

Conversion of Parallel Directives in the CSIRO Global Climate Model (GCM) to OpenMP Form

Student(s): Keith Hsuan

Association: Victorian Partnership for Advanced Computing

Supervisor(s): Steve Quenette (VPAC), Robert Bell (CSIRO)

Discipline: Ocean and Climate Modelling

Research Objective

This project was a collaborative effort between VPAC and CSIRO's High Performance Computing and Atmospheric Research divisions. It involved the conversion of the Global Climate Modelling Code from using old automatic parallelisation directives to the newer OpenMP standard.

Motivation / Significance

CSIRO's Global Climate Modelling (GCM) code has been developed over a number of years for a variety of platforms. As such, the code has incorporated automatic parallelisation directives from each of the platforms; namely NEC, Cray, SGI and Fuji. By removing these legacy directives and replacing them with OpenMP directives, we hope to increase the portability of this code. However, as the code has been optimized specifically for vector based machines, it is likely that the performance of the code will be quite poor on clusters such as Grendel.

The GCM code was written to model global climate patterns, and calculates the effects of the sun, vegetation and ice over time. It is written such that at each time step, factors affecting climate are calculated across each section of the world, and the model updated. As such, the code is best parallelised at the loop level, with each loop representing a section of the globe.

Science Background

While the parallelisation of the GCM code would have been possible, and more efficient using an explicit MPI scheme, it was decided that using an automatic parallelisation scheme such as OpenMP would be ideal.

This would allow the conversion to remain simple, as the OpenMP API is based on Cray automatic parallelisation directives, which were already present in the code.

The OpenMP parallelisation scheme operates by beginning execution on a single "master" thread, with "slave" threads being created when the program enters a parallel construct. These slave threads are then remerged with the master thread upon exit of the section. In the GCM code, all parallelisations are done at the loop level, with each thread completing a single iteration of the loop. Care must be taken in these cases that the result of an iteration does not depend on a previous result – for these situations, the OpenMP API has included the construct such as ORDERED to ensure that each iteration of the loop is executed in the same order as a serial code.

Methodology

- Translate legacy directives to OpenMP naively;
- Test translation on CSIRO's SX-5 and verify;
- Profile program and increase parallelisation where possible;
- Port to other machines (Grendel, SGI)

Coding Aspects

The GCM code was written predominantly in Fortran-77, with several routines in Fortran-90 and a single C subroutine. The OpenMP API was written specifically for Fortran programs, thus very little extra coding was required beyond that of the actual translation.

SI0307- VPAC Case Study

In general, the GCM code used three main libraries beyond those associated with OpenMP; these were the NetCDF library, which is a data-handling format, LAPACK, and SciLib, both of which are used for mathematical analysis. While all these libraries were available on the CSIRO SX-5, the port to Grendel required the substitution of SciPort over LAPACK and SciLib, both of which are Cray proprietary libraries.

Results

The conversion of the GCM code was completed, and the code was tested successfully on both the NEC SX-5 at CSIRO, as well as on Grendel at VPAC. However, the port to the SGI Altix was incomplete; although the code was operational serially, the parallel version failed due to a segmentation fault. Although a similar problem had occurred during the original conversion, this had been fixed and tested on both the SX-5 and Grendel; it is unknown why there is a problem on the Altix.

After the conversion, the two fully successful implementations were tested against the original code to ensure numerical accuracy was maintained, and to determine if there was any performance benefit or degradation in the translation. For this, test cases of 250 timesteps and 1000 timesteps were run with all three versions of the code, on 1, 2 and 4 processors, while the 1000 timestep case was also tested across 8 processors on the SX-5, as Grendel does not support SMP programs requiring more than four processors. The results for a single run of each case are summarised below:

250 Timesteps			
	1	2	4
Original	795.7	430.1	238.8
OpenMP	786.6	446.2	246.8
Grendel	4558	2731	

1000 Timesteps				
	1	2	4	8
Original	3076	1740.6	896.8	600.9
OpenMP	3079.6	1740.4	936.3	571.3
Grendel		10835	6677	

Also, CPU utilisation times were obtained from a single run of the 1000 timestep, 8 processor case:

Min. CPUs used	1	2	3	4
Original	600.9	570.4	568	566
OpenMP	571.3	544.3	543.9	543.6
Min. CPUs used	5	6	7	8
Original	561.5	541.1	474.3	315.8
OpenMP	543	540	522	406.9

Discussion:

The first result which is interesting is the exceptionally poor performance of the code on Grendel, as opposed to that on the SX-5. This can largely be explained by the differences in architecture of the two machines; the SX-5 is a vector based machine, while the AlphaServer SC is scalar. As the code was originally written with operation on SX-5 in mind, it is optimised specifically for this type of architecture, causing a large degradation in performance. However, the performance difference between the original code and the OpenMP conversion appears to be very slight; although the 1000 timestep case too approximately 5% longer on four processors for the OpenMP version, the same case ran slightly faster over eight processors. It was expected that we would see some slight performance degradation in the OpenMP conversion since we were no longer using the native format. Conversely, since OpenMP is supported by the proprietary Fortran compiler supplied by NEC, it can be assumed that OpenMP support is reasonably well implemented.

A potential problem with OpenMP style parallelisation schemes is the difficulty of load balancing; that is, to ensure that all processors are performing useful work at all times, and are not waiting for extra information. It appears that this is not a large issue with our parallelisation, as the CPU use times for the 1000 timestep case seem to be reasonably high, at around 90% for most cases. However, for the eight processor test, the last two processors in the original code, and the last processor in the OpenMP version are substantially less utilised than the others. This may be due to number of iterations the main, parallelised loops are executed in the code. If the number of iterations is not divisible by eight, one would then expect that one or more of the processors would be starved of data at some times.

Conclusion

The conversion of the GCM code to utilise the OpenMP standard instead of individual compiler directives was completed successfully, and verified on the NEC SX-5 as well as VPAC's Grendel cluster. However, a further port to an SGI Altix was incomplete. While performance across platforms varied significantly due to architecture, it was found that there was little performance difference

SI0307- VPAC Case Study

between the original version of the code and the new OpenMP version. Further, it was found that load balancing was reasonable, with most processors being utilised for at least 90% of the allocated time.

References

OpenMP website: <http://www.openmp.org>