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

$$f_k(\vec{x},\vec{p},t) = \frac{n_0}{N_c} \sum_{i=1}^{N_k} \phi(\vec{x}-\vec{x}_i^{\,k}(t))\,\delta^3(\vec{p}-\vec{p}_i^{\,k}(t))$$

$$\phi(\vec{x}-\vec{x}_i^{\,k}(t)) = \Pi_{j=1}^3\, S_j(x_j, x_{ij}^k(t))$$

$$S_j(x_j, x_{ij}^k(t)) = \begin{cases} 1-\left|\frac{x_j - x_{ij}^k(t)}{\Delta x_j}\right| & ,\ x_{ij}^k(t)-\Delta x_j \le x_j \le x_{ij}^k(t)+\Delta x_j \\ 0 & ,\ \text{else} \end{cases}$$

$$\frac{d\vec{x}_i(t)}{dt} = \vec{v}_i(t)$$

$$\frac{d\vec{p}_i(t)}{dt} = \frac{q}{\Pi_{n=1}^3 \Delta x_n} \int_{x_i-\Delta x}^{x_i+\Delta x} dx \int_{y_i-\Delta x}^{y_i+\Delta x} dy \int_{z_i-\Delta x}^{z_i+\Delta x} dz\, \phi(\vec{x}-\vec{x}_i(t))\,[\vec{E}(\vec{x},t)+\vec{v}_i(t)\times\vec{B}(\vec{x},t)] = \frac{dW_{jkl}(t)}{dt}$$

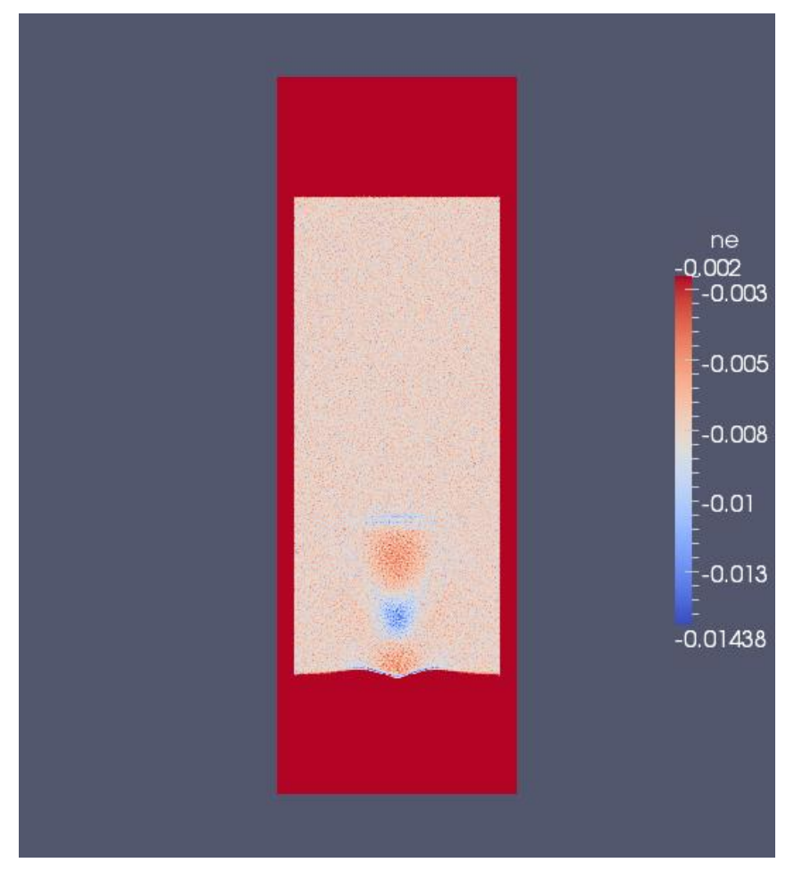
$$\vec{W}_{jkl}(t) = \frac{1}{2}\,\vec{V}_{jkl-1}(t)\left(\frac{1}{2}+\frac{z_l-z_i}{\Delta z}\right)^2 + \vec{V}_{jkl}(t)\left(\frac{3}{4}-\frac{(z_l-z_i)^2}{\Delta z^2}\right) + \frac{1}{2}\,\vec{V}_{jkl+1}(t)\left(\frac{1}{2}-\frac{z_l-z_i}{\Delta z}\right)^2$$

$$\vec{V}_{jkl}(t) = \frac{1}{2}\,\vec{U}_{jk-1l}(t)\left(\frac{1}{2}+\frac{y_k-y_i}{\Delta y}\right)^2 + \vec{U}_{jkl}(t)\left(\frac{3}{4}-\frac{(y_k-y_i)^2}{\Delta y^2}\right) + \frac{1}{2}\,\vec{U}_{jk+1l}(t)\left(\frac{1}{2}-\frac{y_k-y_i}{\Delta y}\right)^2$$

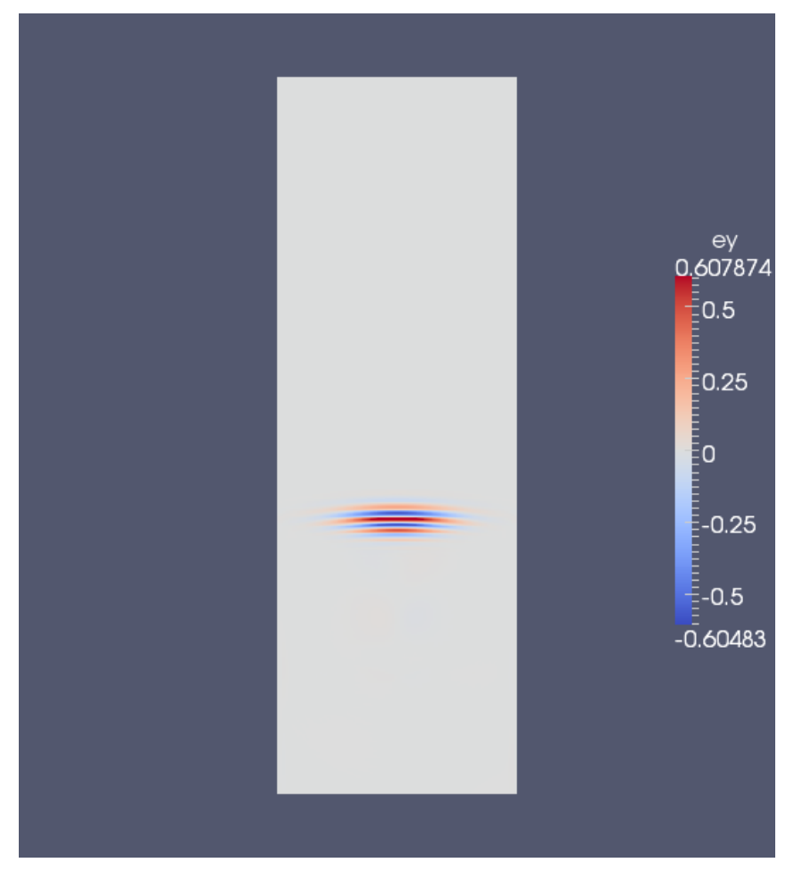
$$\vec{U}_{jkl}(t) = \frac{1}{2}\,\vec{F}_{j-1kl}(t)\left(\frac{1}{2}+\frac{x_j-x_i}{\Delta x}\right)^2 + \vec{F}_{jkl}(t)\left(\frac{3}{4}-\frac{(x_j-x_i)^2}{\Delta x^2}\right) + \frac{1}{2}\,\vec{F}_{j+1kl}(t)\left(\frac{1}{2}-\frac{x_j-x_i}{\Delta x}\right)^2$$

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



parameter	value
electron density	$10^{25}\,\mathrm{m}^{-3}$
laser intensity	$10^{22}\,\mathrm{W/m}^2$
plasma length	$10^{-5}\,\mathrm{m}$
particle number	$2.75 \cdot 10^7$
mesh size	200×600
current deposition scheme	current conserving



Particle pushing on the GPU

$$\vec{x}_i^{\,n} \overset{\vec{v}_i^{\,n}}{\overbrace{\longrightarrow}} \vec{x}_i^{\,n+\frac{1}{2}}$$

$$\vec{p}_i^{\,n} \overset{\vec{E}_{jkl}^{\,n+\frac{1}{2}},\ \vec{B}_{jkl}^{\,n+\frac{1}{2}}}{\overbrace{\longrightarrow}} \vec{p}_i^{\,n+1}$$

$$\vec{x}_i^{\,n+\frac{1}{2}} \overset{\vec{v}_i^{\,n+1}}{\overbrace{\longrightarrow}} \vec{x}_i^{\,n+1}$$

$$\vec{j}_{jkl}^{\,n} \overset{\vec{x}_i^{\,n},\ \vec{x}_i^{\,n+1}}{\overbrace{\longrightarrow}} \vec{j}_{jkl}^{\,n+1}$$

For efficient computation the particle loop (see left) requires partitioning of particle and field data at the same time. Particle advance and current deposition are treated separately. The simplest implementation of the particle loop in CUDA uses a separate thread for each particle to be pushed.

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 band width 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)

kernel	particles/sec
no field caching	$1.99 \cdot 10^8$
cache 1×1 blocks	$6.24 \cdot 10^8$
cache 2×2 blocks	$9.01 \cdot 10^8$
cache 4×4 blocks	$9.72 \cdot 10^8$
cache 8×8 blocks	$9.62 \cdot 10^8$
cache 16×16 blocks	$5.31 \cdot 10^8$

```
Kernel 1: no field caching
m = threadIdx + blockDim * blockIdx
while m < number of particles
    update position $\vec{x}_i^{\,n+\frac{1}{2}} = \vec{x}_i^{\,n} + \frac{\Delta t}{2}\,\vec{v}_i^{\,n}$
    interpolate force $\vec{F}_i^{\,n+\frac{1}{2}}$ making use of $\vec{E}_i^{\,n+\frac{1}{2}}, \vec{B}_i^{\,n+\frac{1}{2}}$
    update momentum $\vec{p}_i^{\,n+1} = \vec{p}_i^{\,n} + \Delta t\,\vec{F}_i^{\,n+\frac{1}{2}}$
    update position $\vec{x}_i^{\,n} = \vec{x}_i^{\,n+\frac{1}{2}} + \frac{\Delta t}{2}\,\vec{v}_i^{\,n+1}$
    m = m + blockDim * gridDim
```

```
Kernel 2: field caching
get block first, block last for blockIdx
cache $\vec{E}^{n+\frac{1}{2}}, \vec{B}^{n+\frac{1}{2}}$ fields for blockIdx in shared mem
for (m = block first + threadIdx; m < block last; m = m + blockDim)
    update position $\vec{x}_i^{\,n+\frac{1}{2}} = \vec{x}_i^{\,n} + \frac{\Delta t}{2}\,\vec{v}_i^{\,n}$
    interpolate force $\vec{F}_i^{\,n+\frac{1}{2}}$ making use of cached $\vec{E}_i^{\,n+\frac{1}{2}}, \vec{B}_i^{\,n+\frac{1}{2}}$
    update momentum $\vec{p}_i^{\,n+1} = \vec{p}_i^{\,n} + \Delta t\,\vec{F}_i^{\,n+\frac{1}{2}}$
    update position $\vec{x}_i^{\,n} = \vec{x}_i^{\,n+\frac{1}{2}} + \frac{\Delta t}{2}\,\vec{v}_i^{\,n+1}$
```

Current aggregation

As particles move each particle contributes to the current density, which is computed on the field mesh and needed to update the electromagnetic fields. Two strategies apply: I) Atomic updates and II) reduction of contributions from all threads within a threadblock. We persue strategy II. The main challenge is that each thread has to compute its current density contribution locally before the reduction into a per threadbock result is possible. Direct reduction in 2D requires 25 floats for each current direction per thread or 300 bytes. Due to shared memory limitations we employ the algorithm sketched below

```
Current aggregation
get cell_first, cell_last for blockIdx
for (m = cell_first + threadIdx; m < cell_last; m = m + blockDim)
    for iz = -2,-1,0,1,2
        calculate $j_x$ (iy = -2,-1,0,1,2, iz)
        reduce $j_x$ over all threads in threadblock
        add $j_x$ to current density $j_x$ in global mem (single thread)
    for iz = -2,-1,0,1,2
        calculate $j_y$ (iy = -2,-1,0,1,2, iz)
        reduce $j_y$ over all threads in threadblock
        add $j_y$ to current density $j_y$ in global mem (single thread)
    for iy = -2,-1,0,1,2
        calculate $j_z$ (iy, iz = -2,-1,0,1,2)
        reduce $j_z$ over all threads in threadblock
        add $j_z$ to current density $j_z$ in global mem (single thread)
```

Results

In 2D the GPU is about 4 times faster than SSE2 optimized code on a recent INTEL XEON CPU

kernel	particles/sec
1D current aggregation	$4.99 \cdot 10^8$
2D current aggregation	$9.11 \cdot 10^7$
2D particle push and current aggregation, SSE2	$1.92 \cdot 10^7$
2D particle push and current aggregation, CUDA	$7.96 \cdot 10^7$

Acknowledgements

This work has been supported by the DFG grant RU 633/3 "AMR-PIC" and the Munich Centre for Advanced Photonics .

Literature

Yu. N. Grigoryev, V. A. Vshivkov, and M. P. Fedoruk, *Numerical "Particle-In-Cell" Methods,* VSP BV, ISBN 90-6764-368-8.
H. Ruhl, *Introduction to Computational Methods in Many Body Physics*, Rinton Press, ISBN 1-58949-009-6.
C. Birdsall and A. B. Langdon, *Plasma Physics via Computer Simulation*. McGraw-Hill, ISBN 0-07-005371-5.