Measurement of Improved Performance for Incompressible Navier-Stokes with Particles Example

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

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

February 8, 2005

In a previous report, the performance of the baseline Navier-Stokes with particles code was documented. A description of the problem was presented including the input and techniques used to measure wall-clock times along with serial and parallel performance results of the baseline code on the NCCS Compaq AlphaServer machine at GSFC named Halem. We also measured the approximate peak memory usage for the benchmark problem in serial.

In this report, we present timing measurements that show an improvement in performance after various code optimizations. The current code and the modified baseline code are provided with this report and are available on the web at http://davis.lbl.gov/NASA.

The summary of the outcome of our performance improvements is as follows. We reduced the wall clock time by factors ranging from 1.6 in serial to to 10.7 for 16 processors (table 1). The amount of speedup monotonically improves with the number of processors, demonstrating improved scalability. These speedups were obtained as a result of code improvements which included the implementation of an adaptive mesh refinement (AMR) capability to the code, replacing the uniform mesh implementation. Because the baseline code was constrained by a serial bottleneck (an FFT solve), adding AMR reduced the effect of this bottleneck and resulted in vastly improved scaling. Other performance improvements were implemented, mostly involving optimizations to the way the particle drag forces are computed. To improve scalability, the particles now use a different processor distribution than the fluid quantities; this enables us to balance the particle loads and fluid-solver loads independently, which greatly improved parallel performance.

Also, these are fixed-size speedups, with all of the performance improvements coming from algorithm and code improvements, as opposed to increasing the number of processors. We were also able to reduce the memory required, ranging from a factor of 2.3 for the serial case to a factor of 7.7 for the 16 processor case on halem.

Because of the speedups obtained here, and the fact that the performance improvements increased with the problem size and number of processors, we feel that we have met the requirements of Milestone G.

Summary of Performance Optimizations

AMR

We implemented an AMR capability in the particle code. This AMR algorithm is non-subcycled (all cells are advanced at the same timestep, regardless of their degree of refinement). This simplified the algorithm design, but entailed a fair amount of code development, since there was no pre-existing non-subcycled AMR time-dependent code in the Chombo framework. One assumption made in the current AMR implementation is that all particles will only be on the finest refinement level. This simplifies the computation of particle forces, and is a reasonable assumption for the types of problems we expect to run.

Load Balance Algorithm

Load balancing was improved by implementing a second processor distribution for the dataholders containing the particles. In the baseline implementation, all data holders were distributed according to the number of points in each box, which provides reasonable results for a fluid code. However, it was observed that this resulted in the particle-related sections of the algorithm being badly unbalanced, especially in the parts of the algorithm where the particle drag forces are deposited on the fluid mesh, and where the particle motion is being computed.

By computing a second processor distribution for the particle data holders based on the number of particles in each box, this load balancing issue was greatly improved. This does result in some extra communication when data needs to be moved from one processor distribution to another, but the additional costs are greatly outweighed by the savings due to better load balancing of the particle work.

Serial Optimizations

The fortran subroutine which computes the kernel for depositing the particle drag force onto the fluid mesh was optimized.

Algorithmic Improvements

The baseline implementation had a fairly restrictive time-step constraint due to the particle update step. By improving the scheme used to update the particle positions and velocities, we were able to greatly ameliorate the timestep restriction.

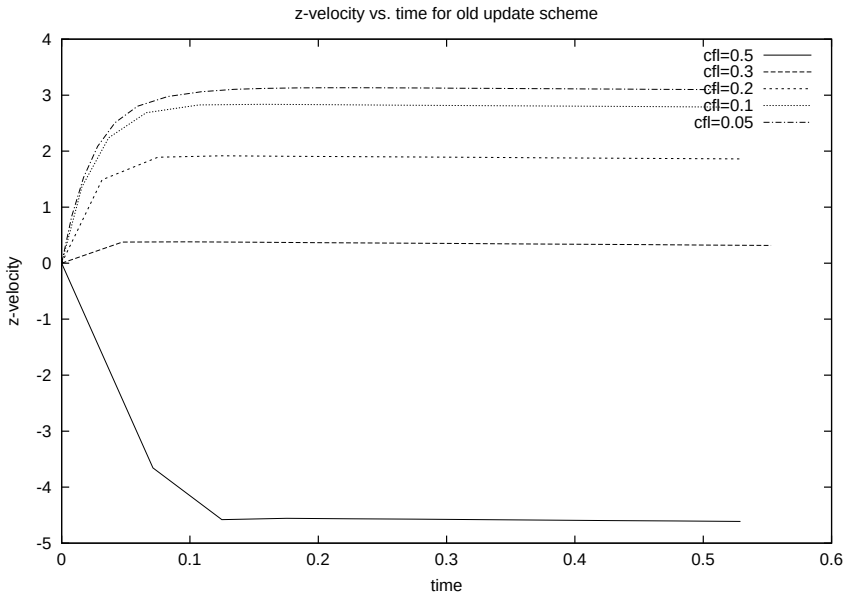
Figure 1 shows the particle velocities vs. time for a single particle in the vortex ring flowfield for a ranges of particle CFL numbers for both the old and new particle update schemes. As can be seen, the new scheme allows the use of a much larger timestep for the same accuracy.

While this has no effect on these performance measurements (since they are 4 timesteps in all cases), when computing real solutions this improvement means that a user can compute about 10 times fewer timesteps to get the same solution accuracy for the particles.

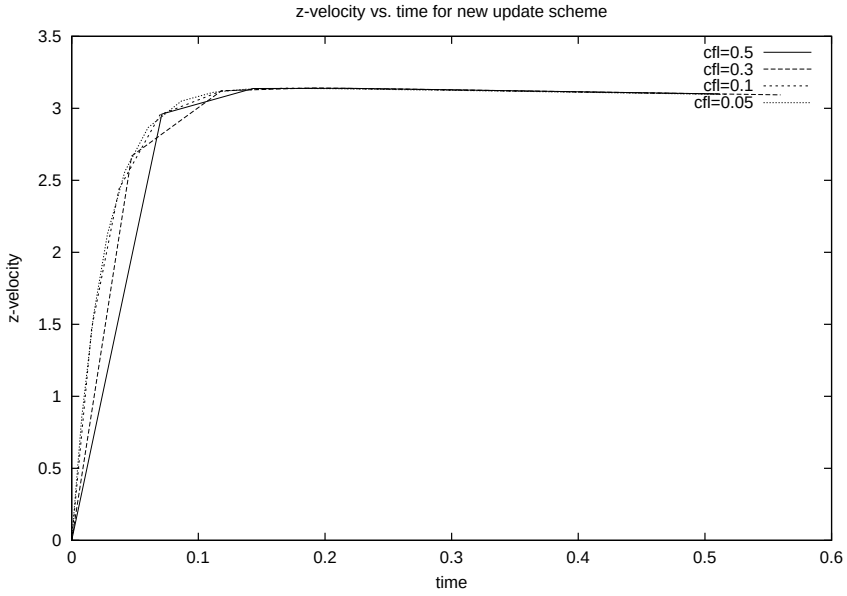
Current Results after Improving Code Performance

Because halem.gsfc.nasa.gov was unavailable when the baseline performance was measured, we re-ran the baseline results on halem to provide a second point of comparison.

The following tables show the performance results of the current code and the original baseline code. The baseline runs are with a problem domain size of 128^3, while the current AMR runs are for a base domain size of 64^3 with one level of refinement. The refinement ratio is two.



(a) old Scheme



(b) new Scheme

Figure 1: Particle velocities vs. time for varying particle CFL numbers for the single particle vortex-ring problem with initial particle position at (0.5, 0.5, 0.5). (a) old particle update scheme, (b) new particle update scheme. (note different $y-$axis scales)

We also include the maximum memory used over the entire run and all processors. This memory measurement is made using the system call **getrusage()** on the halem machine. This system call retrieves information about resources used by the current process such as the maximum resident set size and is an accurate measurement of the total memory usage of the code.

Table 1 shows performance results of the baseline and current code. The table includes a column that shows the improvement over the baseline code measurements.

The "Baseline run time" and "AMR run time" are the measured wall-clock time to compute 4 timesteps and do not include setup overhead or writing large data sets to the filesystem (I/O).

Num Procs	Baseline Run time (s)	Baseline Max Mem (MB)	AMR Run time (s)	Max Mem (MB)	improvement over baseline
1	1383.6	1196.2	888.2	528.7	1.6
2	1228.4	900.5	498.5	308.0	2.5
4	1273.5	657.3	271.9	181.0	4.7
8	1105.2	585.7	155.9	107.1	7.1
16	1074.1	536.2	100.0	69.6	10.7
32			75.3	63.9	

Table 1: Results on Halem

Parallel Scaling Performance

To evaluate the scaling behavior of the particle code, we ran a modified version of the benchmark problem in the previous section. Instead of a single plane of particles, we instead ran the vortex-ring problem with a cloud of 32,768 particles arranged in a $32 \times 32 \times 32$ array, spanning from 15 cm to 85 cm in the $x-$ and $y-$ directions, and from 25 cm to 75 cm in the $z-$direction in the 100 cm cubic domain. The number of particles was increased relative to the standard benchmark problem to ensure there were enough particles to distribute over 128 processors. We ran the problem with three base grid sizes – 32^3, 64^3, and 128^3.

The work due to the particles has two components – the part from updating the particle states (position and velocity), and the work from projecting the particle drag forces on to the computational mesh.

The particle update work includes interpolating the fluid velocities from the mesh to the particles, updating the forces on the particles, and then solving the ODE's to update the particle positions and velocities. We expect this to scale more or less directly with the number of particles, although the interpolation of fluid velocities to particles has some dependence on grid size. The total amounts for this work, along with the total AMR

run time, are shown in Table 2, which demonstrate that this part of the update takes a negligible amount of time, and scales quite well.

Because the particle spreading radius (ϵ in Section 4 in [MC03]) and the discrete delta function radius (see Eqn 5 in [MC03]) are held fixed as we reduce the mesh spacing (these are held fixed as the mesh spacing decreases because the particles in this problem represent physical particles, rather than point charges), the work involved in the particle-fluid drag force projection should also increase by a factor of 8 as the mesh spacing is halved, in the same way as the fluid solver.

So, for these scaling tests, the number of particles is held fixed in these runs while we decrease the mesh spacing. This is consistent with the approach taken for the AMR Navier-Stokes code [CMK05]. As we double the linear size of the problem holding the number of particles constant, the asymptotic computational size (both in memory and CPU time) of the problem increases by a factor of 8 in 3 dimensions.

We compute a scaled efficiency by comparing the run time between two runs which differ by a factor of two in base grid size and a factor of 8 in number of processors. Run times and maximum memory usage for this problem are shown in Table 2, while the resulting efficiencies are shown in Table 3.

Prob size	Num Procs	Max Mem (MB)	AMR Run (sec)	Particle update (sec)
32x32x32	8	72.5	100.4	0.59
64x64x64	16	115.7	178.4	1.2
64x64x64	64	75.8	103.4	0.38
128x128x128	32	317.8	597.3	3.73
128x128x128	64	175.1	352.4	2.24
128x128x128	128	117.3	220.8	1.41

Table 2: Current parallel performance of AMRINS with particles code for vortex-ring problem with 32,768 particles

Base Problem Size	Num Procs	Large Problem Size	Large num processors	Scaled Efficiency
32x32x32	8	64x64x64	64	0.97
64x64x64	16	128x128x128	128	0.81

Table 3: Scaled Efficiencies computed from Table 2

Bibliography

[CMK05] P. Colella, D. F. Martin, and N. D. Keen. Measurement of AMRINS parallel performance. Technical report, Applied Numerical Algorithms Group, Lawrence Berkeley Laboratory, 2005.

[MC03] Dan Martin and Phil Colella. Incompressible Navier-Stokes with particles design document. Technical report, Applied Numerical Algorithms Group, Lawrence Berkeley Laboratory, 2003.