Measurement of AMR Self-Gravity Parallel Performance

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

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

October 19, 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`.

The inputs files used for these benchmarks may be found in the AMRSelfGravity download tarfile, in the `Chombo/example/AMRSelfGravity/exec/` directory; the `selfGravityBenchmark64.inputs` file was used for the $64 \times 64 \times 64$ case, while the `selfGravityBenchmark128.inputs` file was used for the $128 \times 128 \times 128$ case. The input used for the runs (for the $64 \times 64 \times 64$ case) is presented in Figure 1.

Table 1 shows the two sizes of benchmark problems used including the respective tagging factor for the cell mass, while Table 2 shows the total number of points updated for each run. In all of the benchmark runs, 15 coarse-level timesteps are completed. The cell mass threshold used for refinement scales by $(\Delta x)^D$ where D is the dimensionality of the problem and. In particular, as the cell spacing is halved, the threshold is reduced by a factor of 8 in three dimensions. Because the dust-collapse problem is very dynamic in structure, we also double the CFL number as we halve the cell spacing, in order to keep the solutions roughly equivalent at each timestep.

Problem size	Cell-mass Threshold Factor	CFL number
64x64x64	1.5e-7	0.25
128x128x128	1.9e-8	0.50

Table 1: Baseline Problem Data

Level	64x64x64	128x128x128
0	393216	31457280
1	2277376	10395648
2	11214848	63078400
totals	17424384	104931328

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

The parallel performance of the AMR self-gravity 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 between two runs which differ by a factor of 2 in base grid size, and a factor

```
charm.problem = dustcollapse
charm.cloud_density= 1.00
charm.cloud_radius = 0.125  #  0.0625
charm.verbosity = 3
charm.max_step = 15
charm.max_time = 10000000.0
charm.domain_length = 1.0
charm.num_cells = 64 64 64
charm.is_periodic = 0 0 0    # 0= non-periodic
charm.max_level          = 2
charm.max_init_ref_level = 2
charm.ref_ratio       = 2 2 2 2 2 2 2 2
charm.regrid_interval = 4 4 4 4 4 4 4 4
charm.tag_buffer_size      = 2
charm.block_factor  = 8
charm.max_grid_size = 32
charm.fill_ratio    = 0.8
charm.use_gradient_refine  = 0
charm.use_num_part_refine  = 0
charm.use_shock_refine     = 0
charm.use_over_dense_refine= 1
charm.use_jeans_refine     = 0
charm.cell_mass_thresh = 1.5e-7 # = rho*dx^Dim = 3.8e-6*rho *(64/nx)^3
charm.gamma = 1.6666666666667
charm.use_fourth_order_slopes = 0
charm.use_prim_limiting = 1
charm.use_char_limiting = 0
charm.use_flattening    = 0
charm.use_artificial_viscosity = 0
charm.artificial_viscosity = 0.2
charm.normal_predictor = PLM
charm.checkpoint_interval = -1
charm.plot_interval       = 0
charm.cfl     = 0.250
charm.initial_cfl   = 0.25
charm.max_dt_growth = 1.10
charm.dt_tolerance_factor = 1.10
charm.rs_tolerance = 1.e-6
charm.max_rs_iter  = 10
charm.max_mach     = 50
charm.bc_lo = 3 3 3  #bcs for lo faces 0==dirc, 1==neumann, 2==inf bc, 3==gauss
charm.bc_hi = 3 3 3  #bcs for hi faces 0==dirc, 1==neumann, 2==inf bc, 3==gauss
charm.force_stencil        = 0
charm.use_delta_phi_corr   = 1 # 1=true; 0=false
```

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

of 8 in number of processors. These are shown in Table 4. As can be seen, the scaled efficiencies computed range from 0.71 (71%) to 0.97.

Prob size	Num Procs	AMR Run secs	Avg Memory MB	Min-Max mem MB
64x64x64	01	1032.8	382	382-382
64x64x64	02	536.99	254	251-257
64x64x64	04	301.42	171	167-176
64x64x64	08	175.96	121	118-122
128x128x128	08	964.59	396	363-414
128x128x128	16	552.00	246	234-266
128x128x128	32	421.44	161	148-177
128x128x128	64	487.53	114	102-128

Table 3: Current parallel performance of AMR self-gravity code for baseline dust collapse problem

Base Problem Size	Num Procs	Large Problem Size	Large num processors	Scaled Efficiency
64x64x64	1	128x128x128	8	1.07
	2		16	0.973
	4		32	0.715

Table 4: Scaled Efficiencies computed from Table 3