Improving the vectorisation of a Gadget kernel: efficiency and potential on multiple platforms

_Luigi Iapichino, Fabio Baruffa_

Leibniz-Rechenzentrum (LRZ), Garching b. München, Germany

**Collaborators:** N. Hammer, A. Karmakar (LRZ)
in the framework of the Intel® Parallel Computing Center in Garching (LRZ – TUM)

**Partners:** M. Petkova, K. Dolag (USM München, Germany)

Thanks to G. Zitzlsberger and Z. Matveev (Intel®) for interesting contributions to our analysis
Overview

- Modernising Gadget3 for the Intel® Xeon Phi™: focus on vectorization of a prototypical loop on particle neighbours.
  - Summary of previous work and status of the code.
  - Bottlenecks for vector performance and proposed solutions.
  - Performance improvements on IVB, HSW and KNC.

- Open questions and outlook.
The Gadget3 code

- Gadget: TreePM N-body + SPH code, numerical simulations of cosmological structure formation.

- Work performed on a stand-alone, pure Open-MP representative code kernel.

- Execution modes:
  - native on Intel® Xeon (tested on IVB and HSW) and
  - native on Xeon Phi™

- Main tools: Intel® Advisor XE 2016, compiler reports, own implemented timers.

- Diagnostics: time measures.
Previous work on the kernel

- Successfully implemented code improvements:
  - Restructuring of the parallelisation strategy as a lockless scheme (OpenMP dynamic scheduling)
  - Data locality: from AoS to SoA
  - Particle selection, instead of particle sorting
Vectorisation: the missing piece

- Initial analysis (Advisor)
  - Most of the kernel time spent in a while loop (tree-walk), not a good candidate to vectorize.
  - Kernel main “compute” loop: scalar time spent here 10% of the kernel on IVB and HSW, 25% on KNC.
  - A significant fraction of the vectorisation potential of this kernel.

- Prototypical loop in Gadget

- Similarity with many other N-Body codes
Obstacles to vectorization efficiency - pseudocode

for (n = 0, n < neighbouring particles (selected)) {
    j = ngblist[n];  // getting the index from the particle data structure (SoA)
    // Problem 1: if statement
    if (particle n within smoothing length) {
        inlined_function1(…..);
        inlined_function2(…..);
    }
    vx += NewPart.Vel[0][j];  // Problem 2: indirect (strided) access to the data
    ...
    v2 += NewPart.Vel[0][j] * NewPart.Vel[0][j] + … ;  // additional load
    // (unnecessary): why does the compiler not reuse it from the register?
}

Leibniz Supercomputing Centre
Our solutions to improve the vector efficiency

- Solution to problem 1:
  - “if” statement moved inside one of the inlined functions.
  - Much more localised masking and reduced overhead.
  - Performance improvement of vectorisation is platform-dependent.

- Optimising data loading: number of loads decreases, but without performance improvement.

- Irregular strided access: problem 2 is a remaining hotspot in our case.

- Performance comparisons:
  - Intel compiler v. 15.
  - Compiler flags –no-vec vs. –xhost (-mmic) to measure vector speed-up.
  - Timers implemented by ourselves.
for (n = 0, n < neighbouring particles (selected)) { 
    j = ngblist[n];  // getting the index from the particle data structure (SoA)
    
inlined_function1(.....);  // the if condition is moved inside the function
    inlined_function2(.....);
    
    vel1 = NewPart.Vel[0][j];  // still strided data access
    ...
    vx += vel1;  // optimised data load
    ...
    v2 += vel1 * vel1 + ... ;
}
Performance on IVB (E5-2650, 16 cores @ 2.6 GHz)

- Moderate increase of loop vector efficiency from 1.8x (before optimisation) to 2.2x.
- „Target vector efficiency“ without strided access: 3.0x
- Where is further vectorisation speed-up hiding itself?
Performance on KNC (5110P, 60 cores @ 1.1 GHz)

- Before optimisation: loop vectorises, but efficiency 1.0x
- After: 2.0x faster already in scalar, plus 4.5x vector efficiency = loop 9.0x faster!
Performance comparison

- Initial version vs. vectorised including all optimisations.
- IVB, HSW: 1 socket w/o hyperthreading. KNC: 1 MIC, 240 threads.
- Once more, performance gain for KNC larger than for Xeon.

- Side remark: good vector performance on KNC and moderate on IVB, but the described loop optimisation does not provide good vec. results on HSW.
Summary and outlook

- Analysis of a representative kernel of Gadget3.
- Optimisation of a prototypical vector loop.
- Vectorised workload in the kernel is quite small, but important lessons learnt in view of backporting to Gadget.
- Encouraging performance on Xeon Phi, in view of using future many-core systems.
- Do the algorithm expose enough parallelism on many-core systems?
- To be further tested on KNC: data prefetching and alignment.
- For future investigation: comparison of energy to solution.