A fast parallel Poisson solver on irregular domains

Peter Arbenz

ETH Zurich
Chair of Computational Science

University of Utrecht, November 16, 2011
Outline

1. Motivation and problem statement
2. The ‘classical’ FFT-based direct solver
3. Our own iterative solver
4. Numerical results
5. Summary and outlook
Outline

1. Motivation and problem statement
2. The ‘classical’ FFT-based direct solver
3. Our own iterative solver
4. Numerical results
5. Summary and outlook
Motivation from beam dynamics

Vlasov-Poisson formulation for particle evolution

- In physical devices like accelerators $10^9 \ldots 10^{14}$ (or more) charged particles are accelerated in electric fields.
- Instead of computing with individual particles one considers a particle density $f(x, v, t)$ in phase space (position-velocity $(x, v)$ space).
- The Vlasov equation describes the evolving particle density

$$\frac{df}{dt} = \partial_t f + v \cdot \nabla_x f + \frac{q}{m_0} (E + v \times B) \cdot \nabla_v f = 0,$$

where $E$ and $B$ are electric and magnetic fields, respectively.
- The charged particles are ‘pushed’ by Newton’s law

$$\frac{dx(t)}{dt} = v, \quad \frac{dv(t)}{dt} = \frac{q}{m_0} (E + v \times B).$$
The determination of $E$ and $B$ is done in the co-moving Lorentz frame where $\hat{B} \approx 0$ and

$$\hat{E} = -\nabla \hat{\phi},$$

where the electrostatic potential $\hat{\phi}$ is the solution of the Poisson problem

$$- \Delta \hat{\phi}(x) \equiv -\nabla^2 \hat{\phi}(x) = \frac{\hat{\rho}(x)}{\varepsilon_0},$$

(1)

equipped with appropriate boundary conditions.

The charge densities $\rho$ is proportional to the particle density.
Particle-in-cell (PIC) method in N-body Simulations

- Interpolate individual particle charges to a rectangular grid
- Discretize the Poisson equation by finite differences on the rectangular grid
- This leads to a system of linear equations

\[ Ax = b. \] (2)

\( b \) denotes the interpolated charge densities at the mesh points.

- Solve the Poisson equation on the mesh in a Lorentz frame

\( \mathcal{O}(n \log n) \) operations needed provided that the domain is rectangular.
Real beam-pipes are not rectangular

Boundary value problem

\[ \nabla^2 \phi = -\frac{\rho}{\epsilon_0} \quad \text{in } \Omega \subset \mathbb{R}^3, \]

\[ \phi = 0 \quad \text{on } \Gamma_1 \]

\[ \frac{\partial \phi}{\partial n} + \frac{1}{d} \phi = 0 \quad \text{on } \Gamma_2 \]

- \( \Omega \subset \mathbb{R}^3 \): simply connected computational domain
- \( \epsilon_0 \): the dielectric constant
- \( \Gamma = \Gamma_1 \cup \Gamma_2 \): boundary of \( \Omega \)
- \( d \): distance of bunch centroid to the boundary

\( \Gamma_1 \) is the surface of an

1. Elliptical-shaped beam-pipe
2. Arbitrary beam-pipe element
Outline

Plan for the lecture

1. Let’s first look at the square/rectangular case.
   - Fast Poisson solver.
2. Then, let’s try to devise a good solver for the irregular domain.
   - Iterative solver with multigrid preconditioner.
3. Compare.
   - Numerical values. Is approximation a pipe by a cube accurate enough?
   - Performance of the algorithm on a parallel computer (Cray XT-5).
Finite-difference discretization

The Poisson equation $-\Delta u = f$ (plus some boundary conditions) on a domain with axis-parallel boundary is often solved by finite differences on a rectangular grid. We define a rectangular grid with grid points that are a distance $h$ apart. In each grid point $(x_i, y_j)$ the Laplacian of $u$ is approximated by finite differences:

$$-\Delta u(x_i, y_j) = -\frac{\partial}{\partial x^2} u(x_i, y_j) - \frac{\partial}{\partial y^2} u(x_i, y_j)$$

$$= \frac{-u(x_{i-1}, y_j) + 2u(x_i, y_j) - u(x_{i+1}, y_j)}{h^2}$$

$$+ \frac{-u(x_i, y_{j-1} + 2u(x_i, y_j) - u(x_i, y_{j+1})}{h^2} + \mathcal{O}(h^2)$$

$$\approx \frac{4u(x_i, y_j) - u(x_{i+1}, y_j) - u(x_{i-1}, y_j) - u(x_i, y_{j+1}) - u(x_i, y_{j-1})}{h^2}$$

Let’s assume homogeneous boundary conditions:

$$u(x_0, y_j) = u(x_{n+1}, y_j) = u(x_i, y_0) = u(x_i, y_{n+1}) = 0$$
The discretization in every (interior) grid point is given by

\[ 4u_{\text{center}} - u_{\text{west}} - u_{\text{south}} - u_{\text{east}} - u_{\text{north}} = h^2 \cdot f_{\text{center}} \]

Finite differences are difficult (or cumbersome) to implement if the shapes of the domains get complicated, i.e., if the boundary is not aligned with the coordinate axes.
Structure of a FD matrix

MATLAB `spy` of a matrix discretization of the Poisson equation \(-\Delta u = f\) in \(\Omega = (0, 1)^2\), \(u = 0\) on \(\partial \Omega\), with finite differences on a \(12 \times 12\) grid.
Outline

1. Motivation and problem statement
2. The ‘classical’ FFT-based direct solver
3. Our own iterative solver
4. Numerical results
5. Summary and outlook
The one-dimensional case

1D Poisson equation

Let’s first consider the 1-dimensional case \((n = 7)\):

\[
T_n u = \begin{bmatrix}
2 & -1 \\
-1 & 2 & -1 \\
-1 & 2 & -1 \\
-1 & 2 & -1 \\
-1 & 2 & -1 \\
-1 & 2 & -1 \\
-1 & 2
\end{bmatrix}
\begin{bmatrix}
u_1 \\
u_2 \\
u_3 \\
u_4 \\
u_5 \\
u_6 \\
u_7
\end{bmatrix}
= \begin{bmatrix}
f_1 \\
f_2 \\
f_3 \\
f_4 \\
f_5 \\
f_6 \\
f_7
\end{bmatrix}
= f.
\]

The method of choice to solve this equation is LU or Cholesky factorization plus forward / backward substitution. Cost: \(O(n)\)

If \(T = LL^T\) then \(u = L^{-T}(L^{-1}f)\).

How do we solve such a system in parallel? What if many systems with the same matrix are to be solved (multiple right hand sides)?
The one-dimensional case (cont.)

A weird (?) approach

- Let
  \[ T = Q\Lambda Q^T, \quad \Lambda = \text{diag}(\lambda_1^{(n)}, \ldots, \lambda_n^{(n)}) \]
  be the spectral decomposition of \( T \). Here, \( Q \) is orthogonal, i.e., \( Q^T = Q^{-1} \).
  Then \( u = Q\Lambda^{-1}Q^Tf \).

- Cost of the ‘naive’ approach: \( O(n^2) \) without computing the spectral decomposition.

- However: \( Q \) and \( \Lambda \) are known \textit{a priori}:
  \[ \lambda_i = 2\left(1 + \cos \frac{i\pi}{n+1}\right), \quad q_{ij} = \frac{1}{2} \sin \frac{ij\pi}{n+1} \quad i, j = 1, \ldots, n. \]

- Furthermore, the \( q_{ij} \) are the coefficients that appear in the Fourier transform of size \( 2(n+1) \). Therefore, multiplication with \( Q \) (and \( Q^T \)) can be done via the Fourier transform.

- If \( n \) is chosen properly, the \textit{FFT} (or Fast Sine Transform) can be applied.

- So, the cost of the above is just \( O(n \log n) \).
The two-dimensional case

Linear equation formulation

In 2D the Poisson equation discretized by finite differences is structureblock tridiagonal,

\[ Au = f, \quad u, f \in \mathbb{R}^{n^2}. \]

The vectors \( u \) and \( f \) of length \( n^2 \) both consist of \( n \) subvectors \( u_j \) and \( f_j \) of length \( n \) each of which corresponds to values in a column of grid points in the square domain, cf. p. 10.

\[
\begin{align*}
  u &= \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} & f &= \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}
\end{align*}
\]
The two-dimensional case (cont.)

Geometric formulation (Sylvester equation)

Let \( U = [u_1, u_2, \ldots, u_n] \) and \( F = [f_1, f_2, \ldots, f_n] \).

Then, we can write the Poisson equation in the form

\[
TU + UT = F.
\]

\[
(TU)_{3,6}
\]

approximates

\[
\frac{\partial}{\partial y^2} u(x_6, y_3).
\]
We use now the spectral decomposition $T = Q\Lambda Q^T$ to obtain

$$Q\Lambda Q^T U + UQ\Lambda Q^T = F$$

or

$$\Lambda \left( Q^T UQ \right) + \left( Q^T UQ \right) \Lambda = \left( Q^T FQ \right)$$

This equation can again interpreted as a matrix equation.
The two-dimensional case (cont.)

Diagonal linear system

\[
\begin{bmatrix}
\lambda_1 I + \Lambda \\
\lambda_2 I + \Lambda \\
\ddots \\
\lambda_n I + \Lambda
\end{bmatrix}
\begin{bmatrix}
\tilde{u}_1 \\
\tilde{u}_2 \\
\vdots \\
\tilde{u}_n
\end{bmatrix}
= 
\begin{bmatrix}
\tilde{f}_1 \\
\tilde{f}_2 \\
\vdots \\
\tilde{f}_n
\end{bmatrix}
\]

The matrix on the left has all possible sums \(\lambda_i + \lambda_j\) with \(i, j = 1, \ldots, n\) in its diagonal.
This diagonal linear system of equations is evidently trivial to solve.

Remark: Kronecker product

We can write the matrix on the left as \(\Lambda \otimes I + I \otimes \Lambda\).
In the same way we can write the original matrix as \(A = T \otimes I + I \otimes T\).

Definition of the Kronecker product

Let $A \in \mathbb{R}^{n \times m}$ and $B \in \mathbb{R}^{p \times q}$. Then

$$A \otimes B := \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1m}B \\ a_{21}B & a_{22}B & \cdots & a_{2m}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}B & a_{n2}B & \cdots & a_{nm}B \end{bmatrix} \in \mathbb{R}^{pn \times qm}.$$
The algorithm proceeds in three steps.

1. Given $F$, compute $\tilde{F} = Q^T F Q$.
   This amounts to applying $n$ FFT’s from both left and right.

2. Solve the diagonal linear system, $(\Lambda \otimes I + I \otimes \Lambda) \tilde{u} = \tilde{f}$.
   This translates into
   \[
   \tilde{u}_{ij} = \frac{\tilde{f}_{ij}}{\lambda_i + \lambda_j}.
   \]

3. Compute $U = Q \tilde{U} Q^T$.
   This amounts to applying $n$ FFT’s from both left and right.
Parallel Algorithm

The parallel algorithm is organized such that the Fourier transforms are always executed locally. The algorithm proceeds in seven steps. We write $X$ for $F, \tilde{F}, U, \tilde{U}$.

Assume that $X$ is stored in block rows.

1. Compute $X \leftarrow XQ$.
2. Transpose $X$: $X \leftarrow X^T$.
3. Compute $X \leftarrow XQ$.
4. Divide by the eigenvalues.
5. Compute $X \leftarrow XQ^T$.
6. Transpose $X$: $X \leftarrow X^T$.
7. Compute $X \leftarrow XQ^T$. 
The parallel 3-dimensional FFT

Let $X$ be a 3-dimensional $n \times n \times n$ array.

We want to apply $n^2$ FFTs in each direction.
Let us assume that we have a $\sqrt{p} \times \sqrt{p}$ processor grid.

**Step 1.** $n^2$ FFTs in $z$-direction

**Step 2.** Transposition in the $y$-$z$ plane. Here, we assume a simple ring topology. All-to-all communication needs $\sqrt{p} - 1$ steps.

**Step 3.** $n^2$ FFTs in $y$-direction
Step 4. Transposition in the $x$-$y$ plane

Step 5. $n^2$ FFTs in $x$-direction
An alternative

Step 1. $n/p$ 2D FFTs in slices.

Step 2. “all-to-all personalized” communication with blocks of size $\left( \frac{n}{\sqrt{p}} \right)^2 \times n$.

Step 3. $n^2/p$ 1D FFTs in last direction.

The complexity is the same as above. But there is just one step of communication.
Outline

1 Motivation and problem statement
2 The ‘classical’ FFT-based direct solver
3 Our own iterative solver
4 Numerical results
5 Summary and outlook
Real beam-pipes are not rectangular [redux]

Boundary value problem

\[ \nabla^2 \phi = -\frac{\rho}{\epsilon_0} \quad \text{in } \Omega \subset \mathbb{R}^3, \]

\[ \phi = 0 \quad \text{on } \Gamma_1 \]

\[ \frac{\partial \phi}{\partial n} + \frac{1}{d} \phi = 0 \quad \text{on } \Gamma_2 \]

- \( \Omega \subset \mathbb{R}^3 \): simply connected computational domain
- \( \epsilon_0 \): the dielectric constant
- \( \Gamma = \Gamma_1 \cup \Gamma_2 \): boundary of \( \Omega \)
- \( d \): distance of bunch centroid to the boundary

\( \Gamma_1 \) is the surface of an

1. Elliptical-shaped beam-pipe
2. Arbitrary beam-pipe element
Goal

Divise an efficient iterative solver for the Poisson equation on irregular domains

- Solve anisotropic electrostatic Poisson equation with an iterative solver
- Easy to specify boundary surface
- **Irregular** domain imbedded in a rectangular grid.
- “Exact” (Dirichlet) boundary conditions
- Achieving good parallel efficiency
- Reuse information available from previous time steps
Finite-difference discretization

1. Second order finite difference scheme: standard 7-point stencil on 3D Cartesian grid.

\[ x' \quad x^* \quad x \quad x'' \]

1. **Constant extrapolation**: \( u(x') = u(x^*) \) and \( x^* \in \Gamma_1 \)
2. **Linear extrapolation**: \( u(x') \) is obtained by linear interpolation of \( u(x) \) and \( u(x^*) \)
   System matrix \( A \) is symmetric positive definite.
3. **Quadratic extrapolation** (Shortley-Weller approximation): \( u(x') \) is obtained by quadratic interpolation of \( u(x) \), \( u(x'') \), and \( u(x^*) \)
   → non-symmetric stencil
   System matrix \( A \) is positive definite but not symmetric. (\( A \) is still an \( M \)-matrix)
Solver

- Large sparse linear system
  \[ Ax = b \]
  with \( A \) (non-)symmetric positive definite.
- Conjugate gradient iterative solver method of choice (?)
- Preconditioned by smoothed aggregation-based algebraic multigrid
Implementation

For preconditioner setup and iterative solver we used TRILINOS (see http://trilinos.sandia.gov)

- **EPETRA**: distributed matrices and vectors
- **AMESOS**: direct coarse level solver
- **AZTECOO**: iterative solver
- **ML**: smoothed aggregation based AMG preconditioner
- **ISORROPIA**: partitioning and load balancing

The Object Oriented Parallel Accelerator Library Framework (OPAL) provides a partitioning of the data based on the underlying rectangular grid. (See http://amas.web.psi.ch/docs/opal)
Multilevel: a simple multigrid V-cycle

1: Approximately solve $K_\ell u = b$ where $\ell$ is the current grid level.
2: procedure multilevel($K_\ell, b_\ell, u_\ell, \ell$)
3:  if $\ell < L$ then
4:     $u_\ell = S_\ell(K_\ell, b_\ell, u_\ell)$; \hspace{1cm} \{Presmoothing\}
5:     $r_\ell = R_\ell(b_\ell - K_\ell u_\ell)$;
6:     $v_{\ell+1} = 0$;
7:     multilevel($K_{\ell+1}, r_{\ell+1}, v_{\ell+1}, \ell + 1$);
8:     $u_\ell = u_\ell + P_\ell v_{\ell+1}$;
9:  end if
10: else
11:     Solve $K_\ell u_\ell = b_\ell$;
12: end if

Preconditioner: Call procedure multilevel($K_0 = K, b, u = 0, L$)
Setup procedure for an abstract multigrid solver

1: Define the number of levels, $L$
2: for level $\ell = 0, \ldots, L - 1$ do
3: \hspace{1em} if $\ell < L - 1$ then
4: \hspace{2em} Define prolongator $P_\ell$;
5: \hspace{2em} Define restriction $R_\ell = P_\ell^T$;
6: \hspace{2em} $K_{\ell+1} = R_\ell K_\ell P_\ell$;
7: \hspace{1em} Define smoother $S_\ell$;
8: \hspace{1em} else
9: \hspace{2em} Prepare for solving with $K_\ell$;
10: \hspace{1em} end if
11: end for
AMG parameters

- “Decoupled” aggregation scheme: aggregates of size $3 \times 3 \times 3$
  - Each processor aggregates its portion of the grid
  - Many aggregates near inter-processor boundaries with non-optimal size
  - Number of vertices is substantially reduced in every coarsening step

  - Estimates for the spectrum of the matrices $A_k$ are needed.

- LU based direct coarse level solver or a few steps of Gauss-Seidel iteration
  - Matrices $A_k$ tend to get dense with increasing level.

- V-cycle.
- Starting vector.

AMG performance critically depends on choice of parameters!
Integration of the solver in OPAL

- Method entry point
- Redistribute solution of last time-step
- Build stencil and RHS
- Build hierarchy
- Build multilevel preconditioner
- Solve the system using LHS as initial guess
- Write solution to IPPL grid
- Store LHS

Reuse preconditioner
Reuse hierarchy

OPAL

MGPoissonSolver
Outline

1 Motivation and problem statement
2 The ‘classical’ FFT-based direct solver
3 Our own iterative solver
4 Numerical results
5 Summary and outlook
Buin: Cray XT4 cluster at the CSCS in Manno (Switzerland)

- 468 AMD dual core Opteron at 2.6 GHz
- 936 GB DDR RAM
- 30 TB Disk
- 7.6 GB/s interconnect bandwidth

Create stencil 7.8%
Create map 2.8%
computeSelfField (301.21s)
ML 26.2%
CG 62.1%
Validation of the Solver

For validation purposes we investigated an axi-symmetric problem with known analytical solution.

<table>
<thead>
<tr>
<th>$h$</th>
<th>$|e_h|_2$</th>
<th>$r$</th>
<th>$|e_h|_\infty$</th>
<th>$r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/64</td>
<td>$2.162 \cdot 10^{-3}$</td>
<td>—</td>
<td>$7.647 \cdot 10^{-3}$</td>
<td>—</td>
</tr>
<tr>
<td>1/128</td>
<td>$1.240 \cdot 10^{-3}$</td>
<td>0.80</td>
<td>$4.153 \cdot 10^{-3}$</td>
<td>0.88</td>
</tr>
<tr>
<td>1/64</td>
<td>$2.460 \cdot 10^{-5}$</td>
<td>—</td>
<td>$6.020 \cdot 10^{-5}$</td>
<td>—</td>
</tr>
<tr>
<td>1/128</td>
<td>$6.226 \cdot 10^{-6}$</td>
<td>1.98</td>
<td>$1.437 \cdot 10^{-5}$</td>
<td>2.07</td>
</tr>
<tr>
<td>1/64</td>
<td>$5.581 \cdot 10^{-6}$</td>
<td>—</td>
<td>$1.689 \cdot 10^{-5}$</td>
<td>—</td>
</tr>
<tr>
<td>1/128</td>
<td>$1.384 \cdot 10^{-7}$</td>
<td>2.01</td>
<td>$4.550 \cdot 10^{-6}$</td>
<td>1.89</td>
</tr>
</tbody>
</table>

Solution error for constant (top), linear (middle), quadratic (bottom) extrapolation.

The convergence rate $r$ is defined by

$$r = \log_2 \left( \frac{\|e_{2h}\|}{\|e_h\|} \right)$$
Comparison with FFT-based Poisson solver

Simulation timings of one solve in the first and second time step, respectively.

<table>
<thead>
<tr>
<th>solver</th>
<th>reusing</th>
<th>mesh size</th>
<th>mesh points</th>
<th>first [s]</th>
<th>second [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT</td>
<td>—</td>
<td>$128 \times 128 \times 256$</td>
<td>4,194,304</td>
<td>12.3</td>
<td>—</td>
</tr>
<tr>
<td>AMG</td>
<td>—</td>
<td>$128 \times 128 \times 256$</td>
<td>3,236,864</td>
<td>49.9</td>
<td>42.2</td>
</tr>
<tr>
<td>AMG hierarchy</td>
<td>—</td>
<td>$128 \times 128 \times 256$</td>
<td>3,236,864</td>
<td>—</td>
<td>35.5</td>
</tr>
<tr>
<td>AMG preconditioner</td>
<td>—</td>
<td>$128 \times 128 \times 256$</td>
<td>3,236,864</td>
<td>—</td>
<td>28.2</td>
</tr>
<tr>
<td>AMG</td>
<td>—</td>
<td>$166 \times 166 \times 256$</td>
<td>5,462,016</td>
<td>81.8</td>
<td>71.2</td>
</tr>
<tr>
<td>AMG hierarchy</td>
<td>—</td>
<td>$166 \times 166 \times 256$</td>
<td>5,462,016</td>
<td>—</td>
<td>60.4</td>
</tr>
<tr>
<td>AMG preconditioner</td>
<td>—</td>
<td>$166 \times 166 \times 256$</td>
<td>5,462,016</td>
<td>—</td>
<td>43.8</td>
</tr>
</tbody>
</table>

Equal number of mesh points (above) and equal mesh spacings (below) for FFT and AMG.
Parallel efficiency

- Obtained for a tube embedded in a $512 \times 512 \times 512$ grid
- ML construction phase is performing poorly
- Influence of problem size on the low performance of the aggregation in ML.
- Problem is too small for the number of processors used.
Parallel efficiency (cont.)

- Obtained for a tube embedded in a $1024 \times 1024 \times 1024$ grid
- Construction phase is performing the worst with an efficiency of 73%
- Problem size is appropriate for the number of processors.
Impact on Physics of OPAL Simulations

- Statistics of the distance of the particles to the $z$-axis.
- Shift of the beam size minimum (beam waist) towards larger $z$ values.
- A smaller minimum $\rightarrow$ self forces are larger when considering the beam pipe.
- Beam pipe radius is an important optimization quantity.
Load balance issues

Data is distributed according to the underlying rectangular grid (induced by the particle code OPAL).

Severe load imbalance.

However, the work of the most heavily loaded core decreases linearly with the number of processors!

Speedup looks good!

Note that we do not have a 1-processor run. (Memory!)
Load balance issues (cont.)

- Isorropia’s recursive coordinate bisection (RCB) algorithm distributes the data perfectly balanced.
- But not entirely along coordinate axis!
- Number of neighbors (messages) increases
- Execution times decrease
- Better load balance (less work for the most loaded processors)
Comparison

<table>
<thead>
<tr>
<th>cores</th>
<th>solution</th>
<th>construction</th>
<th>application</th>
<th>total ML</th>
<th>iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>63.12 [1.00]</td>
<td>32.09 [1.00]</td>
<td>52.73 [1.00]</td>
<td>84.80 [1.00]</td>
<td>20</td>
</tr>
<tr>
<td>1024</td>
<td>33.54 [0.94]</td>
<td>16.31 [0.98]</td>
<td>28.04 [0.94]</td>
<td>44.35 [0.96]</td>
<td>20</td>
</tr>
<tr>
<td>2048</td>
<td>18.56 [0.85]</td>
<td>8.10 [0.99]</td>
<td>15.66 [0.84]</td>
<td>23.76 [0.89]</td>
<td>21</td>
</tr>
</tbody>
</table>

Table 1: Times in seconds and relative parallel efficiencies. The original data distribution is used, and the coarsest AMG level is solved iteratively.

<table>
<thead>
<tr>
<th>cores</th>
<th>solution</th>
<th>construction</th>
<th>application</th>
<th>total ML</th>
<th>iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>51.08 [1.00]</td>
<td>25.65 [1.00]</td>
<td>44.89 [1.00]</td>
<td>70.55 [1.00]</td>
<td>20</td>
</tr>
<tr>
<td>1024</td>
<td>27.38 [0.93]</td>
<td>12.96 [0.99]</td>
<td>24.51 [0.92]</td>
<td>37.07 [0.95]</td>
<td>20</td>
</tr>
<tr>
<td>2048</td>
<td>14.76 [0.87]</td>
<td>6.69 [0.96]</td>
<td>13.10 [0.86]</td>
<td>19.79 [0.89]</td>
<td>19</td>
</tr>
</tbody>
</table>

Table 2: Times in seconds and relative parallel efficiencies. Data is distributed by RCB. The coarsest AMG level is solved iteratively.
Summary

- Conjugate gradient solver for Poisson equation on rectangular grid with special treatment of irregular boundary.
- Elliptic and arbitrary domains based on real geometries.
- Smoothed aggregation based algebraic Multigrid preconditioning.
- Non-symmetric equations resulting from quadratic boundary treatment converge well with PCG.
- Reducing time to solution (20 and 40%) by reusing hierarchy or preconditioner.
- Attaining good parallel efficiency: 73% for the worst performing phase.
- Compared to FFT more flexibilities for only a small performance loss.
- Considerable impact on physics (in particular, for narrow beam pipes).
- Further Work: Adaptive mesh refinement (AMR).