NATIVE MODE PORTING CASE STUDY

Adrian Jackson

adrianj@epcc.ed.ac.uk

Native mode porting

- Porting large FORTRAN codes
  - No code changes
  - Re-compile
  - Add linking to MKL
- MPI parallelised code
  - Some hybrid or OpenMP (small numbers of threads)
- Native mode to reduce code modifications required

GS2

- Flux-tube gyrokinetic code
  - Initial value code
  - Solves the gyrokinetic equations for perturbed distribution functions together with Maxwell's equations for the turbulent electric and magnetic fields
  - Linear (fully implicit) and Non-linear (dealiased pseudo-spectral) collisional and field terms
  - 5D space – 3 spatial, 2 velocity
  - Different species of charged particles
- Advancement of time in Fourier space
- Non-linear term calculated in position space
  - Requires FFTs
  - FFTs only in two spatial dimensions perpendicular to the magnetic field
- Heavily dominated by MPI time at scale
  - Especially with collisions

New hybrid implementation

- Funneled communication model
- OpenMP done at a high level in the code
- Single parallel region per time step
  - Better can be achieved (single parallel region per run)
- Some code excluded but computationally expensive code all hybridised

MPI processes	OpenMP threads	Execution time (seconds)
192	1	16.54
96	2	18.34
64	3	16.46
48	4	30.86
32	6	28.3

epcc

Port to Xeon Phi

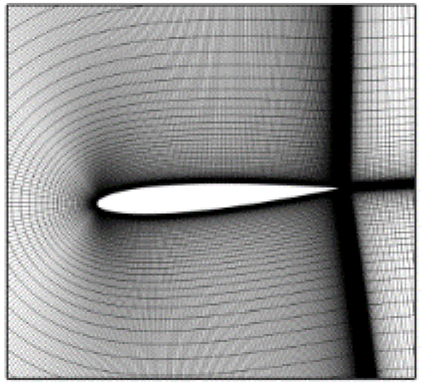
- Pure MPI code performance:
  - ARCHER (2x12 core Xeon E5-2697, 16 MPI processes): 3.08 minutes
  - Host (2x8 core Xeon E5-2650, 16 MPI processes): 4.64 minutes
  - 1 Phi (176 MPI processes): 7.34 minutes
  - 1 Phi (235 MPI processes): 6.77 minutes
  - 2 Phis (352 MPI processes): 47.71 minutes
- Hybrid code performance
  - 1 Phi (80 MPI processes, 3 threads each): 7.95 minutes
  - 1 Phi (120 MPI processes, 2 threads each): 7.07 minutes

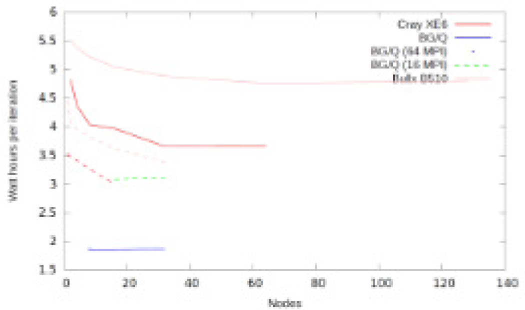
Complex number optimisation

- Much of GS2 uses FORTRAN Complex numbers
  - However, often imaginary and real parts are treated separately
  - Can affect vectorisation performance
- Work underway to replace with separate arrays
  - Initial performance numbers demonstrate performance improvement on Xeon Phi
  - 2-3% for a single routine when using separate arrays

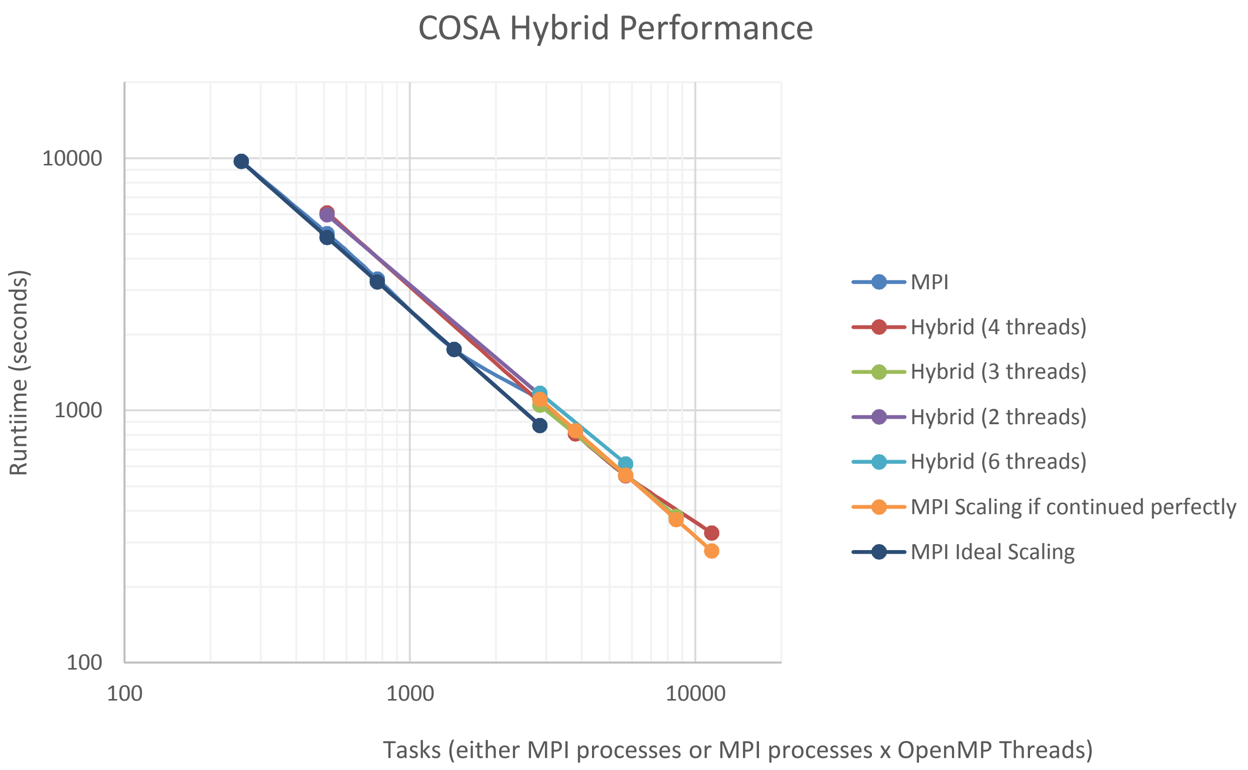
COSA

- Fluid dynamics code
  - Harmonic balance (frequency domain approach)
  - Unsteady navier-stokes solver
  - Optimise performance of turbo-machinery like problems
  - Multi-grid, multi-level, multi-block code
  - Parallelised with MPI and with MPI+OpenMP





epcc


COSA Hybrid Performance
Runtiime (seconds)
10000
1000
100
100
1000
10000
Tasks (either MPI processes or MPI processes x OpenMP Threads)
MPI
Hybrid (4 threads)
Hybrid (3 threads)
Hybrid (2 threads)
Hybrid (6 threads)
MPI Scaling if continued perfectly
MPI Ideal Scaling

epcc

Xeon Phi Performance

Configuration	Number of hardware elements	Occupancy	Runtime (s)
8 MPI processes	1/2	8/16	2105.71
16 MPI processes	2/2	16/16	1272.54
64 MPI processes	1/2	64/240	3874.45
64 MPI processes 3 OpenMP threads	1/2	192/240	2963.58
118 MPI processes 4 OpenMP threads	2/2	472/480	2118.05
128 MPI processes 3 OpenMP threads	2/2	384/480	1759.30

- Hardware:
  - 2 x Xeon Sandy Bridge 8-core E5-2650 2.00GHz
  - 2 x Xeon Phi 5110P 60-core 1.05GHz
- Test case
  - 256 blocks
  - Maximum 7 OpenMP threads

Serial optimisations

- Manual removal of floating point loop invariants divisions

```
do ipde = 1,4
      fac1 = fact * vol(i,j)/dt
end do


recip = 1.0d / dt
do ipde = 1,4
      fact1 = fact * vol(i,j) * recip
end do
```

- Provides ~15% speedup so far on Xeon Phi
  - No real benefit noticed on host
  - Changes the results

I/O

- Identified that reading input is now significant overhead for this code
  - Output is done using MPI-I/O, reading is done serially
  - File locking overhead grows with process count
  - Large cases ~GB input files
- Parallelised reading data
  - Reduce file locking and serial parts of the code
- One or two orders of magnitude improvement in performance at large process counts
  - 1 minute down to 5 seconds

Future work

Configuration	Number of hardware elements	Occupancy	Runtime (s)
8 MPI processes	1/2	8/16	2105.71
16 MPI processes	2/2	16/16	1272.54
128 MPI processes	1/2	128/240	1903.51
64 MPI processes 3 OpenMP threads	1/2	192/240	2214.56
128 MPI processes 3 OpenMP threads	2/2	384/480	1503.45

- Further serial optimisation
  - Cache blocking
- 3D version of the code now developed
  - Porting optimised and hybrid version to this

epcc