Performance of a 3D Spectral code on the Cray T3D and IBM SP2 parallel supercomputers

Clive F. Baillie and John Wilson

Department of Computer Science
University of Colorado, Boulder, CO

Nicholas H. Brummell

Joint Institute for Laboratory Astrophysics
University of Colorado, Boulder, CO

Joe Werne

National Center for Atmospheric Research, Boulder, CO

We have been running both 3D pseudo-spectral and 3D fully spectral codes on the Cray C-90 and KSR1 shared memory supercomputers for some time. With the emergence of the new generation of distributed memory supercomputers, in particular the Cray T3D and IBM SP2, we are updating our codes to take advantage of such machines. In order to estimate the performance we might expect, we have written a parallel code for the kernel of our algorithms – the 3D fast Fourier transform (FFT) – and benchmarked it. Here we present the initial results.

It is almost universally accepted that the best way to implement any multi-dimensional FFT on a parallel computer with distributed memory (i.e. where the cost of communicating with other processors is relatively high) is to perform FFTs in-processor always, thus requiring across-processor transposes of the data in order to make the other dimensions local when necessary. This methodology ensures that the FFT executes at maximum performance and removes the burden of writing a parallel FFT at the cost of the global data migration required for a transpose.

For our codes, we can arrange the 3D FFT as 3 1D FFTs requiring only one parallel (across-processor) transpose and one serial (in-processor) transpose, as follows. Suppose that in real space the 3D data is stored as (x,y,z), and in Fourier space as (kz,ky,kx). Then, for example, the forward 3D FFT from real to Fourier space looks like

```
               FFT                TRANSP                FFT
a(x,y,z)      =====>  b(kx,y,z)   =====>  a(y,kx,z)    =====>

              TRANSP                FFT
b(ky,kx,z)    =====>  a(z,ky,kx)  =====>  b(kz,ky,kx)
```

(and the inverse 3D FFT is exactly the reverse of this). Note that the FFT always transforms the first index utilizing cache to its full extent and leaving the last two dimensions free for parallelization. Since we shall be running large problems, on the order of 512^3 on roughly 256 processors, we only need to parallelize the last index. Hence, for the first 2 FFTs the z-dimension of the data is decomposed across processors whilst for the last FFT it is parallel in the x-dimension. Therefore the first transpose is serial (in-processor) and the second is the only parallel operation in which communication between all processors occurs.

Since this latter global communication will be the bottleneck of such algorithms, we have written an efficient parallel transpose using the low-level `shmem_get` communication routine on the T3D, and using the fast implementation of MPI (called MPI-f) on the SP2. The transpose is written in such a way as to minimize the number of messages sent, since it is the latency of message passing which dominates in these latest parallel machines with very fast (hardware) interconnects.

In the Tables below we list the wall-clock time in seconds for one FFT, for the serial transpose, for the parallel transpose, and for the whole 3D FFT, running on different numbers of processors with problem sizes of 256^3 and 512^3. The first two tables are for the Cray T3D and the other two for the IBM SP2. For comparison, the code takes 11.3 seconds for 256^3 on one processor of the Cray C-90. For 512^3 it is slightly too large (i.e. more than 256MW) to run.

We see that for the 512^3 problem on 64 processors, the SP2 is about twice as fast as the T3D. However the code scales better on the T3D and so ultimately, with the larger number of processors available, the T3D can operate at almost twice the speed. Notice that the T3D contends and even exceeds the full speed of a 16 processor C-90 when more than 64 processors are used. Lastly we note that the SP2 time for 256^3 on 32 processors is much larger than one would expect. This is due to the fact that each of the processors in the SP2 is actually a fully configured workstation and the interconnection network between them is a complicated switch. This creates a great variability in run times even for the same program run on the same processors. When we have time on a dedicated SP2 we will re-run these cases to get a better result.

Number of Processors	FFT	Sequential Transpose	Parallel Transpose	Total
32	0.357	0.121	0.288	1.568
64	0.212	0.060	0.150	0.897
128	0.141	0.040	0.076	0.575
256	0.107	0.030	0.055	0.431

Times for 256^3 3D FFT on Cray T3D

Number of Processors	FFT	Sequential Transpose	Parallel Transpose	Total
64	1.827	0.435	1.173	7.041
128	1.096	0.261	0.614	4.134
256	0.731	0.174	0.420	2.766

Times for 512^3 3D FFT on Cray T3D

Number of Processors	FFT	Sequential Transpose	Parallel Transpose	Total
4	1.003	0.290	3.314	6.775
16	0.273	0.078	0.541	1.451
32	0.152	0.044	1.696	2.204
64	0.091	0.026	0.721	1.050

Times for 256^3 3D FFT on IBM SP2

Number of Processors	FFT	Sequential Transpose	Parallel Transpose	Total
16	2.174	0.587	6.348	13.477
32	1.139	0.250	3.024	6.835
64	0.633	0.171	2.310	4.406

Times for 512^3 3D FFT on IBM SP2