CASE STUDENTSHIP AT DURHAM UNIVERSITY: USING NEXT-GENERATION COMPUTER TECHNOLOGY TO CREATE A VIRTUAL UNIVERSE

Main Supervisor:	Prof. Richard Bower	r.g.bower@durham.ac.uk (ICC)
Office:	Ogden Centre West 216	
2nd Supervisor:	Dr. Robert Maskell	robert.maskell@intel.com (Intel Corp.)
Funding:	EPSRC CASE studentship	

To apply e-mail r.g.bower@durham.ac.uk

Project Description:

Recently the world-renowned computer simulation group at Durham's Institute for Computational Cosmology (ICC) has begun a close collaboration with the Intel computer and processor manufacturer. The aim of the collaboration is to demonstrate novel computing technologies using cosmological computer simulations as a test-bed. We are now pleased to be able to offer a PhD CASE studentship, jointly supervised by Robert Maskell, Intel's director of research. Hosted at the ICC, this is primarily a Computer Science PhD and will focus on computational aspects of cosmological simulations using the next-generation SWIFT computer code.

The next steps in cosmological modelling, simulating larger volumes at greater detail, require an order of magnitude increase in processing power or simulation efficiency. The SWIFT simulation code combines major improvements in algorithms (e.g. the fast multipole method and multiscale time-step hierarchy), highly scalable parallelisation (fine-grained task-based parallelism within nodes and asynchronous MPI communications between them) and SIMD vectorisation. The code delivers a factor ∼20x speed-up over current competing codes through our implementation of task-based parallelism and novel algorithms.

However, as the speed of the simulation rises, the bottleneck is quickly becoming the time it takes to output the simulation data and process it. Our solution to this "big data" problem is twofold: first, instead of writing a complete snapshot at fixed intervals, we write only the changing quantities of interest for particles whenever they experience sufficient change; and secondly, we stream this data into a continuous and incremental particle log on each node's local storage in parallel with the rest of the computation, thus avoiding IO latencies. In this way, output is generated piecewise, adapting to the speed of each particle's movement, and the output is committed to disk in the background during the computation without (a) stopping for I/O and (b) overloading the I/O sub-system.

The second strand to the solution is to minimise the data that is output and stored for long periods on conventional hard drives and to integrate post-processing tools within the simulation. Since most of the data products required for scientific analysis (e.g. the rate at which stars form in a galaxy or their masses) only evolve slowly over the course of a simulation this is an ideal oportunity to take advantage of new memory technologies. Instead of storing the ever-growing amount of raw data and painfully post-processing it, in-flight analysis enables us to only permanently store the particle log data for scientifically interesting regions, dramatically reducing the disk-space footprint of the simulation. Our solution will be based on Intels 3D xPoint technology.

The benefits of these approaches have important applications beyond the cosmological computer simulations that the ICC undertakes, and the PhD student will work closely with Intel to champion this approach across a wide range of scientific disciplines.

A few useful links are given below, the attached poster summarises some of the innovations in SWIFT:
- The SWIFT cosmological simulation code webpage: `www.swiftsim.com`
- The SWIFT cosmological simulation code repository: `https://gitlab.cosma.dur.ac.uk/swift/swiftsim`
- A presentation summarizing the state of SWIFT: `http://www.intel.com/content/www/us/en/events/hpcdevcon/parallel-programming-track.html#swift`
- Schaller M. et al., 2016, 'SWIFT: Using task-based parallelism, fully asynchronous communication, and graph partition-based domain decomposition for strong scaling on more than 100,000 cores, Proceedings of the PASC Conference, Lausanne, Switzerland (`https://arxiv.org/abs/1606.02738`)
- Gonnet P., Chalk A., Schaller M., 2016, 'QuickSched: Task-based parallelism with dependencies and conflicts (`https://arxiv.org/abs/1601.05384`)

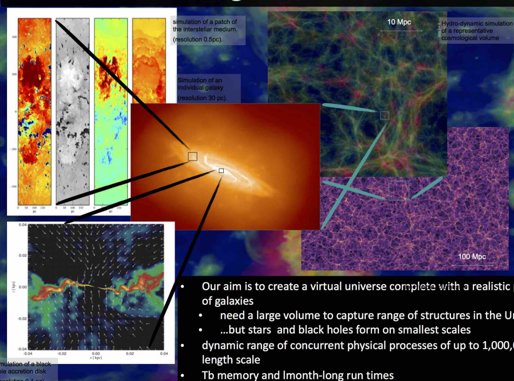
SWIFT is a novel code tackling grand-challenge problems in cosmology, such as the formation of galaxies.
It is being built as a demonstrator for the **QuickSched** fine-grained task parallel library.
QuickSched is open source and can be applied across a wide range of science domains.

Intel Parallel Computing Centre,
Institute for Computational Cosmology,
Durham University, UK

Simulating the Universe with SWIFT and QuickSched

swiftsim.com

SWIFT a hydrodynamics code for re-creating the Universe



- Our aim is to create a virtual universe complete with a realistic population of galaxies
  - need a large volume to capture range of structures in the Universe
  - …but stars and black holes form on smallest scales
- dynamic range of concurrent physical processes of up to 1,000,000 in length scale
- Tb memory and lmonth-long run times
  - demands very effective strong scaling
- these are extreme, but not unique to astrophysics
  - astrophysics is good open-access demonstrator for new technologies
  - close affinities with Molecular dynamics

QuickSched: SIMD Parallelism

```
for (pjd = 0; pjd < count_j - (count_j % VEC_SIZE) && sort_j[pjd].d < di;
     pjd += VEC_SIZE) {
  vector vf_pjx[3], vf_dx[3];
  for (k = 0; k < 3; k++) vf_pjx[k].v = vec_load(&parts_j_xs[k][pjd]);
  for (k = 0; k < 3; k++) vf_dx[k].v = vf_pix[k].v - vf_pjx[k].v;

  vector vf_r2;
  vf_r2.v = vf_dx[0].v * vf_dx[0].v;
  for (k = 1; k < 3; k++) vf_r2.v += vf_dx[k].v * vf_dx[k].v;

  vector vf_mask;
  vf_mask.v = vec_cmp_lt(vf_r2.v, vf_hig2.v);
  int mask = vec_cmp_result(vf_mask.v);
```

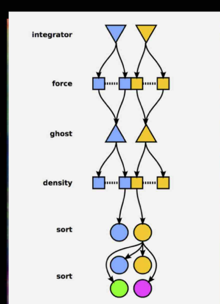
Each QuickSched task is optimised to use SIMD parallelism.

- task size is chosen to allow all data to fit into L2
- Choice of data structure is critical. We repack data into vector friendly structures as they are sorted
- We streamline vector calculations by masking unwanted calculations rather than complicating vector loops
- Current speed up is factor 2.5

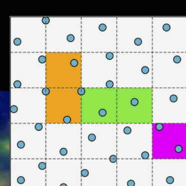
QuickSched: Fine-Grained Task Parallelism

SWIFT uses a modern approach to parallel computing:

- Break work into simple pieces
- Create graph showing links between tasks
  - What must be done first ? (i.e. '*dependencies*')
  - What cannot be done at the same time as something else? (i.e. '*conflicts*')
  - Assign a priority
- Implies a re-design of the algorithm but gives a nice and clean picture.
- Each thread picks up a task and executes it.
- When all tasks are done, the calculation ends.

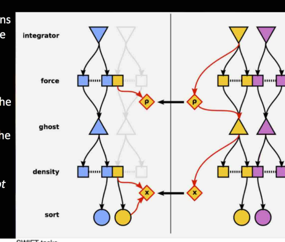


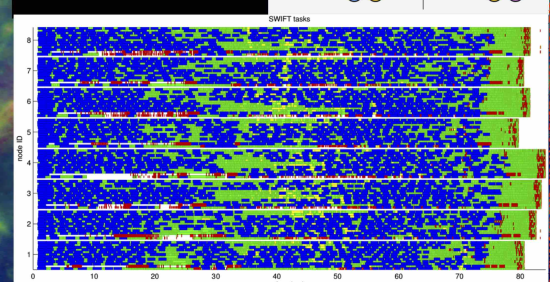
- A task is a series of operations acting on a (small) chunk of memory.
  - Tasks operating on the same memory *conflict*.
  - Tasks can *depend* on each others.
  - For Smooth Particle Hydrodynamics, we will have tasks spanning one cell or over two cells.
- All the (millions of) tasks are then put in a queue and our scheduler (QuickSched) assigns them to the different threads.



QuickSched: Distributed Memory Parallelism with asynchronous MPI

- In SWIFT, the domain decomposition happens along the cell edges, i.e. the particle cells are individual resources.
- We have to copy the particle data twice:
  - Once to send the particle positions for the density computation,
  - Once to send the particle densities for the force computation.
- Two send/recv tasks per border cell, i.e. *a lot* of communication tasks.





- Most experienced MPI users will advise against creating so many send/recv tasks.
- Most usual analysis and benchmark tools do not support our approach ! → Only Intel Trace Collector does.
- This has been tested on up to 8192 nodes with 32 threads each, i.e. 262144 threads
- Since all communication is asynchronous, we don't really care about latencies.
- Spreading the communication throughout the computation actually reduces load on the network.