Particle-In-Cell simulations on the GPU
Kai Germaschewski, University of New Hampshire, Durham, USA
Hartmut Ruhl, Ludwig-Maximilians-University, Munich, Germany

Introduction

The Particle-In-Cell (PIC) method is an established and versatile approach to the kinetic simulation of plasma. The method makes use of quasi-elements to approximate one-particle distribution functions and a grid to solve Maxwell’s equations. The approach is computationally expensive. For most applications the computational load in the particles exceeds the one in the Maxwell solver by far. Hence, we focus on a GPU implementation of particle pushing. While numerous implementations of the PIC method on classical distributed compute platforms exist GPU implementations are still rare. Here, we present a CUDA implementation of a quasi-particle pusher on a GPU and compare its performance with an SSE2-optimized CPU version of the latter.

Equations of motion

The distribution function based on quasi-elements and the equations of motion for quasi-elements are given by

\[ f(x,E) = \frac{3}{2} \sum \left( \frac{E-E_0}{T_0} \right)^2 \left( \begin{array}{c} \frac{1}{2} \left( \frac{x-x_0}{L_0} \right)^2 + \left( \frac{E-E_0}{T_0} \right) \frac{1}{2} \left( \frac{x-x_0}{L_0} \right) \right) \right) \]

where \( f(x,E_0) = \frac{3}{2} \sum \left( \frac{E-E_0}{T_0} \right)^2 \left( \begin{array}{c} \frac{1}{2} \left( \frac{x-x_0}{L_0} \right)^2 + \left( \frac{E-E_0}{T_0} \right) \frac{1}{2} \left( \frac{x-x_0}{L_0} \right) \right) \right) \]

The particle loop consists of loading the particle data, loading and interpolating the field data to the particle location as indicated in section “Equations of motion”, updating the particles and storing the data back. Global memory is used since particle data are only needed once in the particle loop. Particle data in global memory are arranged in a way that memory access is coalesced. The measured bandwidth is 1.4 billion particles/sec. The field data are stored in shared memory since typically many particles use the same field values. Up to 125 field values for each field component are needed in 3D. To make field caching possible particles need to be sorted to cells. The following table shows results for particle pushing on a TESLA C1060 card with and without field caching (see kernels 1 and 2)

<table>
<thead>
<tr>
<th>Kernel 1: no field caching</th>
<th>1D current aggregation</th>
<th>1D current aggregation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D current aggregation</td>
<td>4.99 \times 10^{10}</td>
<td>9.11 \times 10^{10}</td>
</tr>
<tr>
<td>2D particle push and current aggregation, SSE2</td>
<td>1.92 \times 10^{11}</td>
<td>2.94 \times 10^{11}</td>
</tr>
<tr>
<td>2D particle push and current aggregation, CUDA</td>
<td>7.96 \times 10^{10}</td>
<td>7.96 \times 10^{10}</td>
</tr>
</tbody>
</table>

Example: Wake field simulation

To demonstrate the performance of our GPU implementation we pick the example of a laser-driven wake field simulation. A wake field occurs when a heavy charged fluid (here ions) and light charged fluid (here electrons) is perturbed by a laser pulse shorter than the plasma wavelength. The plasma is sub-critical so that the laser can propagate through it. The simulation parameters are

\[ \begin{array}{c|c}
\text{parameter} & \text{value} \\
\hline
\text{particle density} & 10^{20} \text{cm}^{-3} \\
\text{laser intensity} & 10^{17} \text{W/cm}^2 \\
\text{plasma length} & 300 \text{cm} \\
\text{plasma number} & 2.75 \times 10^{19} \\
\text{need-see} & 300 \\
\text{current-deposition scheme} & \text{current conserving} \\
\end{array} \]