared with the Navier-Stokes. The MHD problem has a higher computation to communication ratio.

Efficient cache use for arbitrary elements

We now reconsider the incomplete parent method described for the Navier-Stokes problem. Fortunately, the optimisation described for the MHD problem can be extended to include those submatrices that are stored once in the incomplete parent method. For non-uniform finite element domains, the submatrices C12, C21 etc can be stored as C15, C51 etc above and further 'do loops' can be computed.

Unrolling the matrix vector computation for each submatrix enables efficient cache reuse and facilitates reasonable, scalable performance, even for finite element meshes with elements of arbitrary shape or material property. Further extending the physics, for example by including temperature dependent behaviour, will increase the size of the element stiffness matrix by the addition of further submatrices. If the methods described here are employed, performance should not suffer as the complexity of the physical model increases.

Convergence variability

A final performance issue will now be mentioned briefly. In serial finite element analysis, any problem will take a fixed number of iterations, say 100, to reach convergence. If the program were executed several times, using the same data file, the program would always take the same number of iterations (100) to achieve the same result.

This was not true for the parallel program. The number of BiCGStab(l) iterations required before the convergence criterion was satisfied varied according to the number of processors used. The observations were not always repeatable. Repeated executions using the same data file and the same number of processors did not always give exactly the same numbers in the results output or exactly the same number of BiCGStab(l) iterations.

The number of iterations could sometimes be less and at other times be more than observed for the serial case, increasing the computational efficiency or contributing to the parallel overhead respectively!

The variability in iteration count was attributed to numerical roundoff resulting from the non-deterministic ordering of quantities summed across various processors. A full explanation is given in Margetts [2002].

Case Study

At this time, the parallel program has only been used to solve one very large Navier-Stokes problem to completion, the lid-driven cavity. The problem is a well documented test case for CFD algorithms. A cubic cavity contains a fluid that is initially at rest. The top surface of the cavity or 'lid' is driven at a constant velocity. A steady state solution is then sought for the motion of the fluid inside the cavity.

As the problem has a symmetry plane, only half needs to be analysed. The domain was subdivided into a quarter of a million finite elements, giving rise to 1 million grid points where values for the pressure and velocity field were to be calculated. With approximately 4 unknowns at each grid point, the computational task was to solve a system of 4 million non-linear simultaneous equations.

Problem Size

256 000	20 node brick elements
1 000 000	Nodes or grid points
4 000 000	Simultaneous equations

It should be noted that the degree of refinement described is not necessary to solve this particular problem. The purpose of the exercise was to demonstrate that the performance of the program, as indicated in figures 1 and 2 could be sustained for a large problem that was run to completion.

Table 1 shows the performance data recorded using 256 processors on Green, CSAR's 512 processor Origin 3800.

Table 1 Performance data versus Reynolds number

Re	Parallel Time Minutes	Serial Time Days	%Peak*	Gflops
10	20	2-3	29	60
100	47	8-9	29	60
1000	180	>1 month	29	60

*includes communication

Visualising the results of the simulation required help from a visualisation expert. Joanna Leng prepared the demonstration using the developer's edition of AVS Express. Figure 5 shows streamlines with advected particles and figure 6 shows slice planes through velocity magnitude data. The arrow indicates the direction of the motion of the lid.

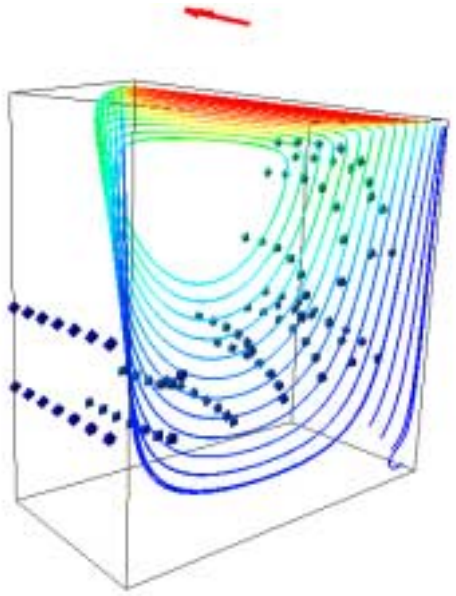

Figure 5 Streamlines with advected particles

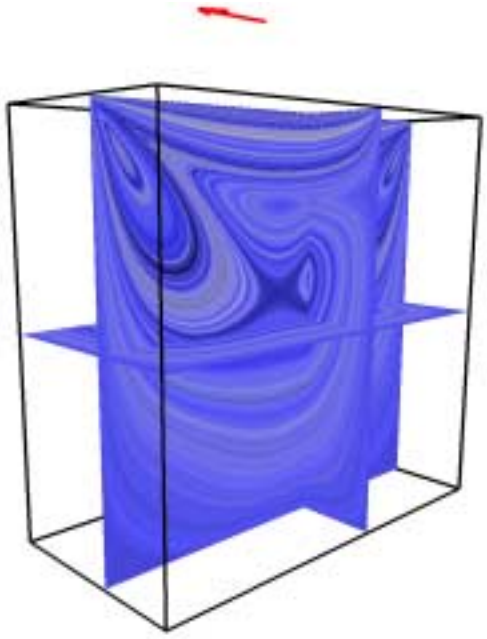

Figure 6 Sliceplanes through the velocity magnitude data

Conclusions

Direct numerical solution of the fully coupled magnetohydrodynamic equations is generally considered too computationally time consuming to be of practical use. Although this may be true with serial machines, fast and efficient, scalable parallel codes are readily achievable using element by element methods. For large coupled 'multiphysics' problems that require parallel computation, the method described becomes 'competitive' when compared with other solution methods.

Future Objectives

The authors have developed more than a scalable, high performance parallel program to solve the equations presented here. The parallelisation strategy has been generalized, enabling the development of a suite of finite element programs for a wide variety of engineering problems. These programs complement the serial programs presented as teaching material in Smith and Griffiths (1998).

There are two aims to this work. Firstly, the programs have been written to have the same general appeal as the original serial programs, engaging the non-specialist in parallel computation. This has been achieved by hiding the parallel components in a library, making the parallel source code strikingly similar to the serial. Secondly, these 'driver' programs will form the basis of a finite element module, putting 'reality' (of material behaviour) into a virtual reality prototyping design tool, currently under development in a Northwest of England collaborative project.

Acknowledgements

The authors would like to thank Professor Ian Smith (ian.smith@man.ac.uk), School of Engineering, University of Manchester for his ongoing collaboration in this work and Joanna Leng (joanna.leng@man.ac.uk), MVC, University of Manchester for visualising the results of the analysis. This work was supported by EPSRC award number 98317397.

About the authors

Lee Margetts is a Research Associate at Manchester Computing, specialising in parallel finite element analysis and geotechnical engineering. He is currently working on a real time virtual prototyping project, involving interdisciplinary collaboration between the University of Lancaster, UMIST, Salford University and industry. Mike Pettipher is HPC User Services Team Leader, Manchester Computing, working both for the HPC Services section of the National Services Group and on projects managed by Manchester Visualization Centre (MVC). Both authors can be contacted at Manchester Computing, Kilburn Building, The University of Manchester, Manchester, M13 9PL, England and on email: lee.margetts@man.ac.uk and mike.pettipher@man.ac.uk respectively.

References

Carey, G.F., Harle, C., Mclay, R. (1997) *MPP solution of Rayleigh –Benard – Marangoni flows*. Technical Paper, Super Computing (SC97).

Margetts, L. (2002) *The convergence variability of BiCGStab(l) in parallel magnetohydrodynamics*. Proc 10th ACME Symposium, Swansea, pp5-9.

Morandini, J. Couvat, Y.dT. Masse, P. Gagnoud, A. (1994). *Modelling of coupled thermo-electro-magneto-hydrodynamic phenomena*. Int. J. Comp. Appl. Tech., 7, 176-184.

Pettipher, M.A. and Smith, I.M. (1997) The *Development of a MPP Implementation of a Suite of Finite Element Codes*. Computer Science, No 1225, pp400-409

Smith, I.M. and Pettipher, M.A (1997) *Finite Elements and Parallel Computation in Geomechanics*. Proc. ACME Symposium, London, pp141-144.

Smith, I.M. and Griffiths, D.V. (1998) *Programming the Finite Element Method*, Third Edition, Wiley and Sons.

Taylor,C.and Hughes,T.G. (1981) *Finite Element Programming of the Navier-Stokes Equation*. Pineridge Press Ltd, Swansea.