Exascale algorithms for numerical weather and climate prediction

Jemma Shipton

9th May 2025

Scientific computing

Scientific computing

Scientific Computing is the collection of tools, techniques, and theories required to solve on a computer mathematical models of problems in Science and Engineering. A majority of these tools, techniques, and theories originally developed in Mathematics, many of them having their genesis long before the advent of electronic computers. This set of mathematical theories and techniques is called Numerical Analysis (or Numerical Mathematics) and constitutes a major part of scientific computing. The development of the electronic computer, however, signaled a new era in the approach to the solution of scientific problems. Many of the numerical methods that had been developed for the purpose of hand calculation (including the use of desk calculators for the actual arithmetic) had to be revised and sometimes abandoned. Considerations that were irrelevant or unimportant for hand calculation now became of utmost importance for the efficient and correct use of a large Computer System. Many of these considerations – programming languages, operating systems, management of large quantities of data, correctness of programs – were subsumed under the new discipline of Computer Science, on which scientific computing now depends heavily. But mathematics itself continues to play a major role in scientific computing: it provides the language of the mathematical models that are to be solved and information about the suitability of a model (Does it have a solution? Is the solution unique?) and it provides the theoretical foundation for the numerical methods and, increasingly, many of the tools from computer science. In summary, then, scientific computing draws on mathematics and computer science to develop the best way to use computer systems to solve problems from science and engineering.

Gene H. Golub and James M. Ortega. Scientific Computing and Differential Equations – An Introduction to Numerical Methods. Academic Press, 1992

Scientific computing

- Scientific Computing is the collection of tools, techniques, and theories required to solve on a computer mathematical models of problems in Science and Engineering.

Scientific computing

- Scientific Computing is the collection of tools, techniques, and theories required to solve on a computer mathematical models of problems in Science and Engineering.
- The development of the electronic computer signaled a new era in the approach to the solution of scientific problems.

Scientific computing

- Scientific Computing is the collection of tools, techniques, and theories required to solve on a computer mathematical models of problems in Science and Engineering.
- The development of the electronic computer signaled a new era in the approach to the solution of scientific problems.
- Considerations that were irrelevant or unimportant for hand calculation now became of utmost importance for the efficient and correct use of a large Computer System.

Scientific computing

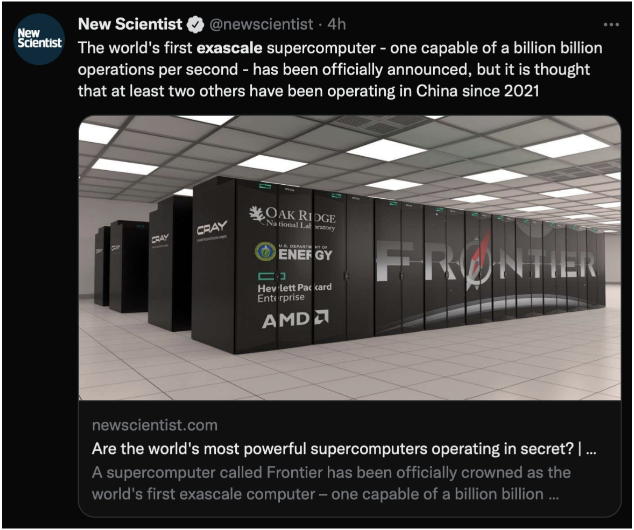
- Scientific Computing is the collection of tools, techniques, and theories required to solve on a computer mathematical models of problems in Science and Engineering.
- The development of the electronic computer signaled a new era in the approach to the solution of scientific problems.
- Considerations that were irrelevant or unimportant for hand calculation now became of utmost importance for the efficient and correct use of a large Computer System.
- Many of these considerations – programming languages, operating systems, management of large quantities of data, correctness of programs – were subsumed under the new discipline of Computer Science.

Scientific computing

- Scientific Computing is the collection of tools, techniques, and theories required to solve on a computer mathematical models of problems in Science and Engineering.
- The development of the electronic computer signaled a new era in the approach to the solution of scientific problems.
- Considerations that were irrelevant or unimportant for hand calculation now became of utmost importance for the efficient and correct use of a large Computer System.
- Many of these considerations – programming languages, operating systems, management of large quantities of data, correctness of programs – were subsumed under the new discipline of Computer Science.
- *But mathematics continues to play a major role in scientific computing: it provides the language of the mathematical models that are to be solved and information about the suitability of a model and it provides the theoretical foundation for the numerical methods and, increasingly, many of the tools from computer science.*

Exascale Computing

Exascale Computing



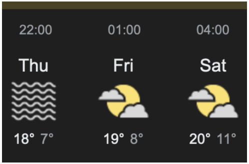
Exascale Computing

Exascale Computing

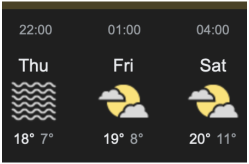
- Trends in processor manufacture mean that we no longer get faster simulations by adding more of the same types of components.
- There is no such thing as a ‘standard’ exascale computer - processors are becoming more specialised.
- The computer processor market is not driven by scientific applications.
- Very high levels of parallelism (or ‘concurrency’) are required.
- Arithmetic is cheap and data movement is expensive - especially accessing data off-node. Communication of data slows down the calculation.

N. Thompson, S. Spanuth, The Decline of Computers as a General Purpose Technology, Commun ACM 2

Numerical weather and climate prediction

Observations	Simulation	Forecast
		

Numerical weather and climate prediction

Observations	Simulation	Forecast
		

Dynamical core

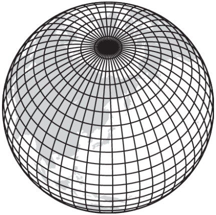
$$
\begin{aligned}
\boldsymbol{u}_t + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u} + 2\boldsymbol{\Omega}\times\boldsymbol{u} + c_p\theta\nabla\Pi + g\boldsymbol{k} &= 0,\\
\rho_t + \nabla\cdot(\rho\boldsymbol{u}) &= 0,\\
\theta_t + (\boldsymbol{u}\cdot\nabla)\theta &= 0.
\end{aligned}
$$

How to solve a PDE on a (super)computer

Solve $\boldsymbol{U}_t + \mathcal{F}(\boldsymbol{U}) = 0$ given $\boldsymbol{U}(0) = \boldsymbol{U}_0$.

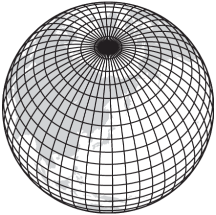
How to solve a PDE on a (super)computer

Solve $\boldsymbol{U}_t + \mathcal{F}(\boldsymbol{U}) = 0$ given $\boldsymbol{U}(0) = \boldsymbol{U}_0$.
Represent spatial derivatives on a grid.



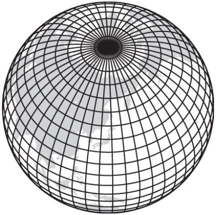
How to solve a PDE on a (super)computer

Solve $\boldsymbol{U}_t + \mathcal{F}(\boldsymbol{U}) = 0$ given $\boldsymbol{U}(0) = \boldsymbol{U}_0$.
Represent spatial derivatives on a grid.
Step forward in time calculating
$\boldsymbol{U}_{n+1} = \mathcal{T}(\boldsymbol{U}_n)$.



How to solve a PDE on a (super)computer

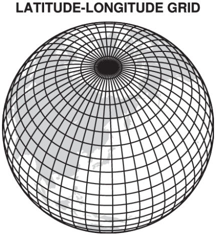
Solve $\boldsymbol{U}_t + \mathcal{F}(\boldsymbol{U}) = 0$ given $\boldsymbol{U}(0) = \boldsymbol{U}_0$.
Represent spatial derivatives on a grid.
Step forward in time calculating $\boldsymbol{U}_{n+1} = \mathcal{T}(\boldsymbol{U}_n)$.



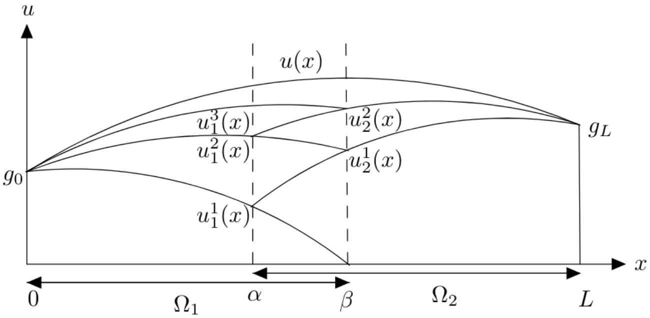
Where's the parallelism?

How to solve a PDE on a (super)computer

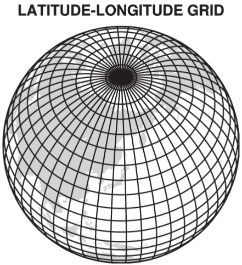
Solve $\boldsymbol{U}_t + \mathcal{F}(\boldsymbol{U}) = 0$ given $\boldsymbol{U}(0) = \boldsymbol{U}_0$.
Represent spatial derivatives on a grid.
Step forward in time calculating
$\boldsymbol{U}_{n+1} = \mathcal{T}(\boldsymbol{U}_n)$.



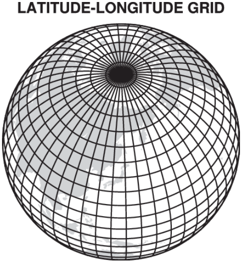
Where's the parallelism?
Spatial domain decomposition.



Changes to algorithms driven by parallelism

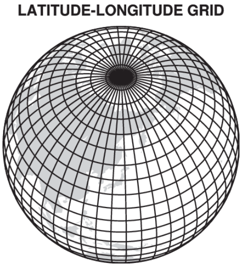


Changes to algorithms driven by parallelism



Challenge: bottleneck due to data communication

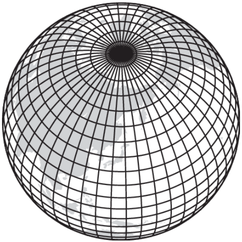
Changes to algorithms driven by parallelism



Challenge: bottleneck due to data communication

Strong scaling: more processors leads to more communication

Changes to algorithms driven by parallelism

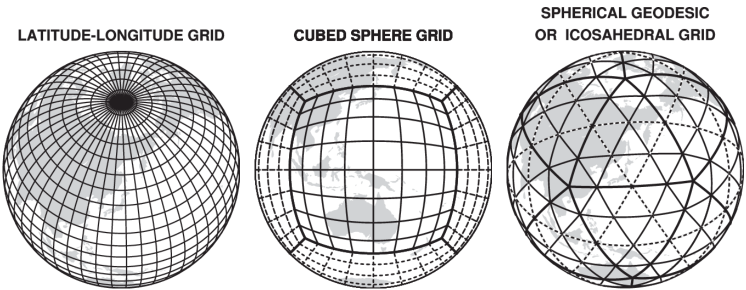


Challenge: bottleneck due to data communication

Strong scaling: more processors leads to more communication

Weak scaling: increase problem size, e.g. increase resolution... but then we have to decrease the timestep

Changes to algorithms driven by parallelism



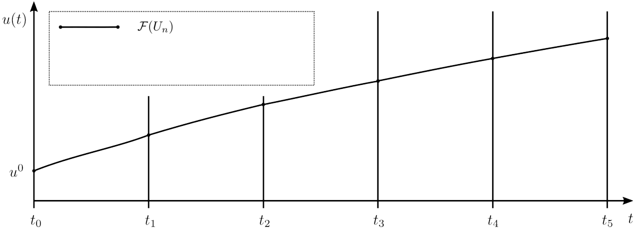
Challenge: bottleneck due to data communication

Strong scaling: more processors leads to more communication

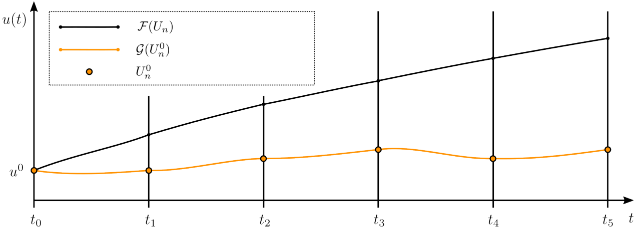
Weak scaling: increase problem size, e.g. increase resolution... but then we have to decrease the timestep

One solution: use a different grid... and then a different algorithm.

Changes to algorithms driven by parallelism

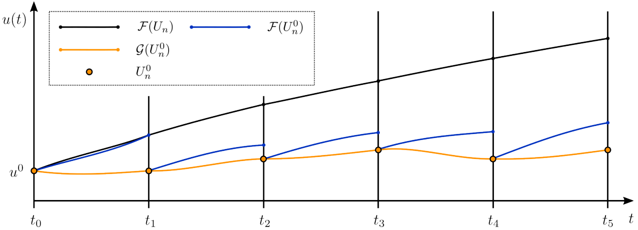


Changes to algorithms driven by parallelism



We partition the time domain into intervals and use G to give us an initial condition for F on each interval.

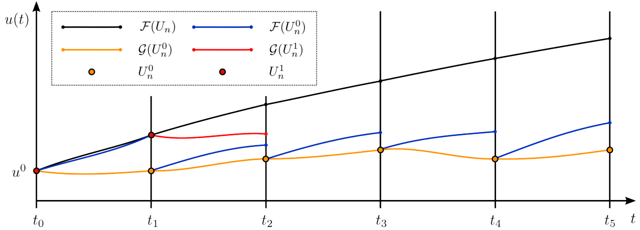
Changes to algorithms driven by parallelism



We partition the time domain into intervals and use G to give us an initial condition for F on each interval.

F can be computed in parallel, although the later time intervals do not have an accurate initial condition.

Changes to algorithms driven by parallelism



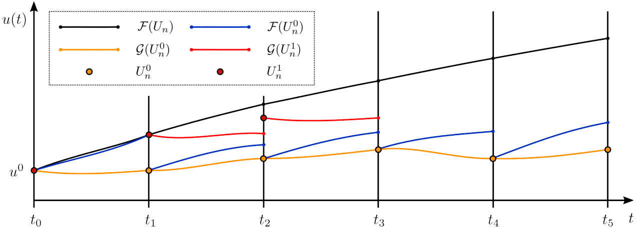
We partition the time domain into intervals and use G to give us an initial condition for F on each interval.

F can be computed in parallel, although the later time intervals do not have an accurate initial condition.

We iterate, correcting the initial condition on each interval:

$$u_0^{k+1} = u^0$$
$$u_{n+1}^{k+1} = F(u_n^k) + G(u_n^{k+1}) - G(u_n^k)$$

Changes to algorithms driven by parallelism



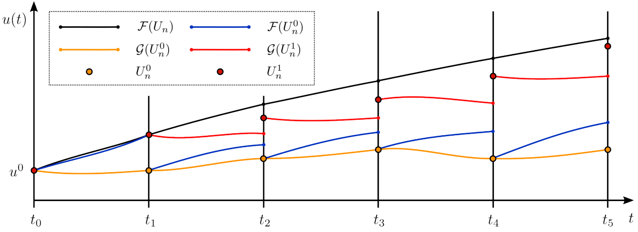
We partition the time domain into intervals and use G to give us an initial condition for F on each interval.

F can be computed in parallel, although the later time intervals do not have an accurate initial condition.

We iterate, correcting the initial condition on each interval:

$$u_0^{k+1} = u^0$$
$$u_{n+1}^{k+1} = F(u_n^k) + G(u_n^{k+1}) - G(u_n^k)$$

Changes to algorithms driven by parallelism



We partition the time domain into intervals and use G to give us an initial condition for F on each interval.

F can be computed in parallel, although the later time intervals do not have an accurate initial condition.

We iterate, correcting the initial condition on each interval:

$$u_0^{k+1} = u^0$$
$$u_{n+1}^{k+1} = F(u_n^k) + G(u_n^{k+1}) - G(u_n^k)$$

Changes to algorithms driven by parallelism

Challenge: We're using more processors but do we get the solution any quicker?!

G must be cheap to evaluate as it is the serial bottleneck in the simulation.

G must be accurate enough that information is communicated to later time slices, enabling fast convergence.

New frontiers in research

- ‘Paradigm shift’ in algorithm development due to hardware changes.
- Convergence of parallel-in-time algorithms is problem specific and the choice of algorithm will depend on the problem.
- Multiscale problems (e.g. adding ‘physics’ to ‘dynamics’)
- What are good test problems? How do we know our code is right?
- What trade offs can be made to optimise a particular problem on a particular architecture?
- Where does machine learning fit in with algorithm development?

What are the practical implications?

- ‘Separation of concerns’: algorithm implementation separate from parallelisation.
- Flexible software is required for rapid prototyping of new algorithms.
- Software becomes complex with a large ‘stack’ of dependencies.
- Performance modelling is really hard - sometimes the only way to find out is to run the code.
- Access to HPC is really important.
- Well-qualified people are ***absolutely essential*** for accessing HPC and building the software.
- Communication between domain scientists, algorithm developers and software/hardware analysts is crucial (and fun!)

`https://www.metoffice.gov.uk/research/approach/modelling-systems/lfric`