Task-based parallelism, and why it is awesome

Pedro Gonnet, SECS/ICC, Durham University

CSAM-15 Workshop on Computational Solar and Astrophysical Modeling, Jülich Supercomputing Centre, September 16th, 2015

Introduction

This talk in a nutshell

Durham University

Durham University

- Forget most of what you've ever learned about parallel computing: most of it was true 20 years ago, but not today.

Durham University

- Forget most of what you've ever learned about parallel computing: most of it was true 20 years ago, but not today.

Introduction
This talk in a nutshell

Durham University

- Forget most of what you've ever learned about parallel computing: most of it was true 20 years ago, but not today.

Durham University

- Forget most of what you've ever learned about parallel computing: most of it was true 20 years ago, but not today.
  $\longrightarrow$ Computers are not getting faster, but more parallel.

Durham University

- Forget most of what you've ever learned about parallel computing: most of it was true 20 years ago, but not today.
  $\longrightarrow$ Computers are not getting faster, but more parallel.
  $\longrightarrow$ Clusters are only growing in the number of cores per nodes.

Durham University

- Forget most of what you've ever learned about parallel computing: most of it was true 20 years ago, but not today.
  $\longrightarrow$ Computers are not getting faster, but more parallel.
  $\longrightarrow$ Clusters are only growing in the number of cores per nodes.
  $\longrightarrow$ Networking hardware is not getting any faster.

Durham University

- Forget most of what you've ever learned about parallel computing: most of it was true 20 years ago, but not today.
  $\longrightarrow$ Computers are not getting faster, but more parallel.
  $\longrightarrow$ Clusters are only growing in the number of cores per nodes.
  $\longrightarrow$ Networking hardware is not getting any faster. Or cheaper.

Durham University

- Forget most of what you've ever learned about parallel computing: most of it was true 20 years ago, but not today.
  $\longrightarrow$ Computers are not getting faster, but more parallel.
  $\longrightarrow$ Clusters are only growing in the number of cores per nodes.
  $\longrightarrow$ Networking hardware is not getting any faster. Or cheaper.
- Task-based parallelism provides good strong scaling for shared-memory parallel computation on a single node.

Durham University

- Forget most of what you've ever learned about parallel computing: most of it was true 20 years ago, but not today.
  $\longrightarrow$ Computers are not getting faster, but more parallel.
  $\longrightarrow$ Clusters are only growing in the number of cores per nodes.
  $\longrightarrow$ Networking hardware is not getting any faster. Or cheaper.
- Task-based parallelism provides good strong scaling for shared-memory parallel computation on a single node.
- It can be used to implement efficient and scalable asynchronous hybrid shared/distributed-memory parallelism.

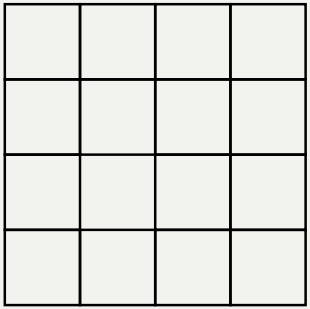
- Forget most of what you've ever learned about parallel computing: most of it was true 20 years ago, but not today.
  $\longrightarrow$ Computers are not getting faster, but more parallel.
  $\longrightarrow$ Clusters are only growing in the number of cores per nodes.
  $\longrightarrow$ Networking hardware is not getting any faster. Or cheaper.
- Task-based parallelism provides good strong scaling for shared-memory parallel computation on a single node.
- It can be used to implement efficient and scalable asynchronous hybrid shared/distributed-memory parallelism.
- The bad news is that since it's a different paradigm, using it will require you to re-write most of your codes.

Forget what you've learned

The problem with distributed-memory parallelism

Durham University

- Distributed-memory parallelism, e.g. using MPI, is based on data decomposition, i.e. each processor is assigned part of the problem to work on and communicates with its neighbours.
- Surface-to-volume ratio problem: As the number of cores increases, the amount of computation per core (volume) decreases while the relative amount of communication (surface) increases, eventually dominating the entire computation.
  $\longrightarrow$ We can always do larger simulations, but not smaller simulations faster.

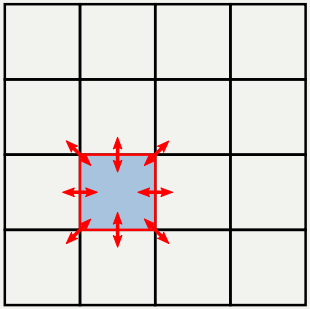


Forget what you've learned

The problem with distributed-memory parallelism

Durham University

- Distributed-memory parallelism, e.g. using MPI, is based on data decomposition, i.e. each processor is assigned part of the problem to work on and communicates with its neighbours.
- Surface-to-volume ratio problem: As the number of cores increases, the amount of computation per core (volume) decreases while the relative amount of communication (surface) increases, eventually dominating the entire computation.
  $\longrightarrow$ We can always do larger simulations, but not smaller simulations faster.

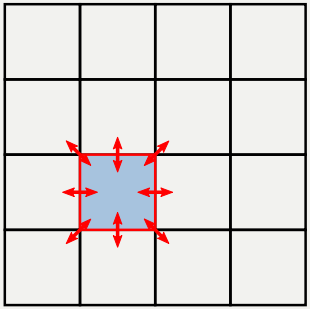


Forget what you've learned

The problem with distributed-memory parallelism

Durham University

- Distributed-memory parallelism, e.g. using MPI, is based on data decomposition, i.e. each processor is assigned part of the problem to work on and communicates with its neighbours.
- Surface-to-volume ratio problem: As the number of cores increases, the amount of computation per core (volume) decreases while the relative amount of communication (surface) increases, eventually dominating the entire computation.
  $\longrightarrow$ We can always do larger simulations, but not smaller simulations faster.

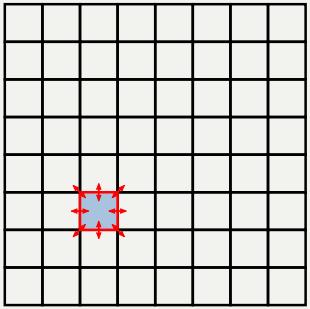


Forget what you've learned

The problem with distributed-memory parallelism

Durham University

- Distributed-memory parallelism, e.g. using MPI, is based on data decomposition, i.e. each processor is assigned part of the problem to work on and communicates with its neighbours.
- Surface-to-volume ratio problem: As the number of cores increases, the amount of computation per core (volume) decreases while the relative amount of communication (surface) increases, eventually dominating the entire computation.
  $\longrightarrow$ We can always do larger simulations, but not smaller simulations faster.

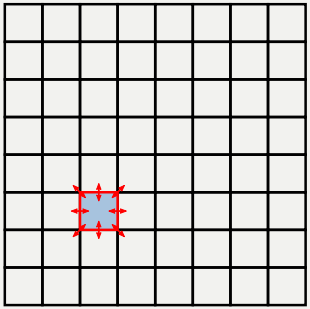


Forget what you've learned

The problem with distributed-memory parallelism

Durham University

- Distributed-memory parallelism, e.g. using MPI, is based on data decomposition, i.e. each processor is assigned part of the problem to work on and communicates with its neighbours.
- Surface-to-volume ratio problem: As the number of cores increases, the amount of computation per core (volume) decreases while the relative amount of communication (surface) increases, eventually dominating the entire computation.
  $\longrightarrow$ We can always do larger simulations, but not smaller simulations faster.

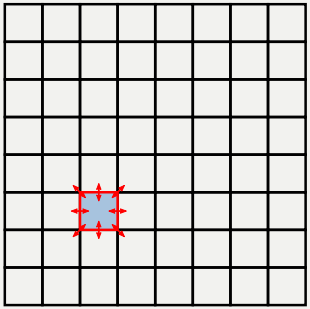


Forget what you've learned

The problem with distributed-memory parallelism

Durham University

- Distributed-memory parallelism, e.g. using MPI, is based on data decomposition, i.e. each processor is assigned part of the problem to work on and communicates with its neighbours.
- Surface-to-volume ratio problem: As the number of cores increases, the amount of computation per core (volume) decreases while the relative amount of communication (surface) increases, eventually dominating the entire computation.
  $\longrightarrow$ We can always do larger simulations, but not smaller simulations faster.

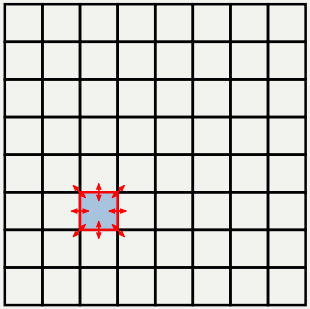


Forget what you've learned

The problem with distributed-memory parallelism

Durham University

- Distributed-memory parallelism, e.g. using MPI, is based on data decomposition, i.e. each processor is assigned part of the problem to work on and communicates with its neighbours.
- Surface-to-volume ratio problem: As the number of cores increases, the amount of computation per core (volume) decreases while the relative amount of communication (surface) increases, eventually dominating the entire computation.
  $\longrightarrow$ We can always do larger simulations, but not smaller simulations faster.



Forget what you've learned

The problem with OpenMP

Durham University

- Shared-memory parallelism using OpenMP, i.e. annotating an inherently serial code, is a simple way to exploit shared-memory parallelism.
- Concurrency problems need to be addressed explicitly, e.g. using barriers or atomic instructions.
- These overheads associated with these two solutions only get worse as the number of cores increases.

```
for ( i = 0 ; i < N ; i++ ) {
    ...
    globalvar += ...
    }
```

Forget what you've learned

The problem with OpenMP

Durham University

- Shared-memory parallelism using OpenMP, i.e. annotating an inherently serial code, is a simple way to exploit shared-memory parallelism.
- Concurrency problems need to be addressed explicitly, e.g. using barriers or atomic instructions.
- These overheads associated with these two solutions only get worse as the number of cores increases.

```
#pragma omp parallel for
for ( i = 0 ; i < N ; i++ ) {
    ...
    globalvar += ...
    }
```

Forget what you've learned

The problem with OpenMP

Durham University

- Shared-memory parallelism using OpenMP, i.e. annotating an inherently serial code, is a simple way to exploit shared-memory parallelism.
- Concurrency problems need to be addressed explicitly, e.g. using barriers or atomic instructions.
- These overheads associated with these two solutions only get worse as the number of cores increases.

```
#pragma omp parallel for
for ( i = 0 ; i < N ; i++ ) {
    ...
    #pragma omp critical
    globalvar += ...
    }
```

Forget what you've learned

The problem with OpenMP

Durham University

- Shared-memory parallelism using OpenMP, i.e. annotating an inherently serial code, is a simple way to exploit shared-memory parallelism.
- Concurrency problems need to be addressed explicitly, e.g. using barriers or atomic instructions.
- These overheads associated with these two solutions only get worse as the number of cores increases.

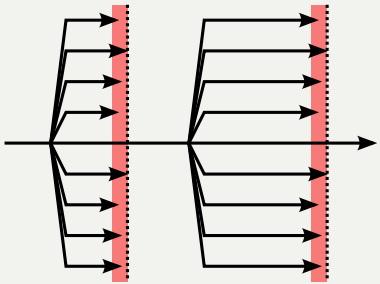
```
#pragma omp parallel for
for ( i = 0 ; i < N ; i++ ) {
    ...
    #pragma omp critical
    globalvar += ...
    }
```

Forget what you've learned

The MPI/OpenMP paradigm is wrong

Durham University

- Both MPI and OpenMP rely on the SPMD (Single Program/Multiple Data) programming model, i.e. the same bit of code is executed by all threads/nodes at more or less the same time.
- This means that serial bits or communication create synchronization points.
- This also means that any expensive bits, e.g. communication or disk I/O, will happen all at the same time.
- This creates bottlenecks throughout the code which eventually kill any potential scaling.

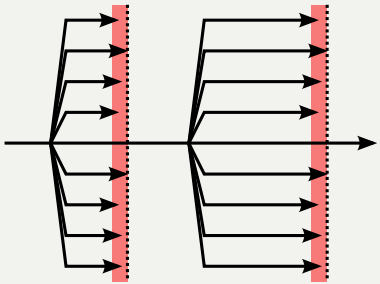


Forget what you've learned

The MPI/OpenMP paradigm is wrong

Durham University

- Both MPI and OpenMP rely on the SPMD (Single Program/Multiple Data) programming model, i.e. the same bit of code is executed by all threads/nodes at more or less the same time.
- This means that serial bits or communication create synchronization points.
- This also means that any expensive bits, e.g. communication or disk I/O, will happen all at the same time.
- This creates bottlenecks throughout the code which eventually kill any potential scaling.

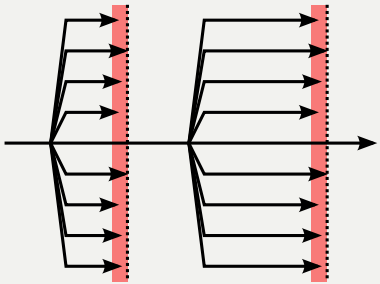


Forget what you've learned

The MPI/OpenMP paradigm is wrong

Durham University

- Both MPI and OpenMP rely on the SPMD (Single Program/Multiple Data) programming model, i.e. the same bit of code is executed by all threads/nodes at more or less the same time.
- This means that serial bits or communication create synchronization points.
- This also means that any expensive bits, e.g. communication or disk I/O, will happen all at the same time.
- This creates bottlenecks throughout the code which eventually kill any potential scaling.

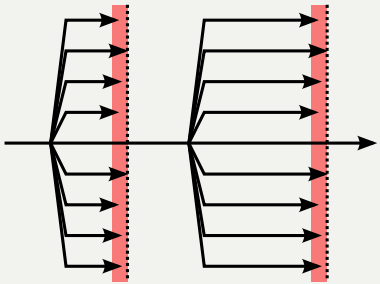


Forget what you've learned

The MPI/OpenMP paradigm is wrong

Durham University

- Both MPI and OpenMP rely on the SPMD (Single Program/Multiple Data) programming model, i.e. the same bit of code is executed by all threads/nodes at more or less the same time.
- This means that serial bits or communication create synchronization points.
- This also means that any expensive bits, e.g. communication or disk I/O, will happen all at the same time.
- This creates bottlenecks throughout the code which eventually kill any potential scaling.



Forget what you've learned

In summary

Durham University

- MPI and OpenMP will do the job for large problems on a small number of machines.
- Both approaches, however, scale badly for fixed-size problems on increasing number of cores.
- Scaling is currently being pushed by pushing the hardware, but this is an incredibly expensive and ultimately limited strategy.
- The problems are inherent to the underlying programming model, and thus cannot be fixed without changing the model.

Forget what you've learned

In summary

Durham University

- MPI and OpenMP will do the job for large problems on a small number of machines.
- Both approaches, however, scale badly for fixed-size problems on increasing number of cores.
- Scaling is currently being pushed by pushing the hardware, but this is an incredibly expensive and ultimately limited strategy.
- The problems are inherent to the underlying programming model, and thus cannot be fixed without changing the model.

Forget what you've learned

In summary

Durham University

- MPI and OpenMP will do the job for large problems on a small number of machines.
- Both approaches, however, scale badly for fixed-size problems on increasing number of cores.
- Scaling is currently being pushed by pushing the hardware, but this is an incredibly expensive and ultimately limited strategy.
- The problems are inherent to the underlying programming model, and thus cannot be fixed without changing the model.

Forget what you've learned

In summary

Durham University

- MPI and OpenMP will do the job for large problems on a small number of machines.
- Both approaches, however, scale badly for fixed-size problems on increasing number of cores.
- Scaling is currently being pushed by pushing the hardware, but this is an incredibly expensive and ultimately limited strategy.
- The problems are inherent to the underlying programming model, and thus cannot be fixed without changing the model.

Forget what you've learned

In summary

Durham University

- MPI and OpenMP will do the job for large problems on a small number of machines.
- Both approaches, however, scale badly for fixed-size problems on increasing number of cores.
- Scaling is currently being pushed by pushing the hardware, but this is an incredibly expensive and ultimately limited strategy.
- The problems are inherent to the underlying programming model, and thus cannot be fixed without changing the model.

Forget what you've learned

In summary

Durham University

- MPI and OpenMP will do the job for large problems on a small number of machines.
- Both approaches, however, scale badly for fixed-size problems on increasing number of cores.
- Scaling is currently being pushed by pushing the hardware, but this is an incredibly expensive and ultimately limited strategy.
- The problems are inherent to the underlying programming model, and thus cannot be fixed without changing the model.

Task-based parallelism
Main concepts

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 concepts

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 concepts

Durham University

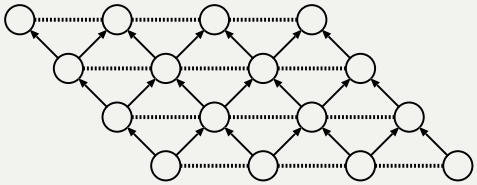
- Shared-memory parallel programming paradigm in which the computation is formulated in an implicitly parallelizable way that automatically avoids most of the problems associated with concurrency and load-balancing.

Task-based parallelism

Main concepts

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.

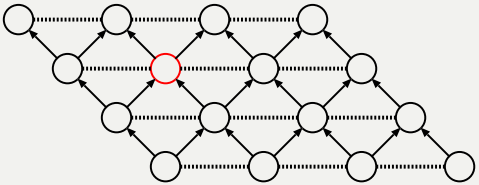


Task-based parallelism

Main concepts

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.

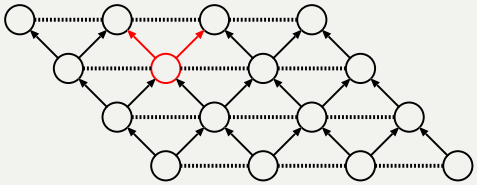


Task-based parallelism

Main concepts

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.

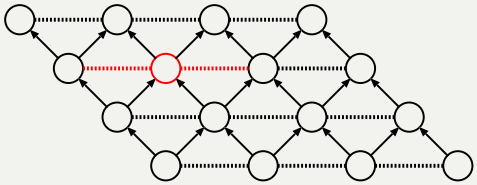


Task-based parallelism

Main concepts

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.

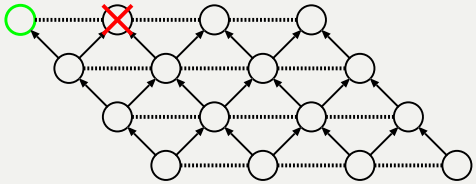


Task-based parallelism

Main concepts

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.

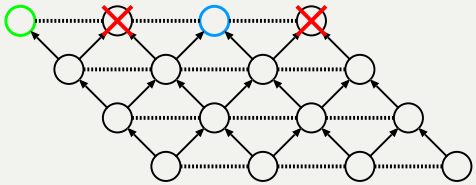


Task-based parallelism

Main concepts

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.

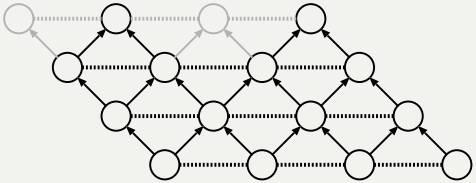


Task-based parallelism

Main concepts

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.

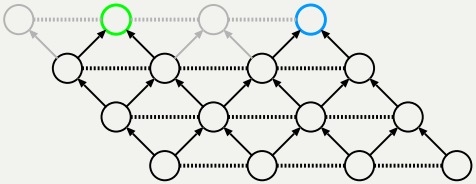


Task-based parallelism

Main concepts

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.

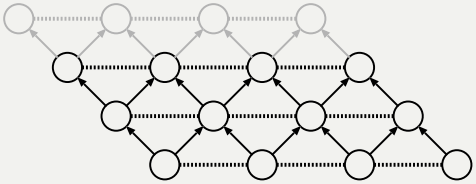


Task-based parallelism

Main concepts

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
Implementations

Durham University

- Several task-based implementations exist, and differ mainly in how tasks and dependencies are created/specified.
- Task spawning: any function can *spawn* a task, i.e. call a function that will be executed as a task. Dependencies are implicitly given by the order in which tasks are spawned, e.g. Cilk, OpenMP 4.0.
- Dependency deduction: tasks and the data they operate on are specified explicitly, dependencies are deduced from the data and the order in which the tasks are created, e.g. QUARK, OmpSs, StarPU.
- Explicit task graph construction: tasks and dependencies are explicitly specified by the user, e.g. Intel TBB, QuickSched.

Task-based parallelism
Implementations

Durham University

- Several task-based implementations exist, and differ mainly in how tasks and dependencies are created/specified.
- Task spawning: any function can *spawn* a task, i.e. call a function that will be executed as a task. Dependencies are implicitly given by the order in which tasks are spawned, e.g. Cilk, OpenMP 4.0.
- Dependency deduction: tasks and the data they operate on are specified explicitly, dependencies are deduced from the data and the order in which the tasks are created, e.g. QUARK, OmpSs, StarPU.
- Explicit task graph construction: tasks and dependencies are explicitly specified by the user, e.g. Intel TBB, QuickSched.

Task-based parallelism
Implementations

Durham University

- Several task-based implementations exist, and differ mainly in how tasks and dependencies are created/specified.
- Task spawning: any function can *spawn* a task, i.e. call a function that will be executed as a task. Dependencies are implicitly given by the order in which tasks are spawned, e.g. Cilk, OpenMP 4.0.
- Dependency deduction: tasks and the data they operate on are specified explicitly, dependencies are deduced from the data and the order in which the tasks are created, e.g. QUARK, OmpSs, StarPU.
- Explicit task graph construction: tasks and dependencies are explicitly specified by the user, e.g. Intel TBB, QuickSched.

Task-based parallelism
Implementations

Durham University

- Several task-based implementations exist, and differ mainly in how tasks and dependencies are created/specified.
- Task spawning: any function can *spawn* a task, i.e. call a function that will be executed as a task. Dependencies are implicitly given by the order in which tasks are spawned, e.g. Cilk, OpenMP 4.0.
- Dependency deduction: tasks and the data they operate on are specified explicitly, dependencies are deduced from the data and the order in which the tasks are created, e.g. QUARK, OmpSs, StarPU.
- Explicit task graph construction: tasks and dependencies are explicitly specified by the user, e.g. Intel TBB, QuickSched.

Task-based parallelism
Implementations

Durham University

- Several task-based implementations exist, and differ mainly in how tasks and dependencies are created/specified.
- Task spawning: any function can *spawn* a task, i.e. call a function that will be executed as a task. Dependencies are implicitly given by the order in which tasks are spawned, e.g. Cilk, OpenMP 4.0.
- Dependency deduction: tasks and the data they operate on are specified explicitly, dependencies are deduced from the data and the order in which the tasks are created, e.g. QUARK, OmpSs, StarPU.
- Explicit task graph construction: tasks and dependencies are explicitly specified by the user, e.g. Intel TBB, QuickSched.

Task-based parallelism

Modelling constraints with resources

Durham University

- In many task-based implementations, conflicts are modeled by adding artificial dependencies between conflicting tasks, introducing additional constraints which can severely impair scalability.
- Instead, we will model conflicts via shared resources, i.e. two or more tasks conflict if they require the same resource.
- A task will only execute if it can get exclusive locks on all its resources.

Task-based parallelism

Modelling constraints with resources

Durham University

- In many task-based implementations, conflicts are modeled by adding artificial dependencies between conflicting tasks, introducing additional constraints which can severely impair scalability.
- Instead, we will model conflicts via shared resources, i.e. two or more tasks conflict if they require the same resource.
- A task will only execute if it can get exclusive locks on all its resources.

Task-based parallelism

Modelling constraints with resources

Durham University

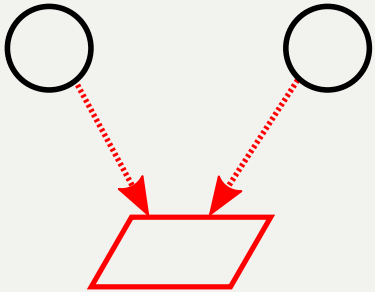
- In many task-based implementations, conflicts are modeled by adding artificial dependencies between conflicting tasks, introducing additional constraints which can severely impair scalability.
- Instead, we will model conflicts via shared resources, i.e. two or more tasks conflict if they require the same resource.
- A task will only execute if it can get exclusive locks on all its resources.

Task-based parallelism

Modelling constraints with resources

Durham University

- In many task-based implementations, conflicts are modeled by adding artificial dependencies between conflicting tasks, introducing additional constraints which can severely impair scalability.
- Instead, we will model conflicts via shared resources, i.e. two or more tasks conflict if they require the same resource.
- A task will only execute if it can get exclusive locks on all its resources.

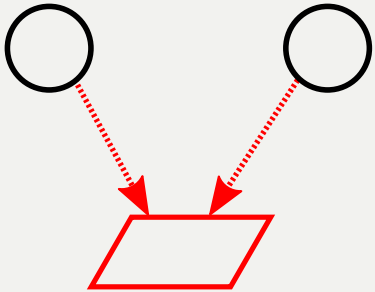


Task-based parallelism

Modelling constraints with resources

Durham University

- In many task-based implementations, conflicts are modeled by adding artificial dependencies between conflicting tasks, introducing additional constraints which can severely impair scalability.
- Instead, we will model conflicts via shared resources, i.e. two or more tasks conflict if they require the same resource.
- A task will only execute if it can get exclusive locks on all its resources.

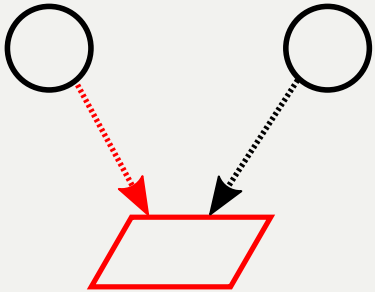


Task-based parallelism

Modelling constraints with resources

Durham University

- In many task-based implementations, conflicts are modeled by adding artificial dependencies between conflicting tasks, introducing additional constraints which can severely impair scalability.
- Instead, we will model conflicts via shared resources, i.e. two or more tasks conflict if they require the same resource.
- A task will only execute if it can get exclusive locks on all its resources.



Task-based parallelism
Main advantages

Durham University

- The order in which the tasks are processed is highly 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.
- However, this usually means that we have to completely re-think our entire computation, e.g. redesign it from scratch to make it task-based.
- The most interesting aspect, though, is what we can do with this representation of our computations.

Task-based parallelism
Main advantages

Durham University

- The order in which the tasks are processed is highly 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.
- However, this usually means that we have to completely re-think our entire computation, e.g. redesign it from scratch to make it task-based.
- The most interesting aspect, though, is what we can do with this representation of our computations.

Task-based parallelism
Main advantages

Durham University

- The order in which the tasks are processed is highly 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.
- However, this usually means that we have to completely re-think our entire computation, e.g. redesign it from scratch to make it task-based.
- The most interesting aspect, though, is what we can do with this representation of our computations.

Task-based parallelism
Main advantages

Durham University

- The order in which the tasks are processed is highly 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.
- However, this usually means that we have to completely re-think our entire computation, e.g. redesign it from scratch to make it task-based.
- The most interesting aspect, though, is what we can do with this representation of our computations.

Task-based parallelism
Main advantages

Durham University

- The order in which the tasks are processed is highly 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.
- However, this usually means that we have to completely re-think our entire computation, e.g. redesign it from scratch to make it task-based.
- The most interesting aspect, though, is what we can do with this representation of our computations.

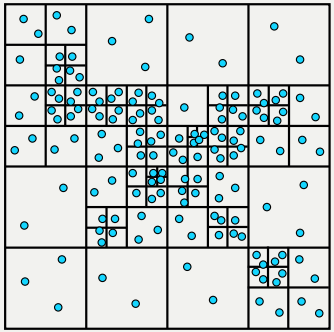
Task-based algorithms
Neighbour-finding with trees

Durham University

Task-based algorithms
Neighbour-finding with trees

Durham University

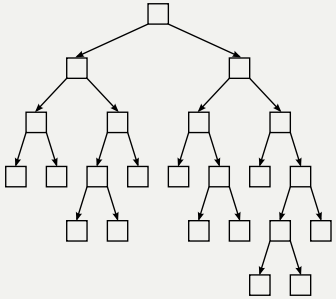
- Spatial trees are the most commonly used approach to neighbour-finding, as the particle distribution can be irregular.
- Neighbour-finding up and down the tree is simple, but has some problems:
  - Worst-case cost in $\mathcal{O}(N^{2/3})$ per particle.
  - Low cache efficiency due to scattered memory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

Durham University

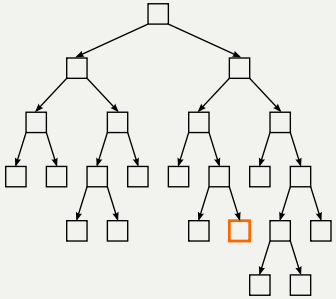
- Spatial trees are the most commonly used approach to neighbour-finding, as the particle distribution can be irregular.
- Neighbour-finding up and down the tree is simple, but has some problems:
  - Worst-case cost in $\mathcal{O}(N^{2/3})$ per particle.
  - Low cache efficiency due to scattered memory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

Durham University

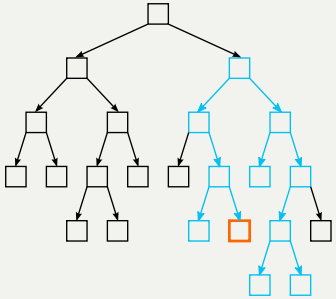
- Spatial trees are the most commonly used approach to neighbour-finding, as the particle distribution can be irregular.
- Neighbour-finding up and down the tree is simple, but has some problems:
  - Worst-case cost in $\mathcal{O}(N^{2/3})$ per particle.
  - Low cache efficiency due to scattered memory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

Durham University

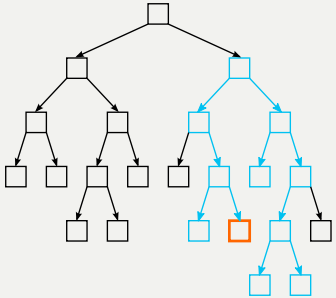
- Spatial trees are the most commonly used approach to neighbour-finding, as the particle distribution can be irregular.
- Neighbour-finding up and down the tree is simple, but has some problems:
  - Worst-case cost in $\mathcal{O}(N^{2/3})$ per particle.
  - Low cache efficiency due to scattered memory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

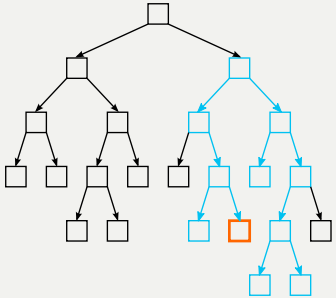
Durham University

- 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
Neighbour-finding with trees

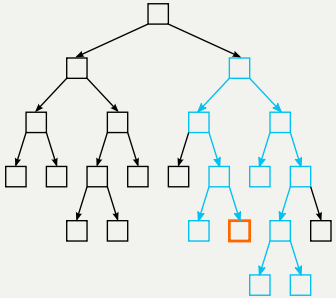
- Spatial trees are the most commonly used approach to neighbour-finding, as the particle distribution can be irregular.
- Neighbour-finding up and down the tree is simple, but has some problems:
  - Worst-case cost in $\mathcal{O}(N^{2/3})$ per particle.
  - Low cache efficiency due to scattered memory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

Durham University

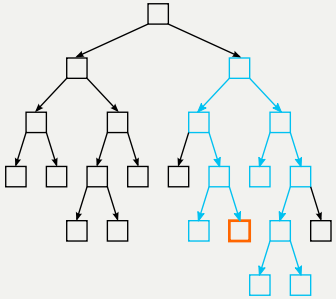
- Spatial trees are the most commonly used approach to neighbour-finding, as the particle distribution can be irregular.
- Neighbour-finding up and down the tree is simple, but has some problems:
  - Worst-case cost in $\mathcal{O}(N^{2/3})$ per particle.
  - Low cache efficiency due to scattered memory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

Durham University

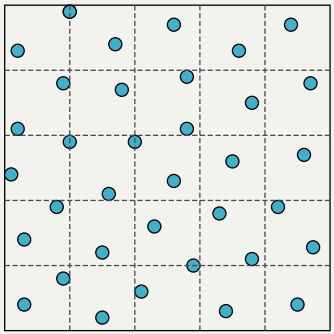
- Spatial trees are the most commonly used approach to neighbour-finding, as the particle distribution can be irregular.
- Neighbour-finding up and down the tree is simple, but has some problems:
  - Worst-case cost in $\mathcal{O}(N^{2/3})$ per particle.
  - Low cache efficiency due to scattered memory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

Hierarchical cell pairs

- 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.
- If the particles in the cell or cell pair are sufficiently small, the task can be split.
- Finally, the particles in each cell pair are first sorted along the cell pair axis to speed-up neighbour-finding.

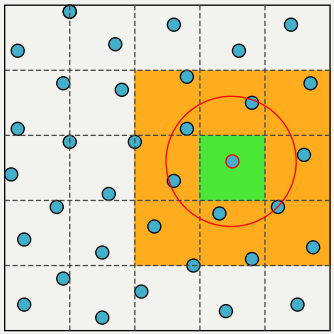


Task-based algorithms

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.
- If the particles in the cell or cell pair are sufficiently small, the task can be split.
- Finally, the particles in each cell pair are first sorted along the cell pair axis to speed-up neighbour-finding.

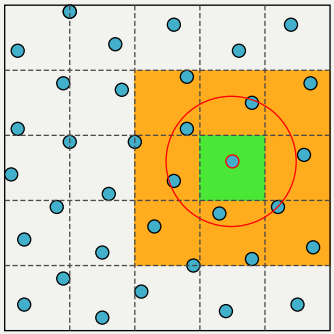


Task-based algorithms

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.
- If the particles in the cell or cell pair are sufficiently small, the task can be split.
- Finally, the particles in each cell pair are first sorted along the cell pair axis to speed-up neighbour-finding.

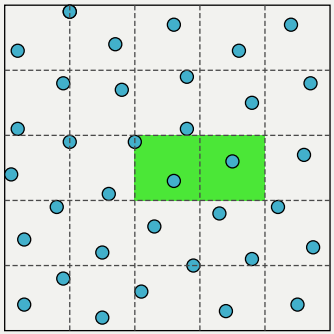


Task-based algorithms

Hierarchical cell pairs

Durham University

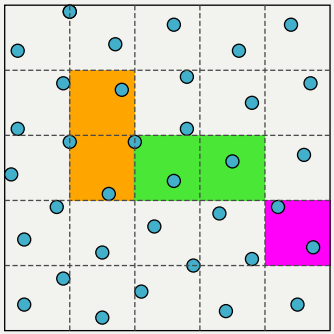
- We start by splitting the simulation domain into rectangular cells of edge length at least $h_{\max}$.
- All interacting particle pairs are then in either in the same cell, or in a pair of neighbouring cells.
- Finding all neighbours within each cell or between each pair of cells can be used as a task.
- If the particles in the cell or cell pair are sufficiently small, the task can be split.
- Finally, the particles in each cell pair are first sorted along the cell pair axis to speed-up neighbour-finding.



Task-based algorithms

Hierarchical cell pairs

- 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.
- If the particles in the cell or cell pair are sufficiently small, the task can be split.
- Finally, the particles in each cell pair are first sorted along the cell pair axis to speed-up neighbour-finding.

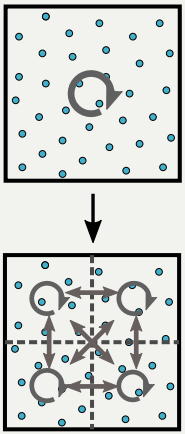


Task-based algorithms

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.
- If the particles in the cell or cell pair are sufficiently small, the task can be split.
- Finally, the particles in each cell pair are first sorted along the cell pair axis to speed-up neighbour-finding.

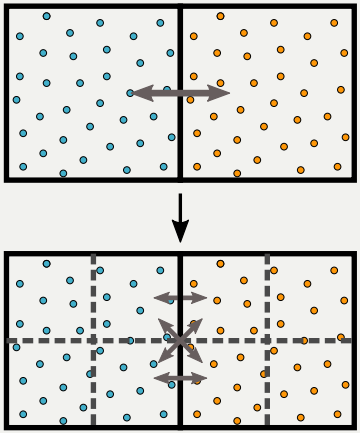


Task-based algorithms

Hierarchical cell pairs

Durham University

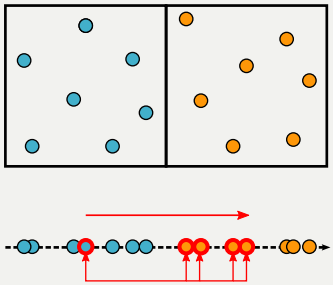
- We start by splitting the simulation domain into rectangular cells of edge length at least $h_{\max}$.
- All interacting particle pairs are then in either in the same cell, or in a pair of neighbouring cells.
- Finding all neighbours within each cell or between each pair of cells can be used as a task.
- If the particles in the cell or cell pair are sufficiently small, the task can be split.
- Finally, the particles in each cell pair are first sorted along the cell pair axis to speed-up neighbour-finding.



Task-based algorithms

Hierarchical cell pairs

- 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.
- If the particles in the cell or cell pair are sufficiently small, the task can be split.
- Finally, the particles in each cell pair are first sorted along the cell pair axis to speed-up neighbour-finding.

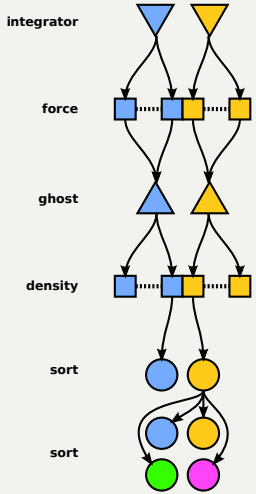


Task-based algorithms

Task hierarchy

Durham University

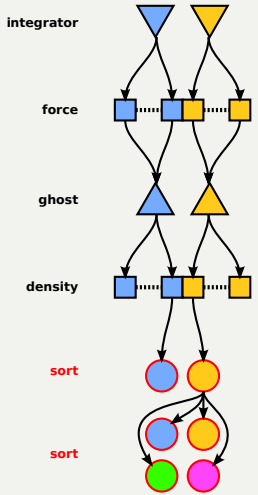
- Three main task types: Sorting, self-interactions, and pair-interactions.
- "Ghost" tasks are added to group dependencies between the density and force tasks of each cell.
- Each pair-interaction task depends on the sort tasks of the cells involved.
- Each sorting task depends on the sorting tasks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



Task-based algorithms

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 pair-interaction task depends on the sort tasks of the cells involved.
- Each sorting task depends on the sorting tasks of its sub-cells (merge-sort).
- Tasks on overlapping cells conflict, i.e. they can not execute concurrently.
- Finally, integrator tasks for each cell depend on the forces having been computed.

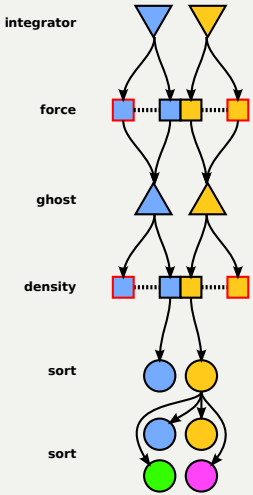


Task-based algorithms

Task hierarchy

Durham University

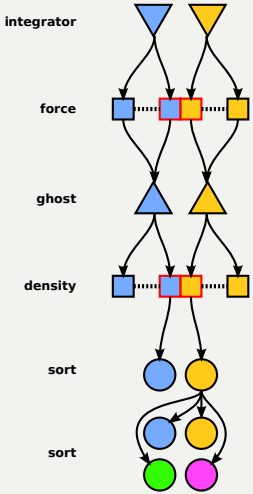
- Three main task types: Sorting, self-interactions, and pair-interactions.
- "Ghost" tasks are added to group dependencies between the density and force tasks of each cell.
- Each pair-interaction task depends on the sort tasks of the cells involved.
- Each sorting task depends on the sorting tasks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



Task-based algorithms

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 pair-interaction task depends on the sort tasks of the cells involved.
- Each sorting task depends on the sorting tasks of its sub-cells (merge-sort).
- Tasks on overlapping cells conflict, i.e. they can not execute concurrently.
- Finally, integrator tasks for each cell depend on the forces having been computed.

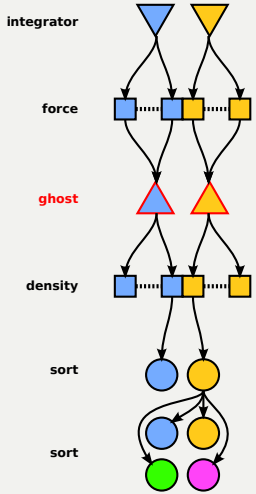


Task-based algorithms

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 pair-interaction task depends on the sort tasks of the cells involved.
- Each sorting task depends on the sorting tasks of its sub-cells (merge-sort).
- Tasks on overlapping cells conflict, i.e. they can not execute concurrently.
- Finally, integrator tasks for each cell depend on the forces having been computed.

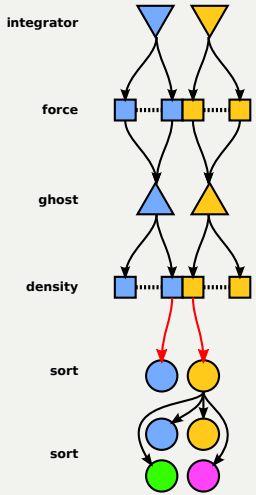


Task-based algorithms

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 pair-interaction task depends on the sort tasks of the cells involved.
- Each sorting task depends on the sorting tasks of its sub-cells (merge-sort).
- Tasks on overlapping cells conflict, i.e. they can not execute concurrently.
- Finally, integrator tasks for each cell depend on the forces having been computed.

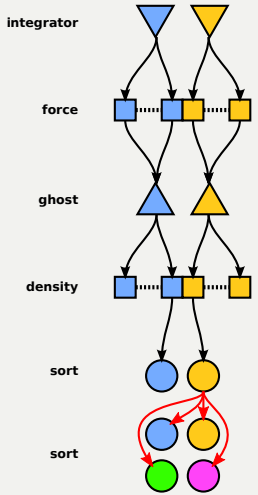


Task-based algorithms

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 pair-interaction task depends on the sort tasks of the cells involved.
- Each sorting task depends on the sorting tasks of its sub-cells (merge-sort).
- Tasks on overlapping cells conflict, i.e. they can not execute concurrently.
- Finally, integrator tasks for each cell depend on the forces having been computed.

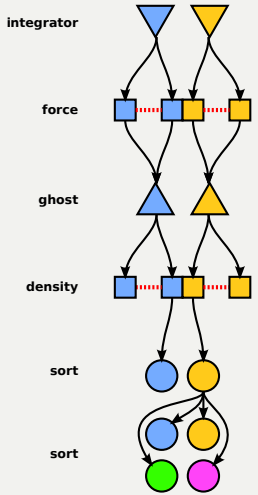


Task-based algorithms

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 pair-interaction task depends on the sort tasks of the cells involved.
- Each sorting task depends on the sorting tasks of its sub-cells (merge-sort).
- Tasks on overlapping cells conflict, i.e. they can not execute concurrently.
- Finally, integrator tasks for each cell depend on the forces having been computed.

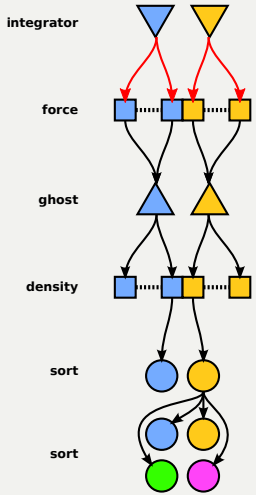


Task-based algorithms

Task hierarchy

Durham University

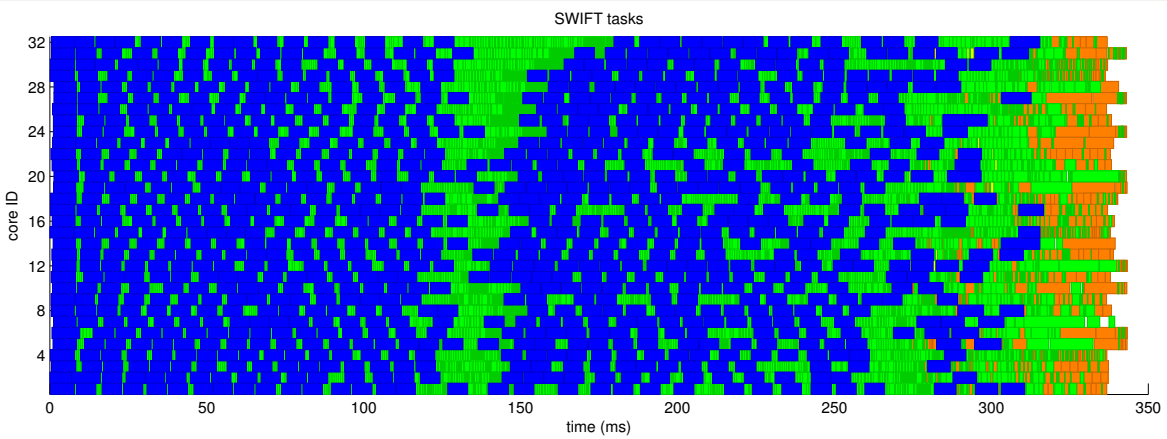
- Three main task types: Sorting, self-interactions, and pair-interactions.
- "Ghost" tasks are added to group dependencies between the density and force tasks of each cell.
- Each pair-interaction task depends on the sort tasks of the cells involved.
- Each sorting task depends on the sorting tasks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



Task-based algorithms

Task-based parallelism in action

Durham University



- Task execution for a single iteration of a 1 M-particle SPH simulation on 32 cores (4×8-core Intel E5-2670).

Hybrid parallelism using tasks

Distributed-memory parallelism

Durham University

- Three main problems:
  - Surface-to-volume ratio problem.
  - Load-balancing accross distributed-memory nodes.
  - Communication latencies between distributed-memory nodes.
- The first problem is implicitly attenuated by using a hybrid shared/distributed-memory parallel scheme.
- The second and third problem can be solved using task-based parallelism, i.e. exploiting the task/resource information.

- Three main problems:
  - Surface-to-volume ratio problem.
  - Load-balancing accross distributed-memory nodes.
  - Communication latencies between distributed-memory nodes.
- The first problem is implicitly attenuated by using a hybrid shared/distributed-memory parallel scheme.
- The second and third problem can be solved using task-based parallelism, i.e. exploiting the task/resource information.

Hybrid parallelism using tasks

Distributed-memory parallelism

Durham University

- Three main problems:
  - Surface-to-volume ratio problem.
  - Load-balancing accross distributed-memory nodes.
  - Communication latencies between distributed-memory nodes.
- The first problem is implicitly attenuated by using a hybrid shared/distributed-memory parallel scheme.
- The second and third problem can be solved using task-based parallelism, i.e. exploiting the task/resource information.

Hybrid parallelism using tasks

Distributed-memory parallelism

Durham University

- Three main problems:
  - Surface-to-volume ratio problem.
  - Load-balancing accross distributed-memory nodes.
  - Communication latencies between distributed-memory nodes.
- The first problem is implicitly attenuated by using a hybrid shared/distributed-memory parallel scheme.
- The second and third problem can be solved using task-based parallelism, i.e. exploiting the task/resource information.

- Three main problems:
  - Surface-to-volume ratio problem.
  - Load-balancing accross distributed-memory nodes.
  - Communication latencies between distributed-memory nodes.
- The first problem is implicitly attenuated by using a hybrid shared/distributed-memory parallel scheme.
- The second and third problem can be solved using task-based parallelism, i.e. exploiting the task/resource information.

Hybrid parallelism using tasks

Distributed-memory parallelism

Durham University

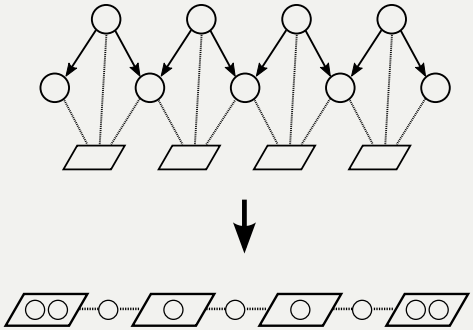
- Three main problems:
  - Surface-to-volume ratio problem.
  - Load-balancing accross distributed-memory nodes.
  - Communication latencies between distributed-memory nodes.
- The first problem is implicitly attenuated by using a hybrid shared/distributed-memory parallel scheme.
- The second and third problem can be solved using task-based parallelism, i.e. exploiting the task/resource information.

Hybrid parallelism using tasks

Task-based domain decomposition

Durham University

- The task DAG and resources can be converted to a weighted graph in which
  - Every resource is a node.
  - Evey task spanning more than one resource is an edge between the resources/nodes it uses.
- The weights for the nodes and edges are set to the computational cost of the tasks involved.
- Partitioning the graph corresponds to *partitioning the work*, and not just the data, involved in a computation.

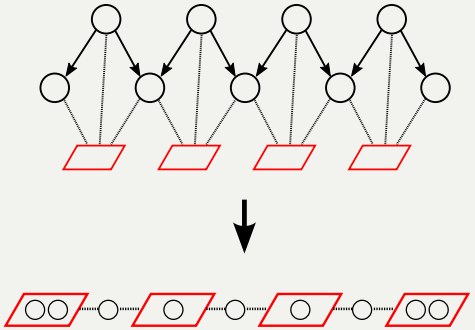


Hybrid parallelism using tasks

Task-based domain decomposition

Durham University

- The task DAG and resources can be converted to a weighted graph in which
  - Every resource is a node.
  - Evey task spanning more than one resource is an edge between the resources/nodes it uses.
- The weights for the nodes and edges are set to the computational cost of the tasks involved.
- Partitioning the graph corresponds to *partitioning the work*, and not just the data, involved in a computation.

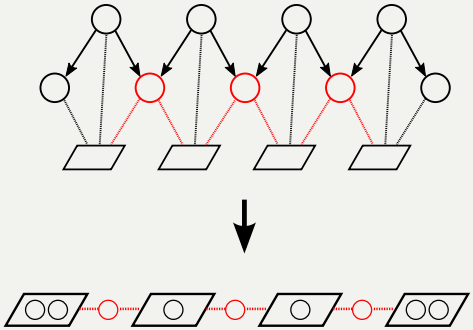


Hybrid parallelism using tasks

Task-based domain decomposition

Durham University

- The task DAG and resources can be converted to a weighted graph in which
  - Every resource is a node.
  - Evey task spanning more than one resource is an edge between the resources/nodes it uses.
- The weights for the nodes and edges are set to the computational cost of the tasks involved.
- Partitioning the graph corresponds to *partitioning the work*, and not just the data, involved in a computation.

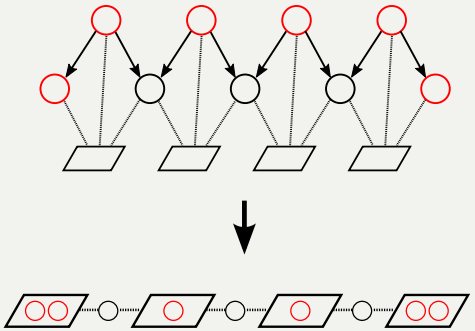


Hybrid parallelism using tasks

Task-based domain decomposition

Durham University

- The task DAG and resources can be converted to a weighted graph in which
  - Every resource is a node.
  - Evey task spanning more than one resource is an edge between the resources/nodes it uses.
- The weights for the nodes and edges are set to the computational cost of the tasks involved.
- Partitioning the graph corresponds to *partitioning the work*, and not just the data, involved in a computation.

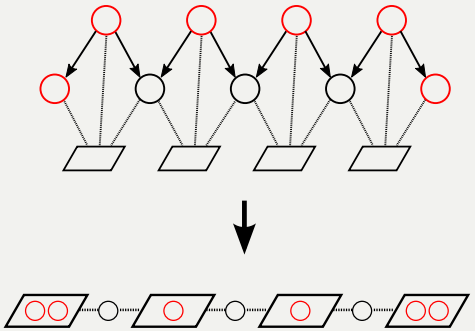


Hybrid parallelism using tasks

Task-based domain decomposition

Durham University

- The task DAG and resources can be converted to a weighted graph in which
  - Every resource is a node.
  - Evey task spanning more than one resource is an edge between the resources/nodes it uses.
- The weights for the nodes and edges are set to the computational cost of the tasks involved.
- Partitioning the graph corresponds to *partitioning the work*, and not just the data, involved in a computation.

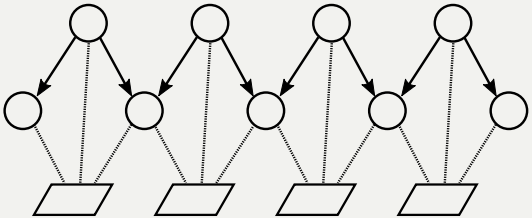


Hybrid parallelism using tasks

Task-based asynchronous communication

Durham University

- Tasks spanning the domain decomposition are duplicated and executed on both nodes.
- Each task distinguishes between *local* and *foreign* resources.
- Foreign resources need to be copied over before they can be used.
- Insert send/receive tasks and dependencies accross nodes for each foreign resource.
- All these tasks and dependencies can be created automatically.

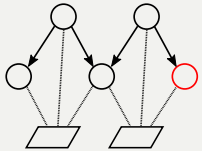


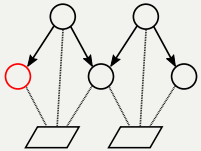
Hybrid parallelism using tasks

Task-based asynchronous communication

Durham University

- Tasks spanning the domain decomposition are duplicated and executed on both nodes.
- Each task distinguishes between *local* and *foreign* resources.
- Foreign resources need to be copied over before they can be used.
- Insert send/receive tasks and dependencies accross nodes for each foreign resource.
- All these tasks and dependencies can be created automatically.



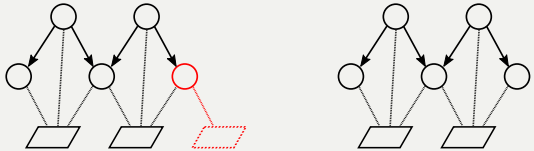


Hybrid parallelism using tasks

Task-based asynchronous communication

Durham University

- Tasks spanning the domain decomposition are duplicated and executed on both nodes.
- Each task distinguishes between *local* and *foreign* resources.
- Foreign resources need to be copied over before they can be used.
- Insert send/receive tasks and dependencies accross nodes for each foreign resource.
- All these tasks and dependencies can be created automatically.

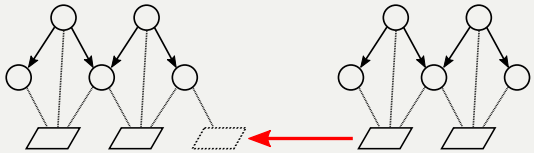


Hybrid parallelism using tasks

Task-based asynchronous communication

Durham University

- Tasks spanning the domain decomposition are duplicated and executed on both nodes.
- Each task distinguishes between *local* and *foreign* resources.
- Foreign resources need to be copied over before they can be used.
- Insert send/receive tasks and dependencies accross nodes for each foreign resource.
- All these tasks and dependencies can be created automatically.

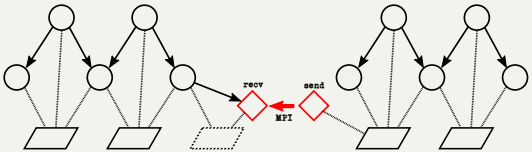


Hybrid parallelism using tasks

Task-based asynchronous communication

Durham University

- Tasks spanning the domain decomposition are duplicated and executed on both nodes.
- Each task distinguishes between *local* and *foreign* resources.
- Foreign resources need to be copied over before they can be used.
- Insert send/receive tasks and dependencies accross nodes for each foreign resource.
- All these tasks and dependencies can be created automatically.

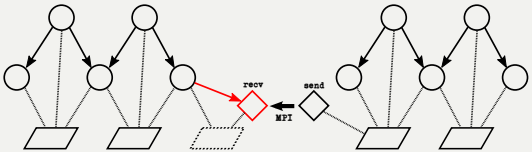


Hybrid parallelism using tasks

Task-based asynchronous communication

Durham University

- Tasks spanning the domain decomposition are duplicated and executed on both nodes.
- Each task distinguishes between *local* and *foreign* resources.
- Foreign resources need to be copied over before they can be used.
- Insert send/receive tasks and dependencies accross nodes for each foreign resource.
- All these tasks and dependencies can be created automatically.

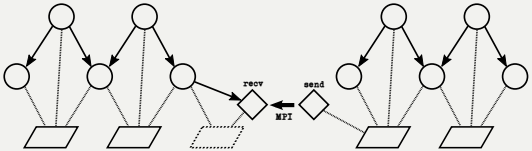


Hybrid parallelism using tasks

Task-based asynchronous communication

Durham University

- Tasks spanning the domain decomposition are duplicated and executed on both nodes.
- Each task distinguishes between *local* and *foreign* resources.
- Foreign resources need to be copied over before they can be used.
- Insert send/receive tasks and dependencies accross nodes for each foreign resource.
- All these tasks and dependencies can be created automatically.

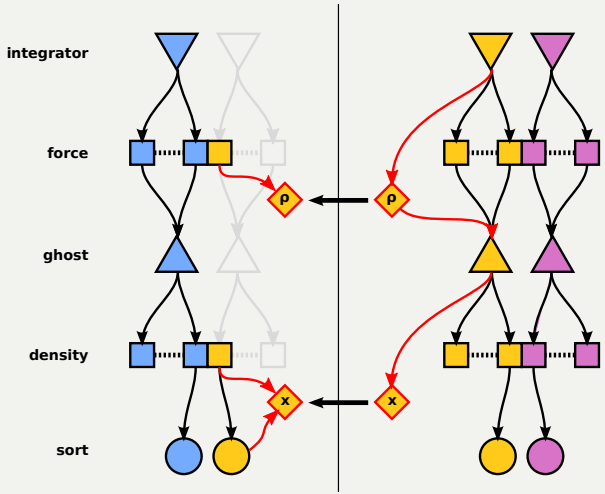


Hybrid parallelism using tasks

What this looks like in SWIFT

Durham University

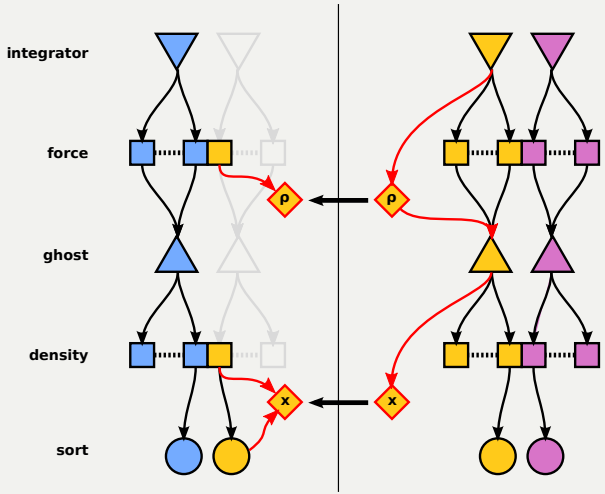
- In SWIFT, the domain decomposition happens along the cell edges, i.e. the particle cells are indidvidual resources.
- We have to copy the particle data twice:
  - Once to send the particle positions for the density computation,
  - Once to send the particle densities for the force computation.
- Two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



Hybrid parallelism using tasks

What this looks like in SWIFT

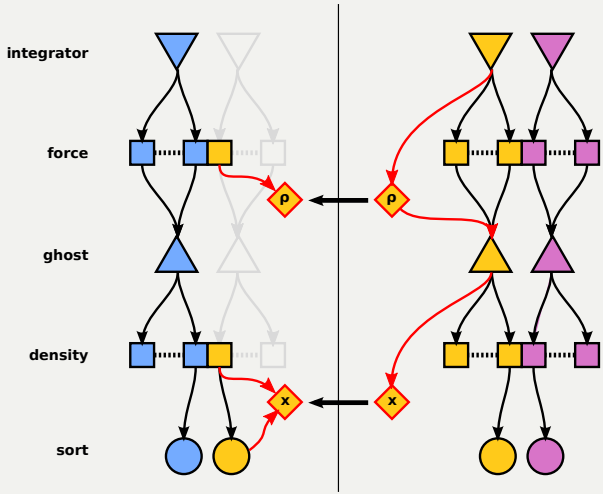
- In SWIFT, the domain decomposition happens along the cell edges, i.e. the particle cells are indidvidual resources.
- We have to copy the particle data twice:
  - Once to send the particle positions for the density computation,
  - Once to send the particle densities for the force computation.
- Two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



Hybrid parallelism using tasks

What this looks like in SWIFT

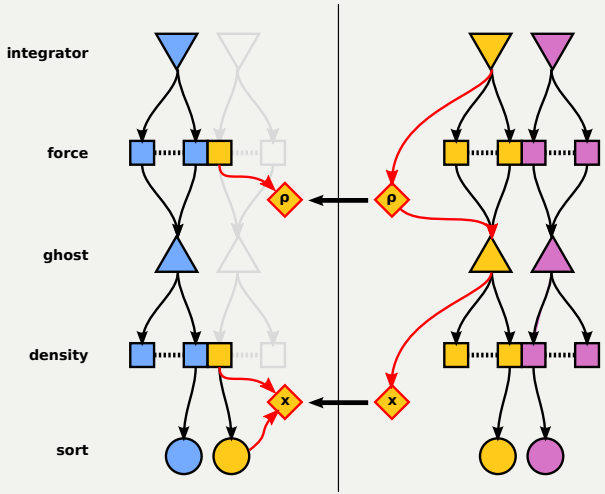
- In SWIFT, the domain decomposition happens along the cell edges, i.e. the particle cells are indidvidual resources.
- We have to copy the particle data twice:
  - Once to send the particle positions for the density computation,
  - Once to send the particle densities for the force computation.
- Two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



Hybrid parallelism using tasks

What this looks like in SWIFT

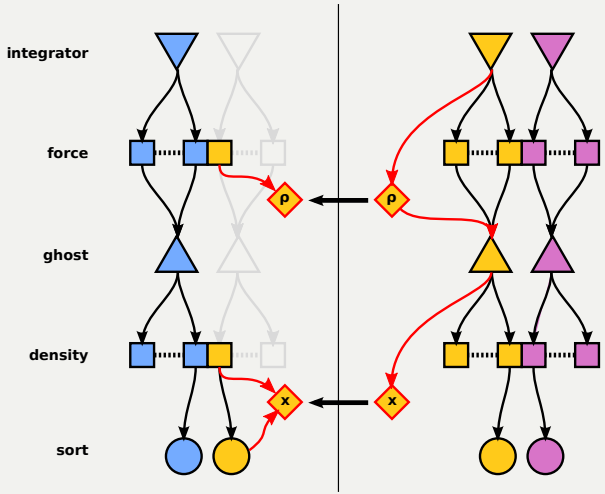
- In SWIFT, the domain decomposition happens along the cell edges, i.e. the particle cells are indidvidual resources.
- We have to copy the particle data twice:
  - Once to send the particle positions for the density computation,
  - Once to send the particle densities for the force computation.
- Two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



Hybrid parallelism using tasks

What this looks like in SWIFT

- In SWIFT, the domain decomposition happens along the cell edges, i.e. the particle cells are indidvidual 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.



Hybrid parallelism using tasks

What this looks like in SWIFT

Durham University

- Communication tasks do not perform any computation:
  - Call `MPI_Isend`/`MPI_Irecv` when enqueued.
  - Dependencies are released when `MPI_Test` says the data has been sent/received.
- While communication is happening, other strictly local tasks are executed.
  $\longrightarrow$ Truly asynchronous communication, latencies are completely masked by the computation.
- Slightly more complicated treatment for dependencies that span nodes, modelled by sending/receiving shared resources.

Hybrid parallelism using tasks

What this looks like in SWIFT

Durham University

- Communication tasks do not perform any computation:
  - Call `MPI_Isend`/`MPI_Irecv` when enqueued.
  - Dependencies are released when `MPI_Test` says the data has been sent/received.
- While communication is happening, other strictly local tasks are executed.
  $\longrightarrow$ Truly asynchronous communication, latencies are completely masked by the computation.
- Slightly more complicated treatment for dependencies that span nodes, modelled by sending/receiving shared resources.

Hybrid parallelism using tasks

What this looks like in SWIFT

Durham University

- Communication tasks do not perform any computation:
  - Call `MPI_Isend`/`MPI_Irecv` when enqueued.
  - Dependencies are released when `MPI_Test` says the data has been sent/received.
- While communication is happening, other strictly local tasks are executed.
  $\longrightarrow$ Truly asynchronous communication, latencies are completely masked by the computation.
- Slightly more complicated treatment for dependencies that span nodes, modelled by sending/receiving shared resources.

Hybrid parallelism using tasks

What this looks like in SWIFT

Durham University

- Communication tasks do not perform any computation:
  - Call `MPI_Isend`/`MPI_Irecv` when enqueued.
  - Dependencies are released when `MPI_Test` says the data has been sent/received.
- While communication is happening, other strictly local tasks are executed.
  $\longrightarrow$ Truly asynchronous communication, latencies are completely masked by the computation.
- Slightly more complicated treatment for dependencies that span nodes, modelled by sending/receiving shared resources.

Hybrid parallelism using tasks

What this looks like in SWIFT

Durham University

- Communication tasks do not perform any computation:
  - Call `MPI_Isend`/`MPI_Irecv` when enqueued.
  - Dependencies are released when `MPI_Test` says the data has been sent/received.
- While communication is happening, other strictly local tasks are executed.
  $\longrightarrow$ Truly asynchronous communication, latencies are completely masked by the computation.
- Slightly more complicated treatment for dependencies that span nodes, modelled by sending/receiving shared resources.

Hybrid parallelism using tasks

What this looks like in SWIFT

Durham University

- Communication tasks do not perform any computation:
  - Call `MPI_Isend`/`MPI_Irecv` when enqueued.
  - Dependencies are released when `MPI_Test` says the data has been sent/received.
- While communication is happening, other strictly local tasks are executed.
  $\longrightarrow$ Truly asynchronous communication, latencies are completely masked by the computation.
- Slightly more complicated treatment for dependencies that span nodes, modelled by sending/receiving shared resources.

Hybrid parallelism using tasks

What this looks like in SWIFT

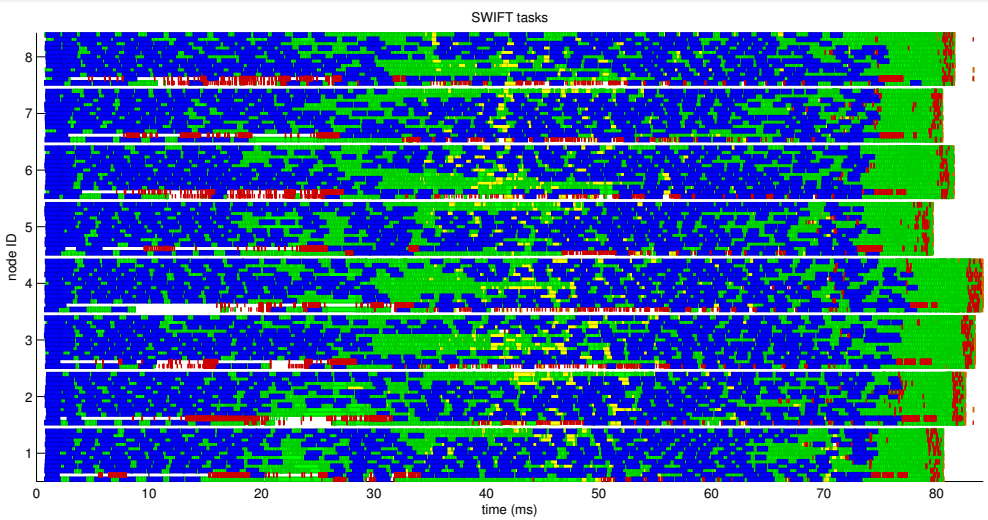
Durham University

- Communication tasks do not perform any computation:
  - Call `MPI_Isend`/`MPI_Irecv` when enqueued.
  - Dependencies are released when `MPI_Test` says the data has been sent/received.
- While communication is happening, other strictly local tasks are executed.
  $\longrightarrow$ Truly asynchronous communication, latencies are completely masked by the computation.
- Slightly more complicated treatment for dependencies that span nodes, modelled by sending/receiving shared resources.

Hybrid parallelism using tasks

What this looks like in SWIFT

Durham University



- 1 M particle SPH simulation using SWIFT on 8 $\times$ 12-core nodes of the COSMA4 cluster.

Hybrid parallelism using tasks

Forget what you've learned

Durham University

- Most experienced MPI users will advise against creating so many send/recv tasks.
- Since all communication is asynchronous, we don't really care about latencies.
- Spreading the communication throughout the computation actually reduces load on the network.

Hybrid parallelism using tasks

Forget what you've learned

Durham University

- Most experienced MPI users will advise against creating so many send/recv tasks.
- Since all communication is asynchronous, we don't really care about latencies.
- Spreading the communication throughout the computation actually reduces load on the network.

Hybrid parallelism using tasks

Forget what you've learned

Durham University

- Most experienced MPI users will advise against creating so many send/recv tasks.
- Since all communication is asynchronous, we don't really care about latencies.
- Spreading the communication throughout the computation actually reduces load on the network.

Durham University

- Platform-independent Open-Source library implementing the task-based parallel model and scheduler with conflicts described herein.
- Plain old C-language library built on top of either `pthreads` or OpenMP, no fancy language/compiler extensions needed.
- Task scheduling on CUDA GPUs with automatic generation of load/unload tasks and their dependencies.
- Under development: Fully automatic hybrid shared/distributed-memory parallelism.

Durham University

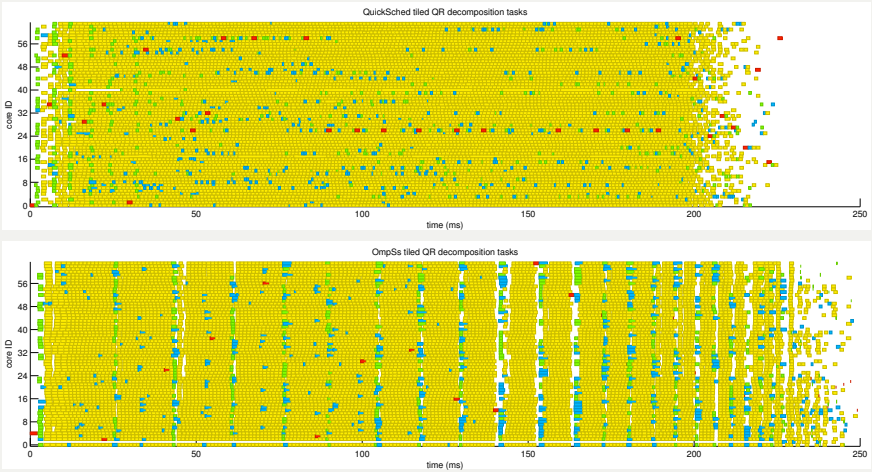
- Platform-independent Open-Source library implementing the task-based parallel model and scheduler with conflicts described herein.
- Plain old C-language library built on top of either `pthreads` or OpenMP, no fancy language/compiler extensions needed.
- Task scheduling on CUDA GPUs with automatic generation of load/unload tasks and their dependencies.
- Under development: Fully automatic hybrid shared/distributed-memory parallelism.

Durham University

- Platform-independent Open-Source library implementing the task-based parallel model and scheduler with conflicts described herein.
- Plain old C-language library built on top of either `pthreads` or OpenMP, no fancy language/compiler extensions needed.
- Task scheduling on CUDA GPUs with automatic generation of load/unload tasks and their dependencies.
- Under development: Fully automatic hybrid shared/distributed-memory parallelism.

Durham University

- Platform-independent Open-Source library implementing the task-based parallel model and scheduler with conflicts described herein.
- Plain old C-language library built on top of either `pthreads` or OpenMP, no fancy language/compiler extensions needed.
- Task scheduling on CUDA GPUs with automatic generation of load/unload tasks and their dependencies.
- Under development: Fully automatic hybrid shared/distributed-memory parallelism.



- Task scheduling in QuickSched (above) and OmpSs (below) for the QR decomposition of a 20482048 matrix on 64 cores.

Durham University

- Close collaboration with the Institute for Computational Cosmology (ICC) at Durham University.
  $\longrightarrow$ Make sure we're building a software that can actually be used.
- Main goal is to replace GADGET2, the most popular Open-Source cosmological simulation code.
  $\longrightarrow$ Is currently 40$\times$ faster than GADGET2.
- Massively multi-scale problems, with millions to billions of particles, run on both desktops and supercomputers.
  $\longrightarrow$ Include support for GPUs in order to take some of the moderate simulations off the cluster and onto desktop workstations.

Durham University

- Close collaboration with the Institute for Computational Cosmology (ICC) at Durham University.
  $\longrightarrow$ Make sure we're building a software that can actually be used.
- Main goal is to replace GADGET2, the most popular Open-Source cosmological simulation code.
  $\longrightarrow$ Is currently 40$\times$ faster than GADGET2.
- Massively multi-scale problems, with millions to billions of particles, run on both desktops and supercomputers.
  $\longrightarrow$ Include support for GPUs in order to take some of the moderate simulations off the cluster and onto desktop workstations.

Durham University

- Close collaboration with the Institute for Computational Cosmology (ICC) at Durham University.
  $\longrightarrow$ Make sure we're building a software that can actually be used.
- Main goal is to replace GADGET2, the most popular Open-Source cosmological simulation code.
  $\longrightarrow$ Is currently 40$\times$ faster than GADGET2.
- Massively multi-scale problems, with millions to billions of particles, run on both desktops and supercomputers.
  $\longrightarrow$ Include support for GPUs in order to take some of the moderate simulations off the cluster and onto desktop workstations.

Software
SWIFT

Durham University

- Close collaboration with the Institute for Computational Cosmology (ICC) at Durham University.
  $\longrightarrow$ Make sure we're building a software that can actually be used.
- Main goal is to replace GADGET2, the most popular Open-Source cosmological simulation code.
  $\longrightarrow$ Is currently 40$\times$ faster than GADGET2.
- Massively multi-scale problems, with millions to billions of particles, run on both desktops and supercomputers.
  $\longrightarrow$ Include support for GPUs in order to take some of the moderate simulations off the cluster and onto desktop workstations.

- Close collaboration with the Institute for Computational Cosmology (ICC) at Durham University.
  $\longrightarrow$ Make sure we're building a software that can actually be used.
- Main goal is to replace GADGET2, the most popular Open-Source cosmological simulation code.
  $\longrightarrow$ Is currently 40$\times$ faster than GADGET2.
- Massively multi-scale problems, with millions to billions of particles, run on both desktops and supercomputers.
  $\longrightarrow$ Include support for GPUs in order to take some of the moderate simulations off the cluster and onto desktop workstations.

- Close collaboration with the Institute for Computational Cosmology (ICC) at Durham University.
  $\longrightarrow$ Make sure we're building a software that can actually be used.
- Main goal is to replace GADGET2, the most popular Open-Source cosmological simulation code.
  $\longrightarrow$ Is currently 40$\times$ faster than GADGET2.
- Massively multi-scale problems, with millions to billions of particles, run on both desktops and supercomputers.
  $\longrightarrow$ Include support for GPUs in order to take some of the moderate simulations off the cluster and onto desktop workstations.

Durham University

- Close collaboration with the Institute for Computational Cosmology (ICC) at Durham University.
  $\longrightarrow$ Make sure we're building a software that can actually be used.
- Main goal is to replace GADGET2, the most popular Open-Source cosmological simulation code.
  $\longrightarrow$ Is currently 40$\times$ faster than GADGET2.
- Massively multi-scale problems, with millions to billions of particles, run on both desktops and supercomputers.
  $\longrightarrow$ Include support for GPUs in order to take some of the moderate simulations off the cluster and onto desktop workstations.

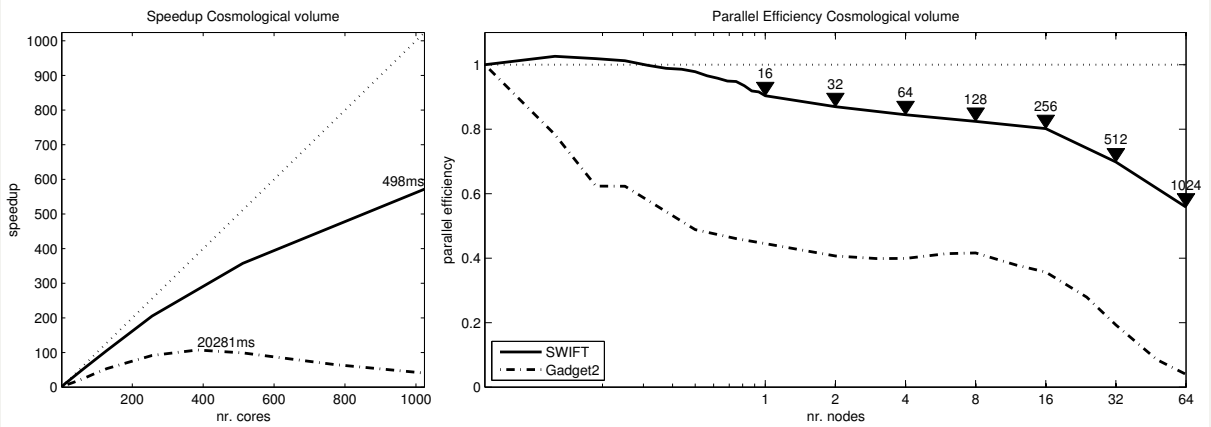
- Close collaboration with the Institute for Computational Cosmology (ICC) at Durham University.
  $\longrightarrow$ Make sure we're building a software that can actually be used.
- Main goal is to replace GADGET2, the most popular Open-Source cosmological simulation code.
  $\longrightarrow$ Is currently 40$\times$ faster than GADGET2.
- Massively multi-scale problems, with millions to billions of particles, run on both desktops and supercomputers.
  $\longrightarrow$ Include support for GPUs in order to take some of the moderate simulations off the cluster and onto desktop workstations.

Software
SWIFT

Durham University

- Close collaboration with the Institute for Computational Cosmology (ICC) at Durham University.
  $\longrightarrow$ Make sure we're building a software that can actually be used.
- Main goal is to replace GADGET2, the most popular Open-Source cosmological simulation code.
  $\longrightarrow$ Is currently 40$\times$ faster than GADGET2.
- Massively multi-scale problems, with millions to billions of particles, run on both desktops and supercomputers.
  $\longrightarrow$ Include support for GPUs in order to take some of the moderate simulations off the cluster and onto desktop workstations.

Software
SWIFT



- 51 M particle SPH simulation using SWIFT on 16 × 16-core nodes of the COSMA5 cluster, strong scaling compared to GADGET2.

Conclusions

Take-home messages

Conclusions
Take-home messages

Durham University

- Task-based parallelism provides good scaling for shared-memory parallel computations.
- More importantly, though, the task/resource decomposition provides an interesting representation of the computation.
- The task-based representation can be used to:
  - Compute domain decompositions that split the actual work, not just the data.
  - Automatically create asynchronous send/recv tasks for hybrid shared/distributed-memory parallel computations.

Conclusions

Take-home messages

Durham University

- Task-based parallelism provides good scaling for shared-memory parallel computations.
- More importantly, though, the task/resource decomposition provides an interesting representation of the computation.
- The task-based representation can be used to:
  - Compute domain decompositions that split the actual work, not just the data.
  - Automatically create asynchronous send/recv tasks for hybrid shared/distributed-memory parallel computations.

Conclusions

Take-home messages

Durham University

- Task-based parallelism provides good scaling for shared-memory parallel computations.
- More importantly, though, the task/resource decomposition provides an interesting representation of the computation.
- The task-based representation can be used to:
  - Compute domain decompositions that split the actual work, not just the data.
  - Automatically create asynchronous send/recv tasks for hybrid shared/distributed-memory parallel computations.

Conclusions
Take-home messages

Durham University

- Task-based parallelism provides good scaling for shared-memory parallel computations.
- More importantly, though, the task/resource decomposition provides an interesting representation of the computation.
- The task-based representation can be used to:
  - Compute domain decompositions that split the actual work, not just the data.
  - Automatically create asynchronous send/recv tasks for hybrid shared/distributed-memory parallel computations.

Thank you for your attention!