Array-of-Struct particles for iPic3D on MIC

Alec Johnson and Giovanni Lapenta

Centre for mathematical Plasma Astrophysics
Mathematics Department
KU Leuven, Belgium

EASC2014
Stockholm, Sweden
April 3, 2014

Abstract: We are porting iPic3D to the MIC for particle processing. iPic3D advances both the electromagnetic field and the particles implicitly, requiring typically 100-200 iterations of the field advance and 3-5 iterations of the particle advance for each cycle. We use particle subcycling to limit particle motion to one cell per cycle, which improves accuracy and simplifies sorting. To accelerate sorting, we represent particles in AoS format in double precision so that particle data exactly fits the cache line width. To vectorize particle calculations, we process particles in blocks: a fast 8x8 matrix transpose implemented in intrinsics converts each 8-particle block between SoA and AoS representation.


porting iPic3D to MIC

Goal: efficiently converged multiscale simulation of plasma

Tool: iPic3D, an implicit particle-in-cell code

Task: port to Xeon + Phi (MIC):

- improve MPI
- use OMP threads
- vectorize

Key issue: data layout of particles

- Ordering:
  - SoA for vectorization (push and sum)
  - AoS for localization (sorting)
- Granularity of particles:
  - grouped by cell: vectorization efficiency
  - grouped by thread subdomain: cache efficiency

Agenda: justify choices

The purpose of this presentation is to justify four algorithm choices:

Two fundamental determinations:

1. **Subcycle particles**:
   - for each particle, break time step into substeps
   - move the particle at most one cell per substep
   - motivation: *accurate simulation of fast-moving particles*
   - benefit: simpler sorting
2. *Use **Array-of-Structs (AoS)** for particles.*
   - motivation: fast sorting
   - can still vectorize via fast transpose/intrinsics

Two secondary determinations:

1. *Use **double precision** for particles.*
   - Vlasov solver via resampling.
   - no mixed precision.
   - particle exactly fits cache line.
2. *Use AoS field to push particles.*
   - motivation: better localization of field data access
   - justification: one transpose per cycle is justified by numerous particle iterations and amortized by many iterations of SoA field solver.

Outline

Equations of iPic3D

iPic3D simulates charged particles interacting with the electromagnetic field. It solves the following equations:

Fields:

$$\partial_t \mathbf{B}(\mathbf{x}) + \nabla \times \bar{\mathbf{E}}(\mathbf{x}) = 0$$

$$\partial_t \mathbf{E}(\mathbf{x}) - c^2 \nabla \times \bar{\mathbf{B}}(\mathbf{x}) = -\bar{\mathbf{J}}(\mathbf{x})/\epsilon_0,$$

Particles:

$$\partial_t \mathbf{v}_\mathrm{p} = \frac{q_\mathrm{p}}{m_\mathrm{p}} \left(\bar{\mathbf{E}}'(\bar{\mathbf{x}}'_\mathrm{p}) + \bar{\mathbf{v}}_\mathrm{p} \times \bar{\mathbf{B}}'(\bar{\mathbf{x}}'_\mathrm{p}) \right),$$

$$\partial_t \mathbf{x}_\mathrm{p} = \bar{\mathbf{v}}_\mathrm{p},$$

Moments (10):

(1) $\sigma(\mathbf{x}) := \sum_\mathrm{p} S(\mathbf{x} - \mathbf{x}_\mathrm{p}) q_\mathrm{p}$

(3) $\mathbf{J}(\mathbf{x}) := \sum_\mathrm{p} S(\mathbf{x} - \mathbf{x}_\mathrm{p}) q_\mathrm{p} \mathbf{v}_\mathrm{p}$

(6) $\mathcal{P}(\mathbf{x}) := \sum_\mathrm{p} S(\mathbf{x} - \mathbf{x}_\mathrm{p}) q_\mathrm{p} \mathbf{v}_\mathrm{p} \mathbf{v}_\mathrm{p}$

The Implicit Moment Method uses these 10 moments (with $\mathbf{E}$ and $\mathbf{B}$) to estimate $\bar{\mathbf{J}}$.

Discretization:

- $\partial_t X := \frac{X^{n+1} - X^n}{\Delta t}$.
- $\overline{X} = \frac{1}{2} X^{n+1} + \frac{1}{2} X^n$.

Current Evolution

$$\partial_t \mathbf{J}_\mathrm{s} + \nabla \cdot \mathcal{P}_\mathrm{s} = \frac{q_\mathrm{s}}{m_\mathrm{s}} \left(\sigma'_\mathrm{s} \bar{\mathbf{E}} + \bar{\mathbf{J}}_\mathrm{s} \times \mathbf{B}' \right)$$

Average current responds linearly to electric field:

$$\bar{\mathbf{J}} = \widehat{\mathbf{J}} + \mathbb{A} \cdot \bar{\mathbf{E}},$$

where:

$$\left[\begin{array}{l} \widehat{\mathbf{J}} := \sum_\mathrm{s} \widehat{\mathbf{J}}_\mathrm{s}, \\ \widehat{\mathbf{J}}_\mathrm{s} := \Pi_\mathrm{s} \cdot \left(\mathbf{J}^n_\mathrm{s} - \frac{\Delta t}{2} \nabla \cdot \mathcal{P}_\mathrm{s} \right), \\ \mathbb{A} := \sum_\mathrm{s} \beta_\mathrm{s} \sigma'_\mathrm{s} \Pi_\mathrm{s}, \\ \Pi_\mathrm{s} := \dfrac{\mathbb{I} - \widetilde{\boldsymbol{B}}_\mathrm{s} \times \mathbb{I} + \widetilde{\boldsymbol{B}}_\mathrm{s} \widetilde{\boldsymbol{B}}_\mathrm{s}}{1 + |\widetilde{\boldsymbol{B}}_\mathrm{s}|^2}, \\ \widetilde{\boldsymbol{B}}_\mathrm{s} := \beta_\mathrm{s} \mathbf{B}', \\ \beta_\mathrm{s} := \frac{q_\mathrm{s} \Delta t}{2 m_\mathrm{s}}. \end{array} \right]$$

Implicit Particle Advance

$$\bar{\mathbf{v}}_\mathrm{p} = \left(\frac{\mathbb{I} - \widetilde{\boldsymbol{B}}_\mathrm{p} \times \mathbb{I} + \widetilde{\boldsymbol{B}}_\mathrm{p} \widetilde{\boldsymbol{B}}_\mathrm{p}}{1 + |\widetilde{\boldsymbol{B}}_\mathrm{p}|^2} \right) \cdot \left(\mathbf{v}^n_\mathrm{p} + \beta_\mathrm{s} \bar{\mathbf{E}}'(\bar{\mathbf{x}}_\mathrm{p}) \right), \text{ where}$$

$$\widetilde{\boldsymbol{B}}_\mathrm{p} := \frac{q_\mathrm{p} \Delta t}{2 m_\mathrm{p}} \bar{\mathbf{B}}'(\bar{\mathbf{x}}_\mathrm{p}), \text{ and}$$

$$\bar{\mathbf{x}}_\mathrm{p} = \mathbf{x}^0_\mathrm{p} + \Delta t \bar{\mathbf{v}}_\mathrm{p}.$$

iPic3D cycle

iPic3D cycles through three tasks:

1. `fields.advance(moments)`
2. `particles[s].move(fields)`
3. `moments[s].sum(particles[s])`

Moving particles consists of pushing and sorting, e.g.:

```
foreach subcycle c:
  foreach particle:
    particle.push(field(cell(particle)))
    particle.sort()
  particles.communicate()
```

Outline

Mapping data to architecture

Balance two goals:

1. flexibility where architectures/algorithms differ
2. best particulars where architectures/algorithms agree

Architecture key attributes:

1. Width of **cache line**: **8 doubles** = 512 bits (fairly universal)
2. Width of **vector unit**:
   - **8 doubles** = 512 bits for MIC
   - 4 doubles = 256 bits for Xeon with AVX
   - 2 doubles = 128 bits for SSE2

Data of **algorithm**:

1. fields: 6 doubles (two vectors) per mesh cell:
   1. B_x magnetic field
   2. B_y magnetic field
   3. B_z magnetic field
   4. ψ (**B** correction potential)
   5. E_x electric field
   6. E_y electric field
   7. E_z electric field
   8. ϕ (**E** correction potential)
2. 100s of particles per mesh cell; 8 doubles (2 vectors + 2 scalars) per particle:
   1. u velocity
   2. v velocity
   3. w velocity
   4. q charge (or particle ID)
   5. x position
   6. y position
   7. z position
   8. t subcycle time

(1) Why subcycle?

Traditionally the implicit moment method moves all particles with the same time step.

We are implementing subcycling:

- For each particle, the global time step is partitioned into substeps.
- Substeps stop particles at cell boundaries.

Benefits of subcycling:

1. Simplifies sorting:
   - SoA vectorization requires sorting particles by mesh cell.
   - Subcycling guarantees that particles move only one mesh cell per subcycle.
   - Without subcycling, particles can move arbitrarily far between sorts.
   - Without subcycling, particles must be sorted with every iteration of the implicit mover.
   - Without subcycling, sorted particle data must include average position data and no longer fits in a single cache line.
2. Subcycling is needed to resolve fast particles accurately.
   - Maxwell's equations need time-averaged current.
   - Subcycling is needed to get correct time-averaged current of fast particles.

(2) AoS particle vectorization

Usually SoA is preferred for vectorization.

But AoS particles can still be vectorized in one of two ways:

Fast matrix transpose

- **8-component particles** (subcycled case):
  - Represent as AoS
  - Process in 8-particle blocks
  - Convert blocks to/from SoA using fast 8x8 matrix blocked transpose (28-36 8-wide vector instructions)
- 12-component particles (non-subcycled case):
  - Consider padding extra components to 8 (faster sort); otherwise:
  - first 8 components handled like 8-component particles
  - last 4 components handled like 4-component particles using fast 4x8↔8x4 transpose (16 8-wide vector instructions).

Physical vectors (intrinsics-heavy):

- **MIC**:
  - process 2 particles at a time
  - concatenate velocity vectors: [u1, v1, w2, q1, u2, v2, w2, q2]
  - concatenate position vectors: [x1, y1, z1, t1, x2, y2, z2, t2]
  - Use physical vector operations (use swizzle for cross-product)
- Xeon
  - process 1 particle at a time (or 2 at a time for single precision)

Pusher times on Xeon Phi *[feasibility studies]*

Pusher times in iPic3D:

```
time        pusher
=========   ========
0.102 sec   SoA       (but also need to sort each iteration)
0.202 sec   AoS_intr  (no sort required, but helps cache)
0.259 sec   AoS       (no sort required, but helps cache)
```

Pusher times for a single iteration:

```
time            pusher
============    ========
.07 Mcycles     SoA
.13 Mcycles     AoS_tran (8-pcl blocks via fast 8x8 transpose)
.21 Mcycles     AoS_intr (2-pcl blocks with intrinsics mover)
```

Pusher times for 4 iterations stopping at cell boundary:

```
time              pusher
============      ========
.36 Mcycles       SoA
.40 Mcycles       AoS_tran (8-pcl blocks via fast 8x8 transpose)
[unimplemented]   AoS_intr (no need to sort with each subcycle)
```

Sorting efficiently

Cache-line-sized particles facilitate sorting:

- can transfer particles directly to memory destination with no-read writes
- no cache contention
- vector unit divides cache line size, so fully utilized

Sort particles by:

1. process subdomain (for MPI),
2. thread subdomain, and
3. mesh cell (for vectorization)

To hide communication latency, overlap process-level communication with thread-level sorting.

- General sort:

```
send exiting particles
sort particles in process
wait on incoming particles
sort incoming particles
```

- Subcycle sort (moving ≤ 1 cell per subcycle):

```
move particles in boundary cells
send particles in ghost cells
move particles in interior cells
move incoming particles
```

Using AoS fields and moments in particle solver

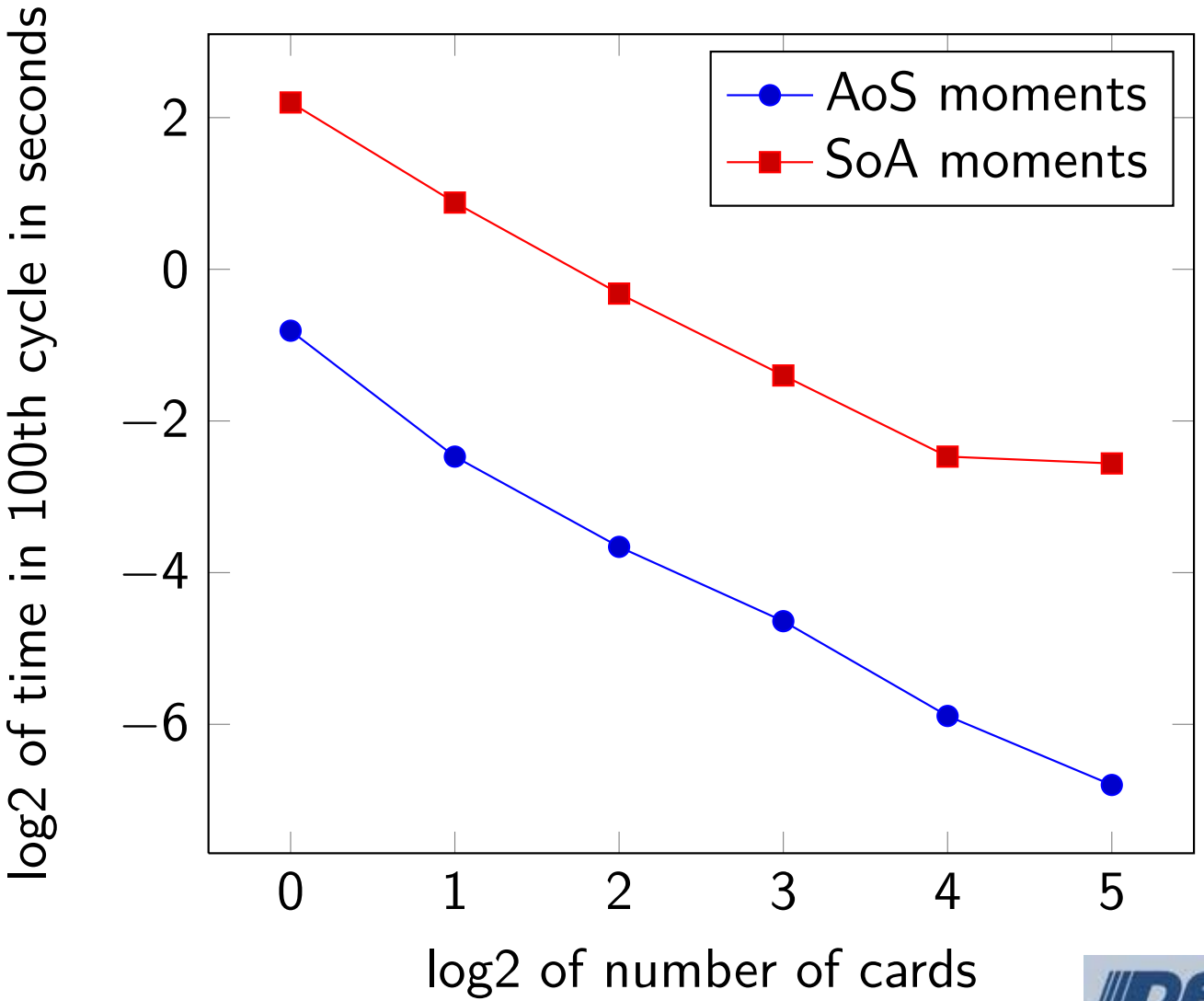
In the field solver we represent fields and moments in SoA format.
This allows better vectorization of the implicit solver.

In the particle solver, we represent fields and moments with AoS format:

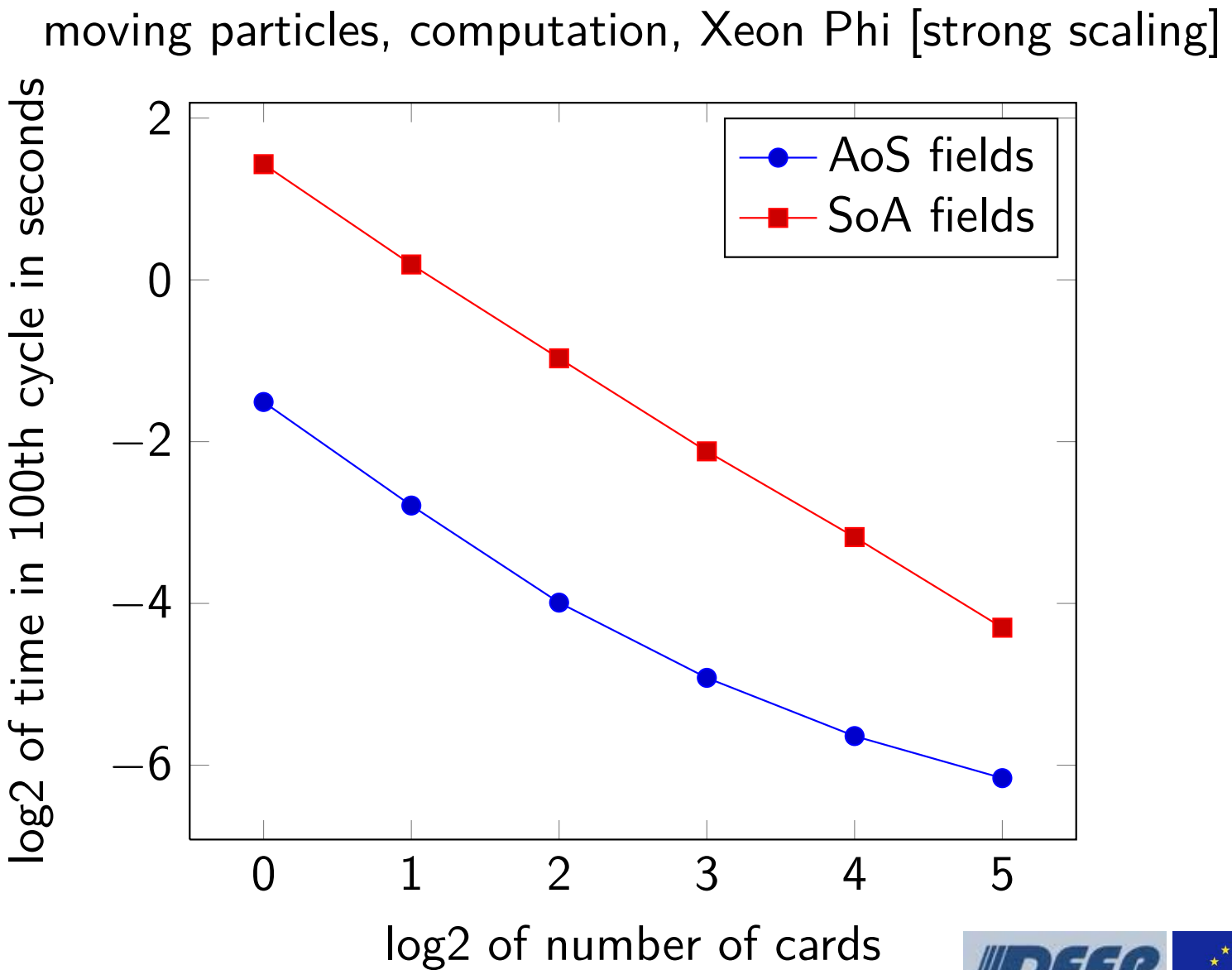
- AoS gives better localization of random access.
- SoA fields and moments offer no benefit to vectorization of particle processing.
- The transpose is done only once per cycle.

Summing moments as AoS

summing moments, computation, Xeon Phi [strong scaling]



Using AoS fields and moments in particle solver



Summary

Trade-offs in SoA vs. AoS particles:

- SoA particles better for vectorization.
- SoA particles require sorting with each iteration or subcycle.
- AoS particles better for sorting.
- AoS allows infrequent sorting.
- fast 8x8 transpose converts between AoS and SoA.

Conclusions:

- Use AoS for basic particle representation.
- Convert to SoA in blocks when beneficial for vectorization.