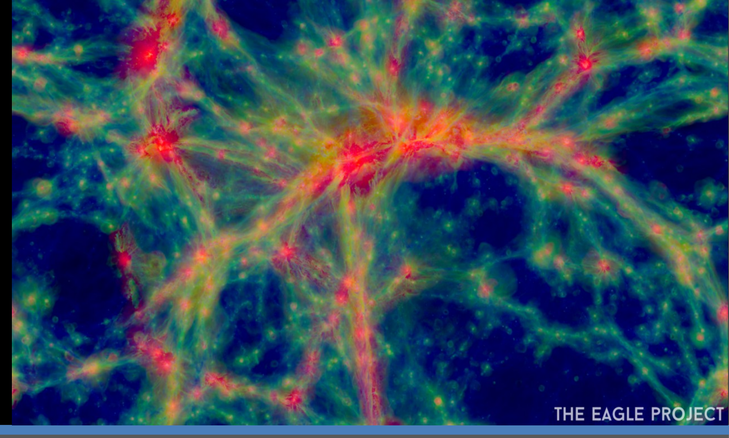
Vectorisation of SWIFT



James Willis | Richard Bower, Matthieu Schaller, Pedro Gonnet | Institute for Computational Cosmology, Durham University
Contact: james.s.willis@durham.ac.uk

Introduction

SWIFT is an SPH code that utilises the concept of task-based parallelism to distribute its workload across multiple processors.

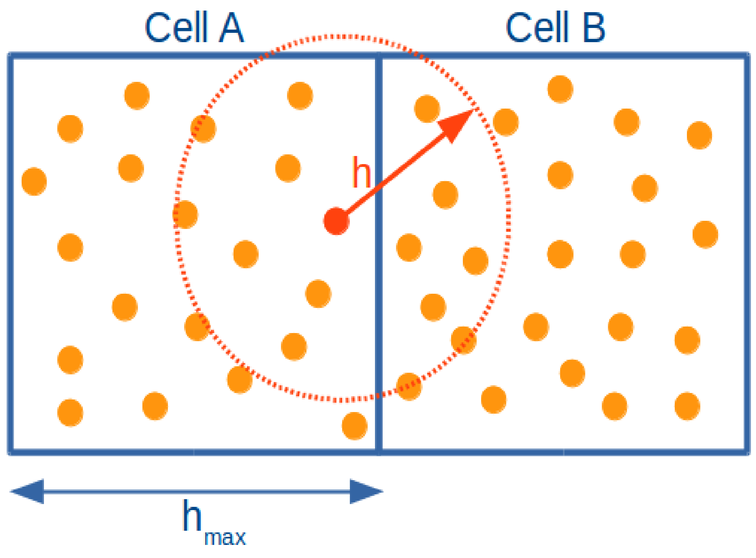
Overview

The improved performance of SWIFT is the result of several features:

- **Task-based parallelism:** decomposes the problem into a task graph that specifies all *dependencies* and *conflicts* between tasks. The tasks are then scheduled on a processor based upon this graph, which provides a better fine-grained load balancing.
- **SIMD vectorisation:** takes advantage of CPUs' vector units to parallelise computation and make use of single-precision values where excessive accuracy is not needed.
- **Hybrid shared/distributed memory parallelism:** tasks that require data from neighbouring nodes are scheduled after the asynchronous transfers are complete, which means that communication latencies can be hidden with computation.
- **Graph-based domain decomposition:** uses information collected from the task graph to decompose the simulation domain such that the work is equally distributed across all nodes.

Problem

The problem we face in SPH is that each particle must be interacted with each of its neighbours that fall within a smoothing length, **h**.



However, it is very time consuming to loop over all particles in a neighbouring cell and test whether they interact or not.

Solution

One solution is to sort the particles along the axis linking two neighbouring cells. This greatly reduces the number of unnecessary computations as you only need to loop over particles that are a distance **h**, on the axis.

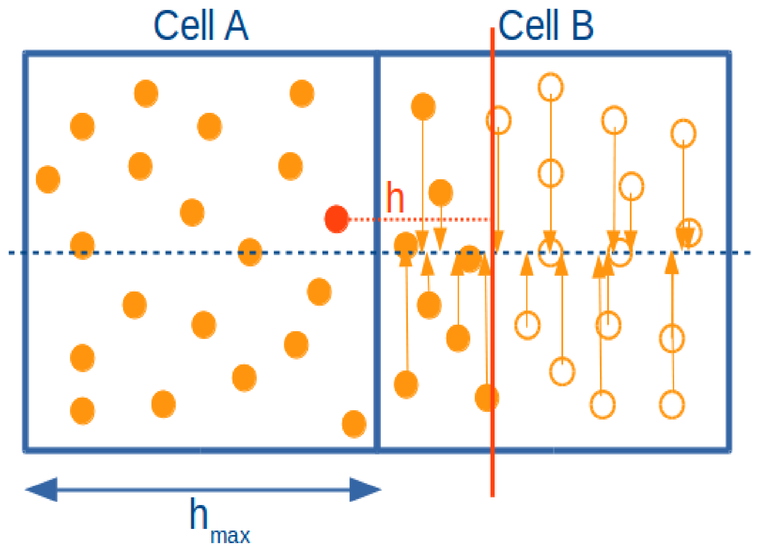


Figure: Sorted particles. Only need to loop over particles that are up to a distance, **h**, on the axis.

SIMD Vectorisation

Vectorisation can improve the performance of an application by making use of a CPUs' vector registers. **SSE, AVX** and **AVX-512** instruction sets utilise these registers to process multiple pieces of data in a single step. Each set can perform concurrent operations on up to: 4,8 and 16 single precision numbers. Thus, the theoretical speed-up obtained from each instruction set is: **4x**, **8x** and **16x** respectively.

In practice the code is sped up by vectorising the most time consuming loops in the code. This can be achieved in one of two ways:

- **Auto-vectorisation**: the compiler automatically vectorises loops based upon a set of criteria
- **Explicit Vectorisation**: the code is modified by the developer to use a set of compiler intrinsics and data types

We wanted to keep the code readable and easy to maintain, so we opted to allow the compiler to vectorise the code for us rather than modifying the code to use intrinsics.

Vectorisation of SWIFT

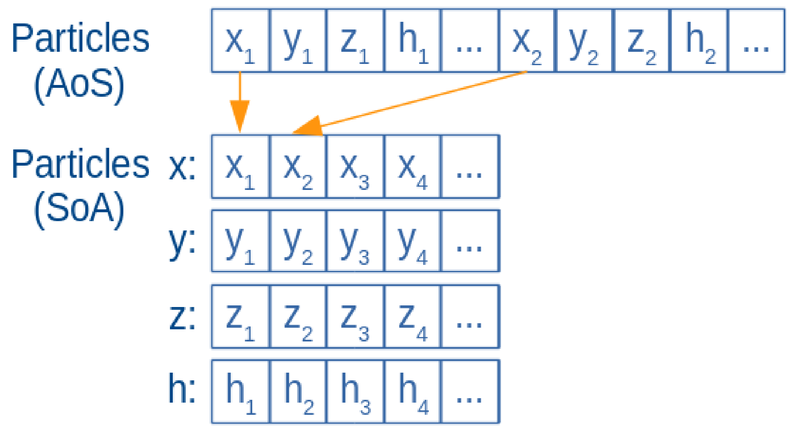
In order to vectorise the code there were a number of changes that needed to be made:

- Remove vector dependencies between variables
- Remove multiple exit conditions from loops
- Careful placement of conditional statements

Once the code was vectorised a benchmark was created to both measure the accuracy and performance gain. After analysing the first benchmark results it became clear that extensive optimisations were needed to achieve the theoretical maximum speed-up.

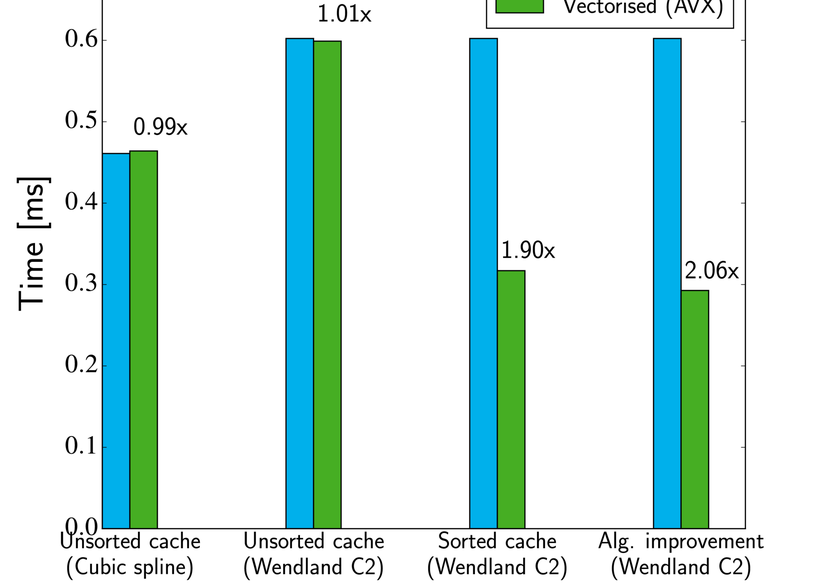
The following optimisations were performed:

- Place particles into a cache formed from a **SoA** (**S**tructure **o**f **A**rrays)
- Sort the cache
- Improve memory access pattern by using a branchless smoothing kernel (Wendland C2)
- Vectorise the loop that locates the exit condition
- Improve the algorithm



Results

The results displayed below show the speed-up gained from each optimised version of the code.



Conclusion

The maximum speed-up obtained from vectorisation (AVX) was ~2x, which is well below the theoretical maximum performance. This gap in performance is due to a number of reasons:

- High vector register pressure
- Partially filled vectors due to low number of interactions between particles

For these reasons we have recently begun to vectorise the code using intrinsics, which give us more control over how the code is executed on the CPU and reduces our reliance on the compiler.

Website
http://icc.dur.ac.uk/swift/

Papers
https://arxiv.org/abs/1606.02738
https://arxiv.org/abs/1601.05384