Optimizing Smoothed Particle Hydrodynamics Code Phantom on Haswell and KNL

Charlene Yang
Chris Bording
Pawsey Supercomputing Centre, Perth, Australia

Daniel Price
Rebecca Nealon
Monash University, Melbourne, Australia

Abstract

This poster presents the optimization work carried out on a Smoothed Particle Hydrodynamics (SPH) code Phantom on Haswell and KNL. With 8 steps of optimization, the performance results show 3x speedup on Haswell and 4x on KNL. A few remarks are provided to highlight the differences on KNL in terms of architecture and optimization strategy. This work will also serve as an example of optimizing other codes, such as those in molecular dynamics, computer graphics and neural networks that also deal with *k*d-tree, neighbour search, irregular computation and random memory accesses.

Codebase and Platforms

Phantom [1] is a smoothed particle hydrodynamics and magnetohydrodynamics code for Astrophysics. It is parallelized with OpenMP and widely used for studies of accretion discs, turbulence and star formation (Fig.1). There are four major subroutines [2], namely, *maketree*, *getneigh*, *densityiterate* and *force*. *maketree* builds a three dimension tree for all the particles (Fig.2), and *densityiterate* and *force* calculate the density and force of a particular particle based on the neighbour list provided by *getneigh* through tree traversal.

Fig.1: A misaligned accretion disc around a spinning black hole

A representative simulation is the *rndisc* simulation, which executes all major subroutines and runs for about 60s on 16 cores for a 10^6- particle, 10-timestep (112-subtimestep) setup. We will be using this as the case of study in this work and will be running on the following two platforms.

	Haswell E5-2690v3	Xeon Phi 7210 (quad-flat)
Cores	12 cores x2 @2.6GHz	64 cores @1.3GHz
TDP	135 W x2	215 W
Memory	64GB DDR	96GB DDR + 16GB MCDRAM
Cache	L1 32K, L2 256K, L3 30M	L1 32K, L2 512K (1M shared)

Optimization Path and Results

Step 1:

Adjust OpenMP scheduling. Since the total number of cells are in the 100,000's and 70% of them are empty or inactive (hence need to be skipped), the chunksize is increased from 10 to 2000 for the two loops in *densityiterate* and *force*.

Step 2:

Adjust number of threads to be a power of 2. Even though there are 24 cores on Haswell, 16 threads prove to have a better load balance since the tree is based on a binary build and every level in the breadth-first stage has a 2^n (n > 0) number of nodes (Fig.3). Number of threads on KNL stays the same, i.e. 64. Extra code will be added to utilize the other cores on Haswell.

Employ nested OpenMP parallel regions. Two serial code regions in *maketree* are only executed by master thread in the breadth-first build. Nested OMP regions are used to improve thread concurrency so that all threads work on node 1, 2 teams of half the threads work on node 2-3 simultaneously, and so on (Fig.3).

Step 3:

Remove linked lists in *maketree* to reduce random memory access and LLC miss. This also helps reduce the serial code since part of it can now be parallelized with the 3 new arrays representing the tree.

Move particles as the tree is built. Particle positions are copied at each level according to tree structure so memory access is contiguous at the next level. Memory movement can be costly but can be vectorized with *Compress* instruction on KNL. This is not available on Haswell, but improved memory access pattern has proved to have enough benefit to overcome this (through vectorization) (Fig.7).

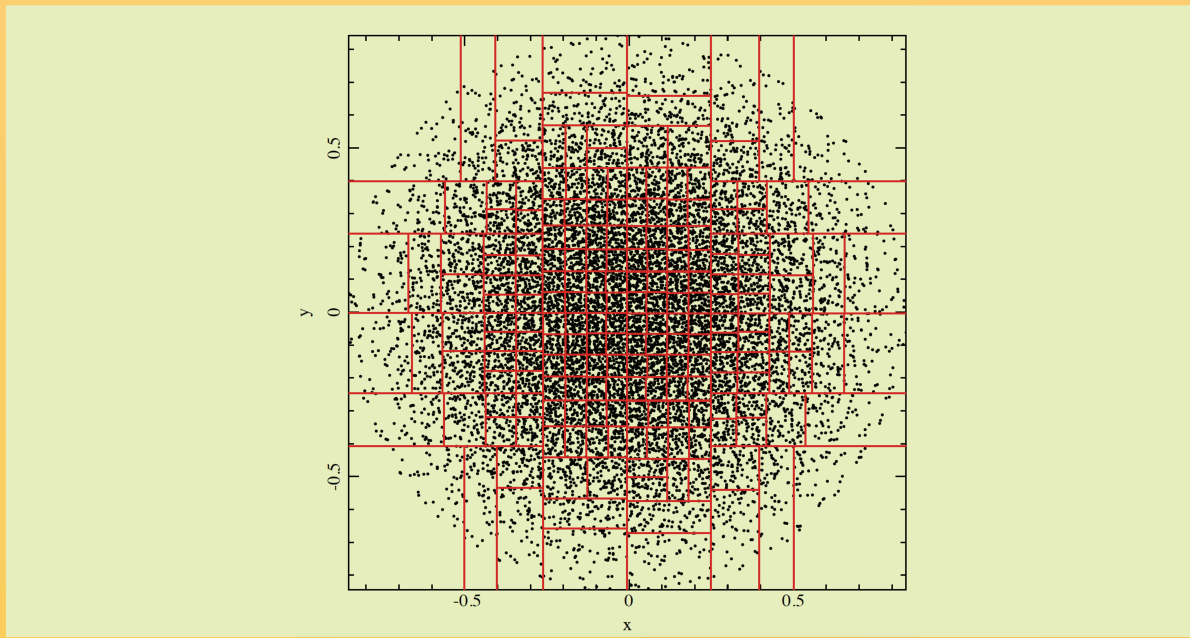

Fig.2: *k*d-tree construction. Each level of the tree recursively splits the particle distribution in half, bisecting the longest axis at the centre of mass until the number of particles in a given cell is < $N_{min.}$

Step 4:

Change data layout from AoS to SoA for *xyzh* in *maketree* to improve vectorization efficiency in the calculation of centre of mass, node size, quadruple moments and other quantities.

Step 5:

Vectorize the inner loop of ***getneigh***. There is dependency in the outer loop since child nodes cannot be determined to be relevant until the parent node has. Inner loop is vectorizable since loop count is known with the new tree representation from Step 3. With branches moved outside, it is efficiently vectorized.

Data structure of cache arrays *xyzcache* and *dxcache* are changed from AoS to SoA to make vectorization in *get_dens_sums* and *compute_force* more efficient. Self particle is excluded from the neighbour list to help remove the *cycle* **statement** in *get_dens_sums* and *compute_force* to make them more efficient in Step 6-7.

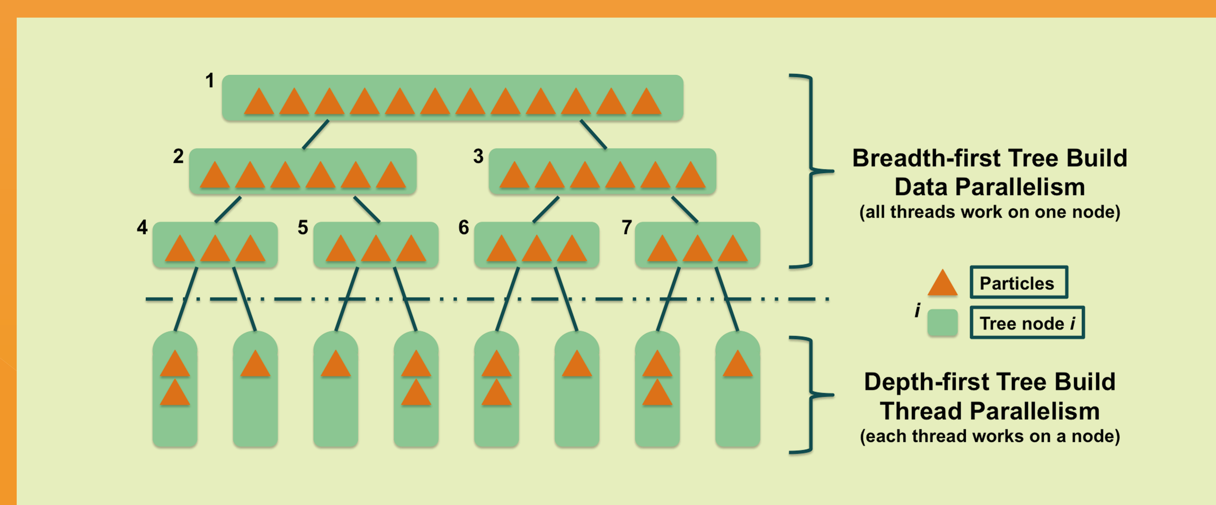

Fig.3: Different parallelisms in the *k*d-tree build

Step 6-7:

Vectorize density and force summation loops in *get_dens_sums* and *compute_force*. **Branches** are brought outside, **functions** are inlined, and **loop fission** is applied based on the sparsity of data.

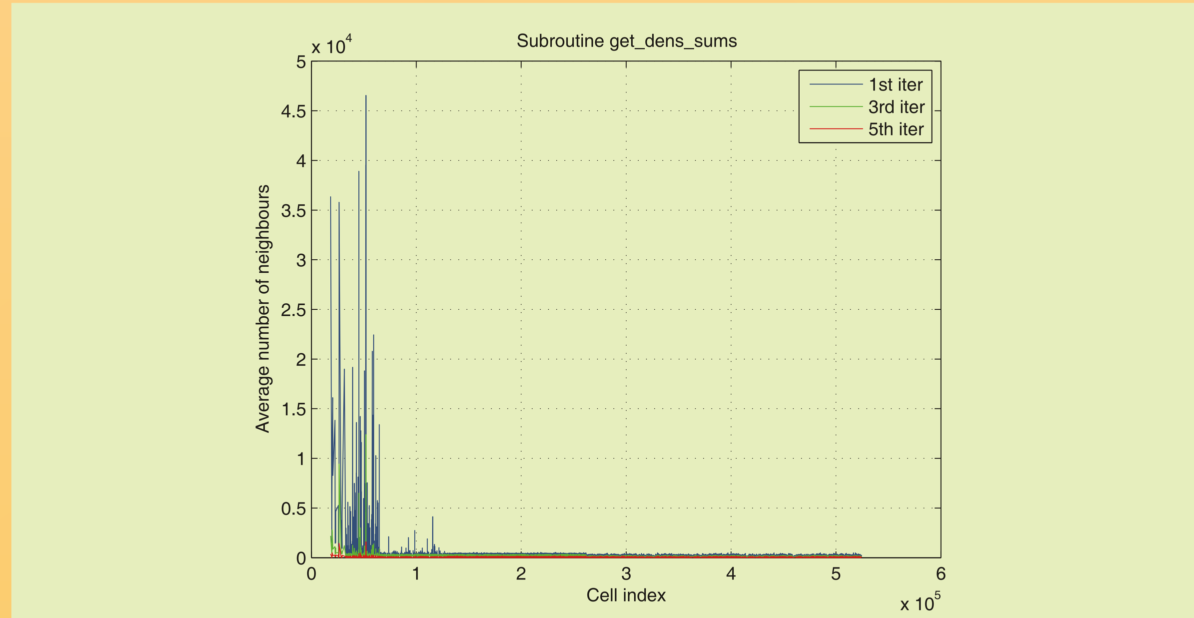

Fig.4: Data sparsity in density summation

Neighbours that satisfy a certain criteria are only 5-10% of the neighbours provided by *getneigh* for some cells, and it gets even more sparse as the iterative process goes on (Fig.4). To make sure summations have dens memory to access, loops are broken up into two parts, loops 1-4 and loop 5 in *get_dens_sums* (Fig.5) (similar in *compute_force*). Sparse data is collected (loop 3) and loops are vectorized before and after the breakpoint. Again, loop 3 can be vectorized with ***Vector Compress*** on KNL but not on Haswell. But there is still improvement observed in performance (Fig.7) which mainly comes from the vectorization of loop 2 and 5.

```
  q2icounter = 0
1 do nn=1,(nneigh/nneighblk)*nneighblk,nneighblk     cache blocking
2     do n = nn,nn+nneighblk-1                        pure compute, vectorized
         calculate rij2, q2i(n)
         enddo
         counter=0
3        do n = nn,nn+nneighblk-1                     vector compress, KNL only
         if (q2i(n) < radkern2hi21) then
             counter=counter+1
             cache(q2icounter+counter)= q2i(n)
         endif
         enddo
         q2icounter=q2icounter+counter
  enddo

4 do n=(nneigh/nneighblk)*nneighblk+1,nneigh          remainder loop
  enddo

5 do n=1,q2icounter                                   pure compute, vectorized
      calculate rhosum
  enddo
```

Fig.5: Loop in *get_dens_sums* is split at *if (q2i < radkerns)*.

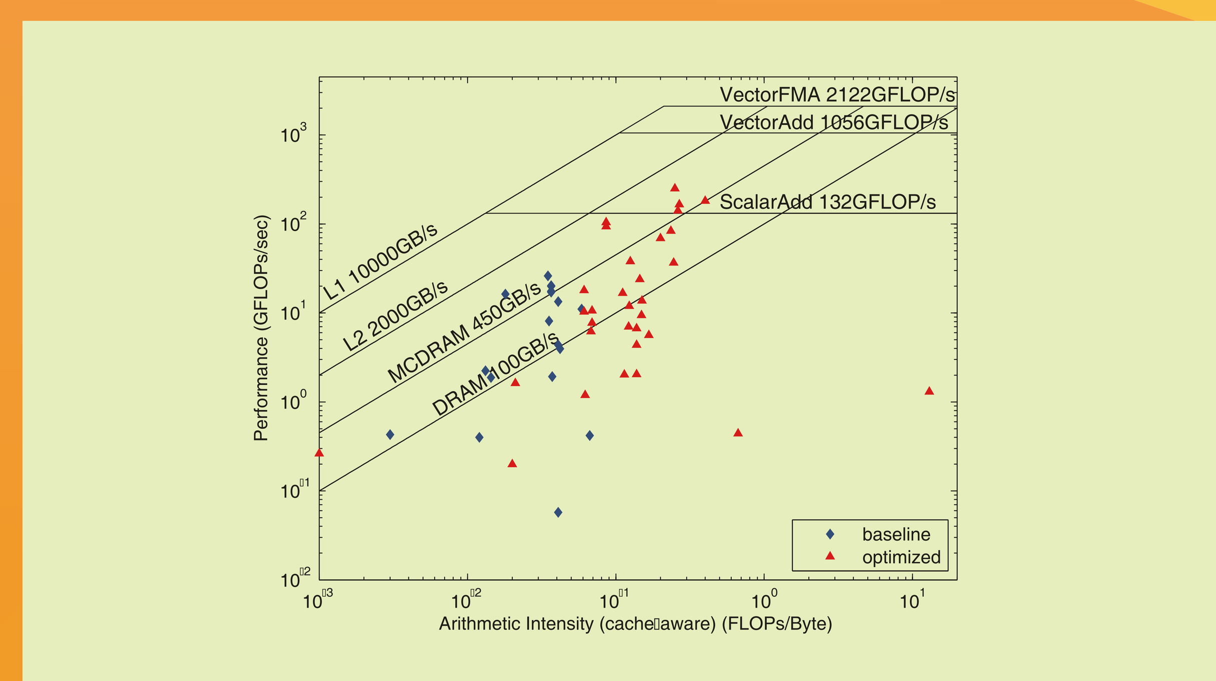

Fig.6: Cache-aware roofline model on KNL

Step 8:

Improve vectorization efficiency in *get_dens_sums* and *compute_force*. Add *CONTIGUOUS* attribute to assumed shape arrays to avoid **multiversioning**. Adjust size of cache arrays to make sure that column length is a multiple of vector length and so all columns are 32byte-aligned on Haswell for **multidimensional arrays** and 64byte on KNL. Use *!$omp simd declare* annotation, *!DIR$ FORCEINLINE* directive, and *-ipo* compilation flag to ensure function calls (and calls within calls) are properly inlined.

After these 8 steps, loops have been moved rightwards and upwards on the roofline model for both Haswell and KNL (Fig.6 for KNL). Walltime has been reduced by 3x on Haswell and 4x on KNL (Fig.7), and this means months of time is saved for a typical simulation in real life with low-to-medium resolution and millions of timesteps.

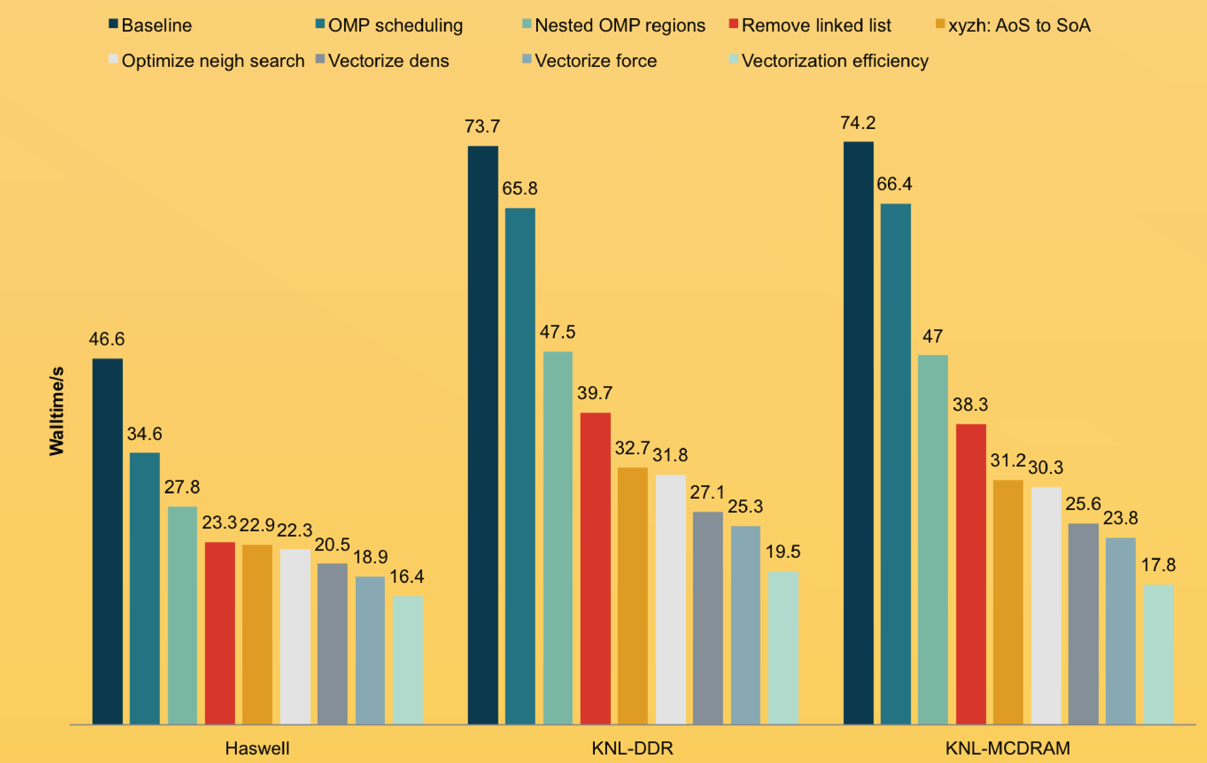

Fig.7: Optimization path

Remarks on Optimization Strategies

- **Thread parallelism:** KNL has more cores than Haswell but all are running at a much lower frequency. Well balanced load is critical for codes to achieve good thread concurrency and thus performance.
- **Data parallelism:** KNL has wider vectors and improved instruction sets. Data parallelism should be exploited as much as possible to auto-vectorize and to use SIMD instructions.
- **Memory awareness:** Data layout, alignment and spacial/temporal locality is more important on KNL than Haswell due to its wider vectors and lack of L3 cache. Vector-friendly and cache-friendly data arrangement is greatly encouraged. MCDRAM also favors cache-friendly codes because even though it has higher bandwidth, its latency still stays close to DDR's.

References

[1] *https://phantomsph.bitbucket.io.*

[2] Daniel J. Price, James Wurster, and Chris Nixon et al. Phantom: A smoothed particle hydrodynamics and magnetohydrodynamics code for astrophysics. *arXiv:1702.03930.*

GOVERNMENT OF WESTERN AUSTRALIA
Australian Government
NCRIS National Research Infrastructure for Australia
An Australian Government Initiative
CSIRO
Curtin University
ECU EDITH COWAN UNIVERSITY
Murdoch UNIVERSITY
THE UNIVERSITY OF WESTERN AUSTRALIA
MONASH University