Kinetic Kelvin-Helmholtz Instability

15360 × 7680 cells, 100 particles per cell performed on 900 GPUs (M2090, TitanDev) in ≈ 24 h wallclock
Particle Noise

Discrete particle effects lead to numerical noise which can

- lead to numerical heating
- result in poor resolution of quantities of interest such as the spectrum of turbulence, $\mathbf{E} \cdot \mathbf{J}$, agyrotropy, etc.
Particle-in-cell: Numerical Heating

While physically total energy should be conserved, particle-in-cell simulations suffer from non-physical numerical heating.

- **Finite Grid Instability.** Aliasing of unresolved grid modes gives rise to a numerical instability if the Debye length is not resolved.

- **Stochastic heating.** Particle noise leads to errors in the electromagnetic fields that heat the plasma linearly ($\propto 1/N$).
Numerical Heating: dependence on particle shape

Remedies: Use more particles, or use higher order particles.

### Heating rate

![Graph showing heating rate vs. particles per cell]

### Performance

(16 core AMD Opteron / Nvidia K20X)

<table>
<thead>
<tr>
<th>Order</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>order 2/1.5</td>
<td>23 M/sec</td>
</tr>
<tr>
<td>order 1</td>
<td>59 M/sec</td>
</tr>
<tr>
<td>order 1 (single)</td>
<td>78 M/sec</td>
</tr>
<tr>
<td>order 1 (SSE2)</td>
<td>94 M/sec</td>
</tr>
<tr>
<td>order 1 (CUDA)</td>
<td>824 M/sec</td>
</tr>
</tbody>
</table>
Numerical Heating: OSIRIS results

Results, using OSIRIS, Very Promising

<table>
<thead>
<tr>
<th>ID</th>
<th>Case</th>
<th>Times, s</th>
<th>Energy drift, %</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Linear</td>
<td>396</td>
<td>$8.2 \times 10^{-2}$</td>
</tr>
<tr>
<td>B</td>
<td>Quadratic</td>
<td>561</td>
<td>$1.9 \times 10^{-3}$</td>
</tr>
<tr>
<td>C</td>
<td>Cubic</td>
<td>852</td>
<td>$6.8 \times 10^{-4}$</td>
</tr>
<tr>
<td>D</td>
<td>Linear, 256 part/cell</td>
<td>788</td>
<td>$4.2 \times 10^{-2}$</td>
</tr>
<tr>
<td>E</td>
<td>Linear, no smoothing</td>
<td>394</td>
<td>1.05</td>
</tr>
</tbody>
</table>
Plasma Simulation Code (PSC)

- 1D, 2D, 3D configuration space
- relativistic, electromagnetic
- boost frame, moving window, PMLs, collisions, ionization...
- modular architecture: switching from legacy Fortran particle pusher to GPU pusher can be done on the command line.
- support for modern hardware (GPUs, Intel MIC)

Color indicates the MPI process responsible for the corresponding part of the domain.
PSC on GPUs

Multi-level decomposition of the problem, expose parallelism

- At the top-level, decompose spatial domain into \textit{patches}. Each MPI process gets assigned one or more patches. Patches communicate via ghost cells / particle exchange.

- (Hybrid level can be introduced here: Each MPI process will distribute patches onto a set of cores or GPUs using OpenMP / threads)

- GPU: Each patch gets further divided into \textit{blocks} (a.k.a. supercells) of multiple cells. These blocks are handled (in parallel) by threadblocks.

- Particles in a block are processed in parallel by threads in the threadblock (GPU) / by SIMD instructions (CPU/MIC).
PSC on GPUs

Particle-in-cell algorithm
for timestep \( n = 0,1,2,... \):

for each particle \( m \):
  advance momentum: \( \vec{p}^n_m \rightarrow \vec{p}^{n+1}_m \)
    (using interpolated \( \vec{E}^{n+1/2}, \vec{B}^{n+1/2} \))
  advance position: \( \vec{x}^{n+1/2}_m \rightarrow \vec{x}^{n+3/2}_m \)
  deposit current density contribution \( \vec{j}^{n+1}_m \) onto mesh.

advance fields: \( \vec{E}^{n+1/2}, \vec{B}^{n+1/2} \rightarrow \vec{E}^{n+3/2}, \vec{B}^{n+3/2} \) using \( \vec{j}^{n+1} \).
### PSC on GPUs – TitanDev/BlueWaters Performance

16-core AMD 6274 CPU, Nvidia Tesla M2090 / Tesla K20X

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Performance [particles/sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D push &amp; V-B current, CPU (AMD)</td>
<td>$130 \times 10^6$</td>
</tr>
<tr>
<td>2D push &amp; V-B current, GPU (M2090)</td>
<td>$565 \times 10^6$</td>
</tr>
<tr>
<td>2D push &amp; V-B current, GPU (K20X)</td>
<td>$710 \times 10^6$</td>
</tr>
</tbody>
</table>

For best performance, need to use GPU and CPU simultaneously. Patch-based load balancing enables us to do that: On each node, we have 1 MPI-process that has $\approx 30$ patches that are processed on the GPU, and 15 MPI-processes that have 1 patch each that are processed on the remaining CPU cores.
PSC on GPUs

Parallel Scalability, GPUs and CPUs

![Graph showing parallel scalability with GPUs and CPUs](image)

- **Top:** Number of Cray XK nodes (16-core AMD CPU + 1 Tesla 20X GPU)
- **Bottom:** Number of Cray XK/XE AMD CPU cores

- **GPU accelerated AMD CPUs**

- **X-axis:** Number of Cray XK nodes (20, 320, 80, 1280, 320, 5120, 1280, 20480, 5120, 81920)
- **Y-axis:** Million particles per second / node

Legend:
- **Blue line:** GPU accelerated
- **Green line:** AMD CPUs