Measurement of AMRINS Parallel Performance

P. Colella
D. F. Martin
N. D. Keen

Applied Numerical Algorithms Group
NERSC Division
Lawrence Berkeley National Laboratory
Berkeley, CA

January 20, 2005

The target platform for this benchmark measurement is a machine named Halem located at GSFC. Halem is the NCCS Compaq AlphaServer SC45 System which consists of 104 symmetric multiprocessor nodes (4 processors per node). Memory is shared within a node.

The Fortran compiler used for this was the native Fortran compiler `f77` with the `-fast` optimization flag. The C++ compiler used was the GNU g++ compiler (version 3.3.1) with flags `-O2 -ftemplate-depth-27`.

A sample input used for the runs (for the $64 \times 64 \times 96$ case) is presented in Figure 1.

Table 1 shows the three sizes of benchmark problems used including the respective vorticity tagging factor, while Table 2 shows the total number of points updated for each run. In all of the benchmark runs, four timesteps are completed.

Problem size	Vorticity Tagging Factor
32x32x48	0.0050
64x64x96	0.0025
96x96x144	0.00167
128x128x192	0.00125

Table 1: Baseline Problem Data

Level	32x32x48	64x64x96	128x128x192
0	196608	1572864	12582912
1	1310720	5783552	23101440
2	38191104	212041728	1325907968
totals	39698432	219398144	1361592320

Table 2: Number of Points Updated Per AMR Level for each Problem Size

Changes were made to the code since the last performance improvement report. The most relevant is a change in the way solver tolerances are implemented in the elliptic solvers which results in better solver performance, while solving to the same accuracy. Other changes include a new advection method which replaces a less accurate previous one. This new advection method added significant source code as well increasing the computational work required to solve the example problem. We also found a bug in the TRANSVERSE_CROSS fortran function. Several performance improvements were made. Several of these performance improvements not only improved the scaling behaviour of the code, but also decreased the overall wall-clock time of all runs.

The parallel performance of the AMRINS code is summarized in Table 3. As we double the linear size of the problem, the computational size of the problem increases by a factor of 8 in 3-dimensions. So, we can compute scaled efficiency by comparing the run time

```
main.max_step = 4
main.max_time = 200.0
main.num_cells = 64 64 96
main.max_level        = 2
main.ref_ratio        = 4 4 4
main.regrid_interval  = 4 4
main.block_factor     = 8
main.max_grid_size    = 48
main.max_base_grid_size = 32
main.fill_ratio       = 0.8
main.grid_buffer_size = 1
main.is_periodic      = 0 0 1
main.cfl              = 0.5

main.checkpoint_interval = -1
main.plot_interval       = -1
main.plotPrefix          = pltNew.
main.verbosity           = 2        # higher number means more verbose

ns.vorticity_tagging_factor = 0.0025
ns.init_shrink   = 1.0
ns.tag_vorticity = 1
ns.project_initial_vel = 1
ns.init_pressures      = 1
ns.num_init_passes     = 1
ns.tags_grow           = 1

ns.specifyInitialGrids  = 0
ns.initVelFromVorticity = 1
ns.backgroundVelocity   = 0.0
ns.viscosity =  0.000001
ns.num_scalars = 1
ns.scal_diffusion_coeffs = 0.00 0.0

ns.viscous_num_smooth_up   = 1 #multigrid solver parameter
ns.viscous_num_smooth_down = 1 #multigrid solver parameter

projection.doSyncProjection = 1
projection.applyFreestreamCorrection = 0
projection.eta = 0.9
projection.numSmoothUp   = 3 # multigrid solver parameter
projection.numSmoothDown = 3 # multigrid solver parameter

# 0 = solidWall, 1=inflow, 2=outflow, 3=symmetry, 4=noShear
physBC.lo = 4 4 4   # physical BC info (overridden if periodic)
physBC.hi = 4 4 4   # physical BC info (overridden if periodic)
physBC.maxInflowVel = 1.0
```

Figure 1: Input file for $64 \times 64 \times 96$ case

between two runs which differ by a factor of 2 in base grid size, and a factor of 8 in number of processors. These are shown in Table 4. As can be seen, the scaled efficiencies range from 0.75 (75%) to 1.13. Cases where the efficiencies are greater than 1 indicate additional efficiencies which come about through AMR, along with the variablilty of the different grid hierarchies generated in each case.

Prob size	Num Procs	Avg Memory MB	Min-Max mem MB	AMR Run secs
32x32x48	1	433	433-433	2837
32x32x48	2	240	239-242	1459
32x32x48	4	143	136-148	823
32x32x48	8	91	80-105	449
32x32x48	16	61	48-78	286
32x32x48	32	43	13-68	221
64x64x96	8	354	327-384	2605
64x64x96	16	209	180-230	1413
64x64x96	32	126	106-166	853
64x64x96	64	85	37-151	597
128x128x192	64	312	256-365	2632
128x128x192	128	197	158-268	1698

Table 3: Current parallel performance of AMRINS code for baseline vortex-ring problem

Base Problem Size	Num Procs	Large Problem Size	Large num processors	Scaled Efficiency
32x32x48	1	64x64x96	8	1.13
	2		16	1.03
	4		32	0.96
	8		64	0.75
32x32x48	1	128x128x192	64	1.07
	2		128	0.86
64x64x96	8	128x128x192	64	0.99
	16		128	0.83

Table 4: Scaled Efficiencies computed from Table 3