SWIFT – SCALING ON NEXT GENERATION ARCHITECTURES

James Willis, Matthieu Schaller, Richard Bower, Pedro Gonnet &
SWIFT Team

Durham University, ICC

Sixth Annual DiRAC Science Day
8th September 2016

TEAM

This work is a collaboration between two 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) and the DiRAC software team.

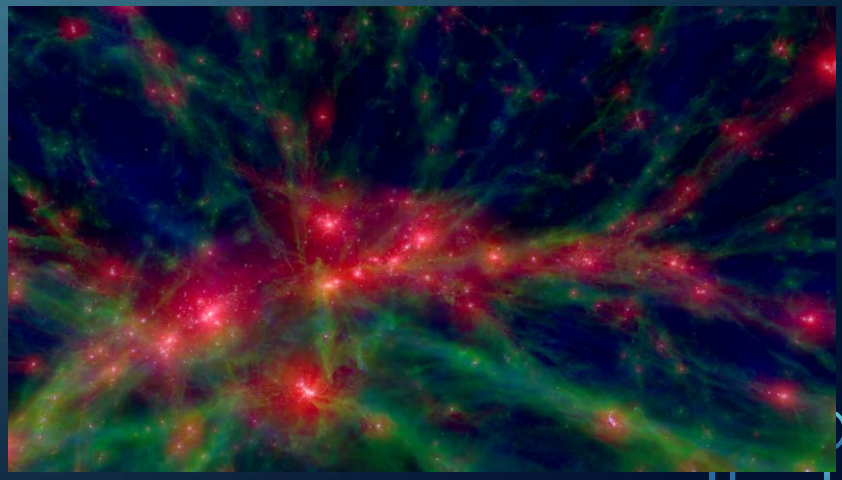
- This research is partly funded by an Intel IPCC since March 2015.

OVERVIEW

- Motivation behind SWIFT
- Problem that we need to solve
- SWIFT's solution to problem
- New architectures (e.g. KNL)
- Challenges faced on KNL
- Scaling results from KNL
- Conclusion

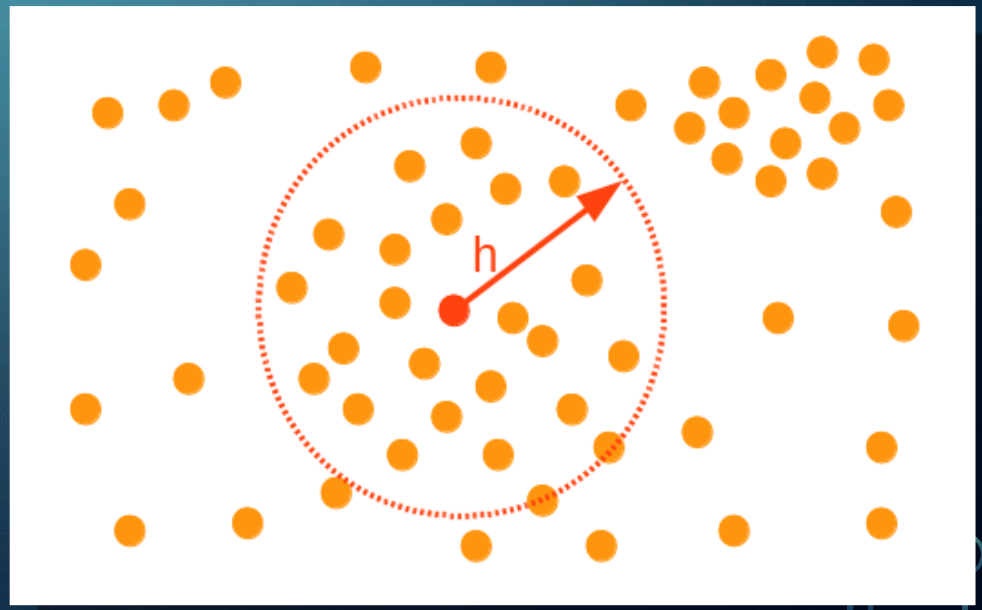
MOTIVATION BEHIND SWIFT

- Create simulations of the formation and evolution of the Universe
- Update 10^9 particles using hydrodynamical and gravitational forces
- Simulate physical processes:
  - Cooling and heating of the gas due to the presence of stars and other emission
  - Formation of stars in cold and dense regions
  - Explosion of supernovae with injection of their energy in the surrounding gas
  - Formation of supermassive black holes



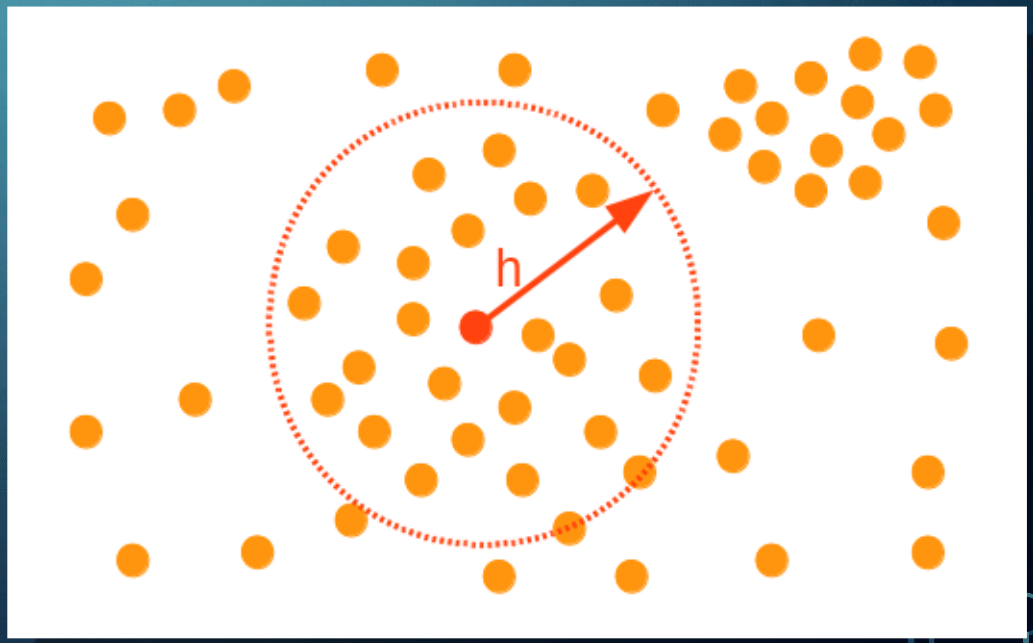
PROBLEM TO SOLVE

- We update each particle using SPH (Smoothed-Particle Hydrodynamics)
- Each particle interacts with its neighbours that are within a smoothing length, h
- The smoothing length varies depending on the particle density of the region



CHALLENGES

- Particles move over time and so to will their neighbour lists
- An interaction between two particles is computationally cheap to carry out (low FLOPs)
- Domain is unstructured leading to large particle density variations
- Domain is constantly evolving

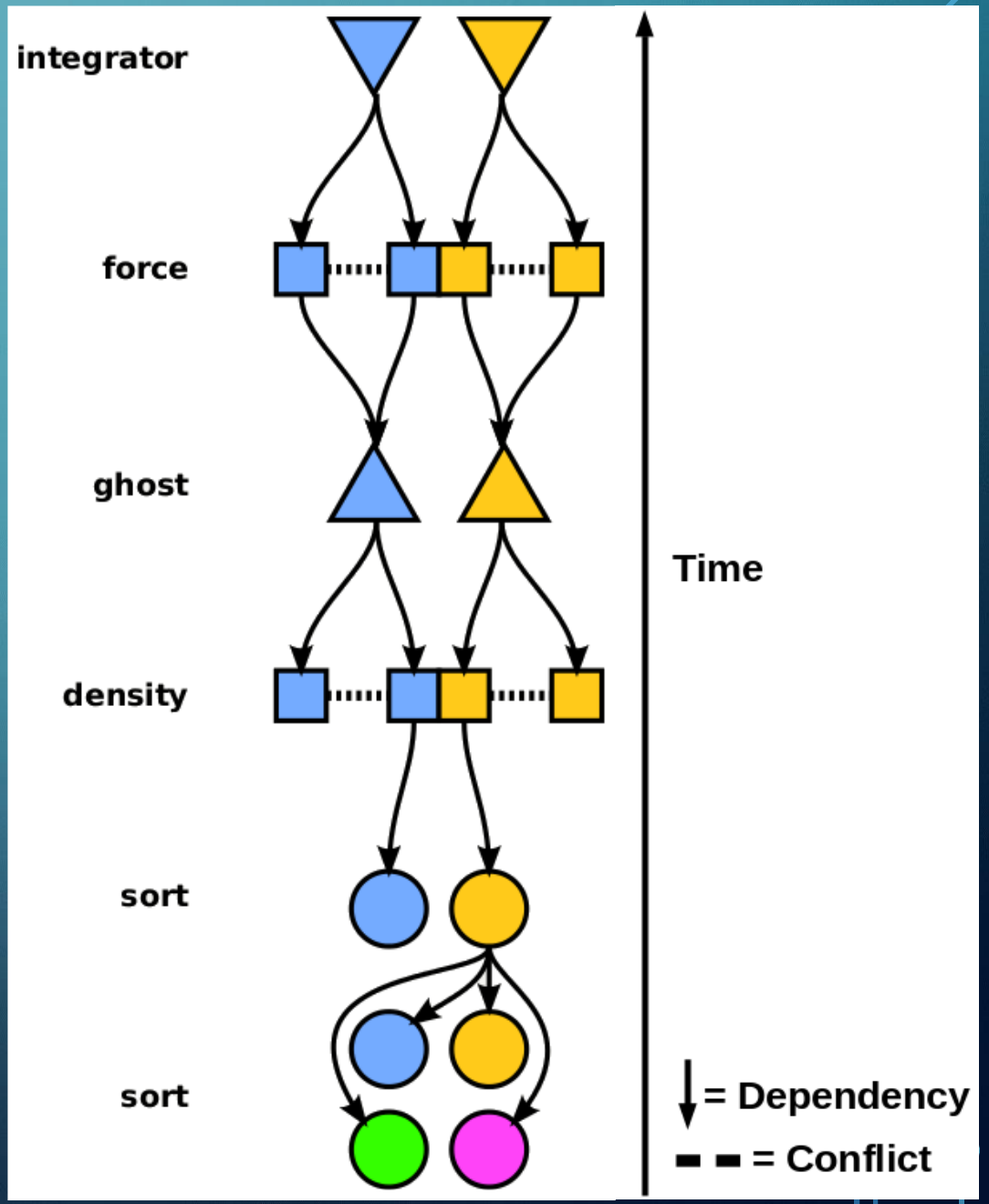


TASK BASED PARALLELISM

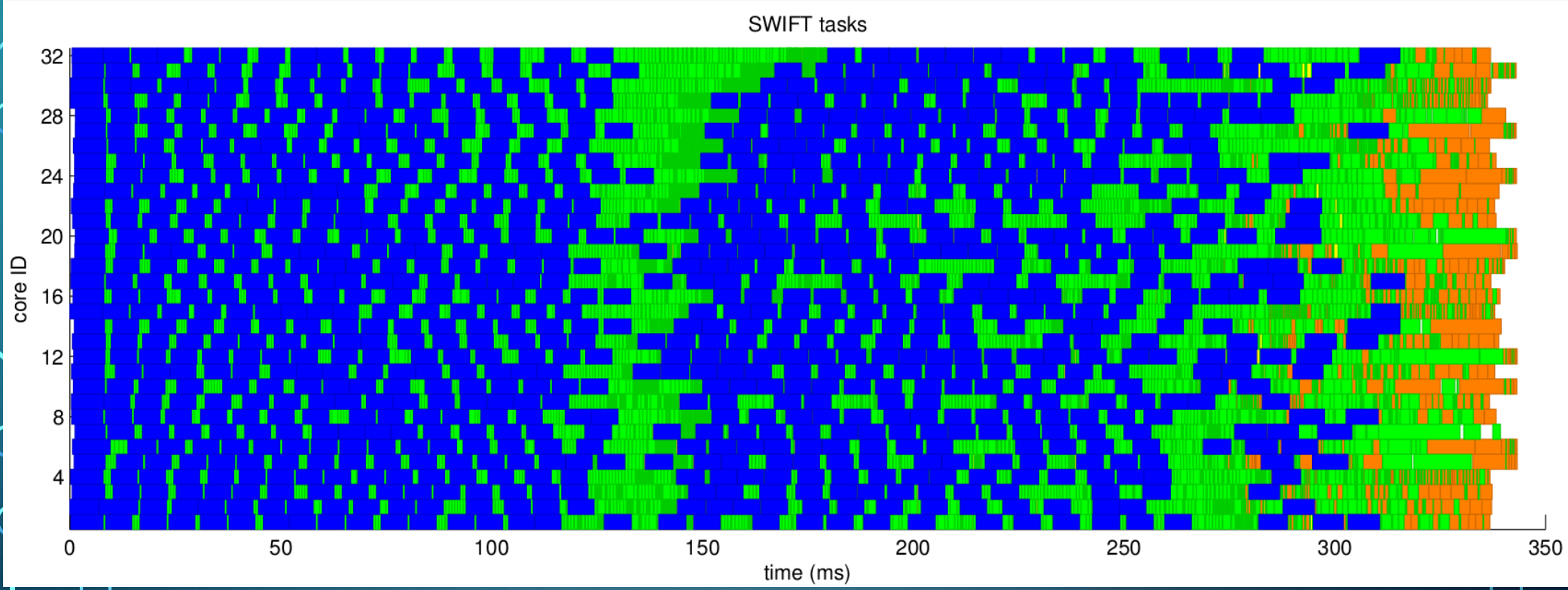
- Shared-memory parallel programming paradigm:
  - Fine-grained tasking
  - Data locality
  - Asynchronous MPI
  - Abstracts the parallelisation completely away from the physics
- Avoids most problems associated with concurrency and load-balancing
- Implemented by our own Open-source library QuickSched (arXiv:1601.05384)

TASK BASED PARALLELISM FOR SPH

- Decomposes 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



TASK BASED PARALLELISM

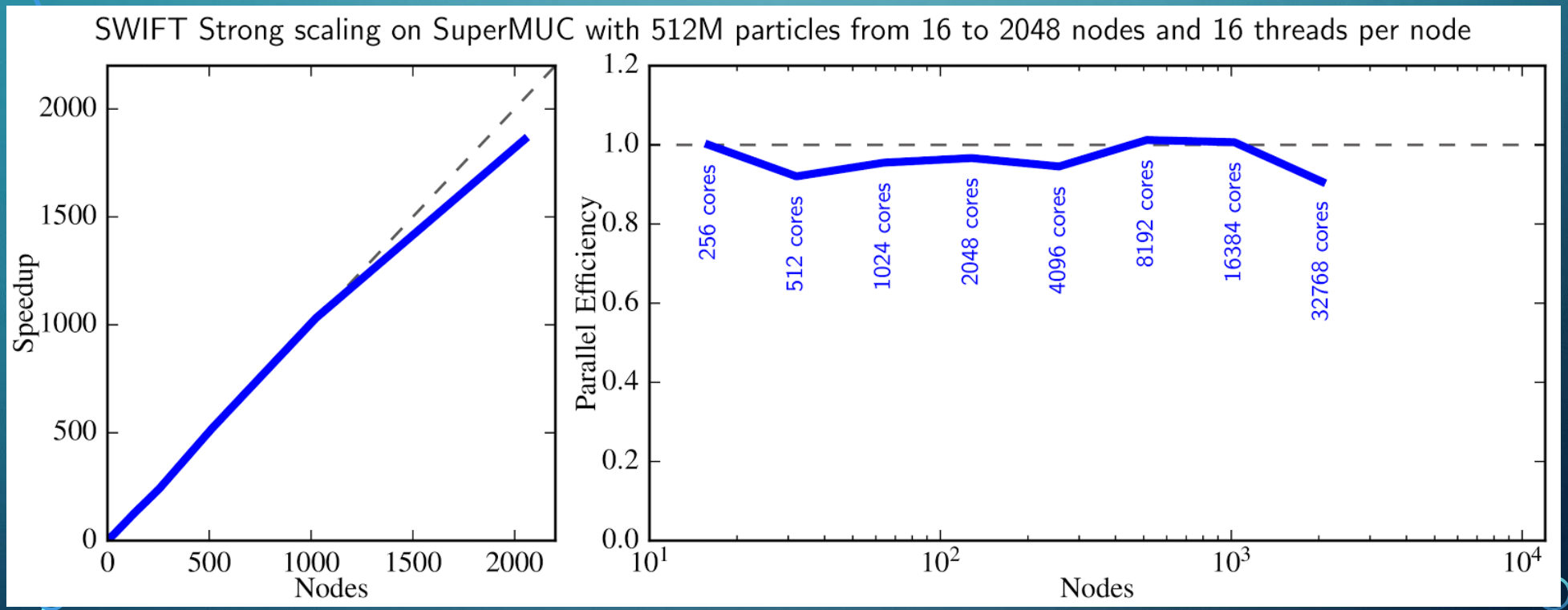


Task graph for one time-step. Orange bars are integration tasks.

Blue and green are particle interaction tasks.

Almost perfect load-balancing is achieved on 32 cores.

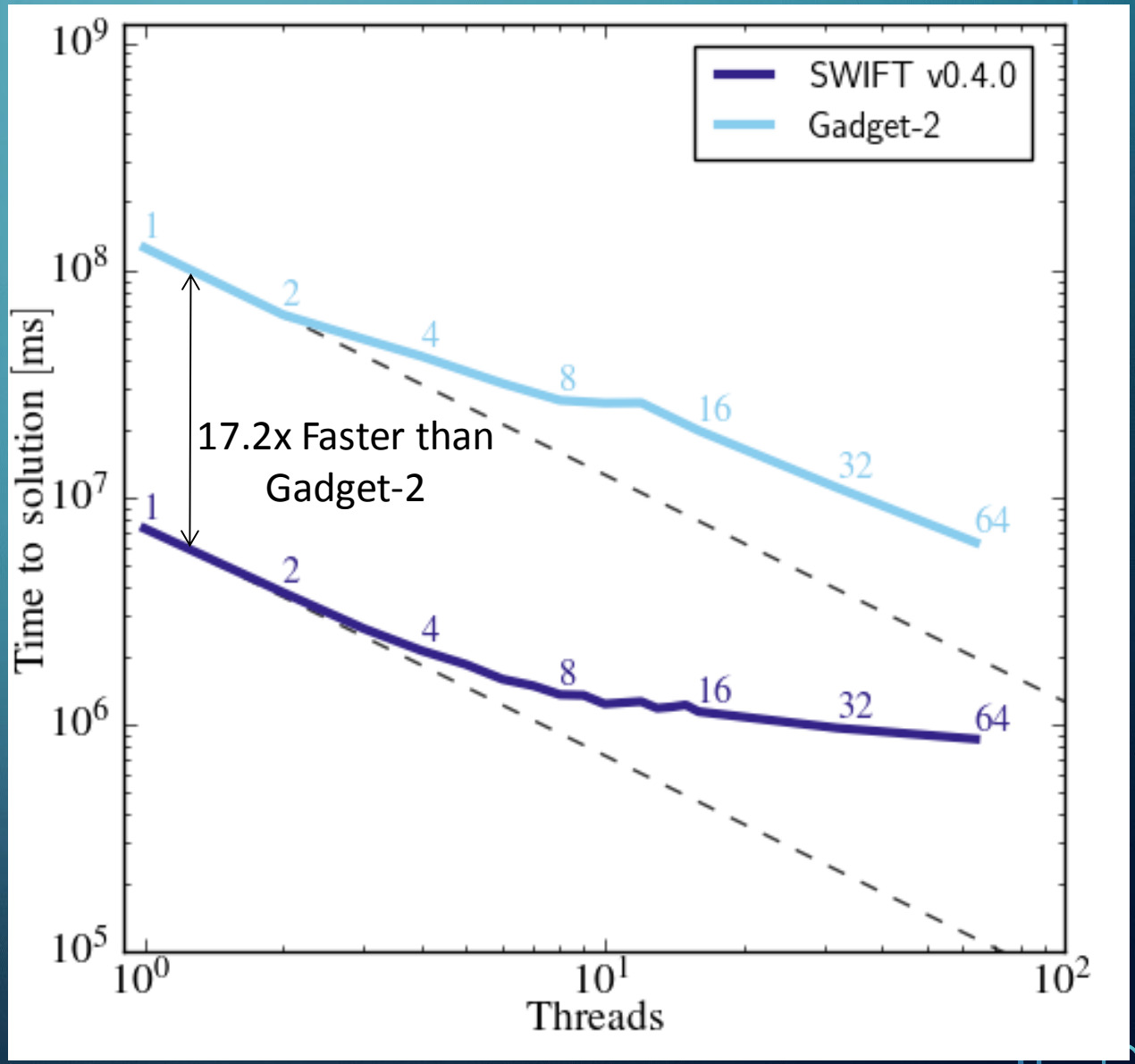
SUPERMUC SCALING



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

SWIFT VS GADGET-2

- On one core SWIFT is ~17.2x faster than Gadget-2
- SWIFT on one core is as fast as Gadget-2 on 64
- Same physics is used with the same level of accuracy



NEXT GENERATION ARCHITECTURES

- CPU clock rates peaking
- Number of cores per chip increasing
- Intel Xeon Phi Knights Landing 64-72 cores
- Lower clock rates ≈1.3GHz
- Intel Xeon Broadwell 22 cores
- AMD Zen 32 cores

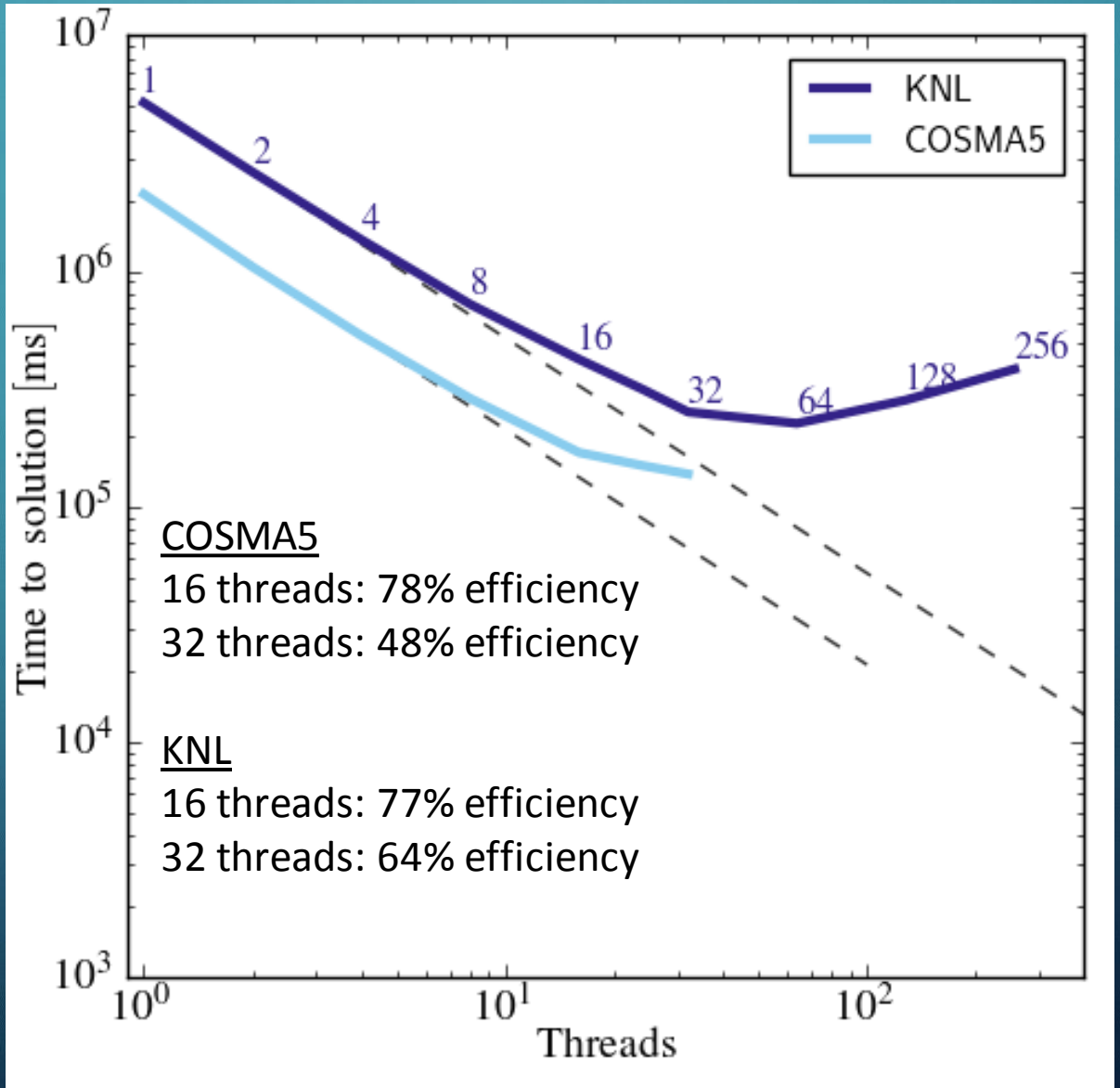
CHALLENGES

- Applications must improve their strong scaling in order to take advantage of this new technology
- Lower clock rates
- Make use of 16-wide vector units to improve performance even further
- Effective ways to utilise MPI

KNL SCALING

- Intel Xeon Phi CPU 7250 @ 1.30GHz
- 64 cores, 4-way Hyper Threading
- Supports up to 256 threads with HT turned on
- 16 GB MCDRAM

KNL SCALING



Note: The KNL's MCDRAM was operating in *cache* mode

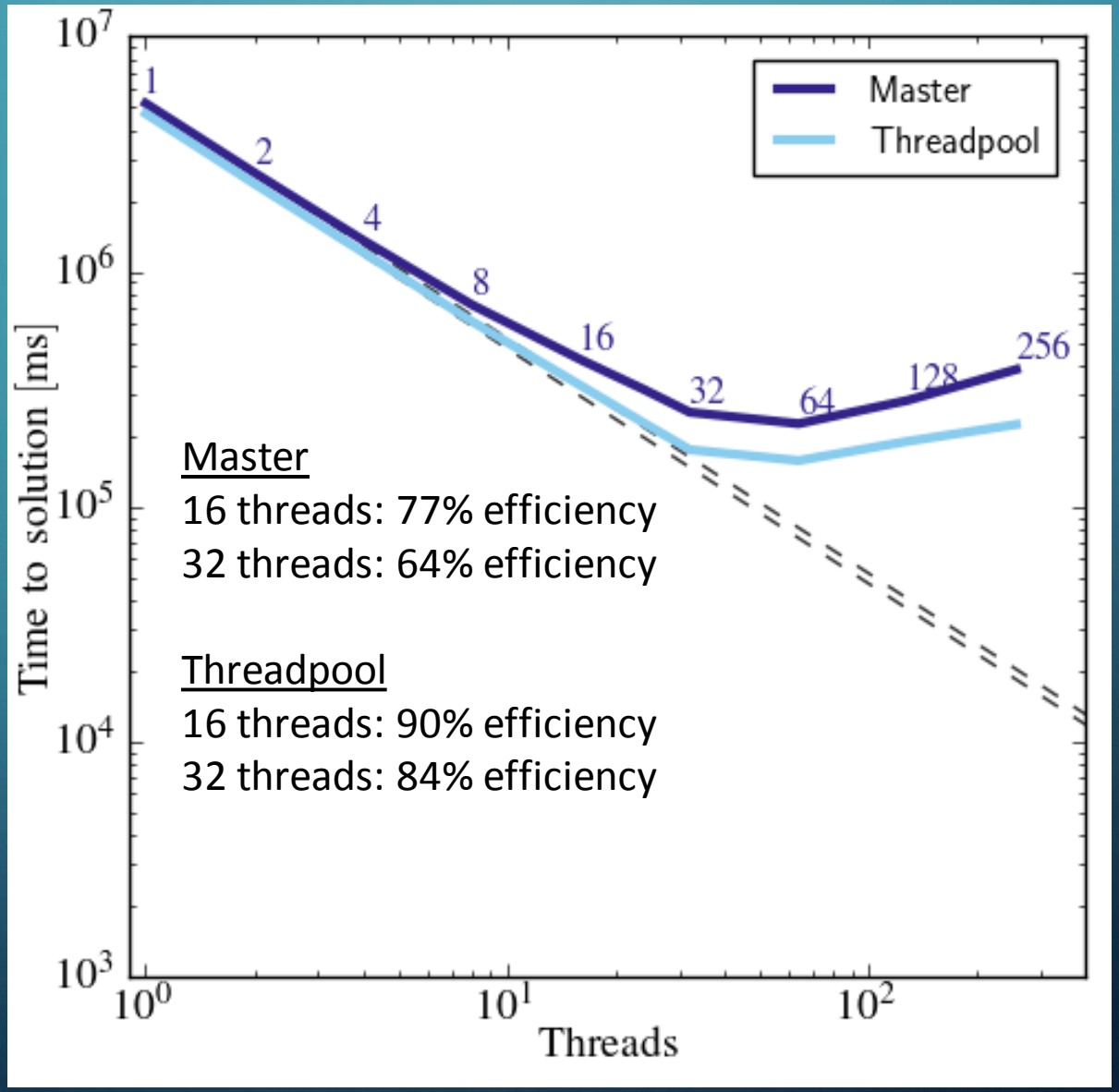
SWIFT THREADPOOL

- Poor scaling on KNL
- Profiled code:
  - Physics perfectly load balanced, scales well
  - Bottlenecks down to scheduler maintenance that is run serially in between time steps

<u>Solution</u>

- Parallelise serial code using a pool of threads
- Each thread is assigned a job to perform
- Created using Pthreads
- Similar to a lightweight version of OpenMP

KNL SCALING



Note: The KNL's MCDRAM was operating in *cache* mode

VECTORISATION

- Performed auto-vectorisation on SWIFT
- Wanted to make it easier for scientists to vectorise code without using intrinsics
- Vectorised with ICC, but we need the same performance with GCC
- Now looking at explicit vectorisation using intrinsics to get better performance

<u>Further Work</u>

- AVX-512 instruction set
- 16-wide vector units
- Masking operations

CONCLUSIONS

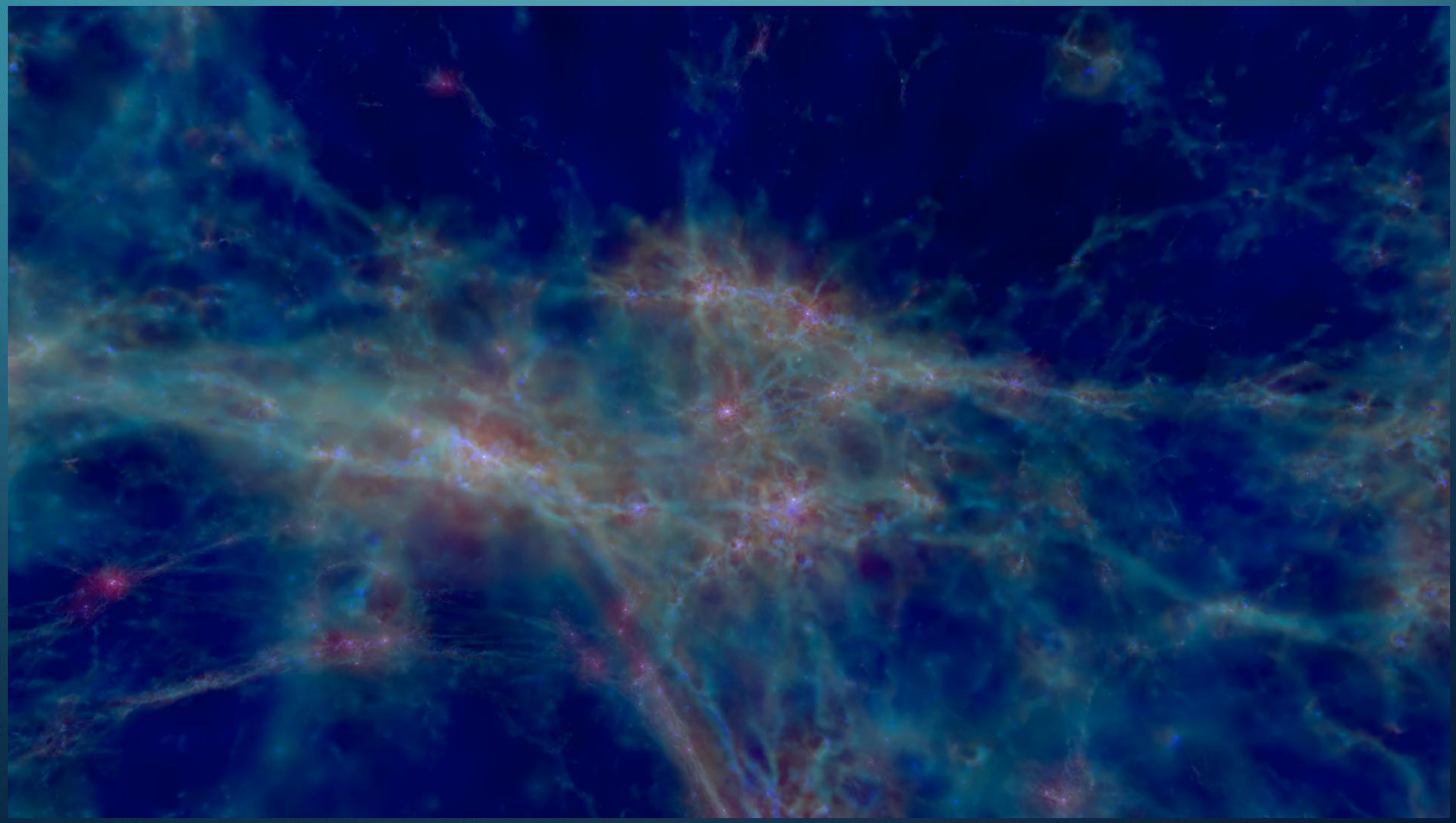
- Improved SWIFT scaling on a many-core system
- Reduced time to solution
- Implemented a lightweight threadpool that parallelises serial jobs
- Physics scales very well up to 100,000 cores
- ~17.2x faster than Gadget-2

<u>Future Work</u>

- Improve efficiency by parallelising serial parts:
  - IO
  - Tree construction
  - Scheduler

QUESTIONS

- Thank you for your attention
- Any questions?
- Website: www.icc.dur.ac.uk/swift/



VECTORISATION CHALLENGES

- SWIFT has high vector register pressure
- SWIFT has low FLOPs
- Opposite to QCD and LAPACK code which have low register pressure and high FLOPs