Performance optimization of the Smoothed Particle Hydrodynamics code Gadget3 on 2\textsuperscript{nd} generation Intel Xeon Phi

Dr. Luigi Iapichino
luigi.iapichino@lrz.de

Leibniz Supercomputing Centre

Supercomputing 2017
Intel booth, Nerve Center
Dr. Luigi Iapichino  
Scientific Computing Expert  
Leibniz Supercomputing Centre

- Member of the Intel Parallel Computing Center (IPCC) @ LRZ/TUM
- Expert in computational astrophysics and simulations

Some of the results shown here are based on work performed with Dr. Fabio Baruffa (now at Intel)
Outline of the talk

- Overview of the code: P-Gadget3.
- Modernization of a code kernel.
- Back-porting to the full code.
- Optimization steps on Knights Landing (KNL).
- Performance results, takeaways from our KNL experience.

Simulation details: www.magneticum.org
Gadget intro

- Leading application for simulating the formation of the *cosmological* large-scale structure (galaxies and clusters) and of processes at sub-resolution scale (e.g. star formation, metal enrichment).

- Publicly available, cosmological TreePM N-body + SPH code.

- First developed in the late 90s as serial code, later evolved as an MPI and a hybrid code.

- Good scaling performance up to $O(100k)$ Xeon cores (SuperMUC@LRZ).

Simulation details: www.magneticum.org
The representative code kernel `subfind_density` was isolated and run as a stand-alone application, avoiding the overhead from the whole simulation.

Focus on node-level performance, through minimally invasive changes.

We use tools from the Intel® Parallel Studio XE (VTune Amplifier and Advisor).

Code optimization through:
- Better threading parallelism;
- Data optimization (AoS → SoA);
- Promoting more efficient vectorization.

Up to 19x faster execution on KNL.

Also available as: https://arxiv.org/abs/1612.06090
Modernizing the threading parallelism of the isolated kernel

- Severe shared-memory parallelization overhead
- At later iterations, the particle list is locked and unlocked constantly due to the recomputation
- Spinning time 41%

thread spinning
Improved performance

- **Lockless scheme**: lock contention removed through "todo" particle list and OpenMP dynamic scheduling.
- Time spent in spinning only 3%

**no spinning**
Improved speed-up of the isolated kernel on KNL

- Knights Landing Processor 7210 @ 1.3 GHz, 64 cores. KMP Affinity: scatter; Configuration Quadrant/Flat.

- On KNL @ 64 threads:
  - speed-up wrt original version: $5.7 \times$
  - parallel efficiency: 73%

- Crucial for target performance: OpenMP threads per MPI task on the full code? On 16 threads on KNL, speed-up improvement $2.3 \times$.

- Remark: the back-porting is based on a different physical workload, where the performance gain is lower (let's discuss this later...)

![Graph showing speed-up improvement with respect to thread count]
Guideline for the optimization on KNL

Optimization for KNL seen as a three-step process:

<table>
<thead>
<tr>
<th>Step</th>
<th>Effort</th>
<th>Expected performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compilation “out of the box”</td>
<td>1 hour</td>
<td>Lower than Haswell (~ 1.5x)</td>
</tr>
<tr>
<td>Optimization without coding (use of AVX512, explore configuration, MCDRAM, MPI/OpenMP)</td>
<td>1 week</td>
<td>Up to 2x over previous step</td>
</tr>
<tr>
<td>Optimization with coding</td>
<td>1-3 months (IPCC: 2 years)</td>
<td>Up to the level of Broadwell</td>
</tr>
</tbody>
</table>

Freely adapted from Leijun Hu, Inspur @ ISC 2017
## Back-porting: development steps on KNL

<table>
<thead>
<tr>
<th>Code version</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 0</td>
<td>v. 2018 Intel compiler and libraries, -xMIC-AVX512.</td>
<td>The code does not benefit from specific cluster or memory modes.</td>
</tr>
<tr>
<td>Optimized</td>
<td>Threading parallelism improved in subfind_density. Other minor improvements.</td>
<td>MPI/OpenMP configuration set by target, not by optimal performance.</td>
</tr>
</tbody>
</table>

Freely adapted from Leijun Hu, Inspur @ ISC 2017
# Performance results

One-node tests, performed on an Intel Xeon Phi (KNL) 7210 @ 1.30GHz with 64 cores. Configuration: Quad/flat with allocation on DDR. 4 MPI tasks, 16 OpenMP threads each.

<table>
<thead>
<tr>
<th>Code version</th>
<th>Time (total) [s]</th>
<th>Time (subfind_density) [s], % of total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>167.4</td>
<td>22.6 (13.5%)</td>
</tr>
<tr>
<td>Step 0</td>
<td>142.1</td>
<td>17.1 (12.1%)</td>
</tr>
<tr>
<td>Optimized</td>
<td>137.1</td>
<td>12.7 (9.3%) 1.8x (isolated kernel: it was 1.4x)</td>
</tr>
</tbody>
</table>

Freely adapted from Leijun Hu, Inspur @ ISC 2017
Understanding results and performance targets

- Based on our experience, 4-8 MPI tasks per KNL should be optimal.

- A complete back-porting should improve the OpenMP layer and move the best performance to the left.

- For comparison: currently the best performance on a Haswell node is for the pure-MPI case!

- Best performance KNL: 53.2s (total), 10.8s (subfind_density, 20.3%).

- Best performance HSW: 42.6s (total), 11.4s (subfind_density, 26.7%).

Parameter study of the MPI / OpenMP ratio on a KNL node.
Summary and outlook

- Along the described development steps, performance improvement on KNL is 1.2x for the whole code, 1.8x for the optimized kernel subfind_density.
- Improvements are portable also on Xeon (ongoing tests on newer versions).
- The improvement of subfind_density is in line with predictions based on the isolated kernel (1.4x), thus verifying our approach.
- Performance gap with Haswell: the original code was 1.7x slower on KNL, the optimized is 1.3x slower. For subfind_density: the original version was 1.50x slower on KNL, the optimized one only 1.16x slower → closing the gap!
- Room for further improvement?
  - Complete back-porting of further steps (data layout, vectorisation);
  - Back-port to other two major routines (~70% total time);
  - Explore and modernize also the MPI layer of the code.

More information: [www.lrz.de/services/compute/labs/astrolab/ipcc](http://www.lrz.de/services/compute/labs/astrolab/ipcc)
Acknowledgements

- Research supported by the Intel® Parallel Computing Center program.
- P-Gadget3 developers: Klaus Dolag, Margarita Petkova, Antonio Ragagnin.
- Research collaborator at Technical University of Munich (TUM): Nikola Tchipev.
- TCEs at Intel: Heinrich Bockhorst, Klaus-Dieter Oertel.
- Thanks to the IXPUG community for useful discussion.
- Special thanks to Colfax Research for granting access to their computing facilities.


To contact me: luigi.iapichino@lrz.de
Back-up: Back-porting the kernel optimizations to the full code

- To ease the back-porting, we defined a new Gadget test problem with a simplified but representative workload (2 * 64³ particles).

- From a physical viewpoint, this workload probes advanced phases of the galaxy evolution (inter-galactic medium is strongly clumped).

- Computationally, a reduced effort for finding particle neighbors!

- Improvement in execution time: 2.3x on Broadwell (Xeon E5-2699v4, 22 cores/socket), 5.3x on KNL. It was 4.7x and 19.1x for the old workload.
todo_partlist = partlist;

while(partlist.length){
  error=0;
  #pragma omp parallel for schedule(dynamic)
  for(auto p:todo_partlist){
    if(something_is_wrog) error=1;
    ngblist = find_neighbours(p);
    sort(ngblist);
    for(auto n:select(ngblist,K))
      compute_interaction(p,n);
  }
  //...check for any error
  todo_particles = mark_for_recomputation(partlist);
}
Back-up: some more KNL wisdom

- Quad-cache is a good starting point, quad-flat with allocation on MCDRAM is worth being tested, SNC modes are for very advanced developers.

- It is unlikely to gain performance with more than 2 threads/core.

- Vectorize whenever possible, use compiler reports and tools to exploit low-hanging fruits.

- Know where your data are located and how they move.

- If optimizations are portable, the effort pays off!