An Incompressible Navier-Stokes with Particles Algorithm and Parallel Implementation

Dan Martin[a], Phil Colella[a], and Noel Keen[a*].

[a]Applied Numerical Algorithms Group, Lawrence Berkeley National Laboratory,
1 Cyclotron Road, Berkeley, CA 94720

We present a variation of an adaptive projection method for computing solutions to the incompressible Navier-Stokes equations with suspended particles. To compute the divergence-free component of the momentum forcing due to the particle drag, we employ an approach which exploits the locality and smoothness of the Laplacian of the projection operator applied to the discretized particle drag force. We present convergence and performance results to demonstrate the effectiveness of this approach.

1. Introduction

Projection methods enable computation of incompressible and low Mach number flows with computational timesteps dictated by advective timescales rather than the more restrictive acoustic timescales. [1] The projection operator $\mathcal{P}$ projects a vector field onto the space of divergence-free vectors through use of the Hodge-Helmholtz decomposition. Given a vector field $\boldsymbol{u}$, there exists a vector field $\boldsymbol{u}_d$ and a scalar ϕ such that:

$$\boldsymbol{u} = \boldsymbol{u}_d + \nabla\phi \tag{1}$$
$$\nabla \cdot \boldsymbol{u}_d = 0. \tag{2}$$

Then, the projection operator may be written in the form:

$$\mathcal{P}\boldsymbol{u} = \boldsymbol{u}_d \tag{3}$$
$$\mathcal{P}\boldsymbol{u} = \Big(\boldsymbol{I} - \nabla(\Delta^{-1})\nabla\cdot\Big)\boldsymbol{u} \tag{4}$$

By refining the computational mesh in regions of the domain where greater accuracy is desired, adaptive mesh refinement (AMR) allows greater computational efficiency, focusing computational resources where they are most needed. We use block-structured local refinement of a Cartesian mesh [1], which enables parallelization in a straightforward way by distributing logically rectangular patches among processors. [2]

*This work supported by the NASA Earth and Space Sciences Computational Technologies Program and by the U.S. Department of Energy: Director, Office of Science, Office of Advanced Scientific Computing, Mathematical Information, and Computing Sciences Division under Contract DE-AC03-76SF00098.

2. Problem Description

We wish to solve the incompressible Navier-Stokes equations with suspended particles. The particles exert drag on the fluid, while particle motion is induced by the particle-fluid drag forces on the particles, so the particles may move with a velocity different from that of the local fluid. The momentum equation and divergence constraint are:

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} = -\frac{1}{\rho}\nabla p + \nu \Delta \boldsymbol{u} + \boldsymbol{f} \tag{5}$$

$$\nabla \cdot \boldsymbol{u} = 0, \tag{6}$$

where $\boldsymbol{u}$ is the velocity field, p is the pressure, ν is the kinematic viscosity, and $\boldsymbol{f}$ is the sum of the drag force exerted by the particles on the fluid:

$$\boldsymbol{f}(\boldsymbol{x}, t) = \sum_{k=1}^{N} \boldsymbol{f}^{(k)}(t)\delta(\boldsymbol{x} - \boldsymbol{x}^{(k)}(t)). \tag{7}$$

N is the number of particles. We denote quantities associated with the kth particle by the superscript (k) (other quantities are assumed to be associated with the fluid); $\boldsymbol{f}^{(k)}$ is the drag force exerted by the kth particle on the fluid, and $\boldsymbol{x}^{(k)}$ is the location of the kth particle. We treat the particle drag force as a point source, spread to the computational mesh using $\delta_\epsilon(\boldsymbol{x})$, a smoothed numerical approximation to the Dirac delta function $\delta(r)$.

The fluid-particle drag force is given by a simple drag law with drag coefficient k_{drag}:

$$\boldsymbol{f}^{(k)}(t) = k_{drag}\big(\boldsymbol{u}^{(k)} - \boldsymbol{u}(\boldsymbol{x}^{(k)}(t))\big). \tag{8}$$

The motion of a particle with mass $m^{(k)}$ is due to the equal and opposite force on the particle from the fluid, along with a gravitational acceleration $\boldsymbol{g}$:

$$\frac{\partial \boldsymbol{u}^{(k)}}{\partial t} = -\frac{\boldsymbol{f}^{(k)}}{m^{(k)}} + \boldsymbol{g} \tag{9}$$

$$\frac{\partial \boldsymbol{x}^{(k)}}{\partial t} = \boldsymbol{u}^{(k)}. \tag{10}$$

We approximate $\delta(r)$ numerically by a discrete delta function δ_ϵ with the properties:

$$\delta_\epsilon = \frac{1}{2^{D-1}\pi\epsilon^D} g(\frac{r}{\epsilon}) \tag{11}$$

$$g(r) \geq 0$$

$$g(r) = 0 \quad \text{for} \quad r > 1$$

$$\int_0^1 g \; r^{D-1} dr = 1$$

The parameter ϵ is the particle spreading radius. We use a quadratic function for $g(r)$.

3. Projecting the Particle Force

Computing the update to the momentum equation requires the divergence-free contribution of the particle-induced forces. There are several options.

The simplest approach is to add the forcing due to particle drag directly to the momentum equation, and then project the resulting velocity field. Unfortunately, this forcing tends to be singular, so taking the derivatives necessary for a projection method is problematic from an accuracy standpoint. [3]

A second approach is to analytically determine the projection of the discrete delta function used to spread the particle force onto the mesh. [3] If the projection operator is $(\boldsymbol{I} - grad(\Delta^{-1})div)$, then we can define $\boldsymbol{K}_\epsilon = \{K_{ij}\}$ such that $\mathcal{P}(\boldsymbol{f}) = \boldsymbol{K}_\epsilon \boldsymbol{f}$:

$$\boldsymbol{K}_\epsilon(\boldsymbol{x}) = (\boldsymbol{I} - grad(\Delta^{-1})div)\delta_\epsilon \tag{12}$$

In an infinite domain, the operators can commute:

$$\boldsymbol{K}_\epsilon(\boldsymbol{x}) = \delta_\epsilon \boldsymbol{I} - grad\ div(\Delta^{-1})\delta_\epsilon \tag{13}$$

Note that $\Delta^{-1}\delta_\epsilon$ may be evaluated analytically with the proper choice of δ_ϵ.

Then, the projection of the forces on the grid may be computed:

$$\mathcal{P}\boldsymbol{f} = \sum_k \boldsymbol{f}^{(k)} \boldsymbol{K}_\epsilon(\boldsymbol{x} - \boldsymbol{x}^{(\boldsymbol{k})}) \tag{14}$$

While this approach avoids the accuracy issues of the first approach, it is expensive. Since $\boldsymbol{K}_\epsilon$ does not have compact support, the cost of this approach is $O(N_p N_g)$, where N_p is the number of particles, and N_g is the number of grid points.

We surmount this with the realization that while the projection of the drag force does not have compact support, the Laplacian of the projected drag force does. Taking the Laplacian of (13), again using the commutability of the operators in an infinite domain,

$$\Delta \boldsymbol{K}_\epsilon = \Delta\delta_\epsilon - grad\ div(\delta_\epsilon \boldsymbol{I}) \tag{15}$$

Note that this *does* have compact support, since $\delta_\epsilon = 0$ for $r > \epsilon$.

Now define a discrete approximation to (15) at a grid location indexed by $\boldsymbol{i}$:

$$\boldsymbol{D}_{\boldsymbol{i}}^{(k)} = \Delta^h \boldsymbol{f}^{(k)} \boldsymbol{K}_\epsilon(\cdot - \boldsymbol{x}^{(k)}) \tag{16}$$

where Δ^h is the discrete Laplacian operator with grid spacing h, and the $(\cdot - \boldsymbol{x}^{(k)})$ signifies evaluation at grid points, i.e. $(\boldsymbol{i}h - \boldsymbol{x}^{(k)})$. Then,

$$(\Delta^h)^{-1}\boldsymbol{D}^{(k)} = \boldsymbol{f}^{(k)} \boldsymbol{K}_\epsilon(\cdot - \boldsymbol{x}^{(k)}) \tag{17}$$

Using the compact support of δ_ϵ. we may evaluate $\boldsymbol{D}$ in the local neighborhood of the particle:

$$\boldsymbol{D}_{\boldsymbol{i}}^{(k)} = \Delta^h \boldsymbol{f}^{(k)} \boldsymbol{K}_\epsilon(\cdot - \boldsymbol{x}^{(k)}) \ \text{ for } |\boldsymbol{i}h - \boldsymbol{x}^{(k)}| < (\epsilon + Ch) \tag{18}$$

where C is a safety factor. Then,

$$\boldsymbol{D_i} = \sum_k \boldsymbol{D_i^{(k)}} \tag{19}$$

$$\mathcal{P}_I \boldsymbol{f}(\boldsymbol{i}h) \approx (\Delta^h)^{-1} \boldsymbol{D} \tag{20}$$

We solve (20) with infinite-domain boundary conditions on $\mathcal{P}_I \boldsymbol{f}(\boldsymbol{x})$ (the subscript I indicates the use of infinite-domain boundary conditions as opposed to the standard projection operator $\mathcal{P}(\boldsymbol{u})$, which includes physical boundary conditions on the velocity).

To better approximate the no-normal-flow boundary condition at physical walls, we also use image particles for all particles near the wall. For each particle within $(Ch + \epsilon)$ of the wall, we add an image particle on the other side of the wall with the opposite velocity field normal to the wall. This provides a first-order approximation to the no-flow boundary condition, taking account of the particles near the wall. The boundary condition will be strictly enforced by the projection step of the fluid update.

4. Discretization of Advance

The particle drag force projection outlined above is combined with the incompressible AMR Navier-Stokes algorithm in [1] to produce an AMR incompressible Navier-Stokes with suspended particles solver. For algorithmic simplicity, this implementation does not refine in time.

We begin with the discrete solution on a locally refined Cartesian mesh at time t^n. The velocity field $\boldsymbol{u}$ and pressure p are cell-centered. Each particle's position $\boldsymbol{x}^{(k),n}$ and velocity $\boldsymbol{u}^{(k),n}$ are also known. To advance the solution from time t^n to time $t^{n+1} = t^n + \Delta t$, we proceed as follows.

We first compute the drag force at time t^n, $\boldsymbol{f}^{(k),n}$ on each particle using (8). Fluid velocities are computed at particle locations using quadratic interpolation of the cell-centered velocity $\boldsymbol{u}^n$.

Using the approach outlined above, we compute the projected force $\mathcal{P}_I(\boldsymbol{f^n})$ using infinite-domain boundary conditions.

Then, we compute a provisional update $\boldsymbol{u}^*$ in much the same way as in [1]:

$$\boldsymbol{u}^* = \boldsymbol{u}^n + \Delta t\Big(-[(\boldsymbol{u}\cdot\nabla)\boldsymbol{u}]^{n+\frac{1}{2}} - grad(p^{n-\frac{1}{2}}) + [\nu\Delta\boldsymbol{u}] + \mathcal{P}_I(\boldsymbol{f}^n)\Big) \tag{21}$$

where the nonlinear advective term $[(\boldsymbol{u}\cdot\nabla)\boldsymbol{u}]^{n+\frac{1}{2}}$ is computed using the second-order upwind scheme outlined in [1], including $\mathcal{P}_I(\boldsymbol{f}^n)$ as a forcing term in the predictor step, and $[\nu\Delta\boldsymbol{u}]$ is computed using a second-order L_0-stable Runge-Kutta scheme [5].

We now update the particle velocities and positions using the analytic solutions for (9-10) (for compactness of notation, $\boldsymbol{u}^{(k),n}$ refers to the particle velocity at time t^n, while $\boldsymbol{u}^n$ refers to the fluid velocity at time t^n interpolated to the particle position $\boldsymbol{x}^{(k),n}$):

$$\boldsymbol{u}^{(k),n+1} = (\boldsymbol{u}^{(k),n} - \boldsymbol{u}^n - \frac{m^{(k)}\boldsymbol{g}}{k_{drag}})e^{-\frac{\Delta t k_{drag}}{m^{(k)}}} + \boldsymbol{u}^n + \frac{m^{(k)}\boldsymbol{g}}{k_{drag}} \tag{22}$$

$$\boldsymbol{x}^{(k),n+1} = \boldsymbol{x}^{(k),n} + \frac{m^{(k)}}{k_{drag}}\Big(\boldsymbol{u}^{(k),n} - \frac{m^{(k)}\boldsymbol{g}}{k_{drag}} - \boldsymbol{u}^n\Big)(1 - e^{-\frac{\Delta t k_{drag}}{m^{(k)}}}) + \Delta t(\boldsymbol{u}^n + \frac{m^{(k)}\boldsymbol{g}}{k_{drag}}) \tag{23}$$

We then use $\boldsymbol{u}^{(k),n+1}$ and $\boldsymbol{x}^{(k),n+1}$ to compute the projected drag force (again using infinite-domain boundary conditions) $\mathcal{P}_I(\boldsymbol{f}^*)$. Then, we modify $\boldsymbol{u}^*$ to make the update second-order in time, and project to complete the update:

$$\boldsymbol{u}^{n+1} = \mathcal{P}\Big(\boldsymbol{u}^* + \Delta t\Big[grad(p^{n-\frac{1}{2}}) - \frac{1}{2}\mathcal{P}_I(\boldsymbol{f}^n) + \frac{1}{2}\mathcal{P}_I(\boldsymbol{f}^*)\Big]\Big) \tag{24}$$

$$grad(p^{n+\frac{1}{2}}) = (\boldsymbol{I} - \mathcal{P})\Big(\boldsymbol{u}^* + \Delta t\Big[grad(p^{n-\frac{1}{2}}) - \frac{1}{2}\mathcal{P}_I(\boldsymbol{f}^n) + \frac{1}{2}\mathcal{P}_I(\boldsymbol{f}^*)\Big]\Big) \tag{25}$$

Note that the projection is also applied to $\mathcal{P}_I(\boldsymbol{f})$; to enforce the physical boundary conditions, since $\mathcal{P}_I(\boldsymbol{f})$ was computed using infinite-domain boundary conditions. We use the approximate cell-centered projection described in [1].

5. Evaluating $\mathcal{P}_I(\boldsymbol{f})$

We want to approximate the divergence-free contribution of the drag force $\mathcal{P}_I(\boldsymbol{f})$.
In indicial notation,

$$(\mathcal{P}_I\boldsymbol{f}(\boldsymbol{x}))_i = \sum_k f_j^{(k)}(\delta_{ij}\Delta - \partial_i\partial_j)(\Delta^{-1})\delta_\epsilon(\boldsymbol{x} - \boldsymbol{x}^{(k)}) \tag{26}$$

Define $K_{ij}^{(k)}(\boldsymbol{x}) = (\delta_{ij}\Delta - \partial_i\partial_j)(\Delta^{-1})\delta_\epsilon(\boldsymbol{x} - \boldsymbol{x}^{(k)})$. Then,

$$\mathcal{P}_I f(\boldsymbol{x})_i = \sum_k f_j^{(k)} K_{ij}^{(k)} \tag{27}$$

Approximate $\boldsymbol{D} = \Delta\boldsymbol{K}$ by $\tilde{\boldsymbol{D}} = \{\tilde{D}_{ij}\}$:

$$\tilde{D}_{ij}^{(k)}(\boldsymbol{x}) = \begin{cases} (\Delta^h K_{ij}^{(k)})(\boldsymbol{x}) & \text{if } \ r < \epsilon + Ch \\ 0 \ \text{ otherwise} \end{cases} \tag{28}$$

Then,

$$\mathcal{P}_I f(\boldsymbol{x})_i = \sum_k f_j^{(k)} K_{ij}^{(k)}(\boldsymbol{x}) \tag{29}$$

$$\Delta^h\mathcal{P}_I f(\boldsymbol{x})_i = \sum_k f_j^{(k)} \Delta^h K_{ij}^{(k)}(\boldsymbol{x}) \tag{30}$$

$$\approx \sum_k f_j^{(k)} \tilde{D}_{ij}^{(k)}(\boldsymbol{x}) \tag{31}$$

$$\mathcal{P}_I f(\boldsymbol{x})_i \approx (\Delta^h)^{-1} \sum_k f_j^{(k)} \tilde{D}_{ij}^{(k)}(\boldsymbol{x}). \tag{32}$$

We solve (32) with infinite-domain boundary conditions on $\mathcal{P}_I\boldsymbol{f}(\boldsymbol{x})$. [4].

6. AMR implementation

By focusing computational effort on the neighborhood of the particles, AMR can increase efficiency and improve parallel performance. The particles are distributed onto the same block-structured meshes as the fluid solver, but with a different processor distribution to balance the particle loads independently from the fluid solver workload. This results in a better overall balance, at the cost of some added communication between the two processor distributions. The code was implemented using the Chombo framework [6].

7. Convergence – Single Particle Settling

We use a simple test problem to demonstrate the accuracy and effectiveness of this approach. A single particle with mass 0.001g starts at rest in a fluid in a 1 m^3 cubic domain. As the particle accelerates downward due to gravity, a velocity is induced in the fluid due to the particle drag. We use $k_{drag} = 0.04$, $\epsilon = 6.25cm$, and $\nu = 0.004\frac{cm^2}{s}$.

Because there is no analytic solution available for this problem, we compute a solution on a uniform 256^3 fine mesh and treat this as the "exact" solution against which we compare other computed solutions. The convergence of the $x-$velocity in the L_2 norm (other velocity components and norms are similar) is shown in Figure 1 for uniform mesh and for a single level of refinement with refinement ratios of 2 and 4. Because the timestep and the cell spacing are reduced simultaneously (the timestep is halved when the cell spacing is halved), the second order convergence in these plots demonstrates convergence in both time and space. Also, if AMR is effective, the errors of the adaptive computations should approach those of the uniform mesh computation with the equivalent resolution (i.e. a 64^3 computation with one level of refinement with a refinement ratio of 2 should have the same error as a 128^3 uniform mesh computation. This is borne out in Figure 1.

Figure 2 shows the serial CPU times on a 2 GHz Opteron processor and the total number of cells advanced for the 256^3 uniform-mesh computation and for the equivalent-resolution adaptive cases with a single refinement level with refinement ratios of 2 and 4. The values are normalized by the uniform-mesh values, to make it easier to evaluate. Even for this simple test case, the use of AMR results in significant savings, both in the number of cells advanced (a crude indicator of memory use), and in CPU time. The space between the two lines represents the overhead due to adaptivity.

8. Parallel Performance – particle cloud with a vortex ring

To demonstrate the parallel performance of this algorithm, we use a different problem which has enough particles to distribute effectively, and which has more complicated fluid dynamics beyond the drag-induced flow. We compute a three-dimensional vortex ring in a 1-meter cube domain with 32,768 particles arranged in a $32 \times 32 \times 32$ array, spanning $15cm \leq x, y \leq 85cm$, and $25cm \leq z \leq 75cm$. For this problem, the vorticity distribution is specified, from which the initial velocity is computed. The vortex ring is specified by a location of the center of the vortex ring (x_0, y_0, z_0), the radius of the center of the local cross-section of the ring from the center of the vortex ring r, and the strength Γ.

The cross-sectional vorticity distribution in the vortex ring is given by $\omega(\rho) = \frac{\Gamma}{a\sigma^2} e^{(\frac{\rho}{\sigma})^3}$. ρ is the local distance from the center of the ring cross-section, $a = 2268.85$, and $\sigma = 2.75$.

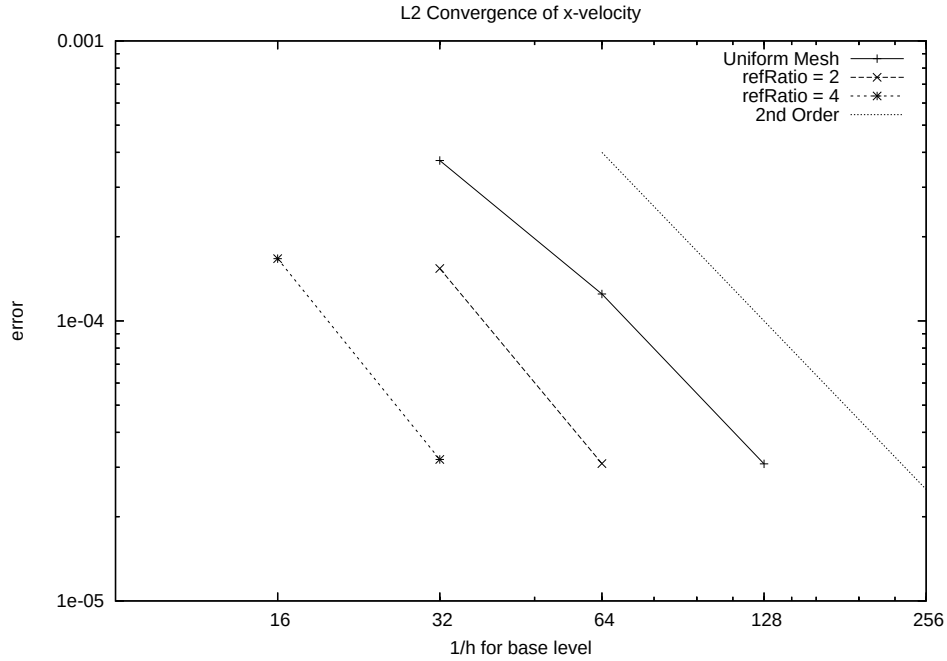

Figure 1. Convergence for the single particle settling problem. $x-$ axis is $\frac{1}{h_0}$, while the $y-$ axis is the L_2 error.

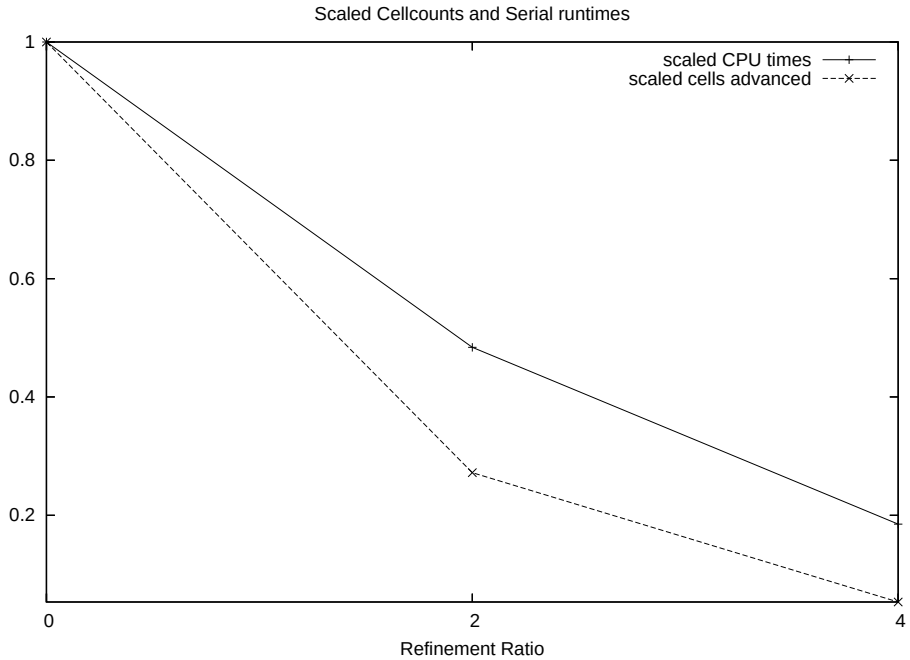

Figure 2. Scaled run times and cell counts for the single particle settling problem.

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 1
Parallel performance 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 2
Scaled Efficiencies computed from Table 1.

The vortex ring is centered at $(50cm, 50cm, 40cm)$, with a radius of 2cm and $\Gamma = 1.5 \times 10^5$.

The number of particles is held fixed while we decrease the mesh spacing. In three dimensions, as we halve the cell spacing while holding the number of particles constant, the asymptotic computational size (both in CPU time and memory) of the problem increases by a factor of 8. The particle spreading radius ϵ is also held fixed at $6.25cm$ as the mesh spacing is decreased because the particles in this problem represent physical particles, rather than point charges. Therefore, the work involved in the particle-fluid drag force projection should also increase by a factor of 8 as the mesh spacing is halved.

Run times and maximum memory usage for this problem on a Compaq AlphaServer ("halem.gsfc.nasa.gov") are shown in Table 1. We compute a scaled efficiency by comparing the CPU times between two runs which differ by a factor of two in base grid size and a factor of 8 in number of processors. The resulting efficiencies are shown in Table 2.

REFERENCES

1. D Martin and P. Colella, J Comp Phys. No. (2000).
2. C A Rendleman et al, Computing and Visualization in Science 3 (2000), 147.
3. R Cortez and M Minion, J Comp Phys 161 (2000) 428.
4. R A James, J Comp Phys 25 (1977).
5. E H Twizell, A B Gumel, and M A Arigu, Advances in Comput. Math. 6 (1996) 333.
6. P Colella et al, "Chombo Software Package for AMR Applications", available at http://seesar.lbl.gov/ANAG/software.html.