SWIFT: Fast algorithms for SPH on multi-core architectures

Fast neighbour-finding and task-based parallelism

Pedro Gonnet, Matthieu Schaller, Tom Theuns, Aidan Chalk
ECS/ICC, Durham University
8th International SPHERIC Workshop, June 5th, 2013

Take-home messages

What this talk is all about

Durham University

Take-home messages

What this talk is all about

Durham University

- In order to continue getting *faster*, programs need to become *more parallel*.

Take-home messages

What this talk is all about

Durham University

- In order to continue getting *faster*, programs need to become *more parallel*.

Take-home messages

What this talk is all about

Durham University

- In order to continue getting *faster*, programs need to become *more parallel*.
  $\longrightarrow$ If a program has reached its maximum degree of parallelism, it won't get any faster.

Take-home messages

What this talk is all about

Durham University

- In order to continue getting *faster*, programs need to become *more parallel*.
  $\longrightarrow$ If a program has reached its maximum degree of parallelism, it won't get any faster. Ever.

Take-home messages

What this talk is all about

Durham University

- In order to continue getting *faster*, programs need to become *more parallel*.
  $\longrightarrow$ If a program has reached its maximum degree of parallelism, it won't get any faster. Ever.
- On shared-memory systems, asynchronous task-based parallelism solves most problems with concurrency and scaling.

Take-home messages

What this talk is all about

Durham University

- In order to continue getting *faster*, programs need to become *more parallel*.
  $\longrightarrow$ If a program has reached its maximum degree of parallelism, it won't get any faster. Ever.
- On shared-memory systems, asynchronous task-based parallelism solves most problems with concurrency and scaling.
  $\longrightarrow$ But we still need to develop task-based algorithms for specific problems, e.g. SPH simulations.

Take-home messages

What this talk is all about

Durham University

- In order to continue getting *faster*, programs need to become *more parallel*.
  $\longrightarrow$ If a program has reached its maximum degree of parallelism, it won't get any faster. Ever.
- On shared-memory systems, asynchronous task-based parallelism solves most problems with concurrency and scaling.
  $\longrightarrow$ But we still need to develop task-based algorithms for specific problems, e.g. SPH simulations.
- Better algorithms alone can lead to speedups of up to a factor of ten.

Take-home messages

What this talk is all about

Durham University

- In order to continue getting *faster*, programs need to become *more parallel*.
  $\longrightarrow$ If a program has reached its maximum degree of parallelism, it won't get any faster. Ever.
- On shared-memory systems, asynchronous task-based parallelism solves most problems with concurrency and scaling.
  $\longrightarrow$ But we still need to develop task-based algorithms for specific problems, e.g. SPH simulations.
- Better algorithms alone can lead to speedups of up to a factor of ten.
  $\longrightarrow$ Better use of both existing and future infrastructure.

Take-home messages

What this talk is all about

Durham University

- In order to continue getting *faster*, programs need to become *more parallel*.
  $\longrightarrow$ If a program has reached its maximum degree of parallelism, it won't get any faster. Ever.
- On shared-memory systems, asynchronous task-based parallelism solves most problems with concurrency and scaling.
  $\longrightarrow$ But we still need to develop task-based algorithms for specific problems, e.g. SPH simulations.
- Better algorithms alone can lead to speedups of up to a factor of ten.
  $\longrightarrow$ Better use of both existing and future infrastructure.

Take-home messages

Case in point

Take-home messages

Case in point

- Cosmological simulation (galaxy formation) with 1.8 M particles in a cubic box of 6.25 Mpc on a 4 $\times$ Intel Xeon X7550 with 32 cores, 2 GHz.

Take-home messages

Case in point

- Cosmological simulation (galaxy formation) with 1.8 M particles in a cubic box of 6.25 Mpc on a 4 $\times$ Intel Xeon X7550 with 32 cores, 2 GHz.

Take-home messages

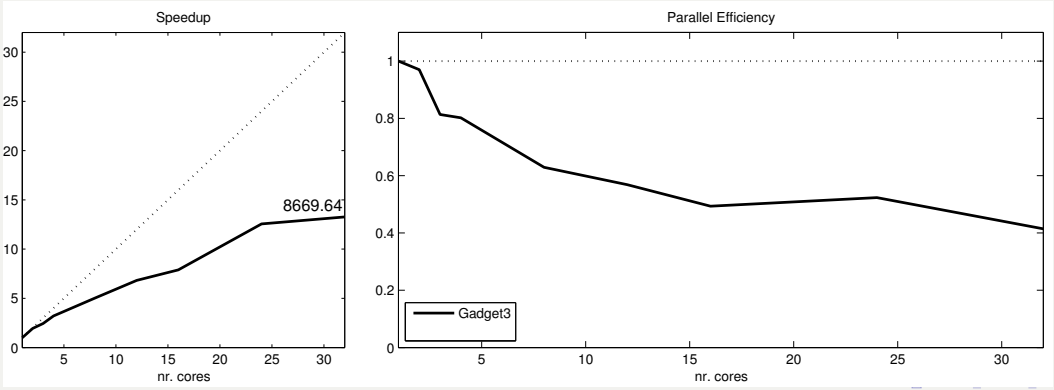
Case in point

- Cosmological simulation (galaxy formation) with 1.8 M particles in a cubic box of 6.25 Mpc on a 4 $\times$ Intel Xeon X7550 with 32 cores, 2 GHz.

Take-home messages

Case in point

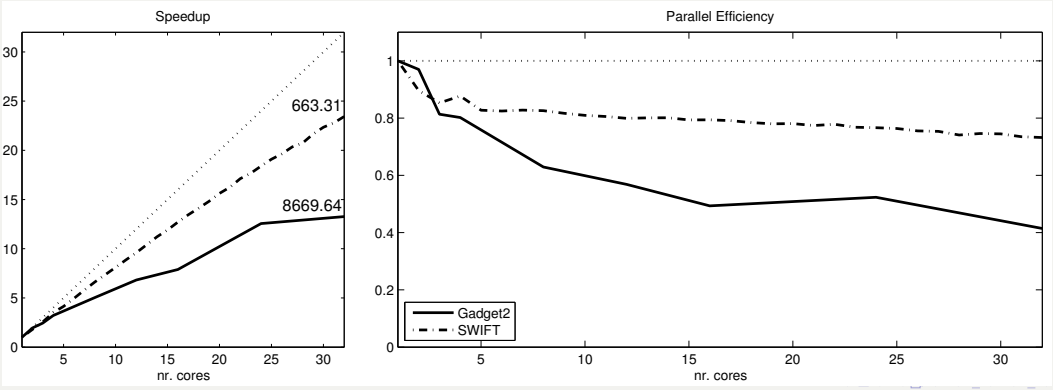
- Cosmological simulation (galaxy formation) with 1.8 M particles in a cubic box of 6.25 Mpc on a 4 $\times$ Intel Xeon X7550 with 32 cores, 2 GHz.
- GADGET-3, MPI-based, used for multi-billion particle simulations.



Take-home messages

Case in point

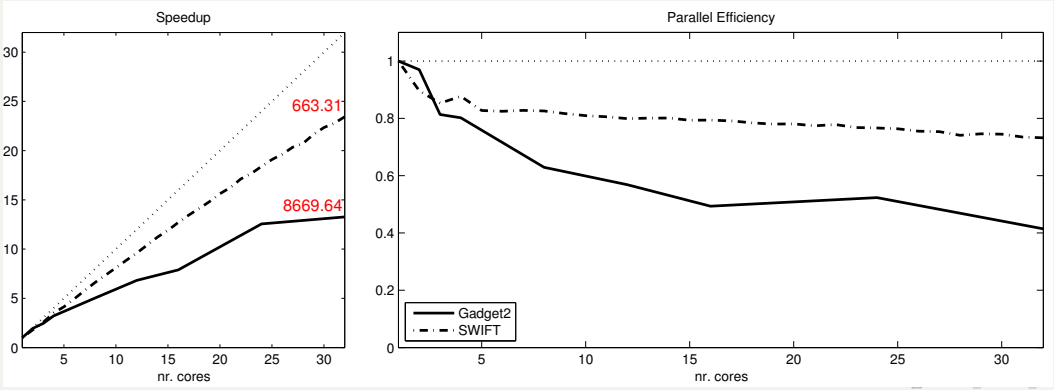
- Cosmological simulation (galaxy formation) with 1.8 M particles in a cubic box of 6.25 Mpc on a 4 $\times$ Intel Xeon X7550 with 32 cores, 2 GHz.
- GADGET-3, MPI-based, used for multi-billion particle simulations.
- SWIFT, our own code for the same problem.



Take-home messages

Case in point

- Cosmological simulation (galaxy formation) with 1.8 M particles in a cubic box of 6.25 Mpc on a 4 $\times$ Intel Xeon X7550 with 32 cores, 2 GHz.
- GADGET-3, MPI-based, used for multi-billion particle simulations.
- SWIFT, our own code for the same problem. 13$\times$ faster on 32 cores.



Task-based parallelism

Main idea

Task-based parallelism

Main idea

- Shared-memory parallel programming paradigm in which the computation is formulated in an implicitly parallelizable way that automatically avoids most of the problems associated with concurrency and load-balancing.

Task-based parallelism

Main idea

- Shared-memory parallel programming paradigm in which the computation is formulated in an implicitly parallelizable way that automatically avoids most of the problems associated with concurrency and load-balancing.

Task-based parallelism

Main idea

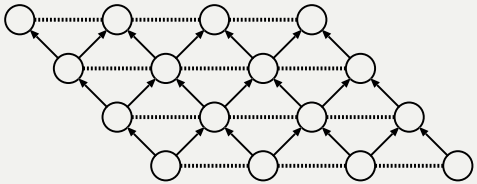
Durham University

- Shared-memory parallel programming paradigm in which the computation is formulated in an implicitly parallelizable way that automatically avoids most of the problems associated with concurrency and load-balancing.

Task-based parallelism

Main idea

- Shared-memory parallel programming paradigm in which the computation is formulated in an implicitly parallelizable way that automatically avoids most of the problems associated with concurrency and load-balancing.
- We first reduce the problem to a set of inter-dependent tasks.

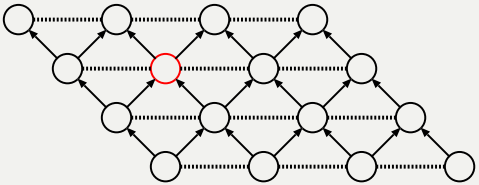


Task-based parallelism

Main idea

Durham University

- Shared-memory parallel programming paradigm in which the computation is formulated in an implicitly parallelizable way that automatically avoids most of the problems associated with concurrency and load-balancing.
- We first reduce the problem to a set of inter-dependent tasks.
- For each task, we need to know:

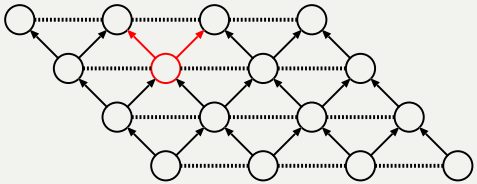


Task-based parallelism

Main idea

Durham University

- Shared-memory parallel programming paradigm in which the computation is formulated in an implicitly parallelizable way that automatically avoids most of the problems associated with concurrency and load-balancing.
- We first reduce the problem to a set of inter-dependent tasks.
- For each task, we need to know:
  - Which tasks it depends on,

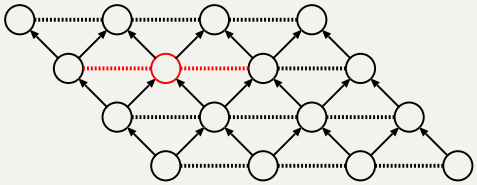


Task-based parallelism

Main idea

Durham University

- Shared-memory parallel programming paradigm in which the computation is formulated in an implicitly parallelizable way that automatically avoids most of the problems associated with concurrency and load-balancing.
- We first reduce the problem to a set of inter-dependent tasks.
- For each task, we need to know:
  - Which tasks it depends on,
  - Which tasks it conflicts with.

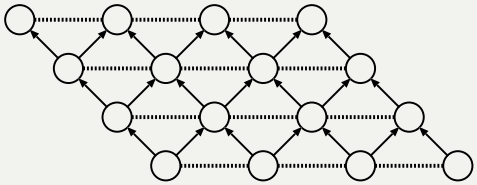


Task-based parallelism

Main idea

Durham University

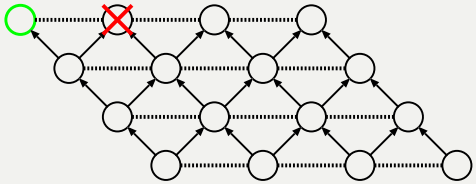
- Shared-memory parallel programming paradigm in which the computation is formulated in an implicitly parallelizable way that automatically avoids most of the problems associated with concurrency and load-balancing.
- We first reduce the problem to a set of inter-dependent tasks.
- For each task, we need to know:
  - Which tasks it depends on,
  - Which tasks it conflicts with.
- Each thread then picks up a task which has no unresolved dependencies or conflicts and computes it.



Task-based parallelism

Main idea

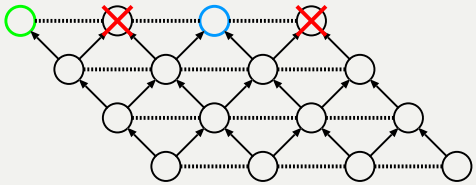
- Shared-memory parallel programming paradigm in which the computation is formulated in an implicitly parallelizable way that automatically avoids most of the problems associated with concurrency and load-balancing.
- We first reduce the problem to a set of inter-dependent tasks.
- For each task, we need to know:
  - Which tasks it depends on,
  - Which tasks it conflicts with.
- Each thread then picks up a task which has no unresolved dependencies or conflicts and computes it.



Task-based parallelism

Main idea

- Shared-memory parallel programming paradigm in which the computation is formulated in an implicitly parallelizable way that automatically avoids most of the problems associated with concurrency and load-balancing.
- We first reduce the problem to a set of inter-dependent tasks.
- For each task, we need to know:
  - Which tasks it depends on,
  - Which tasks it conflicts with.
- Each thread then picks up a task which has no unresolved dependencies or conflicts and computes it.

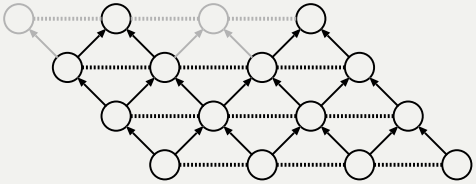


Task-based parallelism

Main idea

Durham University

- Shared-memory parallel programming paradigm in which the computation is formulated in an implicitly parallelizable way that automatically avoids most of the problems associated with concurrency and load-balancing.
- We first reduce the problem to a set of inter-dependent tasks.
- For each task, we need to know:
  - Which tasks it depends on,
  - Which tasks it conflicts with.
- Each thread then picks up a task which has no unresolved dependencies or conflicts and computes it.

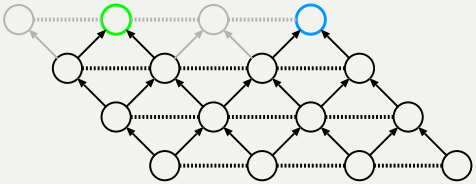


Task-based parallelism

Main idea

Durham University

- Shared-memory parallel programming paradigm in which the computation is formulated in an implicitly parallelizable way that automatically avoids most of the problems associated with concurrency and load-balancing.
- We first reduce the problem to a set of inter-dependent tasks.
- For each task, we need to know:
  - Which tasks it depends on,
  - Which tasks it conflicts with.
- Each thread then picks up a task which has no unresolved dependencies or conflicts and computes it.

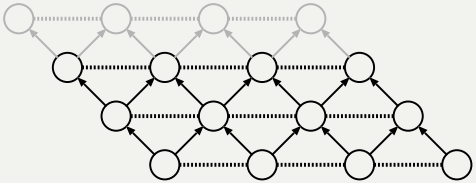


Task-based parallelism

Main idea

Durham University

- Shared-memory parallel programming paradigm in which the computation is formulated in an implicitly parallelizable way that automatically avoids most of the problems associated with concurrency and load-balancing.
- We first reduce the problem to a set of inter-dependent tasks.
- For each task, we need to know:
  - Which tasks it depends on,
  - Which tasks it conflicts with.
- Each thread then picks up a task which has no unresolved dependencies or conflicts and computes it.



Task-based parallelism

Main advantages

Task-based parallelism
Main advantages

- The order in which the tasks are processed is dynamic and adapts automatically to load imbalances.

Task-based parallelism

Main advantages

Durham University

- The order in which the tasks are processed is dynamic and adapts automatically to load imbalances.
- If the dependencies and conflicts are specified correctly, we do not have to worry about concurrency at the level of the individual tasks.

Task-based parallelism
Main advantages

- The order in which the tasks are processed is dynamic and adapts automatically to load imbalances.
- If the dependencies and conflicts are specified correctly, we do not have to worry about concurrency at the level of the individual tasks.
  $\longrightarrow$ No need for expensive explicit locking, synchronization, or atomic operations.

Task-based parallelism

Main advantages

- The order in which the tasks are processed is dynamic and adapts automatically to load imbalances.
- If the dependencies and conflicts are specified correctly, we do not have to worry about concurrency at the level of the individual tasks.
  $\longrightarrow$ No need for expensive explicit locking, synchronization, or atomic operations.
- Each task has exclusive access to the data it is working on, thus improving cache efficiency.

Task-based parallelism

Main advantages

Durham University

- The order in which the tasks are processed is dynamic and adapts automatically to load imbalances.
- If the dependencies and conflicts are specified correctly, we do not have to worry about concurrency at the level of the individual tasks.
  $\longrightarrow$ No need for expensive explicit locking, synchronization, or atomic operations.
- Each task has exclusive access to the data it is working on, thus improving cache efficiency.
- The same approach can be applied to more unconventional many-core systems such as GPUs.

Task-based parallelism

Main advantages

Durham University

- The order in which the tasks are processed is dynamic and adapts automatically to load imbalances.
- If the dependencies and conflicts are specified correctly, we do not have to worry about concurrency at the level of the individual tasks.
  $\longrightarrow$ No need for expensive explicit locking, synchronization, or atomic operations.
- Each task has exclusive access to the data it is working on, thus improving cache efficiency.
- The same approach can be applied to more unconventional many-core systems such as GPUs.
- However, this usually means that we have to re-think our entire computation, e.g. redesign it from scratch to make it task-based.

Task-based algorithms for SPH

Neighbour-finding with trees

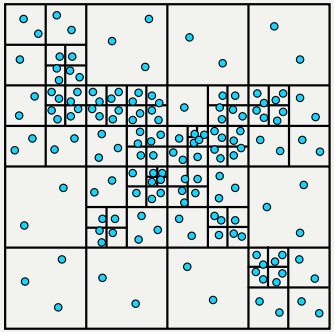
Durham University

Task-based algorithms for SPH

Neighbour-finding with trees

Durham University

- In multi-resolution SPH, Spatial trees are the most commonly used approach to neighbour-finding, as the particle distribution can be irregular.

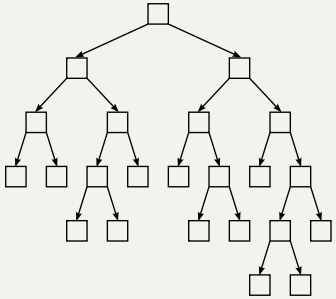


Task-based algorithms for SPH

Neighbour-finding with trees

Durham University

- In multi-resolution SPH, Spatial trees are the most commonly used approach to neighbour-finding, as the particle distribution can be irregular.

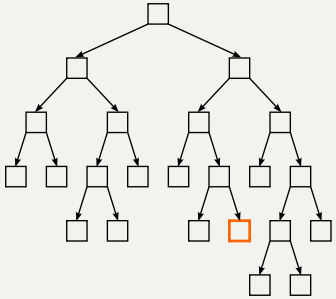


Task-based algorithms for SPH

Neighbour-finding with trees

Durham University

- In multi-resolution SPH, Spatial trees are the most commonly used approach to neighbour-finding, as the particle distribution can be irregular.
- Neighbour-finding up and down the tree is simple, but has some problems:

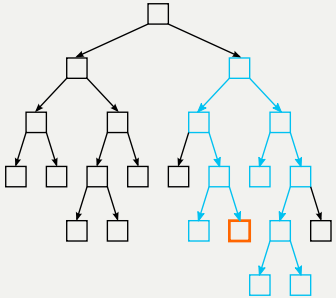


Task-based algorithms for SPH

Neighbour-finding with trees

Durham University

- In multi-resolution SPH, Spatial trees are the most commonly used approach to neighbour-finding, as the particle distribution can be irregular.
- Neighbour-finding up and down the tree is simple, but has some problems:

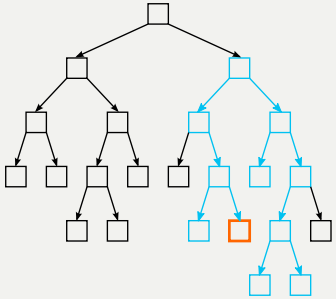


Task-based algorithms for SPH

Neighbour-finding with trees

Durham University

- In multi-resolution SPH, Spatial trees are the most commonly used approach to neighbour-finding, as the particle distribution can be irregular.
- Neighbour-finding up and down the tree is simple, but has some problems:
  - Worst-case cost in $\mathcal{O}(N^{2/3})$ per particle.

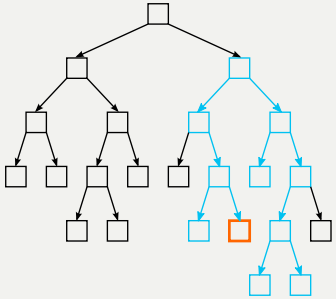


Task-based algorithms for SPH

Neighbour-finding with trees

Durham University

- In multi-resolution SPH, Spatial trees are the most commonly used approach to neighbour-finding, as the particle distribution can be irregular.
- Neighbour-finding up and down the tree is simple, but has some problems:
  - Worst-case cost in $\mathcal{O}(N^{2/3})$ per particle.
  - Low cache efficiency due to scattered memory access.

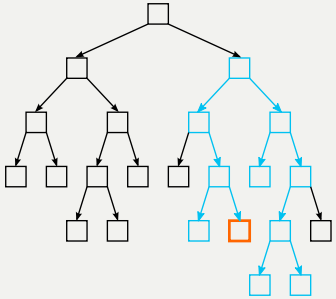


Task-based algorithms for SPH

Neighbour-finding with trees

Durham University

- In multi-resolution SPH, Spatial trees are the most commonly used approach to neighbour-finding, as the particle distribution can be irregular.
- Neighbour-finding up and down the tree is simple, but has some problems:
  - Worst-case cost in $\mathcal{O}(N^{2/3})$ per particle.
  - Low cache efficiency due to scattered memory access.
  - Symmetries cannot be exploited, i.e. each particle pair is found twice.

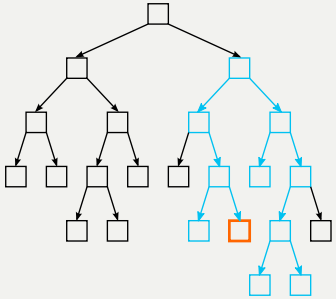


Task-based algorithms for SPH

Neighbour-finding with trees

Durham University

- In multi-resolution SPH, Spatial trees are the most commonly used approach to neighbour-finding, as the particle distribution can be irregular.
- Neighbour-finding up and down the tree is simple, but has some problems:
  - Worst-case cost in $\mathcal{O}(N^{2/3})$ per particle.
  - Low cache efficiency due to scattered memory access.
  - Symmetries cannot be exploited, i.e. each particle pair is found twice.
- Parallelization is trivial, but only because symmetries are not exploited.



Task-based algorithms for SPH

Hierarchical cell pairs

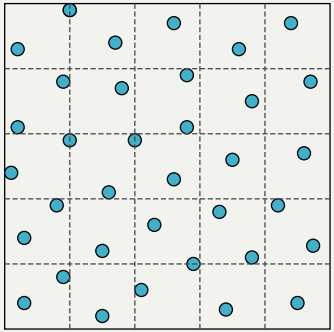
Durham University

Task-based algorithms for SPH

Hierarchical cell pairs

Durham University

- We start by splitting the simulation domain into rectangular cells of edge length at least $h_{\max}$.

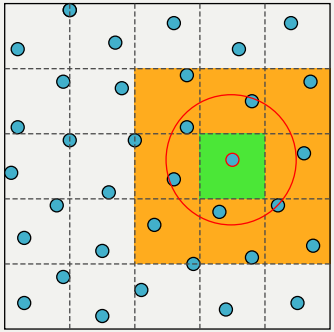


Task-based algorithms for SPH

Hierarchical cell pairs

Durham University

- We start by splitting the simulation domain into rectangular cells of edge length at least $h_{\max}$.
- All interacting particle pairs are then in either in the same cell, or in a pair of neighbouring cells.

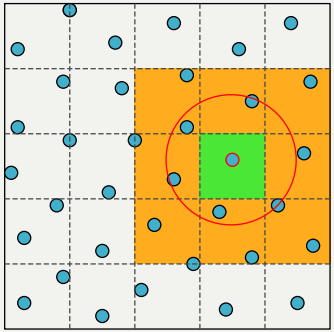


Task-based algorithms for SPH

Hierarchical cell pairs

Durham University

- We start by splitting the simulation domain into rectangular cells of edge length at least $h_{\max}$.
- All interacting particle pairs are then in either in the same cell, or in a pair of neighbouring cells.

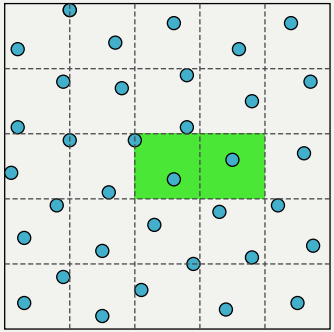


Task-based algorithms for SPH

Hierarchical cell pairs

Durham University

- We start by splitting the simulation domain into rectangular cells of edge length at least $h_{\max}$.
- All interacting particle pairs are then in either in the same cell, or in a pair of neighbouring cells.
- Finding all neighbours within each cell or between each pair of cells can be used as a task.

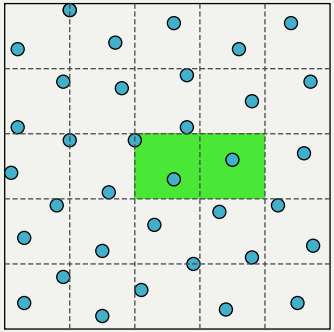


Task-based algorithms for SPH

Hierarchical cell pairs

Durham University

- We start by splitting the simulation domain into rectangular cells of edge length at least $h_{\max}$.
- All interacting particle pairs are then in either in the same cell, or in a pair of neighbouring cells.
- Finding all neighbours within each cell or between each pair of cells can be used as a task.

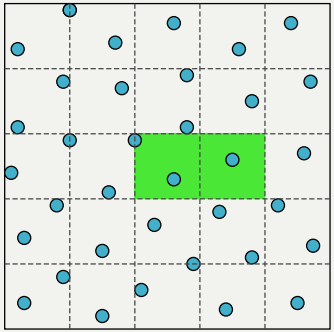


Task-based algorithms for SPH

Hierarchical cell pairs

Durham University

- We start by splitting the simulation domain into rectangular cells of edge length at least $h_{\max}$.
- All interacting particle pairs are then in either in the same cell, or in a pair of neighbouring cells.
- Finding all neighbours within each cell or between each pair of cells can be used as a task.
  $\longrightarrow$ Instead of traversing the tree for each particle, we traverse a list of cells and cell pairs and compute all interactions.



Task-based algorithms for SPH

Hierarchical cell pairs

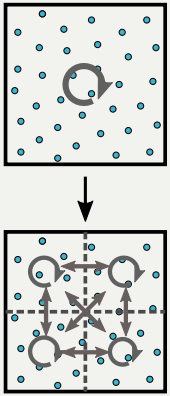
Durham University

- In a mutli-resolution setting, the initial cell-based decomposition can be extremely inefficient.

Task-based algorithms for SPH

Hierarchical cell pairs

- In a mutli-resolution setting, the initial cell-based decomposition can be extremely inefficient.
- If the particles in a cell are sufficiently small, the self-interaction task can be split.

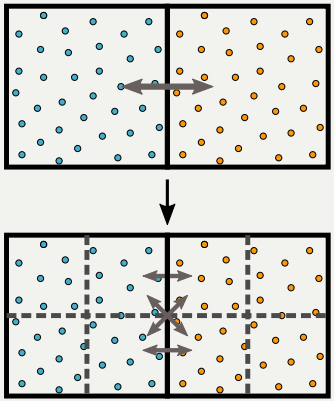


Task-based algorithms for SPH

Hierarchical cell pairs

Durham University

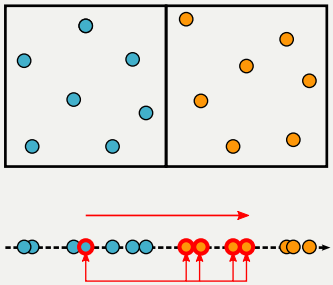
- In a mutli-resolution setting, the initial cell-based decomposition can be extremely inefficient.
- If the particles in a cell are sufficiently small, the self-interaction task can be split.
- Likewise, if the particles in a cell-pair are sufficiently small, the task can be split as well.



Task-based algorithms for SPH

Hierarchical cell pairs

- In a mutli-resolution setting, the initial cell-based decomposition can be extremely inefficient.
- If the particles in a cell are sufficiently small, the self-interaction task can be split.
- Likewise, if the particles in a cell-pair are sufficiently small, the task can be split as well.
- Finally, the particles in each cell pair are first sorted along the cell pair axis to speed-up neighbour-finding.



Task-based algorithms for SPH

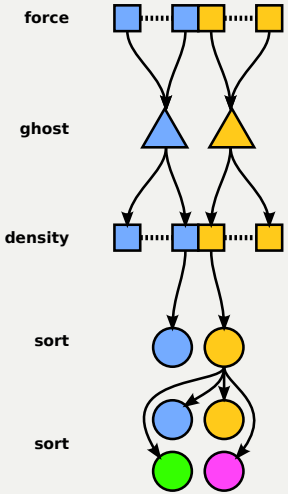
Task hierarchy

Task-based algorithms for SPH

Task hierarchy

Durham University

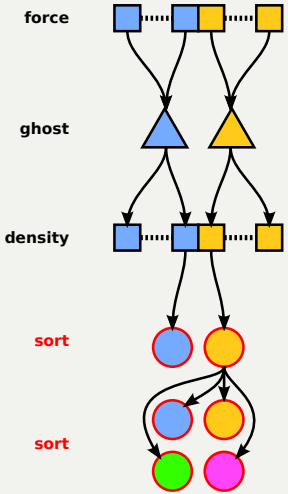
- Three main task types: Sorting, self-interactions, and pair-interactions.



Task-based algorithms for SPH

Task hierarchy

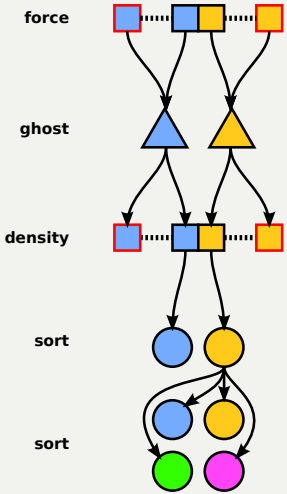
- Three main task types: Sorting, self-interactions, and pair-interactions.



Task-based algorithms for SPH

Task hierarchy

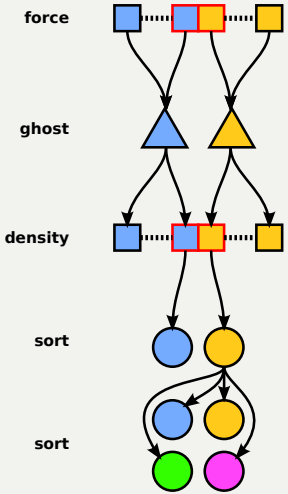
- Three main task types: Sorting, self-interactions, and pair-interactions.



Task-based algorithms for SPH

Task hierarchy

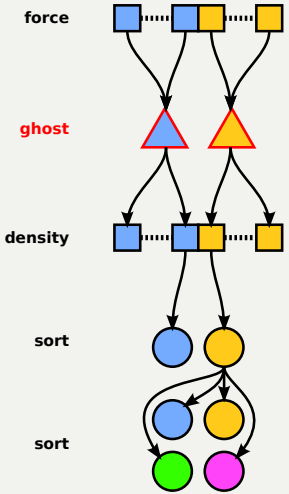
- Three main task types: Sorting, self-interactions, and pair-interactions.



Task-based algorithms for SPH

Task hierarchy

- Three main task types: Sorting, self-interactions, and pair-interactions.
- "Ghost" tasks are added to group dependencies between the density and force tasks of each cell.

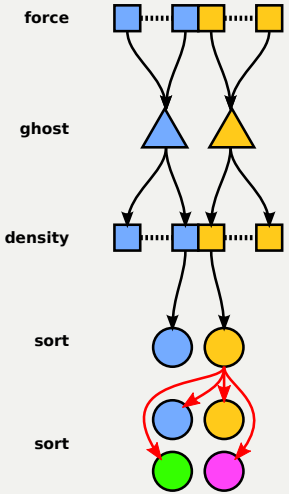


Task-based algorithms for SPH

Task hierarchy

Durham University

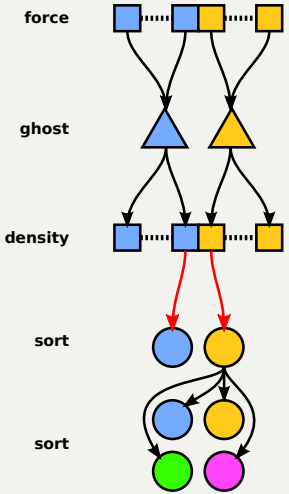
- Three main task types: Sorting, self-interactions, and pair-interactions.
- "Ghost" tasks are added to group dependencies between the density and force tasks of each cell.
- Each sorting task depends on the sorting tasks of its sub-cells (merge-sort).



Task-based algorithms for SPH

Task hierarchy

- Three main task types: Sorting, self-interactions, and pair-interactions.
- “Ghost” tasks are added to group dependencies between the density and force tasks of each cell.
- Each sorting task depends on the sorting tasks of its sub-cells (merge-sort).
- Each pair-interaction task depends on the sort tasks of the cells involved.

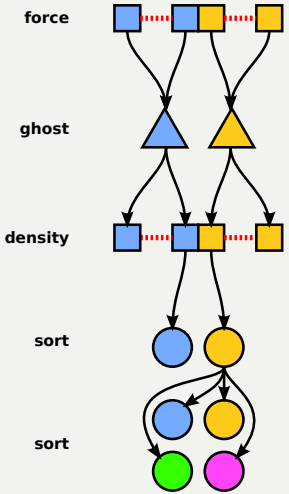


Task-based algorithms for SPH

Task hierarchy

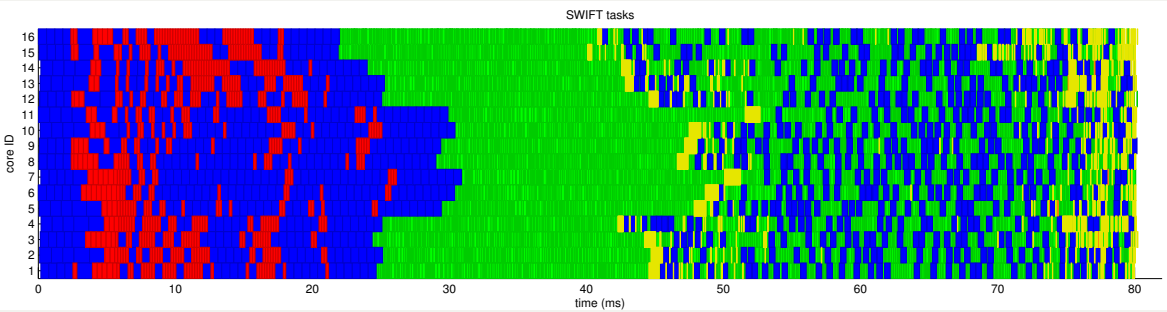
Durham University

- Three main task types: Sorting, self-interactions, and pair-interactions.
- "Ghost" tasks are added to group dependencies between the density and force tasks of each cell.
- Each sorting task depends on the sorting tasks of its sub-cells (merge-sort).
- Each pair-interaction task depends on the sort tasks of the cells involved.
- Tasks on overlapping cells conflict, i.e. they can not execute concurrently.



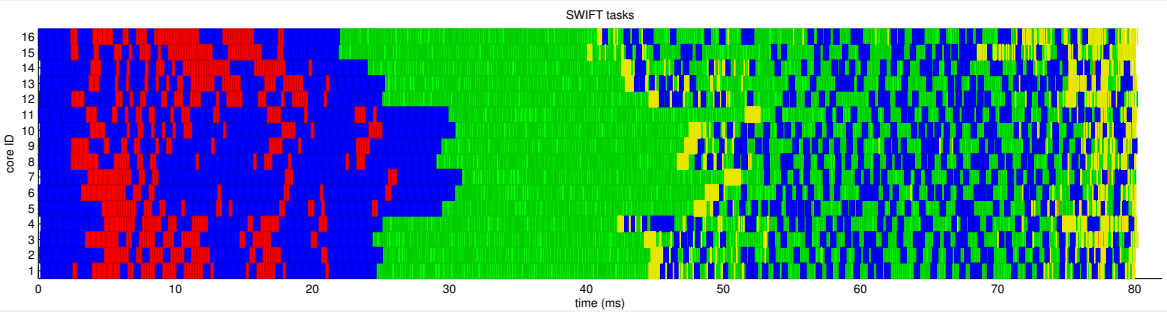
Task-based algorithms for SPH

Dynamic task allocation



Task-based algorithms for SPH

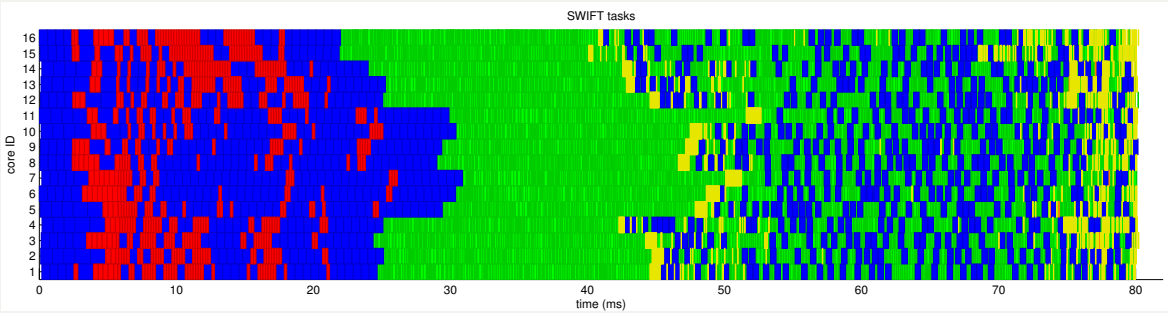
Dynamic task allocation



- Each core has it's own task queue and uses work-stealing when empty.

Task-based algorithms for SPH

Dynamic task allocation



- Each core has it's own task queue and uses work-stealing when empty.
- Each core has a preference for tasks involving cells which were used previously to improve cache re-use.

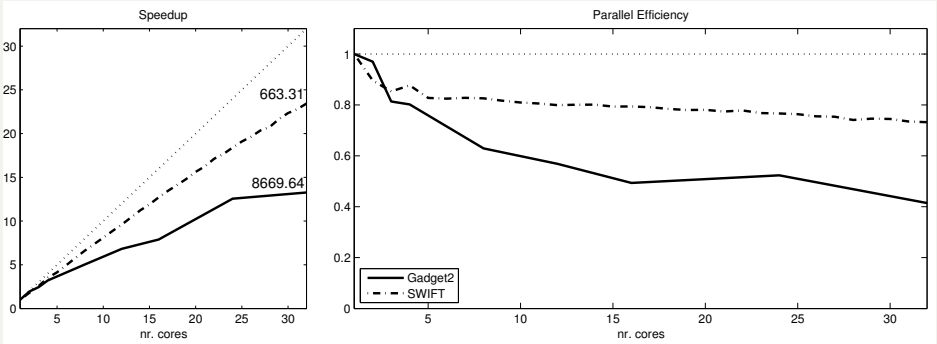
Task-based algorithms for SPH

Parallel efficiency and scaling

Durham University

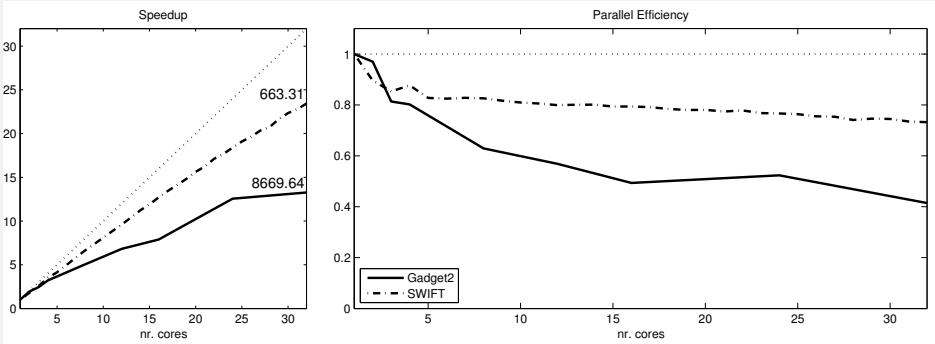
Task-based algorithms for SPH

Parallel efficiency and scaling



Task-based algorithms for SPH

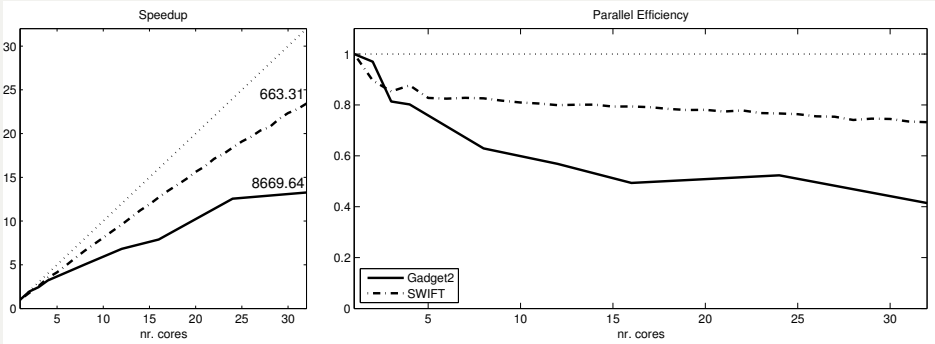
Parallel efficiency and scaling



- Parallel efficiency of 75% on 32 cores of an 4× Intel Xeon X7550 shared-memory machine.

Task-based algorithms for SPH

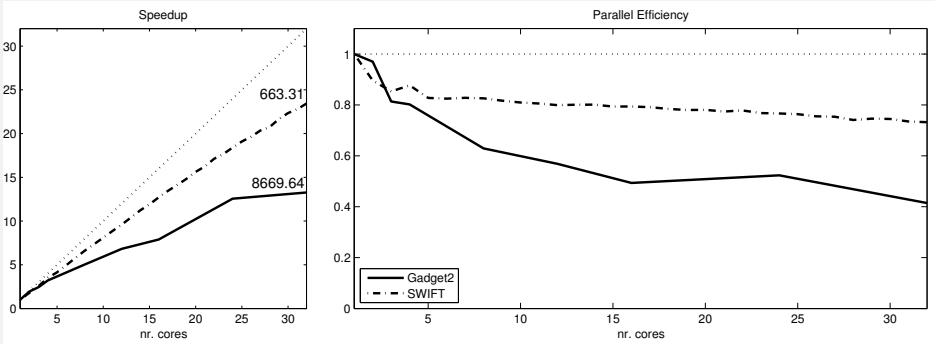
Parallel efficiency and scaling



- Parallel efficiency of 75% on 32 cores of an 4× Intel Xeon X7550 shared-memory machine.
- No NUMA-related effects: Each task operates exclusively on a small contiguous region of memory which usually fits in the lower level caches.

Task-based algorithms for SPH

Parallel efficiency and scaling



- Parallel efficiency of 75% on 32 cores of an 4× Intel Xeon X7550 shared-memory machine.
- No NUMA-related effects: Each task operates exclusively on a small contiguous region of memory which usually fits in the lower level caches.

Task-based algorithms for SPH

Further work

Task-based algorithms for SPH

Further work

Durham University

- SWIFT is currently an Open-Source research code, but we aim to be able to do production runs in Computational Cosmology.

Task-based algorithms for SPH

Further work

- SWIFT is currently an Open-Source research code, but we aim to be able to do production runs in Computational Cosmology.

Task-based algorithms for SPH

Further work

Durham University

- SWIFT is currently an Open-Source research code, but we aim to be able to do production runs in Computational Cosmology.
- Modular structure, easy to add new physics, kernel functions, etc…

Task-based algorithms for SPH

Further work

- SWIFT is currently an Open-Source research code, but we aim to be able to do production runs in Computational Cosmology.
- Modular structure, easy to add new physics, kernel functions, etc…
- Hybrid shared/distributed-memory parallelism:

Task-based algorithms for SPH

Further work

Durham University

- SWIFT is currently an Open-Source research code, but we aim to be able to do production runs in Computational Cosmology.
- Modular structure, easy to add new physics, kernel functions, etc…
- Hybrid shared/distributed-memory parallelism:
  - Distributed-memory parallelism between the nodes of a cluster,.

Task-based algorithms for SPH

Further work

Durham University

- SWIFT is currently an Open-Source research code, but we aim to be able to do production runs in Computational Cosmology.
- Modular structure, easy to add new physics, kernel functions, etc…
- Hybrid shared/distributed-memory parallelism:
  - Distributed-memory parallelism between the nodes of a cluster,.
  - Task-based shared-memory parallelism between the cores of a single node.

Task-based algorithms for SPH

Further work

Durham University

- SWIFT is currently an Open-Source research code, but we aim to be able to do production runs in Computational Cosmology.
- Modular structure, easy to add new physics, kernel functions, etc…
- Hybrid shared/distributed-memory parallelism:
  - Distributed-memory parallelism between the nodes of a cluster,.
  - Task-based shared-memory parallelism between the cores of a single node.
  - SIMD/SIMT parallelism within each task on a single core.

Task-based algorithms for SPH

Further work

Durham University

- SWIFT is currently an Open-Source research code, but we aim to be able to do production runs in Computational Cosmology.
- Modular structure, easy to add new physics, kernel functions, etc…
- Hybrid shared/distributed-memory parallelism:
  - Distributed-memory parallelism between the nodes of a cluster,.
  - Task-based shared-memory parallelism between the cores of a single node.
  - SIMD/SIMT parallelism within each task on a single core.
- In a task-based hybrid setup, sending and receiving data asynchronously can be implemented as tasks to hide communication latencies.

Task-based algorithms for SPH

Further work

Durham University

- SWIFT is currently an Open-Source research code, but we aim to be able to do production runs in Computational Cosmology.
- Modular structure, easy to add new physics, kernel functions, etc…
- Hybrid shared/distributed-memory parallelism:
  - Distributed-memory parallelism between the nodes of a cluster,.
  - Task-based shared-memory parallelism between the cores of a single node.
  - SIMD/SIMT parallelism within each task on a single core.
- In a task-based hybrid setup, sending and receiving data asynchronously can be implemented as tasks to hide communication latencies.
- Mixed-precision floating point arithmetic in order to better exploit SIMD vectorization, yet without losing precision.

Task-based algorithms for SPH

Further work

Durham University

- SWIFT is currently an Open-Source research code, but we aim to be able to do production runs in Computational Cosmology.
- Modular structure, easy to add new physics, kernel functions, etc…
- Hybrid shared/distributed-memory parallelism:
  - Distributed-memory parallelism between the nodes of a cluster,.
  - Task-based shared-memory parallelism between the cores of a single node.
  - SIMD/SIMT parallelism within each task on a single core.
- In a task-based hybrid setup, sending and receiving data asynchronously can be implemented as tasks to hide communication latencies.
- Mixed-precision floating point arithmetic in order to better exploit SIMD vectorization, yet without losing precision.

Task-based algorithms for SPH

Further work

Durham University

- SWIFT is currently an Open-Source research code, but we aim to be able to do production runs in Computational Cosmology.
- Modular structure, easy to add new physics, kernel functions, etc…
- Hybrid shared/distributed-memory parallelism:
  - Distributed-memory parallelism between the nodes of a cluster,.
  - Task-based shared-memory parallelism between the cores of a single node.
  - SIMD/SIMT parallelism within each task on a single core.
- In a task-based hybrid setup, sending and receiving data asynchronously can be implemented as tasks to hide communication latencies.
- Mixed-precision floating point arithmetic in order to better exploit SIMD vectorization, yet without losing precision.
- Tasks can also be shared between the CPU and a GPU.

Conclusions
Take-home messages

Durham University

- In order to continue getting *faster*, programs need to become *more parallel*.
  $\longrightarrow$ If a program has reached its maximum degree of parallelism, it won't get any faster. Ever.
- On shared-memory systems, asynchronous task-based parallelism solves most problems with concurrency and scaling.
  $\longrightarrow$ But we still need to develop task-based algorithms for specific problems, e.g. SPH simulations.
- Better algorithms alone can lead to speedups of up to a factor of ten.
  $\longrightarrow$ Better use of both existing and future infrastructure.

Conclusions

Take-home messages

Durham University

- In order to continue getting *faster*, programs need to become *more parallel*.
  $\longrightarrow$ If a program has reached its maximum degree of parallelism, it won't get any faster. Ever.
- On shared-memory systems, asynchronous task-based parallelism solves most problems with concurrency and scaling.
  $\longrightarrow$ But we still need to develop task-based algorithms for specific problems, e.g. SPH simulations.
- Better algorithms alone can lead to speedups of up to a factor of ten.
  $\longrightarrow$ Better use of both existing and future infrastructure.
- If you have an interesting problem, or just need a fast code, let us know!

Conclusions

Take-home messages

Durham University

- In order to continue getting *faster*, programs need to become *more parallel*.
  $\longrightarrow$ If a program has reached its maximum degree of parallelism, it won't get any faster. Ever.
- On shared-memory systems, asynchronous task-based parallelism solves most problems with concurrency and scaling.
  $\longrightarrow$ But we still need to develop task-based algorithms for specific problems, e.g. SPH simulations.
- Better algorithms alone can lead to speedups of up to a factor of ten.
  $\longrightarrow$ Better use of both existing and future infrastructure.
- If you have an interesting problem, or just need a fast code, let us know!

Durham University

Thank you for your attention!