Implementation of a Particle in Cell algorithm in PEANO

Dirk Roose, Bart Verleye, Denis Jarema, Pawan Kumar
In close cooperation with Tobias Weinzierl and Kristof Unterweger of the TU Munich

Intel Exascience lab Leuven
Introduction

Aims: A parallel AMR based fluid code, with the possibility to refine the physics locally by means of an implicit electromagnetic particle in cell algorithm. (Parallel) Ingredients:
  ▶ AMR datastructure
  ▶ PDE solver
  ▶ Multi-layer data structure: b.c. exchange between fluid and PIC
  ▶ Dynamic load balancer
A parallel grid traversal package:

Features:
- AMR and Multi-level data structure
- Parallel aspects, e.g. MPI, hidden from the user
- Hybride
- Efficient cache usage
- Low memory overhead

Constraints:
- Cell based
- Random order of cell access

Not included:
- Solvers (it is not PETSc)
- Optimal load-balancers
- Data structures for particles

Figure 2.2: $k$-spacetree construction scheme for four different computational domains ($d=2, k=3$). Each illustration consists of four layers top-down: The layers one, two and three correspond to the three initial recursion steps, i.e. they show the spacetrees of height one, two and three. The bottom level shows a significant finer resolution of the domain.
Grid traversal
What happens during a traversal

- before the traversal begins, global values, e.g. the time-step, are loaded from the state.
- before a cell is handled, all vertices of the cell are touched.
- the user defines what happens when a vertex is touched the first time. This is only executed once per iteration, and thus not for every cell it is bordering.
- once all vertices are touched, the cell is handled, e.g. the new potential is computed.
- once all cells that border a vertex are handled, touchVertexLastTime is executed.
- once all cells are handled, global values in the state can be changed.
What happens during a traversal

1. Touch vertex 1 first time
2. Touch vertex 2 first time
3. Touch vertex 6 first time
4. Touch vertex 5 first time
5. Handle cell A
6. Touch vertex 3 first time
7. Touch vertex 4 first time
8. Handle cell B
9. ...continue till cells C and D are handled
10. Touch vertices last time in reverse order
Particle in Cell Simulation

- Explicit or implicit
- Electrostatic or electromagnetic
- Large grids: e.g. $100^3$
- For accurate results, as many particles per cell as possible, e.g. 3 species, 200 particles/species/cell
- Particles begin evenly distributed, but can group → need for an optimal dynamic loadbalancer
Explicit electrostatic Particle in Cell:

- Implicit electromagnetic; too hard to begin with
- Explicit electromagnetic: no solver needed, so not a good example
- Explicit electrostatic: physically not very useful but has all the good ingredients

On a regular grid:
PEANO is in development, Space-tree + MPI was not yet included in PEANO-II
Particle to Grid

- The density of the particles determine the RHS of the Poisson equation
- A particle has the same size as a cell
- 1 particle contributes to 4 cells
- e.g. the red part is the proportion of the particle’s density that is contributed to cell 1
- This contribution has to be computed for every particle and every cell
Update velocity and location

- From the potential, the electric field in every direction is computed
- The electric field determines the new velocity of the particles
- The inverse of the particle to cell operation
- The update of the location of the particles is a particle only operation, once the velocity and time-step are known
- For every particle an extra check is needed to see if it has to move vertex
A Particle data structure was not provided in PEANO

All vertices are always fully loaded into the cache, which results in a very low efficiency for the SOR cycles

New feature in PEANO: the heap, i.e. a pointer to a vector

Particles are shared among 4 cells, and are therefore stored in the common vertex

A cell has access to 8 vertices (3D), and can move a particle from one vertex to another

If a particle has to be moved further:

- On a regular grid, different iterations are needed to put all such particles in place
- On a spacetree grid, the particles move via the higher level(s)
On a spacetree grid, the particles move via the higher level(s): The particle moves from the finest green level, via the higher blue level to the orange level, and then back via blue to the green level.
The general approach:

- Define required data for cells and vertices in *definition* files
- Define mappings/adapters in *definition* files, e.g. SOR-step-mapping
- Pre-process → glue code
- Implement events that happen during the traversal of the grid: touch-vertex, handle-cell, ...
- Compile with gcc
- Add locks, compile with tbb or openmp
- Implement `mergeWithNeighbour (c++)`
- Compile with MPI or hybrid
- Tuning parameters: Grain size (multicore), size of the MPI message as number of vertices (MPI)
The average of the potential in a cell, and the potential of the neighboring cell are stored in a common vertex so that the stencil can be computed:

\[ m_{ab} = \frac{f_a + f_b}{2} \]

\[ m_{ab}^{(n+1)} = m_{ab}^{(n)} + \frac{f_b^{(n+1)} - f_b^{(n)}}{2} \]
Distributed updates at the partition boundaries

- At the start, old values in ABCD, and averages of old values in vertices 2, 5, 8, 4 and 2bis, 5bis and 8bis.
- If A is updated, it will use the old value of D and B, stored in vertex 5, and update the values in vertex 5. However, vertex 5bis is not updated at this time.
- D can now be updated with the new value of A, stored in vertex 5.
- B however, cannot use the new A, as the vertices at the boundaries are merged only after the mapping is completed.
- Residual is updated during the traversal, i.e. not always with the updated neighbor values
- If distributed memory: at the partition boundaries, always old values are used
Why a CG solver in Peano?

▶ Technical reason
  ▶ Inconvenient to assemble the matrix
  ▶ Even if we manage, the matrix is large (>10x)
  ▶ Fits in the multi-level and multi-physics framework

▶ Other reasons
  ▶ Fast development time, most glue codes auto-magically created
  ▶ A reliable and tested software platform
  ▶ Developers fast to respond to the issues
Numerical experiments: CG in Peano

- Scalability of 5 iterations of CG
- Grain size = 500 (for tbb autotuning)

Figure: Scalability of 5 iterations of CG in Peano framework
Numerical experiments: CG in Peano

- Scalability of 5 iterations of CG
- Grain size = 500 (for tbb autotuning)

**Figure:** Scalability of 5 iterations of CG in Peano framework
Numerical experiments: CG in Peano

- Scalability of 5 iterations of CG
- Grain size = 500 (for tbb auto-tuning)

**Figure:** Scalability of 5 iterations of CG in Peano framework
Comments on CG results in Peano

- Sequentially about 5-6 times slower than matrix based method
  - Reason: insufficient in-lining (to be fixed with Peano developers)
  - Dot product involves several function calls
- Scalability suffers from NUMA
- But could solve a problem of size 10x larger than matrix based methods
- Possible future work
  - Implement preconditioners for Poisson solver on adaptive grid
  - Collaborate on tiling/blocking techniques in Peano (with Wim, Peter)
  - Keep vector operations (dot product) outside Peano? (as in iPIC3D)
  - Optimize the code for MIC and/or resolve NUMA issue (with Peano dev.)
Shared memory results

- PIC time step: time (s) for each traversal (speed-up between brackets)
- Averaged over 10 time steps
- Total = time for 1 time step with 20 SOR iterations

<table>
<thead>
<tr>
<th>nb cores</th>
<th>Part. To Grid</th>
<th>SOR</th>
<th>Update E-field and Velocity</th>
<th>Update Location</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>201</td>
<td>0.34</td>
<td>70</td>
<td>206</td>
<td>483</td>
</tr>
<tr>
<td>4</td>
<td>48 (4.1)</td>
<td>0.17</td>
<td>18 (3.8)</td>
<td>48 (4.2)</td>
<td>117 (4.1)</td>
</tr>
<tr>
<td>8</td>
<td>23 (8.7)</td>
<td>0.14</td>
<td>10 (7.0)</td>
<td>24 (8.5)</td>
<td>59 (8.1)</td>
</tr>
<tr>
<td>12</td>
<td>19 (10.5)</td>
<td>0.15</td>
<td>12 (5.8)</td>
<td>18 (11.4)</td>
<td>52 (9.2)</td>
</tr>
</tbody>
</table>

- $64^3$ grid, 600 particles/vertex = 157 286 400 particles
- Constant Grainsize: 500, might not be optimal for the 12 core simulation
- On Lynx: Intel X5660 2.8Ghz node
Distributed memory results

- PIC time step: time (s) for each traversal (speed-up between brackets)
- Averaged over 10 time steps
- Total = time for 1 time step with 20 SOR iterations

<table>
<thead>
<tr>
<th>nb cores</th>
<th>Part. To Grid</th>
<th>SOR</th>
<th>Update E-field and Velocity</th>
<th>Update Location</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>201</td>
<td>0.34</td>
<td>70</td>
<td>206</td>
<td>483</td>
</tr>
<tr>
<td>4</td>
<td>34 (5.9)</td>
<td>0.13 (2.4)</td>
<td>17 (3.8)</td>
<td>34 (5.8)</td>
<td>89 (5.38)</td>
</tr>
<tr>
<td>1x8</td>
<td>16 (11.8)</td>
<td>0.081 (4.1)</td>
<td>9 (7.1)</td>
<td>17 (12.1)</td>
<td>45 (10.6)</td>
</tr>
<tr>
<td>2x4</td>
<td>16 (12.0)</td>
<td>0.083 (4.0)</td>
<td>9 (7.2)</td>
<td>16 (12.2)</td>
<td>44 (10.8)</td>
</tr>
<tr>
<td>4x2</td>
<td>16 (112.4)</td>
<td>0.076 (4.4)</td>
<td>8 (7.9)</td>
<td>16 (12.5)</td>
<td>42 (11.3)</td>
</tr>
<tr>
<td>8x1</td>
<td>16 (12.5)</td>
<td>0.071 (4.7)</td>
<td>8.5 (8.2)</td>
<td>16 (13.7)</td>
<td>40 (11.8)</td>
</tr>
<tr>
<td>64</td>
<td>2.7 (74.4)</td>
<td>0.066 (5.1)</td>
<td>2.0 (35.0)</td>
<td>2.3 (89.5)</td>
<td>8.3 (58.1)</td>
</tr>
<tr>
<td>125</td>
<td>2.4 (83.7)</td>
<td>0.30 (1.1)</td>
<td>1.7 (41.1)</td>
<td>1.2 (171.6)</td>
<td>11 (42.8)</td>
</tr>
</tbody>
</table>

- $64^3$ grid, 600 particles/vertex = 157 286 400 particles
- On Lynx: Intel X5660 2.8Ghz nodes
Time for each 'traversal' (MPI version)

- **SOR**
- **Update E-field and Velocity**
- **Update Location**
- **Particle to Grid**
- **Total**

**Number of cores**

- $1 \times 10^{-2}$
- $1 \times 10^{-1}$
- $1 \times 10^0$
- $1 \times 10^1$
- $1 \times 10^2$
- $1 \times 10^3$

**Time (s)**

- $1 \times 10^3$
- $1 \times 10^2$
- $1 \times 10^1$
- $1 \times 10^0$
- $1 \times 10^{-1}$
- $1 \times 10^{-2}$
Implementation of a Particle in Cell algorithm in PEANO
Implementation of a Particle in Cell algorithm in PEANO
Implementation of a Particle in Cell algorithm in PEANO
Initiation and boundary conditions

- **Particles**
  - Initial position: Random around the vertex location, inside the vertex coverage
  - Initial velocity: Random
  - Boundary condition: Bounce back

- **Potential**
  - Boundary condition: zero Dirichlet

- **Tests**
  - Number of particles stays constant
  - Total density stays constant
  - Conservation of momentum: When particles are initiated only in the middle, momentum stays constant until particles hit the wall