Porting the Particle in Cell Code PSC to the New Intel Xeon Phi Architecture

Jeffrey Picard[1], Kai Germaschewski[1], H. Karimabadi[2]
[1] University of New Hampshire [2] Sciber Quest, Inc

University of New Hampshire

Introduction

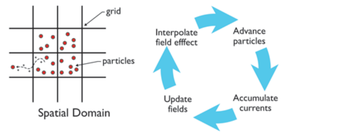
The purpose of this project was to port the particle in cell code, PSC, to Intel's new Xeon Phi processor architecture and to measure the performance benefits the architecture provides. The Xeon Phi is built especially for high performance and parallel programming in mind and has 50+ cores per card. In addition the Xeon Phi has 512 bit SIMD vectors for vectorizing computation. This is a significant increase over the 128 bit SIMD vectors in Intel's Xeon architecture. Getting the computational kernel of PSC, which uses hardware dependent instructions, to use the 512 bit SIMD vectors provided a substantial increase in performance for this code and would likely have analogous effects on similar codes.

Outline

Introduction
Kinetic Plasma Simulations
Plasma Simulation Code
Intel MIC Basics
H3D on MIC
PSC on GPUs
PSC on MIC
Summary / Outlook

Particle-in-Cell (PIC) Plasma Codes

- Fully kinetic (electrons and ions are treated as particles)
- Hybrid (ions are treated as particles, electrons as fluid)
- Multi-Fluid, One-Fluid (ions and electrons are treated as fluids)



Source: LANL

Simulation Codes

Plasma Simulation Code (PSC)

- 3D and reduced spatial dimensions (1D, 2D)
- 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.

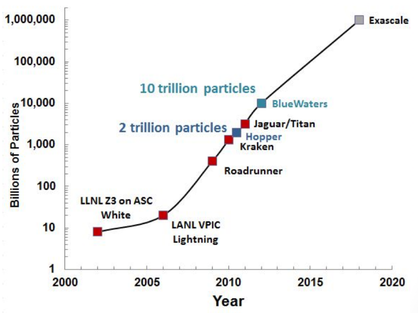
H3D Hybrid Code

- 3D spatial dimensions
- ions represented as particles
- electrons represented as fluid

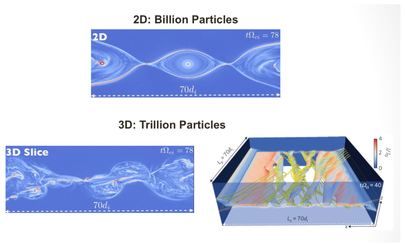
What does it take to model an experiment?

Parameters/size	physics	Cost, CPU-hrs
2D: M/m=100; Ω_{pe}/Ω_{ce}=2 1024 x 2048 cells	Basic physics of the diffusion region, role of the external drive	1.5 x 10^4
2D: M/m=1836; Ω_{pe}/Ω_{ce}=2 5300 x 10000 cells	Realistic influence of binary collisions, realistic kinetic physics of trapping, etc	6.5 x 10^6
3D: M/m=300; Ω_{pe}/Ω_{ce}=2 2000 x 4000x4000 cells	Influence of current-aligned instabilities	9 x 10^8
3D: M/m=1836; Ω_{pe}/Ω_{ce}=80 (2x10^5) x (4x10^5) x (4x10^5) cells	Realistic physical parameters	9 x 10^{18}

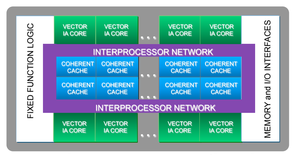
Progress in Particle Simulations



First glimpse of 3D effects



Intel MIC Architecture: An Intel Co-Processor Architecture



Many cores, and many, many more threads

Standard IA programming and memory model

Standard networking protocols

Source: Kirk, Skaugen, ISC 2010 keynote

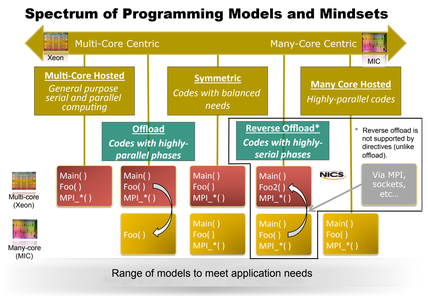
Intel® Knights Corner Technical Specifications

Intel® Knights Corner (Intel® KNC) is the first commercial product employing the Intel® Many Integrated Core (Intel® MIC) architecture.

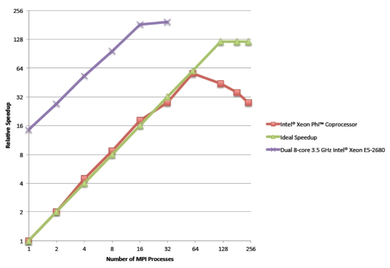
Current KNC deployments utilize pre-production hardware; thus, current performance does not necessarily indicate that of the commercial product.

Core Count	> 50 cores
IO Bus	PCIe
Operating System on Card	Linux-based
Networking Capability	IP-Addressable

Spectrum of Programming Models and Mindsets



H3D Scaling on Beacon



PSC on Accelerators

Multi-level decomposition of the problem, expose parallelism

- At the top-level, decompose spatial domain into *patches*. Each MPI process gets assigned one or more patches. Patches communicate via ghost cells / particle exchange.
- (Hybrid level could 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 *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 Accelerators

Particle-in-cell algorithm

for timestep n = 0,1,2,...:

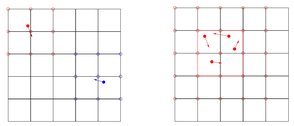
for each particle m:

advance momentum: $\vec{p}_m^n \to \vec{p}_m^{n+1}$
(using interpolated $\vec{E}^{n+1/2}, \vec{B}^{n+1/2}$)

advance position: $\vec{x}_m^{n+1/2} \to \vec{x}_m^{n+3/2}$

deposit current density contribution $\vec{j}_m^{n+1}$ onto mesh.

advance fields: $\vec{E}^{n+1/2}, \vec{B}^{n+1/2} \to \vec{E}^{n+3/2}, \vec{B}^{n+3/2}$ using $\vec{j}^{n+1}$.



PSC on GPUs – TitanDev/BlueWaters Performance

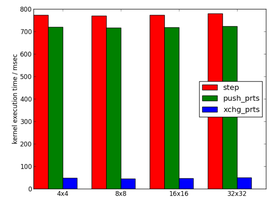
16-core AMD 6274 CPU, Nvidia Tesla M2090 / Tesla K20X

Kernel	Performance [particles/sec]
2D push & V-B current, CPU (AMD)	130 × 10^6
2D push & V-B current, GPU (M2090)	565 × 10^6
2D push & V-B current, GPU (K20X)	710 × 10^6

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 ≈ 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 MIC

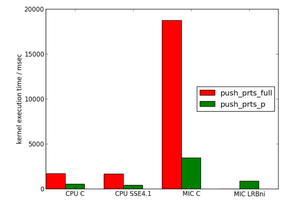
Varying the number of patches



512 × 512 mesh, divided into $n \times n$ patches, run on 16 Intel CPU cores.

PSC on MIC

Single core performance



LRBni intrinsics: speed up of 3.9×, that's 2.1× slower than same kernel on a CPU core.

PSC on MIC

Vectorizing PSC for MIC

- Start from SSE4.1 version that pushes 4 particles at a time.
- Implement SIMD emulation layer, initially for vectors of 4 floats.
- Change vector length to 16 in emulation.
- Replace emulation by _mm512_* intrinsics.
- Look at assembler code.

```
for (int k = 0; k < ppsc->nr_kinds; k++) {
  dq_kind[k] = .5f * ppsc->coeff.eta * dt
    * ppsc->kinds[k].q / ppsc->kinds[k].m;
}
```

PSC on MIC

Vectorizing PSC for MIC

kernel version	run time
scalar version:	609 ms
512-emu version:	2244 ms
512-mic version:	373 ms
forceinline:	169 ms
data layout:	139 ms
field access:	123 ms

Achieved a speed up of 5× over scalar code.

PSC on MIC

Data layout issues

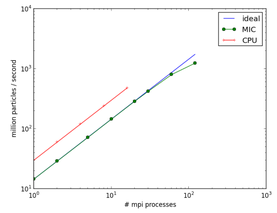
- Data layout used previously was array-of-struct.
- To be able to use SIMD instructions, need to re-arrange. For vectors of 4 floats, can be done easily with transpose.
- On MIC, can use gather/scatter intrinsics
- But this is what happens:

```
..L18
        vgatherdps 4(%rdx,%zmm0,4), %zmm8{%k4}          #418.10
        jkzd      ..L17, %k4    # Prob 50%              #418.10
        vgatherdps 4(%rdx,%zmm0,4), %zmm8{%k4}          #418.10
        jknzd     ..L18, %k4    # Prob 50%
                  %#418.10
..L17
```

- A simple load/store benchmark showed improvement of 66.634 ms -> 31.934 ms when avoiding scatter/gather by changing data layout to array-of-struct-of-simd-vectors. Overall about 20% speed-up.

PSC on MIC

Strong Scaling



push_prts_p

Summary / Outlook

- Initial results of running PSC on Intel MIC look promising. Performance for particle momentum update is comparable to current GPUs (> 1 billion particles / second).
- Current deposition still needs to be ported / optimized.
- Load balancing of particle-in-cell using subdivision into many patches and using a space-filling curve to distribute the load works well and provides flexibility in adapting the code to heterogeneous architectures.
- Particle-in-cell can be accelerated by GPUs and Intel MIC significantly, but involves a lot of manual work for proper tuning.
  - Intel MIC is less effort to port (vectorize) than GPUs.
  - Debugging is easier for MIC.
  - Cannot rely on compiler, even with intrinsics still need to look at the generated assembly.
  - Sorting is another open issue.