

Performance optimisation of research codes for the Supercomputing Wales programme: three case studies

Ed Bennett

@QuantumofEd



Swansea University
Prifysgol Abertawe

@SwanseaUni



SUPERCOMPUTING WALES
UWCHGYFRIFIADURA CYMRU

@SuperCompWales



@sa2c_swansea

HPC-LEAP Conference, Cambridge University

11 July 2018

Slides: git.io/fNTis

In this talk

- The Supercomputing Wales programme
- What is a Research Software Engineer?
- Case studies:
 - Throughput optimization of a life sciences application
 - Improving parallel efficiency of a CFD application
 - Parallelising a lattice field theory code

The Supercomputing Wales programme

- £15m investment by EU, Welsh Government, and four Welsh Universities

The Supercomputing Wales programme

- £15m investment by EU, Welsh Government, and four Welsh Universities
 - Cardiff

The Supercomputing Wales programme

- £15m investment by EU, Welsh Government, and four Welsh Universities
 - Cardiff
 - Swansea

The Supercomputing Wales programme

- £15m investment by EU, Welsh Government, and four Welsh Universities
 - Cardiff
 - Swansea
 - Aberystwyth

The Supercomputing Wales programme

- £15m investment by EU, Welsh Government, and four Welsh Universities
 - Cardiff
 - Swansea
 - Aberystwyth
 - Bangor

The Supercomputing Wales programme

- £15m investment by EU, Welsh Government, and four Welsh Universities
 - Cardiff
 - Swansea
 - Aberystwyth
 - Bangor
- Two HPC facilities

The Supercomputing Wales programme

- £15m investment by EU, Welsh Government, and four Welsh Universities
 - Cardiff
 - Swansea
 - Aberystwyth
 - Bangor
- Two HPC facilities
 - 12,000 Skylake cores

The Supercomputing Wales programme

- £15m investment by EU, Welsh Government, and four Welsh Universities
 - Cardiff
 - Swansea
 - Aberystwyth
 - Bangor
- Two HPC facilities
 - 12,000 Skylake cores
 - 34 Nvidia V100 GPUs

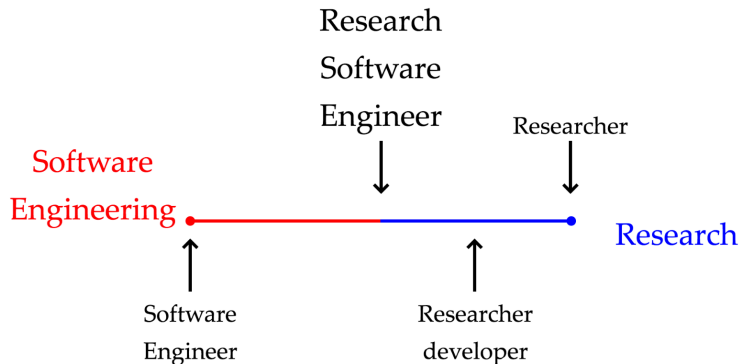
The Supercomputing Wales programme

- £15m investment by EU, Welsh Government, and four Welsh Universities
 - Cardiff
 - Swansea
 - Aberystwyth
 - Bangor
- Two HPC facilities
 - 12,000 Skylake cores
 - 34 Nvidia V100 GPUs
 - 2 Bullion Analytics Factories

The Supercomputing Wales programme

- £15m investment by EU, Welsh Government, and four Welsh Universities
 - Cardiff
 - Swansea
 - Aberystwyth
 - Bangor
- Two HPC facilities
 - 12,000 Skylake cores
 - 34 Nvidia V100 GPUs
 - 2 Bullion Analytics Factories
- 15 Research Software Engineers

What is a Research Software Engineer?



(from Simon Hettrick, <https://goo.gl/znRRzb>)

Supercomputing Wales RSEs

- RSEs:

Supercomputing Wales RSEs

- RSEs:
 - Know software engineering

Supercomputing Wales RSEs

- RSEs:
 - Know software engineering
 - Know research

Supercomputing Wales RSEs

- RSEs:
 - Know software engineering
 - Know research
 - Develop software for research

Supercomputing Wales RSEs

- RSEs:
 - Know software engineering
 - Know research
 - Develop software for research
- Supercomputing Wales RSEs:

Supercomputing Wales RSEs

- RSEs:
 - Know software engineering
 - Know research
 - Develop software for research
- Supercomputing Wales RSEs:
 - Do all this

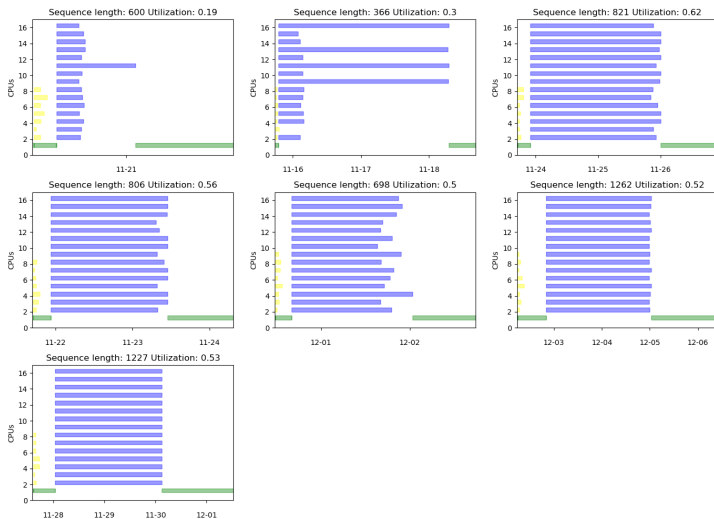
Supercomputing Wales RSEs

- RSEs:
 - Know software engineering
 - Know research
 - Develop software for research
- Supercomputing Wales RSEs:
 - Do all this
 - Know about HPC

Supercomputing Wales RSEs

- RSEs:
 - Know software engineering
 - Know research
 - Develop software for research
- Supercomputing Wales RSEs:
 - Do all this
 - Know about HPC
 - Adapt, optimize, and benchmark research software for HPC

Throughput optimisation for life sciences



Options considered

- Serialise

Options considered

- Serialise
- Separate jobs, SLURM dependencies

Options considered

- Serialise
- Separate jobs, SLURM dependencies
- Allocate a single job, run many jobs within it

Options considered

- Serialise
- Separate jobs, SLURM dependencies
- Allocate a single job, run many jobs within it
 - Require a queue manager to organise this

METAQ (<https://git.io/fNUxC>)

- Written in bash

METAQ (<https://git.io/fNUxC>)

- Written in bash
- Filesystem-based

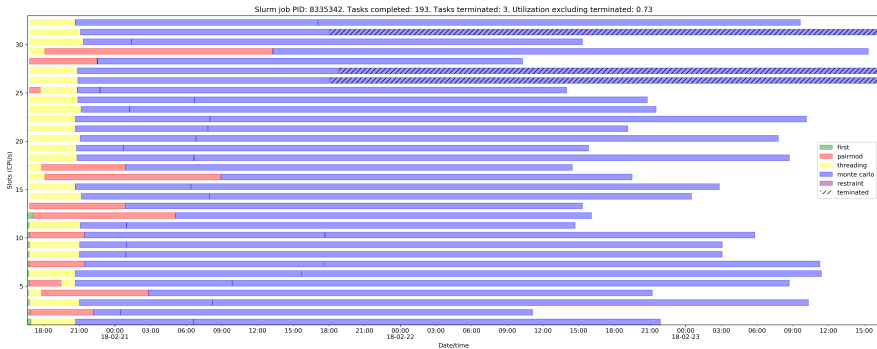
METAQ (<https://git.io/fNUxC>)

- Written in bash
- Filesystem-based
- No support for dependencies

METAQ (<https://git.io/fNUxC>)

- Written in bash
- Filesystem-based
- No support for dependencies
 - Now implemented

Initial results of throughput optimisation



Ongoing work

- Predictions of runtime for each job step

Ongoing work

- Predictions of runtime for each job step
- Collecting data from current runs to allow this

Parallel optimisation of computational fluid dynamics software

- Solver for the 2D Boltzmann equation

Parallel optimisation of computational fluid dynamics software

- Solver for the 2D Boltzmann equation
 - BGK operator

Parallel optimisation of computational fluid dynamics software

- Solver for the 2D Boltzmann equation
 - BGK operator
 - Two-step discontinuous Taylor–Galerkin

Parallel optimisation of computational fluid dynamics software

- Solver for the 2D Boltzmann equation
 - BGK operator
 - Two-step discontinuous Taylor–Galerkin
- Position and velocity degrees of freedom

Parallel optimisation of computational fluid dynamics software

- Solver for the 2D Boltzmann equation
 - BGK operator
 - Two-step discontinuous Taylor–Galerkin
- Position and velocity degrees of freedom
- Initially position-space parallelised (domain decomposition)

Parallel optimisation of computational fluid dynamics software

- Solver for the 2D Boltzmann equation
 - BGK operator
 - Two-step discontinuous Taylor–Galerkin
- Position and velocity degrees of freedom
- Initially position-space parallelised (domain decomposition)
 - Memory bound

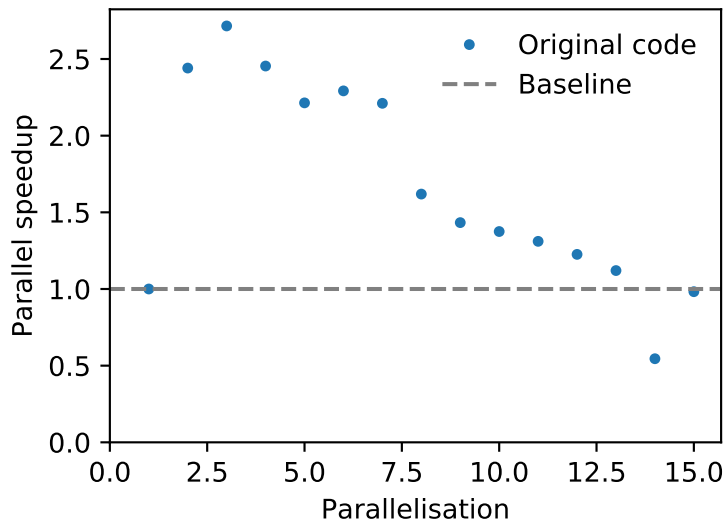
Parallel optimisation of computational fluid dynamics software

- Solver for the 2D Boltzmann equation
 - BGK operator
 - Two-step discontinuous Taylor–Galerkin
- Position and velocity degrees of freedom
- Initially position-space parallelised (domain decomposition)
 - Memory bound
 - Performance not explicitly considered

Parallel optimisation of computational fluid dynamics software

- Solver for the 2D Boltzmann equation
 - BGK operator
 - Two-step discontinuous Taylor–Galerkin
- Position and velocity degrees of freedom
- Initially position-space parallelised (domain decomposition)
 - Memory bound
 - Performance not explicitly considered
- Fortran 77 code

Initial performance (28304 elements, 40×40 velocity)



Optimisations implemented

- Parallelise in velocity space

Optimisations implemented

- Parallelise in velocity space
 - Less communication on this axis

Optimisations implemented

- Parallelise in velocity space
 - Less communication on this axis
- Optimise domain decomposition

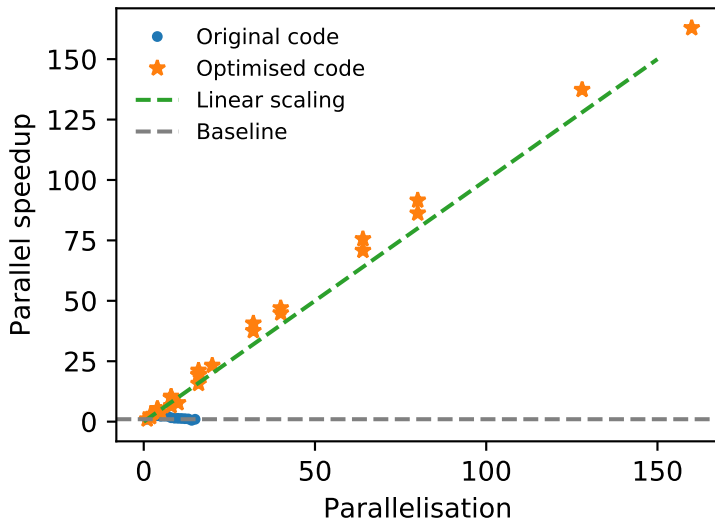
Optimisations implemented

- Parallelise in velocity space
 - Less communication on this axis
- Optimise domain decomposition
 - Better load balancing

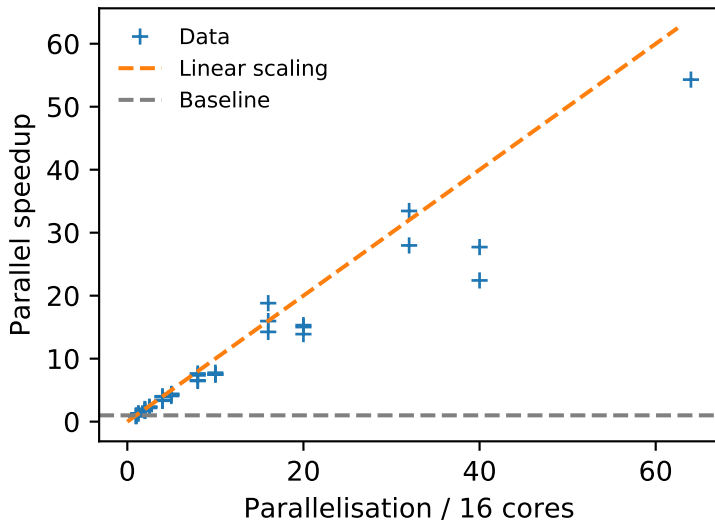
Optimisations implemented

- Parallelise in velocity space
 - Less communication on this axis
- Optimise domain decomposition
 - Better load balancing
 - Less communication

Optimised performance (28304 elements, 40×40 velocity)



Optimised performance (402,534 elements, 80×80 velocity)



Parallelising lattice field theory code

- 2+1d Thirring model

Parallelising lattice field theory code

- 2+1d Thirring model
- RHMC algorithm

Parallelising lattice field theory code

- 2+1d Thirring model
- RHMC algorithm
- Serial Fortran 77 (and FORTRAN IV) code

Parallelising lattice field theory code

- 2+1d Thirring model
- RHMC algorithm
- Serial Fortran 77 (and FORTRAN IV) code
- Domain Wall Fermions

Parallelising lattice field theory code

- 2+1d Thirring model
- RHMC algorithm
- Serial Fortran 77 (and FORTRAN IV) code
- Domain Wall Fermions
 - Highly vectorisable

Parallelising lattice field theory code

- 2+1d Thirring model
- RHMC algorithm
- Serial Fortran 77 (and FORTRAN IV) code
- Domain Wall Fermions
 - Highly vectorisable
- Need for:

Parallelising lattice field theory code

- 2+1d Thirring model
- RHMC algorithm
- Serial Fortran 77 (and FORTRAN IV) code
- Domain Wall Fermions
 - Highly vectorisable
- Need for:
 - Larger volumes \Rightarrow longer time per iteration

Parallelising lattice field theory code

- 2+1d Thirring model
- RHMC algorithm
- Serial Fortran 77 (and FORTRAN IV) code
- Domain Wall Fermions
 - Highly vectorisable
- Need for:
 - Larger volumes \Rightarrow longer time per iteration
 - Stronger coupling \Rightarrow more iterations

Parallelising lattice field theory code

- 2+1d Thirring model
- RHMC algorithm
- Serial Fortran 77 (and FORTRAN IV) code
- Domain Wall Fermions
 - Highly vectorisable
- Need for:
 - Larger volumes \Rightarrow longer time per iteration
 - Stronger coupling \Rightarrow more iterations
- \Rightarrow Need parallelism

Initial tests

- Tune compiler options: 40% performance improvement

Initial tests

- Tune compiler options: 40% performance improvement
 - Before: `ifort -O3 -heap-arrays`

Initial tests

- Tune compiler options: 40% performance improvement
 - Before: `ifort -O3 -heap-arrays`
 - After:
`ifort -ipo -no-prec-div -fp-model fast=2 -xHost -O3 -heap-arrays`

Initial tests

- Tune compiler options: 40% performance improvement
 - Before: `ifort -O3 -heap-arrays`
 - After:
`ifort -ipo -no-prec-div -fp-model fast=2 -xHost -O3 -heap-arrays`
- Automatic multithreading (`-parallel`)

Initial tests

- Tune compiler options: 40% performance improvement
 - Before: `ifort -O3 -heap-arrays`
 - After:
`ifort -ipo -no-prec-div -fp-model fast=2 -xHost -O3 -heap-arrays`
- Automatic multithreading (`-parallel`)
 - Near linear scaling to 4 threads

Initial tests

- Tune compiler options: 40% performance improvement
 - Before: `ifort -O3 -heap-arrays`
 - After:
`ifort -ipo -no-prec-div -fp-model fast=2 -xHost -O3 -heap-arrays`
- Automatic multithreading (`-parallel`)
 - Near linear scaling to 4 threads
 - Rapid fall off afterwards

Initial tests

- Tune compiler options: 40% performance improvement
 - Before: `ifort -O3 -heap-arrays`
 - After:
`ifort -ipo -no-prec-div -fp-model fast=2 -xHost -O3 -heap-arrays`
- Automatic multithreading (`-parallel`)
 - Near linear scaling to 4 threads
 - Rapid fall off afterwards
- Deliver $\sim 5\times$ improvement to researcher before starting MPI

Refactoring

- Implement regression testing

Refactoring

- Implement regression testing
- Add per-site random numbers

Refactoring

- Implement regression testing
- Add per-site random numbers
- Remove indirection

Refactoring

- Implement regression testing
- Add per-site random numbers
- Remove indirection
- Move to free-form Fortran

Refactoring

- Implement regression testing
- Add per-site random numbers
- Remove indirection
- Move to free-form Fortran
- Replace loop operations with array operations where possible

```
      do 1000 i=1,10000  
1000    sum_x = sum_x + x(i)
```

becomes

```
sum_x = sum(x)
```

Refactoring

- Implement regression testing
- Add per-site random numbers
- Remove indirection
- Move to free-form Fortran
- Replace loop operations with array operations where possible

```
do 1000 i=1,10000  
1000    sum_x = sum_x + x(i)
```

becomes

```
sum_x = sum(x)
```

- `implicit none`

Refactoring

- Implement regression testing
- Add per-site random numbers
- Remove indirection
- Move to free-form Fortran
- Replace loop operations with array operations where possible

```
do 1000 i=1,10000  
1000 sum_x = sum_x + x(i)
```

becomes

```
sum_x = sum(x)
```

- `implicit none`
- `common` blocks → modules

Refactoring

- Implement regression testing
- Add per-site random numbers
- Remove indirection
- Move to free-form Fortran
- Replace loop operations with array operations where possible

```
      do 1000 i=1,10000  
1000    sum_x = sum_x + x(i)
```

becomes

```
sum_x = sum(x)
```

- `implicit none`
- `common` blocks → modules
- Global parameters → module

Adding MPI

- 1-site halo in space-time

Adding MPI

- 1-site halo in space-time
- Explicit halo communications in serial

Adding MPI

- 1-site halo in space-time
- Explicit halo communications in serial
- Add parallel communications

Adding MPI

- 1-site halo in space-time
- Explicit halo communications in serial
- Add parallel communications
 - Broadcast parameters

Adding MPI

- 1-site halo in space-time
- Explicit halo communications in serial
- Add parallel communications
 - Broadcast parameters
 - Use MPI-IO

Adding MPI

- 1-site halo in space-time
- Explicit halo communications in serial
- Add parallel communications
 - Broadcast parameters
 - Use MPI-IO
 - Communicate halo

Adding MPI

- 1-site halo in space-time
- Explicit halo communications in serial
- Add parallel communications
 - Broadcast parameters
 - Use MPI-IO
 - Communicate halo
 - Use collective reductions (sum, max, etc.)

Adding MPI

- 1-site halo in space-time
- Explicit halo communications in serial
- Add parallel communications
 - Broadcast parameters
 - Use MPI-IO
 - Communicate halo
 - Use collective reductions (sum, max, etc.)
 - Seed RNG based on rank

Adding MPI

- 1-site halo in space-time
- Explicit halo communications in serial
- Add parallel communications
 - Broadcast parameters
 - Use MPI-IO
 - Communicate halo
 - Use collective reductions (sum, max, etc.)
 - Seed RNG based on rank
 - Handle correlation functions

Adding MPI

- 1-site halo in space-time
- Explicit halo communications in serial
- Add parallel communications
 - Broadcast parameters
 - Use MPI-IO
 - Communicate halo
 - Use collective reductions (sum, max, etc.)
 - Seed RNG based on rank
 - Handle correlation functions
 - Hide communications

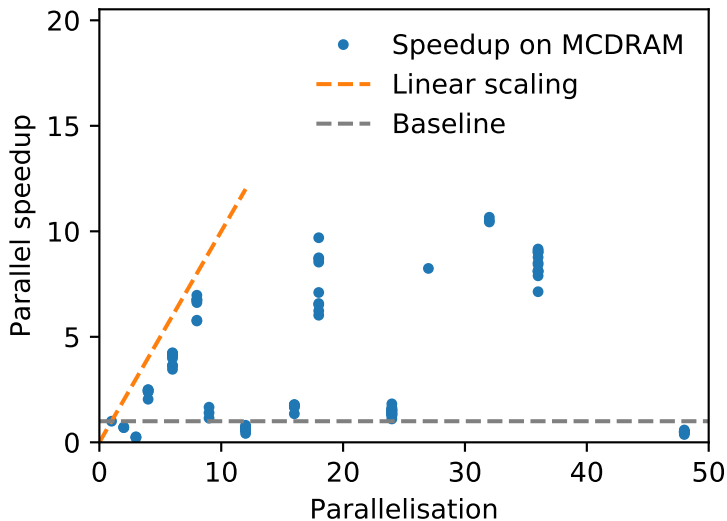
Adding MPI

- 1-site halo in space-time
- Explicit halo communications in serial
- Add parallel communications
 - Broadcast parameters
 - Use MPI-IO
 - Communicate halo
 - Use collective reductions (sum, max, etc.)
 - Seed RNG based on rank
 - Handle correlation functions
 - Hide communications
 - Minimise changes to existing code structure

Adding MPI

- 1-site halo in space-time
- Explicit halo communications in serial
- Add parallel communications
 - Broadcast parameters
 - Use MPI-IO
 - Communicate halo
 - Use collective reductions (sum, max, etc.)
 - Seed RNG based on rank
 - Handle correlation functions
 - Hide communications
 - Minimise changes to existing code structure
- Test, test, test

MPI Performamnce



Ongoing work

- Full physics tests

Ongoing work

- Full physics tests
- Investigate unexpected MPI slowdowns

Ongoing work

- Full physics tests
- Investigate unexpected MPI slowdowns
- Test combining with automated multithreading

Other projects

- Port openQCD to AVX 512 (KNL, Skylake)

Other projects

- Port openQCD to AVX 512 (KNL, Skylake)
- Port Windows GPU code to Linux

Other projects

- Port openQCD to AVX 512 (KNL, Skylake)
- Port Windows GPU code to Linux
- Prepare benchmark report supporting PRACE application

Other projects

- Port openQCD to AVX 512 (KNL, Skylake)
- Port Windows GPU code to Linux
- Prepare benchmark report supporting PRACE application
- Develop accessible desktop client for HPC

Other projects

- Port openQCD to AVX 512 (KNL, Skylake)
- Port Windows GPU code to Linux
- Prepare benchmark report supporting PRACE application
- Develop accessible desktop client for HPC
- Group theory-based memory compression for $Sp(2N)$ lattice gauge theory

Other projects

- Port openQCD to AVX 512 (KNL, Skylake)
- Port Windows GPU code to Linux
- Prepare benchmark report supporting PRACE application
- Develop accessible desktop client for HPC
- Group theory-based memory compression for $Sp(2N)$ lattice gauge theory
- Speeding up inter-process communication between Python and C++

Thanks for listening!



Slides: git.io/fNTis

Thanks for listening!

- Interested?



Slides: git.io/fNTis

Thanks for listening!

- Interested?
- Currently recruiting



Thanks for listening!

- Interested?
- Currently recruiting
- bit.ly/swansea-rse-2018



Thanks for listening!

- Interested?
- Currently recruiting
- bit.ly/swansea-rse-2018
- Deadline: 20 July (next Friday)

