# Here there be dragons

## The HPC Code Development Journey

Phil Tooley
University of Sheffield

8 April 2019

scheduler memory

user interface

libraries MPI language

testing

performance CUDA bugs

documentation licensing

OpenMP throughput compilers

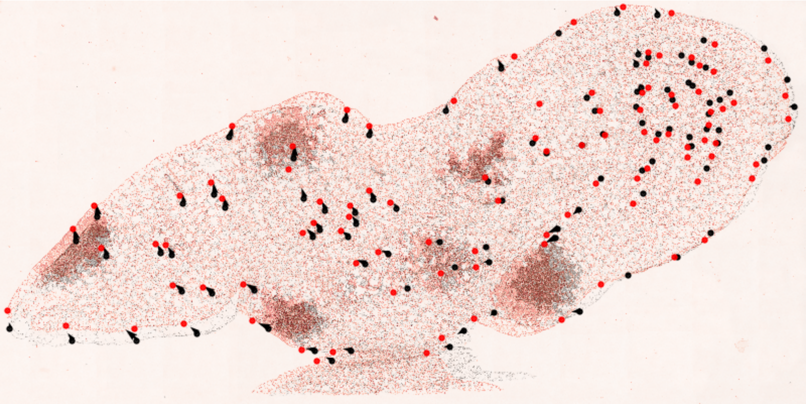
version control algorithms

# What makes an application HPC?

Anything that your laptop can't do on its own...

- More compute power
- More memory
- Specialist hardware
- Large dataset storage
- Better visualisation
- etc...

# A real world example — image registration



# How do I write code for an HPC?

Work out what your code needs:

## Large Memory - A bigger node than your laptop

- Code is fast enough, but need more memory
- No changes to "laptop code"
- Use HPC large memory nodes for larger datasets

## High Throughput - Run many jobs at once

- Code is fast enough, but need to run lots
- No changes to "laptop code"
- Use HPC to run many jobs in parallel

# How do I write code for an HPC?

Work out what your code needs:

Shared Memory - Multiple cores on one node

- Code fits in a single node's memory, but needs to be faster
- Parallel algorithms for "bottlenecks" in the code
- Can often be added to existing serial code

Message Passing - Multiple cores on multiple nodes

- Need multiple nodes to fit problem in memory
- Parallelise data structures and algorithms
- Serial code usually needs a complete rewrite

# How do I write code for an HPC?

Work out what your code needs:

## GPU Accelerator

- Code fits in a single node's memory, but needs to be faster
- Use 1000s of parallel threads for computation
- Needs large, parallel problems for efficient use of hardware

# You can use parallel libraries to help

## Low level parallel libraries

- OpenMP - Shared memory helpers for C/C++/Fortran
- MPI - MPI standard implemented by many vendors, available for many languages
- CUDA - Run many parallel threads on Nvidia GPUs
- OpenACC - OpenMP-like primitives for GPU/Intel MIC offload
- Intel TBB - threading building blocks for parallel applications

No maths in these libraries. Just tools to help you write parallel code.

# But don't reinvent the wheel

## Lots of higher level libraries available

- General high performance math - GSL, MKL, NAG
- Linear Algebra - LaPACK, BLAS, Intel MKL
- ODE/PDE solvers - PETSc, PVODE, FEniCS
- Scientific frameworks - PETSc, Trilinos
- Cluster computing - Dask, Hadoop, Spark
- Domain specific frameworks - Tensorflow, OpenFOAM, MFEM

# Managing dependencies

## Use a dependency manager

- Give manager a list of required packages and versions
- Dependency manager coordinates installation
- Loaded using modules just like system provided software
- Install multiple versions of a library side-by-side

## Easier installation and reproducibility

- Easier installation for users
- Reproducible environment across platforms
- Easy testing of dependency versions

# Writing pFIRE

## pFIRE - Image Registration Code

- Computationally intensive linear algebra
- Must scale to very large 3D images — need lots of memory!
- MPI is the best choice – look for an MPI linear algebra library

## PETSc

- High performance, parallel linear algebra library
- Supports MPI or CUDA for parallelism
- High performance matrix solvers
- Lots of helpful routines for matrix and vector manipulation

# Parallel linear Algebra with PETSc

```
Mat A;                                          // Create a 100000×100000 matrix called A
MatCreate(MPI_COMM_WORLD, &A);
MatSetSizes(A, PETSC_DECIDE, PETSC_DECIDE,
            100000, 100000);

Vec X, Y;                                       // Create compatible sized vectors X and Y
MatCreateVecs(A, &X, &Y);

                                                // Insert values into the matrix and vector Y
MatSetValue(A, row, col, value, INSERT_VALUES); // Set a matrix value
VecSetValue(Y, loc, value, INSERT_VALUES);      // Set a vector value

                                                // Solve the equation AX = Y
KSP solver;
KSPCreate(MPI_COMM_WORLD, &solver);             // Create Krylov solver object
KSPSetOperator(solver, A, A);
KSPSetUp(ksp);
KSPSolve(ksp, Y, X);                            // Solve for X given M and Y
```

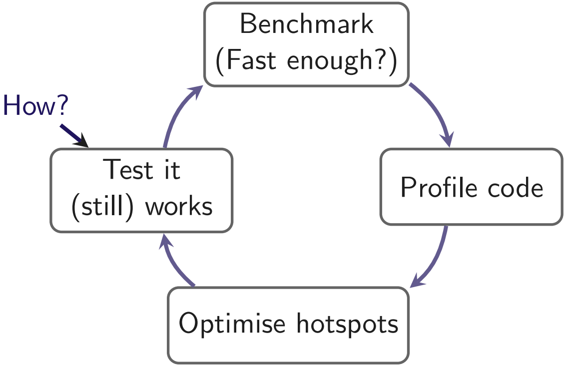
# Performance and Profiling

Ignore code performance **until it works!**

- Test often
- Focus on hot spots
- Decide what is "fast enough"

Lots of tools — free and £££

- Code profilers — Slow sections
- Memory profilers — Usage and access
- MPI Profilers — Communication



# Testing and Benchmarking

Testing early and often helps catch mistakes

- Test individual functions:
  - Ensure correct output for valid input
  - Graceful failures with invalid input
- Test full program behaviour (integration tests):
  - Identify useful test cases with known results
  - Test on different machines/architectures
  - Regression tests: compare to previous versions

Lots of tools to help automate this

# Conclusions

Writing high quality HPC code is not easy, but...

- Lots of options for applying HPC to a problem
  - determine appropriate approach based on current and future needs
- Using high-level frameworks provides all the building blocks
  - e.g PETSc for parallel linear algebra, Spark for big data
- Small changes can bring big gains
  - parallelising just the slowest part can really help with speed
- Software development best practice helps create robust software
  - regular testing finds bugs early, version control makes fixing them easier

With the right tools, *anyone* can write good HPC code

Thank You

# Useful Resources

- Slides: https://github.com/ptooley/hpc_dragons
- RSE Code Clinic:

  Free 30 minute advice sessions: https://rse.shef.ac.uk/support/code-clinic/
- Best Practice Guidance

  RSE@Sheffield: https://rse.shef.ac.uk
  SSI: https://software.ac.uk
- HPC Programming

  PRACE Training: http://www.training.prace-ri.eu/
  LLNL Online Training: https://hpc.llnl.gov/training/tutorials
  GPU@Sheffield: http://gpucomputing.shef.ac.uk/
  ShARC Documentation: http://docs.hpc.shef.ac.uk/en/latest/sharc/

CompBioMed

# COMPBIOMED CONFERENCE 2019

www.compbiomed-conference.org

**25-27 SEPTEMBER 2019**

**IET London**
**2 Savoy Place. London, WC2R 0BL**

**Conference Chair**
**Prof Peter Coveney**

**Plenary Speakers**

- **Andrew Hopkins (Exscientia)**
- **William L. Jorgensen (Yale)**
- **Amanda Randles (Durham, NC)**
- **Anne M. Robertson (Pittsburgh)**
- **Oliver Röhrle (Stuttgart)**

**Conference Themes**

**Biomedical Applications (including)**

Cardiac & Bloodflow, Neuromusculoskeletal, Brain
Molecular Medicine

**Methodology (including)**

Uncertainty Quantification
Role of Theory, Modelling and Simulation in Biomedicine

**Technology & Outreach (including)**

Education, Training & Public Awareness
Regulatory Science and *in silico* Trials