A Scaleable Paradigm for Effectively-Dense Matrix Formulated Applications

G. Cheng, G. C. Fox and K. A. Hawick
Northeast Parallel Architectures Center,
Syracuse University, New York 13244-4100,
hawick@npac.syr.edu

(*Draft* Extended Abstract)

1 December 1993

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 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.

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 [Duff]. 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 [DongarraEtAl] or as proprietary code optimised specifically for one vendor's machine [TMC,Intel]. 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.

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) to control the modular application components.

3 Industrial Applications Examples

Computational Electromagnetics and Radar Cross-Sections

Computational Chemistry

4 Discussion

This present work is based on our experiences with matrix formulated problems in a variety of industrial applications. One of these is described more fully in the paper "An Interactive Remote Visualisation Environment for an Electromagnetic Scattering Simulation on a High Performance Computing System", by G.Cheng et al.

It is intended that the final revision of this current paper will present more of this industrial work in fully assimilated form.