Porting a Legacy Global Lagrangian PIC Code on Many-Core and GPU-Accelerated Architectures

PASC Conference, Basel

July 3rd, 2018

Noé Ohana ¹, Andreas Jocksch ², Emmanuel Lanti ¹, Aaron Scheinberg ¹, Stephan Brunner ¹, Claudio Gheller ², Laurent Villard ¹

¹ SPC, Lausanne
² CSCS, Lugano

Acknowledgments: Alberto Bottino, Ben McMillan, Alexey Mishchenko, Thomas Hayward

This work has been carried out within the framework of the EUROfusion Consortium and has received funding from the Euratom research and training programme 2014-2018 under grant agreement No 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission.
1. Introduction

2. Refactoring

3. Numerical performance
   - Single node
   - Multinode

4. Conclusion
Problem:

Development timescale of legacy code ORB5 [Tran, 1999, Jolliet, 2009] exceeds HPC architecture evolution timescale
Motivation

1. Introduction
2. Refactoring
3. Results
4. Conclusion

Problem:
- Development timescale of legacy code ORB5 [Tran, 1999, Jolliet, 2009] exceeds HPC architecture evolution timescale

Adopted strategy:
- Disentangle numerical kernels and physical modules $\Rightarrow$ modularization
- More and more cores per compute node $\Rightarrow$ trade multitasking for multithreading
- Keep portability $\Rightarrow$ directive-based approach (OpenMP and OpenACC)
- Develop testbed code with fundamental kernels (PASC project)
Recipe

1. Introduction
2. Refactoring
3. Results
4. Conclusion

Timeline:

- 2014: usual ORB5 development (new physics)
- 2015: "big merge"
- 2016: ORB5 refactoring
- 2017: PASC project

Tools:

- Profiler
- Continuous integration tool: Jenkins
Global gyrokinetic PIC code:

- Equations derived from variational formulation with consistent ordering [Tronko, 2017]
- Ad-hoc or MHD equilibrium
- Electromagnetic perturbations
- Different field solver options (long wavelength approximation, Padé approximation, or all orders)
- Different gyro-averaging options (\(\langle \nabla \phi \rangle\) or \(\nabla \langle \phi \rangle\))
- Multiple gyrokinetic species
- Fixed or adaptive number of Larmor points
- Drift-kinetic, adiabatic or hybrid electrons
- Inter- and intra-species collisions
- Heat sources
- Strong flows
- Advanced diagnostics
Numerical features:

- Variational formulation of field equations with finite element B-splines up to third order
- Control variate schemes
- Electromagnetic cancellation problem in Ampère’s law solved with enhanced control variate or pullback scheme [Mishchenko, 2014]
- Runge-Kutta of fourth order time integrator
- Magnetic coordinates, straight-field-line
- Field-aligned Fourier filter
- Noise control (Krook operator, coarse graining, quadtree)
- Original parallelization: 2 levels of MPI (domain decomposition and cloning)
- New parallelization: hybrid MPI+OpenMP or MPI+OpenACC
MPI parallelization scheme

1. Introduction
2. Refactoring
3. Results
4. Conclusion

- 1D domain decomposition plus domain cloning:

  Reduce/broadcast operations between clones
  All-to-all (parallel data transpose and particle move) and neighbor (guard cells) communications between subdomains
Key modifications prior to multithreading

1. Introduction
2. Refactoring
3. Results
4. Conclusion

- Data structures
  - Structure of arrays instead of arrays of structure
  - Pack variables for host-to-device memory transfers
Key modifications prior to multithreading

1. Introduction
2. Refactoring
3. Results
4. Conclusion

- **Data structures**
  - Structure of arrays instead of arrays of structure
  - Pack variables for host-to-device memory transfers

- **Modularization**
  - Stages of a time step:
Key modifications prior to multithreading

- Data structures
  - Structure of arrays instead of arrays of structure
  - Pack variables for host-to-device memory transfers

- Modularization
  - Stages of a time step:
Single node simulations with $10^6$ ions (4 Larmor points each), $10^6$ electrons, and $128 \times 32 \times 4$ cubic splines, on Broadwell CPU ($2 \times 18$ cores, 2.1GHz).

Full MPI

36 clones, 1 thread

<table>
<thead>
<tr>
<th>Build</th>
<th>Deposit</th>
<th>Field solve</th>
<th>Get field</th>
<th>Gyro-average</th>
<th>Push</th>
<th>Diagnostics</th>
<th>Other modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.14</td>
<td>0.053</td>
<td>0.27</td>
<td>0.14</td>
<td>0.72</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Up to 20% speed-up by filling sockets with OpenMP threads.
Single node simulations with $10^6$ ions (4 Larmor points each), $10^6$ electrons, and $128 \times 32 \times 4$ cubic splines, on Broadwell CPU ($2 \times 18$ cores, 2.1GHz)

- Full MPI
  - 36 clones, 1 thread: Wall clock time per step (s)
  - 18 clones, 2 threads: Wall clock time per step (s)
  - 12 clones, 3 threads: Wall clock time per step (s)
  - 9 clones, 4 threads: Wall clock time per step (s)
  - 6 clones, 6 threads: Wall clock time per step (s)
  - 4 clones, 9 threads: Wall clock time per step (s)
  - 3 clones, 12 threads: Wall clock time per step (s)
  - 2 clones, 18 threads: Wall clock time per step (s)
  - 1 clone, 36 threads: Wall clock time per step (s)

- Full OpenMP
  - 1 clone, 36 threads: Wall clock time per step (s)

Up to 20% speed-up by filling sockets with OpenMP threads.
Single node simulations with $10^6$ ions (4 Larmor points each), $10^6$ electrons, and $128 \times 32 \times 4$ cubic splines, on Broadwell CPU (2 × 18 cores, 2.1GHz).

<table>
<thead>
<tr>
<th>Clones</th>
<th>Threads</th>
<th>Build</th>
<th>Larmor</th>
<th>Deposit</th>
<th>Field</th>
<th>Get</th>
<th>Gyro-average</th>
<th>Push</th>
<th>Diagnostics</th>
<th>Other</th>
<th>Wall clock time per step (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>18</td>
<td>0.14</td>
<td>0.055</td>
<td>0.053</td>
<td>0.27</td>
<td></td>
<td></td>
<td>0.14</td>
<td></td>
<td>0.72</td>
<td>0.18</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>0.14</td>
<td>0.054</td>
<td>0.054</td>
<td>0.27</td>
<td></td>
<td></td>
<td>0.14</td>
<td></td>
<td>0.7</td>
<td>0.13</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0.14</td>
<td>0.054</td>
<td>0.054</td>
<td>0.26</td>
<td>0.12</td>
<td></td>
<td>0.14</td>
<td></td>
<td>0.69</td>
<td>0.15</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>0.14</td>
<td>0.054</td>
<td>0.054</td>
<td>0.26</td>
<td></td>
<td></td>
<td>0.14</td>
<td></td>
<td>0.65</td>
<td>0.12</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>0.12</td>
<td>0.055</td>
<td>0.055</td>
<td>0.25</td>
<td>0.12</td>
<td></td>
<td>0.12</td>
<td></td>
<td>0.63</td>
<td>0.26</td>
</tr>
<tr>
<td>36</td>
<td>1</td>
<td>0.13</td>
<td>0.054</td>
<td>0.054</td>
<td>0.26</td>
<td>0.12</td>
<td></td>
<td>0.12</td>
<td></td>
<td>0.64</td>
<td>0.26</td>
</tr>
</tbody>
</table>

Up to 20% speed-up by filling sockets with OpenMP threads.
Single node simulations with $10^6$ ions (4 Larmor points each), $10^6$ electrons, and $128 \times 32 \times 4$ cubic splines, on Broadwell CPU (2 × 18 cores, 2.1GHz)
**Single node simulations with** $10^6$ ions (4 Larmor points each), $10^6$ electrons, and $128 \times 32 \times 4$ cubic splines, on Broadwell CPU (2 $\times$ 18 cores, 2.1GHz)

<table>
<thead>
<tr>
<th>Clone Count</th>
<th>Threads</th>
<th>Wall Clock Time per Step (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 clones, 36 threads</td>
<td>1 thread</td>
<td>0.14</td>
</tr>
<tr>
<td>4 clones, 9 threads</td>
<td>2 threads</td>
<td>0.12</td>
</tr>
<tr>
<td>6 clones, 6 threads</td>
<td>3 threads</td>
<td>0.14</td>
</tr>
<tr>
<td>12 clones, 3 threads</td>
<td>6 threads</td>
<td>0.14</td>
</tr>
<tr>
<td>18 clones, 2 threads</td>
<td>9 threads</td>
<td>0.14</td>
</tr>
<tr>
<td>36 clones, 1 thread</td>
<td>18 threads</td>
<td>0.14</td>
</tr>
</tbody>
</table>

**Up to 20% speed-up by filling sockets with OpenMP threads**
Single node simulations with $10^6$ ions (4 Larmor points each), $10^6$ electrons, and $128 \times 32 \times 4$ cubic splines, on Haswell CPU (12 cores, 2.6GHz)
Single node simulations with $10^6$ ions (4 Larmor points each), $10^6$ electrons, and $128 \times 32 \times 4$ cubic splines, on Haswell CPU (12 cores, 2.6GHz)

Wall clock time per step (s)

- Full MPI, Intel compiler (12 clones)
  - Build: 0.28
  - Larmor: 0.15
  - Deposit: 0.1
  - Field solve: 0.59
  - Get field: 0.24
  - 1.4

- Full OpenMP, Intel compiler (24 threads)
  - Build: 0.21
  - Larmor: 0.12
  - Deposit: 0.49
  - Field solve: 0.17
  - 1.2 (80%)

- Full MPI, PGI compiler (12 clones)
  - Build: 0.3
  - Larmor: 0.12
  - Deposit: 0.2
  - Field solve: 0.67
  - 0.27
  - 1.7 (117%)
Single node simulations with $10^6$ ions (4 Larmor points each), $10^6$ electrons, and $128 \times 32 \times 4$ cubic splines, on Haswell CPU (12 cores, 2.6GHz).

PGI compiler $\sim 20\%$ slower than Intel.
Single node simulations with $10^6$ ions (4 Larmor points each), $10^6$ electrons, and $128 \times 32 \times 4$ cubic splines, on Haswell CPU (12 cores, 2.6GHz)

![Graph showing wall clock time per step (s) for different compiler configurations.]

- PGI compiler $\sim 20\%$ slower than Intel
Single node simulations with $10^6$ ions (4 Larmor points each), $10^6$ electrons, and $128 \times 32 \times 4$ cubic splines, on Haswell CPU (12 cores, 2.6GHz).

- **PGI compiler** $\sim$20% slower than Intel
- **MPI+OpenACC version** $\sim$2.7 times faster than MPI only, and $\sim$2.2 times faster than MPI+OpenMP
Different physics (electro-static or -magnetic, adiabatic or kinetic electrons, adaptive number of Larmor points, ...) with similar particle and cell resolutions.
Different physics (electro-static or -magnetic, adiabatic or kinetic electrons, adaptive number of Larmor points, ...) with similar particle and cell resolutions.

- KNL (64 cores, 1.3GHz) performance similar to Haswell CPU.
**Architecture comparisons**

- Different physics (electro-static or -magnetic, adiabatic or kinetic electrons, adaptive number of Larmor points, ...) with similar particle and cell resolutions

<table>
<thead>
<tr>
<th>Build</th>
<th>Larmor</th>
<th>Deposit</th>
<th>Field solve</th>
<th>Get field</th>
<th>Gyro-average</th>
<th>Push</th>
<th>Diagnostics</th>
<th>Other modules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Best CPU Haswell</strong></td>
<td>0.21</td>
<td>0.12</td>
<td>0.49</td>
<td>0.17</td>
<td>1.2 (80%*)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Best CPU Broadwell</strong></td>
<td>0.23</td>
<td>0.12</td>
<td>0.6 (41%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Best KNL</strong></td>
<td>0.19</td>
<td></td>
<td>0.55</td>
<td>0.27</td>
<td>1.3 (89%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CPU Haswell + GPU</strong></td>
<td>0.12</td>
<td>0.21</td>
<td>0.54 (37%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ITG**

<table>
<thead>
<tr>
<th>Build</th>
<th>Larmor</th>
<th>Deposit</th>
<th>Field solve</th>
<th>Get field</th>
<th>Gyro-average</th>
<th>Push</th>
<th>Diagnostics</th>
<th>Other modules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Best CPU Haswell</strong></td>
<td>0.16</td>
<td>0.23</td>
<td>0.23</td>
<td>0.71 (75%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Best CPU Broadwell</strong></td>
<td>0.23</td>
<td>0.42</td>
<td>0.4 (42%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Best KNL</strong></td>
<td>0.16</td>
<td>0.24</td>
<td>0.26</td>
<td>0.24</td>
<td>1.1 (113%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CPU Haswell + GPU</strong></td>
<td>0.16</td>
<td>0.24</td>
<td>0.31 (33%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SAW**

<table>
<thead>
<tr>
<th>Build</th>
<th>Larmor</th>
<th>Deposit</th>
<th>Field solve</th>
<th>Get field</th>
<th>Gyro-average</th>
<th>Push</th>
<th>Diagnostics</th>
<th>Other modules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Best CPU Haswell</strong></td>
<td>0.23</td>
<td>0.78</td>
<td>0.49</td>
<td>0.59</td>
<td>2.2 (84%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Best CPU Broadwell</strong></td>
<td>0.13</td>
<td>0.34</td>
<td>0.18</td>
<td>0.26</td>
<td>1 (38%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Best KNL</strong></td>
<td>0.18</td>
<td>0.8</td>
<td>0.34</td>
<td>0.67</td>
<td>0.12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CPU Haswell + GPU</strong></td>
<td>0.16</td>
<td>0.73</td>
<td>0.24</td>
<td>1.3 (48%)</td>
<td>2.3 (86%)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**KNL (64 cores, 1.3GHz) performance similar to Haswell CPU**
Architecture comparisons

**1. Introduction**

**2. Refactoring**

**3. Results**

3.1 Single node

3.2 Multinode

**4. Conclusion**

Different physics (electro-static or magnetic, adiabatic or kinetic electrons, adaptive number of Larmor points, ...) with similar particle and cell resolutions

**ITG**

**SAW**

**TEM**

**CPU Haswell + GPU**

**Best KNL**

**Best CPU Broadwell**

**Best CPU Haswell**

**KNL (64 cores, 1.3GHz) performance similar to Haswell CPU**

**Weaker GPU performance for SAW due to control variate iterations on CPU**

**GPU performance similar to Broadwell CPU**

**Exact factors are case-dependent**
Application to production run

ITG production run on 256 nodes with $256 \cdot 10^6$ ions (4 Larmor points each) and $256 \times 512 \times 256$ cubic splines

<table>
<thead>
<tr>
<th>Build Larmor</th>
<th>Deposit</th>
<th>Field solve</th>
<th>Get field</th>
<th>Gyro-average</th>
<th>Push</th>
<th>Parallel move</th>
<th>Diagnostics</th>
<th>Other modules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Full MPI, BEFORE REFACTORIZING</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.4 (230%)</td>
</tr>
<tr>
<td>(12 clones per node)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Full MPI</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.9 (100%)</td>
</tr>
<tr>
<td>(12 clones per node)</td>
<td>0.24</td>
<td>0.59</td>
<td>0.31</td>
<td>0.17</td>
<td>0.19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MPI+OpenMP</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.8 (96%)</td>
</tr>
<tr>
<td>(24 threads per node)</td>
<td>0.17</td>
<td>0.49</td>
<td>0.26</td>
<td>0.37</td>
<td>0.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MPI+OpenACC</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.3 (68%)</td>
</tr>
<tr>
<td>(1 GPU per node)</td>
<td>0.61</td>
<td>0.22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Wall clock time per step (s)

0 0.5 1 1.5 2 2.5 3 3.5 4 4.5
Application to production run

1. Introduction
2. Refactoring
3. Results
   3.1 Single node
   3.2 Multinode
4. Conclusion

ITG production run on 256 nodes with $256 \cdot 10^6$ ions (4 Larmor points each) and $256 \times 512 \times 256$ cubic splines

<table>
<thead>
<tr>
<th>Build Larmor</th>
<th>Deposit</th>
<th>Field solve</th>
<th>Get field</th>
<th>Gyro-average</th>
<th>Push</th>
<th>Parallel move</th>
<th>Diagnostics</th>
<th>Other modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full MPI, BEFORE REFACTORIZING</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.4 (230%)</td>
</tr>
<tr>
<td>(12 clones per node)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.4</td>
</tr>
<tr>
<td>Full MPI</td>
<td>0.24</td>
<td>0.59</td>
<td>0.31</td>
<td>0.17</td>
<td>0.19</td>
<td>1.9 (100%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(12 clones per node)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI+OpenMP</td>
<td>0.17</td>
<td>0.49</td>
<td>0.26</td>
<td>0.37</td>
<td>0.2</td>
<td>1.8 (96%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(24 threads per node)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI+OpenACC</td>
<td>0.61</td>
<td>0.22</td>
<td>1.3 (68%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1 GPU per node)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Timings of communication-bound stages (field solver and parallel move) become non-negligible.

Multithreaded versions bring less speed-up than on single node because they do not improve inter-node communication.

Full MPI version already 2.3 times faster than before refactoring.

Noé Ohana - PASC18
Achievements:

- Single code making efficient use of different architectures
- Good compromise between efficiency, ergonomy, and "future-proofness" (flexibility to accommodate for new physics and new architectures)

Future work:

- Turn on sorting when particle-to-field operations are significant
- Reduce amount of communications in solver
T.M. Tran, K. Appert, M. Fivaz, G. Jost, J. Vaclavik and L. Villard
Global gyrokinetic simulation of ion-temperature-gradient-driven instabilities using particles
*Theory of Fusion Plasmas, Int. Workshop (Editrice Compositori, Bologna, Societa Italiana di Fisica)*, 45, 1999

S. Jolliet
Gyrokinetic particle-in-cell global simulations of ion-temperature-gradient and collisionless-trapped-electron-mode turbulence in tokamaks

A. Mishchenko, A. Könies, R. Kleiber, and M. Cole
Pullback transformation in gyrokinetic electromagnetic simulations
*Physics of Plasmas*, 21, 2014

N. Tronko, A. Bottino, C. Chandre and E. Sonnendrüecker
Hierarchy of second order gyrokinetic Hamiltonian models for particle-in-cell codes
*Plasma Physics and Controlled Fusion*, 59, 2017