Tutorial Instruction Sheet:
Analysing Parallel I/O Using Allinea MAP

Keeran Brabazon, ARM
ISC 2017, Frankfurt

Summary

In this tutorial you will learn the following:

- How to open an Allinea MAP file and find metrics salient to file I/O
- Interpret the metrics presented to give insight into the operation of I/O routines using different I/O libraries
- How to extract information relevant to a selection of a performance profile in Allinea MAP

The exercises will give you an insight into how to get started with the analysis of I/O performance from a program run. There is extra work that can be done, and you are welcome to continue to perform an investigation into I/O with the use of the Allinea tools. If your institution does not have an installation of the Allinea tools you may request a trial from https://www.allinea.com/get-your-free-allinea-forge-and-allinea-performance-reports-trial

Getting started

Installing the Tools

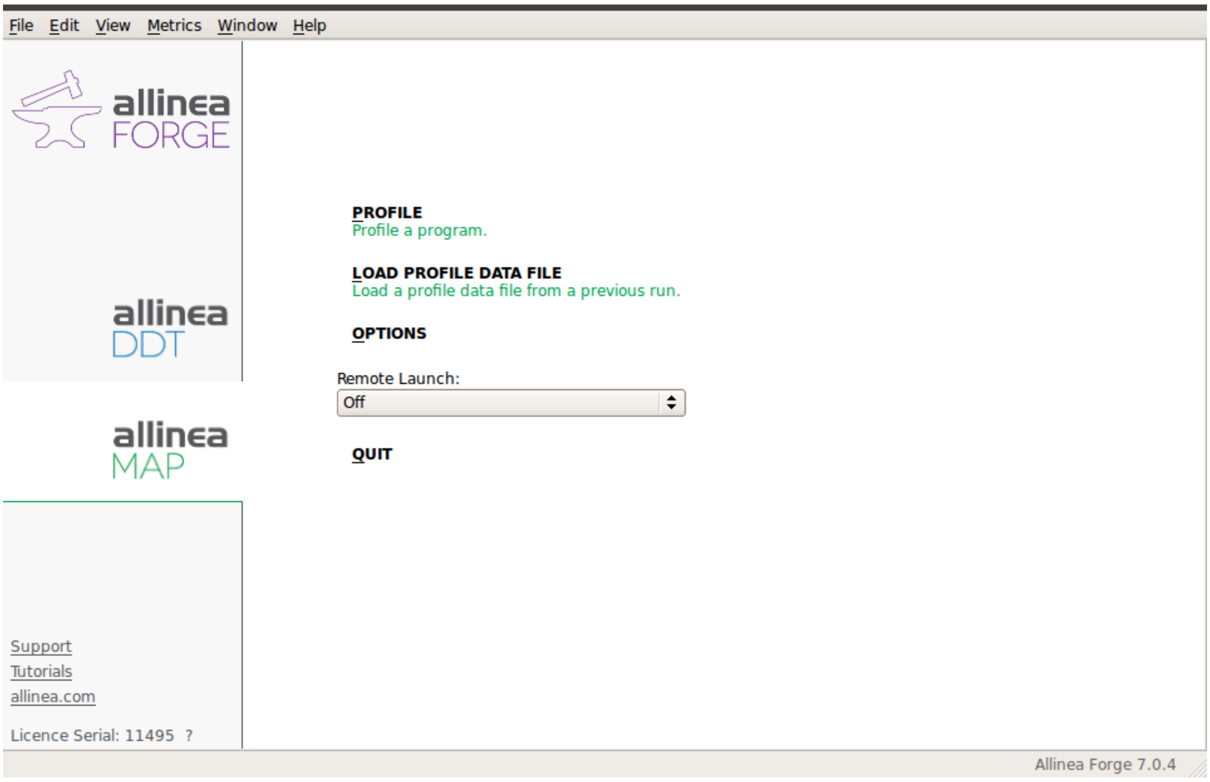
- The tools need to be installed locally (i.e. not on ARCHER)
- Navigate to https://www.allinea.com/products/forge/download
- Choose the download for the appropriate platform you are using (Windows and Mac have front-end clients)
- Follow the installation instructions at https://www.allinea.com/user-guide/forge/Installation.html#x5-70002

Getting necessary resources

- Download the source files of the `benchio` program introduced in an earlier session onto your local machine
- Ensure that you have a folder called `TutorialData` with a set of MAP profiles (files ending in .map) as well as a set of output files from the benchmark runs. This file should have been provided with the folder.

Opening a profile

- Launch the Forge GUI, and navigate to the MAP pane



PROFILE
Profile a program.

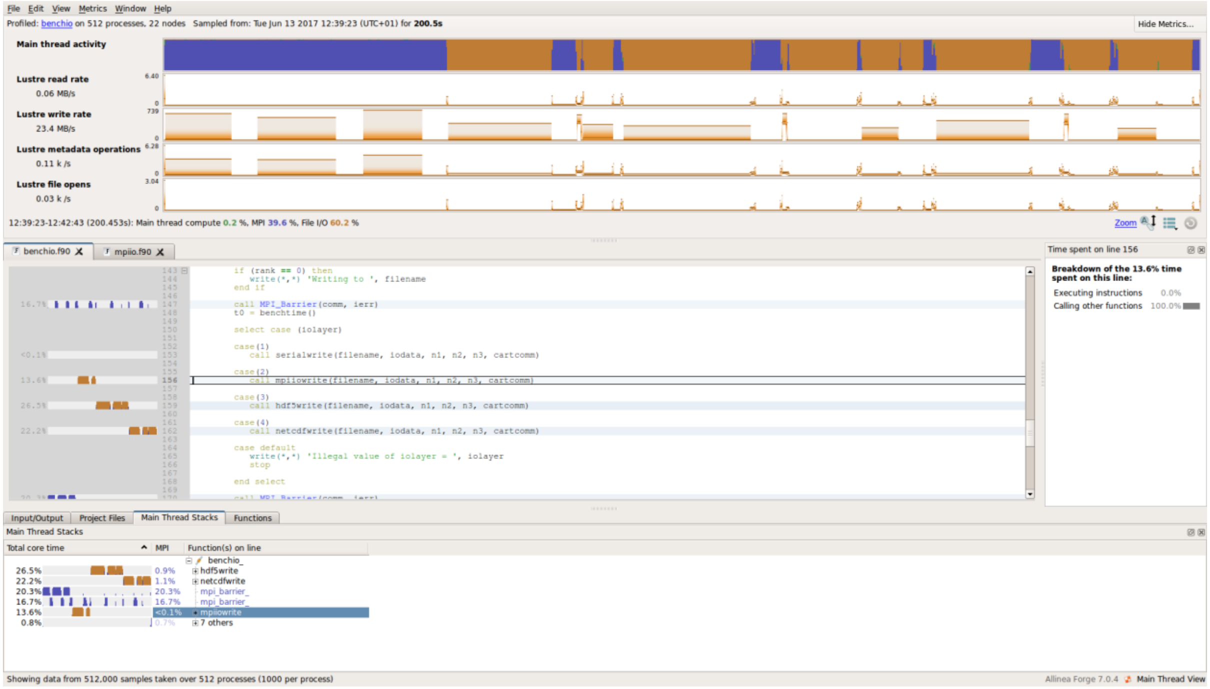
LOAD PROFILE DATA FILE
Load a profile data file from a previous run.

Remote Launch: Off

Licence Serial: 11495 ?

Allinea Forge 7.0.4

- Select the 'Load Profile Data File' option
- Select the file called benchio_512p_128_1.map from the folder in which this document was contained and the profile will be loaded
- Show the Lustre metric information by selecting the 'Metrics -> Preset: Lustre' menu option



Profiled: benchio on 512 processes, 22 nodes Sampled from: Tue Jun 13 2017 12:39:23 (UTC+01) for **200.5s**

Lustre read rate 0.06 MB/s

Lustre write rate 23.4 MB/s

Lustre metadata operations 0.11 k /s

Lustre file opens 0.03 k /s

12:39:23-12:42:43 (200.453s): Main thread compute 0.2 %, MPI 39.6 %, File I/O 60.2 %

```
        if (rank == 0) then
           write(*,*) 'Writing to ', filename
        end if

        call MPI_Barrier(comm, ierr)
        t0 = benchtime()

        select case (iolayer)

        case(1)
           call serialwrite(filename, iodata, n1, n2, n3, cartcomm)

        case(2)
           call mpiiowrite(filename, iodata, n1, n2, n3, cartcomm)

        case(3)
           call hdf5write(filename, iodata, n1, n2, n3, cartcomm)

        case(4)
           call netcdfwrite(filename, iodata, n1, n2, n3, cartcomm)

        case default
           write(*,*) 'Illegal value of iolayer = ', iolayer
           stop

        end select

        call MPI_Barrier(comm, ierr)
```

Time spent on line 156

Breakdown of the 13.6% time spent on this line:

Executing instructions	0.0%
Calling other functions	100.0%

Main Thread Stacks

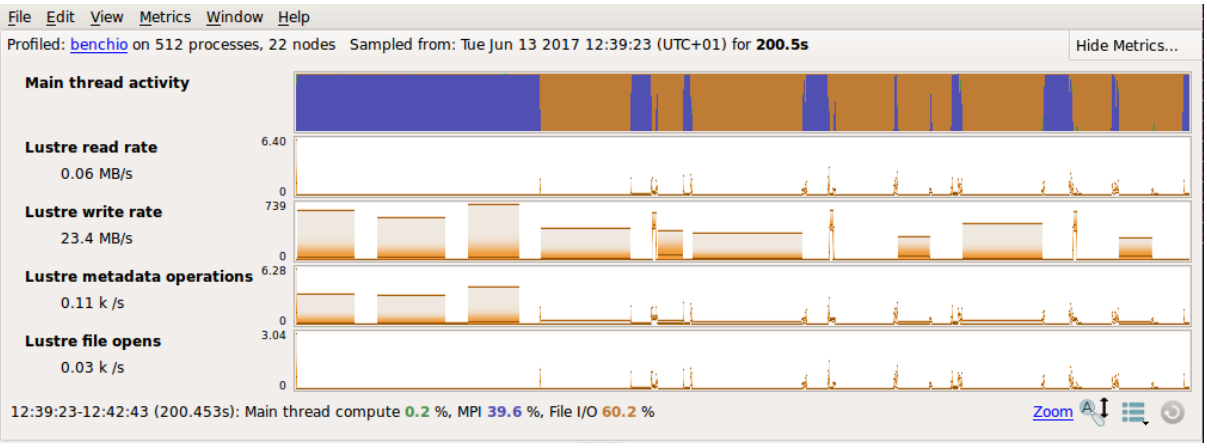
Total core time	MPI	Function(s) on line
		benchio_
26.5%	0.9%	hdf5write
22.2%	1.1%	netcdfwrite
20.3%	20.3%	mpi_barrier_
16.7%	16.7%	mpi_barrier_
13.6%	<0.1%	mpiiowrite
0.8%	0.7%	7 others

Showing data from 512,000 samples taken over 512 processes (1000 per process)

Allinea Forge 7.0.4 Main Thread View

Layout of a MAP profile

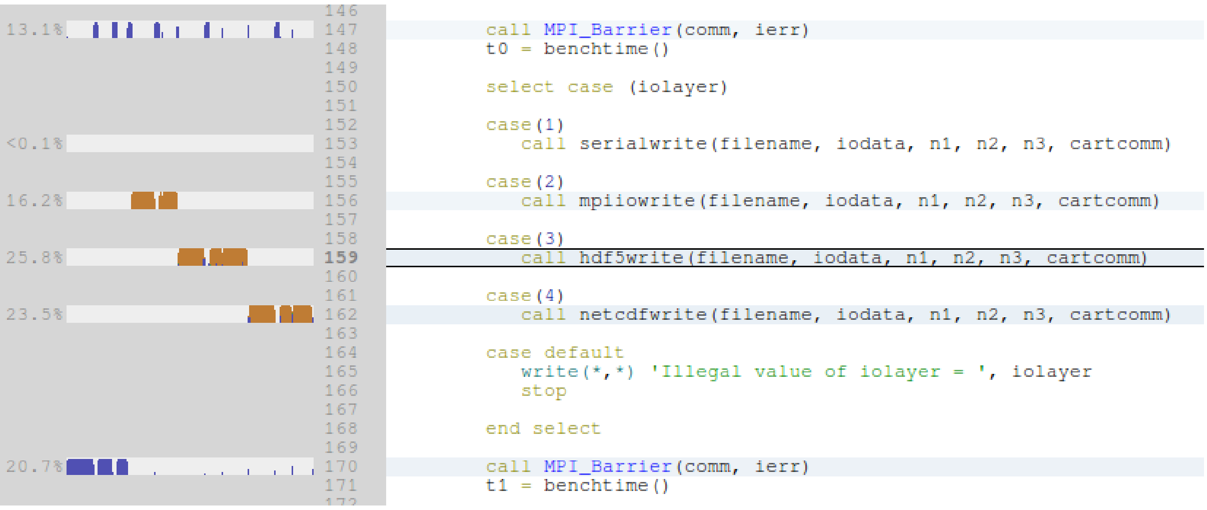
- The top-most section of a profile contains graphs over time of different metrics
- The 'Main Thread Activity' timeline is a colour-coded histogram of activity. Blue indicates MPI activity, orange is I/O. Other colours indicate activity not measured in these programs (e.g. green for compute activity such as floating point (vector) operations and memory accesses).



Profiled: benchio on 512 processes, 22 nodes Sampled from: Tue Jun 13 2017 12:39:23 (UTC+01) for **200.5s**

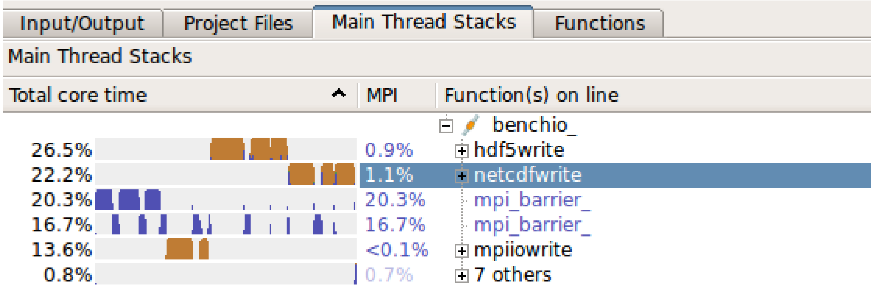
12:39:23-12:42:43 (200.453s): Main thread compute 0.2 %, MPI 39.6 %, File I/O 60.2 %

- Central section shows the source code.
- Each line on which activity was recorded has a graph similar to the 'Main thread activity' timeline which indicates when activity was recorded on that line



```
call MPI_Barrier(comm, ierr)
t0 = benchtime()

select case (iolayer)

case(1)
   call serialwrite(filename, iodata, n1, n2, n3, cartcomm)

case(2)
   call mpiiowrite(filename, iodata, n1, n2, n3, cartcomm)

case(3)
   call hdf5write(filename, iodata, n1, n2, n3, cartcomm)

case(4)
   call netcdfwrite(filename, iodata, n1, n2, n3, cartcomm)

case default
   write(*,*) 'Illegal value of iolayer = ', iolayer
   stop

end select

call MPI_Barrier(comm, ierr)
t1 = benchtime()
```

- Lower-most section has several tabs to show different data
- Default tab shows the call stacks in which activity was recorded.
- Activity timelines in the left-hand margin indicate when activity was recorded in the call stacks



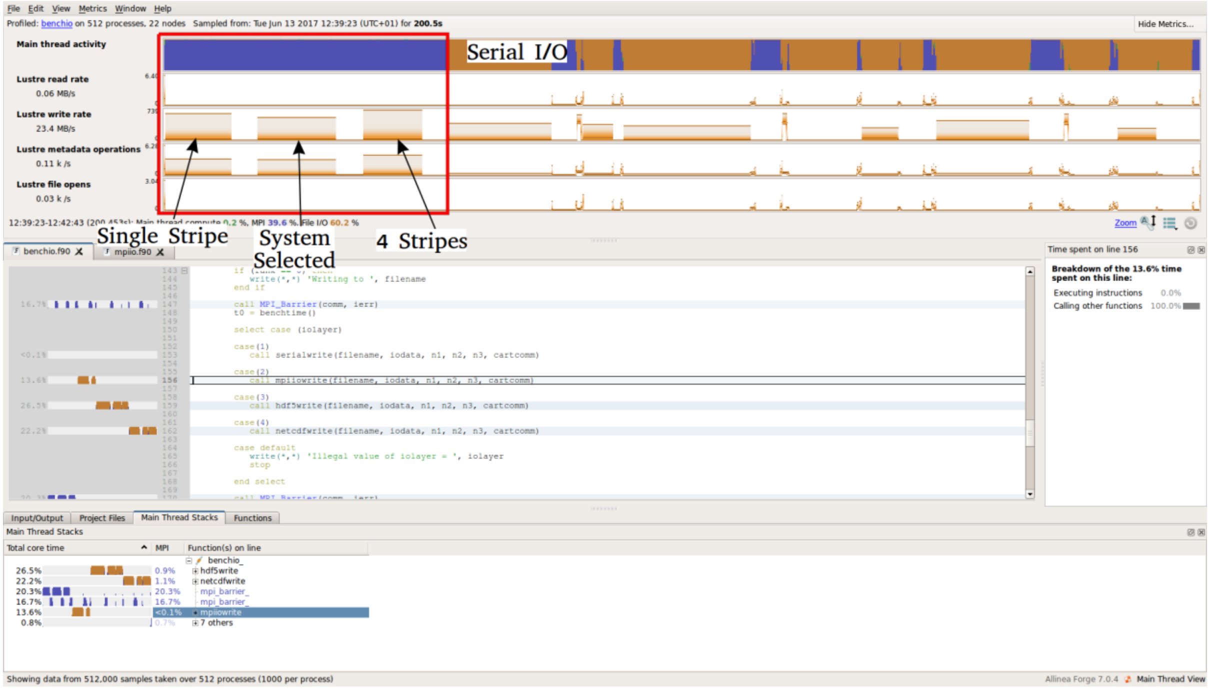
Main Thread Stacks

Total core time	MPI	Function(s) on line
		benchio_
26.5%	0.9%	hdf5write
22.2%	1.1%	netcdfwrite
20.3%	20.3%	mpi_barrier_
16.7%	16.7%	mpi_barrier_
13.6%	<0.1%	mpiiowrite
0.8%	0.7%	7 others

Matching a profile to a program

- Ensure the profile benchio_512p_128_1.map is open
- Recall the program profiled performs I/O using the paradigms
  - Serial I/O
  - MPI I/O
  - HDF5 I/O
  - NetCDF I/O

  and that it writes data to files which have
  - 1 stripe
  - System selected striping
  - Default system striping (this is 4 on ARCHER)
- From the write rate identify the writes using Serial I/O



Profiled: benchio on 512 processes, 22 nodes Sampled from: Tue Jun 13 2017 12:39:23 (UTC+01) for 200.5s

Main thread activity

Lustre read rate 0.06 MB/s

Lustre write rate 23.4 MB/s

Lustre metadata operations 0.11 k /s

Lustre file opens 0.03 k /s

12:39:23-12:42:43 (200.453s): Main thread compute 0.2 %, MPI 39.6 %, File I/O 60.2 %

```
      if (rank == 0) then
         write(*,*) 'Writing to ', filename
      end if

      call MPI_Barrier(comm, ierr)
      t0 = benchtime()

      select case (iolayer)

      case(1)
         call serialwrite(filename, iodata, n1, n2, n3, cartcomm)

      case(2)
         call mpiiowrite(filename, iodata, n1, n2, n3, cartcomm)

      case(3)
         call hdf5write(filename, iodata, n1, n2, n3, cartcomm)

      case(4)
         call netcdfwrite(filename, iodata, n1, n2, n3, cartcomm)

      case default
         write(*,*) 'Illegal value of iolayer = ', iolayer
         stop

      end select

      call MPI_Barrier(comm, ierr)
```

Time spent on line 156

Breakdown of the 13.6% time spent on this line:

Executing instructions	0.0%
Calling other functions	100.0%

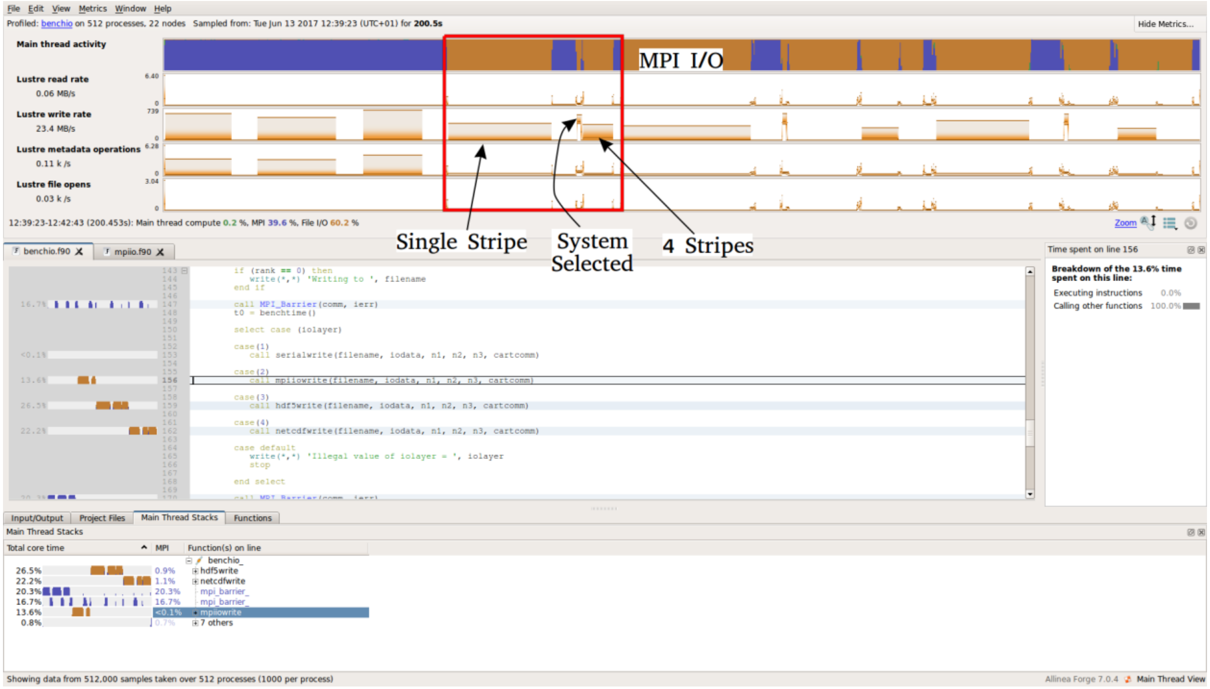
Main Thread Stacks

Total core time	MPI	Function(s) on line
		benchio_
26.5%	0.9%	hdf5write
22.2%	1.1%	netcdfwrite
20.3%	20.3%	mpi_barrier_
16.7%	16.7%	mpi_barrier_
13.6%	<0.1%	mpiiowrite
0.8%	0.7%	7 others

Showing data from 512,000 samples taken over 512 processes (1000 per process)

Allinea Forge 7.0.4 Main Thread View

- The writes using MPI I/O



Profiled: benchio on 512 processes, 22 nodes Sampled from: Tue Jun 13 2017 12:39:23 (UTC+01) for **200.5s**

Main thread activity

Lustre read rate 0.06 MB/s

Lustre write rate 23.4 MB/s

Lustre metadata operations 0.11 k /s

Lustre file opens 0.03 k /s

12:39:23-12:42:43 (200.453s): Main thread compute 0.2 %, MPI 39.6 %, File I/O 60.2 %

```
16.7%       call MPI_Barrier(comm, ierr)
```

```
      if (rank == 0) then
         write(*,*) 'Writing to ', filename
      end if

      call MPI_Barrier(comm, ierr)
      t0 = benchtime()

      select case (iolayer)

      case(1)
         call serialwrite(filename, iodata, n1, n2, n3, cartcomm)

      case(2)
         call mpiiowrite(filename, iodata, n1, n2, n3, cartcomm)

      case(3)
         call hdf5write(filename, iodata, n1, n2, n3, cartcomm)

      case(4)
         call netcdfwrite(filename, iodata, n1, n2, n3, cartcomm)

      case default
         write(*,*) 'Illegal value of iolayer = ', iolayer
         stop

      end select

      call MPI_Barrier(comm, ierr)
```

Per-line time: line 147 16.7%, line 153 <0.1%, line 156 13.6%, line 159 26.5%, line 162 22.2%, line 170 20.3%

Time spent on line 156

Breakdown of the 13.6% time spent on this line:

Executing instructions 0.0%
Calling other functions 100.0%

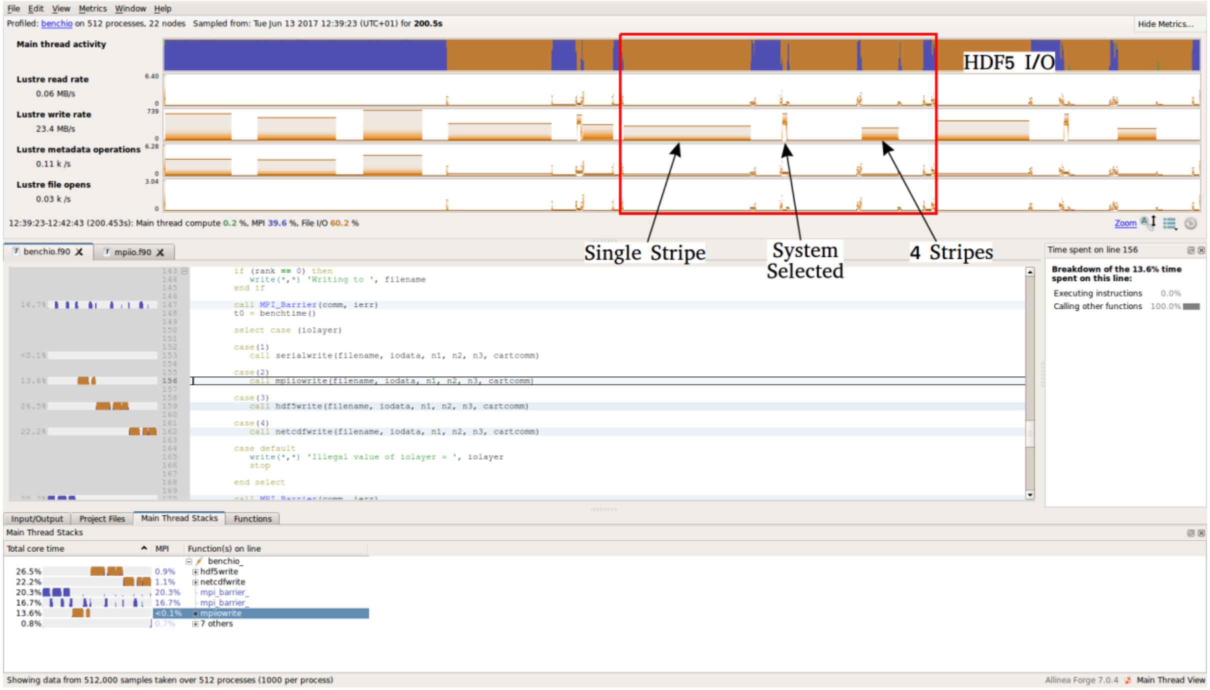
Main Thread Stacks

Total core time	MPI	Function(s) on line
		⊟ benchio_
26.5%	0.9%	⊞ hdf5write
22.2%	1.1%	⊞ netcdfwrite
20.3%	20.3%	mpi_barrier_
16.7%	16.7%	mpi_barrier_
13.6%	<0.1%	⊞ mpiiowrite
0.8%	0.7%	⊞ 7 others

Showing data from 512,000 samples taken over 512 processes (1000 per process)

Allinea Forge 7.0.4 Main Thread View

- And the writes using HDF I/O and NetCDF I/O



Profiled: benchio on 512 processes, 22 nodes Sampled from: Tue Jun 13 2017 12:39:23 (UTC+01) for **200.5s**

Main thread activity

Lustre read rate 0.06 MB/s

Lustre write rate 23.4 MB/s

Lustre metadata operations 0.11 k /s

Lustre file opens 0.03 k /s

12:39:23-12:42:43 (200.453s): Main thread compute 0.2 %, MPI 39.6 %, File I/O 60.2 %

```
      if (rank == 0) then
         write(*,*) 'Writing to ', filename
      end if

      call MPI_Barrier(comm, ierr)
      t0 = benchtime()

      select case (iolayer)

      case(1)
         call serialwrite(filename, iodata, n1, n2, n3, cartcomm)

      case(2)
         call mpiiowrite(filename, iodata, n1, n2, n3, cartcomm)

      case(3)
         call hdf5write(filename, iodata, n1, n2, n3, cartcomm)

      case(4)
         call netcdfwrite(filename, iodata, n1, n2, n3, cartcomm)

      case default
         write(*,*) 'Illegal value of iolayer = ', iolayer
         stop

      end select

      call MPI_Barrier(comm, ierr)
```

Time spent on line 156

Breakdown of the 13.6% time spent on this line:

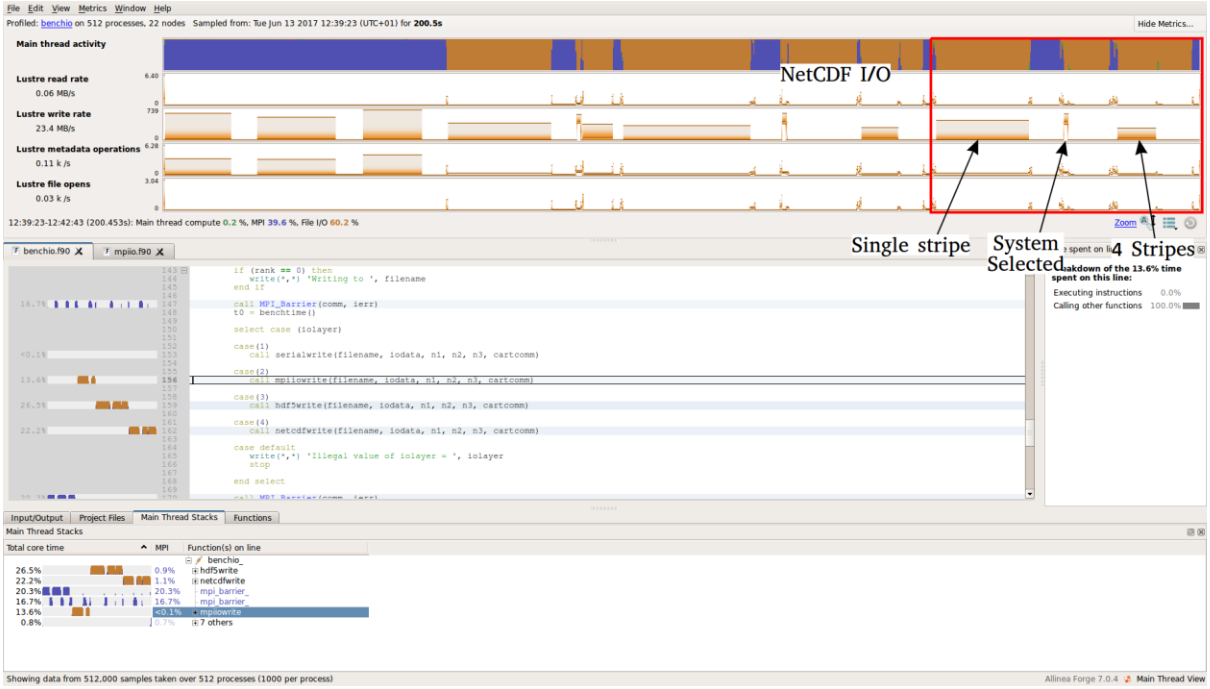
Executing instructions 0.0%
Calling other functions 100.0%

Main Thread Stacks

Total core time	MPI	Function(s) on line
		⊟ benchio_
26.5%	0.9%	⊞ hdf5write
22.2%	1.1%	⊞ netcdfwrite
20.3%	20.3%	mpi_barrier_
16.7%	16.7%	mpi_barrier_
13.6%	<0.1%	⊞ mpiiowrite
0.8%	0.7%	⊞ 7 others

Showing data from 512,000 samples taken over 512 processes (1000 per process)

Allinea Forge 7.0.4 Main Thread View



Profiled: benchio on 512 processes, 22 nodes Sampled from: Tue Jun 13 2017 12:39:23 (UTC+01) for **200.5s**

12:39:23-12:42:43 (200.453s): Main thread compute 0.2 %, MPI 39.6 %, File I/O 60.2 %

```
      if (rank == 0) then
         write(*,*) 'Writing to ', filename
      end if

      call MPI_Barrier(comm, ierr)
      t0 = benchtime()

      select case (iolayer)

      case(1)
         call serialwrite(filename, iodata, n1, n2, n3, cartcomm)

      case(2)
         call mpiiowrite(filename, iodata, n1, n2, n3, cartcomm)

      case(3)
         call hdf5write(filename, iodata, n1, n2, n3, cartcomm)

      case(4)
         call netcdfwrite(filename, iodata, n1, n2, n3, cartcomm)

      case default
         write(*,*) 'Illegal value of iolayer = ', iolayer
         stop

      end select

      call MPI_Barrier(comm, ierr)
```

…e spent on li…

…eakdown of the 13.6% time spent on this line:

Executing instructions	0.0%
Calling other functions	100.0%

Main Thread Stacks

Total core time	MPI	Function(s) on line
		⊟ benchio_
26.5%	0.9%	⊞ hdf5write
22.2%	1.1%	⊞ netcdfwrite
20.3%	20.3%	mpi_barrier_
16.7%	16.7%	mpi_barrier_
13.6%	<0.1%	⊞ mpiiowrite
0.8%	0.7%	⊞ 7 others

Showing data from 512,000 samples taken over 512 processes (1000 per process)

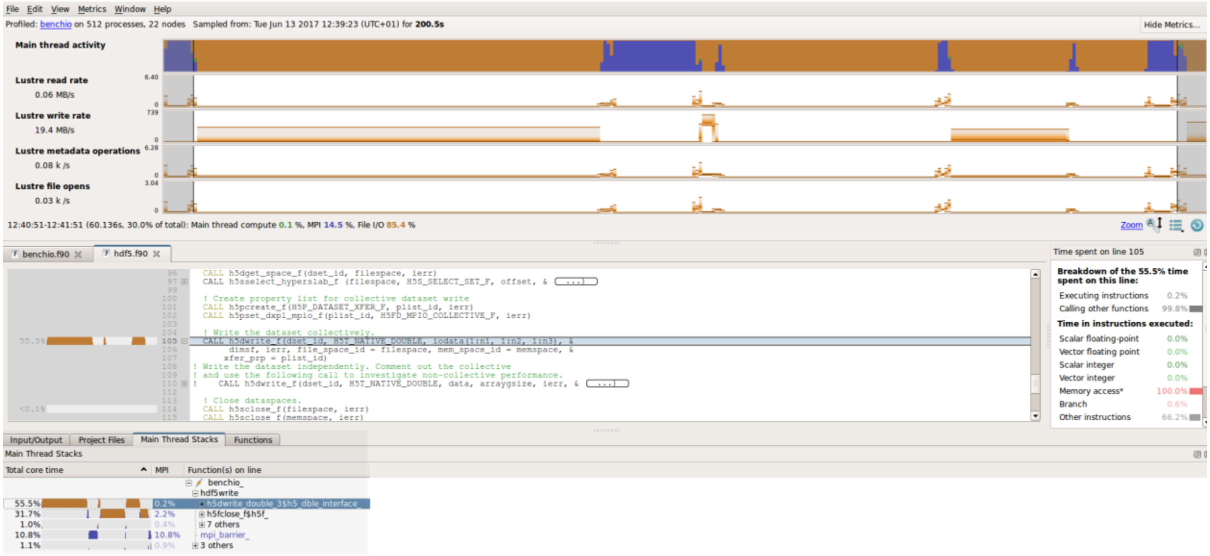
Investigation of HDF5 I/O

This section starts demonstrates how to investigate the performance of I/O in the applications that have been considered. Only the performance of the HDF5 output data is considered. Interpreting the performance of the other methods of I/O presented is left as an exercise.

- Look at the bandwidth reported in the output file benchio_512p_128_1.o which is for the MAP profile benchio_512p_128_1.map
- Output for HDF5 is as follows

```
------
HDF5
------

Writing to unstriped/hdf5.dat
time =  26.017146230908111 , rate =  314.86927610331014  MiB/s
Deleting: unstriped/hdf5.dat

Writing to striped/hdf5.dat
time =  15.180744504323229 , rate =  539.63097776048153  MiB/s
Deleting: striped/hdf5.dat

Writing to defstriped/hdf5.dat
time =  12.465544771635905 , rate =  657.17143936140462  MiB/s
Deleting: defstriped/hdf5.dat
```

- Identify when writing to HDF5 files was taking place by looking at the activity timeline in the stack view in the lower-most pane of the GUI
- Left click on the metrics timeline and drag to select when the `hdf5write` procedure was active. Left click to zoom in



Profiled: benchio on 512 processes, 22 nodes   Sampled from: Tue Jun 13 2017 12:39:23 (UTC+01) for 200.5s

12:40:51-12:41:51 (60.136s, 30.0% of total): Main thread compute 0.1 %, MPI 14.5 %, File I/O 85.4 %

```
 96   CALL h5dget_space_f(dset_id, filespace, ierr)
 97   CALL h5sselect_hyperslab_f (filespace, H5S_SELECT_SET_F, offset, & ...
 99
100   ! Create property list for collective dataset write
101   CALL h5pcreate_f(H5P_DATASET_XFER_F, plist_id, ierr)
102   CALL h5pset_dxpl_mpio_f(plist_id, H5FD_MPIO_COLLECTIVE_F, ierr)
103
104   ! Write the dataset collectively.
105   CALL h5dwrite_f(dset_id, H5T_NATIVE_DOUBLE, iodata(1:n1, 1:n2, 1:n3), &
106        dimsf, ierr, file_space_id = filespace, mem_space_id = memspace, &
107       xfer_prp = plist_id)
108 ! Write the dataset independently. Comment out the collective
109 ! and use the following call to investigate non-collective performance.
110 !    CALL h5dwrite_f(dset_id, H5T_NATIVE_DOUBLE, data, arraygsize, ierr, & ...
112
113   ! Close dataspaces.
114   CALL h5sclose_f(filespace, ierr)
115   CALL h5sclose_f(memspace, ierr)
```

Time spent on line 105

Breakdown of the 55.5% time spent on this line:

Executing instructions	0.2%
Calling other functions	99.8%

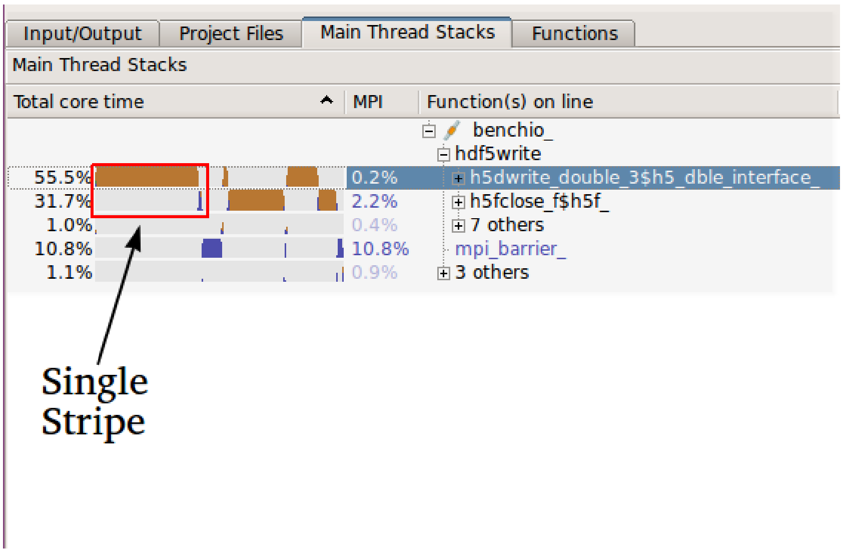
Time in instructions executed:

Scalar floating-point	0.0%
Vector floating point	0.0%
Scalar integer	0.0%
Vector integer	0.0%
Memory access*	100.0%
Branch	0.6%
Other instructions	66.2%

Main Thread Stacks

Total core time	MPI	Function(s) on line
		benchio_
		hdf5write
55.5%	0.2%	h5dwrite_double_3$h5_dble_interface_
31.7%	2.2%	h5fclose_f$h5f_
1.0%	0.4%	7 others
10.8%	10.8%	mpi_barrier_
1.1%	0.9%	3 others

- Expand the `hdf5write` procedure in the call stack in the lower-most section of the screen
- The activity is split into three parts – first for write to file with single stripe, then system selected stripe, then default striping.

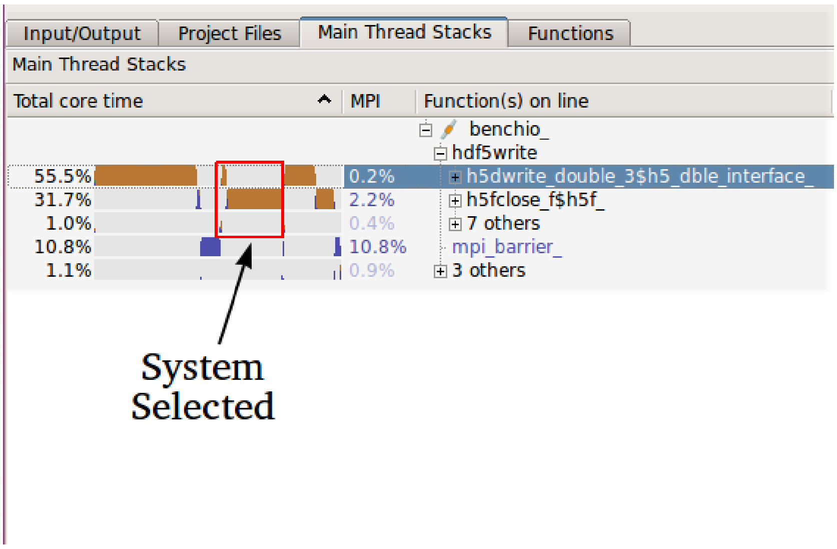


Input/Output | Project Files | Main Thread Stacks | Functions

Main Thread Stacks

Total core time	MPI	Function(s) on line
		⊟ benchio_
		⊟ hdf5write
55.5%	0.2%	⊞ h5dwrite_double_3$h5_dble_interface_
31.7%	2.2%	⊞ h5fclose_f$h5f_
1.0%	0.4%	⊞ 7 others
10.8%	10.8%	mpi_barrier_
1.1%	0.9%	⊞ 3 others

Single Stripe

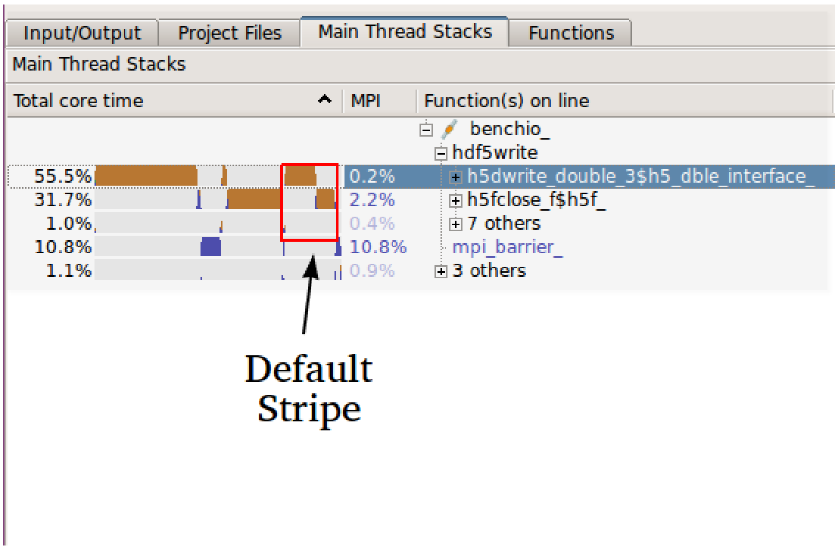


Input/Output | Project Files | Main Thread Stacks | Functions

Main Thread Stacks

Total core time	MPI	Function(s) on line
		⊟ benchio_
		⊟ hdf5write
55.5%	0.2%	⊞ h5dwrite_double_3$h5_dble_interface_
31.7%	2.2%	⊞ h5fclose_f$h5f_
1.0%	0.4%	⊞ 7 others
10.8%	10.8%	mpi_barrier_
1.1%	0.9%	⊞ 3 others

System Selected



Input/Output | Project Files | Main Thread Stacks | Functions

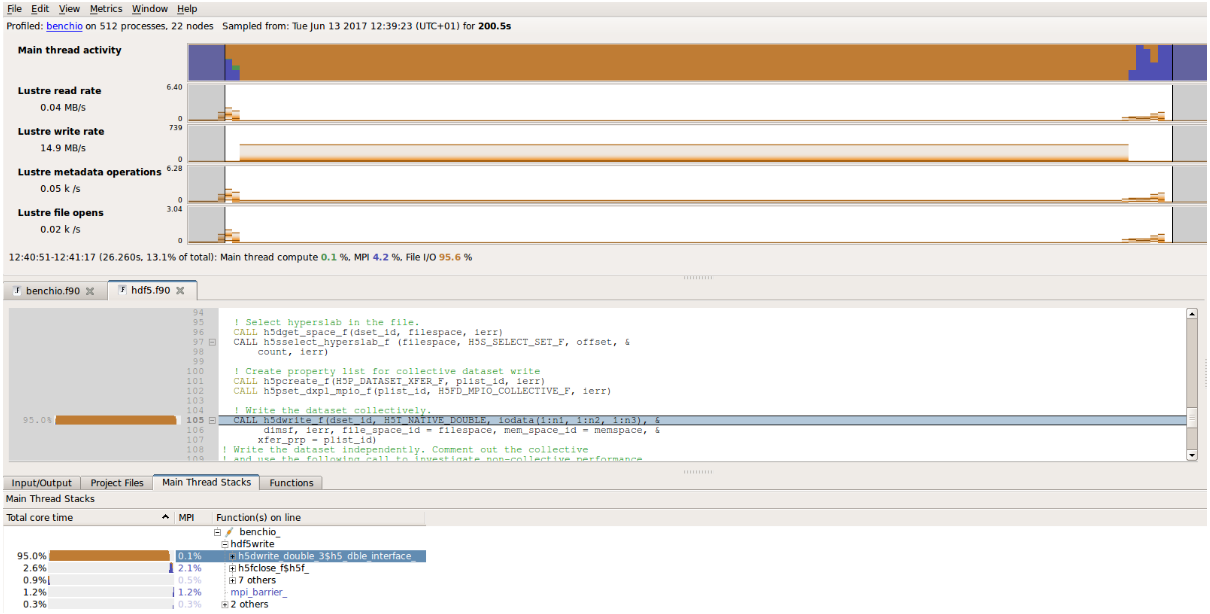
Main Thread Stacks

Total core time	MPI	Function(s) on line
		⊟ benchio_
		⊟ hdf5write
55.5%	0.2%	⊞ h5dwrite_double_3$h5_dble_interface_
31.7%	2.2%	⊞ h5fclose_f$h5f_
1.0%	0.4%	⊞ 7 others
10.8%	10.8%	mpi_barrier_
1.1%	0.9%	⊞ 3 others

Default Stripe

- Activity recorded in writing doubles and closing the file is different for each of the files

- To zoom out at any point right click in the metrics view in the top-most pane of the GUI
- Are bytes transferred to Lustre when closing the file? (Hint: look in the 'Lustre write rate' metric in the top-most section of the GUI)
- Look at the code. Does the bandwidth reported take into account only the time when the program is writing data? (Hint: Search in benchio.f90 for where procedure `benchtime` is called)
- Estimate what the bandwidth of the I/O was across the whole of a call to hdf5write. We demonstrate how to do this for the write to file with a single stripe.
- Select the section of the profile from the first HDF5 write to the end of the first `h5fclose`



Profiled: benchio on 512 processes, 22 nodes   Sampled from: Tue Jun 13 2017 12:39:23 (UTC+01) for **200.5s**

12:40:51-12:41:17 (26.260s, 13.1% of total): Main thread compute 0.1 %, MPI 4.2 %, File I/O 95.6 %

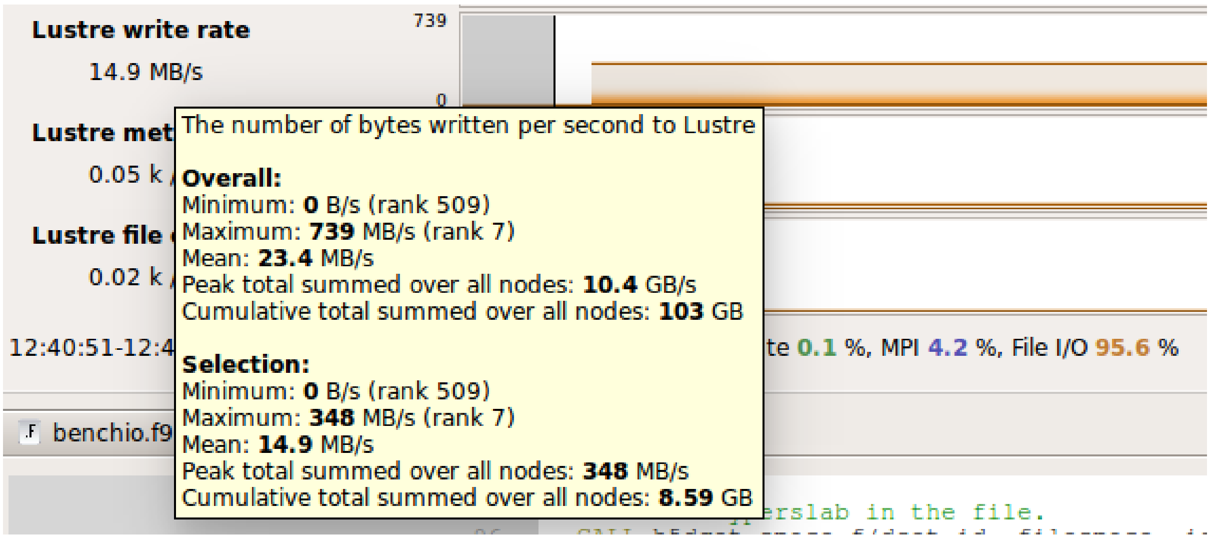
```
  ! Select hyperslab in the file.
  CALL h5dget_space_f(dset_id, filespace, ierr)
  CALL h5sselect_hyperslab_f (filespace, H5S_SELECT_SET_F, offset, &
      count, ierr)

  ! Create property list for collective dataset write
  CALL h5pcreate_f(H5P_DATASET_XFER_F, plist_id, ierr)
  CALL h5pset_dxpl_mpio_f(plist_id, H5FD_MPIO_COLLECTIVE_F, ierr)

  ! Write the dataset collectively.
  CALL h5dwrite_f(dset_id, H5T_NATIVE_DOUBLE, iodata(1:n1, 1:n2, 1:n3), &
       dimsf, ierr, file_space_id = filespace, mem_space_id = memspace, &
      xfer_prp = plist_id)
! Write the dataset independently. Comment out the collective
! and use the following call to investigate non-collective performance
```

95.0%

Main Thread Stacks

Total core time	MPI	Function(s) on line
		benchio_
		hdf5write
95.0%	0.1%	h5dwrite_double_3$h5_dble_interface_
2.6%	2.1%	h5fclose_f$h5f_
0.9%	0.5%	7 others
1.2%	1.2%	mpi_barrier_
0.3%	0.3%	2 others

- Hover the mouse over the 'Lustre write rate' label to get a tool-tip up



The number of bytes written per second to Lustre

Overall:
Minimum: **0** B/s (rank 509)
Maximum: **739** MB/s (rank 7)
Mean: **23.4** MB/s
Peak total summed over all nodes: **10.4** GB/s
Cumulative total summed over all nodes: **103** GB

Selection:
Minimum: **0** B/s (rank 509)
Maximum: **348** MB/s (rank 7)
Mean: **14.9** MB/s
Peak total summed over all nodes: **348** MB/s
Cumulative total summed over all nodes: **8.59** GB

- The label states that the metric is collected over all *nodes*, not over the processes. As there are 3 nodes used in this run (this information is displayed near the top of the GUI window) what is the combined bandwidth over all the nodes?
- How does this compare to the bandwidth reported in the benchmark output?

- How long does the closing of the file take for this file?
- How long does the write and the closing of the file take for the other HDF5 files?
- Does the time taken to close the file correlate to the number of stripes of the file? (Hint: file with the system defined striping has 48 stripes for this run, and default is 4 stripes)

Further investigation

The following questions may be answered in your own time, using the profiles provided in `TutorialData`. You may also perform you own investigation and ask questions about other parts of the MAP profiles that have been provided. The questions posed here should make you think about what the profiles are displaying, and how to draw conclusions about the operation of the I/O routines on the ARCHER system at EPCC.

- Which is the fastest method of performing I/O?
  - For 8 processes
  - For 64 processes
  - For 512 processes
- Which method of I/O seems to be the most scalable (i.e. has the highest bandwidth going to scale)?
- Which method of I/O has the least amount of overhead associated with it? In this context I define overhead to be the amount of time spent in I/O routines in which no data is being written to the file-system.
- Does the overhead associated with using libraries (such as MPI I/O, HDF5 and NetCDF) scale with the number of processes used, the file size or the number of stripes contained in a file?
- What is the best method of parallel I/O to use in general? (Hint: This is a very tricky question to answer, and if you find a good answer then please publish it!)

Resources for Allinea tools

If you are interested in getting to know the Allinea tool suite you may download our trial package. This is available from https://s3-eu-west-1.amazonaws.com/com.allinea.resources/trial_package.tar.gz

Visit https://www.allinea.com/products/downloads/evaluation-resources for more resources getting started. If you would like to use the tools on your own applications you are free to do so.

The licence for the Allinea tools with which you have been supplied is valid until Tuesday 20th June 2017. If you would like to continue testing the Allinea tools after this date please request a trial from https://www.allinea.com/get-your-free-allinea-forge-and-allinea-performance-reports-trial