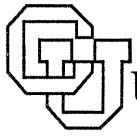
COMPUTER SCIENCE ASPECTS OF GRAND CHALLENGE COMPUTING

C.F. Baillie and O.A.McBryan

CU-CS-702-94 January 1994



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

COMPUTER SCIENCE ASPECTS OF GRAND CHALLENGE COMPUTING

CU-CS-702-94 January, 1994

C.F. Baillie and O.A. McBryan

Department of Computer Science
University of Colorado at Boulder
Campus Box 430
Boulder, Colorado 80309-0430 USA

COMPUTER SCIENCE ASPECTS OF GRAND CHALLENGE COMPUTING

C.F. Baillie and O.A. McBryan
Department of Computer Science
University of Colorado
Boulder, CO 80309, USA

ABSTRACT

We describe computer science and computational science aspects of three major interdisciplinary grand challenge efforts currently underway at the University of Colorado. These challenges involve both fluid and structure mechanics and a mix of both regular and highly irregular grids. Computer science challenges include performance monitoring, better runtime support, automatic code generation for various MPP, protocols for high-speed networking, and optimization of communication and I/O to mass storage.

INTRODUCTION

The University of Colorado is involved in three major interdisciplinary grand challenge efforts covering both CFD and Structural Mechanics. NSF HPCC Grand Challenge funding is supporting work in Coupled Fields and in Geophysical/Astrophysical Turbulence, while a NASA HPCC award is supporting work on Rotationally Constrained Turbulence. We will focus here on the computational science aspects of these grand challenges.

The Coupled Fields Grand Challenge is oriented toward the solution of problems involving interacting "fields" using heterogeneous approaches - including heterogeneity in the algorithms, computational methods and hardware utilized. For example, researchers in Aerospace Science, led by Charbel Farhat, are developing full aeroelastic simulations. These simulations model both a structure and the fluid in which it is immersed, and incorporate interactions between these systems. Computationally it has been shown that the fluid part of the simulation is readily amenable to massively parallel processing, whereas the structure part, which involves highly irregular grids, is more suited to a vector supercomputer or a shared-memory machine. The strategy then is to use heterogeneous parallel computing with the fluid simulation running on one machine and the structure on another.

The Geophysical/Astrophysical Turbulence and Rotationally Constrained Turbulence Grand Challenge efforts involve simulations of both rotating and non-rotating convective turbulence, such as are found in the earth's oceans, in the fast zonal jets of the giant planets and in the convective zone of the sun. This work involves researchers from the Dept of Astrophysical, Planetary and Atmospheric Sciences, and from NCAR, University of Minnesota and University of Chicago. The codes here tend to be of the pseudo-spectral type.

COUPLED FIELDS

The full aeroelastic problem involves a strong coupling of the fluid and structure codes in a time-dependent computation. The idea is to "fly" an aircraft structure for a significant time, rather than simply determine the airflow of a fixed aircraft profile. The critical point is that an aircraft frame is a flexible structure, not a rigid body. This fact is familiar to anyone who has looked out of the window during a flight - the wings obviously oscillate up and down. Understanding and simulating these perturbations of shape is an essential part of a full numerical wind-tunnel simulation. Indeed it is perfectly possible that a plane could appear to fly well based on a single rigid body simulation, but yet when flexibility is included, the wing oscillations would be sufficiently strong to destroy the structure.

The fluid and structure parts of the aeroelastic prob-

Table 1: Times to solve a typical fluid problem in seconds and speedups on the KSR1 for a fluid grid with 131,035 vertices.

Number of Processors	KSR1	
	Time	Speedup
1	70.230	1.0
8	8.200	8.6
12	5.085	13.8
20	3.290	21.3
32	2.185	32.1
48	1.585	44.3

Table 2: Times to solve structure problem in seconds and speedups for structure with approximately 50,000 equations on various numbers of processors of the C-90 and KSR1.

Number of Processors	C-90		KSR1	
	Time	Speedup	Time	Speedup
1	44.1	1.0	3272	1.0
2	23.6	1.9	1601	2.0
4	12.3	3.6	935	3.5
8	7.8	5.6	573	5.7
16	-	-	358	9.1
32	-	-	262	12.5

lem are fairly conventional algorithms. Each has been developed by the aerospace researchers in our team and both have been parallelized for several architectures, including CRAY YMP-8 and C-90, Intel iPSC and Paragon, and Thinking Machines CM-200 and CM-5 (Lanteri and Farhat 1993).

The fluid code involves an explicit time step which is therefore very fast. Furthermore fairly regular grid structures can be utilized here, so that the problems of parallelizing this code are relatively straightforward. Over several years this code has been ported to a range of scalar, vector and MPP machines. Results of these studies show that this code runs well on various distributed memory systems such as the KSR1, Intel Paragon and CM-5. Table 1 shows typical fluid code results from a KSR1 computer. These performance numbers show a substantial loss of potential performance relative to peak rate. Primarily this is due to the irregular nature of the grids used. For regular grids, much higher performance can be attained - for example we have demonstrated performance as high as 25 Gflops on the 1024 node CM-5 computer for Shallow Water type equations (McBryan 1993).

The structure code is a complex finite element code. The code is either implicit or at least locally implicit, and grids are highly irregular. Furthermore different types of elements are needed in different areas of the structure. All of this makes for a code that is quite difficult to parallelize on an MPP. This code has also been ported to a range of systems over several years. Results of these experiments indicate that the structure code behaves much better on the conventional vector supercomputers such as CRAY C-90 than on message passing MPP systems. Among MPP machines, the structure code runs best on the Kendall Square Research KSR1. Table 2 shows typical results for parallel versions of the structure code on the C-90 and KSR1 supercomputers.

The full aeroelastic simulation requires a coupled solution over a large number of timesteps. Starting with a suitable initial frame, and an initial velocity, the initial airflow is computed and determines the pressure on all parts of the plane body. These forces produce a deformation of the structure, which is calculated using the structure code. This results in a new shape for the aircraft, which is used to compute a new flow, and in turn a new deformation and soon. Thus the computation reduces to a large number of timesteps, each involving separate fluid and structure components. The only flow of information between these two phases occurs on the boundary of the aircraft and consists of exchange of the shape and pressure data. Computationally there is actually a third factor: the numerical grids used for the fluid and structure need to match at the shared surface. This means that these grids are all time-dependent and in a sense there are then three coupled quantities - fluid, structure and grid. However we will ignore that issue here.

Because the fluid codes are solved explicitly, it is necessary to take a very short timestep. The struc-

ture codes however can accommodate a much longer timestep because they are solved implicitly. Furthermore the changes in pressure forces on the aircraft resulting from single fluid timesteps are so small that their effect on aircraft shape is not significant. Experiment has shown that one gets good results by performing a structure timestep once every 100 fluid timesteps. Furthermore, with realistic grid sizes, the structure computation takes about 100 times as long as a fluid step. Consequently there is the possibility for a very well balanced computation in which the fluid steps are run on one machine and the structure computations are overlapped in time on a different machine.

The exchange of data between the phases occurs only on the boundary. If there are $O(N^3)$ grid points in the fluid, where N is a spatial resolution parameter, the boundary will have an area of $O(N^2)$, and consequently for high resolution computations the data exchange between the two phases will take much less time than even the explicit fluid timestep. Of course the value of N above for which this statement is true depends strongly on the characteristics of the communication network - specifically on the latency and bandwidth available. Introducing practical values for the various computational parameters involved indicates that bandwidths in the range of hundreds of Mbits/sec will be needed.

Our Grand Challenges Group has now developed an heterogeneous version of the aeroelastic code, based on PVM to provide the underlying communication. We are testing this program using three machines that are located in Boulder - a 208-node Intel Paragon, 64-node KSR1 and a 32-node CM-5. Both the fluid and structure codes run on all three machines, and we have developed and tested a matcher code that provides the required communication of boundary information between the two phases. We have tested all of the phases running on all combinations of pairs of parallel machines and are currently measuring time spent communicating the aircraft boundary data.

Current experiments are based on 1.5 Mbit T1 lines that interconnect the three supercomputer locations in Boulder. This lines are adequate for testing the software, but are far too slow to provide an effective rate of interface data exchange. Spurred by this challenging computation we have recently developed plans with USWest, the local telephone franchise, to interconnect our three supercomputers with a dedicated 150 Mbit/sec ATM network. The network will become operational in February 1994, and we will report initial measurements in the conference presentation at the SCS meeting in La Jolla in April.

Challenging issues related to the ATM network will include the effectiveness of various communication protocols both at the low level (for example IP, raw ATM, ...) and at the high level (for example PVM, MPI, ...). We intend to develop software that will allow heterogeneous computations of this scale to be monitored while running. Other challenges are related to I/O - for example there are not large mass storage facilities at all of these supercomputer locations. Addressing the problems of getting I/O from the supercomputers to remote storage facilities will become critical once the network communication is optimized. Obviously the ATM network will again play a critical role here. In the longer term we would like to generate heterogeneous code automatically using some of the same tools currently in use for the turbulence project discussed below. Ultimately we would like to maintain a single source version of the aeroelastic code, and generate automatically versions for individual MPP systems, as well as heterogeneous versions for various configurations.

TURBULENCE

All of the pseudo-spectral turbulence codes were originally running on Cray's. They contained a few directives to help the compiler multitask them. We parallelized them for the KSR1 cache-only shared-memory machine using program transformation tools based on Sage++ (Gannon *et al* 1993) to optimize single processor performance and help generate parallel "tiling" directives. We shall describe one such code in detail, giving performances for the original Cray version on the C-90 and the parallel KSR1 version.

The code simulates turbulent compressible convection in a three-dimensional rectilinear geometry employing a hybrid pseudo-spectral/finite-difference spatial discretization and a mixture of implicit Crank-Nicolson and explicit 2-level Adams-Bashforth schemes for time-stepping the linear and non-linear terms of the underlying equations respectively (Cattaneo *et al* 1990). The pseudo-spectral nature of the code means that Fourier transforms (FFTs) are performed in the two horizontal spatial directions and finite-differences in the third vertical direction. Since FFTs incur long-

range communications whereas finite-differences have only short-range, the data is decomposed for parallel processing by splitting up the domain into horizontal planes, assigning a number of planes to each processor so as to keep the FFT communications local in processor memory. This is done both for the C-90 and the KSR1.

There are two main steps in parallelizing code for a machine like the KSR1: optimizing single processor performance and adding parallel "tiling" directives (Baillie, Macdonald and Sun 1993). Single processor performance optimization is necessary because as is common in all mirco-processors today, the KSR1 custom processor chip contains a cache, called the "subcache" in order to distinguish it from the 32 MByte local cache. This subcache is physically 2-way associative. This means that addresses separated by certain "magic numbers" get mapped into the same area of the subcache which can hold at most only two of them. Thus if several of these addresses are accessed by the program one after another, it gives rise to the problem known as "subcache thrashing" which seriously degrades performance. This pattern of address reference is in fact exactly what occurs in a Fortran program stepping through the second (or higher) dimension of an array (since Fortran arrays are stored in column-major order). The trivial fix for this problem is, of course, not to have arrays whose first dimension size leads to a magic number. This is most easily done by picking odd numbers which are not a power of two. Coincidentally this is precisely what is done in codes designed for vector computers like the Cray but for a different reason, namely in order to avoid memory bank conflicts. Therefore the turbulence codes we are working with already avoid this problem.

The second step is to add parallel "tiling" directives. Tiling on the KSR1 is just splitting up nested loops into blocks. This is similar to "domain decomposition" for distributed memory machines, except that there the data must be explicitly decomposed among the processors' local memory, whereas on the KSR1 the data is simply put in the global memory initially and becomes decomposed automatically as the code runs by the on-demand caching strategy. This causes the first iteration of the algorithm to be slower than the subsequent ones, but since many thousands of iterations are required this is a negligible overhead. Of the four possible methods of tiling (slice, mod, grab and wavefront), the slice method was utilized, as previously mentioned by sending multiple horizontal planes to each processor. Thus the tiled code for Fourier transforming a variable $a(nx, ny, nz)$ in real space to $b(ny, nx, nz)$ in phase space looks like:

```
c*ksr* tile(k,tilesize=(k:ksize))
       do k = 1,nz

c fft first dimension: a -> temxy
        call fft(a(1,1,k), temxy, nx, ny)

c swap dimensions: temxy -> temyx
        do i = 1,nx
         do j = 1,ny
          temyx(j,i) = temxy(i,j)
         end do
        end do

c fft second dimension: temyx -> b
        call fft(temyx, b(1,1,k), ny, nx)

       end do
c*ksr* end tile
```

We see that the horizontal 2d Fourier transform is done as two 1d FFTs which entails a swap of the first and second array dimensions inbetween. Note that the tile statement specifies the number of horizontal planes per processor as $tilesize = (k : ksize)$. For maximum efficiency the tiled loop should be the outermost and the variable which is tiled should be the last dimension of the arrays.

The resulting parallel tiled code beats a single Cray Y-MP processor, and is a little slower than a single C-90 processor, using 32 processors of the KSR1. In Table 3 we give times per iteration and speedups for the code running on various numbers of processors of the Cray C-90 and KSR1, for problem size $96 \times 96 \times 65$. The Cray C-90 speedups are not very impressive but this is mainly due to the fact that the runs were not done in "dedicated mode". On the KSR1 we see superlinear speedups due to the fact that the data does not fit in one processor's memory. However since the KSR1 is a shared memory machine the code still runs with the data being swapped to the memory of other processors.

To better understand what is going on we calculate the "incremental speedups": in going from 4 to 8 pro-

Table 3: Times per iteration in seconds and speedups for $96 \times 96 \times 65$ problem on various numbers of processors of the C-90 and KSR1.

Number of Processors	C-90		KSR1	
	Time	Speedup	Time	Speedup
1	1.96	1.00	124.8	1.00
2	1.09	1.85	58.2	2.14
4	0.74	2.80	20.0	6.24
8	0.62	3.39	9.37	13.3
16	0.48	4.56	4.96	25.2
32	-	-	2.94	42.4

cessors this is 2.13, from 8 to 16 it is 1.89 and from 16 to 32 1.69. Thus for larger numbers of processors the speedup falls off. We have attributed this fact to the amount of time spent in "barriers". Barriers are those events which synchronize the processors at the end of parallel loops before entering a serial section or another parallel loop. If the load balance across processors is not perfect then some processors will have to wait in the barrier for others to finish. On examining the barrier performance of the KSR1, it appears as though the barrier can wait even though all processors have completed their individual tasks, and that this extraneous wait time can increase linearly with the number of processors used. Eventually, the time spent in barriers starts to be a significant percentage of the calculation time, for example about 20% for 32 processors. Therefore our computer science department colleagues have developed more efficient barriers for the KSR1 (and other shared-memory machines), which we hope will be incorporated into the KSR1 compiler in the near future (Grunwald and Vajracharya 1993).

ACKNOWLEDGMENTS

The work reported here was supported by NSF Grand Challenge Applications Group Grant ASC-9217394 and by NASA HPCC Group Grant NAG5-2218.

REFERENCES

Baillie, C.F., A. E. Macdonald and S. Sun. 1993. "Porting the Quasi-Nonhydrostatic Meteorological Model to the Kendall Square Research KSR1." In *Proc. High Performance Computing in the Geosciences*, F.-X. Le Dimet, ed. Kluwer Academic, Amsterdam.

Cattaneo, F., N.H. Brummell, J. Toomre, A. Malagoli and N.E. Hurlburt. 1990. "Turbulent compressible convection." *Astrophys. J.* **370** (March): 282-294.

Gannon, D., F. Bodin, S. Srinivas, N. Sundaresan, S. Narayana and J. Gotwals. 1993. "Sage++, An Object Oriented Toolkit for Program Transformations." Technical Report. Dept of Computer Science, Indiana University, Indiana (Dec).

Grunwald, D. and S. Vajracharya. 1993. "Efficient Barriers for Distributed Shared Memory Computers." Technical Report. Dept of Computer Science, University of Colorado, Colorado (Nov).

Lanteri, S. and C. Farhat. 1993. "Viscous Flow Computations on MPP Systems: Implementations and Performance Results for Unstructured Grids." Siam JSC, to appear.

McBryan, O. 1993. "Performance of the Shallow Water Equations on the CM-200 and CM-5 Parallal Supercomputers." Proceedings of the Fifth ECMWF Workshop on the use of Parallel Processors in Meteorology: Parallel Supercomputing in Atmospheric Science, eds. G.-R Hoffman and T. Kauranne, World Scientific, London.