Numerical Techniques 2022–2023

7. Parallel computing and implications in NWP

Daan Degrauwe

daan.degrauwe@meteo.be

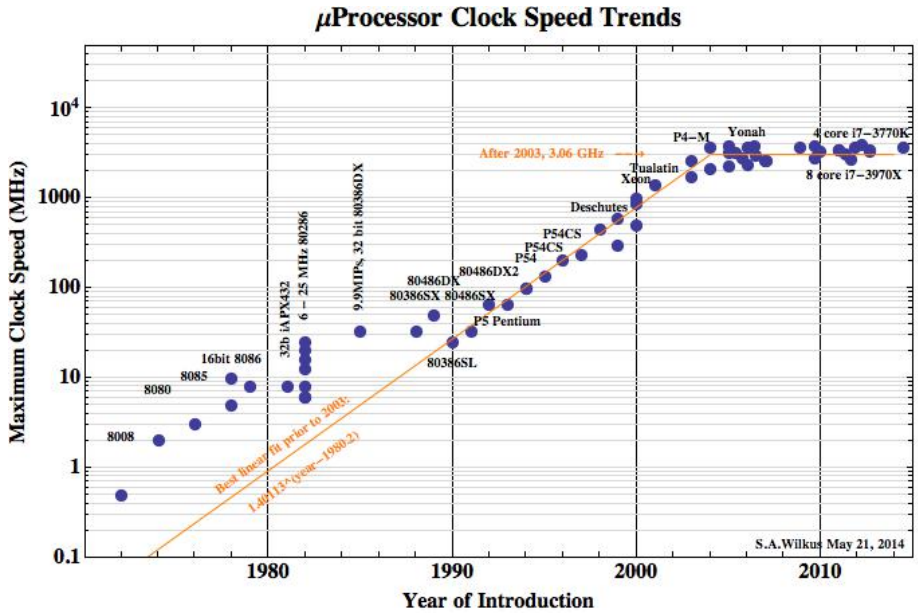
Postgraduate Studies in Weather and Climate Modeling

Ghent University

Content

- General aspects of parallel computing:
  - Introduction
  - Amdahl's law
  - Shared and distributed memory
- NWP-specificities
  - Design of an NWP model; 'physics' and 'dynamics'
  - Parallelization in physics
  - Parallelization in dynamics
  - The future?

- We want to run forecasts as quickly as possible. . .
- Until $\sim$ 2003, performance was increased by making processors *faster*



- It seems that increasing the clock frequency above 3GHz isn't beneficial (power consumption!)

- So we'll just use *more* processors instead of faster ones.
- For example: ECMWF's Cray machine

with 260'000 cores, or RMI's machine with 2688 cores

Unparallelizable problems

- Not everything goes faster when you use multiple processors!
- For instance: Newton-Raphson iterative method to find a root of a function $f(x)$:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

  x_i is required for the calculation of x_{i+1}, so the different iterations cannot be performed by different processors.

- IO (Input/Output), i.e. writing and reading to disk is another example.

- Consider a program consisting of two parts
  1. a part that cannot be parallelized: the *serial* part S. This part is responsible for a fraction σ of the total time T_1 on a single processor.
  2. a parallelizable part P. This parts is responsible for a fraction $1 - \sigma$ of the total time on a single processor.
- Then in a parallel environment with n processors, the total time will be

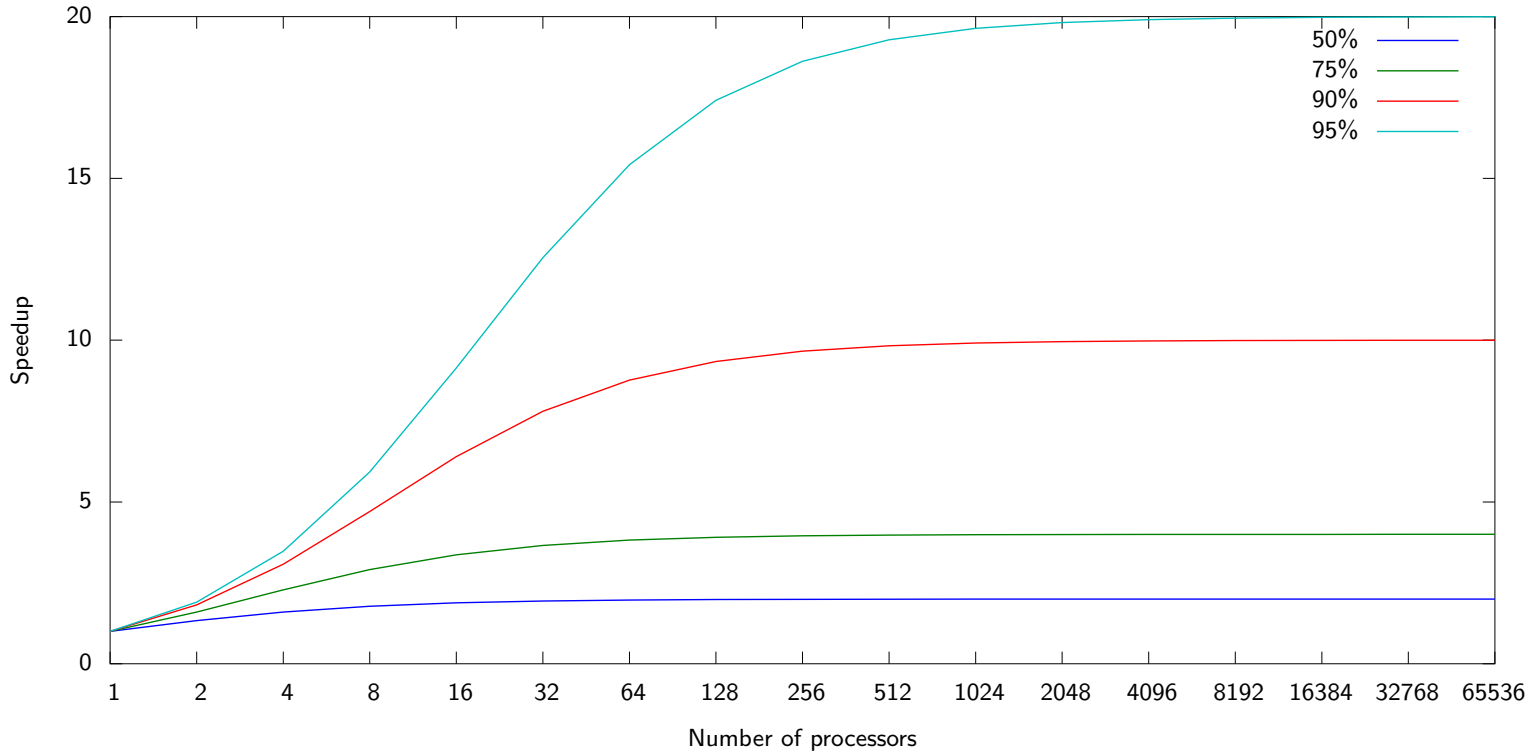
$$T_n = \sigma T_1 + \frac{1-\sigma}{n} T_1$$

So the speed-up is

$$\frac{T_1}{T_n} = \frac{1}{\sigma + \frac{1-\sigma}{n}}$$

Amdahl's law

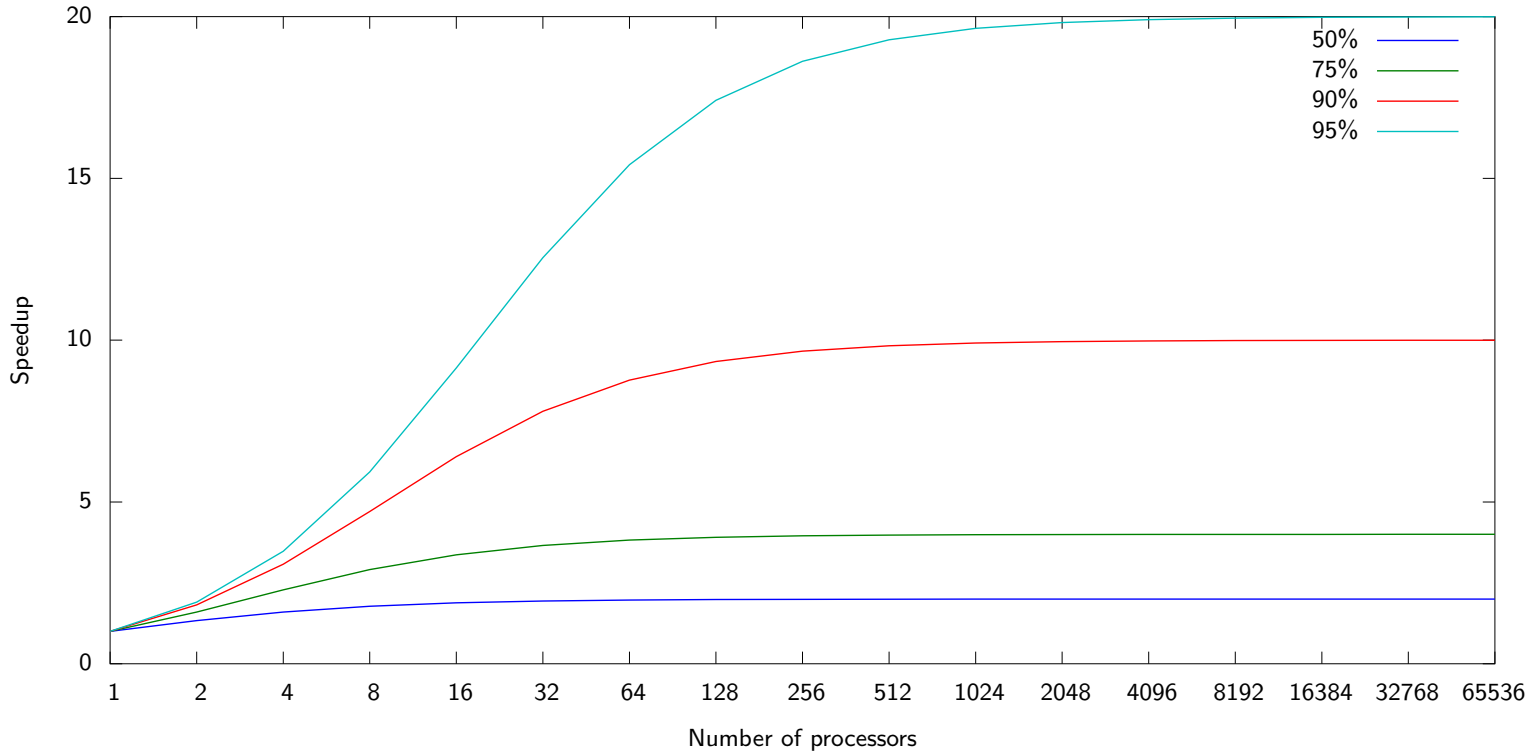
- Speedup as a function of the number of processors:



- So there's a limit to the speed-up that can be achieved by parallelizing! We call this the *scalability* of a program.

Amdahl's law

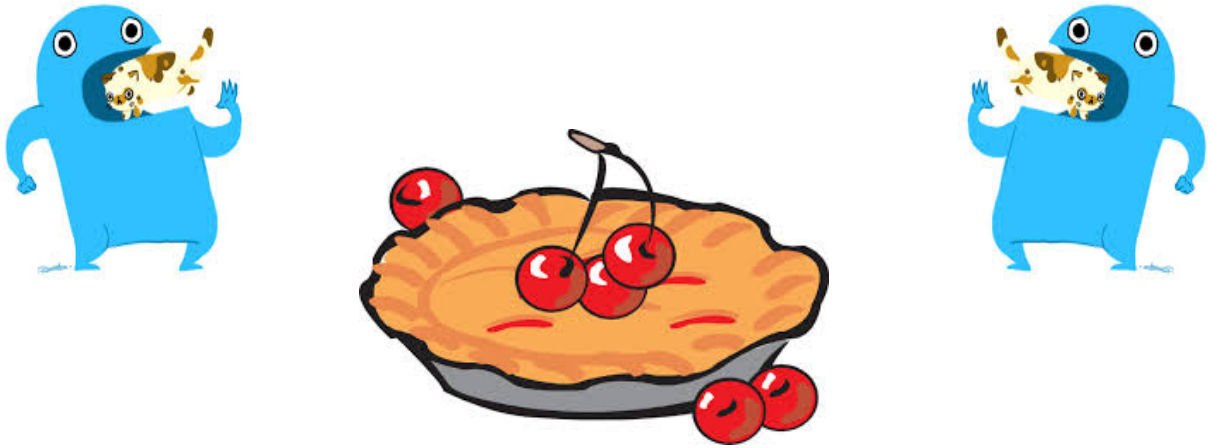
- Speedup as a function of the number of processors:



- So there's a limit to the speed-up that can be achieved by parallelizing! We call this the *scalability* of a program.
- But in fact, this isn't even the biggest challenge in NWP/Climate on high-performance systems.

Shared and distributed memory

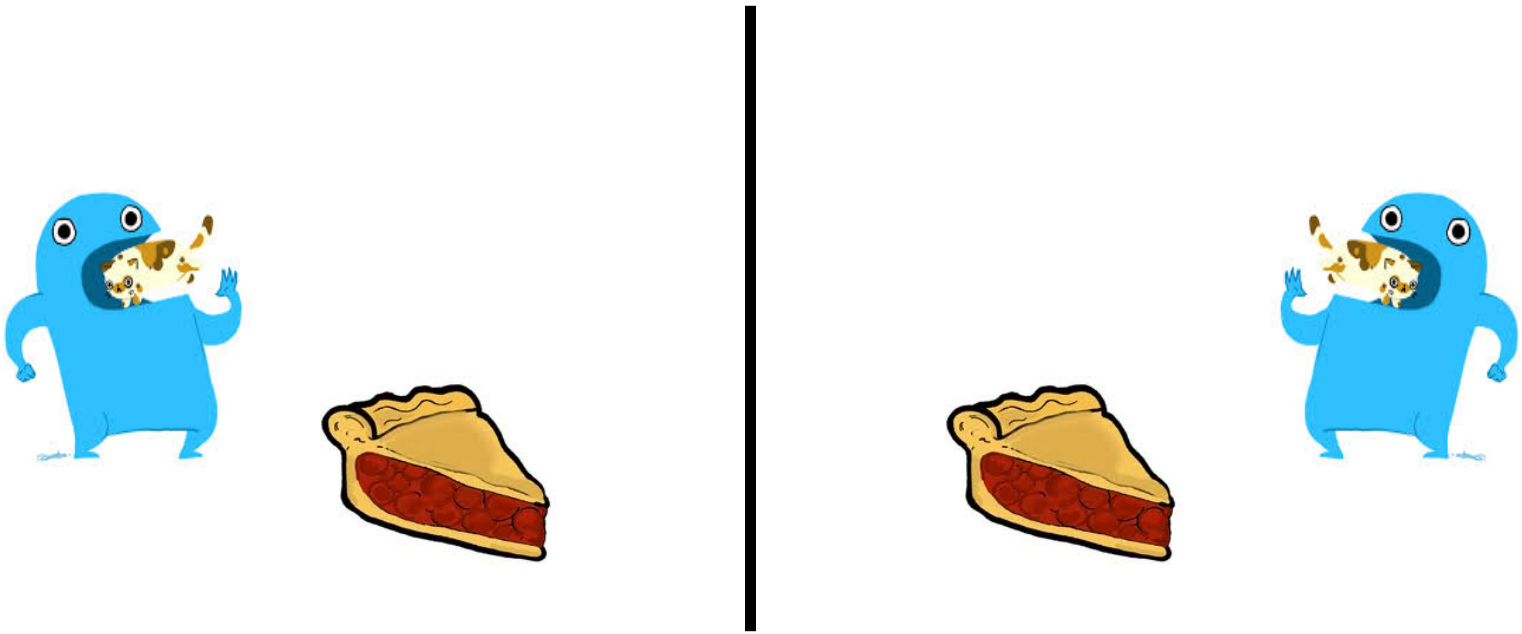
- Memory is the part of a computer where data are stored temporarily, right before/after the numerical operations.
- The most convenient setup is where all processors can access some central memory, and divide the work



- We call this a *shared* memory environment
- The standard for shared-memory parallelization is *OpenMP*

Shared and distributed memory

- Due to hardware constraints, shared memory becomes unfeasible for large systems
- In such systems, every processor is attributed his own piece of memory, and other processors cannot access it.

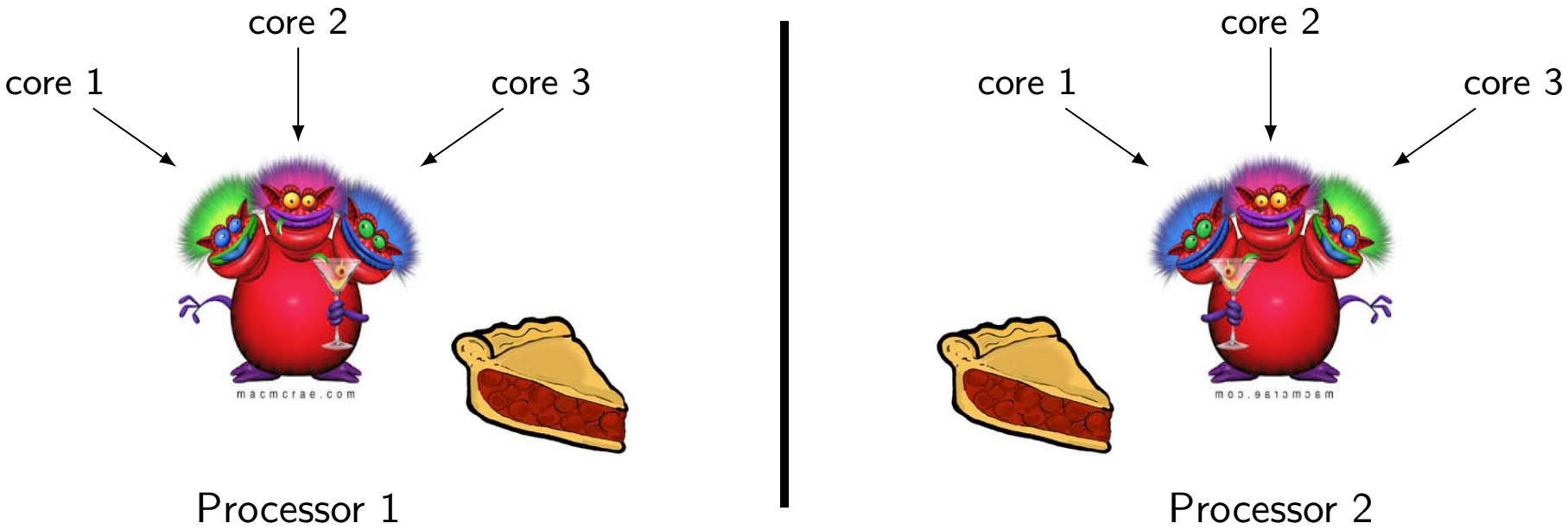


- We call this a *distributed* memory environment

Shared and distributed memory

- In a distributed memory environment, communication between processors has to be coded explicitly through *messages*.
- For instance:
  1. Processor 1 reads some data from a file;
  2. Processor 1 *sends* (pieces of) the data to the other processors $2 - n$;
  3. Processors $1 - n$ process their piece of data;
  4. Processors $2 - n$ *send* their result back to processor 1;
  5. Processor 1 writes the results to a file.
- The standard for distributed memory parallelization is *MPI* (Message Passing Interface).
- The cost of communications (slowly) *increases* with the number of processors:
  - number of messages increases
  - message sizes decrease
  - processors are further apart

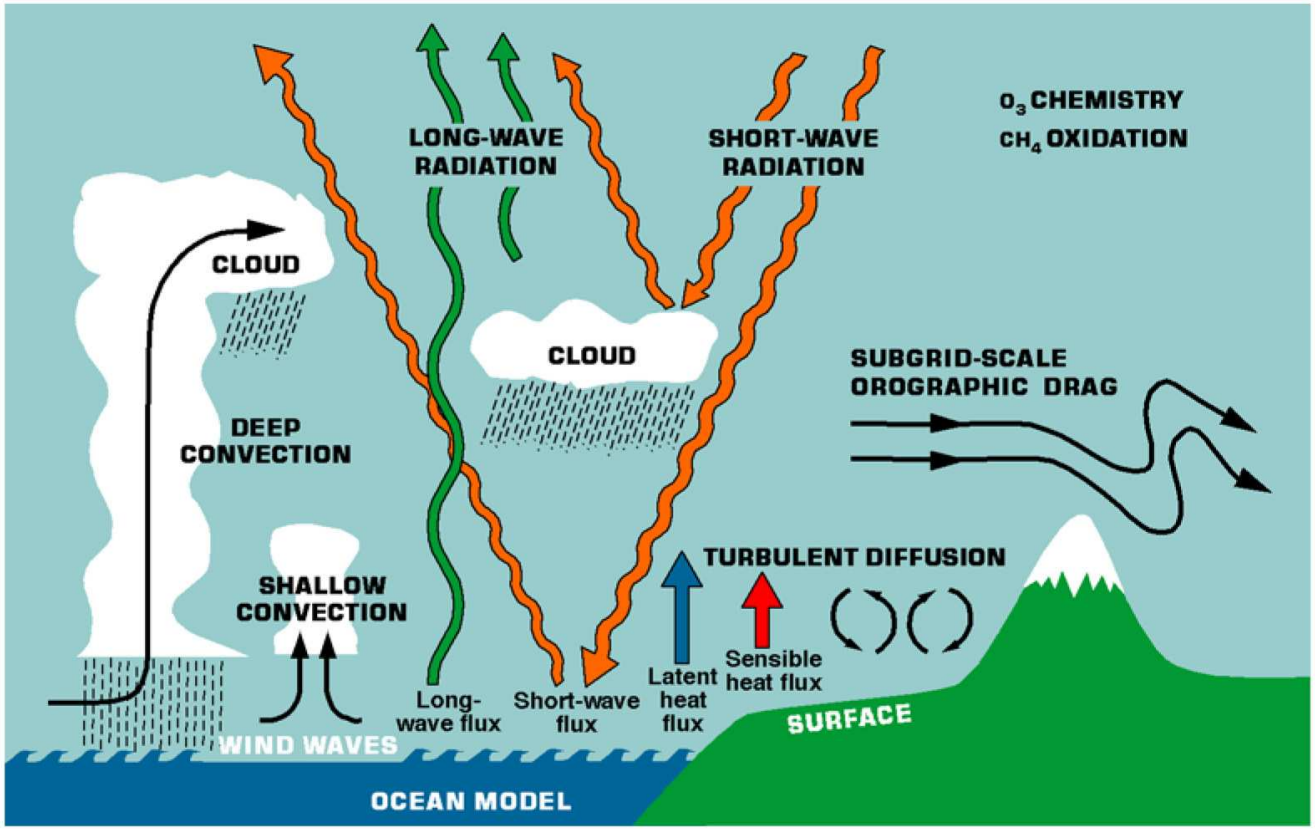
- Multicore processors allow for hybrid systems:
  - *within* a processor (i.e. between different cores of the same processor), memory is shared, and OpenMP is used for parallelization
  - *between* processors, memory is distributed, and MPI is used for communication



Design of an NWP model

- Almost all NWP models consist of a ‘dynamical part’ and a ‘physical part’.
- Dynamics (3D) are conservation of mass, energy, momentum; perfect gas law.
- The dynamics of the ALADIN/ARPEGE/IFS model are
  - spectral
  - semi-implicit
  - semi-Lagrangian
  - for LAM: Davies coupling to global model (ARPEGE/IFS)
  - terrain-following hybrid pressure coordinate
  - hydrostatic and nonhydrostatic

- The physics part models the *unresolved* or *subscale* phenomena: turbulence, radiation, diffusion, clouds, precipitation, surface, ...

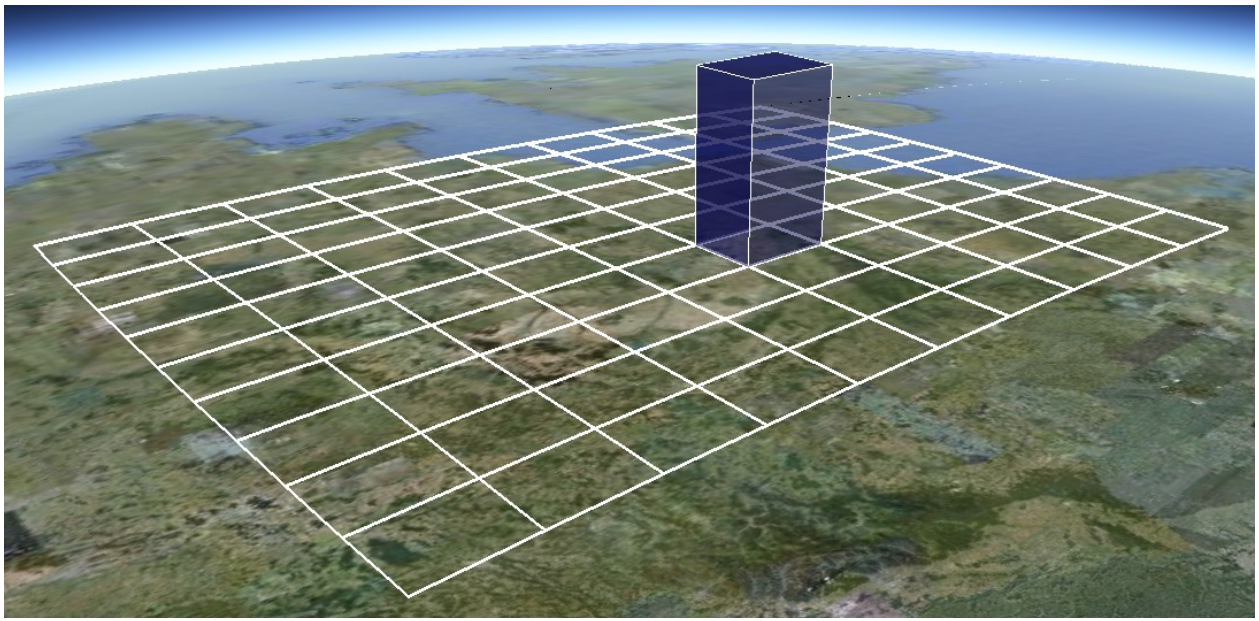


A single timestep of the spectral ALADIN model looks like:

1. calculate derivatives in spectral space
2. apply explicit part of *dynamics*
3. Fourier transform to gridpoint space
4. semi-Lagrangian interpolations and nonlinear dynamical terms
5. calculate *physics* contributions to energy and momentum equations
6. apply *boundary conditions*
7. Fourier transform to spectral space
8. solve implicit operator (*dynamics*)

Parallelization in the physics

- Most physics phenomena act mainly in the vertical (e.g. precipitation, stability, ...)
- The physics are computed in vertical columns



- There's no interaction (so no communication!) between the columns
- So the physics scale very well (quasi-perfectly!).
- (the column-approach may become problematic for very high resolutions)

Dynamics parallelization: spectral dynamics

- The spectral dynamics are parallelized by distributing the wavelengths over the processors.
- The Fourier transforms are quite communication-intensive!

$$f(x_j) = \sum_{k=-N/2}^{k=N/2} \alpha_k \exp(ikx_j)$$

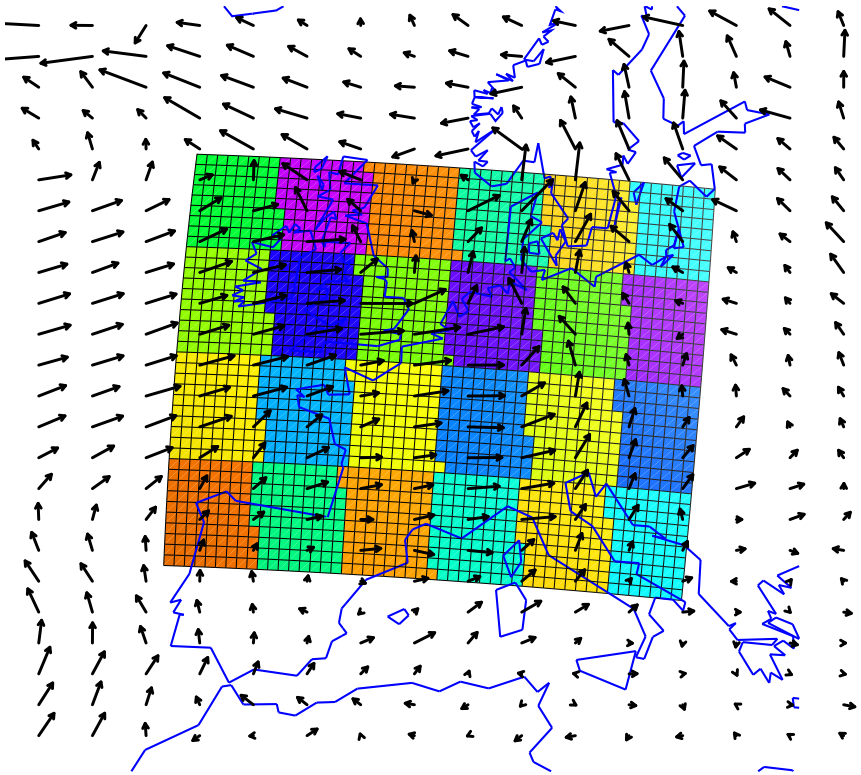
$$\alpha_k = \frac{1}{N} \sum_{j=1}^{N} f(x_j) \exp(-ikx_j)$$

  So a wave amplitude depends on *all* gridpoints, and vice versa; a gridpoint value is the combination of *all* composing waves.

- We say that the Fourier transform is *not local* (as opposed to finite differences)
- **Note**: this nonlocality is also the key to the high accuracy of spectral methods.

Dynamics parallelization: semi-Lagrangian advection

- Semi-Lagrangian calculations require interaction with nearby processors: the trajectory departure points may well belong to another processor.



Dynamics parallelization: implicit timestepping

- Implicit timestepping is also not local, even when using finite differences.
- For example, the trapezium scheme for the advection equation

$$\frac{\phi_j^{n+1} - \phi_j^n}{\Delta t} + \frac{c}{2}\left(\frac{\phi_{j+1}^{n+1} - \phi_{j-1}^{n+1}}{2\Delta x} + \frac{\phi_{j+1}^{n} - \phi_{j-1}^{n}}{2\Delta x}\right) = 0$$

leads to a tridiagonal linear system that needs to be solved every timestep:

$$\mathbf{T}\phi^{n+1} = \mathbf{R}$$

where $\mathbf{T}$ is a tridiagonal matrix, and $\mathbf{R}$ is the right-hand side (depending on ϕ^n).

Dynamics parallelization: implicit timestepping

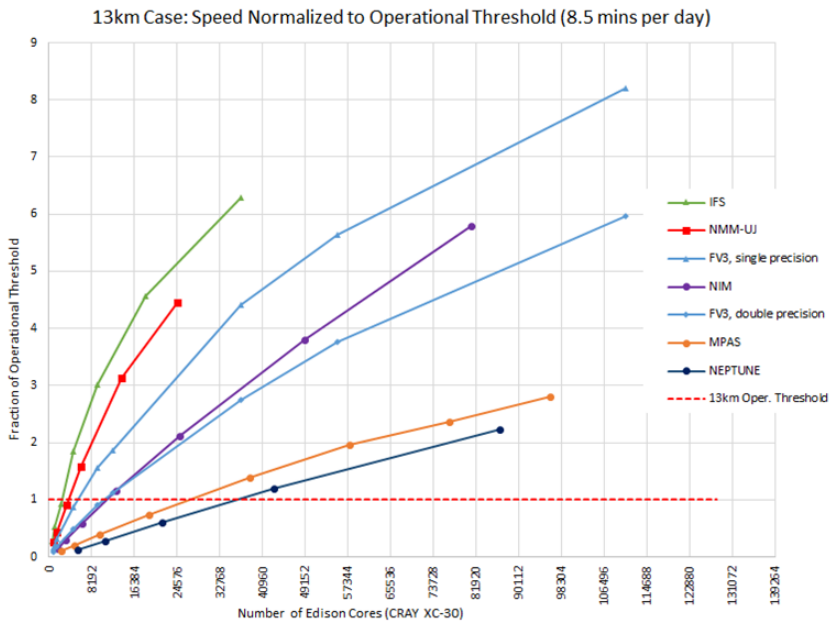
- So the next timestep's solution is obtained as

$$\phi^{n+1} = \mathbf{T}^{-1}\mathbf{R}$$

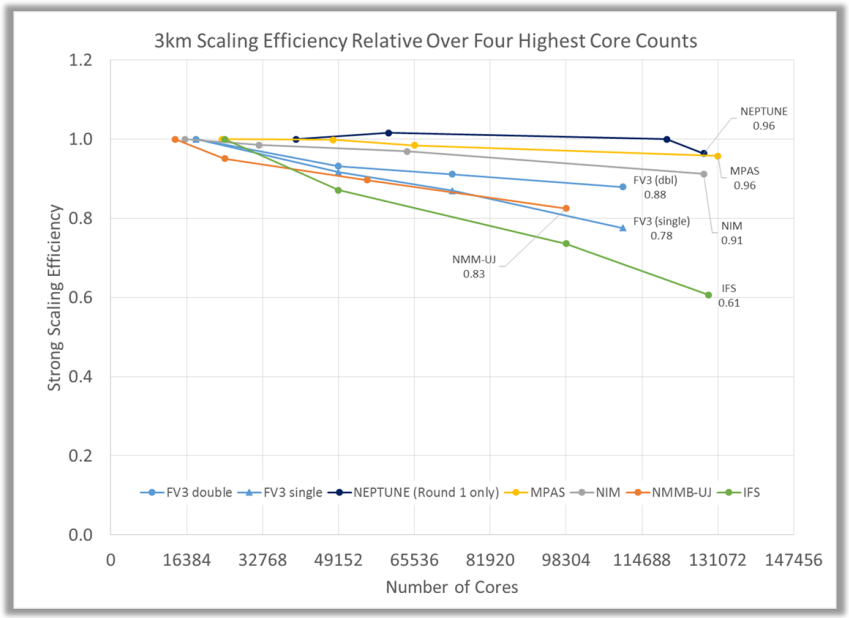
- But the inverse of a tridiagonal matrix is a full matrix!
- So the solution in a point at $t + \Delta t$ depends on the values of *all* gridpoints at t, meaning you need communications between all processors.

The future?

- The efficiency of a scheme is not the only criterion.
- Although spectral semi-implicit semi-Lagrangian is quite efficient (since it allows very large timesteps and has a good accuracy), it doesn't scale very well



IFS speed



IFS scaling

The future?

- As machines get larger, there's a trend towards methods requiring less communications:
  - spectral $\rightarrow$ finite differences
  - implicit (trapezium) $\rightarrow$ explicit (Runge-Kutta)
  - Lagrangian $\rightarrow$ Eulerian
- Maybe intermediate solutions such as HEVI (Horizontal Explicit, Vertical Implicit) are the future?
- There's always a tradeoff between accuracy and computational cost
- Power consumption may become a bigger challenge than communications