Task dependency graph in ocean model parallelisation

Piotr Piotrowski

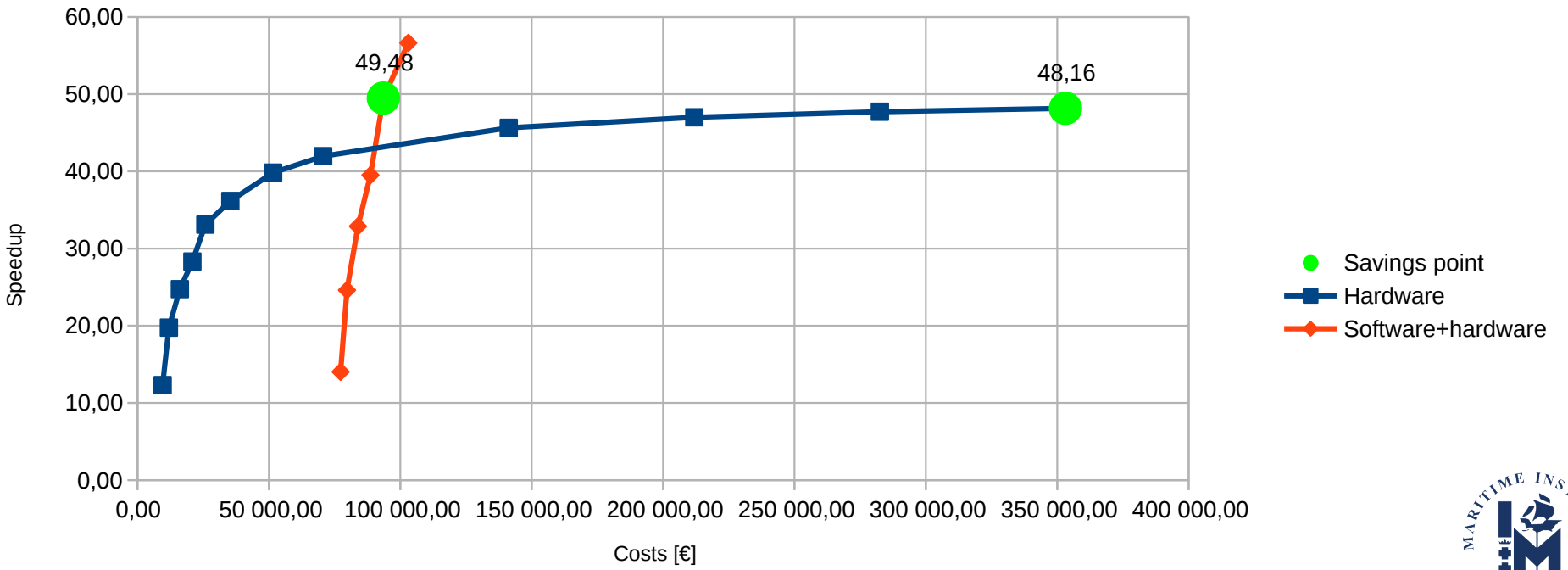
Maritime Institute in Gdańsk

Brussels 2018

How to save €250 000

Is your model written in Fortran + MPI + OpenMP?
Do you want to couple your models?

- Amdahl's law: there is a limit to improvement by adding resources
- Example: an ocean model with 98% parallelisable computations



How to improve software

Example:

- an ocean model with 98% parallelisable computations
- the most costly function, advection, constitutes 45% of computations
- the model runs on 64 CPUs

Possible improvements:

- double the number of processors
- optimise advection twofold
- optimise the unparallelisable 2% of computations twofold

Buy more processors

Before:

$$S = \frac{1}{(1-p)+\frac{p}{n}} = \frac{1}{(1-0.98)+\frac{0.98}{64}} \approx 28.32$$

After doubling the number of processors:

$$S' = \frac{1}{(1-p)+\frac{p}{n}} = \frac{1}{(1-0.98)+\frac{0.98}{2 \cdot 64}} \approx 36.16$$

Maximum speedup:

$$S_{max} = \lim_{n \to \infty} \frac{1}{(1-p)+\frac{p}{n}} = \frac{1}{(1-0.98)+\frac{0.98}{\infty}} = 50$$

S – speedup

p – parallelisable computations fraction

n – number of processors

Optimise the most computation intensive function

Before:

$$S = \frac{1}{(1-p)+\frac{p}{n}} = \frac{1}{(1-0.98)+\frac{0.98}{64}} \approx 28.32$$

After optimising the advection function twofold:

$$S' = \frac{1}{(1-p)+\frac{p}{n}} = \frac{1}{(1-0.98)+\frac{0.98-\frac{0.45}{2}}{64}} \approx 31.45$$

Maximum speedup (after buying more CPUs):

$$S_{max} = \lim_{n\to\infty} \frac{1}{(1-p)+\frac{p}{n}} = \frac{1}{(1-0.98)+\frac{0.98-\frac{0.45}{2}}{\infty}} = 50$$

S – speedup

p – parallelisable computations fraction

n – number of processors

Optimise the unparallelisable part

Before:

$$S = \frac{1}{(1-p)+\frac{p}{n}} = \frac{1}{(1-0.98)+\frac{0.98}{64}} \approx 28.32$$

After optimising the unparallelisable 2% of computations twofold:

$$S' = \frac{1}{(1-p)+\frac{p}{n}} = \frac{1}{\frac{1-0.98}{2}+\frac{0.98}{64}} \approx 39,51$$

Maximum speedup (after buying more CPUs):

$$S_{max} = \lim_{n\to\infty} \frac{1}{(1-p)+\frac{p}{n}} = \frac{1}{\frac{1-0.98}{2}+\frac{0.98}{\infty}} = 100$$

S – speedup

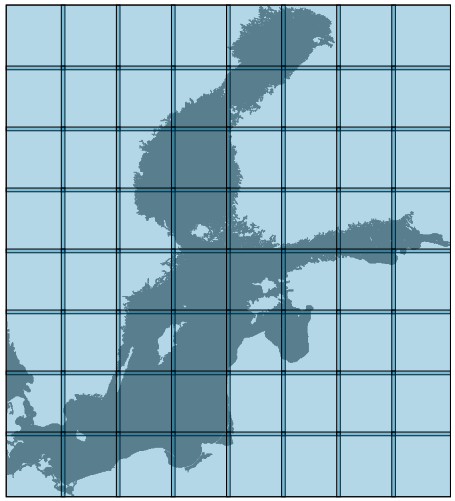
p – parallelisable computations fraction

n – number of processors

The unparallelisable part

The unparallelisable part consists of:

- halo region swaps
- communication
- workload imbalance
- barriers



The more complex the models is, the more communication it requires and the larger the unparallelisable part becomes.

“The Speedee Service System”

How to proceed?

The very same way the McDonald brothers when revolutionising fast food: they redesigned the kitchen and focused on the workflow efficiency.

Redesign your software and focus on computations workflow efficiency.

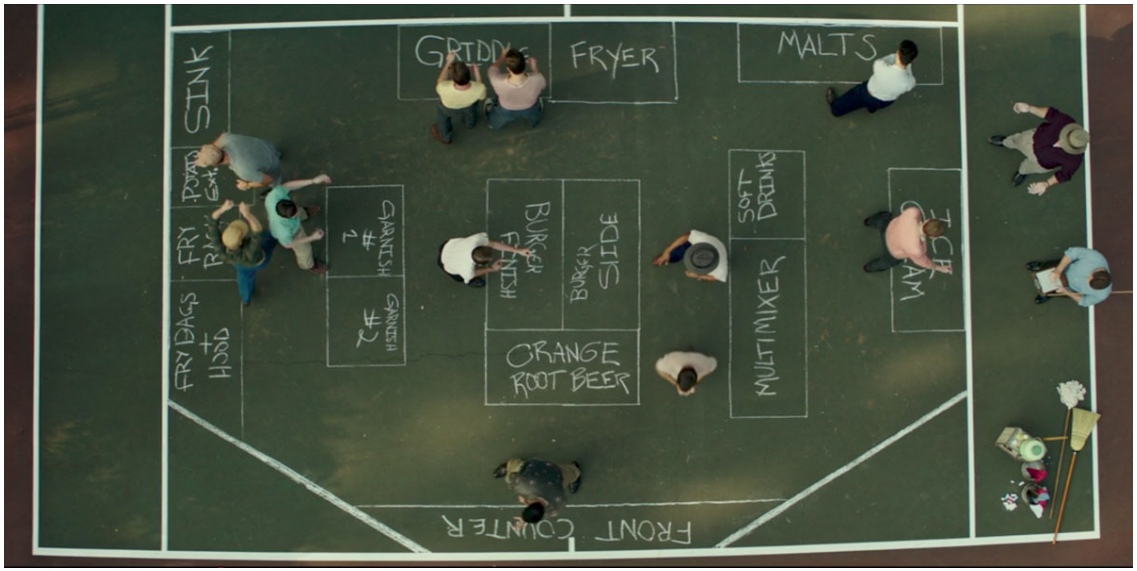


Figure: An image from the movie “The Founder” (2016)

MPI

- MPI was designed for distributed memory systems.
- Good performance in distributed memory systems.
- Inefficient use of shared memory systems.
- Two copy operations during communication.

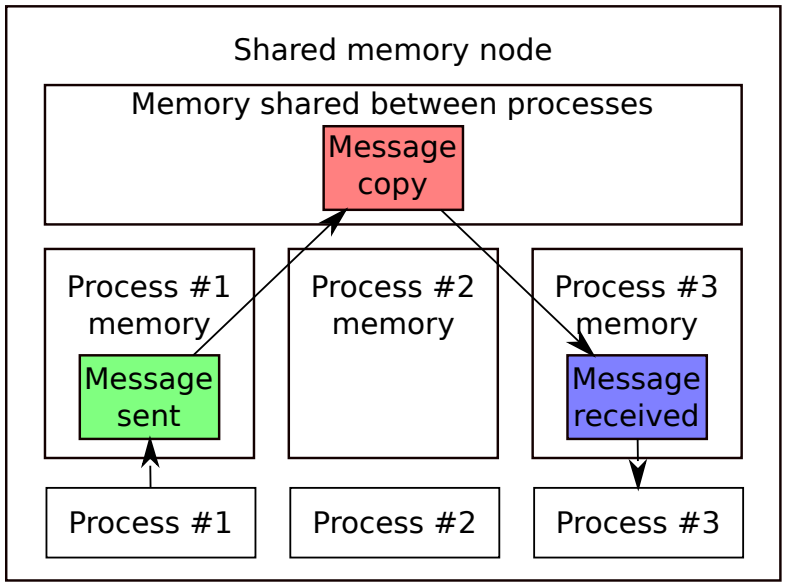


Figure: MPI communication

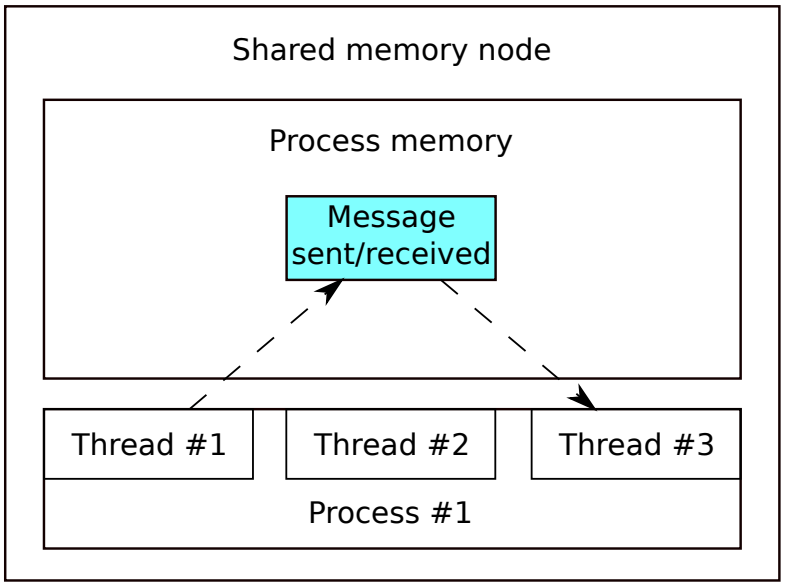
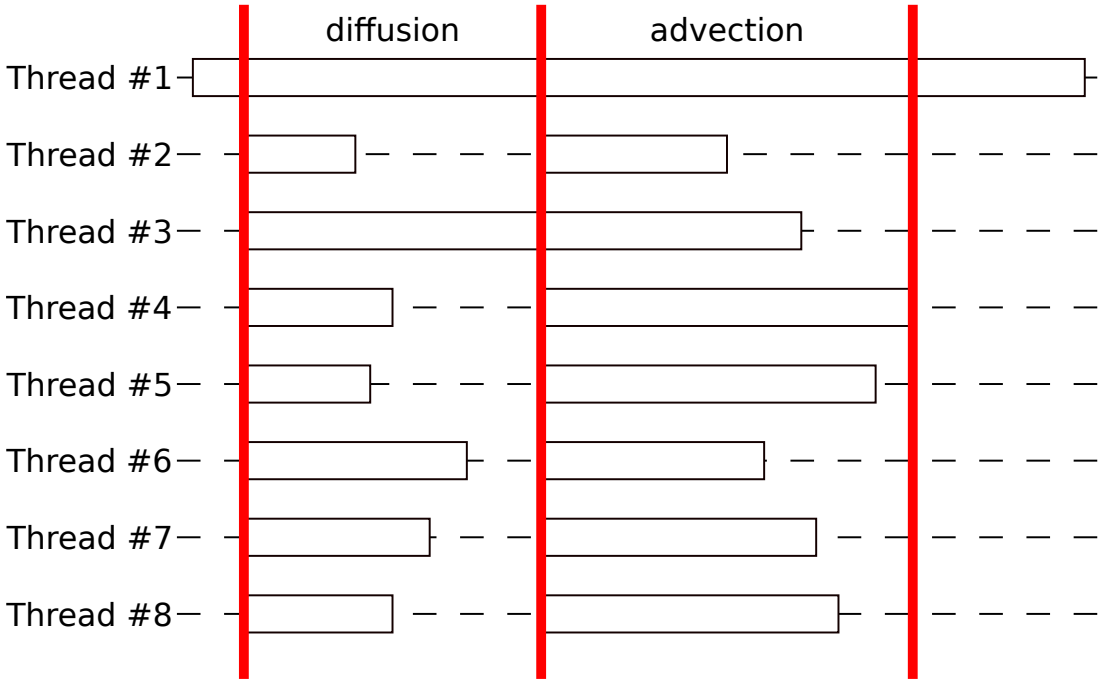


Figure: Thread communication (no MPI)

OpenMP

- OpenMP was designed for shared memory systems.
- Simple to use, but limited.
- Designed for data parallelism.
- Poor task based parallelism.
- Essentially a sequential application with some work offloading.



Distributed vs shared memory systems

Adapting old programs to fit new machines usually means adapting new machines to behave like old ones.

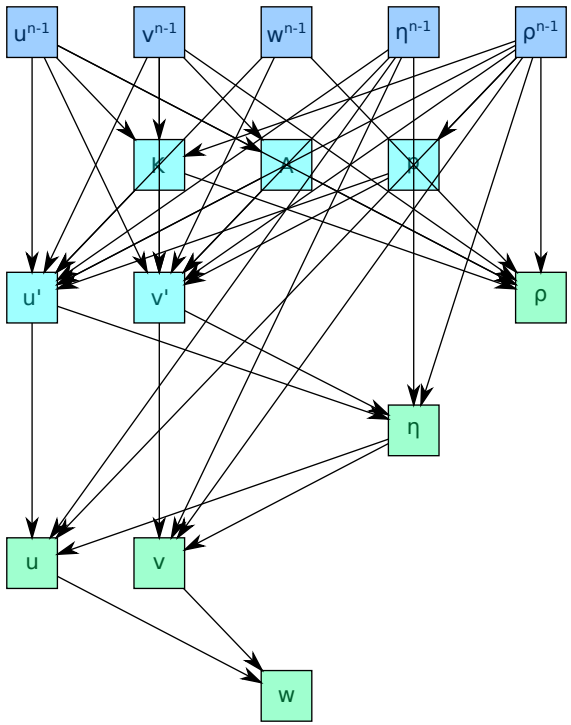
Perlis, A. J. Special Feature: Epigrams on programming. SIGPLAN Not. 17, 9 (1982), 7–13.

	Distributed memory	**Shared memory**
halo regions	yes	no
communication	by copying	zero-copy
partitioning	static	dynamic
task management	poor	advanced

A shared memory system used like a distributed memory system, is a more expensive distributed memory system with a slightly faster network.

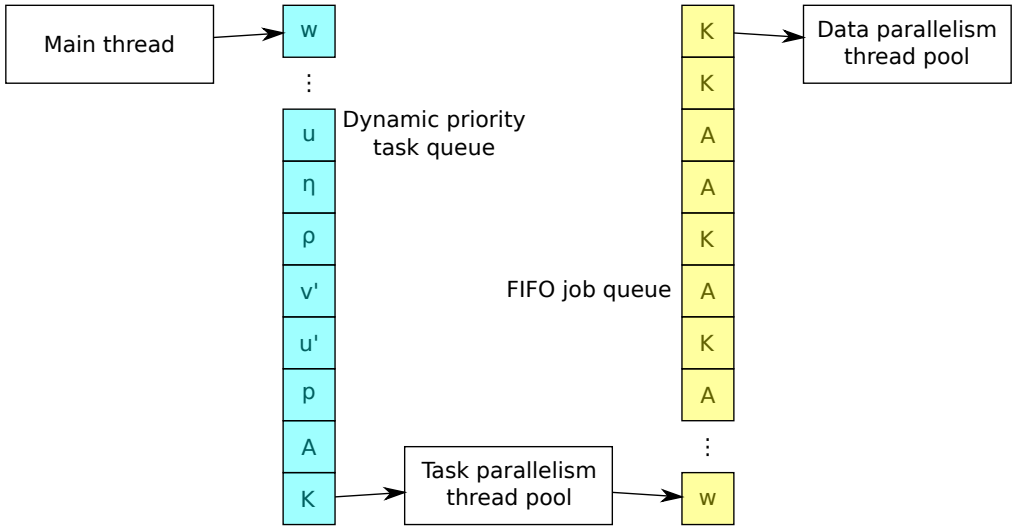
Task dependency graph

- Shows the order of computations
- Shows which computations can be done in parallel – several tasks can execute simultaneously
- A new task is started as soon as all prerequisites are computed
- Reduces CPU idle time by improving the computations workflow
- Improves CPU utilisation



Threads and queues

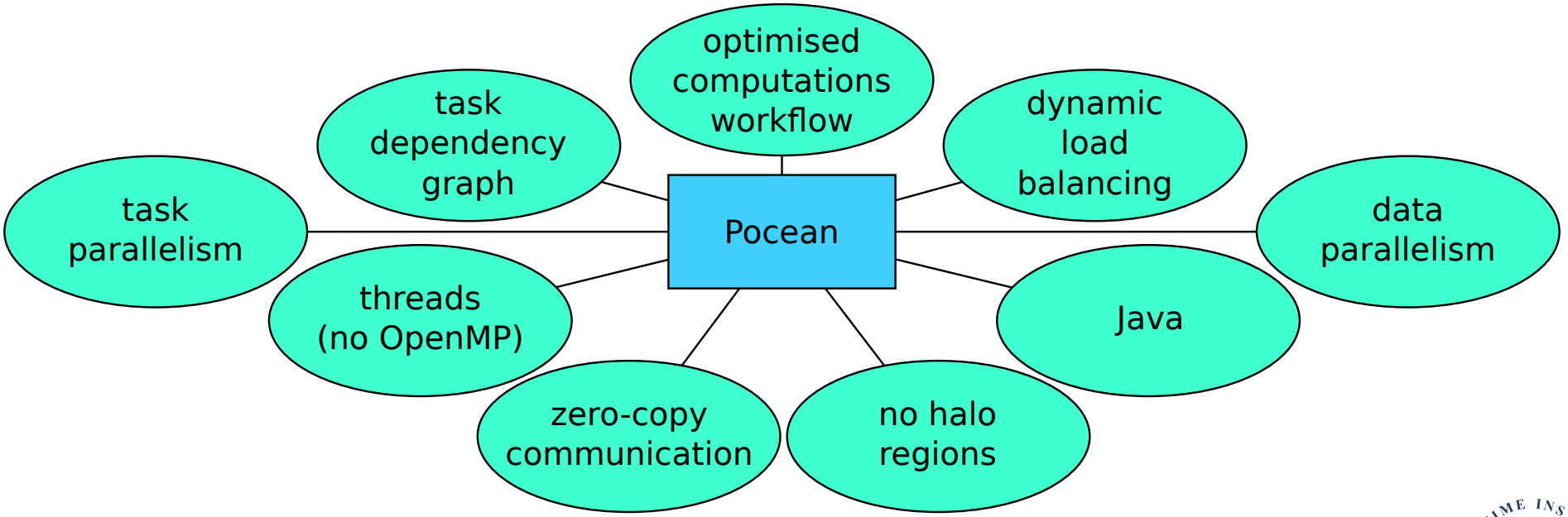
- The main thread constructs the dependency graph for all time steps.
- It inserts tasks into a dynamic priority queue.
- The task parallelism thread pool selects a task as soon as its prerequisites are done and executes the task.
- Each task divides computations and dispatches small jobs (e.g. computations of density for a single water line) to a FIFO queue.
- The data parallelism thread pool executes jobs from the FIFO queue.



Is this possible to implement?

Yes, it is possible to implement.

A proof of concept ocean model *Pocean* – a parallel 3D baroclinic circulation ocean model:



Any questions?