Improving node-level performance in Gadget: data structure and data locality

Luigi Iapichino

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

Collaborators: V. Karakasis, 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)
Opimisation strategy

- Gadget3: publicly available, cosmological TreePM N-body + SPH code. Good scaling performance up to 130,000 Sandy Bridge cores (SuperMUC, Extreme Scaling workshop 2013 @ LRZ).
- However: performance optimization at node level and the use of accelerators had gone largely unexplored before our work.
- Initial analysis: most of the code components consist of two sub-phases of nearly equal execution time (40 to 45% for each of them).
- The most suitable for the optimization and execution on Intel® Xeon Phi™ (higher floating-point rate, sustainable cache and memory b/w requirements, but data cache misses) will be the target of our work.
- Isolation of a typical kernel (subfind_density):
  - Run as a stand-alone separate kernel (same input as original: sandbox model!).
  - Avoid the overhead of the whole simulation → Quick prototyping, allows native mode on the Xeon Phi™.
  - Later: port optimizations back to the original code.
• Current data organisation: Array of Structures (AoS), 224 bytes per particle.

• Motivation: highly optimized for performance at large MPI task numbers.

• Outcome: data cache misses, code is memory latency bound. Data structure hinders vectorisation.

• In the kernel: ~17 iterations, 1.5M particles to be processed.
Proposed solution: SoA

- New particle data structure: defined as Structure of Arrays (SoA).
- From the original set, only variables used in the kernel are included in the SoA: ~ 60 bytes per particle.
- Software gather / scatter routines.
- Gather from old to new data structure, compute with it, scatter back to old. Example of change in the data structure approach:

```c
struct new_particle_data {
    MyDoublePos *Pos[3];
    MyFloat *Vel[3];
    short int *Type;
    MyIDType *ID;
    MyFloat *Mass;
    int *DM_NumNgb;
    MyFloat *DM_Hsml;
    MyFloat *DM_Density;
    MyFloat *DM_VelDisp;
};

void gather_particle_data(struct new_particle_data *dst, const struct particle_data *src, size_t N) {
    int i;

    #pragma omp parallel for
    for (i = 0; i < N; i++) {
        dst->Vel[1][i] = src[i].Vel[1];
        dst->Vel[2][i] = src[i].Vel[2];
        dst->Type[i] = src[i].Type;
        dst->ID[i] = src[i].ID;
        ...;
    }
}
```
Gather+scatter overhead small when compared both to execution time and to performance gain.

Node-level performance improvement: +22% on the Xeon, +41% on the Xeon Phi™. Xeon/Xeon Phi™: 0.28

Bottleneck on memory latency is solved: Memory latency metric (VTune) from 0.208 to 0.098.

Data structure is now vectorisation-ready, although vectorisation has been completely disabled at this stage.

Cache behaviour: improved performance by ~40%.

<table>
<thead>
<tr>
<th>Stall type</th>
<th>% cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>AoS</td>
<td></td>
</tr>
<tr>
<td>L1D miss</td>
<td>8.49 %</td>
</tr>
<tr>
<td>L2 miss</td>
<td>7.99 %</td>
</tr>
<tr>
<td>LLC miss</td>
<td>16.27 %</td>
</tr>
<tr>
<td>TOTAL</td>
<td>32.75 %</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stall type</th>
<th>% cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoA</td>
<td></td>
</tr>
<tr>
<td>L1D miss</td>
<td>3.75 %</td>
</tr>
<tr>
<td>L2 miss</td>
<td>3.16 %</td>
</tr>
<tr>
<td>LLC miss</td>
<td>12.32 %</td>
</tr>
<tr>
<td>TOTAL</td>
<td>19.23 %</td>
</tr>
</tbody>
</table>
Insights and next steps

- Work on a representative Gadget3 kernel.
- Data structure and data locality: a first step towards vectorisation.
- Also part of our work:
  - Shared-memory parallelisation improvements
  - Other algorithmic improvements: selecting nearest particles.
- In general: optimisation is a win-win game, but the Xeon Phi™ wins more.
- Coming soon:
  - Lockless parallelisation scheme.
  - Port node-level code improvements back to Gadget3.