Meteorology 5344, Fall 2007
Computational Fluid Dynamics
Dr. M. Xue

Computer Problem #l: Optimization Exercises

Due Thursday, September 20

Exercise 1.

This exercise is designed to acquaint you with the basics of the OSCER Topdawg Linux Supercomputer (topdawg.oscer.ou.edu) for running programs in single processor and shared-memory parallel (SMP) mode. You will also experience and learn fundamental techniques of code optimization.

Topdawg is made up of over 500 2-CPU Xeon nodes. Additional details on the system can be found at http://www.oscer.ou.edu/docs/hardsoft_dell_cluster_pentium4xeon.php.

Log onto Topdawg (ssh topdawg.oscer.ou.edu -l your_login), then enter

```
cd
cp –rp /home/mxue/cfd2007 .
cd cfd2007
```

Compile Fortran program hw1.f90 using Intel Fortran compiler ifort, with the following sets of options separately:

```
ifort –O0 -o hw1_1cpu.exe hw1.f90
ifort –O1 -o hw1_1cpu.exe hw1.f90
ifort –O2 -o hw1_1cpu.exe hw1.f90
ifort –O3 -o hw1_1cpu.exe hw1.f90
```

Run executable hw1_1cpu.exe using the following command, for each set of the compilation options used above. The program will print out CPU times used by various sections of code in the program. The ‘date’ command gives the current wall clock time, and the time difference before and after the hw1 execution approximately gives the wall clock time used by your job. The CPU times used should be the total of all CPUs used when more than one CPUs are used.

```
date; hw1_1cpu.exe; date
```

Recompile hw1.f90 with automatic share-memory parallelization turned on. Run the job using 1 or 2 CPUS (or number of threads).

```
ifort –O3 –parallel -o hw1_smp.exe hw1.f90

setenv OMP_NUM_THREADS 1
date; hw1_smp.exe; date

setenv OMP_NUM_THREADS 2
```

```
date; hw1_smp.exe; date
```

Consult the man pages of ifort (man ifort) for information on the compiler options.

Collect the CPU and wall clock timings by running each command 5 times and take the average. Make table(s) of the CPU times used by all sections. Examine carefully the structure and content of the code sections and the similarities and differences among codes that do the same or very similar things. Discuss the timing results in the context of possible optimizations such as superscalar operations, pipelining, vectorization, (automatic) shared-memory parallelization, memory access pattern, cache utilization, subroutine inlining, and any other observations that you feel important or interesting.

Note that when you run the above commands interactively on the login node of Topdawg, the wall clock time will be affected by the machine load at the time. It's best that you pick a time when the login node is light. 'top' command shows the active processes.

Exercise 2.

This exercise is designed to help you gain some handson experiences running a large atmospheric prediction model, the Advanced Regional Prediction System (ARPS, http://www.caps.ou.edu/ARPS), in single CPU, 2-CPU SMP and multi-CPU DMP modes, on a super-scalar DSM parallel system with shared-memory nodes, and to help you understand certain optimization and parallelization issues.

The MPI version of ARPS uses the horizontal domain decomposition strategy discussed in class. The shared-memory parallelization relies on Intel compiler's automatic parallelization capability, which performs loop-level parallelization by analyzing the code.

Step 1: Log onto Topdawg. Copy ARPS source code package into your home directory, unzip and untar the package.

```
ssh topdawg.oscer.ou.edu -l your_login
cd
cd cfd2007
cp /home/mxue/arps5.2.8.tar.gz .
gunzip arps5.2.8.tar.gz
tar xvf arps5.2.8.tar
```

The ARPS package, arps5.2.8.tar.gz was downloaded from the ARPS web site at http://www.caps.ou.edu/ARPS.

Step 2: Compile and build several versions of ARPS executable.

```
cd arps5.2.8
makearps clean          ! clean off existing object codes and executables if any.
                          'makearps help' lists a set of options for makearps.
makearps arps           ! builds arps executable using default (usually
                          high)optimization level. Watch and note the
                          compilation to see what compiler options are used.
```

```
                          'man ifort tells you what those options mean. The
                          executable is bin/arps.
mv bin/arps bin/arps_highopt  ! rename the arps executable

makearps clean          ! clean off existing object codes
makearps –opt 0 arps    ! builds arps executable with minimum optimization
mv bin/arps bin/arps_noopt  ! rename the arps executable

makearps clean          ! clean off existing object codes
makearps –p arps        ! builds arps executable with automatic shared-memory
                          parallelization. Again what the compilation to see
                          what compiler options are used and compare with those
                          used by the first compilation with default compilation
                          level. Note the main difference.
mv bin/arps bin/arps_omp  ! rename the arps executable

makearps clean          ! clean off existing object codes
makearps arps_mpi       ! builds the distributed-memory parallel version of
                          ARPS, using MPI. The executable is bin/arps_mpi.
```

Now all executables you need are built. Do 'ls –l bin' to be sure.

Step 3: Copy a directory containing example batch scripts (*.cmd), arps input (*.input) and sounding data (may20.snd) files, into your ARPS directory:

```
cd
cd cfd2007
cd test
```

Inside directory test, you will find files named *.input which are the input files contains ARPS configuration parameters. These files are configured to make identical simulations of a supercell thunderstorm for 2 hour, using a 67x67x35 computational grid (set by nx, ny and nz in *.input). For MPI runs, the domain decomposition is specified by parameters nproc_x and nproc_y. For example, in arps_mpi4cpu.input, nproc_x and nproc_y are set to 2, i.e., the computational domain is divided into 2x2 subdomains and distributed over 4 processors. nproc_x=1 and nproc_y=4 or nproc_x=4 and nproc_y=1 should also work although the efficiency may be different because the innermost loops (usually for i index in the x direction) have different length.

Step 4: Examine and submit batch scripts

Three batch scripts are provided in directory test called *.bsub, that run 8 different ARPS jobs, using the ARPS executables created earlier, there were compiled with different optimization/ parallelization options, and they will use different number of processors, in single-CPU, shared-memory or distributed-memory MPI mode.

Enter

```
bsub < arps_1node.bsub
```

to submit a batch job that contains that following ARPS jobs using a single node, with 1 or 2 CPUs, with different levels of optimization.

```
#Run ARPS executable compiled without automatic shared-memory parallelization
and with minimum optimization

setenv OMP_NUM_THREADS 1
date; ~/cfd2007/arps5.2.8/bin/arps_noopt < ~/cfd2007/test/arps_noopt_1cpu.input
>! ~/cfd2007/test/arps_noopt_1cpu.output; date

#Run ARPS executable compiled without automatic shared-memory parallelization
but with high-level of single CPU optimizations

date; ~/cfd2007/arps5.2.8/bin/arps_highopt <
~/cfd2007/test/arps_highopt_1cpu.input > ~/cfd2007/test/arps_highopt_1cpu.output;
date

#Run ARPS executable compiled with automatic shared-memory parallelization using
different number of CPUS/threads

setenv OMP_NUM_THREADS 1
date; ~/cfd2007/arps5.2.8/bin/arps_omp < ~/cfd2007/test/arps_omp1cpu.input >!
~/cfd2007/test/arps_omp1cpu.output; date

setenv OMP_NUM_THREADS 2
date; ~/cfd2007/arps5.2.8/bin/arps_omp < ~/cfd2007/test/arps_omp2cpu.input >!
~/cfd2007/test/arps_omp2cpu.output; date
```

Enter

```
bsub < arpsmpi_2cpu.bsub

date; mpirun.lsf ~/cfd2007/arps5.2.8/bin/arps_mpi < ~/cfd2007/test/arps1x2.input
> ~/cfd2007/test/arps_mpi1x2.output; date
```

to submit a batch job that contains that following ARPS jobs using a single node, with 1 or 2 CPUs, with different levels of optimization.

Enter

```
bsub < arpsmpi_4cpu.bsub

date; mpirun.lsf ~/cfd2007/arps5.2.8/bin/arps_mpi < ~/cfd2007/test/arps2x2.input
> ~/cfd2007/test/arps_mpi2x2.output; date

date; mpirun.lsf ~/cfd2007/arps5.2.8/bin/arps_mpi < ~/cfd2007/test/arps1x4.input
> ~/cfd2007/test/arps_mpi1x4.output; date

date; mpirun.lsf ~/cfd2007/arps5.2.8/bin/arps_mpi < ~/cfd2007/test/arps4x1.input
> ~/cfd2007/test/arps_mpi4x1.output; date
```

Enter

```
bjobs
```

to check the status of your batch jobs.

Step 5: Examine timing statistics in the output file and discuss the results

Look into the output file created by each run (called *.output). At the end of the file, there are timing statistics like the following. Put the total CPU time and wall clock time for all runs into a table and discuss the timing results. Compare the shared and distributed-memory parallel performance (speed up) relative to each other and relative to the single processor run. Discuss the impact of optimization, etc. Use graphs (e.g., histograms) to show the timings.

'ls –l runname.*' (where runname is arps1x1 etc.) will show output files runname.hdf000000, runname.hdf003600 and runname.hdf007200 (7200 here is 7200 s or 2 hours). The interval between the creation times of the 2 and 0 hour files (output at initial and end times of model run) is pretty much the wall clock time used by the job and should be close to that at the end of runname.output (e.g., arps_highopt_1cpu.input) file.

Example of Time Statistics Printed at the end of ARPS output file:

ARPS CPU Summary:

Process	CPU time		WALL CLOCK time (Processor mean for MPI job)	
Initialization :	4.449320E-01s	0.04%	2.785000E-01s	0.04%
Data output :	1.313003E+01s	1.05%	6.774600E+00s	1.08%
Wind advection :	2.325549E+01s	1.86%	1.164485E+01s	1.86%
Scalar advection:	7.649486E+01s	6.10%	3.832890E+01s	6.11%
Coriolis force :	0.000000E+00s	0.00%	0.000000E+00s	0.00%
Buoyancy term :	1.950153E+01s	1.56%	9.787250E+00s	1.56%
Misc Large tstep:	2.017043E+01s	1.61%	1.010230E+01s	1.61%
Small time steps:	3.937366E+02s	31.42%	1.971152E+02s	31.42%
Radiation :	1.197052E-02s	0.00%	1.750000E-03s	0.00%
Soil model :	0.000000E+00s	0.00%	0.000000E+00s	0.00%
Surface physics :	0.000000E+00s	0.00%	0.000000E+00s	0.00%
Turbulence :	2.484785E+02s	19.83%	1.243197E+02s	19.81%
Comput. mixing :	9.466899E+01s	7.55%	4.740025E+01s	7.55%
Rayleigh damping:	7.084208E+00s	0.57%	3.550250E+00s	0.57%
TKE src terms :	6.236527E+01s	4.98%	3.118030E+01s	4.97%
Gridscale precp.:	6.950378E-03s	0.00%	2.000000E-03s	0.00%
Cumulus (NO):	0.000000E+00s	0.00%	0.000000E+00s	0.00%
Microph (warmra):	1.113814E+02s	8.89%	5.569140E+01s	8.88%
Hydrometero fall:	1.283047E+01s	1.02%	6.450250E+00s	1.03%
Bound.conditions:	1.573137E+01s	1.26%	7.819600E+00s	1.25%
Message passing :	1.210785E+02s	9.66%	6.038205E+01s	9.62%
Miscellaneous :	3.281700E+01s	2.62%	1.657510E+01s	2.64%

Entire model :	1.253188E+03s	6.274050E+02s
Without Init/IO :	1.239614E+03s	6.203519E+02s

The total CPU time is the sum of CPU time used by all processors.