An introduction to parallel computing

Mike Giles

University of Oxford

My background

- maths undergraduate
- worked at Rolls-Royce in the summers, saw power of computational modelling for design of aircraft gas turbines
- PhD in aeronautical engineering, developing computational fluid dynamics methods

As an academic:

- collaborated with Rolls-Royce for 25 years, first in an engineering dept, then a computer science dept
- then moved to a maths dept and switched to computational finance, and more generally stochastic simulation

Mathematical modelling

- essential first step in any new application area
- often leads to PDEs
- sometimes also leads to stochastic models with modelling of uncertainty
- important to make sure models are well-posed mathematically
- simple examples provide insight
- asymptotics can reveal role of key parameters, and simplified models

Computational science & engineering

However, quantitative predictions usually require computation; computation is now the “third leg” of science along with theory and experiment

- sometimes small calculations (a few seconds), sometimes very big (a month or more)
- sometimes on a laptop, sometimes on a supercomputer
- often use software packages written by someone else, but you might be that “someone” in the future!

Computational science & engineering

Computational engineering has completely changed the way industry works

Products used to be designed through a lot of experimental trial-and-error, guided by engineers with outstanding experience / insight / ingenuity

Now done largely computationally, with final experiments to verify predicted behaviour

Computational science & engineering

Computational science has also changed completely

Used to be driven experimentally, with theory being developed to understand experimental data, at least qualitatively

Now theory leads to computations which match experiments to high accuracy, and can discover new features to be investigated experimentally

Computational science & engineering

In my dept:

- modelling of molten glass extrusion, leading to better mobile phone glass covers
- stochastic modelling of filters with application to removal of arsenic in groundwater in India/Bangladesh, as well as industrial filters
- applications to mathematical finance used in banks in London and New York
- stochastic simulation of biochemical reactions at very low concentrations
- modelling of battery performance for electric vehicles
- stochastic modelling of groundwater flow
- modelling of plasma in fusion reactors
- several data science applications

Progress in scientific computing

Over the past 60 years, there has been huge progress in the development of efficient numerical algorithms:

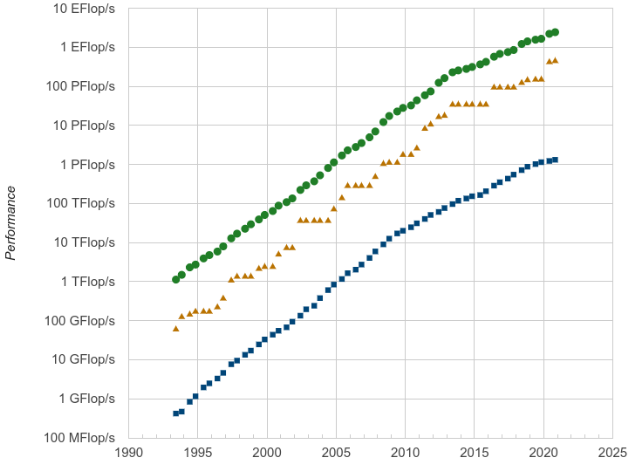
- Monte Carlo algorithm
- FFT
- conjugate gradient method
- multigrid methods
- finite element method
- interior point method in optimisation
- multipole methods

Numerical analysis is an important branch of mathematics

Progress in scientific computing

Computer power has advanced even more – over 1,000,000× performance growth in last 30 years:

`top500.org` performance evolution



Progress in scientific computing

However, things are beginning to tail off a bit – computer hardware is no longer improving at the same rate.

The power consumption is getting ridiculous – the #1 system, Frontier at the Oak Ridge National Laboratory, USA, needs 21 MW, and just a single CPU these days takes up to 350W.

This is partly because there used to be power savings from making the circuitry smaller, which offset the increase due to more circuitry, but the thickness of the "wires" on a chip is down to a few atoms.

Also, the operating frequency is no longer going up – been stuck at around 3GHz for several years, at least when all of the chip is working.

Progress in scientific computing

Manufacturers continue to increase the total amount of circuitry, in part by using bigger chips, so the main improvement in capability has come from multiple cores.

Data for Intel Core i5 desktop CPUs (like ones in computer lab)

year	cores	power	frequency
2015	2-4	15-65 W	1.8-3.6 GHz
2019	4-6	35-95 W	1.8-4.6 GHz
2023	8-14	45-125 W	1.3-5.1 GHz

Notes:

- data from `https://ark.intel.com/`
- CPUs run at lower frequencies when many cores are active
- 2023 CPU is a mix of “performance” and “efficiency” cores

Progress in scientific computing

The fastest Intel CPUs cuse up to 350W and have up to 60 complex cores, each of which is very powerful

In comparison, the fastest NVIDIA GPU has 16896 much simpler cores, and uses up to 700W

To program those we need to use CUDA, NVIDIA's proprietary extension to C/C++

Two model applications

In our CUDA practicals we will be looking at two particular applications:

- Practical 2: approximation of an SDE (Stochastic Differential Equation) and Monte Carlo estimation of an average output
- Practical 3: approximation of the solution to an ellptic PDE (Partial Differential Equation)

ODEs

A standard ODE (Ordinary Differential Equation) looks like

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(y)$$

which we sometimes also write as

$$\mathrm{d}y = f(y)\,\mathrm{d}t$$

This can be integrated over a short time interval to give

$$y(\Delta t) = y(0) + \int_0^{\Delta t} f(y(t))\,\mathrm{d}t$$

ODEs

If we then make the approximation $f(y(t)) \approx f(y(0))$ we get

$$y(\Delta t) \approx y(0) + \Delta t\, f(y(0))$$

Repeating this for one timestep after another gives us the Forward Euler approximation

$$\widehat{y}_{n+1} = \widehat{y}_n + \Delta t\, f(\widehat{y}_n)$$

where

$$\widehat{y}_n \approx y(n\Delta t)$$

For approximating ODEs there are other much more accurate methods, but this is a good starting point for SDEs

SDEs

With SDEs we add an extra random term

$$\mathrm{d}y = f(y)\,\mathrm{d}t + \sigma(y)\,\mathrm{d}W$$

where $\mathrm{d}W$ is the increment in a random Brownian motion

This is a model used for stock prices where there is uncertainty each day about what will happen.

The corresponding Euler-Maruyama approximation is

$$\widehat{y}_{n+1} = \widehat{y}_n + f(\widehat{y}_n)\,\Delta t + \sigma(\widehat{y}_n)\,\Delta W_n$$

where ΔW_n is a Normal random variable with zero mean and variance Δt, so $\Delta W_n = \sqrt{\Delta t}\; Z_n$ where Z_n is a standard Normal random variable with zero mean, and unit variance

SDEs

We can't predict the future, so not interested in one particular $y(t)$

Instead we want to run lots of independent simulations, using different random numbers, to simulate different possible futures

For each one, we evaluate how much money we will get from a financial "option" – let's call it $P^{(n)}$ for the n^{th} simulation

We then get the Monte Carlo estimate for the average value of P by averaging over N independent simulations:

$$\frac{1}{N}\sum_{n=1}^{N} P^{(n)}$$

The error in this estimate is proportional to $1/\sqrt{N}$, so we often want to do at least 10^5 simulations, hence the need for parallel execution

PDEs

If we have a square slab of metal, with its edges held at different temperatures, then the temperature distribution is described by the elliptic partial differential equation

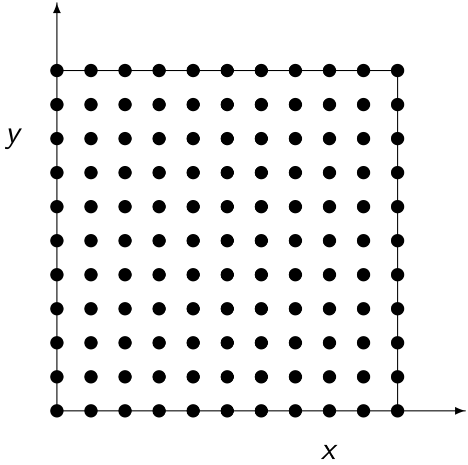
$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \qquad 0 < x < 1,\ 0 < y < 1$$

with $u(x, y)$ on the four edges at $x{=}0,\ x{=}1,\ y{=}0,\ y{=}1$

In simple cases like this we can calculate the solution, but in more difficult cases we need to use a numerical approximation

PDEs

Suppose we use a finite difference grid with uniform spacing $\Delta x = \Delta y$, and we want an approximation $U_{i,j} \approx u(i\Delta x, j\Delta y)$



PDEs

If we fit a quadratic function

$$f(x) = a + b\,x + \tfrac{1}{2}c\,x^2$$

through the three values $U_{i-1,j}, U_{i,j}, U_{i+1,j}$ then

$$c \equiv f''(x) = \frac{1}{\Delta x^2}\left(U_{i+1,j} - 2\,U_{i,j} + U_{i-1,j}\right)$$

and so this is an approximation to

$$\frac{\partial^2 u}{\partial x^2}$$

PDEs

Similarly,

$$\frac{1}{\Delta y^2} \left(U_{i,j+1} - 2\, U_{i,j} + U_{i,j-1} \right)$$

is an approximation to

$$\frac{\partial^2 u}{\partial y^2}$$

and so putting the two together gives the PDE approximation

$$\frac{1}{\Delta x^2} \left(U_{i+1,j} - 2\, U_{i,j} + U_{i-1,j} \right) + \frac{1}{\Delta y^2} \left(U_{i,j+1} - 2\, U_{i,j} + U_{i,j-1} \right) = 0$$

PDEs

With $\Delta x = \Delta y$, this simplifies to

$$U_{i,j} = \tfrac{1}{4} \left(U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1} \right)$$

and one way to solve this set of simultaneous equations (for $0{<}i{<}I, 0{<}j{<}J$ with $I{=}J{=}1/\Delta x$) is to use the Jacobi iteration

$$U_{i,j}^{(n+1)} = \tfrac{1}{4} \left(U_{i+1,j}^{(n)} + U_{i-1,j}^{(n)} + U_{i,j+1}^{(n)} + U_{i,j-1}^{(n)} \right)$$

Starting from $U_{i,j}^{(0)}{=}0$, $U_{i,j}^{(n)}$ converges (slowly) to the solution – there are much better numerical methods for this problem but it is a useful testcase for parallel computing