SWIFT: Using Task-Based Parallelism, Fully Asynchronous Communication and Vectorization to achieve Maximal HPC performance

James S. Willis

Computational Scientist

Institute for Computational Cosmology, Durham University, UK

Durham University

This work is a collaboration between 2 departments at Durham University (UK):

- The Institute for Computational Cosmology,
- The School of Engineering and Computing Sciences,

  with contributions from the astronomy group at the university of Ghent (Belgium), St-Andrews (UK), Lausanne (Switzerland) and the DiRAC software team.

This research is partly funded by an Intel IPCC since January 2015.

Overview

- Motivation behind SWIFT
- Problem that we need to solve
- SWIFT’s solution to the problem

Durham University

What we do and how we do it

- Astronomy / Cosmology simulations of the formation of the Universe and galaxy evolution.
- EAGLE project[1]: 48 days of computing on 4096 cores. >500 TBytes of data products (post-processed data is public!). Most cited astronomy paper of 2015 (out of >26000).
- Simulations of gravity and hydrodynamic forces

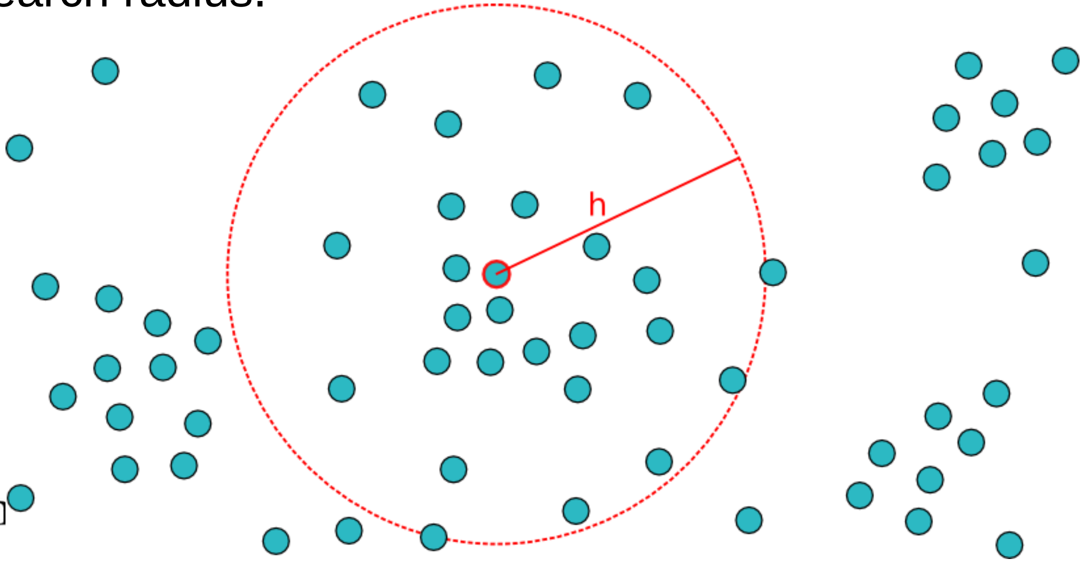
One simulated galaxy out of the EAGLE virtual universe.

Durham University

1) www.eaglesim.org

SPH: The problem to solve

For a set of N ($>10^9$) particles, we want to exchange hydrodynamical forces between all neighbouring particles within a given (time and space variable) search radius.



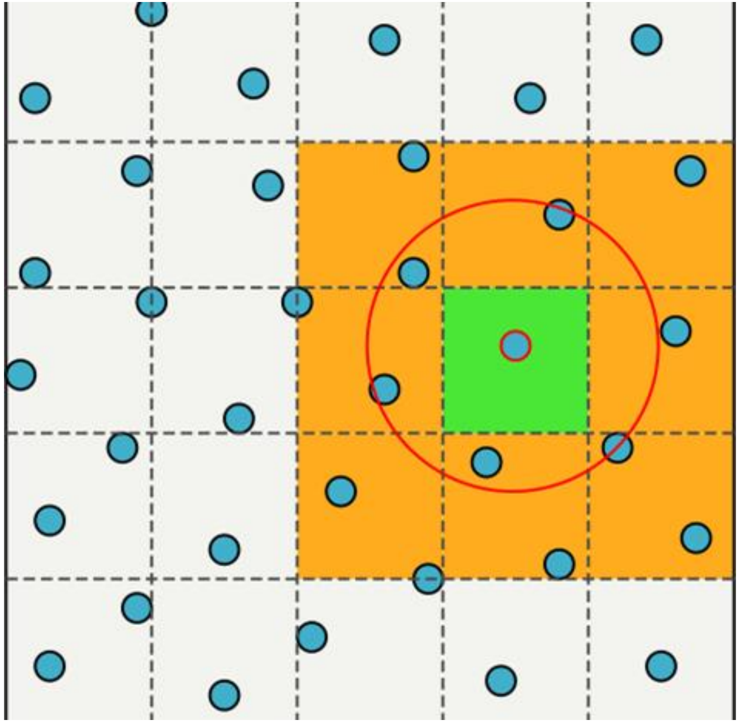
SPH: Challenges

- Particles are unstructured in space, large density variations.
- Particles will move and their neighbour lists will evolve over time
- Interactions between particles are computationally cheap to perform (low flop/byte ratio)

SPH: The SWIFT solution

We need to make the problem regular and predictable for load balancing:

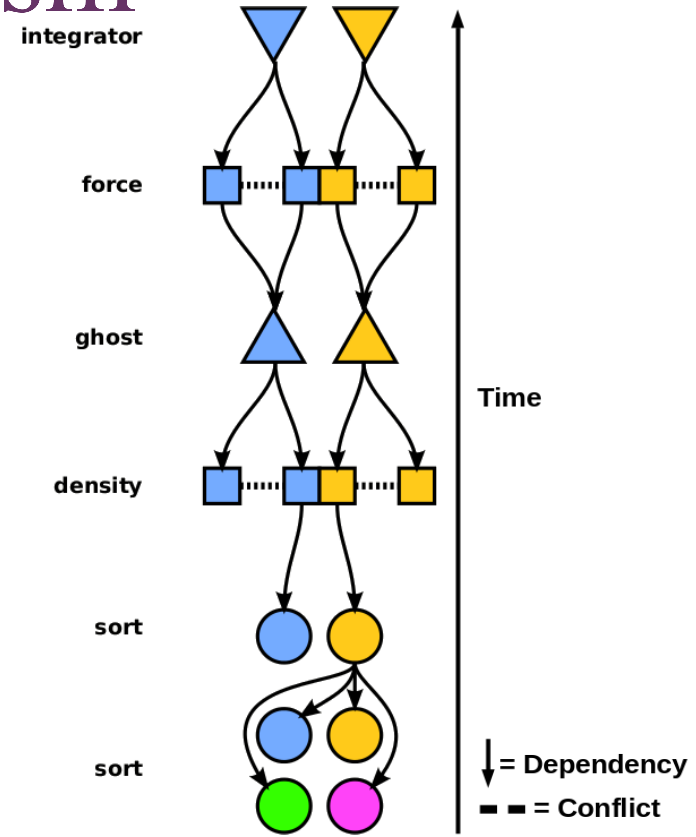
- Neighbour search is performed via the use of an adaptive grid constructed recursively until we get ~500 particles per cell
- Cell spatial size matches search radius
- Particles only interact with partners in their own cell or one of the 26 neighbouring cells



Durham University

Task Based Parallelism

- Decompose the problem into a set of inter-dependent tasks which form a task graph
- Each task has a set of dependencies and conflicts
- Each thread then executes a task that has no unresolved dependencies or conflicts



Durham University

SPH: The SWIFT solution

```
for(int ci=0; ci < nr_cells; ++ci) {  // loop over all cells
    for(int cj=0; cj < 27; ++cj) {    // loop over all 27 cells neighbouring cell ci

        const int count_i = cells[ci].count;
        const int count_j = cells[cj].count;

        for(int i = 0; i < count_i; ++i) {
            for(int j = 0; j < count_j; ++j) {

                struct part *pi = &parts[i];
                struct part *pj = &parts[j];

                INTERACT(pi, pj);   // symmetric interaction
} } } }
```

Durham University

SPH: The SWIFT solution

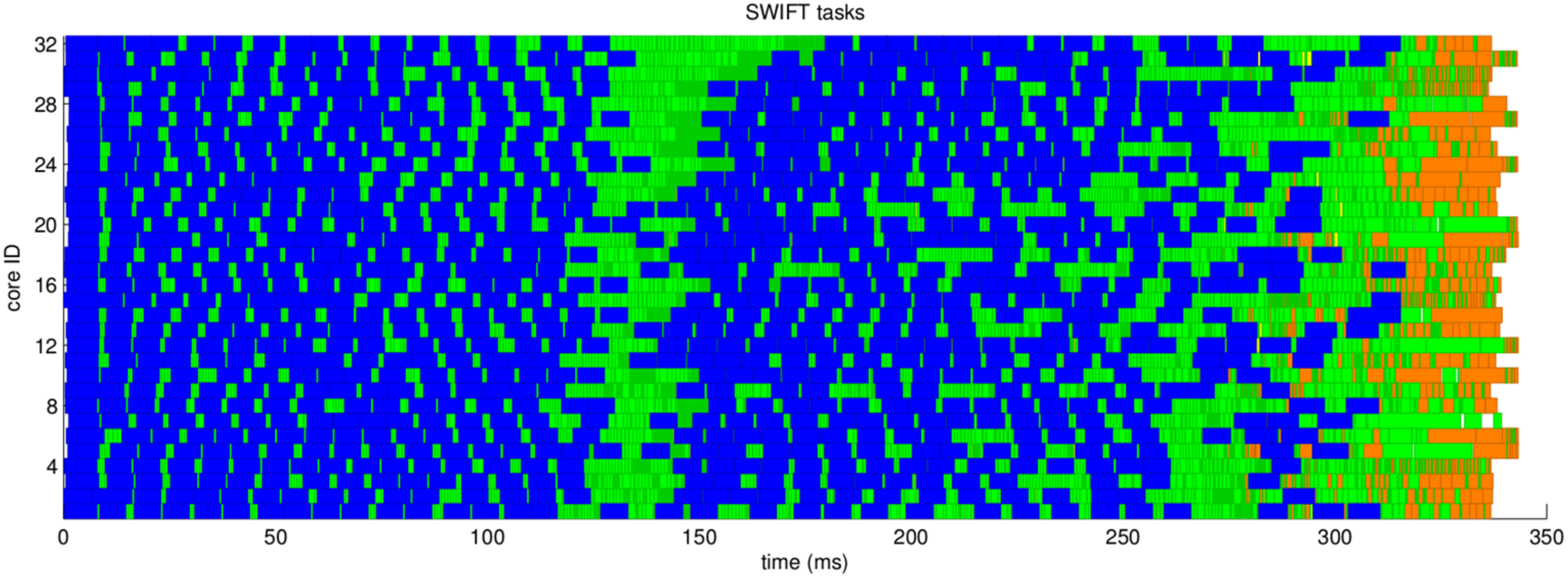
Threads + MPI

```
for(int ci=0; ci < nr_cells; ++ci) {  // loop over all cells
    for(int cj=0; cj < 27; ++cj) {    // loop over all 27 cells neighbouring cell ci
----------------------------------------------------------------------------------
        const int count_i = cells[ci].count;
        const int count_j = cells[cj].count;

        for(int i = 0; i < count_i; ++i) {
            for(int j = 0; j < count_j; ++j) {

                struct part *pi = &parts[i];
                struct part *pj = &parts[j];

                INTERACT(pi, pj);   // symmetric interaction
} } } }
```

Vectorization

Durham University

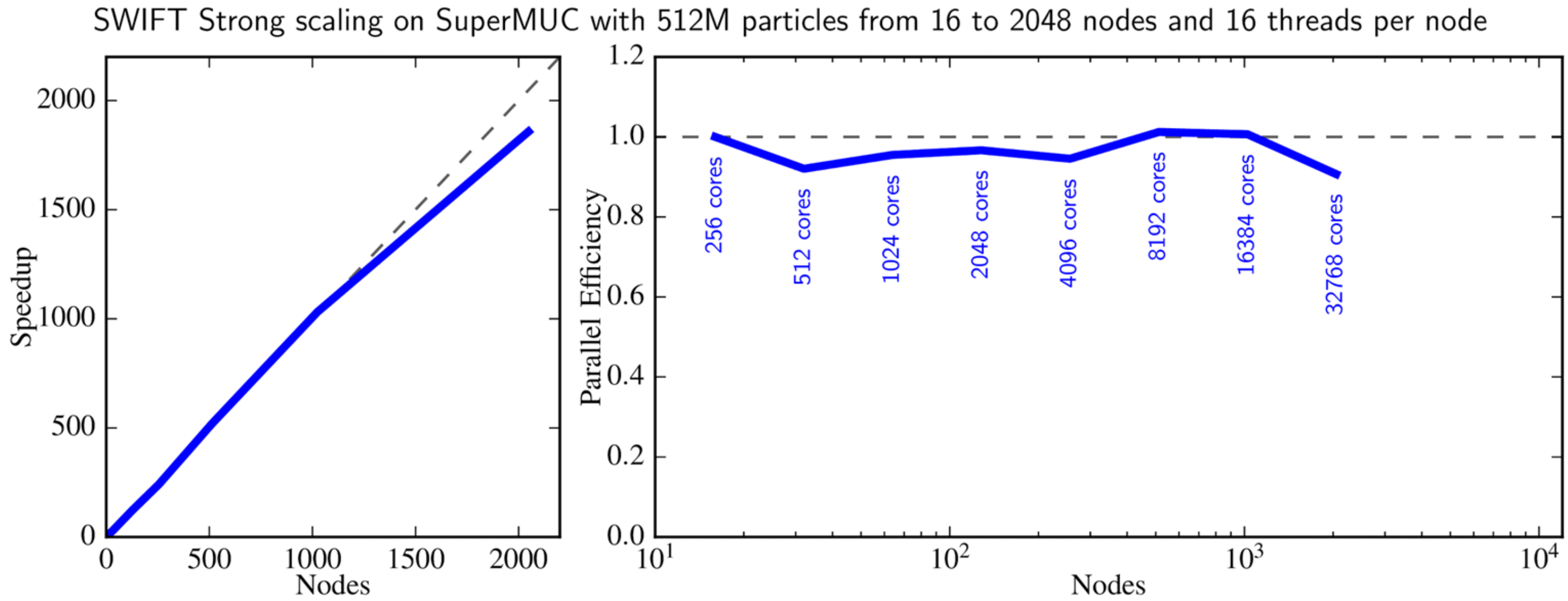
Single node parallel performance



Task graph for one time-step. Colours correspond to different types of task. Almost perfect load-balancing is achieved on 32 cores.

Durham University

Scaling results: SuperMUC



System: x86 architecture - 2 Intel Sandy Bridge Xeon E5-2680 8C at 2.7 GHz with 32 GByte of RAM per node.

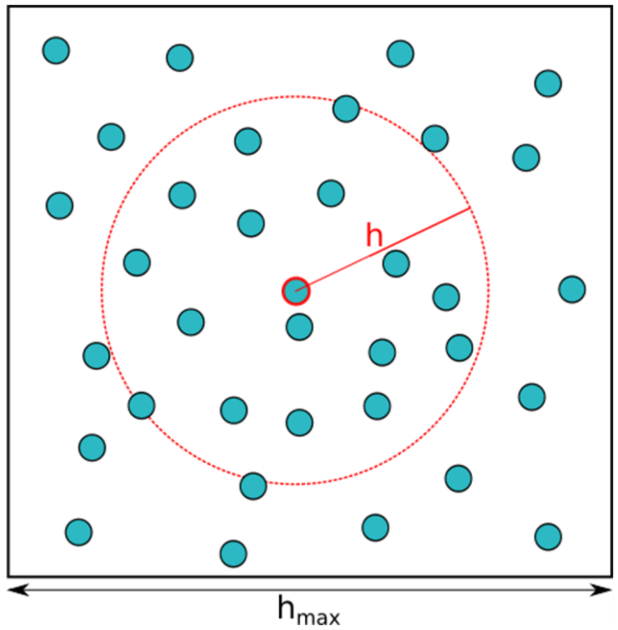
Durham
University

SIMD strategy

Example of a task interacting all particles within one cell.

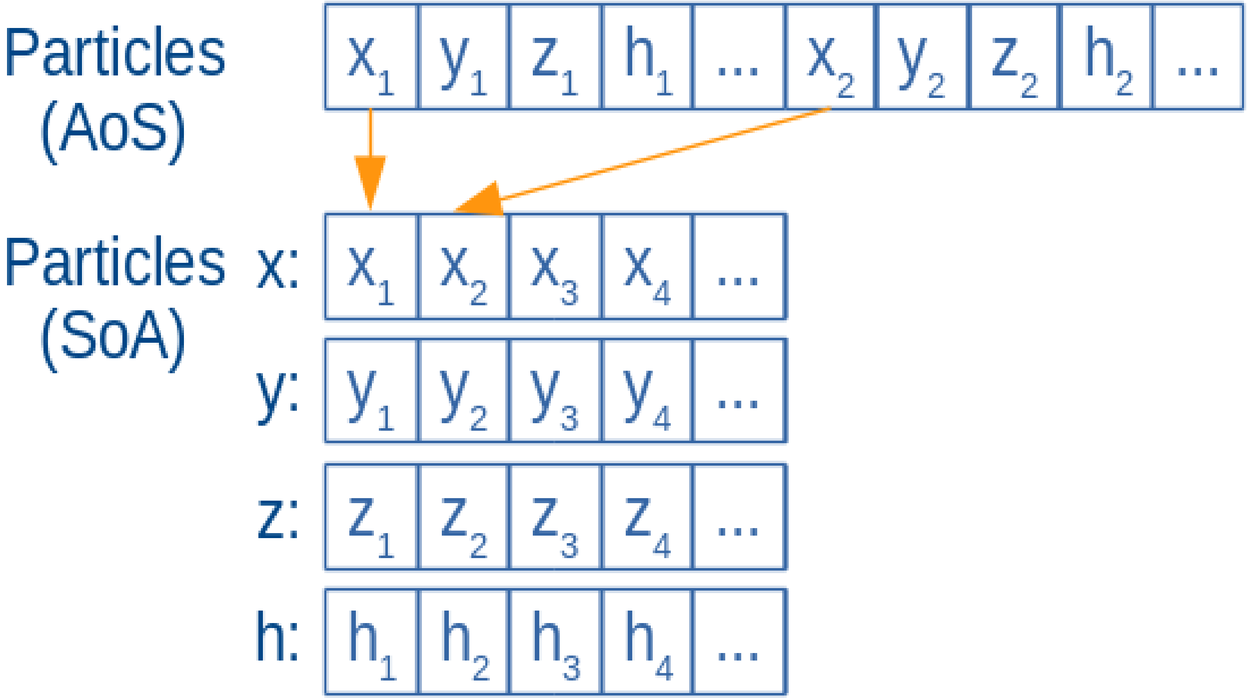
Thanks to our task-based parallel framework:

- No need to worry about MPI
- No need to worry about threading or race conditions
- Full problem is held in the L2 cache

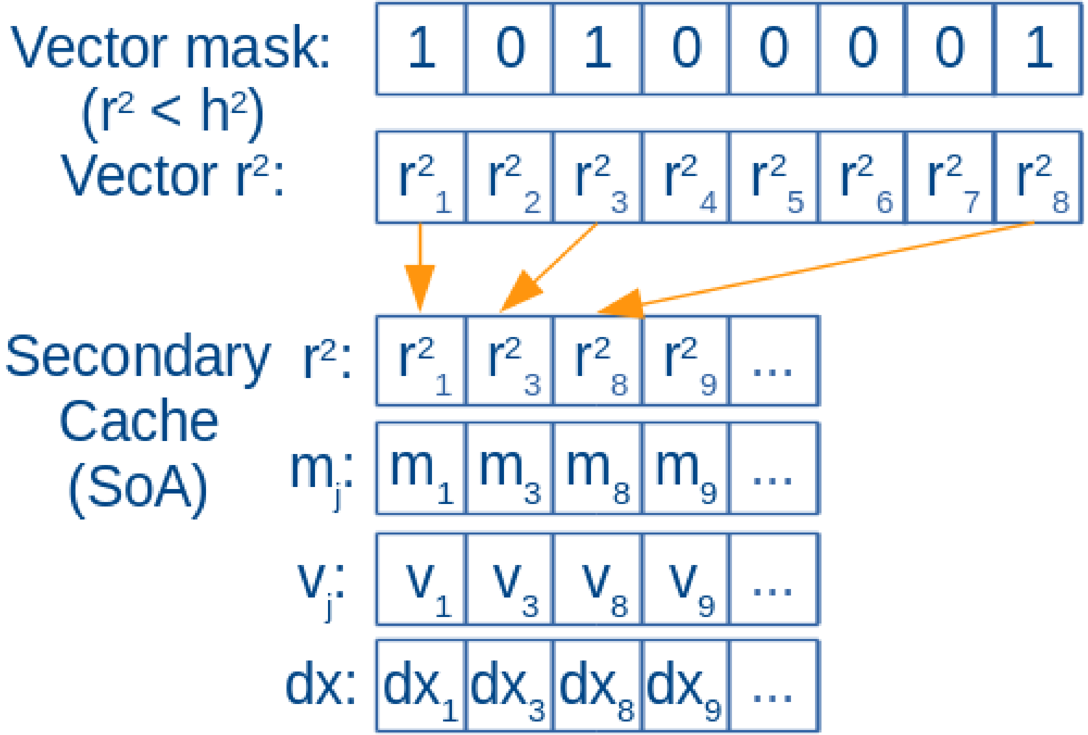


Durham University

Step 1: Form local cache of particles



Step 2: Find pairs and pack them into a 2nd cache



Durham University

Step 3: Process all pairs in the 2nd cache

```
vector densitySum;
density = setzero();

for (int pjd = 0; pjd < icount; pjd+=VEC_SIZE) {
  INTERACT(&c2_r2[pjd], &c2_dx[pjd], &c2_dy[pjd],
           &c2_dz[pjd], &c2_m[pjd], &c2_v[pjd],
           &densitySum);
}

VEC_HADD(densitySum,pi);
```

Durham University

Vectorization results

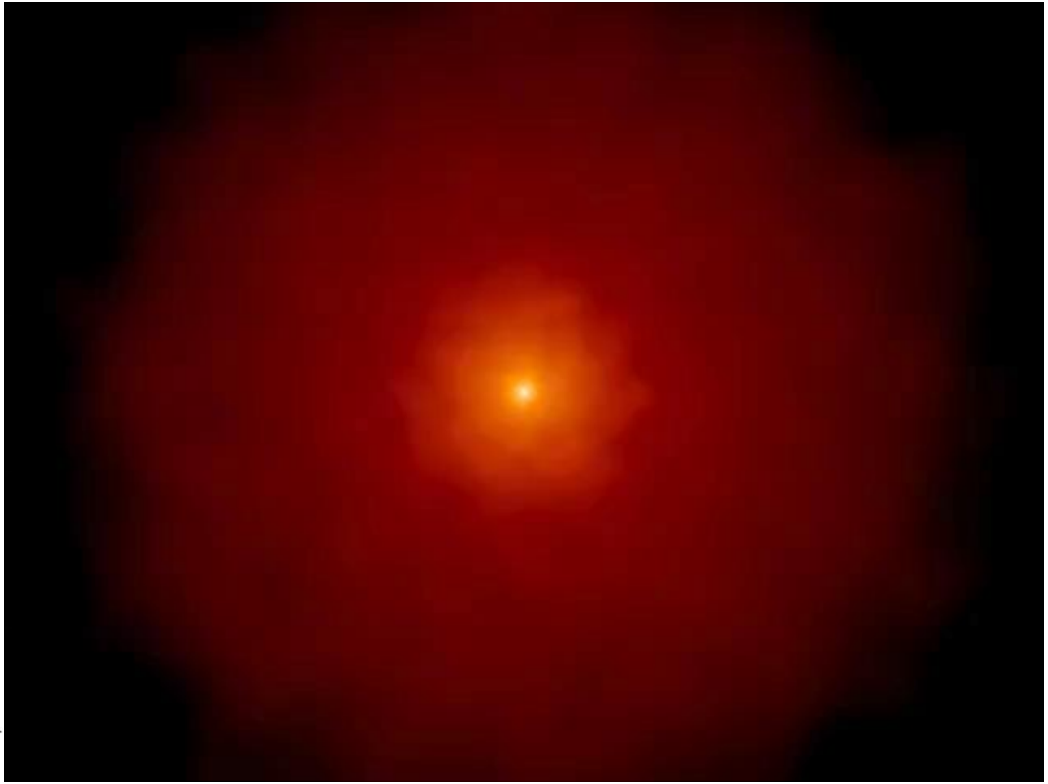
CFLAGS	Speed-up over naïve brute force	Speed-up over best serial version
-O3 **-xAVX**	**2.93x**	1.94x
-O3 **-xCORE-AVX2**	**3.64x**	2.74x
-O3 **-xMIC-AVX512**	**4.37x**	2.80x

Better than the factor of 2x obtained from the auto-vectorizer

In the scalar case, there is a faster algorithm with the comparison shown here for fairness

Durham University

Formation of a galaxy on a KNL



Durham University

Conclusions

- Completely open-source software including all the examples and scripts
- ~30,000 lines of C without fancy language extensions
- Good parallel performance up to 32,000+ cores thanks to:
  - Task-based parallelism
  - Improved data locality
  - Asynchronous MPI communication
  - SIMD strategy

Durham University

Questions

- Thank you for your attention
- Any questions?
- The code is free to download at: http://icc.dur.ac.uk/swift/