Parallel Numerical Algorithms

Chris Johnson (course organiser)

Room 3407

Chris Johnon
chrisj@epcc.ed.ac.uk

Introduction

- Most common algorithms or computational patterns in scientific computing
- Common across different applications
- Which, when, why (why not) in serial and in parallel
- Algorithmic complexity
- Error in numerical computing

- Verification
  - is my program/code correct?
- Significant mathematical content
  - simple as possible to understand numerical algorithms

S. Lawson, M. Woodgate, R. Steijl, and G. Barakos
http://www.hector.ac.uk/casestudies/ucav.php

Course structure

- 10 weeks - two lectures per week + one exercise class
  - Monday 1000 – 1050 lecture – rm 6301
  - Tuesday 1000 – 1050 exercise – rm 1028
    - Tomorrow general coding session – not PNA specific
  - Wednesday 1000 - 1050 lecture – rm 6301
- Lectures
  - available in Learn before class
  - Print if desired and annotate and elaborate
- Exercise classes
  - available in Learn before class
  - You are expected to complete the problems or practical exercises in your own time
  - Bring your work to the class, Tutors will help you when you get stuck
- Assessment: Exam at the end of the semester

Lecturers

- Four lecturers
  - Chris Johnson - rm 3407 (Course organiser, Matrices, PDEs)
  - Rupert Nash - rm 3409 (Representing numbers on computers)
  - Iain Bethune - rm 2404 (Fast Fourier Transforms)
  - Toni Collis - rm 2410 (Many body systems)

Timetable

- Floating Point numbers:
  - 2 lectures, 1 exercise
- Fast Fourier Transforms
  - 3 lectures, 1 exercise
- Dense linear algebra
  - 2 lectures, 1 exercise
- PDEs
  - 3 lectures, 1 exercise
- N-body methods
  - 3 lectures, 1 exercise
- Sparse linear algebra
  - 3 lectures, 2 exercises
- Full timetable in Learn

Full PNA timetable

Mon 21-Sep	10:00	Lect:	Introduction to PNA [CJ]
Tues 22-Sep	*10:00*	*Prac:*	*General MSc practical*
Wed 23-Sep	10:00	Lect:	Representing nos. on computers [RN]
Mon 28-Sep	10:00	Lect:	Representing nos. contd. [RN]
Tues 29-Sep	*10:00*	*Prac:*	*Ex FP errors, etc.*
Wed 30-Sep	10:00	Lect:	Introduction to FFTs [IB]
Mon 05-Oct	10:00	Lect:	Numerical libraries for FFTs (FFTW) [IB]
Tues 06-Oct	*10:00*	*Prac:*	*Ex FFT image processing*
Wed 07-Oct	10:00	Lect:	Parallel FFTs [IB]
Mon 12-Oct	10:00	Lect:	Dense LA intro [CJ]
Tues 13-Oct	*10:00*	*Prac:*	*Ex LU factorisation by hand/libs*
Wed 14-Oct	10:00	Lect:	LA libraries [CJ]
Mon 19-Oct	10:00	Lect:	Discretised PDEs. Pollution model [CJ]
Tues 20-Oct	*10:00*	*Prac:*	*Ex LU fact contd.*
Wed 21-Oct	10:00	Lect:	BVPs. Relaxation methods [CJ]
Mon 26-Oct	10:00	Lect:	Time-dependent PDEs (pollution) [CJ]
Tues 27-Oct	*10:00*	*Prac:*	*Ex Pollution PDE [CJ]*
Wed 28-Oct	10:00	Lect:	Intro to N-Body methods 1 [CJ]
Mon 02-Nov	10:00	Lect:	Intro to N-Body methods 2 [TC]
Tues 03-Nov	*10:00*	*Prac:*	*Ex Orbits, 2-body gravitational system*
Wed 04-Nov	10:00	Lect:	Computational chemistry [TC]
Mon 09-Nov	10:00	Lect:	Matrix splitting. Sparse matrices [CJ]
Tues 10-Nov	*10:00*	*Prac:*	*Ex Sparse mat storage, CG and BiCGSTAB*
Wed 11-Nov	10:00	Lect:	Intro to Krylov subspace methods [CJ]
Mon 16-Nov	10:00	Lect:	Advanced KS methods [CJ]
Tues 17-Nov	*10:00*	*Prac:*	*Ex Time dependent PDE*
Tues 24-Nov	*10:00*	*Prac:*	*Ex Parallel Matrix-vector exercise [CJ]*

Exercise class problems

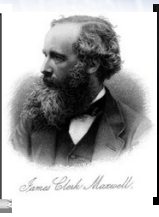
- Problems can be done in C or Fortran
  - Only some can be done in Java
  - competent Java programmer should be able to complete exercises in C
  - demonstrate and develop ideas essential to the course
- Significant mathematical content to the course
  - This not examinable, but you are expected to perform simple algebraic manipulation
- Complete all the exercises
  - you will complete the exam

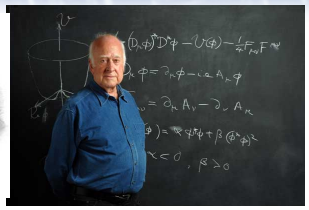
```
Subroutine times_vector_CSR(M, x, y)
  ! This routine calculates y = M*x for CSR-matrix m and vectors x and y
  type(csr), intent(in)                  :: M
  real(kind=16), dimension(M%ncol), intent(in)  :: x
  real(kind=16), dimension(M%nrow), intent(out) :: y
  integer                                :: row, col, ctr

  ! zero the vector before multiplying
  y = 0.0
  ctr = 1

  do row = 1,M%nrow
     do col = M%rowStart(row), M%rowStart(row+1)-1
        y(row) = y(row) + M%value(ctr)*x(M%colIndex(ctr))
        ctr = ctr + 1
     end do
  end do

end Subroutine times_vector_CSR
```

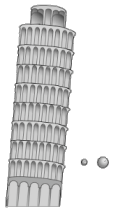

The scientific method

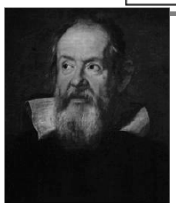
Theory: Mathematical equations provide a description or model

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}$$
$$\nabla \times \mathbf{H} = \mathbf{J} + \frac{\partial \mathbf{D}}{\partial t}$$
$$\nabla \cdot \mathbf{D} = \rho$$
$$\nabla \cdot \mathbf{B} = 0$$

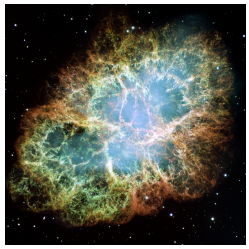






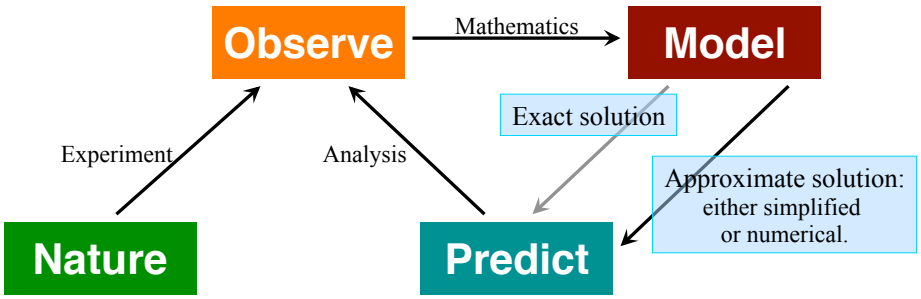


- Experiment
  - Inference from data
  - Test hypothesis

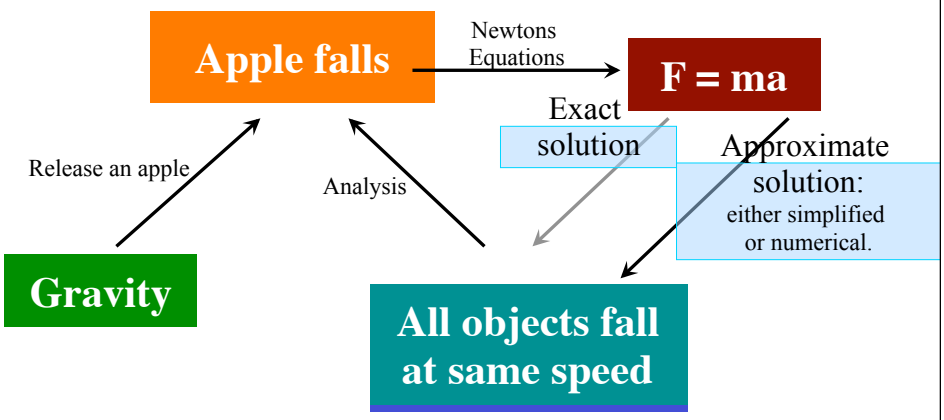


- Computer simulation
  - Too big, small, difficult or dangerous for standard methods
  - Explore the space of solutions

Computer Simulation in Science



Example: Gravity



The Seven Dwarfs of HPC

- Dense Linear Algebra
- Sparse Linear Algebra

} Arise from discretising PDEs, solved with different methods

- Spectral Methods
  - transform domain problem, e.g. Fourier Transform (FT)
- N-body Methods
  - Particle-particle e.g. orbits, molecular dynamics
- Structured Grids
  - Data in a regular multidimensional, data updated in sequence
  - Lattice methods
- Unstructured Grids
  - Data in irregular multidimensional grid, triangulation, graph partitionings
- MapReduce
  - Repeated application of a function, final results aggregated
    - Monte Carlo, statistical methods

What is tractable?

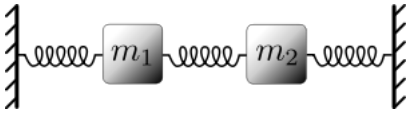
- How big a problem can we tackle?
  - On standard computers
    - Run out of time
    - Run out of space
  - On state-of-the-art machines
    - Run out of time
    - Run out of space
    - Run out of precision

Scientific computations are limited by available computing resources

Complexity

- What size of problem can be tackled depends on the complexity of the problem
  - How much resource consumed to solve
  - Depends on problem
  - Depends on algorithm
- System size, N
  - number of degrees of freedom
  - number of points simulated
  - dimension of problem (DoF & simulated points)

- 1-d system
- 2 grid points - 1DoF
- N = 2



Big O notation

- How many operations does an algorithm take to solve a system of size N?

$$\Omega = f(N)$$ **Say "omega is *some function* of N"**

- We only care about the *dominant* or *leading* behaviour of *f*

$$f(N) = 3N^3 - 8N^2 + 27N - 3762$$

$$f(N) \sim \mathcal{O}(N^3)$$ **Say "*of order* N cubed"**

- Examples: For a system of size N
  - Conjugate gradient O(N) flops (to solve N unknowns of a linear system)
  - LU factorisation takes $O(N^3)$
  - Bubble sort takes $O(N^2)$ to sort N numbers
  - Fast Fourier Transform O(N logN) to transform a vector of length N

Performance

- Algorithmic/computation time also depends on number of processing elements (nodes/cores/threads etc), *P*
- Let $T(N,P)$ be the time taken
  - system of size *N*
  - *P* elements
- Speed up, *S*

$$S(N,P) = \frac{T(N,1)}{T(N,P)} \quad \not< P \quad \ll P$$

- Parallel efficiency, *E*

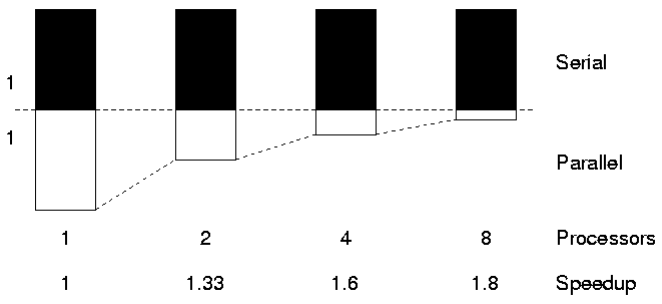
$$E(N,P) = \frac{S(N,P)}{P} = \frac{T(N,1)}{P \times T(N,P)} \quad \not< 1 \quad \ll 1$$

The Serial Component

- Amdahl's law

"the performance improvement to be gained by parallelisation is limited by the proportion of the code which is serial"

Gene Amdahl, 1967

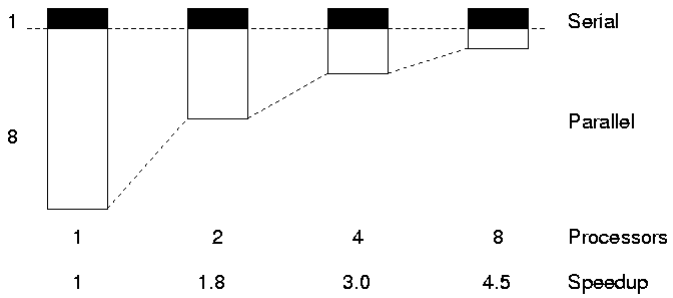


Amdahl's law

- Assume a fraction α is completely serial
  - time is sum of serial and potentially parallel
- Parallel time $T(N,P) = \alpha T(N,1) + \frac{(1-\alpha)T(N,1)}{P}$
  - parallel part 100% efficient
- Parallel speedup $S(N,P) = \frac{T(N,1)}{T(N,P)} = \frac{P}{\alpha P + (1-\alpha)}$
  - for α = 0, *S* = *P* as expected (ie *E* = 100%)
  - otherwise, speedup limited by 1/ α for any *P*
  - impossible to effectively utilise large parallel machines?

Gustafson's Law

- Need larger problems for larger numbers of CPUs



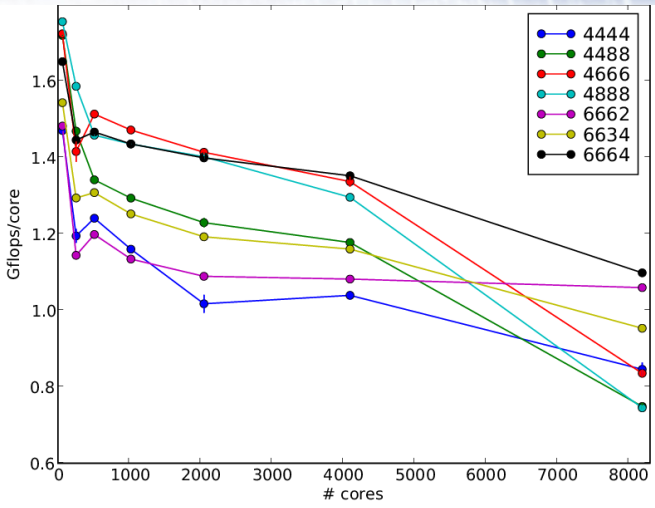
Strong versus Weak scaling

- Two different ways to present scaling data
- Weak scaling: local volume is fixed
  - system size changes
  - different problem
  - constant efficiency → flat lines on the plot (*c.f.* Gustafson's Law)
- Strong scaling: global volume is fixed
  - system size is fixed
  - local volume is changing

$$P \uparrow: \frac{V}{P} \downarrow: \frac{W}{P} \downarrow: C \uparrow$$

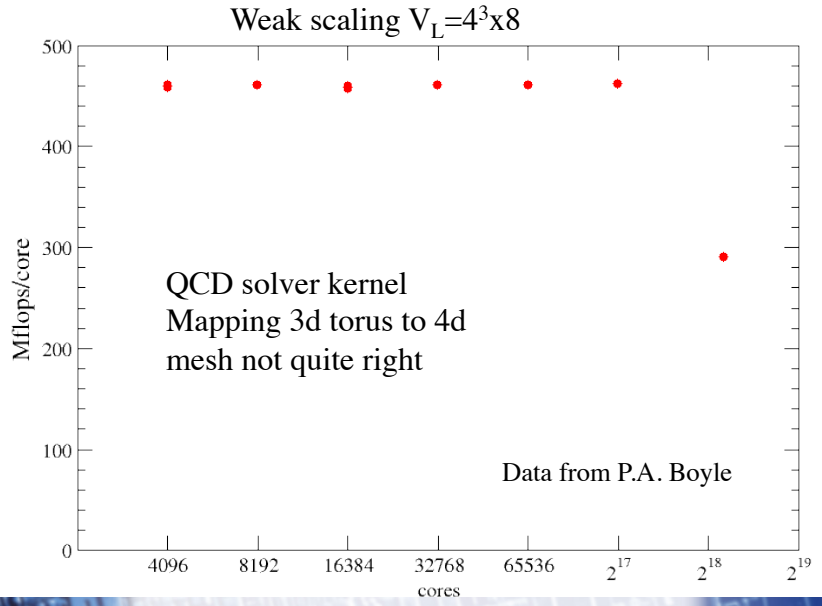
- Achieving weak scaling is hard
- Achieving strong scaling is harder

Weak scaling on HECToR XT4



4 dimensional QCD code called Chroma
performance of solver for different local volumes

Extreme scaling: BG/P @ FZJ Germany



Limits of computation: Complexity theory

- Polynomial time algorithms
  - –computation is polynomial function of problem size, N
  - –*Feasible* calculations — May still be intractable on an actual computer
- Exponential time algorithms
  - –These are *unfeasible* calculations
  - –Classic example is integer factorisation
    - –Find the two prime factors of an integer → $n = pq$
    - –Actually *sub-exponential* → $O\left(2^{(\log N)^{1/3}}\right)$
  - –Cryptography
    - –Public-key system such as RSA
    - –200 digit number to equivalent of 75 years to factorise
    - –Quantum computer reduces this to polynomial time
      - – Break public-key encryption!

Errors and uncertainty

- No analytic solution
  - No exact answer
- Finite problem, finite model, finite resource
  - Rounding errors
  - Uncertainty in input data - `GiGo`
- How accurate do we need our answer to be
  - How do we know how accurate we are?
- Use examples to illustrate this point
  - Planetary orbits
  - Pollution problem

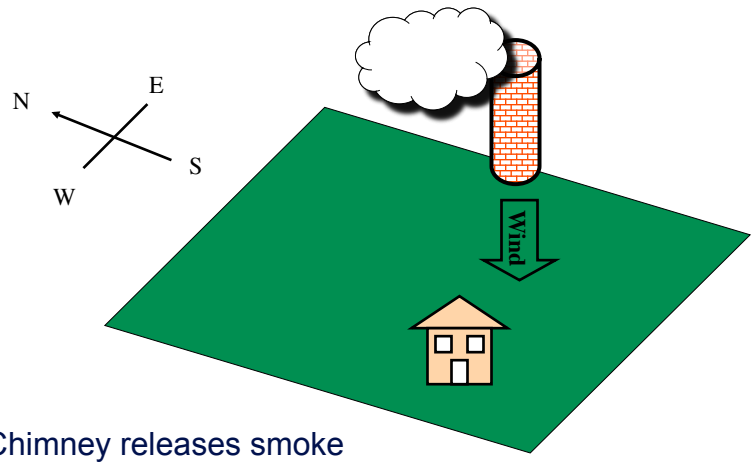
Mathematical models

- Problems in `Engineering, Mathematics, Physics, Chemistry, Finance, Biology, Complex systems` represented by differential equations [partial, ordinary, integral etc]
- Express relationships between different quantities and how they *change* depending on position in space and time
- Example Newton's second Law

$$\vec{F} = m\vec{a} = m\frac{d\vec{v}}{dt} = m\frac{d^2\vec{x}}{dt^2}$$

- Vector ordinary differential equation
  - Ordinary → change of only one variable, time)

Modelling spread of pollution



- Chimney releases smoke
  - how much arrives at house with prevailing north-easterly wind?

Laplace's Equation

- Convection-Diffusion equation
  - Let *u(x,y)* be the amount of pollution at point *(x,y)*

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) u(x,y) = 0$$

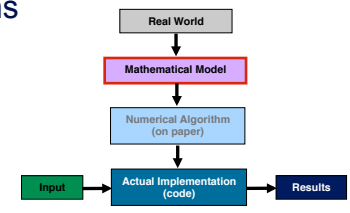
  - Two dimensional problem, u depends on x and y
  - Scalar elliptical partial differential equation
  - Often written as

$$\nabla^2 u(x,y) = 0$$

- We can introduce Boundary Conditions
  - chimney (source of pollution)

$$\nabla^2 u(x,y) = b(x,y)$$

  - Poisson's equation (inhomogeneous eqn)



Linear equations

- Add a wind term $\left(a_x \frac{\partial}{\partial x} + a_y \frac{\partial}{\partial y}\right) u(x,y)$

$$\left(\nabla^2 + \vec{a} \cdot \vec{\nabla}\right) u(x,y) = b(x,y)$$

$$\mathbf{A}u = b$$

Differential operator

Encodes how the system behaves

Linear in u

The solution

Solving Linear equations is hard!

Solving non-Linear equations is harder!

Boundary conditions

Different for each problem

Linear equations

- Add a wind term $\left(a_x \frac{\partial}{\partial x} + a_y \frac{\partial}{\partial y}\right) u(x,y)$

$$\left(\nabla^2 + \vec{a} \cdot \vec{\nabla}\right) u(x,y) = b(x,y)$$

$$\mathbf{A}u = b$$

Differential operator

Encodes how the system behaves

Linear in u

The solution

Solving Linear equations is hard!

Solving non-Linear equations is harder!

Boundary conditions

Different for each problem

Linear equations

- Add a wind term $\left(a_x \frac{\partial}{\partial x} + a_y \frac{\partial}{\partial y}\right) u(x,y)$

$$\left(\nabla^2 + \vec{a} \cdot \vec{\nabla}\right) u(x,y) = b(x,y)$$

$$\mathbf{A}u = b$$

Differential operator

Encodes how the system behaves

Linear in u

The solution

Solving Linear equations is hard!

Solving non-Linear equations is harder!

Boundary conditions

Different for each problem

Linear equations

- Add a wind term $\left(a_x \frac{\partial}{\partial x} + a_y \frac{\partial}{\partial y}\right) u(x,y)$

$$\left(\nabla^2 + \vec{a} \cdot \vec{\nabla}\right) u(x,y) = b(x,y)$$

$$\mathbf{A}u = b$$

Differential operator

Encodes how the system behaves

Linear in u

The solution

Solving Linear equations is hard!

Solving non-Linear equations is harder!

Boundary conditions

Different for each problem

Linear equations

- Add a wind term $\left(a_x \frac{\partial}{\partial x} + a_y \frac{\partial}{\partial y}\right) u(x,y)$

$$\left(\nabla^2 + \vec{a} \cdot \vec{\nabla}\right) u(x,y) = b(x,y)$$

$$\mathbf{A}u = b$$

General form

Differential operator

Encodes how the system behaves

Linear in u

The solution

Solving Linear equations is hard!

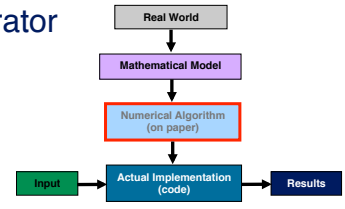
Solving non-Linear equations is harder!

Boundary conditions

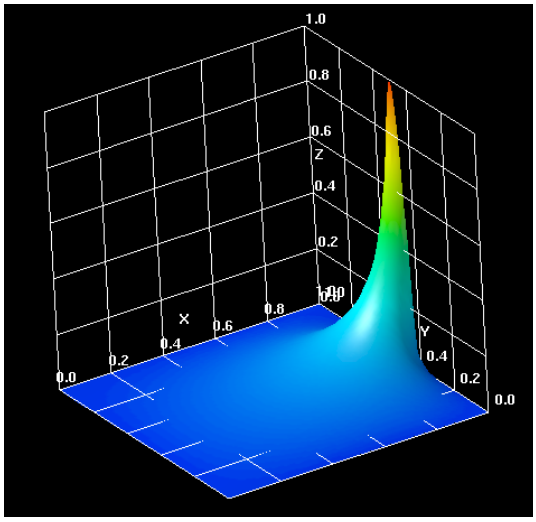
Different for each problem

Numerical solution

- Typically no analytic solution
- Numerical method finds an approximate solution
- Replace infinitesimal differential equation with finite difference equation
  - c.f. also finite element method, see later
- Replace continuous domain, *D*, with a discrete mesh, *M*
  - Mesh spacing, h, $N = M^d$
- Differential operator → difference operator
  - many different schemes for this
- u and b → vectors of size N
- A is NxN matrix



Solution



Numerical error

Depends on discretisation scheme

Number of points on mesh M, or grid spacing, h

Error ε

$$\epsilon = f(h)$$

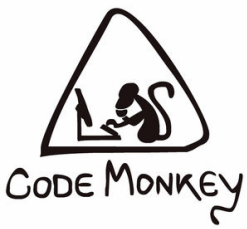
f scheme dependent

Smaller error → more computation

Implementation

This is your task!

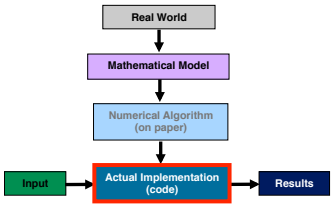
Each exercise looks at implementing a different algorithm to solve the (sometimes same) problem



Code errors → wrong answer

Verification?

Rounding errors see next lecture



Checking Correctness

- This is a real issue!
  - Verification
- Unit tests - code
  - each module or functional/algorithmic component has a test?
  - what about the tests?
- Integration tests
  - Do the functional pieces interact correctly?
- Replicate known results
  - Analytic solution
  - other codes

Checks & Sums

- Conservation of energy

$$E_K(t) = \sum_i \frac{1}{2} m_i \vec{v}_i(t)$$

$$E_P(t) = \sum_{\langle i,j \rangle} U(|\vec{r}_j(t) - \vec{r}_i(t)|)$$

$$E_T = E_P(t) + E_K(t) = C$$

Constant: Not time dependant

Similarly Conservation of linear momentum $\vec{P} = \sum_i m_i \vec{v}_i(t)$

Linear momentum should also be conserved in each dimension individually, e.g. *x, y, z*

Necessary but not sufficient

Implementation could pass, and still be wrong

Verifying the unknown

- Sometimes answer is not known
- Physical symmetries of the problem should be respected
  - simulation should be invariant under changes

$$f(x) \rightarrow f(x') = f(x + a)$$ Translations

$$f(x) \rightarrow f(x') = f(x \cdot \theta)$$ Rotations

$$f(x) \rightarrow f'(x) = f(x) + \phi$$ Rotations of field

- This is problem specific

Necessary but not sufficient

Implementation could pass, and still be wrong

Summary

- Most problems in Science and Engineering have no analytic solution
- Solve with numerical methods
  - these are approximations
  - Understanding the error
  - Verification
- Introduce the most commonly used algorithmic or computational patterns
- How and when to use them

References

- PA Wiegert, KA Innanen and S Mikkola *An asteroidal companion to the Earth*, *Nature* **387**, 685-686 (12 June 1997)
- V. Szebehely, *Theory of orbits, (Academic, New York, 1967)*
- *A View from Berkeley* http://view.eecs.berkeley.edu/wiki/Main_Page
- C.H. Papadimitriou, *Computational Complexity,* (Addison-Wesley, Reading, MA 1994)
- Rivest, R.; A. Shamir; L. Adleman (1978). *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems* *Comm. ACM* **21** (2): pp.120–126. http://theory.lcs.mit.edu/~rivest/rsapaper.pdf
- RSA factoring challenge http://www.rsa.com/rsalabs/node.asp?id=2092