A Scalable Paradigm for Effectively-Dense Matrix Formulated Applications

G. Cheng, G. C. Fox, K. A. Hawick

Northeast Parallel Architectures Center
Syracuse University, 111 College Place,
Syracuse, NY 13244-4100, USA.
Tel. 315-443-3933, Fax. 315-443-3933,
Email: hawick@npac.syr.edu

Abstract

There is a class of problems in computational science and engineering which require formulation in full matrix form and which are generally solved as dense matrices either because they are dense or because the sparsity can not be easily exploited. Problems such as those posed by computational electromagnetics, computational chemistry and some quantum physics applications frequently fall into this class. It is not sufficient just to solve the matrix problem for these applications as other components of the calculation are usually of equal computational load on current computer systems, and these components are consequently of equal importance to the end user of the application. We describe a general method for programming such applications using a combination of distributed computing systems and of more powerful back-end compute resources to schedule the components of such applications. We show how this not only improves computational performance but by making more memory available, allows hitherto impracticably large problems to be run. We illustrate this problem paradigm and our method of solution with problems in electromagnetics, chemistry and physics, and give a detailed performance analysis of a typical electromagnetics application. We discuss a method for scheduling the computational components using the Application Visualisation System (AVS).

1 Introduction

Consider a typical problem in electromagnetics simulation involving the computation of a scattering field pattern from a conducting object. The scatterer could be as simple as an open-ended waveguide or as complex as a full aircraft. The general formulation of such a problem is to find some appropriate discretisation scheme to represent the geometry and to solve the appropriate equations. In this particular case, Maxwell's equations for the electric field must be solved subject to the appropriate boundary conditions [12].

Many application problems can be formulated as matrix equations including many of the field equations and partial differential equations arising in science and engineering. For some of these the problem is most efficiently expressed not as a fully stored matrix but as the field variables themselves. A typical example of this would be a wave equation in an elastic media stored as a displacement field and with a local updating algorithm used on the discretised mesh. The matrix formulation of this problem results in a multiply-banded matrix which would be wasteful of storage.

Some problems are so complicated in terms of the algorithm that they must be formulated as a full matrix which itself may have a sufficiently simple structure that some storage packing can be applied and a sparse matrix solver algorithm applied. Matrices like this often arise in computational fluid dynamics or structural engineering applications where finite elements or volumes are employed. The resulting matrix is by no means trivial but can be factored efficiently by an advanced sparse technique such as the frontal method [7]. In other problems it is not obvious how to exploit the sparsity for an efficient solution, and the matrix must be solved as though it was dense. It is this last case we wish to consider in more detail.

Parallel Solvers

On serial implementations and to a certain extent on traditional vector systems the matrix-solving component often dominates the application's computational complexity to such an extent that it is sufficient to focus all effort on this part of the application alone in terms of efficient coding and algorithms. A great deal of effort has been expended in writing fast efficient matrix factorisation and solver codes and libraries. There has been sufficient awareness of the importance of this problem that factorisation and solver code has been written for many of the parallel and high performance computing systems available today either as a portable library [4] or as proprietary code optimised specifically for one vendor's machine [22, 20]. Vendor-supplied code often has the additional capability to work with matrix sizes larger than the available machine memory. These so called out-of-core solvers typically make use of low level system facilities to be able to move blocks of the matrix between an attached high speed disk and the distributed node memory. Although the communications limitations inevitably degrades performance over an in-core solution, the capability of solving a very large problem size at all can be very valuable.

Generally these packages (for in-core solution) are excellent and can be applied very effectively to solve the matrix which lies at the heart of the application. However, by using such an efficient solver, and by reducing the time taken in the factorising and solving phases of an application, the other parts, which may have been very minor on a serial implementation, now become as significant in terms of compute time on a parallel system.

Computational Phases of an Application

A typical application code might have four identifiable phases: assembling the matrix; factorising the matrix; solving one or more right hand sides (RHS) using the factored matrix; and disassembling the solution into the desired form. If the problem is characterised by the matrix edge size n, which might be the number of elements, patches or nodes depending on the particular problem, then the number of computational operations in each of these will typically be of order n^2 for the assembly, n^3 for the factorisation, n^2 for the RHS solution and of order n or sometimes n^2 for the disassembly of the solution. On a serial implementation the time to completion for each of these phases of the computation is just proportional to the operation count. For a parallel computer system, processors can be added to reduce the effective powers of n in the factorisation and RHS solution phases by one. Our main point in this paper is that the matrix assembly and possibly the disassembly too, must be similarly reduced in completion time or the benefits of using the parallel factorisation and solve will be wasted.

The total time to completion for the application is generally the only matter of interest to the end user, and so it is necessary to consider all phases of the computation. In a particularly complex problem with an irregular mesh for example, the matrix assembly may in fact have the same power of n dependence as the factorisation, but may in fact be weighted by a higher constant if for example some particularly expensive operations such as square roots or trigonometrical functions are involved.

Problem	A	B	C	D
N	126	280	861	1087
A Assembly	4.598	22.711	158.362	2124.677
b Assembly	0.011	0.129	1.545	6.734
Factor	0.129	0.701	12.597	81.544
Solve	0.026	0.190	2.837	14.486
Disassembly	1.603	0.118	1.177	8.293

Table 1: Times (seconds) for components of a CEM Simulation Code

We illustrate this phenomena with some times (table 1) taken from running a method of moments computational electromagnetics code on an Intel iPSC. Our (prototype) parallel implementation uses a COMPLEX version of the ScaLAPACK library codes described in [4], The distribution of compute times for the various algorithmic components shows how the matrix factorisation is still dominated by the time to assemble the matrix at small to medium problem sizes. This is no longer true for very large problems sizes, but the regime of interest to us is close to the cross over point, and it is important to consider parallel implementations of the matrix assembly too.

In addition there are some applications which will require the solution of many medium sized matrices and where the matrix assembly will continue to dominate the completion time for the whole application.

Memory Considerations

A further consideration arises from the storage requirements of our problem. A typical implementation

of a matrix factoriser/solver will be able to partition the n^2 storage required to store the matrix itself across the distributed memory of its processors and thus allow a very large matrix to be stored. Some of the effort in achieving this will again be lost however, if the solution and RHS vectors are not also partitioned. Furthermore if the assembly and/or disassembly is not also similarly partitioned, then these phases of the computation must be carried out on a single compute node of the system. This is viable on some hybrid systems where there is a definite "master/slaves" relationship between the compute nodes of the system so that there is one or more nodes with extra memory. This is sometimes the case on systems which employ a so-called "front-end" which can be heavily configured with memory. Nevertheless it is likely that unless the overall machine's memory configuration can be easily changed or tuned for one particular problem, then some benefit of using a distributed memory parallel computer system will be wasted.

For very large systems which must use an out-of-core solver, these arguments are no longer so simple. In such a case the performance degradation due to communications between node memory and even the fastest available concurrent file systems is considerable, This degradation may affect the matrix assembly, factorisation, RHS solution and solution-disassembly in a very machine-dependent manner. This will be determined by the particular memory/disk/caching system and hierarchical structure on a given architecture. Characterising this and designing an optimal portable solution is a very difficult problem at present and we do not consider this further here.

Scheduling Computational Tasks

Our solution to this class of problems is to consider the scheduling of the phases of the computations on a hetero-architecture where distributed workstations for example are used in conjunction with a high performance computer system to overlap as much of the computation as possible and minimise the total time to completion.

It is often possible to partition the problem assembly phase into computationally distinct tasks. An important example concerns the assembly of the RHS vectors in contradistinction to assembly of matrix itself. It may be possible to farm out these tasks to separate compute nodes. This is particularly attractive on a parallel system which allows the compute nodes to act independently as well as in the loosely synchronous fashion necessary for the factorisation and solution phases.

Certain problems may also allow the matrix itself to be decomposed as a sum of sub-matrices, and it is also computationally advantageous to treat these sub-matrix assemblies as separate tasks.

The design goal is generally to achieve the lowest possible time to completion of the *whole* application code. This is generally consistent with making the most efficient use of *all* parts of the hetero-architecture including the concurrent file system if one is available *and* the distributed compute resources (front-end or additional workstations) *as well* as the main back-end compute engine. It is consequently necessary to exploit any task parallelism that presents itself as well as the data parallelism generally employed in the factorisation and RHS solution.

2 Application Visualisation System and Compute Modules

It is not desirable for the end-user of an application code to have to make extensive configurational changes to the code to allow its efficient execution on our proposed heterogeneous system. Indeed it is preferable that this scheduling be done at as high a level as possible within the application code, and be easy for the programmer to tune too.

Our approach involves using the Application Visualisation System (AVS) [1] to control the modular application components. This is illustrated in figure 3 and discussed below.

3 Industrial Applications Examples

A number of industrial applications that employ dense matrix methods are reviewed in [10]. In this section we illustrate the our paradigm using a specific electromagnetics problem. We also discuss the paradigm's applicability to problems in computational chemistry.

Computational Electromagnetics and Radar Cross-Sections

Electromagnetic scattering (EMS) is a widely encountered problem in electromagnetics [12, 15, 24], with important industrial applications such as microwave equipment, radar, antenna, aviation, and electromagnetic compatibility design.

Here, we consider one specific problem as illustrated in figure 1. Above an infinite conductor plane, there is

an incident EM field in free space. Two slots of equal width on the conducting plane are interconnected to a microwave network behind the plane. The microwave network represents the load of waveguides, for example, a microwave receiver. The incident EM field penetrates the two slots which are filled with insulating materials such as air or oil. Connected by the microwave network, the EM fields in the two slots interact with each other, creating two equivalent magnetic current sources in the two slots. A new scattered EM field is then formed above the slots. We simulate this phenomena and calculate the strength of the scattered EM field under various physical circumstances. The presence of the two slots and the microwave load in this application requires simulation models with high resolution and therefore a high performance computing system. The problem geometry is shown in figure 2.

The moment method [13, 14, 19] is used as the numerical model for the EMS problem, which can be represented as:

$$\{[\ Y^a\ ] + [\ Y^b\ ]\}\vec{V} = \vec{I}$$

$$[\ H\ ] = \mathcal{L}\{f\ (\vec{V}, \vec{M}, [\ H_0^2\ ])\}$$

where,

$[\ Y^a\ ]$: equivalent admittance matrix of the free space;

$[\ Y^b\ ]$: equivalent admittance matrix of the microwave network;

$\vec{V}$: coefficient vector;

$\vec{I}$: the excitation vector;

$\vec{M}$: a vector of mode functions;

$[\ H_0^2\ ]$: matrix of Hankel functions;

f : a function;

$\mathcal{L}$: a linear operator on f;

$[\ H\ ]$: final matrix of the simulated EMS field strength.

It is interesting to note that for this problem we can split the matrix factorisation into several model-specific parts. In particular from previous work [18] we note:

1. Calculations of $[\ Y^a\ ]$, $[\ Y^b\ ]$, $\vec{I}$, $\vec{M}$, and $[\ H_0^2\ ]$ can be done independently;

2. Computation of $[\ Y^a\ ]$, $[\ H_0^2\ ]$, and the linear solver for $\vec{V}$ have significant communication requirements and are computationally intensive;

3. $[\ Y^b\ ]$ is a sparse matrix and calculation of $\vec{M}$ requires little time. Calculation time for $[\ Y^a\ ]$, $[\ Y^b\ ]$ and $\vec{I}$ are relatively balanced.

Generally, we can decompose our problem onto a heterogeneous computing system or "metacomputer" that makes use of both functional parallelism and pipelining. In this application, functional parallelism consists of graphical I/O (i.e., user interaction, 3D rendering) and decomposed simulation computations which are handled concurrently by different components of the metacomputer. Pipelining combines calculations and communications among different processors or groups of processor that are carried out simultaneously in consecutive stages of the simulation. We have employed a heterogeneous computing system built from a number of SUN and IBM workstations and a Connection Machine [23]. Figure 3 shows a screendump of the scheduling and visualisation tool. The user can: interactively adjust the model parameters; and schedule the parts of the computation on computer components of the metacomputer; and observe the data fields of the model solution. The full details of this work are given in [2], but in brief, we find it is possible to build a robust simulation system where all the high performance computing components are utilised efficiently. We find the turnaround time for a simulation run is of the order of 8 seconds, which is quite satisfactory for this type of interactive code.

We observe that by allocating the computational components across different machines, we have been able to reduce the amount of memory that is used by for example workspace and also scalar quantities on the parallel architecture. This is important since it frees up more in-core memory to be used by the factorisation and solving code and allows larger problems sizes to be run.

Visualisation is very important in helping scientists to understand this problem under various physical conditions. In using an environment like AVS to "glue" our heterogeneous system components together, we also have the capability to incorporate a number of user-friendly interactive controls or "widgets" for adjusting input parameters and displaying output rapidly and meaningfully (See Figure 3). This can increase the efficiency of the scientist employing such a simulation tool.

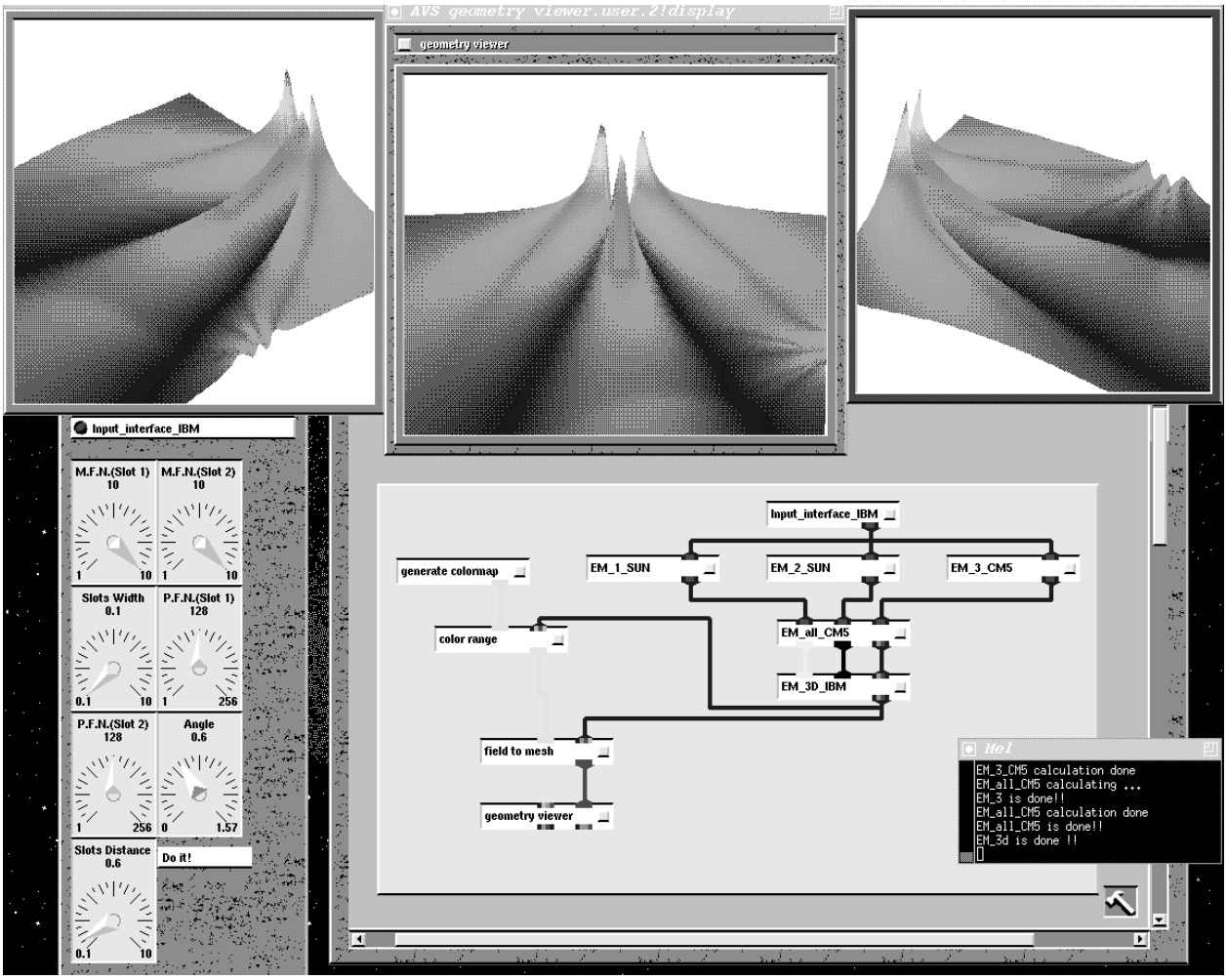

Figure 3: The Graphical User Interface on the Local Machine

Computational Chemistry and Quantum Physics

In the Rayleigh-Ritz variational method for solving the Schroedinger equation, the eigenvalues of the resulting matrix represent upper bounds of the exact eigenvalues and the solution vectors give the coefficients in a series expansion of the wave functions. In some applications there are additional problem constraints that make the matrix sparse to some extent. Nevertheless it is often the case that the sparseness is not easily exploitable and a dense solution method must be employed.

For such matrix eigenvalue problems, typically only the set of smallest eigenstates are actually of interest. These correspond to the lowest vibrational frequencies of a structure or to the lowest energy levels of a chemical molecule or an atomic nucleus. The computational problem is then to obtain these values from a matrix that may be very large and expensive to store as well as make computations on.

A key observation both for factorising/solving large matrices and also for finding the eigenvalues is that the computations can be set up in terms of matrix-vector products [6]. Furthermore, it is possible to implement these elemental operations of linear algebra using memory-saving pointer techniques for sparse (but irregular) systems, and also for distributed memory systems [5]. The Davison algorithm and its variants is based on use of elemental matrix-vector product operations [8, 9, 17]. Work at our own center has involved porting the MOPAC computational chemistry code [16] with techniques like this.

In such problems, the matrix assembly and disassembly can be considerably more computationally expensive than the elemental eigensolution algorithm. We believe there is considerable scope for combining heterogeneous compute-engines for simulation problems in computational chemistry where the set of algorithmic sub-components favour differing computer architectures.

4 Discussion

We have presented a range of application problems whose computational performance is split across more than one major component. While it may be possible to implement these components optimally on the same computer platform, this is often not the case. For such applications a heterogeneous computer system which can make use of functional parallelism, data-parallelism and pipelining.

We believe there are a number of matrix formulated problems that can benefit from the heterogeneous system scheduling and visualisation paradigm that we have presented.

We are presently working on a more general computational electromagnetics code which allows solution of a full range of CEM problems, and not just the slotted conductor problem presented above. We aim to implement this code on a range of distributed memory computing systems, and to experiment further with functional decomposition of the relevant assembly and disassembly application components. We are also considering different distributed data layouts for these components compared to that required by the matrix factoring algorithm.

We have already successfully applied this technique to other supercomputing applications problems such as financial modeling [3] and believe it is of broad importance to other industrial applications problems [11].

This general paradigm is well suited to rapid-prototyping certain simulation and modeling applications that require both interactive data visualisation and high performance computing. At the software environment level, this model only requires support of high level data visualisation and networking facilities. The technology for building such systems exists already in the form of products such as AVS[1] from AVS Inc. and Explorer[21] from Silicon Graphics Inc..

Finally we make the general observation that since the dominant factor in the cost of a parallel computing system is (at the time of writing) its memory, it is vital that this memory be used to best effect. Our paradigm allows the parallel system to store only those data that are necessary for its computations, and for other scalar data and data used in other parts of the computation to stored just once, on another part of our metacomputer.

References

[1] Advanced Visual Systems Inc. *AVS 4.0 Developer's Guide*, May 1992 and *AVS 4.0 User's Guide*, May 1992.

[2] G. Cheng, Y. Lu, G. C. Fox, K.Mills and T. Haupt, *An Interactive Remote Visualisation Environment for an Electromagnetic Scattering Simulation on a High Performance Computing System*, Proc. Supercomputing 1993, Portland, Oregon, November 15, 1993, PP317-326.

[3] G. Cheng, K. Mills and G. C. Fox, *An Interactive Visualisation Environment for Financial Modeling on Heterogeneous Computing Systems*, Proc. of the 6th SIAM Conference on Parallel Processing for Scientific Computing, March 1993, Norfolk, VA.

[4] J. Choi, J. J. Dongarra, R. Pozo, and D. W. Walker, *Scalapack: A scalable linear algebra library for distributed memory concurrent computers.* In Proceeding of the Fourth Symposium on the Frontiers of Massively Parallel Computation, PP 120-127. IEEE Computer Society Press, 1992.

[5] J. J. Dongarra and R. A. van de Geijn. *Two-dimensional basic linear algebra communication subprograms*, Technical Report LAPACK working note 37, Computer Science Department, University of Tennessee, Knoxville, TN, October 1991.

[6] J. J. Dongarra, R. A. van de Geijn and D. W. Walker, *A look at scalable dense linear algebra libraries*, in IEEE Proceedings of the Scalable High-Performance Computing Conference, PP372-379, IEEE Publisgers, 1992.

[7] I. S. Duff, A. M. Erisman, and J. K. Reid, Direct Methods for Sparse Matrices, Clarendon Press, Oxford 1986.

[8] E. R. Davison, *Monster Matrices: their eigenvalues and eigenvectors*, Computers in Physics, Vol. 7, No. 5, Sep/Oct 1993.

[9] E. R. Davison, J. Comput. Phys. 17, 87 (1975); Comput. Phys. Commun. 53, 49 (1989).

[10] A. Edelman, *Large dense numerical linear algebra in 1993: The parallel computing influence*, International Journal of Supercomputing Applications, 1993.

[11] G. C. Fox, *Parallel Computing in Industry: An Initial Survey*, in Proc. of Fifth Australian Supercomputing Conference, World Congress Centre, Melbourne, Australia, December, 1992.

[12] R. F. Harrington, *Time-Harmonic Electromagnetic Fields*, *McGraw-Hill Book Company*, New York (1961).

[13] R. F. Harrington, *Field Computation by Moment Methods*, *the Macmillan Co.*, New York (1968). Reprinted by *Krieger Publishing Co.*, Malabar, FL (1982).

[14] R. F. Harrington, *Matrix Methods For Field Problems*, *Proc. IEEE*, vol. 55, No. 2, pp. 136-149, Feb. 1967.

[15] E. C. Jordon and K. G. Balmain, *Electromagnetic Waves and Radiating Systems*, Second Edition, *Prentice-Hall, Inc.*, Englewood Cliffs, New Jersey (1969).

[16] G. von Laszewski, Parallelization of MOPAC, Northeast Parallel Architecture Center, Technical Report, September 14, 1992.

[17] B. Liu, in *Numerical Algorithms in Chemistry; Algebraic Methods*, edited by C. Moler and I. Shavitt (Lawrence Berkeley Laboratory, Berkeley, CA, 1978).

[18] Y. Lu, A. G. Mohamed, G. C. Fox and R. F. Harrington, *Implementation of Electromagnetic Scattering from Conductors Containing Loaded Slots on the Connection Machine CM-2*, Proc. of the 6th SIAM Conference on Parallel Processing for Scientific Computing, March 1993, Norfolk, VA.

[19] Y. Lu and R. F. Harrington, *Electromagnetic Scattering from a Plane Conducting Two Slots Terminated by Microwave Network(TE Case)*, Technical Report, TR-91-2, ECE Department, Syracuse University, August 1991.

[20] D. S. Scott and E. Castro-Leon, *Solving Large Out of Core Systems of Linear Equations using the iPSC/8609*, in Progress in Electromagnetic Research: Computational Electromagnetics and Supercomputer Architecture, edited by Tom Cwik and Jean Patterson, 1992.

[21] Silicon Graphics Inc. *Iris Explorer User's Guide*, 1992.

[22] Connection Machine Scientific Software Library (CMSSL), Thinking Machines Corporation, Cambridge, Massachusetts, June 1992.

[23] Thinking Machines Corporation, *The Connection Machine CM-5 technical summary*, Technical Report, Cambridge, MA, pp. 340-353, October 1991.

[24] J. Van Bladel and C. M. Butler, *Aperture Problems*, (Proc. NATO Adv. Study Inst. on Theoretical Methods for Determining the Interaction of Electromagnetic Waves with Structures,) Ed. by J. Skwirzynski, *Sythoff and Noordhoff international Publishers*, 1979.