ARCHER VIRTUAL TUTORIAL

HYBRIDISATION

ADDING OPENMP TO MPI FOR THE PLASMA SIMULATION CODE GS2

Adrian Jackson, EPCC
Colin Roach, CCFE

epcc

THE UNIVERSITY OF EDINBURGH

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) 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
- Originally developed in the US
  - Heavily developed and optimised in UK

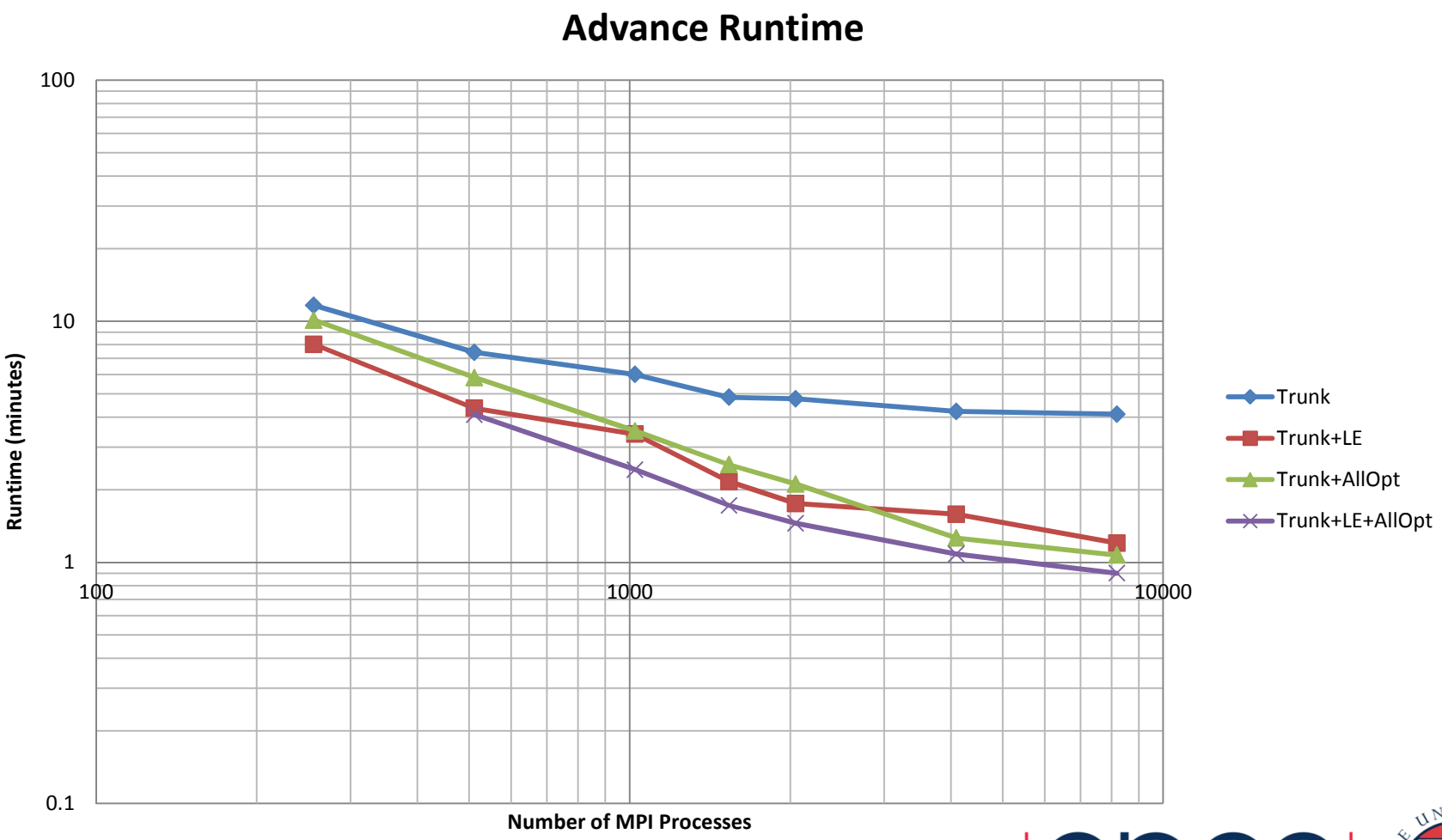
epcc | THE UNIVERSITY OF EDINBURGH

GS2 structure

- Initialisation
  - Evaluate distribution function (lots of times)
- Computation
  - Time loop
    - Evaluate distribution function
      - Calculate nonlinear terms
      - Calculate linear terms
      - Do some other stuff
      - Calculate collisions
    - Calculate fields
    - Evaluate distribution function

epcc
THE UNIVERSITY OF EDINBURGH

GS2 Performance

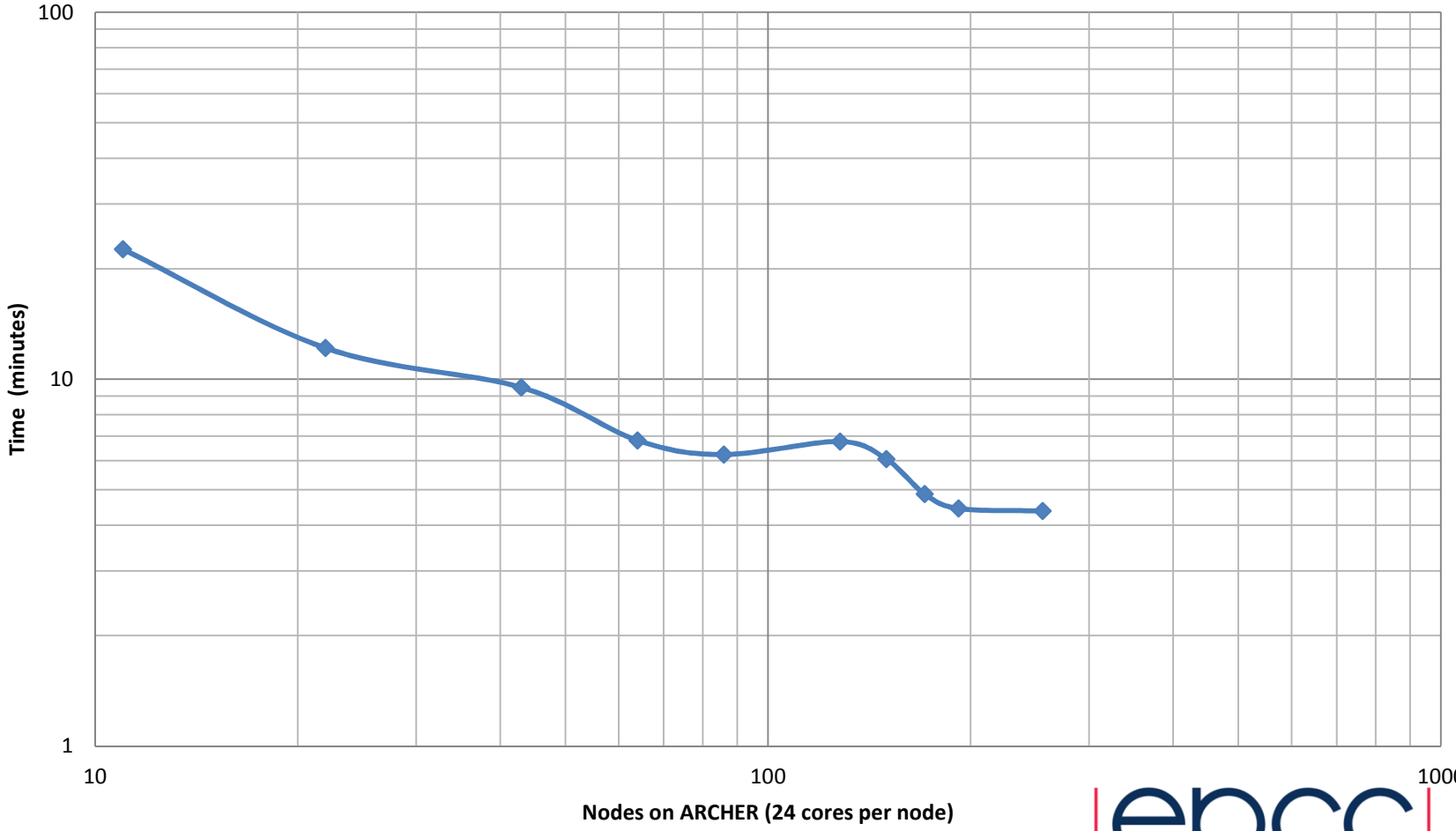


epcc

THE UNIVERSITY OF EDINBURGH

GS2 Performance

- MPI performance

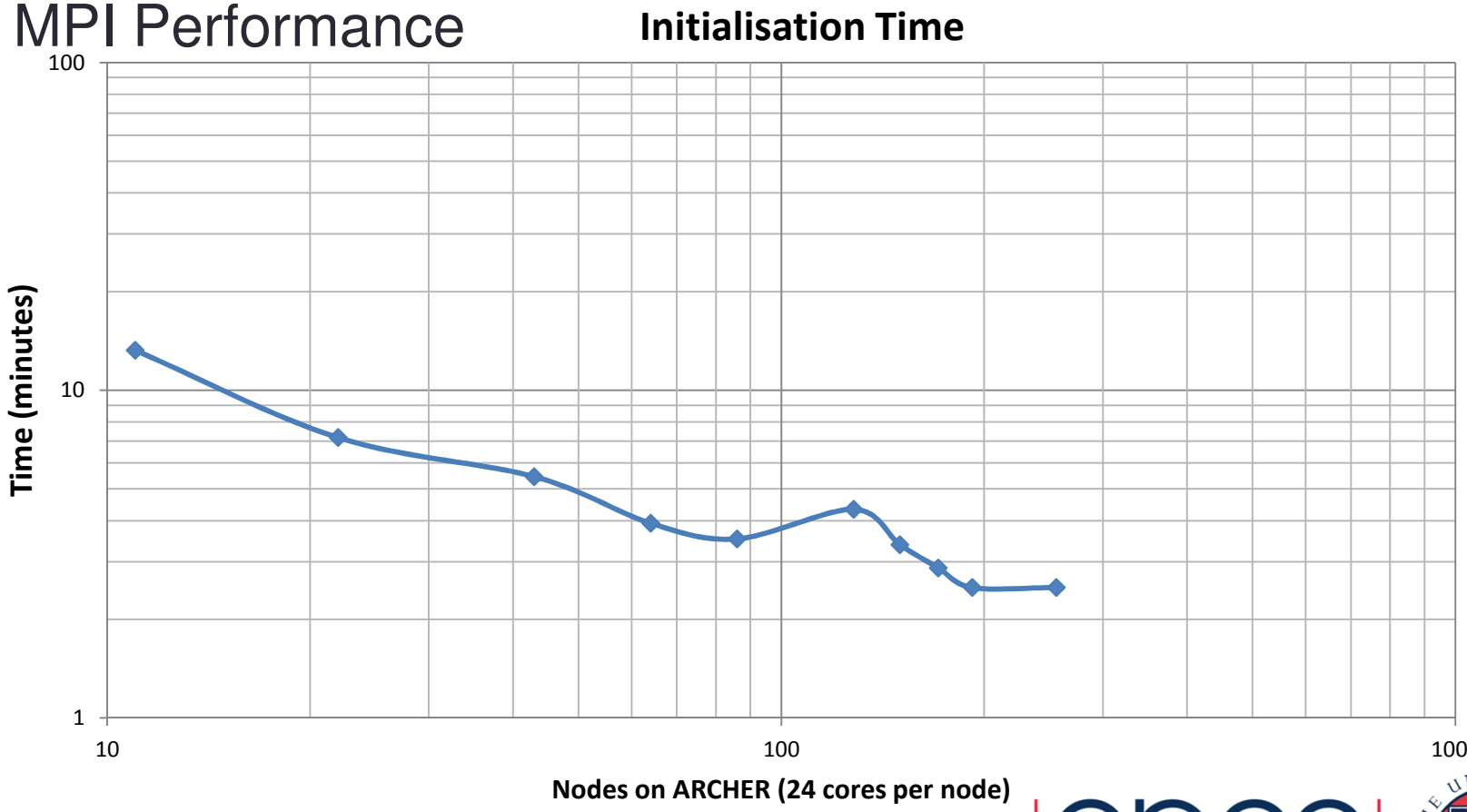


epcc

THE UNIVERSITY OF EDINBURGH

GS2 Performance

- MPI Performance

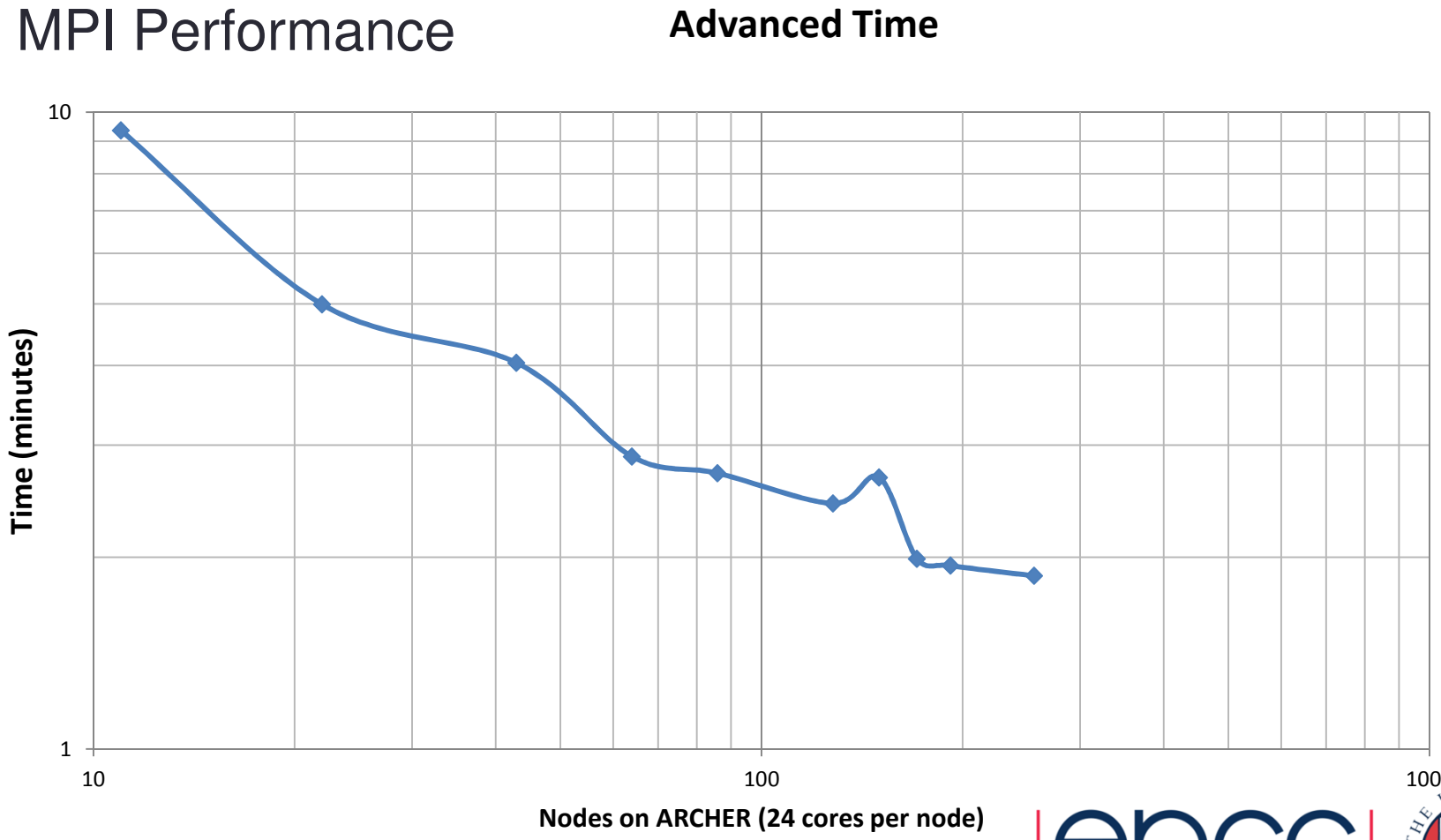


epcc

THE UNIVERSITY OF EDINBURGH

GS2 Performance

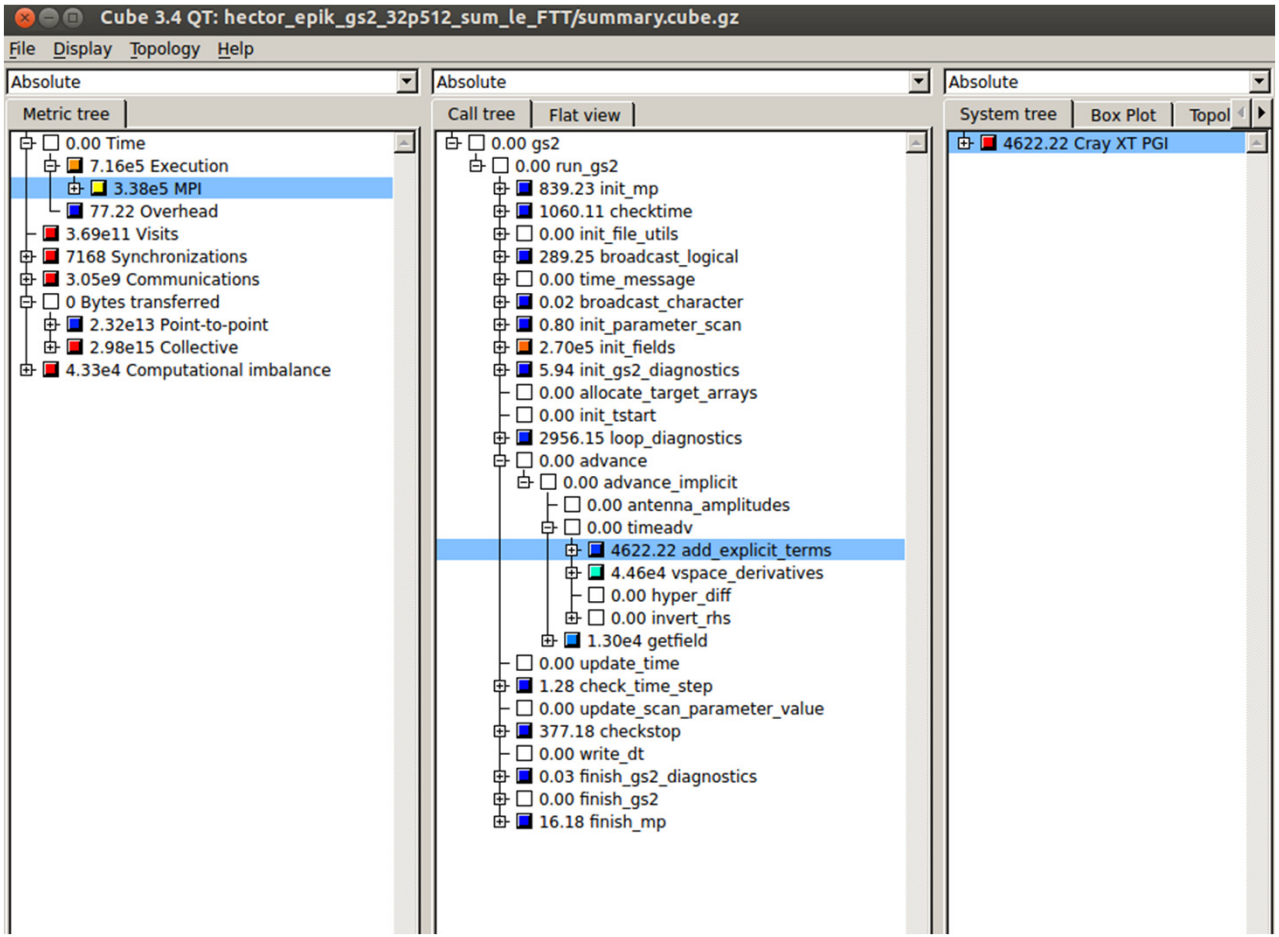
- MPI Performance



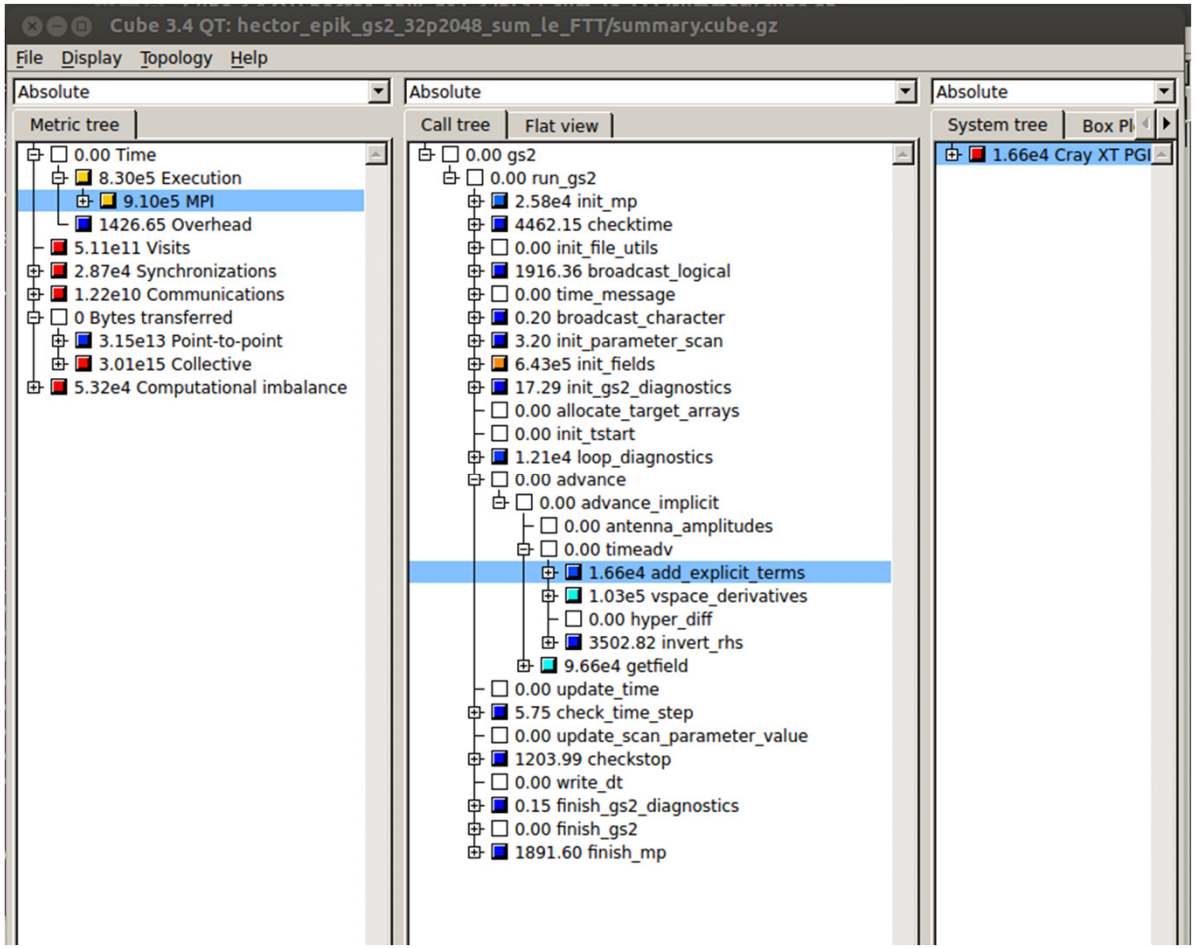
epcc

THE UNIVERSITY OF EDINBURGH

Initial scalasca: comms 512 procs



Initial scalasca: comms 2048 procs



epcc

THE UNIVERSITY OF EDINBURGH

CrayPat results, full test cases 512 cores

```
 Samp% |    Samp |    Imb. |  Imb. |Group
       |         |    Samp | Samp% | Function
       |         |         |       |  PE=HIDE

 100.0% | 80297.2 |      -- |    -- |Total
|-------------------------------------------------------------------------------
| 47.0% | 37713.5 |      -- |    -- |MPI
||------------------------------------------------------------------------------
||  27.7% | 22207.4 | 27269.6 | 55.2% |mpi_recv
||  10.8% |  8696.7 |  6267.3 | 42.0% |MPI_SEND
||   5.6% |  4494.3 |  7790.7 | 63.5% |MPI_ALLREDUCE
||   2.2% |  1788.8 |    68.2 |  3.7% |MPI_BARRIER
||==============================================================================
| 43.5% | 34957.6 |      -- |    -- |USER
||------------------------------------------------------------------------------
||   8.5% |  6803.0 |  1173.0 | 14.7% |collisions_mp_conserve_diffuse_le_layout_
||   5.1% |  4132.3 |   632.7 | 13.3% |collisions_mp_conserve_lorentz_le_layout_
||   3.2% |  2543.0 |   486.0 | 16.1% |dist_fn_mp_get_source_term_
||   2.7% |  2180.2 |   859.8 | 28.3% |le_grids_mp_integrate_moment_lec_
||   2.7% |  2162.4 |  1019.6 | 32.1% |dist_fn_arrays_mp_g_adjust_
||   2.6% |  2053.4 |   155.6 |  7.1% |dist_fn_mp_invert_rhs_1_
||   2.5% |  1999.9 |   451.1 | 18.4% |dist_fnget_source_term_mp_set_source_
||   2.3% |  1835.8 |   235.2 | 11.4% |nonlinear_terms_mp_add_nl_
||   1.7% |  1350.4 |   418.6 | 23.7% |dist_fn_mp_getan_
||   1.7% |  1342.5 |   281.5 | 17.4% |redistribute_mp_c_redist_33_inv_
||   1.7% |  1334.8 |   319.2 | 19.3% |redistribute_mp_c_redist_33_
||   1.5% |  1219.0 |   271.0 | 18.2% |collisions_mp_solfp_lorentz_le_layout_..0
||   1.4% |  1144.1 |   261.9 | 18.7% |dist_fn_mp_getfieldeq1_
||   1.4% |  1136.7 |   226.3 | 16.6% |collisions_mp_solfp_ediffuse_le_layout_
||   1.2% |   940.8 |   185.2 | 16.5% |gs2_transforms_mp_transform2_5d_accel_
||==============================================================================
|  9.5% |  7602.4 |      -- |    -- |ETC
||------------------------------------------------------------------------------
||   3.9% |  3153.0 |   917.0 | 22.6% |__intel_memset
||   1.4% |  1159.2 |   659.8 | 36.3% |copy
||   1.3% |  1035.1 |   153.9 | 13.0% |hc2cb_12
|==============================================================================
```

epcc

THE UNIVERSITY OF EDINBURGH

CrayPat results, full test case 1024 cores

```
 Samp% |    Samp |    Imb. |  Imb. |Group
       |         |    Samp | Samp% | Function
       |         |         |       |  PE=HIDE

 100.0% | 55501.4 |      -- |    -- |Total
|-------------------------------------------------------------------------------
| 65.0% | 36071.4 |      -- |    -- |MPI
||------------------------------------------------------------------------------
||  41.5% | 23028.4 | 12175.6 | 34.6% |mpi_recv
||  11.4% |  6325.0 |  5567.0 | 46.9% |MPI_ALLREDUCE
||   7.7% |  4248.6 |  3371.4 | 44.3% |MPI_SEND
||   3.0% |  1690.7 |    57.3 |  3.3% |MPI_BARRIER
||   1.2% |   683.9 |   919.1 | 57.4% |mpi_bcast
||==============================================================================
| 28.8% | 15982.8 |      -- |    -- |USER
||------------------------------------------------------------------------------
||   4.1% |  2260.4 |   393.6 | 14.8% |collisions_mp_conserve_diffuse_le_layout_
||   2.3% |  1268.3 |   320.7 | 20.2% |dist_fn_mp_get_source_term_
||   2.1% |  1190.2 |   246.8 | 17.2% |collisions_mp_conserve_lorentz_le_layout_
||   1.8% |  1015.2 |   171.8 | 14.5% |dist_fn_mp_invert_rhs_1_
||   1.8% |   995.1 |   283.9 | 22.2% |dist_fn_mp_getfieldeq1_
||   1.8% |   993.7 |   229.3 | 18.8% |dist_fnget_source_term_mp_set_source_
||   1.5% |   846.7 |   211.3 | 20.0% |nonlinear_terms_mp_add_nl_
||   1.5% |   822.9 |   453.1 | 35.5% |dist_fn_arrays_mp_g_adjust_
||   1.3% |   733.5 |   174.5 | 19.2% |le_grids_mp_integrate_moment_lec_
||   1.2% |   692.0 |   332.0 | 32.5% |dist_fn_mp_getan_
||   1.1% |   606.4 |   141.6 | 18.9% |redistribute_mp_c_redist_33_
||   1.1% |   605.8 |   188.2 | 23.7% |redistribute_mp_c_redist_22_
||   1.1% |   602.9 |   173.1 | 22.3% |redistribute_mp_c_redist_33_inv_
||==============================================================================
|  6.2% |  3436.2 |      -- |    -- |ETC
||------------------------------------------------------------------------------
||   3.0% |  1688.9 |   484.1 | 22.3% |__intel_memset
||   1.2% |   684.4 |   167.6 | 19.7% |apply
|==============================================================================
```

epcc

THE UNIVERSITY OF EDINBURGH

Decompositions

- For GS2 increased MPI processes means splitting data domain further:
  - Can decompose: species, energy, ky, kx, lambda differently (right to left)
  - xyles, yxles, lyxes, yxels, lxyes, lexys
- K space
  - Main domain for computations
  - g_lo(ig,isgn,iglo): ig, isgn local; iglo decomposed according to “layout”
  - Have to go to real space for non-linear computations
  - Have to go to collision space for collision computations

epcc
THE UNIVERSITY OF EDINBURGH

Design decisions to consider

- What level to add OpenMP at?
  - Parallel region for each subroutine called:
    - Pros: Easy to control where parallelism occurs and variable sharing, easy to exclude code that we don't want to parallelise with OpenMP, don't have to worry about multiple program units (i.e. FORTRAN modules)
    - Cons: Can't nest parallel regions, hard for routines that are called from different places, more overheads as a parallel region created and destroyed for each routine call, enables lots of code to be easily not parallelised (performance impct)
  - High level parallel region for multiple subroutines:
    - Pros: Reduced overheads as only one parallel region call for a whole group of subroutines, forces more code to be at least included in parallel regions
    - Cons: Orphaned directives and data scoping means having to rely on implicit definitions of variable sharing, requires more work to exclude code that cannot be parallelised with OpenMP

epcc
THE UNIVERSITY OF EDINBURGH

Design decisions to consider

- How to handle MPI communications, what level of threaded MPI communications to support/require?
  - MPI_Init_thread replaces MPI_Init
    - Supports 4 different levels:
      - **MPI_THREAD_SINGLE** Only one thread will execute.
      - **MPI_THREAD_FUNNELED** The process may be multi-threaded, but only the main thread will make MPI calls (all MPI calls are funneled to the main thread).
      - **MPI_THREAD_SERIALIZED** The process may be multi-threaded, and multiple threads may make MPI calls, but only one at a time: MPI calls are not made concurrently from two distinct threads (all MPI calls are serialized).
      - **MPI_THREAD_MULTIPLE** Multiple threads may call MPI, with no restrictions.
  - Where to do MPI communications:
    - Single or funneled:
      - Pros: Don't have to change MPI implemented in the code
      - Cons: Only one thread used for communications leaves cores inactive, not parallelising all the code
    - Serialized
      - Pros: Can parallelism MPI code using OpenMP as well, meaning further parallelism
      - Cons: Still not using all cores for MPI communications, requires thread safe version of the MPI library
    - Multiple:
      - Pros: All threads can do work, not leaving idle cores
      - Cons: May requires changes to MPI code to create MPI communicators for separate threads to work on, and for collective communications. Can require ordered OpenMP execution for MPI collectives, experience shows fully threaded MPI implementations slower than ordinary MPI
- ARCHER runtime variable:
  - export MPICH_MAX_THREAD_SAFETY=funneled

Hybrid implementation

- Funneled communication model
  - Actually in practise done with OpenMP Master regions and barriers
- OpenMP done at a high level in the code
  - Implemented at the Evaluate distribution function level
  - Called twice per time step (and many times in initialisation)
- Single parallel region per time step
  - Orphaned OpenMP directives
  - Better can be achieved (single parallel region per run)
- Some code excluded but computationally expensive code all hybridised

epcc
THE UNIVERSITY OF EDINBURGH

Main changes

- Add OpenMP
  - Main loop
    - !$OMP PARALLEL DEFAULT(SHARED)
    - !$OMP WORKSHARE
    - gnew=gnew+g_fixpar
    - !$OMP END WORKSHARE
    - !$OMP END PARALLEL
  - Use pre-processing to turn OpenMP on and off
    - Not generally necessary (although useful for OpenMP routines and for compilers like the Cray compiler)
    - #ifdef OPENMP
    - !$OMP END PARALLEL
    - #endif

epcc | THE UNIVERSITY OF EDINBURGH

Main changes

- Default shared used at the highest level
  - All other data visibility functionality controlled by OpenMP default behaviour
  - Default behaviour
    - Module and global variables shared
    - Function/subroutine variables private
- Some subroutine variables we want to be shared

```
subroutine integrate_moment_c34 (g, total, all, full_arr)
…
complex, dimension(:,:,:,:),allocatable :: total_small
….
 allocate(total_small(-ntgrid:ntgrid,g_lo%it_min:g_lo%it_max,g_lo%ik_min:g_lo%ik_max,g_lo%is_min:g_lo%is_max))
…
#ifdef OPENMP
!$OMP DO PRIVATE(iglo,ik,it,ie,is,il)
#endif
    do iglo = g_lo%llim_proc, g_lo%ulim_proc
       ik = ik_idx(g_lo,iglo)
       it = it_idx(g_lo,iglo)
       ie = ie_idx(g_lo,iglo)
       is = is_idx(g_lo,iglo)
       il = il_idx(g_lo,iglo)

       !Perform local sum
       c34_total_small(:, it, ik, is) = c34_total_small(:, it, ik, is) + &
            w(ie)*wl(:,il)*(g(:,1,iglo)+g(:,2,iglo))
    end do
#ifdef OPENMP
!$OMP END DO
#endif
```

epcc | THE UNIVERSITY OF EDINBURGH

Main changes

- Move variable definition to module level (or global level)

```
Allocate memory only on one OpenMP process
#ifdef OPENMP
!$OMP BARRIER
!$OMP MASTER
#endif
      allocate(c34_total_small(-ntgrid:ntgrid,g_lo%ntheta0,g_lo%naky,g_lo%nspec))
#ifdef OPENMP
!$OMP END MASTER
!$OMP BARRIER
#endif
```

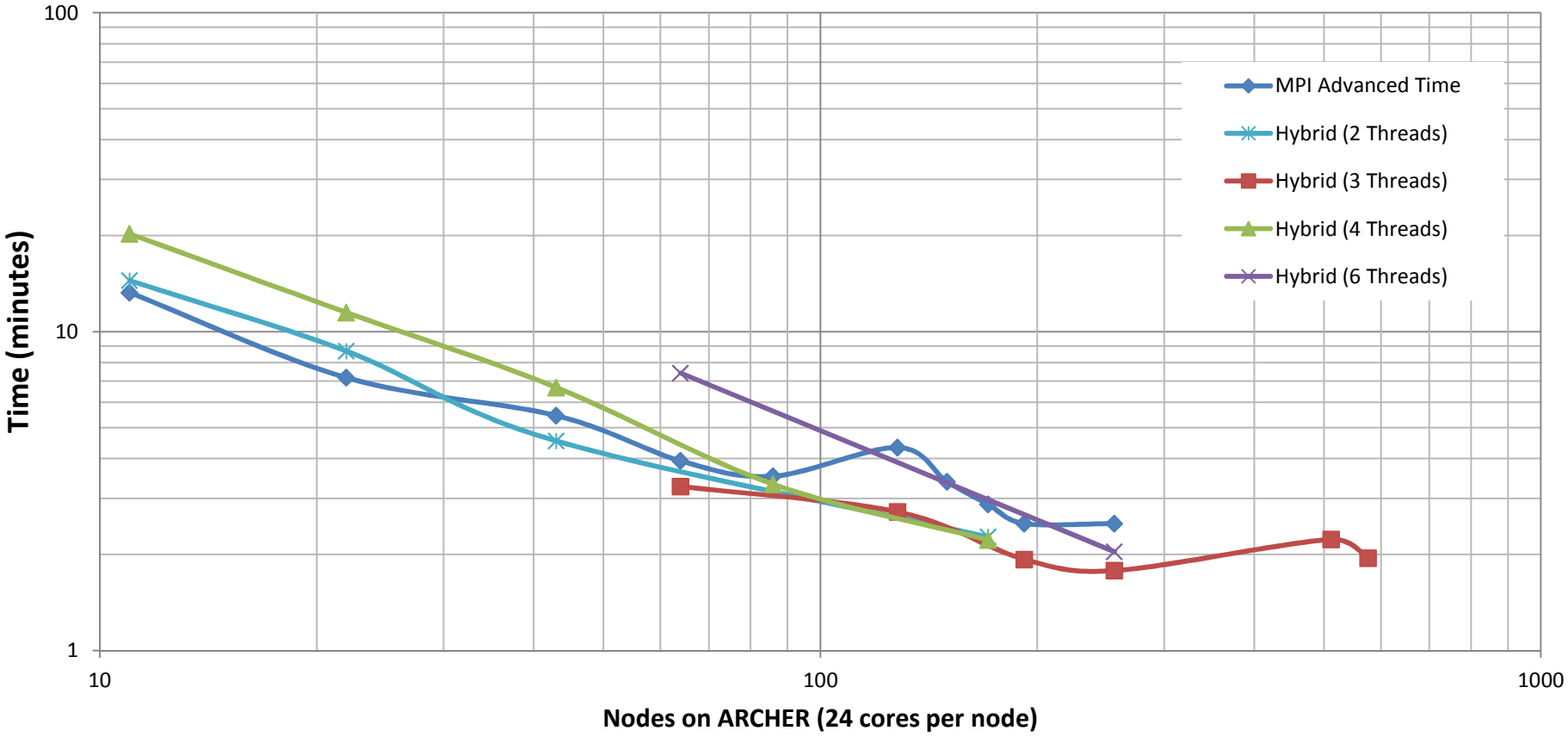
- Nastier for multiple routines in the same module that have local variables of the same name:
  - real, dimension (:,:,:), allocatable :: cdll_vns
  - real, dimension (:,:,:,:), allocatable ::vpadelnu
  - complex, dimension (:), allocatable :: **cdll_v0y0, cdll_v1y1, cdll_v2y2**
  - complex, dimension (:,:,:), allocatable :: **cdll_gtmp**
  - complex, dimension (:,:,:), allocatable :: **cdsl_gtmp**
  - real, dimension (:,:,:), allocatable :: cdsl_vns
  - complex, dimension (:,:,:,:), allocatable :: **cdsl_v0y0, cdsl_v1y1, cdsl_v2y2**
  - real, dimension (:,:,:,:), allocatable :: vpanud
  - complex, dimension (:), allocatable :: **clll_v0y0, clll_v1y1, clll_v2y2**
  - complex, dimension (:,:,:), allocatable :: **clll_gtmp**
  - real, dimension (:,:,:), allocatable :: clsl_vns
  - complex, dimension (:,:,:,:), allocatable :: **clsl_v0y0, clsl_v1y1, clsl_v2y2**
  - complex, dimension (:,:,:), allocatable :: **clsl_gtmp**
  - complex, dimension (:,:), allocatable :: slsl_glz, slsl_glzc
  - complex, dimension (:), allocatable :: slll_tmp
  - complex, dimension (:,:), allocatable :: sesl_ged

epcc | THE UNIVERSITY OF EDINBURGH

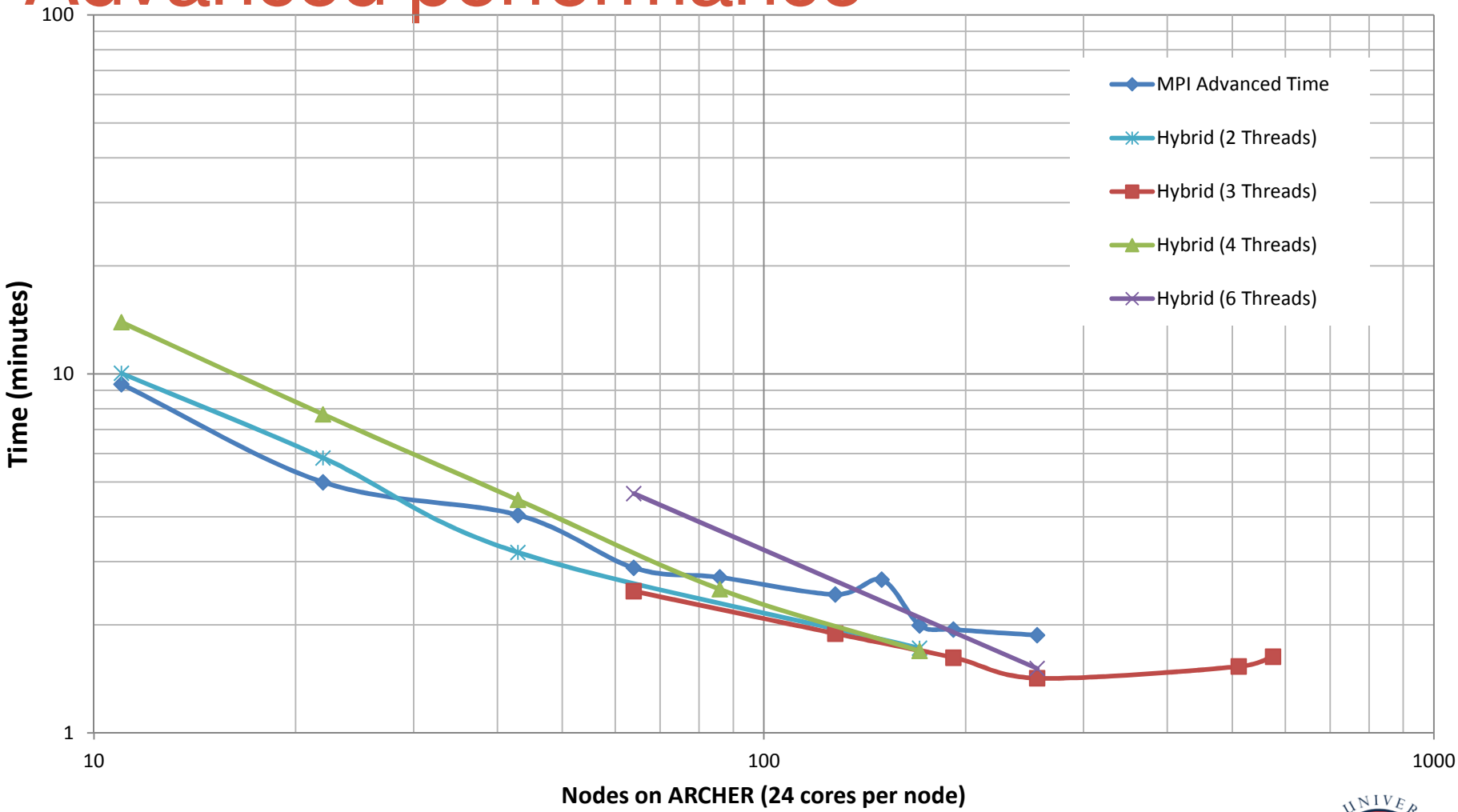
Running on ARCHER

```
#PBS -l select=22
export MPICH_MAX_THREAD_SAFETY=funneled

export OMP_NUM_THREADS=2
aprun -n 256 -N 12 -S 6 -d $OMP_NUM_THREADS -ss ./gs2_hybrid gs2.in

export OMP_NUM_THREADS=4
aprun -n 128 -N 6 -S 3 -d $OMP_NUM_THREADS -ss ./gs2_hybrid gs2.in

export OMP_NUM_THREADS=3
aprun -n 176 -N 8 -S 4 -d $OMP_NUM_THREADS -ss ./gs2_hybrid gs2.in
```

Initialisation performance



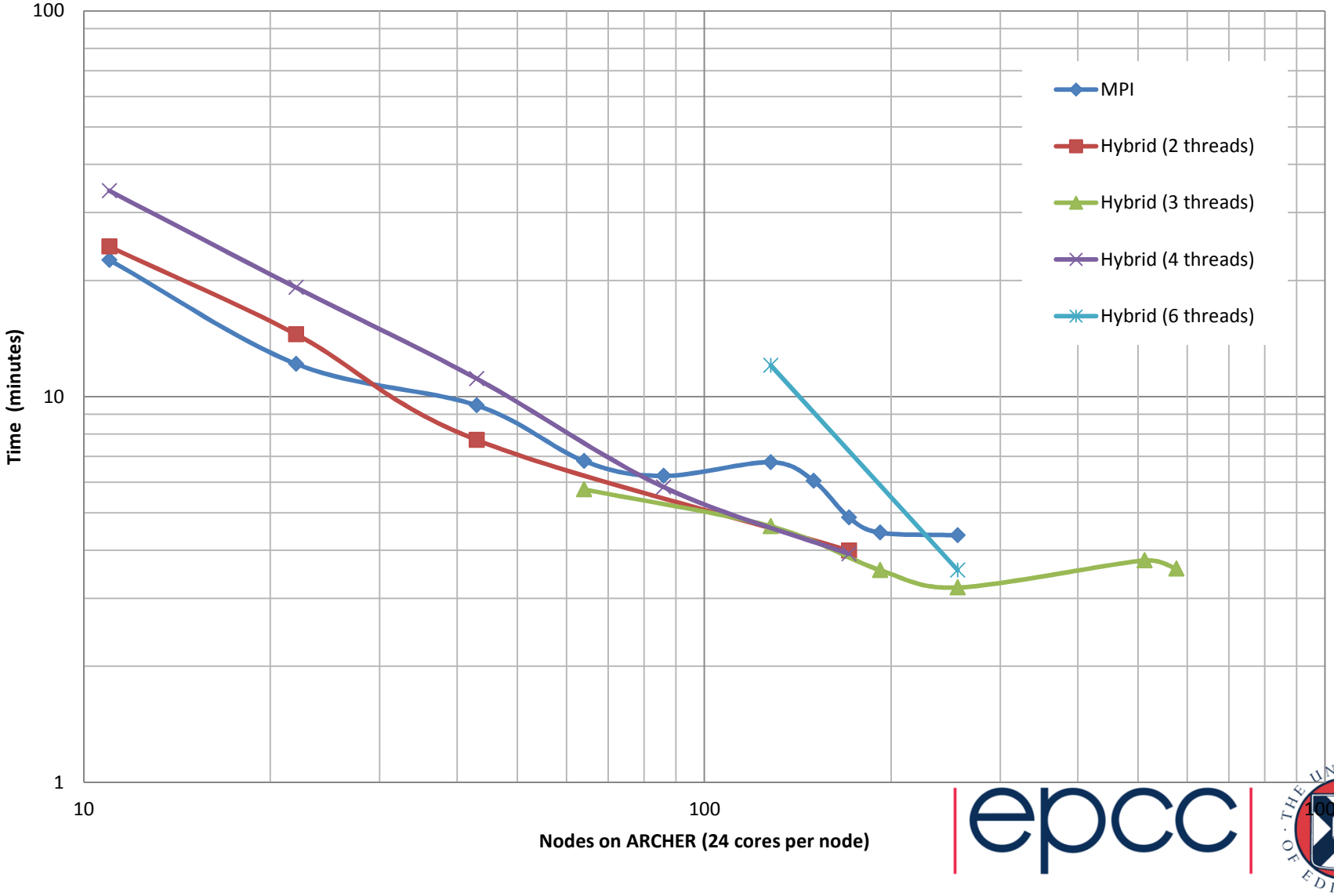
Advanced performance



epcc

THE UNIVERSITY OF EDINBURGH

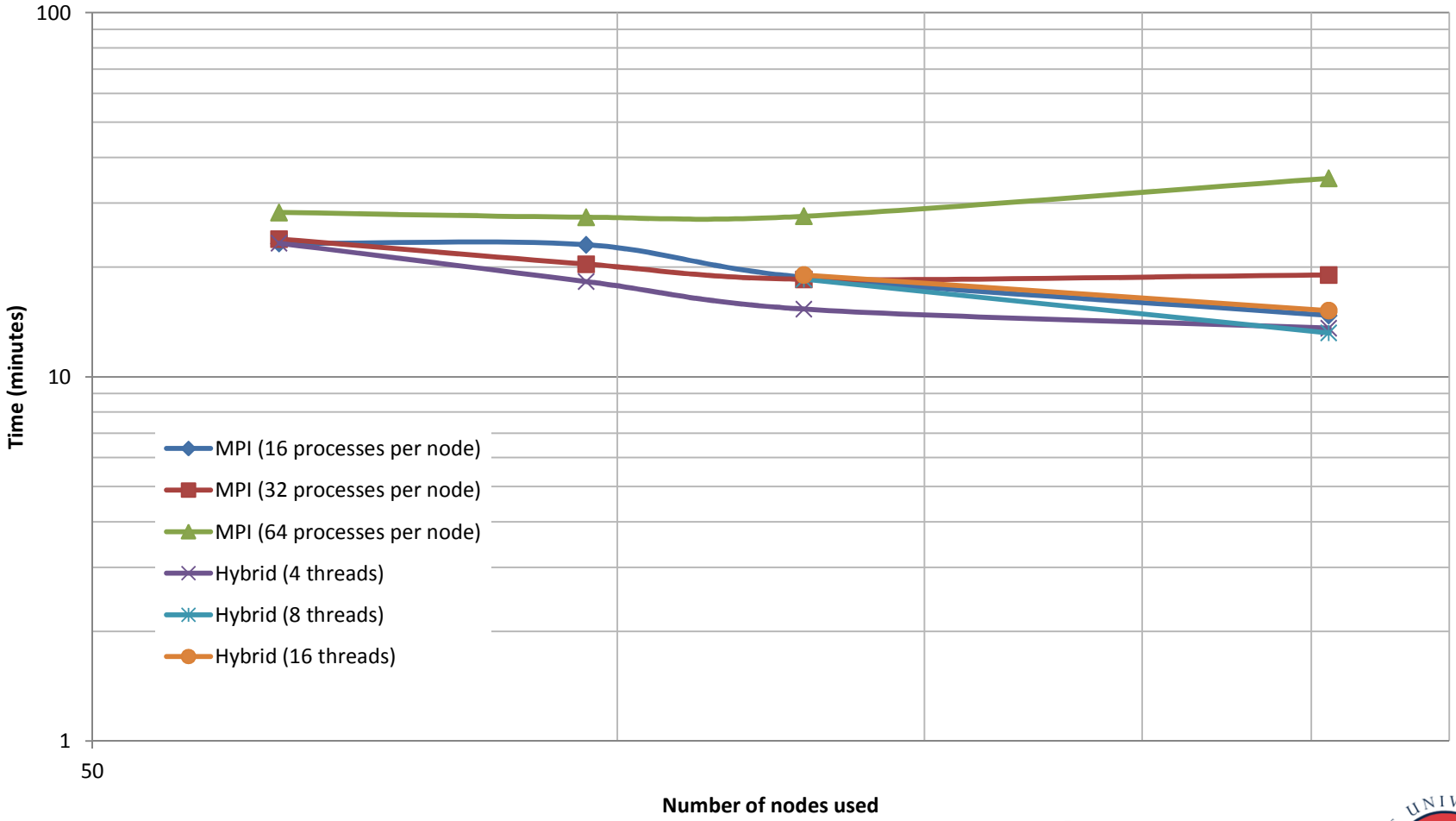
Total runtime



epcc

THE UNIVERSITY OF EDINBURGH

Total runtime – BG/Q



epcc

THE UNIVERSITY OF EDINBURGH

GS2 Port to Xeon Phi

- Profiling of vectorisation of GS2 shows good performance
- 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 Phi's (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

Acknowledgements

- Adrian Jackson, EPCC
- Colin Roach, EUTATOM/CCFE Fusion Association
- David Dickenson, York University
- Adrian Jackson was funded by the EPSRC Plasma Physics HEC Consortia EP/L000237/1. The Xeon Phi work was funded by EPCC's IPCC collaboration.

epcc | THE UNIVERSITY OF EDINBURGH