LOAD BALANCE AND PARALLEL I/O

Optimising COSA for scale

Adrian Jackson

adrianj@epcc.ed.ac.uk

@adrianjhpc

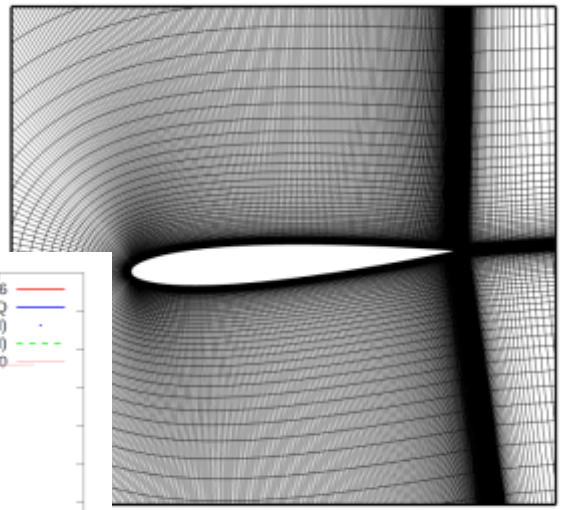
Sergio M. Campobasso

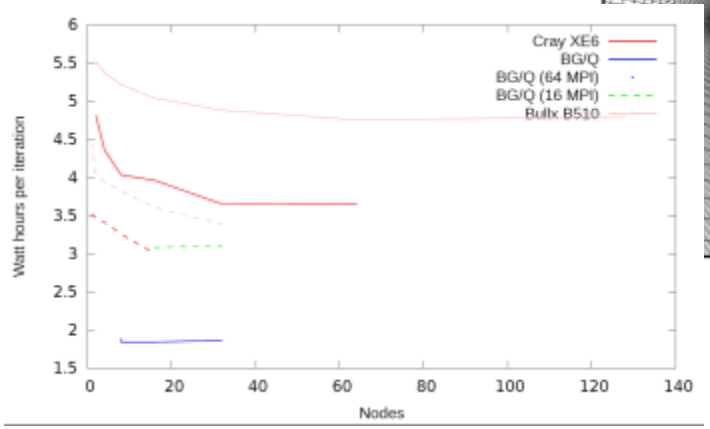
University of Lancaster

epcc

COSA

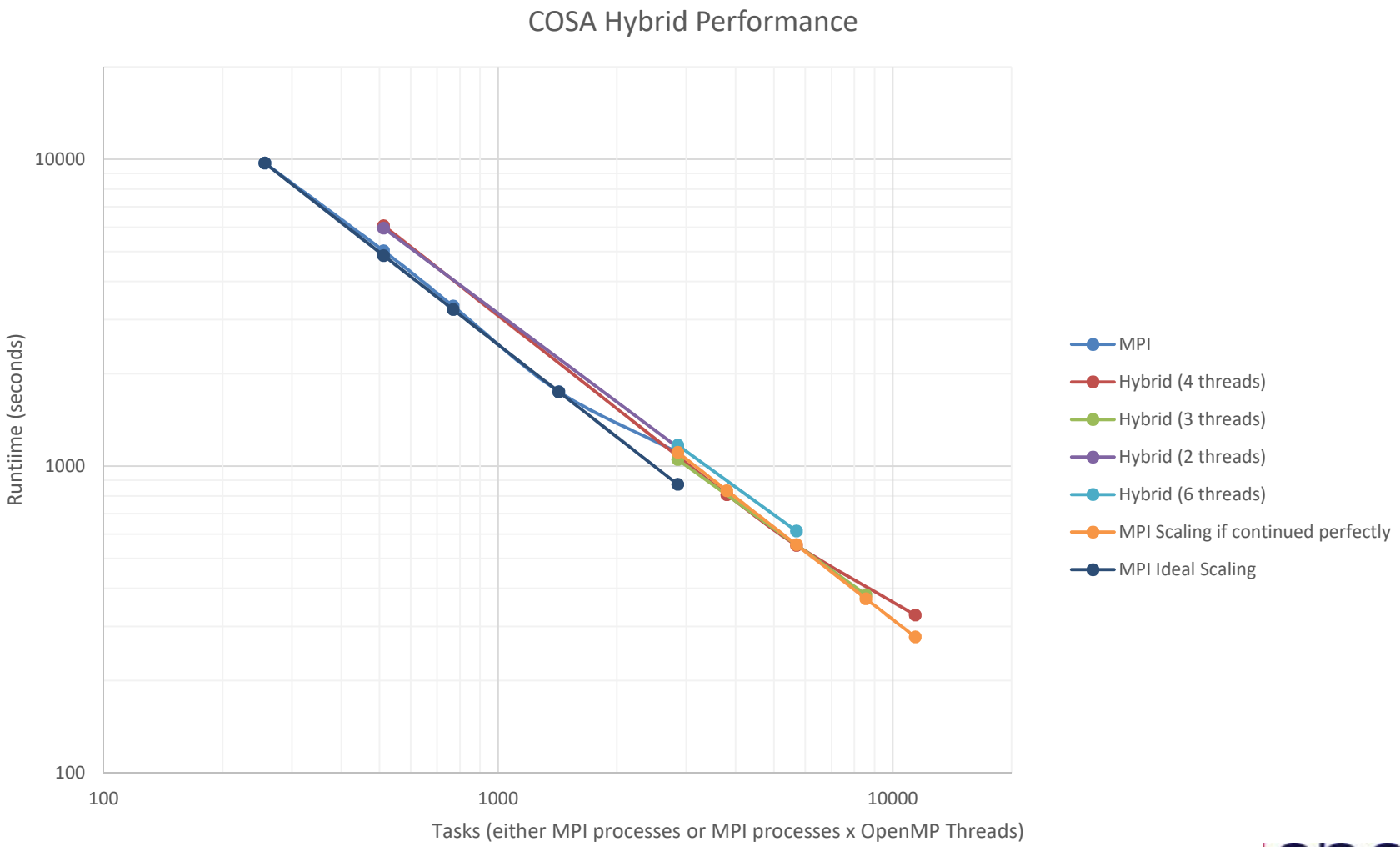
- Fluid dynamics code
  - Harmonic balance (frequency domain approach)
  - Unsteady navier-stokes solver
  - Optimise performance of turbo-machinery like problems
  - Multi-grid, multi-level, multi-block code
  - Parallelised with MPI and with MPI+OpenMP





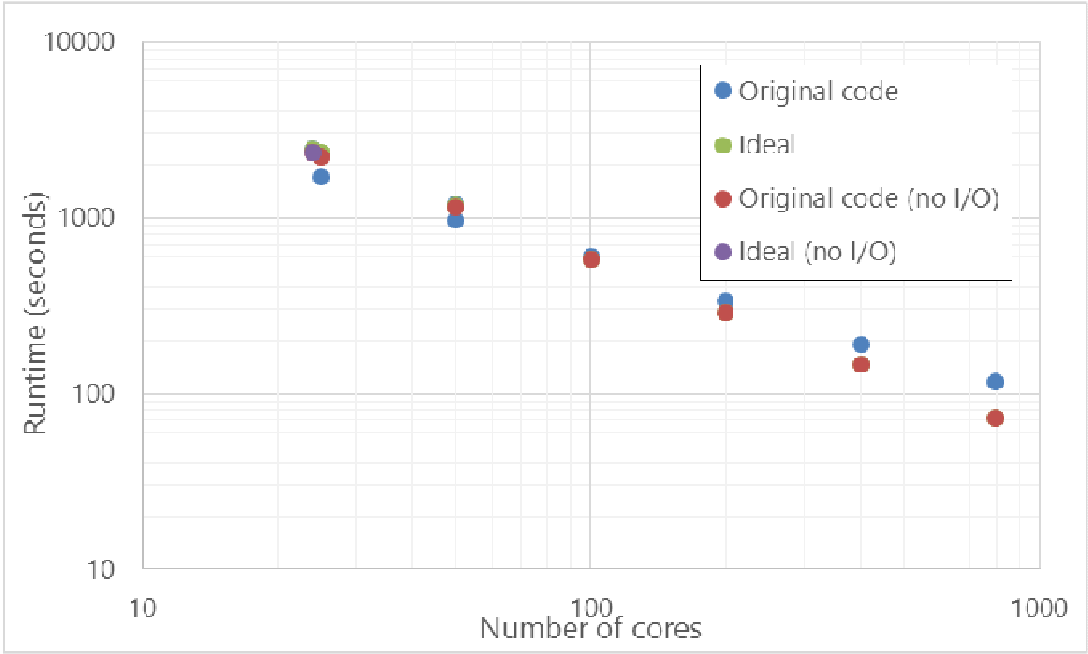
epcc

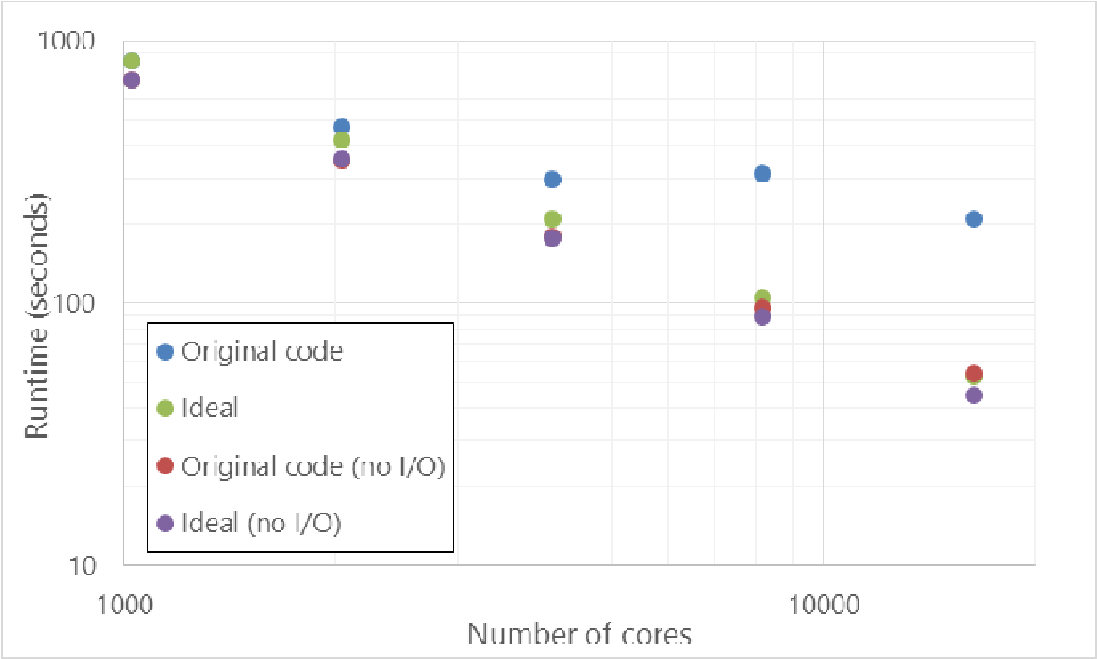
COSA Performance



epcc

Parallel Performance





epcc

Performance

100 processes

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

 100.0% | 61,775.2 |       -- |    -- |Total
|-----------------------------------------------------
|  78.1% | 48,217.5 |       -- |    -- |USER
||----------------------------------------------------
||  19.0% | 11,747.3 |  3,485.7 | 23.1% |vflux_
||   8.8% |  5,418.5 |    857.5 | 13.8% |roflux_
||   6.1% |  3,764.0 |  1,434.0 | 27.9% |muscl_
||   4.7% |  2,909.9 |  1,479.1 | 34.0% |q_face_
||   4.1% |  2,555.1 |    364.9 | 12.6% |tridi_
||   3.9% |  2,380.3 |  1,327.7 | 36.2% |bresid_
||   3.7% |  2,302.4 |  1,445.6 | 39.0% |muscl_bi_
||   3.2% |  1,972.9 |  1,146.1 | 37.1% |rtst_
||====================================================
|  19.0% | 11,742.3 |       -- |    -- |MPI
||----------------------------------------------------
||  11.1% |  6,831.8 | 33,844.2 | 84.0% |mpi_waitany
||   5.3% |  3,262.0 |    910.0 | 22.0% |MPI_FILE_WRITE
|====================================================
```

800 processes

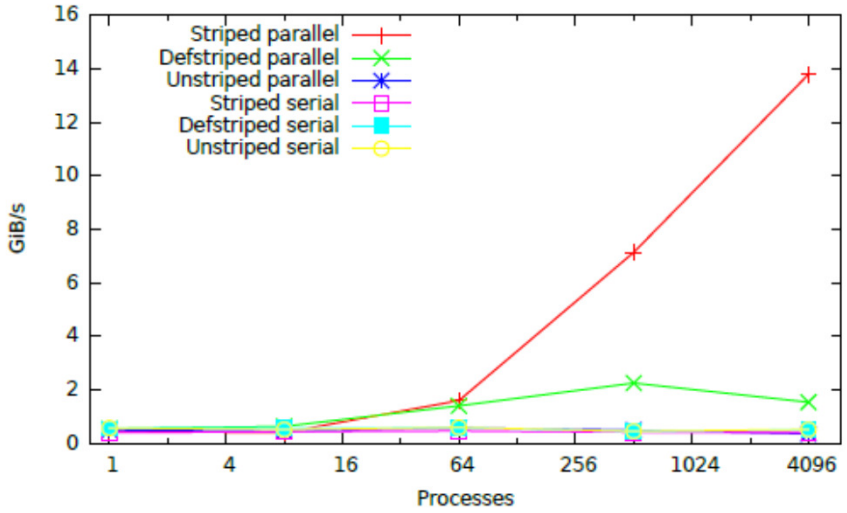
```
 Samp% |     Samp |    Imb. |  Imb. |Group
       |          |    Samp | Samp% | Function
       |          |         |       |  PE=HIDE

 100.0% | 10,461.0 |      -- |    -- |Total
|-----------------------------------------------------
|  56.3% |  5,889.8 |      -- |    -- |USER
||----------------------------------------------------
||  13.1% |  1,375.1 |   464.9 | 25.3% |vflux_
||   6.5% |    683.7 |   132.3 | 16.2% |roflux_
||   4.5% |    469.1 |   199.9 | 29.9% |muscl_
||   3.4% |    355.9 |   192.1 | 35.1% |q_face_
||   3.1% |    320.7 |    82.3 | 20.4% |tridi_
||   2.8% |    293.7 |   193.3 | 39.7% |bresid_
||====================================================
|  41.4% |  4,333.4 |      -- |    -- |MPI
||----------------------------------------------------
||  14.2% |  1,482.2 | 1,111.8 | 42.9% |MPI_FILE_WRITE
||  10.5% |  1,093.8 | 4,369.2 | 80.1% |mpi_waitany
||   8.4% |    877.7 |   885.3 | 50.3% |mpi_file_open
||   7.0% |    730.0 |   802.0 | 52.4% |MPI_BARRIER
|====================================================
```

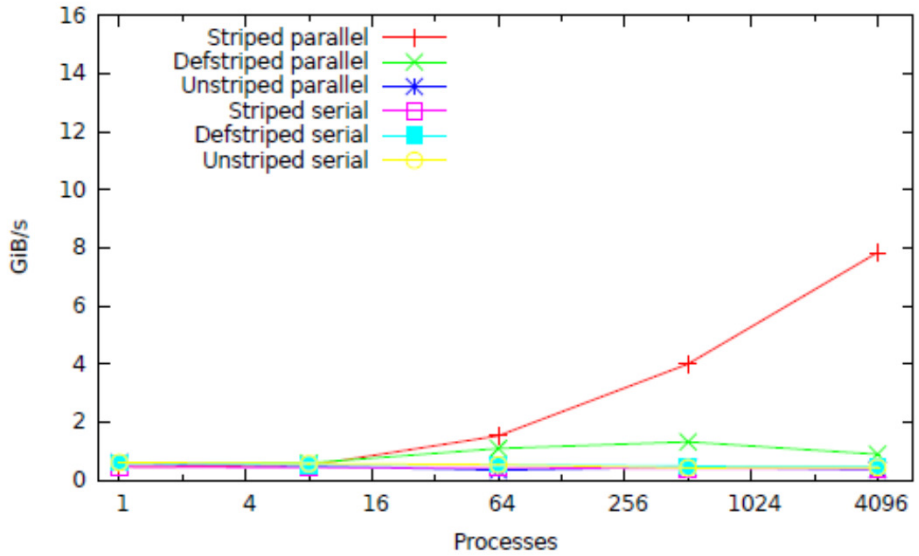
epcc

Data decomposition

- Data domain split up into blocks (multi-block, multi-level code)
  - Blocks may be of different sizes
  - Processes may have different numbers of blocks
- Suggests individual MPI-I/O file writing
  - Each process writes its blocks separately to a file, in parallel
  - Not collective

Collective vs individual performance





epcc

Data format

- ASCII to Binary data
  - MPI-I/O is binary format
  - Conversion compresses the files
    - 3x smaller for a lot of COSA data files
- Supporting serial file reading and writing more complex
  - Supporting no MPI library
  - Mimic Fortran binary format
  - Write size of line (in bytes) and beginning and end of each line
  - Involves 3x writes

Fortran file writing

- Standard I/O done with Fortran file writing
- Implicit loops over data structure, i.e. restart file:

```
write(fid) (((((q(i,j,k,ipde,n),i=-1,imax1),
 j= 1,jmax1),k=-1,kmax1),ipde=1,npde),n=0,2*nharms)
```

- Data structures have halo data
- Stored for restart files but not for data files
- Translated initially to something like this:

```
do n = 0,2*nharms
   do l=1,npde
      do k=-1,kmax1
         do j=-1,jmax1
            call setupfile(fid,disp)
            call mpi_file_write(fid, q(-1,j,k,l,n), imax+3,
     &           MPI_DOUBLE_PRECISION, MPI_STATUS_IGNORE, ierr)
            disp = disp + doublesize*(imax+3)
         end do
       end do
   end do
end do
```

Fortran file writing

- i.e. data file:

```
write(line1,'(''ZONE T="arturo",I='',i4,'', J='',i4,'', K='',i4,'',F=POINT, DT=(SINGLE SINGLE SINGLE DOUBLE DOUBLE
DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE)'')') imax1,jmax1,kmax1
write(fid(n),'(a)') line1
do k=0,kmax
  do j=0,jmax
    do i=0,imax
      write (fid(n),10) (var1(i,j,k,ipde,n),ipde=1,npde),(var2(i,j,k,ipde,n),ipde=1,npde)
     end do
   end do
 end do
10  format(3e16.8,7e22.14)
```

- Originally translated to:

```
write(line1,'(''ZONE T="arturo HB, mode '',i2,''"",I='',i4,''
    &,J='',i4,'',F=POINT, DT=(SINGLE SINGLE DOUBLE DOUBLE DOUBLE DOUBLE
    & DOUBLE DOUBLE)'')') n,imax1,jmax1
          call setupfile(fid(n),disp,MPI_INTEGER)
          call mpi_file_write(fid(n),integersize,1,
     &          MPI_INTEGER,MPI_STATUS_IGNORE,ierr)
          disp = disp + integersize
          call setupfile(fid(n),disp,MPI_INTEGER)
          call mpi_file_write(fid(n),typechar,1,
     &          MPI_INTEGER,MPI_STATUS_IGNORE,ierr)
          disp = disp + integersize
          call setupfile(fid(n),disp,MPI_INTEGER)
          call mpi_file_write(fid(n),integersize,1,
     &          MPI_INTEGER,MPI_STATUS_IGNORE,ierr)
          disp = disp + integersize
          call setupfile(fid(n),disp,MPI_INTEGER)
          call mpi_file_write(fid(n),integersize,1,
     &          MPI_INTEGER,MPI_STATUS_IGNORE,ierr)
          disp = disp + integersize
          call setupfile(fid(n),disp,MPI_INTEGER)
          call mpi_file_write(fid(n),111,1,
     &          MPI_INTEGER,MPI_STATUS_IGNORE,ierr)
          disp = disp + integersize
          call setupfile(fid(n),disp,MPI_INTEGER)
          call mpi_file_write(fid(n),integersize,1,
     &          MPI_INTEGER,MPI_STATUS_IGNORE,ierr)
          disp = disp + integersize
          call setupfile(fid(n),disp,MPI_INTEGER)
          call mpi_file_write(fid(n),111*charactersize,1,
     &          MPI_INTEGER,MPI_STATUS_IGNORE,ierr)
          disp = disp + integersize
          call setupfile(fid(n),disp,MPI_CHARACTER)
          call mpi_file_write(fid(n),line1,111,
     &          MPI_CHARACTER,MPI_STATUS_IGNORE,ierr)
          disp = disp + charactersize*111
          call setupfile(fid(n),disp,MPI_INTEGER)
          call mpi_file_write(fid(n),111*charactersize,1,
     &          MPI_INTEGER,MPI_STATUS_IGNORE,ierr)
          disp = disp + integersize
```

epcc

Data file continued

```
      do k=0,kmax
        do j=0,jmax
          do i=0,imax
            do ipde=1,npde
               tempdata(tempindex) = var1(i,j,k,ipde,n)
               tempindex = tempindex + 1
            end do
            do ipde=1,npde
               tempdata(tempindex) = var2(i,j,k,ipde,n)
               tempindex = tempindex + 1
            end do
          end do
        end do
      end do

      call setupfile(fid(n),disp,MPI_DOUBLE_PRECISION)
      call mpi_file_write(fid(n),tempdata(1),datasize,
     &     MPI_DOUBLE_PRECISION,MPI_STATUS_IGNORE,ierr)
      disp = disp + datasize*doublesize
```

Restart optimisation

- Convert this:

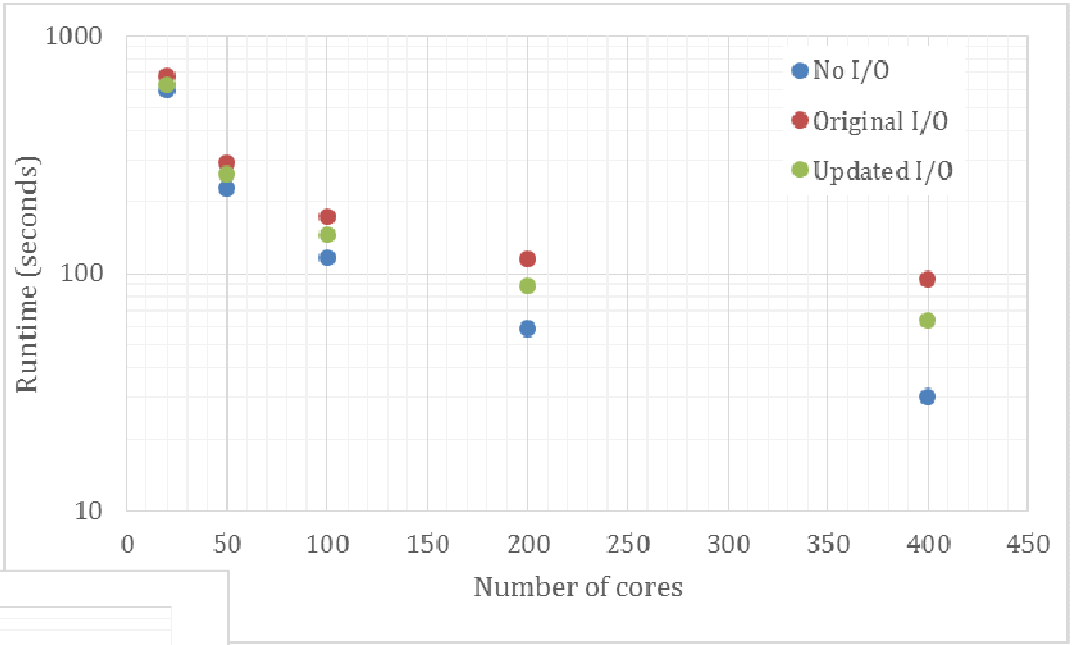
```
do n = 0,2*nharms
    do l=1,npde
       do k=-1,kmax1
          do j=-1,jmax1
             call setupfile(fid,disp)
             call mpi_file_write(fid, q(-1,j,k,l,n), imax+3,
     &         MPI_DOUBLE_PRECISION, MPI_STATUS_IGNORE, ierr)
             disp = disp + doublesize*(imax+3)
          end do
       end do
    end do
 end do
```

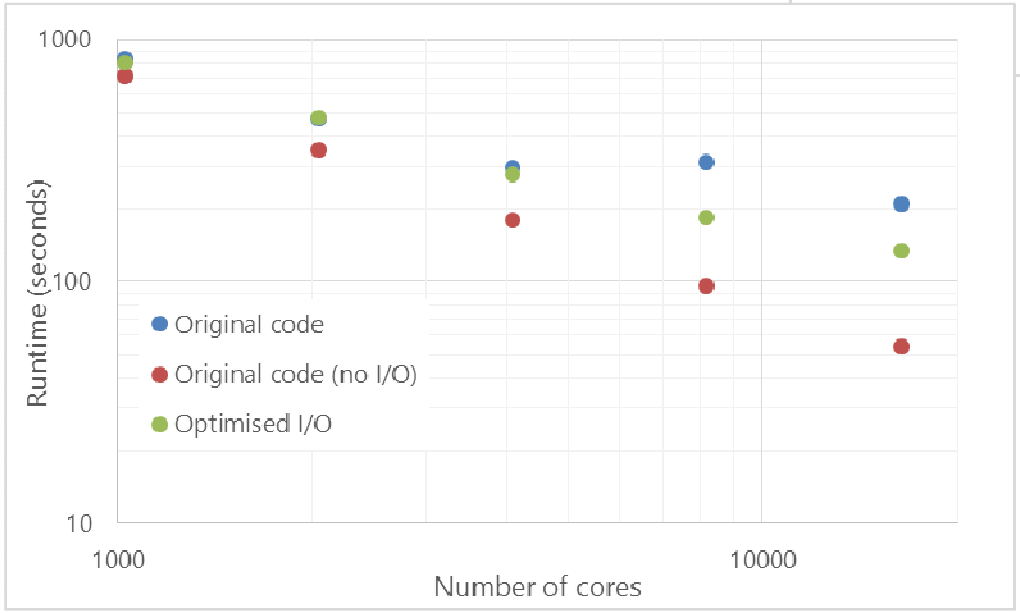
- To this:

```
      do n = 0,2*nharms
         call setupfile(fid,disp)
         call mpi_file_write(fid, linelength, 1,
     &        MPI_INTEGER, MPI_STATUS_IGNORE, ierr)
         disp = disp + integersize

         call setupfile(fid,disp)
         call mpi_file_write(fid, q(-1,-1,-1,1,n),
     &        linelength/doublesize,
     &        MPI_DOUBLE_PRECISION, MPI_STATUS_IGNORE, ierr)
         disp = disp + linelength

         call setupfile(fid,disp)
         call mpi_file_write(fid, linelength, 1,
     &        MPI_INTEGER, MPI_STATUS_IGNORE, ierr)
         disp = disp + integersize
      end do
```

epcc

Update I/O performance



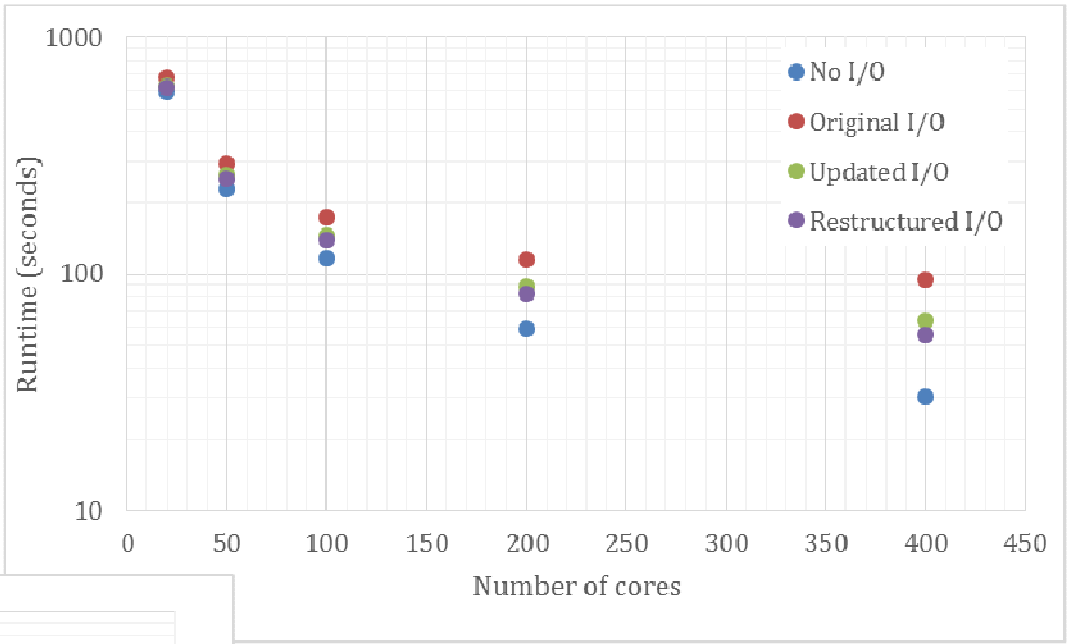


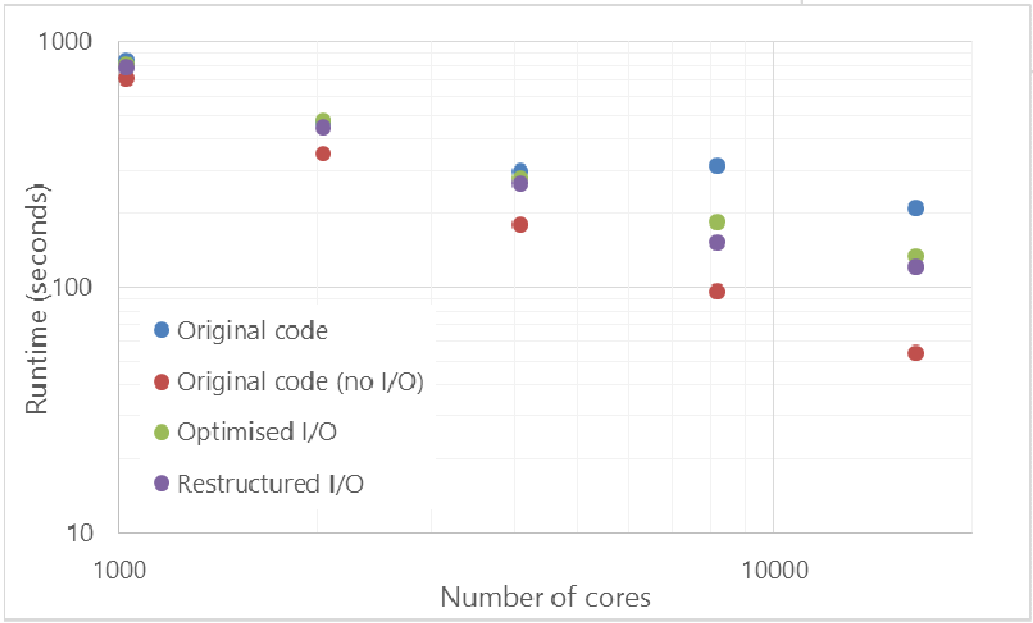
epcc

Further I/O optimisations

- Restructure how the output values are calculated
  - Storing temporary values used in the calculations rather than re-reading them
- Parallelise the input of the grid
  - Original code does serial reads from every process for relevant data
  - File locking can impact performance at large process counts

Optimised I/O performance





epcc

I/O data formats

- MPI-I/O gives best achievable performance
- Not a “user friendly” format
- HDF5, NetCDF, etc… provide more functionality data formats
  - Metadata, data structure, I/O tools,…
- For CFD there are specific formats often used
  - CGNS
  - TecPlot
  - Reading/Writing these formats enables integration with other tools

I/O data formats

- CGNS: CFD General Notation System
  - http://cgns.github.io/
- Tree structured data
  - Set of different types of data that can be written
- CGNS serial and parallel functionality
  - Metadata and data can be done serially
  - Data can be done serially or in parallel
  - cg_ serial function
  - cgp_ parallel function

Example: Metadata write

```
if(mrank .eq. 0) then

        call cg_open_f('rest.cgns',CG_MODE_WRITE,fid,ierr)
        if(ierr .ne. CG_OK) then
           write(*,*) 'cg_open_f restart error'
           call cg_error_print_f()
        end if

        call cg_base_write_f(fid,'gridbase',3,3,basenum,ierr)
        if(ierr .ne. CG_OK) then
           write(*,*) 'cg_base_write_f error'
           call cg_error_print_f()
        end if

        do i = 1,blocksize
           blocknum = i
           write(zonename, "(A5,I6)") "block",blocknum
           call cg_zone_write_f(fid,basenum,zonename,sizes,
    &           Structured,zonenum,ierr)
           if(ierr .ne. CG_OK) then
              write(*,*) 'cg_zone_write_f error'
              call cg_error_print_f()
           end if
           call cg_goto_f(fid,basenum,ierr,'Zone_t',zonenum,'end')
           if(ierr .ne. CG_OK) then
              write(*,*) 'cg_goto_f error'
              call cg_error_print_f()
           end if
           write(linkpath,'(a,i6,a)') 'gridbase/block',zonenum,
    &            '/GridCoordinates'
           call cg_link_write_f('GridCoordinates','mesh.cgns',
    &           linkpath,ierr)
           if(ierr .ne. CG_OK) then
              write(*,*) 'cg_link_write_f error'
              call cg_error_print_f()
           end if
           call cg_user_data_write_f('User Data',ierr)
        end do

        call cg_close_f(fid,ierr)
        if(ierr .ne. CG_OK) then
           write(*,*) 'cg_close_f error'
           call cg_error_print_f()
        end if

     end if
```

epcc

Example: Data write

```
 basenum = 1

 call cgp_open_f('rest.cgns',CG_MODE_MODIFY,fid,ierr)

 solnum = 1

  do i = 1,blocksize
     zonenum = i
     write(zonename, "(A5,I6)") "block",zonenum
     call cg_zone_read_f(fid,basenum,zonenum,
&         tempzonename,tempsizes,ierr)
     if(trim(tempzonename) .ne. zonename) then
        write(*,*) 'error block name: ',zonename,tempzonename
     end if
     call cg_goto_f(fid,basenum,ierr,'Zone_t',zonenum,
&         'UserDefinedData_t',solnum,'end')
     call cgp_array_write_f(fieldname,RealDouble,5,qsizes,
&         arraynum,ierr)
     if(ierr .ne. CG_OK) then
        write(*,*) 'cg_array_write_f error'
        call cg_error_print_f()
     end if
  end do
```

epcc

CGNS Performance

- 512 test read and write of restart file (~40 GB):
  - MPI-I/O read file 3 seconds
  - CGNS write file 533 seconds
  - CGNS read file: 233 seconds

- Not designed for large number of blocks
  - Designed for single block applications
  - Metadata overhead destroying performance

Tecplot

- Grid based data format
  - Support for common CFD grid formats
  - Support tecplot tools
  - http://www.tecplot.com/
- Supports 3 different file formats
  - ASCII (legacy format, requires conversion by tecplot tools to be used by their products)
  - Binary (plt)
  - Binary partitioned (szplt) (designed for large scale parallel)

Tecplot

```
   if(tecini142(trim(titlename),trim(varlist),
&        filename,
&        trim(pwd)//char(0),
&        fileformat,filetype,debug,isdouble) .ne. 0) then
       write(*,*) 'error initialising tecini'
    end if
    do j=1,nblocks

       imax1 = blockindexes(1,j)
       jmax1 = blockindexes(2,j)
       kmax1 = blockindexes(3,j)

       datasize = imax1*jmax1*kmax1

       write (blocknumname, "(I5)") j
          if(teczne142('block'//trim(blocknumname)//char(0),
&               zonetype, imax1, jmax1, kmax1, imaxmax, jmaxmax, kmaxmax, simtime, strandid, parentzone, isblock, nfconns, fnmode,
&               tnfnodes,ncbfaces,tnbconns,Null, Null, Null, shrconn) .ne. 0) then
             write(*,*) 'error setting up zone'
          end if

          if(tecdat142(datasize*10,tempdata,isdouble) .ne. 0) then
             write(*,*) 'error writing block data'
          end if

    end do

    if(tecend142() .ne. 0) then
       write(*,*) 'error calling tecend'
    end if
```

epcc

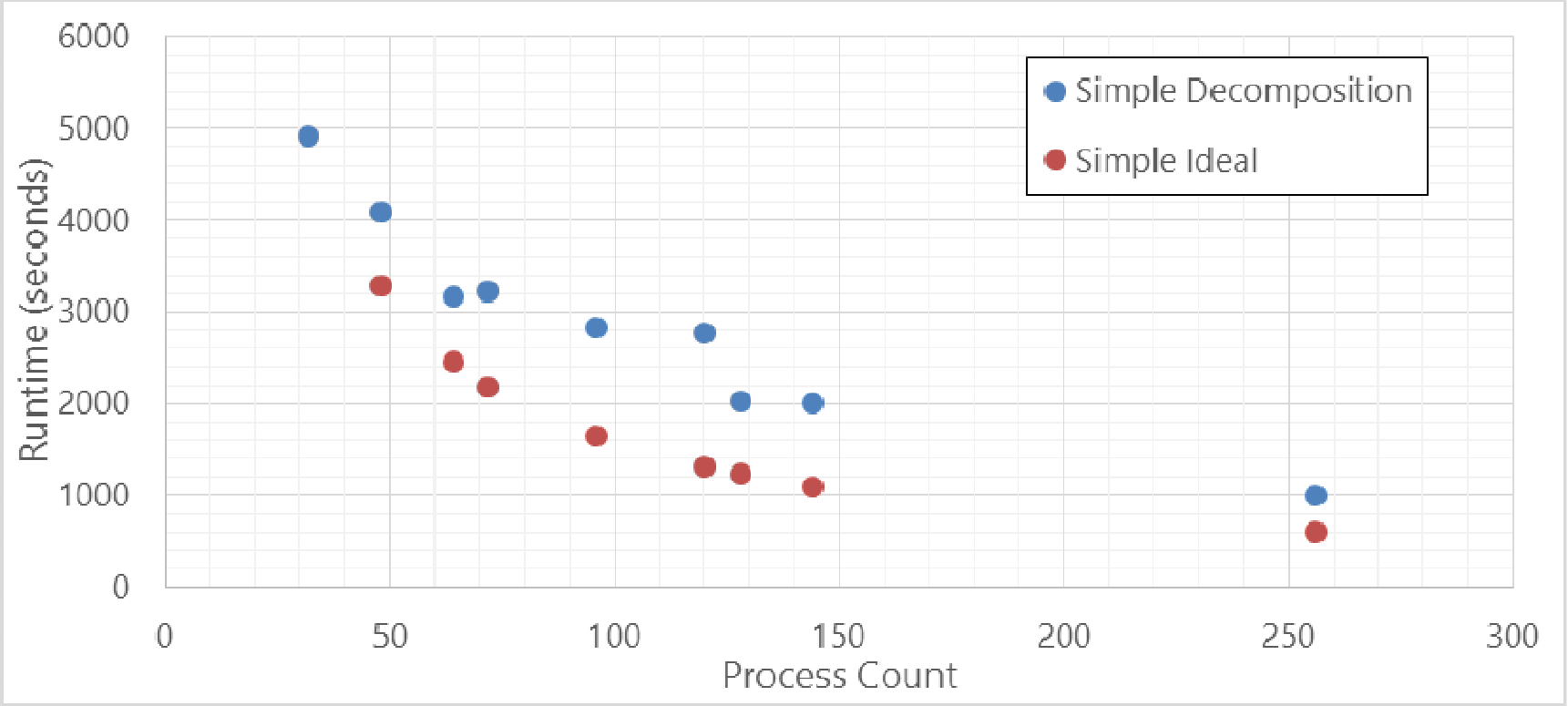
Tecplot performance

- Only implemented serial so far
- plt is 10x faster than szplt
  - Working with tecplot on this
- Next step to parallelise the tecplot writing

I/O library issues and future work

- Libraries designed to deal with single large blocks
- Don’t work well with data decomposition done in the program
  - Working with CGNS and tecplot to fix this
- Need to try Tecplot parallel functionality using szplt
- Need to try creating MPI datatypes for each block and then using collective I/O

Load balancing

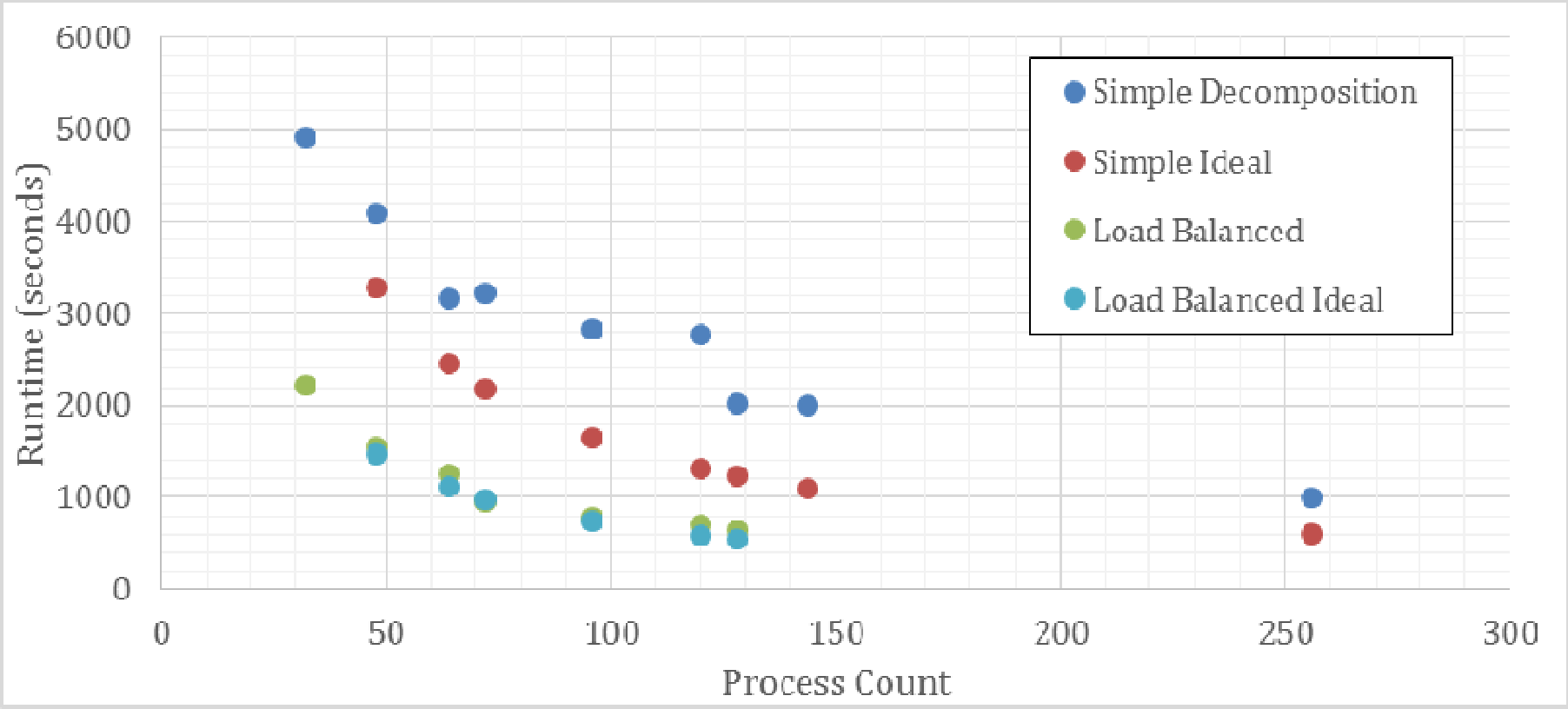


- Load balance is done outside the code
  - Input file with pre-defined blocks is provided
  - Blocks distributed across processes to balance the number of blocks per process
  - Makes manual process of grid decomposition challenging and time consuming

Load balancing

- Use METIS to evaluate size of each block and decompose work across processes
  - Allocate different numbers of blocks to processes to balance overall work
  - Requires number of blocks >> number of MPI processes
- Can also take communication costs into account
  - Balance work and reduce boundary area
- Enables automated block generation
  - Take block decomposition directly from grid generator

Performance



- 256 block grid, high load imbalance
  - 90% difference in grid points between largest and smallest blocks

epcc

Performance

- Also enables full node utilisation
  - Original decomposition favours number of processes to be exact divisor of number of blocks
  - 24 cores per node

ARCHER Nodes Used	MPI Processes Used	Original decomposition	Load balance decomposition
2	32	4924	2220
2	48	4081	1544
3	64	3157	1247
3	72	3231	968
4	96	2833	792
5	120	2764	714
6	128	2016	646

Summary

- Optimising both performance and usability
  - Usability of the code, making it quicker to prepare simulations, as important as performance
  - Looking to optimise the whole workflow as well as parallel performance
    - Input preparation, Simulation runs, Output analysis
    - This work aimed to help all three
      - Input preparation: Load balancing to reduce effort require/allow more uneven block decomposition
      - Simulation runs: I/O optimisation to reduce I/O cost and improve scaling, load balancing allows all cores in nodes to be used efficiently
      - Output analysis: Tools to convert output data into formats that tools can read and display
- Overall load balance shown to give 3-4x performance improvement for high imbalance case
- I/O cost reduce by 70% for large cases and core counts