Vectorisation efficiency in a Gadget kernel: dealing with conditionals and data access

Luigi Iapichino

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

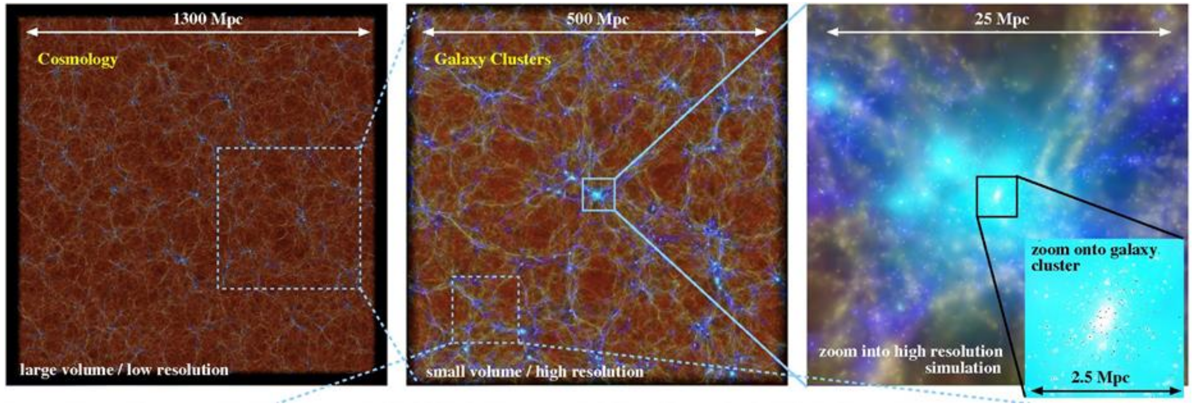
Collaborators: N. Hammer, A. Karmakar (LRZ)

in the framework of the Intel® Parallel Computing Center in Garching (LRZ – TUM)

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

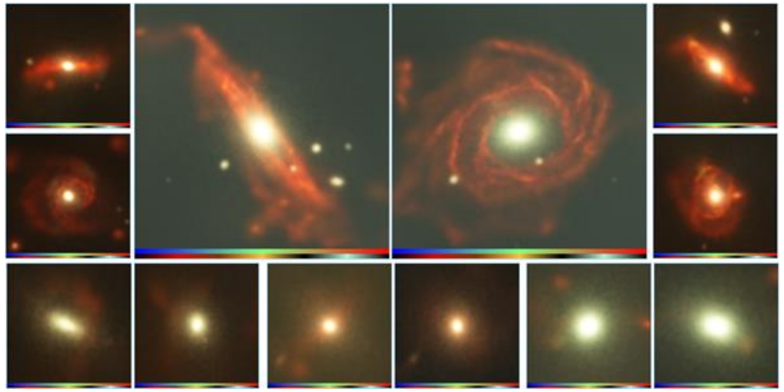
Background

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



- Work performed on a stand-alone, representative code kernel

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



- Main tools: Intel® Advisor 2016, compiler reports

Motivations of this work

- Successfully implemented code improvements:
  - Particle selection, instead of particle sorting
  - Restructuring of the parallelisation strategy as a lockless scheme (OpenMP dynamic scheduling)
  - Data locality: from AoS to SoA
  - Previous performance improvement with respect to original baseline: 5.8x on Xeon IVB, 13.3x on KNC.
- Vectorisation: work (in progress) on the kernel main compute loop
  - Roughly 90% of the vectorisation potential of this kernel
  - Prototype loop in the Gadget code
- 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)

        if (particle n within smoothing length) {              // Problem 1: if statement
           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?
}
```

Results

- Original vector efficiency: 36%, Advisor estimates a gain of 1.4x (host system: Xeon IVB node, using AVX)
- Optimising data loading: number of loads decreases, estimated efficiency goes to 42%

- Solution to problem 1:
  - "if" statement moved inside one of the inlined functions, resulting in a much more localised masking and reduced overhead.
  - Advisor efficiency now > 90% on IVB, although the *measured* speed-up on the loop is ~ 2.3x.
  - On a HSW node, using AVX2: both Advisor estimate and measure match better, speed-up ~ 3.0x.

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

- In the Gadget kernel under consideration, the time spent in vector loops is small
  - Overall gain in performance is ~ 1.1x both on Xeon and on KNC.
  - However: useful lessons to be learnt in view of backporting, applicable to several similar loops in Gadget.

- More todo: analysis of inlined functions (Advisor 2016 Upgrade 1), and work on data alignment

Optimised pseudocode

```
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: next exposed hotspot
        …
        vx += vel1;                                    // optimised data load
        …
        v2 += vel1 * vel1 + … ;
}
```

Backup – additional analysis

- Analysis in collaboration with G. Zitzlsberger and Z. Matveev (Intel)

- Performance of the considered loop on IVB vs. HSW: in the latter, one can greatly benefit from AVX2 ISA
- Thus, simplified code generation and FMAs -> better performance even in the scalar version
- This results also in better Advisor “gain estimate” prediction on HSW
- Inlined functions: analysis available on Advisor 2016 Upgrade 1