Hewlett Packard Enterprise

Introduction to perftools

LUMI Advanced Workshop

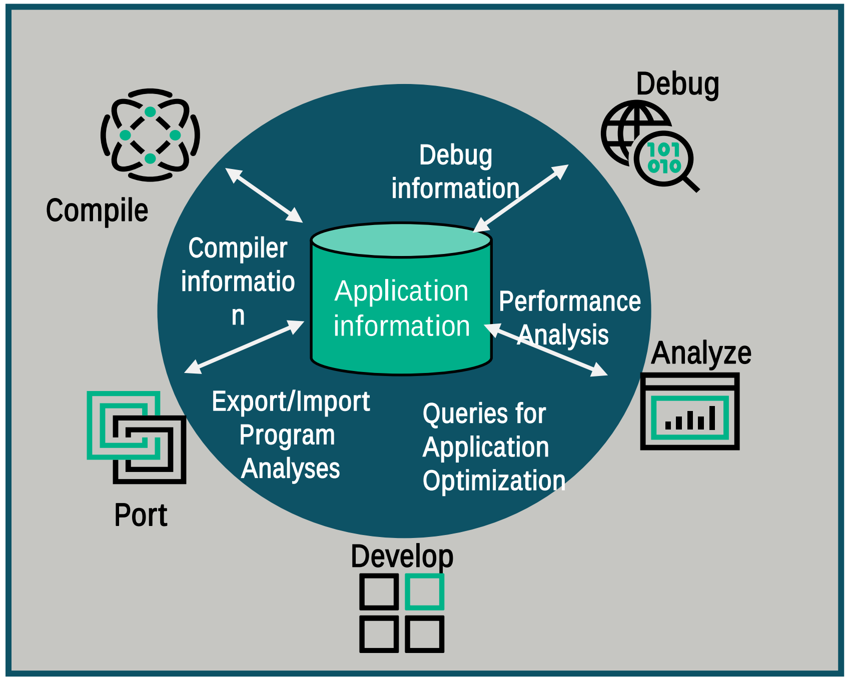
March 5-7, 2025

Agenda

- Introduction
  - General Remarks on Performance Analysis
  - Possible Bottlenecks and Remedies
- Perftools
  - Landscape and Properties
  - Sampling vs. Tracing
  - Performance analysis with `perftools-lite`
  - Apprentice2
- Demo

Motivation

- Very tempting to skip performance analysis when tests validate and time to solution is smaller after port or algorithm improvement.
- But performance does matter!
  - Might still be inefficient most of the of time.
  - Poor code performance can affect other users as well, for example because of bad I / O access patterns.
  - Want to efficiently use expensive resources and get as much information as possible for the allocated resources.
  - Simulating larger models may only be feasible after optimization.
- Applies to various scenarios
  - Code has been ported to a new system or otherwise significantly changed.
  - Application is running in production since a while.
  - Makes extensive use of third-party libraries (distributed ML, dense LA, …) or even fully proprietary or it is mostly home-grown code.

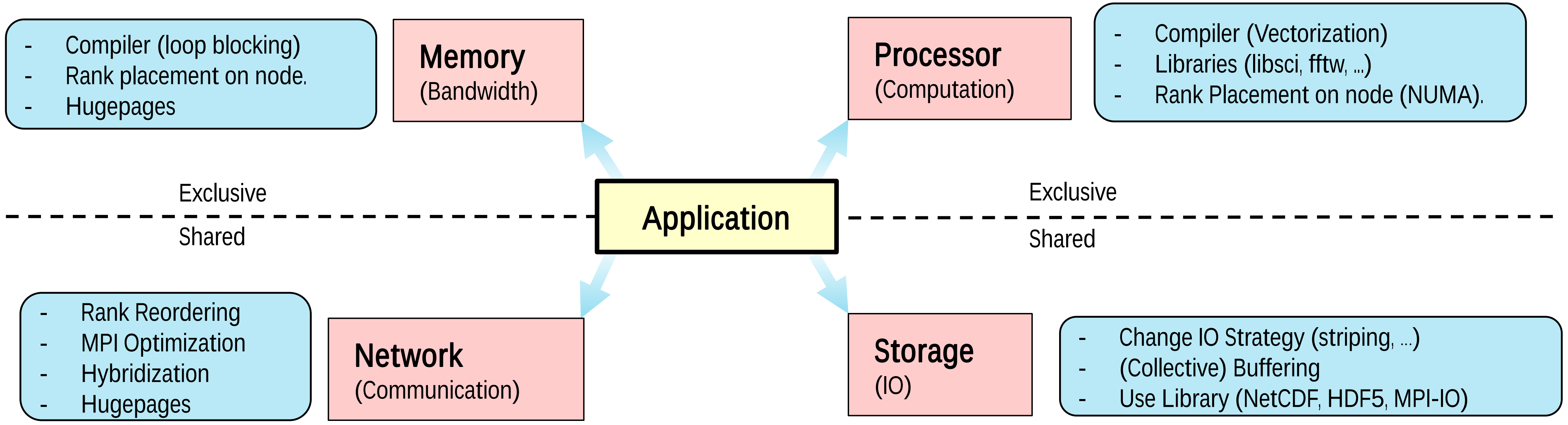


General Remarks on Performance Analysis

- Performance is usually associated to FLOPs/sec.
  - But what if your code does many scattered memory references like in Graph Analytics and not many FLOPs ?
  - Chose the metric which suits your application (like time to solution or updates/sec instead of FLOPs/sec) and keep that metric throughout the optimization process.
- Use examples with different sizes for your experiments.
  - Small, medium, large, where node count differs by an order of magnitude (strong or weak scaling or both).
    - Try the same model with different grid sizes or number of particles
  - Do not use fully artificial examples but rather meaningful representatives of your target (large scale) simulation.

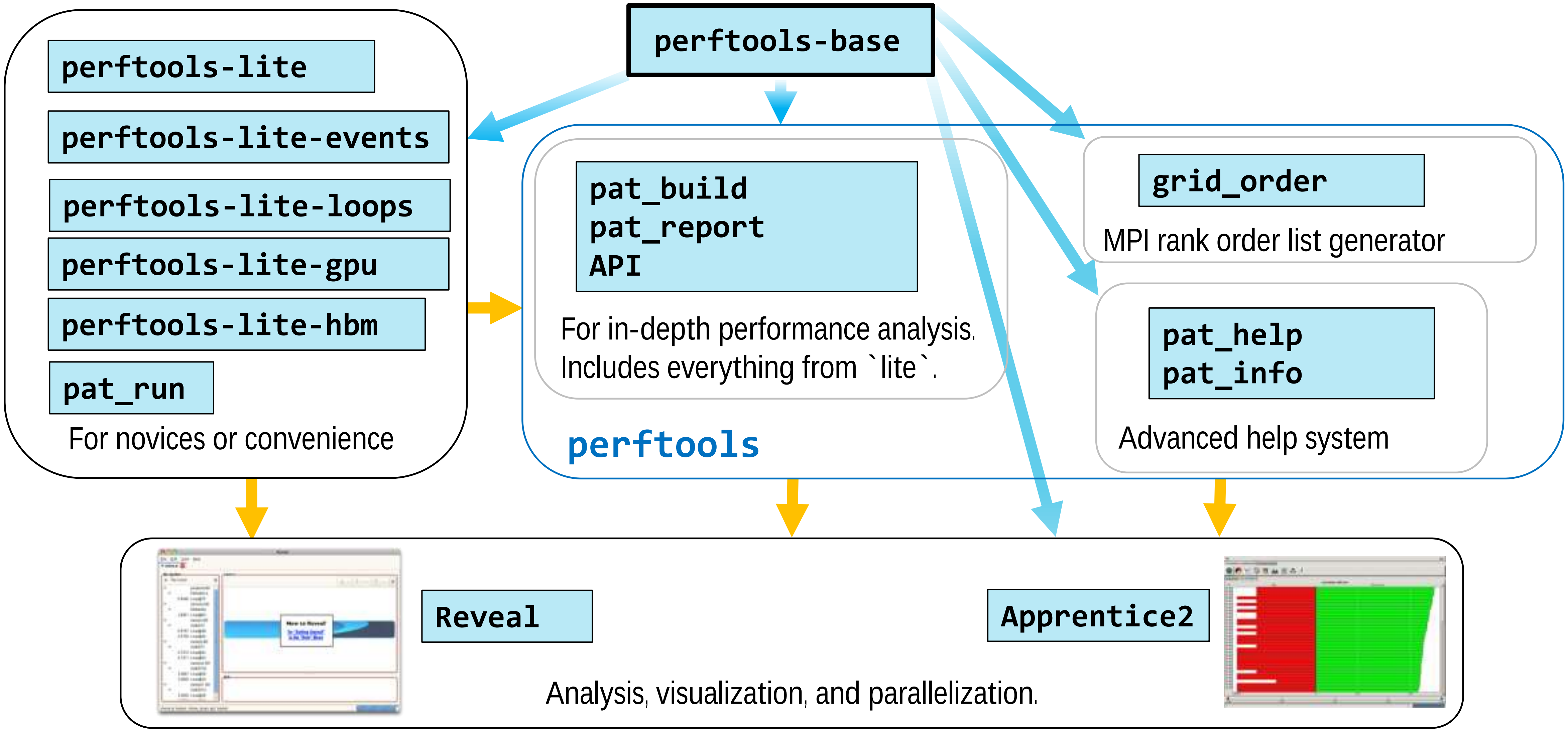
1. Start with low hanging fruits, i.e. avoid code modifications first.
   - Compiler flags, manual rank reordering, optimized libraries, huge pages, use hyperthreads, …
2. Use performance analysis tools
   - Identify critical code regions, guided rank reordering, Automatic parallelization (OpenMP), …
   - A good understanding of the workflow of your application (Communication, Computation, IO, …) helps to better interpret the profiles.

Bottlenecks and Remedies

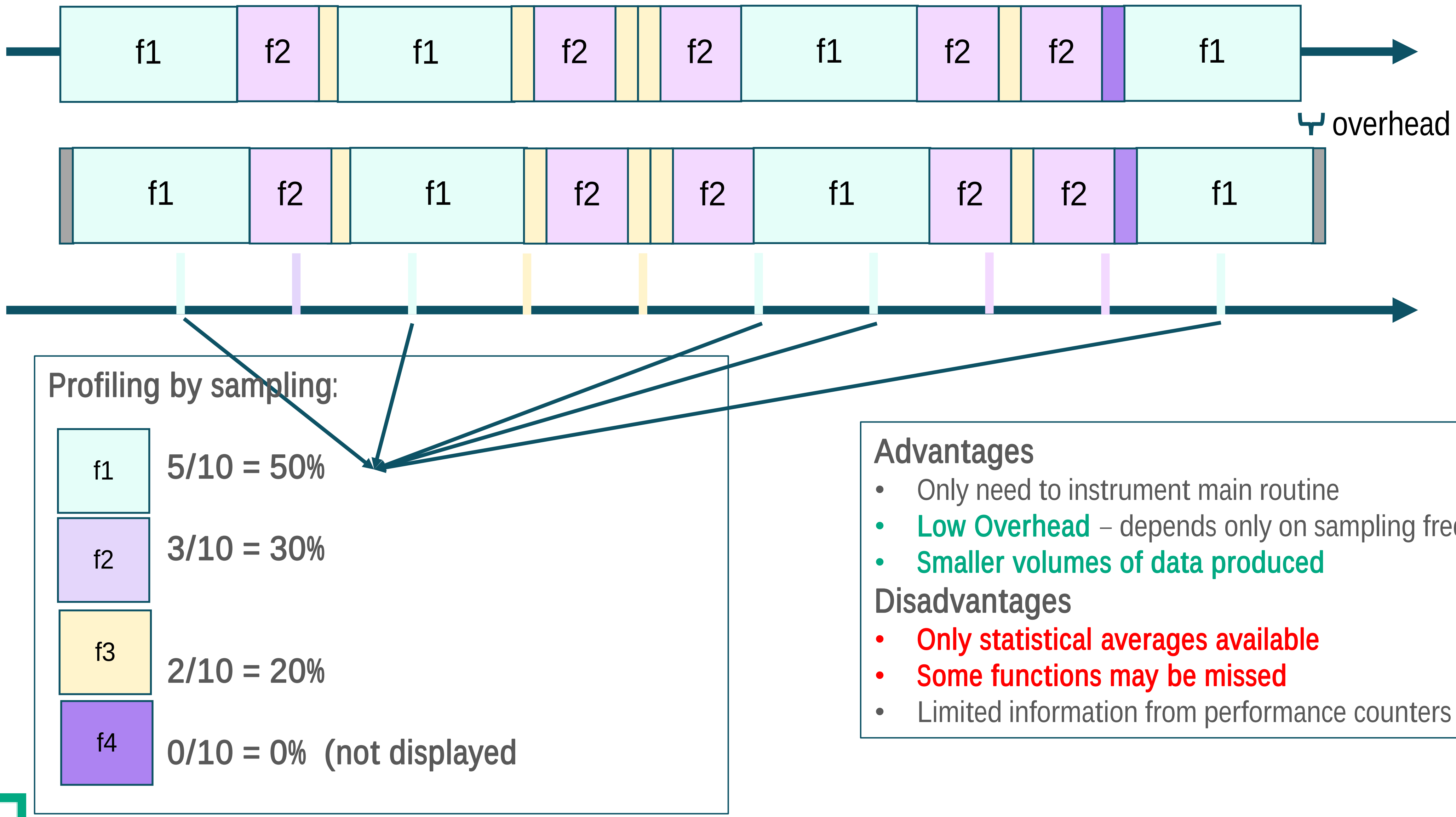


- Good: One bottleneck which can be easily resolved without creating a new one.
- Bad: Several bottlenecks interacting with each other and changing over time.
- Need a profiler to identify bottleneck(s) and a model to estimate optimization potential.

Perftools Landscape



First fundamental way of profiling : sampling



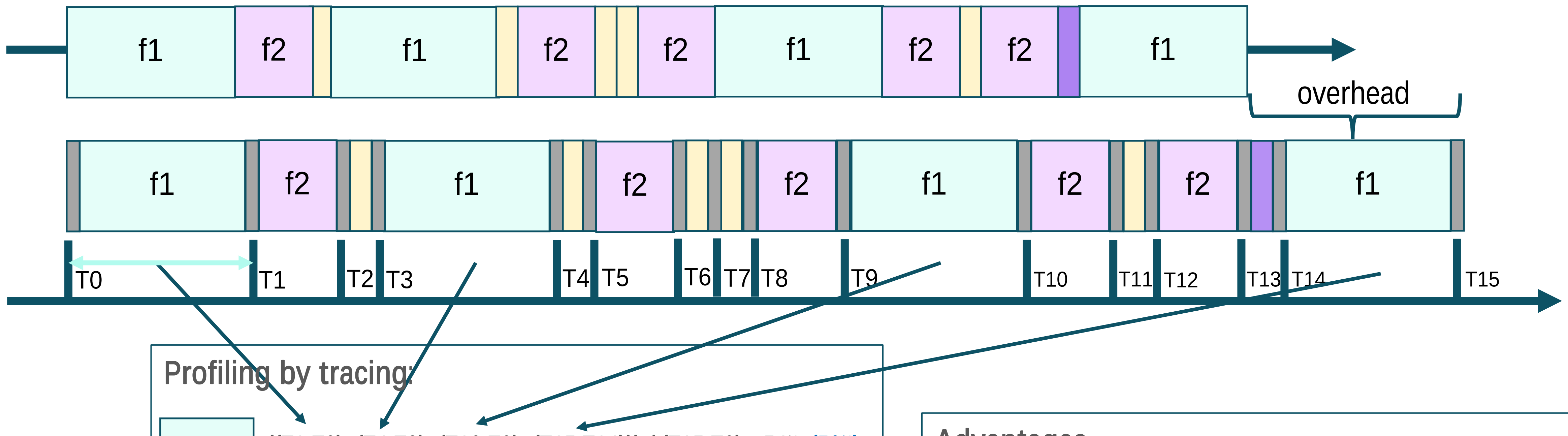
Advantages

- Only need to instrument main routine
- Low Overhead – depends only on sampling frequency
- Smaller volumes of data produced

Disadvantages

- Only statistical averages available
- Some functions may be missed
- Limited information from performance counters

Second fundamental way of profiling : tracing



Profiling by tracing:

f1	((T1-T0)+(T4-T3)+(T10-T9)+(T15-T14))) / (T15-T0) ~ 54% (50%)
f2	((T2-T1)+ . . . +(T13-T12)) / (T15-T0) ~ 35% (30%)
f3	((T3-T2)+ . . . +(T12-T11) / (T15-T0) ~ 9% (20%)
f4	((t14-T13) / (T15-T0) ~ 2% (0%)

Advantages
- More accurate and more detailed information
- Data collected from every traced function call

Disadvantages
- Overheads increases with number of function calls
- Huge volumes of data generated

Sampling outputs (xx%)

Other Experiments

- Guided Tracing: Combining Sampling and Tracing
  - Trace only functions that are not small (i.e. very few lines of code) and contribute a lot to application's run time.
  - Automatic performance Analysis (APA) is an automated way to do this. (Not intended for `perftools-lite`)
    1. In a first round identify important routines with a sampling experiment.
    2. Then do a tracing of only these important routines to reduce overhead.
- Loop Work Estimates
  - Special flavor of event tracing targeting loops.
  - Only for Cray programming environment.
  - Useful starting point for porting to multicore or GPUs.
  - Can be used as input to Reveal.
  - Also available with `perftools-lite-loops`

Perftools-lite

An easy-to-use version of the Perftools Performance Measurement and Analysis Tool

Test 1: Generate a Sampling Profile

```
$> module load perftools-base
$> module load perftools-lite
```

- Subsequent compiler invocations (`cc,CC,ftn`) will automatically insert necessary hooks for profiling (not always up-to-date with latest third-party compilers)

```
$> make clean; make
```

- Resulting executable `app.exe` is automatically instrumented. Object files needed!
- Not instrumented application stored as `app.exe+orig`

```
$> srun -n 8 app.exe >& job.out
```

- The report is printed to stdout.
- Successful execution creates a `app.exe+*/` directory for further analysis. Change this directory name with the `PAT_RT_EXPDIR_NAME` environment variable.
- Use the environment variable `PAT_RT_SAMPLING_INTERVAL` to change sampling interval (default is 10000 microseconds)

Output: Sampling

Regular program output

```
CrayPat/X:  Version 22.06.0 Revision 4b5ab6256  05/21/22 02:03:49
Sequential version array size
 mimax = 513 mjmax = 513 mkmax = 1025
Parallel version array size
 mimax = 259 mjmax = 259 mkmax = 515
imax = 257 jmax = 257 kmax =513
I-decomp = 2 J-decomp = 2 K-decomp =2
 Start rehearsal measurement process.
 Measure the performance in 100 times.

 MFLOPS: 15638.324504 time(s): 5.779363 2.794488e-04

 Now, start the actual measurement process.
 The loop will be excuted in 50 times
 This will take about one minute.
 Wait for a while

cpu : 28.858915 sec.
Loop executed for 50 times
Gosa : 2.763023e-04
MFLOPS measured : 15658.861128
Score based on Pentium III 600MHz : 189.025364
```

```
##################################################
#                                                #
#    CrayPat-lite Performance Statistics         #
#                                                #
##################################################

CrayPat/X:  Version 22.06.0 Revision 4b5ab6256  05/21/22 02:03:49
Experiment:                  lite  lite/sample_profile
Number of PEs (MPI ranks):      8
Numbers of PEs per Node:        8
Numbers of Threads per PE:      1
Number of Cores per Socket:    16

Execution start time:  Sat Aug 20 23:43:46 2022
System name and speed:  nid005017  2.626 GHz (nominal)
AMD   Trento             CPU  Family: 25  Model: 48  Stepping:  1
Core Performance Boost:  All 8 PEs have CPB capability

Avg Process Time:     36.92 secs
High Memory:       15,544.6 MiBytes     1,943.1 MiBytes per PE
I/O Write Rate:   10.359011 MiBytes/sec
```

General job information

Sampling summary

```
Notes for table 1:

  This table shows functions that have significant exclusive sample
   hits, averaged across ranks.
  For further explanation, use:  pat_report -v -O samp_profile ...

Table 1:  Profile by Function

  Samp% |    Samp | Imb. |  Imb. | Group
        |         | Samp | Samp% |  Function=[MAX10]
        |         |      |       |   PE=HIDE

 100.0% | 3,693.2 |   -- |    -- | Total
|-----------------------------------------------------
|  84.1% | 3,107.8 |   -- |    -- | USER
||----------------------------------------------------
||  79.6% | 2,938.8 | 65.2 |  2.5% | jacobi
||   4.6% |   169.0 |  2.0 |  1.3% | initmt
||====================================================
|  14.7% |   541.5 |   -- |    -- | ETC
||----------------------------------------------------
||  13.3% |   490.2 | 31.8 |  7.0% | __cray_memcpy_ROME
||   1.4% |    50.9 |  4.1 |  8.6% | __cray_memset_ROME
||====================================================
|   1.1% |    41.8 |   -- |    -- | MPI
|=====================================================

...
```

Further analysis

```
Notes for table 3:

  This table shows energy and power usage for the nodes with the
    maximum, mean, and minimum usage, as well as the sum of usage over
    all nodes.
    Energy and power for accelerators is also shown, if applicable.
  For further explanation, use:  pat_report -v -O program_energy ...

Table 3:  Program energy and power usage (from Cray PM)

   Node |     Node |   Process | PE=HIDE
 Energy |    Power |      Time |
    (J) |      (W) |           |
----------------------------------------
 25,504 |  687.169 | 37.114590 | Total
========================================

Notes for table 4:

  This table show the average time and number of bytes written to each
    output file, taking the average over the number of ranks that
    wrote to the file.  It also shows the number of write operations,
    and average rates.
  For further explanation, use:  pat_report -v -O write_stats ...

Table 4:  File Output Stats by Filename

      Avg |      Avg |  Write Rate | Number |    Avg | Bytes/ | File
Name=!x/^/(proc|sys)/
    Write |    Write | MiBytes/sec |     of | Writes |   Call |  PE=HIDE
 Time per |  MiBytes |             | Writer |    per |        |
   Writer |      per |             |  Ranks | Writer |        |
     Rank |   Writer |             |        |   Rank |        |
          |     Rank |             |        |        |        |
|-----------------------------------------------------------------------
| 0.000052 | 0.000582 |   11.224027 |      1 |   18.0 |  33.89 | stdout
|=======================================================================

...
===============  End of CrayPat-lite output  ========================
```

Test 2: Generate an Event Profile

```
$> module sw perftools-lite perftools-lite-events
```

- If `perftools-lite` module not loaded, load subsequently `perftools-base` and `perftools-lite-events`.

```
$> rm app.exe; make
```

- Only relink of `app.exe` necessary if object files and user libraries have been generated with another `perftools-lite*` module.
- Otherwise do a `make clean; make`

```
$> srun -n 8 app.exe >& job.out
```

- The report is printed to stdout.
- Successful execution creates a `app.exe+*/` directory for further analysis.

Output: Event Tracing

Regular program output

```
CrayPat/X:  Version 22.06.0 Revision 4b5ab6256
05/21/22 02:03:49
 Sequential version array size
  mimax= 1025  mjmax= 513  mkmax= 513
 Parallel version  array size
  mimax= 515  mjmax= 259  mkmax= 259
  imax= 513  jmax= 257  kmax= 257
  I-decomp=  2  J-decomp=  2  K-decomp=  2

  Start rehearsal measurement process.
  Measure the performance in 3 times.
   MFLOPS: 15604.522995096944   time(s):
5.7918816709999987,  4.324619949E-4
 Now, start the actual measurement process.
 The loop will be excuted in 50  times.
 This will take about one minute.
 Wait for a while.
  Loop executed for  50  times
  Gosa : 4.21404955E-4
  MFLOPS: 15611.16692542893   time(s): 28.947083569
  Score based on Pentium III 600MHz : 188.449631
```

```
############################################################
#                                                          #
#     CrayPat-lite Performance Statistics                  #
#                                                          #
############################################################

CrayPat/X:  Version 22.06.0 Revision 4b5ab6256  05/21/22
02:03:49
Experiment:                  lite  lite-events
Number of PEs (MPI ranks):      8
Numbers of PEs per Node:        8
Numbers of Threads per PE:      1
Number of Cores per Socket:    64
Execution start time:  Fri Nov 18 16:02:17 2022
System name and speed:  nid001050  2.078 GHz (nominal)
AMD   Milan              CPU  Family: 25  Model:  1
Stepping:  1
Core Performance Boost:  All 8 PEs have CPB capability


Avg Process Time:    37.05 secs
High Memory:      15,667.4 MiBytes     1,958.4 MiBytes per PE
I/O Write Rate:   3.182425 MiBytes/sec
```

General job information

Event tracing summary. Note difference to sampling

```
Notes for table 1:

  This table shows functions that have significant exclusive time,
    averaged across ranks.
  For further explanation, use:  pat_report -v -O profile ...

Table 1:  Profile by Function Group and Function

  Time% |      Time |    Imb. |  Imb. |    Calls | Group
        |           |    Time | Time% |          |  Function=[MAX10]
        |           |         |       |          |   PE=HIDE

 100.0% | 37.007250 |      -- |    -- | 1,461.4 | Total
|-------------------------------------------------------------------
|  98.4% | 36.410057 |      -- |    -- |     3.0 | USER
||------------------------------------------------------------------
||  76.9% | 28.457104 | 0.371909 |  1.5% |     1.0 | jacobi_
||  15.4% |  5.690022 | 0.078645 |  1.6% |     1.0 | himenobmtxp_
||   6.1% |  2.262931 | 0.004594 |  0.2% |     1.0 | initmt_
||==================================================================
|   1.6% |  0.585681 |      -- |    -- |   983.0 | MPI
||------------------------------------------------------------------
||   1.3% |  0.482364 | 0.614413 | 64.0% |   180.0 | MPI_WAITALL
|===================================================================

Observation:  MPI utilization

    No suggestions were made because all ranks are on one node.
```

```
Notes for table 2:

  This table shows energy and power usage for the nodes with the
    maximum, mean, and minimum usage, as well as the sum of usage over
    all nodes.
    Energy and power for accelerators is also shown, if applicable.
  For further explanation, use:  pat_report -v -O program_energy ...

Table 2:  Program energy and power usage (from Cray PM)

   Node |    Node |   Process | PE=HIDE
 Energy |   Power |      Time |
    (J) |     (W) |           |
----------------------------------------
  9,055 | 244.407 | 37.048880 | Total
========================================
```

```
Notes for table 3:

  This table show the average time and number of bytes written to each
    output file, taking the average over the number of ranks that
    wrote to the file.  It also shows the number of write operations,
    and average rates.
  For further explanation, use:  pat_report -v -O write_stats ...

Table 3:  File Output Stats by Filename

      Avg |      Avg |  Write Rate | Number |    Avg | Bytes/ | File
Name=!x/^/(proc|sys)/
    Write |    Write | MiBytes/sec |     of | Writes |   Call |  PE=HIDE
 Time per |  MiBytes |             | Writer |    per |        |
   Writer |      per |             |  Ranks | Writer |        |
     Rank |   Writer |             |        |   Rank |        |
          |     Rank |             |        |        |        |
|----------------------------------------------------------------------------
| 0.000024 | 0.000008 |    0.321773 |      8 |    1.0 |   8.00 | stderr
| 0.000022 | 0.000613 |   27.644603 |      1 |   18.0 |  35.72 | stdout
|============================================================================
```

```
Program invocation:  himeno.exe

For a complete report with expanded tables and notes, run:
  pat_report /pfs/lustrep2/projappl/project_465000297/alfiolaz/work/perftools-
lite/test/expfile.lite-events.2046565

For help identifying callers of particular functions:
  pat_report -O callers+src
/pfs/lustrep2/projappl/project_465000297/alfiolaz/work/perftools-
lite/test/expfile.lite-events.2046565
To see the entire call tree:
  pat_report -O calltree+src
/pfs/lustrep2/projappl/project_465000297/alfiolaz/work/perftools-
lite/test/expfile.lite-events.2046565

For interactive, graphical performance analysis, run:
  app2 /pfs/lustrep2/projappl/project_465000297/alfiolaz/work/perftools-
lite/test/expfile.lite-events.2046565

================  End of CrayPat-lite output  ==========================
```

Further analysis

Test 3: Generate a Loop Profile (CCE only)

```
$> module sw perftools-lite perftools-lite-loops
```

- If perftools-lite module not loaded, load subsequently perftools-base and perftools-lite-loops. Only for `PrgEnv-cray`.

```
$> make clean; make
```

- Need to clean everything and rebuild.
- Compiler drivers will use `-h profile_generate` for Fortran or `-finstrument-loops` for C implicitly. This flag turns off OpenMP and significant compiler loop restructuring optimizations except for vectorization.

```
$> srun –n 8 app.exe >& job.out
```

- Successful execution creates a `app.exe+*/` directory for further analysis.
- The report is printed to stdout.

Output: Loop Profile

Regular program output

```
CrayPat/X:  Version 22.06.0 Revision 4b5ab6256  05/21/22 02:03:49
Sequential version array size
 mimax = 513 mjmax = 513 mkmax = 1025
Parallel version array size
 mimax = 259 mjmax = 259 mkmax = 515
imax = 257 jmax = 257 kmax =513
I-decomp = 2 J-decomp = 2 K-decomp =2
 Start rehearsal measurement process.
 Measure the performance in 100 times.

 MFLOPS: 15652.801400 time(s): 5.774017 2.794488e-04

 Now, start the actual measurement process.
 The loop will be excuted in 50 times
 This will take about one minute.
 Wait for a while

cpu : 28.869171 sec.
Loop executed for 50 times
Gosa : 2.763023e-04
MFLOPS measured : 15653.298226
Score based on Pentium III 600MHz : 188.958211
```

```
#################################################################
#                                                               #
#            CrayPat-lite Performance Statistics                #
#                                                               #
#################################################################

CrayPat/X:  Version 22.06.0 Revision 4b5ab6256  05/21/22 02:03:49
Experiment:                  lite  lite-loops
Number of PEs (MPI ranks):      8
Numbers of PEs per Node:        8
Numbers of Threads per PE:      1
Number of Cores per Socket:    64
Accelerator Model: AMD MI200 Memory: 32.00 GB Frequency: 1.09 GHz


Execution start time:  Sun Aug 21 00:29:19 2022
System name and speed:  nid005013  2.361 GHz (nominal)
AMD   Trento               CPU  Family: 25  Model: 48  Stepping:  1
Core Performance Boost:  All 8 PEs have CPB capability



Avg Process Time:     37.12 secs
High Memory:      15,593.3 MiBytes 1,949.2 MiBytes per PE
```

General job information

Loop Statistics by function

```
Notes for table 1:

  This table shows a nested view of loops that have significant inclusive time, averaged across ranks. Intervening function calls
    are not shown (as if all functions were inlined).For each loop, the table shows its inclusive time, the number of
    times it was executed, and the average number of iterations for each execution. Times in this table include overhead from loop
    instrumentation. For an alternative view that shows min and max iterations, and exclusive times, use the option:  -O loop_times
  For further explanation, use:  pat_report -v -O loop_nest ...

Table 1:  Nested view of Loop Inclusive Time

   Incl |  Incl | Loop Exec |  Loop | Calltree=/[.]LOOP[.]
  Time% |  Time |           | Trips |  PE=HIDE
        |       |           |   Avg |

 100.0% | 37.08 |        -- |    -- | Total
|-------------------------------------------------------
1  93.4% | 34.64 |         2 |  30.0 | jacobi.LOOP.1.li.236
||------------------------------------------------------
2|  79.0% | 29.31 |        60 | 255.0 | jacobi.LOOP.2.li.240
|||-----------------------------------------------------
3||  79.0% | 29.31 |    15,300 | 255.0 | jacobi.LOOP.3.li.241
||||----------------------------------------------------
4|||  78.2% | 29.00 | 3,901,500 | 511.0 | jacobi.LOOP.4.li.242
|||=====================================================
2|  13.5% |  5.02 |        60 | 255.0 | jacobi.LOOP.5.li.263
|||-----------------------------------------------------
3||  13.5% |  5.02 |    15,300 | 255.0 | jacobi.LOOP.6.li.264
||||----------------------------------------------------
4|||   9.6% |  3.54 | 3,901,500 | 511.0 | jacobi.LOOP.7.li.265
||======================================================
1   4.3% |  1.61 |         1 | 259.0 | initmt.LOOP.4.li.191
||------------------------------------------------------
2|   4.3% |  1.61 |       259 | 259.0 | initmt.LOOP.5.li.192
|||-----------------------------------------------------
3||   4.3% |  1.59 |    67,081 | 515.0 | initmt.LOOP.6.li.193
||======================================================
1   2.2% |  0.82 |         1 | 257.0 | initmt.LOOP.1.li.210
||------------------------------------------------------
2|   2.2% |  0.82 |       257 | 257.0 | initmt.LOOP.2.li.211
|||-----------------------------------------------------
3||   2.1% |  0.79 |    66,049 | 513.0 | initmt.LOOP.3.li.212
|=======================================================
Program invocation:  himeno.exe

For a complete report with expanded tables and notes, run:
  pat_report /pfs/lustrep3/users/lazzaroa/exercises/perftools-lite/C/himeno.exe+44287-8745533t

For help identifying callers of particular functions:
  pat_report -O callers+src /pfs/lustrep3/users/lazzaroa/exercises/perftools-lite/C/himeno.exe+44287-8745533t
To see the entire call tree:
  pat_report -O calltree+src /pfs/lustrep3/users/lazzaroa/exercises/perftools-lite/C/himeno.exe+44287-8745533t

For interactive, graphical performance analysis, run:
  app2 /pfs/lustrep3/users/lazzaroa/exercises/perftools-lite/C/himeno.exe+44287-8745533t

================  End of CrayPat-lite output  ==========================
```

Subroutine

Line number

Nested Loops

Generate an Event Profile for GPU Experiments

```
$> module load perftools-lite-gpu
```

- If `perftools-lite` module not loaded, load subsequently `perftools-base` and `perftools-lite-gpu`.

```
$> rm app.exe; make
```

- Only relink of `app.exe` necessary if object files and user libraries have been generated with another `perftools-lite*` module.
- Otherwise do a `make clean; make`

```
$> srun -n 8 app.exe >& job.out
```

- The report is printed to stdout.
- Successful execution creates a `app.exe+*/` directory for further analysis.

Output: GPU Profile (MPI+ompenMP offload)

Regular program output

```
CrayPat/X:  Version 24.03.0 Revision 1705b3005 sles15.5_x86_64
02/20/24 20:53:05
Sequential version array size
 mimax = 513 mjmax = 513 mkmax = 1025
Parallel version array size
 mimax = 259 mjmax = 259 mkmax = 515
imax = 257 jmax = 257 kmax =513
I-decomp = 2 J-decomp = 2 K-decomp =2
 Start rehearsal measurement process.
 Measure the performance in 10 times.

 MFLOPS: 8127.332464 time(s): 11.120444 4.351904e-04

 Now, start the actual measurement process.
 The loop will be excuted in 50 times
 This will take about one minute.
 Wait for a while

cpu : 54.424242 sec.
Loop executed for 50 times
Gosa : 4.239466e-04
MFLOPS measured : 8303.243579
Score based on Pentium III 600MHz : 100.232298
```

```
#################################################################
#                                                               #
#            CrayPat-lite Performance Statistics                #
#                                                               #
#################################################################

CrayPat/X:  Version 24.03.0 Revision 1705b3005 sles15.5_x86_64  02/20/24
20:53:05
Experiment:                   lite  lite-gpu
Number of PEs (MPI ranks):       8
Numbers of PEs per Node:         8
Numbers of Threads per PE:       1
Number of Cores per Socket:     64
Accelerator Model: AMD MI200 Series; Memory: 64.00 GB; Frequency: 1.70 GHz


Execution start time:  Wed Jan  8 11:09:12 2025
System name and speed:  nid005976  2.000 GHz (nominal)
AMD   Trento               CPU  Family: 25  Model: 48  Stepping:  1
Core Performance Boost:  All 8 PEs have CPB capability


Avg Process Time:    67.78 secs
High Memory:      15,162.1 MiBytes     1,895.3 MiBytes per PE
I/O Read Rate:    9.722530 MiBytes/sec
I/O Write Rate:   7.356752 MiBytes/sec
```

General job information

gpu profiling summary.

```
Table 1:  Caller View by Acc Id of Accelerator Time and Bytes Transferred

  Time% |      Time | Function
        |           |  Caller
        |           |   PE=HIDE

 100.0% | 67.624946 | Total
|---------------------------------------------------------------------------
|  88.4% | 59.774645 | __omp_offloading_73ac72ce_d201d046_jacobi_l242_debug__.REGION@li.244
|        |           |  __omp_offloading_73ac72ce_d201d046_jacobi_l242_debug__.REGION@li.244(ovhd):himeno.c:line.244
|   7.9% |  5.361571 | __omp_offloading_73ac72ce_d201d046_jacobi_l268_debug__.REGION@li.270
|        |           |  __omp_offloading_73ac72ce_d201d046_jacobi_l268_debug__.REGION@li.270(ovhd):himeno.c:line.270
|   3.0% |  2.055567 | initmt
|        |           |  main:himeno.c:line.102
|===========================================================================
```

```
Table 3:  Program Energy and Power Usage from Cray PM

PE=HIDE


==============================================================
  Total
--------------------------------------------------------------
  PM Energy Node   563 W    38,137 J
  PM Energy Cpu     84 W     5,672 J
  PM Energy Memory  78 W     5,287 J
  PM Energy Acc0    98 W     6,630 J
  PM Energy Acc1    92 W     6,256 J
  PM Energy Acc2    95 W     6,452 J
  PM Energy Acc3    93 W     6,300 J
  Process Time          67.775526 secs
==============================================================
```

```
Table 5:  File Output Stats by Filename

      Avg |       Avg |  Write Rate | Number |     Avg | Bytes/ | File Name=!x/^/(proc|sys)/
    Write |     Write | MiBytes/sec |     of |  Writes |   Call |  PE=HIDE
 Time per |   MiBytes |             | Writer |     per |        |
   Writer |       per |             |  Ranks |  Writer |        |
     Rank |    Writer |             |        |    Rank |        |
          |      Rank |             |        |         |        |
|---------------------------------------------------------------------------
| 0.000080 | 0.000587 |    7.356752 |      1 |    18.0 |  34.17 | stdout
|===========================================================================
```

```
Program invocation:  himeno.exe

For a complete report with expanded tables and notes, run:
  pat_report
/pfs/lustrep4/projappl/project_462000031/bracconi/Training/training_exercises/p
erftools-lite-gpu/C/expfile..9012372

For help identifying callers of particular functions:
  pat_report -O callers+src
/pfs/lustrep4/projappl/project_462000031/bracconi/Training/training_exercises/p
erftools-lite-gpu/C/expfile..9012372
To see the entire call tree:
  pat_report -O calltree+src
/pfs/lustrep4/projappl/project_462000031/bracconi/Training/training_exercises/p
erftools-lite-gpu/C/expfile..9012372

For interactive, graphical performance analysis, run:
  app2
/pfs/lustrep4/projappl/project_462000031/bracconi/Training/training_exercises/p
erftools-lite-gpu/C/expfile..9012372
```

Further analysis

Observations and Remarks

- No intervention needed for build system and batch scripts.
  - Only make sure to use the compiler driver wrappers `CC`, `cc`, and `ftn`.
- What we did not see with these simple tests:
  - `perftools-lite` can produce rank reordering files for MPI to optimize the communication. Not visible here because of small portion of time spent in communication or job size.
  - The resulting `app+exe*/` directory can be processed with `pat_report`, Apprentice2, and Reveal for further analysis. From the sample experiment one can retrieve hardware performance counter information.
- Tailored profiling, i.e. for specific routines, trace groups, or specific portions of the code is not possible.
  - Need the regular `perftools` module for this in-depth analysis.
- `CRAYPAT_LITE` environment variable can be used to distinct output files.
- Record Subset of PEs during execution: `export PAT_RT_EXPFILE_PES=0,4,5,10`
- Use `CRAYPAT_LITE_WHITELIST` for binaries you DO want instrumented (rest ignored).

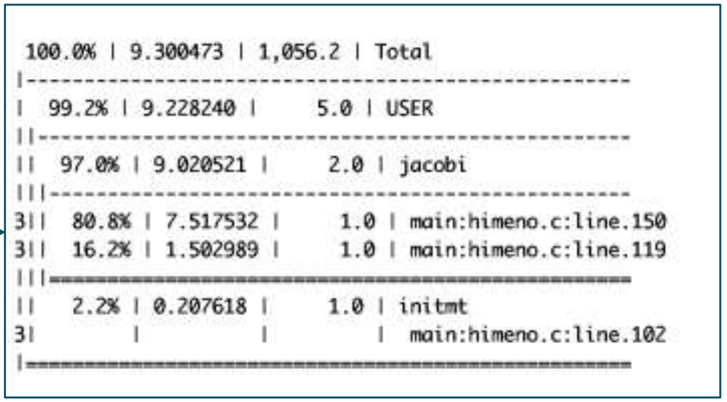
Further analysis (without re-running)

- Generate full report

```
$> pat_report app.exe+pat*/ > rpt
```

- Generate report with call tree (or by callers)

```
$> pat_report -O calltree+src
$> pat_report -O callers+src
```



```
 100.0% | 9.300473 | 1,056.2 | Total
|-----------------------------------------------------
|  99.2% | 9.228240 |     5.0 | USER
||----------------------------------------------------
||  97.0% | 9.020521 |     2.0 | jacobi
|||---------------------------------------------------
3||  80.8% | 7.517532 |      1.0 | main:himeno.c:line.150
3||  16.2% | 1.502989 |      1.0 | main:himeno.c:line.119
|||===================================================
||   2.2% | 0.207618 |     1.0 | initmt
3|        |          |         |  main:himeno.c:line.102
|====================================================
```

- Show each MPI rank or each OpenMP thread in report

```
$> pat_report -s pe=ALL
$> pat_report -s th=ALL
```

- Generate a preview of data before processing the full report

```
$> pat_report -Q1
```

  - Produces report from single (lexically first) '.ap2' file
  - Useful for jobs with large number of processes

Further analysis (without re-running)

- Generate report from specific subset of ranks

```
$> pat_report –s filter_input=‘pe==0’
```

  - Report with only PE 0 data

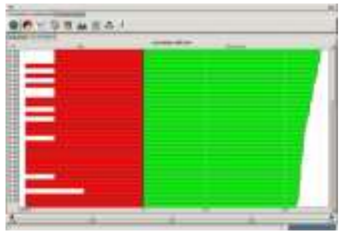
```
$> pat_report –s filter_intput=‘pe<5’
```

  - Report with data from first 5 ranks
  - Use `pat_help report filtering` for more details

- Don’t see an expected function?
  - Use the `pat_report –P` option to disable pruning.
  - You should be able to see the caller/callee relationship with `pat_report -P -O callers`
  - Use ‘`pat_report –T`’ to see functions that didn’t take much time
  - Still don’t see it? Check the compiler listing to see if the function was inlined.

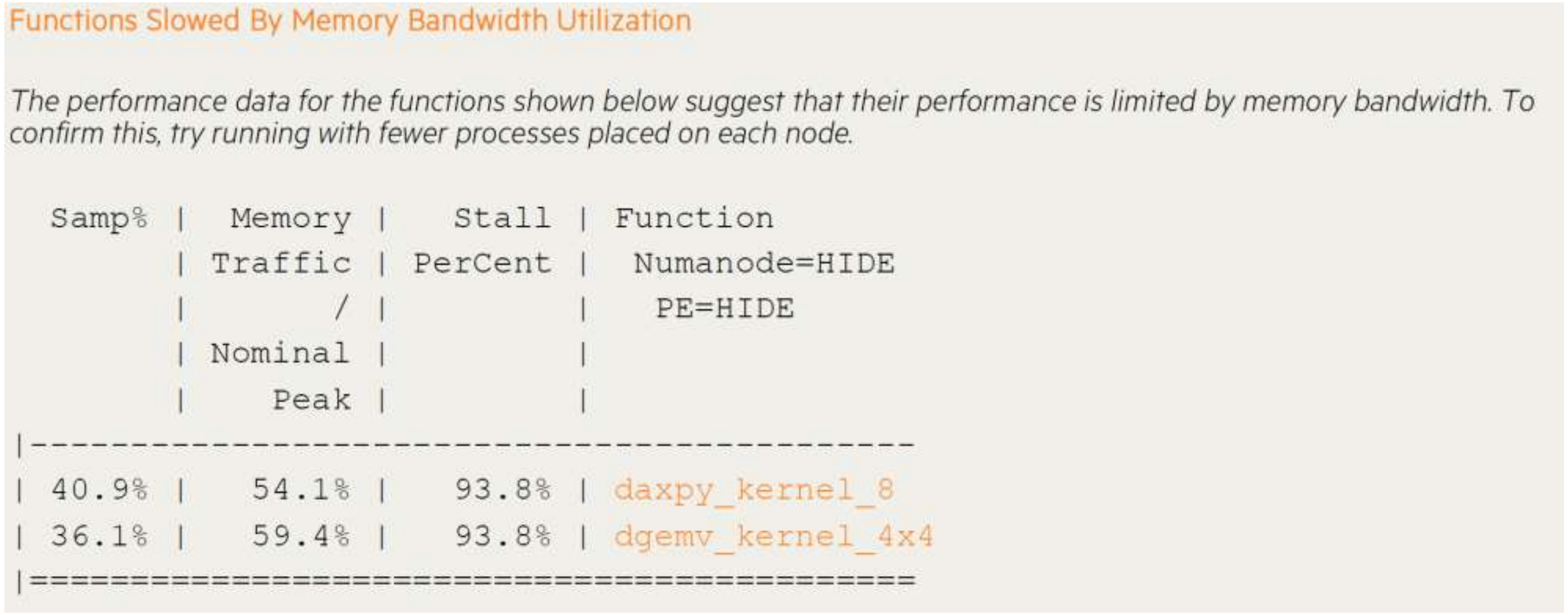
- Also try the GUI for analyzing performance analysis results.

```
$> app2 app.exe+*/
```



On-node analysis

- Memory bandwidth sensitivity guidance



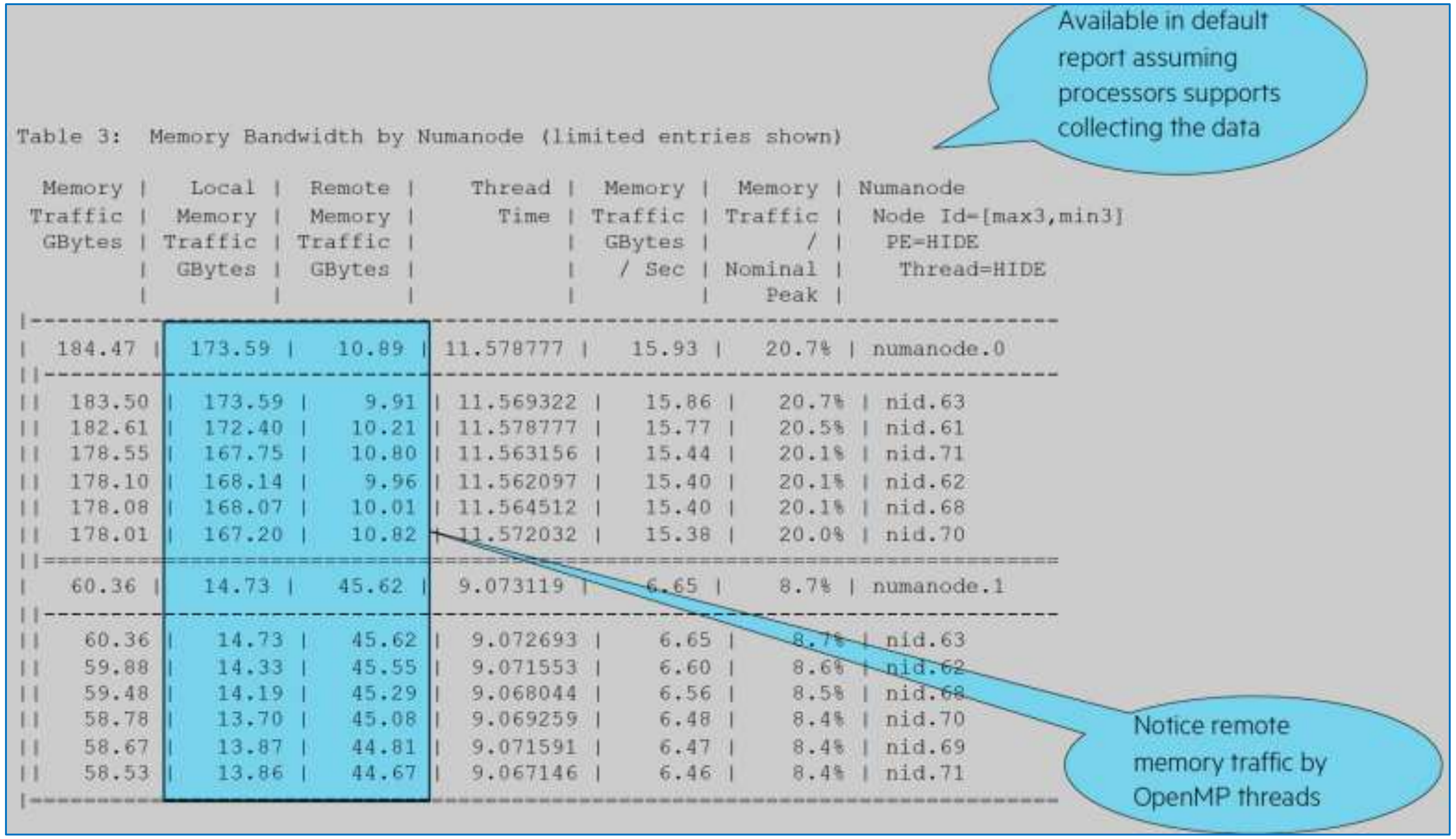
Functions Slowed By Memory Bandwidth Utilization

The performance data for the functions shown below suggest that their performance is limited by memory bandwidth. To confirm this, try running with fewer processes placed on each node.

```
  Samp% |  Memory |   Stall | Function
        | Traffic | PerCent |  Numanode=HIDE
        |       / |         |   PE=HIDE
        | Nominal |         |
        |    Peak |         |
|-------------------------------------------
| 40.9% |   54.1% |   93.8% | daxpy_kernel_8
| 36.1% |   59.4% |   93.8% | dgemv_kernel_4x4
|===========================================
```

On-node analysis

- Example traffic from an MPI+OpenMP run.

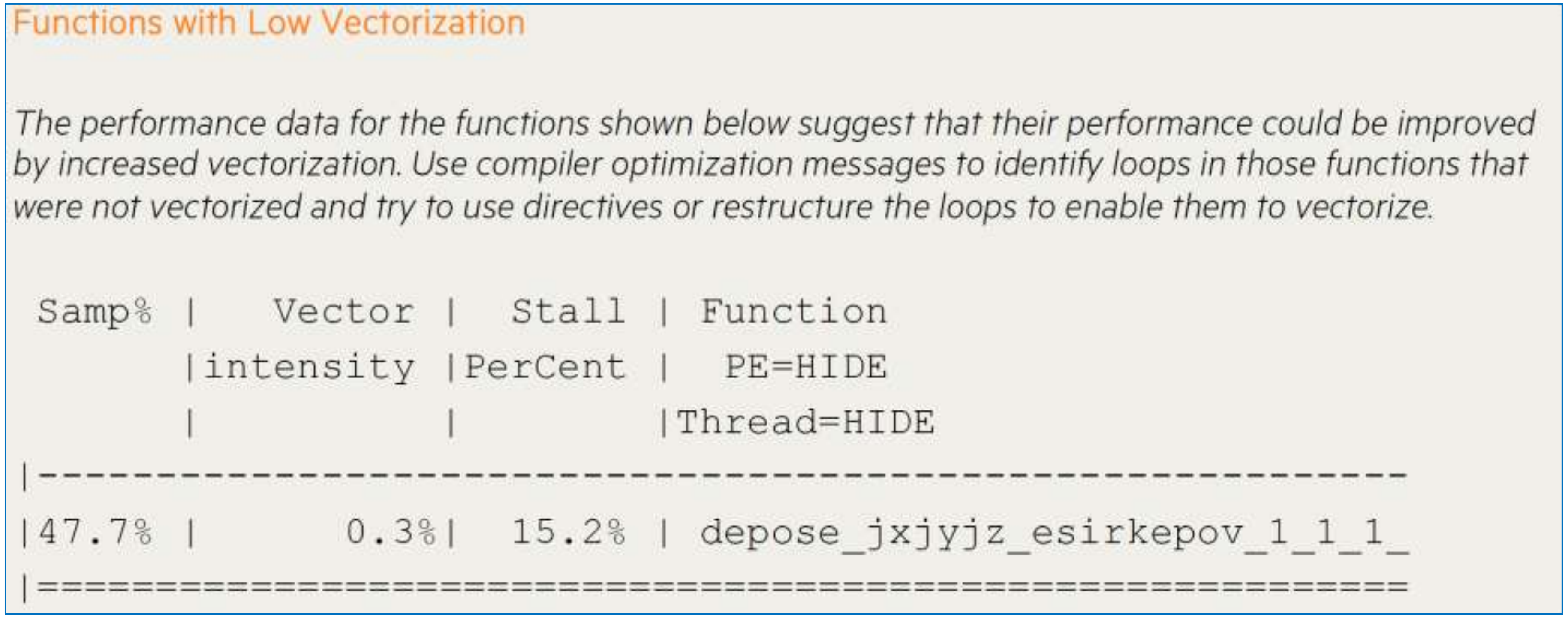

Available in default report assuming processors supports collecting the data
Notice remote memory traffic by OpenMP threads

```
Table 3:  Memory Bandwidth by Numanode (limited entries shown)

  Memory |  Local |  Remote |    Thread | Memory |  Memory | Numanode
 Traffic | Memory |  Memory |      Time | Traffic| Traffic |  Node Id=[max3,min3]
  GBytes | Traffic| Traffic |           | GBytes |       / |   PE=HIDE
         | GBytes |  GBytes |           |  / Sec | Nominal |    Thread=HIDE
         |        |         |           |        |    Peak |
|----------------------------------------------------------------------------
|  184.47 |  173.59 |   10.89 | 11.578777 |   15.93 |   20.7% | numanode.0
||---------------------------------------------------------------------------
||   183.50 |  173.59 |    9.91 | 11.569322 |   15.86 |   20.7% | nid.63
||   182.61 |  172.40 |   10.21 | 11.578777 |   15.77 |   20.5% | nid.61
||   178.55 |  167.75 |   10.80 | 11.563156 |   15.44 |   20.1% | nid.71
||   178.10 |  168.14 |    9.96 | 11.562097 |   15.40 |   20.1% | nid.62
||   178.08 |  168.07 |   10.01 | 11.564512 |   15.40 |   20.1% | nid.68
||   178.01 |  167.20 |   10.82 | 11.572032 |   15.38 |   20.0% | nid.70
||===========================================================================
|   60.36 |   14.73 |   45.62 |  9.073119 |    6.65 |    8.7% | numanode.1
||---------------------------------------------------------------------------
||    60.36 |   14.73 |   45.62 |  9.072693 |    6.65 |    8.7% | nid.63
||    59.88 |   14.33 |   45.55 |  9.071553 |    6.60 |    8.6% | nid.62
||    59.48 |   14.19 |   45.29 |  9.068044 |    6.56 |    8.5% | nid.68
||    58.78 |   13.70 |   45.08 |  9.069259 |    6.48 |    8.4% | nid.70
||    58.67 |   13.87 |   44.81 |  9.071591 |    6.47 |    8.4% | nid.69
||    58.53 |   13.86 |   44.67 |  9.067146 |    6.46 |    8.4% | nid.71
|----------------------------------------------------------------------------
```

On-node analysis

- Low vectorization guidance.



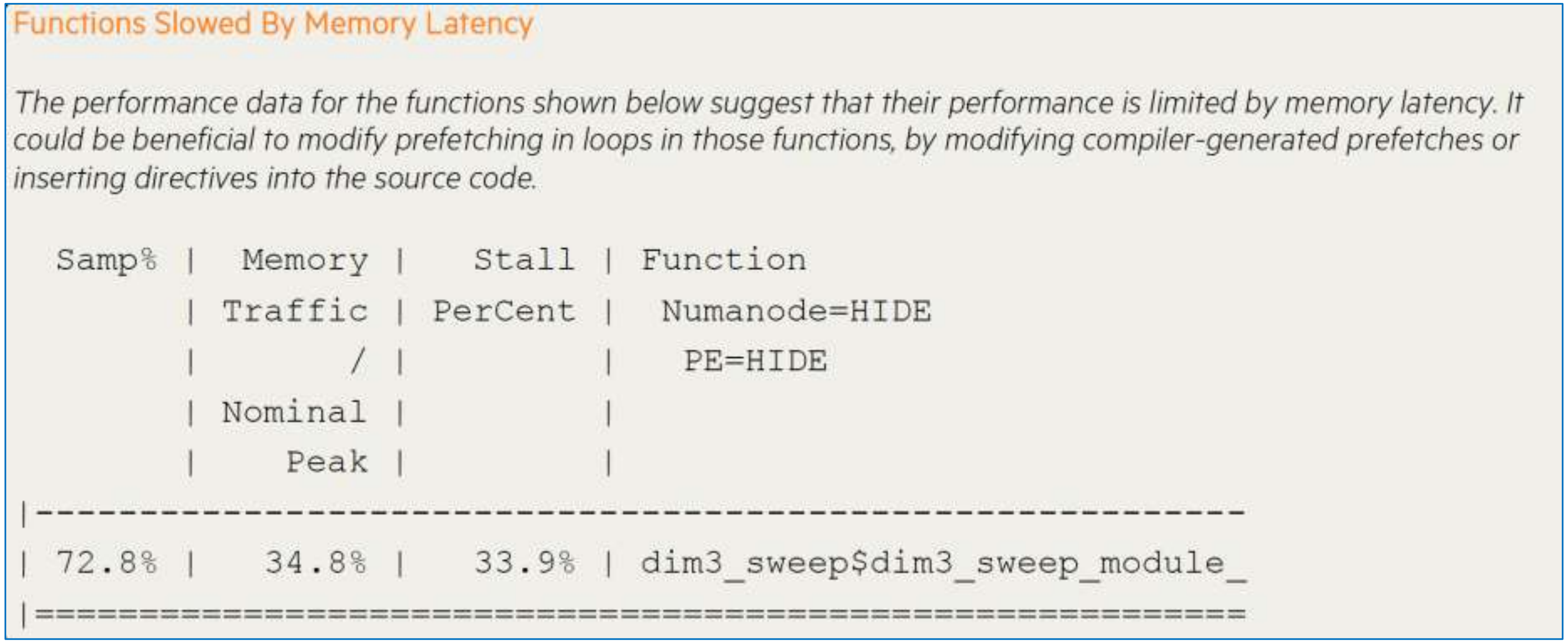
Functions with Low Vectorization

The performance data for the functions shown below suggest that their performance could be improved by increased vectorization. Use compiler optimization messages to identify loops in those functions that were not vectorized and try to use directives or restructure the loops to enable them to vectorize.

```
 Samp% |    Vector |   Stall | Function
       |intensity |PerCent |   PE=HIDE
       |          |        |Thread=HIDE
|-------------------------------------------------------
|47.7% |      0.3%|  15.2% | depose_jxjyjz_esirkepov_1_1_1_
|=======================================================
```

On-node analysis

- Memory latency sensitivity guidance.



Functions Slowed By Memory Latency

The performance data for the functions shown below suggest that their performance is limited by memory latency. It could be beneficial to modify prefetching in loops in those functions, by modifying compiler-generated prefetches or inserting directives into the source code.

```
  Samp% |   Memory |    Stall | Function
        |  Traffic | PerCent |  Numanode=HIDE
        |        / |          |   PE=HIDE
        |  Nominal |          |
        |     Peak |          |
|-----------------------------------------------------
| 72.8% |    34.8% |    33.9% | dim3_sweep$dim3_sweep_module_
|=====================================================
```

Documentation

- Module help
  - module help perftools-base
  - module help perftools
  - module help perftools-lite
  - module help perftools-lite-*
- Man pages
  - man pat_build
  - man pat_report
- Advanced help system
  - pat_help
  - pat_info [[.ap2_file] [experiment_data_directory]…]
- Web pages:
  - https://cpe.ext.hpe.com/docs/24.03/performance-tools/index.html

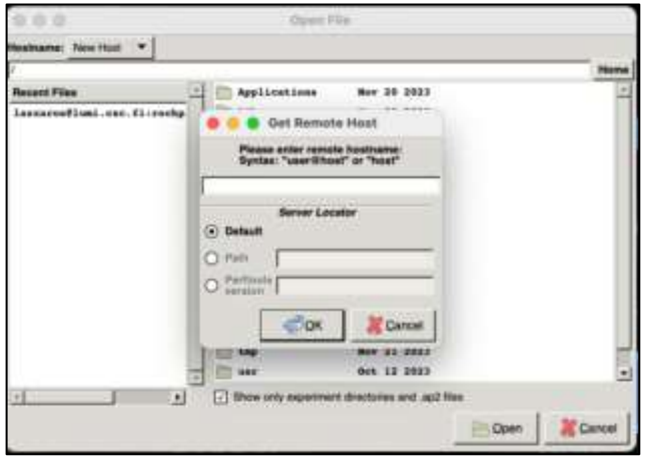
APPRENTICE2

Display your performance analysis results

Apprentice2 – Desktop installer with remote access to LUMI

- Remote visualization of the PAT data
- Installers for Windows and MacOS available at
  - /opt/cray/pe/perftools/<version>/share/desktop_installers/
  - No Linux package!
  - Note that the Windows version doesn’t support ssh key authentication
    – Cannot be used for LUMI
- MacOS application startup window:

- Open PAT data on LUMI via `File -> Open Remote…`
  - Insert the `<login>@lumi.csc.fi` with `Default` path
  - Navigate to your PAT data directories
  - Click to open the directory of the PAT experiment



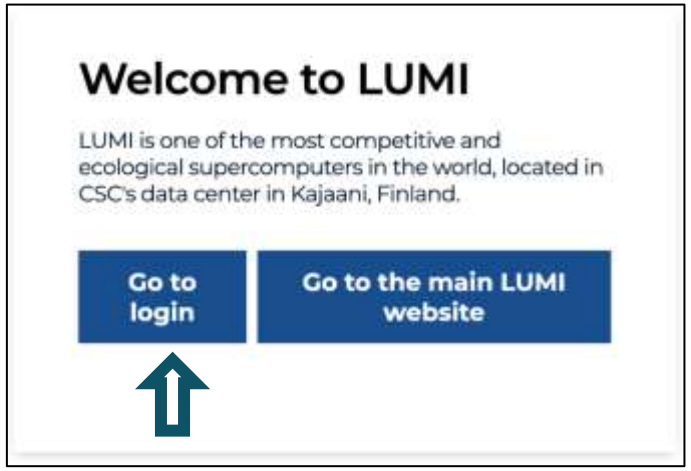
Apprentice2 – Desktop installer with copy of the PAT data

- It is possible to copy the PAT directories on your laptop/desktop
  - Depending on the experiment, it can be quite large

- It works on Windows

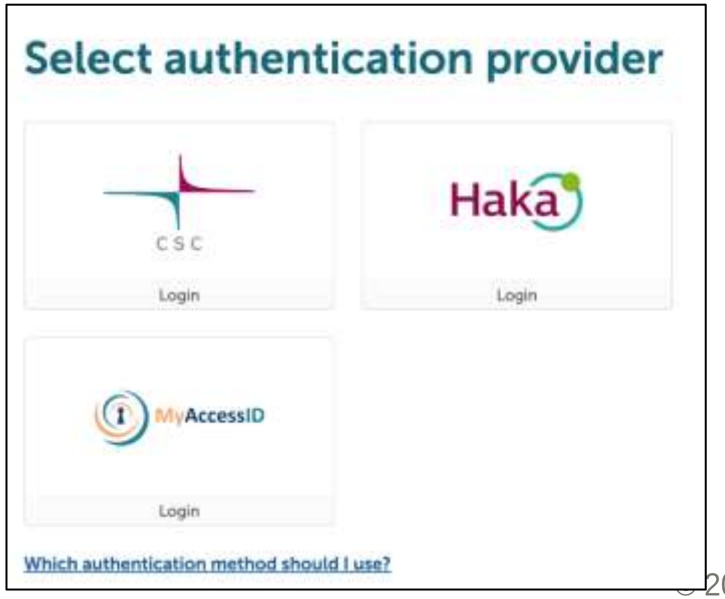
Apprentice2 – LUMI GUI export (1)

- X export from LUMI (e.g. ssh -X) possible but it can be slow
- A better solution is to use VNC from LUMI
  - More info via `module spider VNC`
- LUMI provides a new user interface (suggested solution)
  - https://www.lumi-supercomputer.eu/introducing-a-new-web-interface-for-lumi/
  - It offers Open OnDemand as a browser-based interface to launch interactive applications at https://www.lumi.csc.fi/public/
    – Follow the instructions at https://docs.lumi-supercomputer.eu/runjobs/webui/

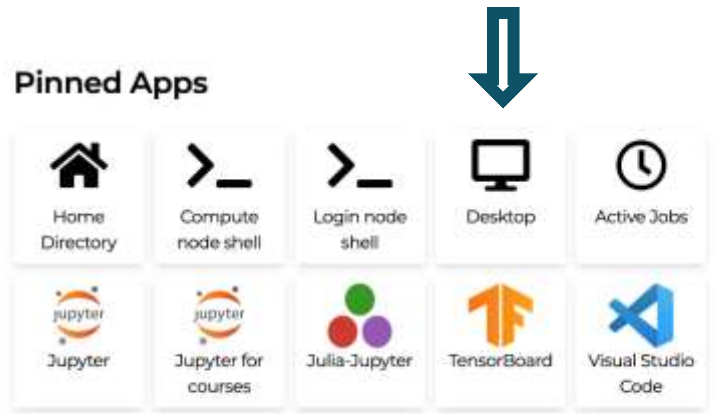


Welcome to LUMI

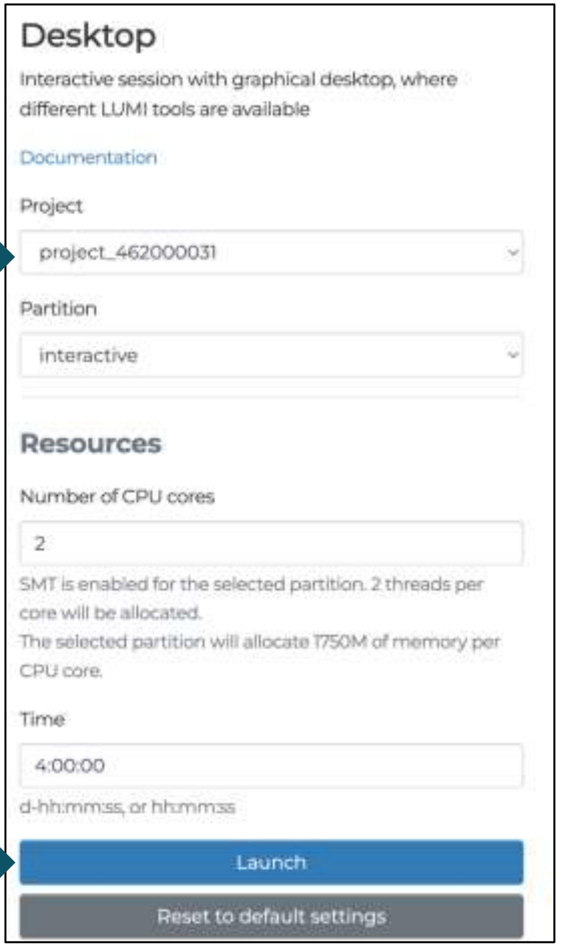
LUMI is one of the most competitive and ecological supercomputers in the world, located in CSC's data center in Kajaani, Finland.



Select authentication provider



Apprentice2 – LUMI GUI export (2)



Desktop

Interactive session with graphical desktop, where different LUMI tools are available

Documentation

Project: project_462000031

Partition: interactive

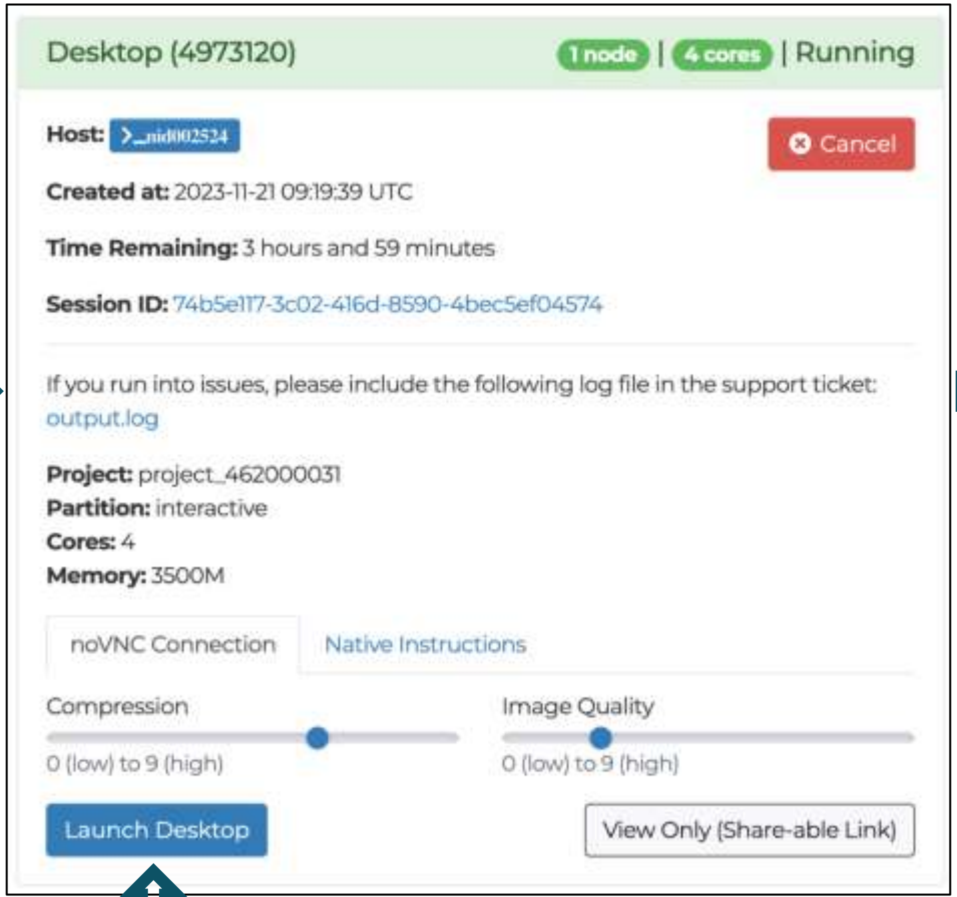
Resources

Number of CPU cores: 2

SMT is enabled for the selected partition. 2 threads per core will be allocated.
The selected partition will allocate 1750M of memory per CPU core.

Time: 4:00:00

d-hh:mm:ss, or hh:mm:ss



Desktop (4973120) | 1 node | 4 cores | Running

Host: >_nid002524

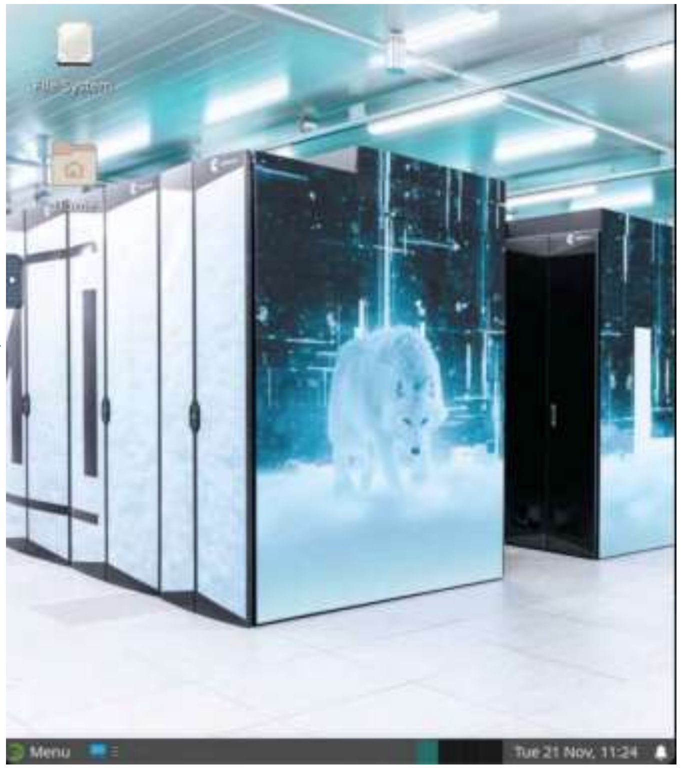
Created at: 2023-11-21 09:19:39 UTC

Time Remaining: 3 hours and 59 minutes

Session ID: 74b5e117-3c02-416d-8590-4bec5ef04574

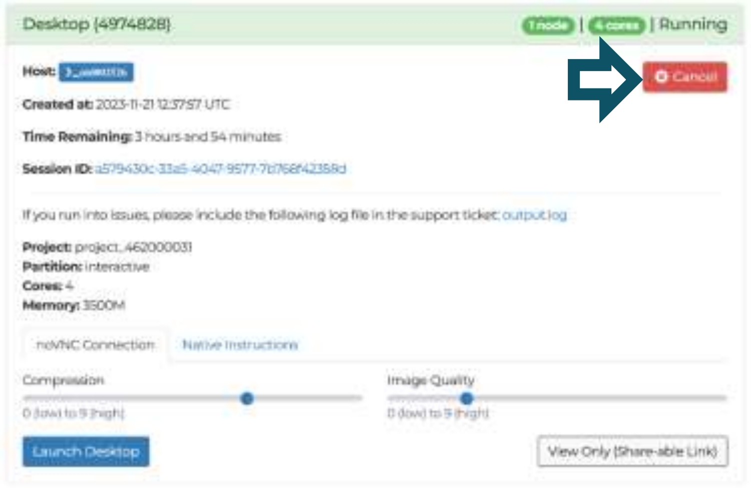
If you run into issues, please include the following log file in the support ticket: output.log

Project: project_462000031
Partition: interactive
Cores: 4
Memory: 3500M



Apprentice2 – LUMI GUI export (3)

- Open a terminal and run
  - `module load perftools-base`
  - `app2 &`
- Now you can use Apprentice2 directly on LUMI
  - Open the PAT directory and analyse the data
- Also, you can use the app2 CLI, e.g.
  - Open a directory: `app2 <experiment_data_directory> &`
  - Compare two experiments: `app2 --compare <experiment_data_directory1> <experiment_data_directory2> &`
  - More info via `man app2`
- When you have finished, remember to cancel (and delete) the remote Desktop allocation


Desktop (4974828) | Running

Host: [illegible]

Created at: 2023-11-21 12:37:57 UTC

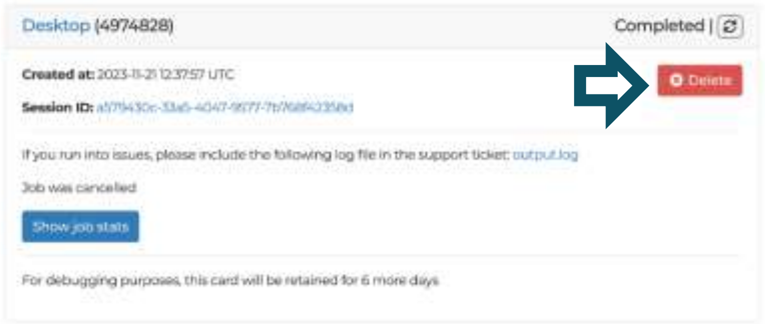
Time Remaining: 3 hours and 54 minutes

Session ID: a579430c-33a5-4047-9577-7b768f42358d

If you run into issues, please include the following log file in the support ticket: output.log

Project: project_462000031
Partition: interactive
Cores: 4
Memory: 3500M

Compression: 0 (low) to 9 (high)
Image Quality: 0 (low) to 9 (high)


Desktop (4974828) Completed |

Created at: 2023-11-21 12:37:57 UTC

Session ID: a579430c-33a5-4047-9577-7b768f42358d

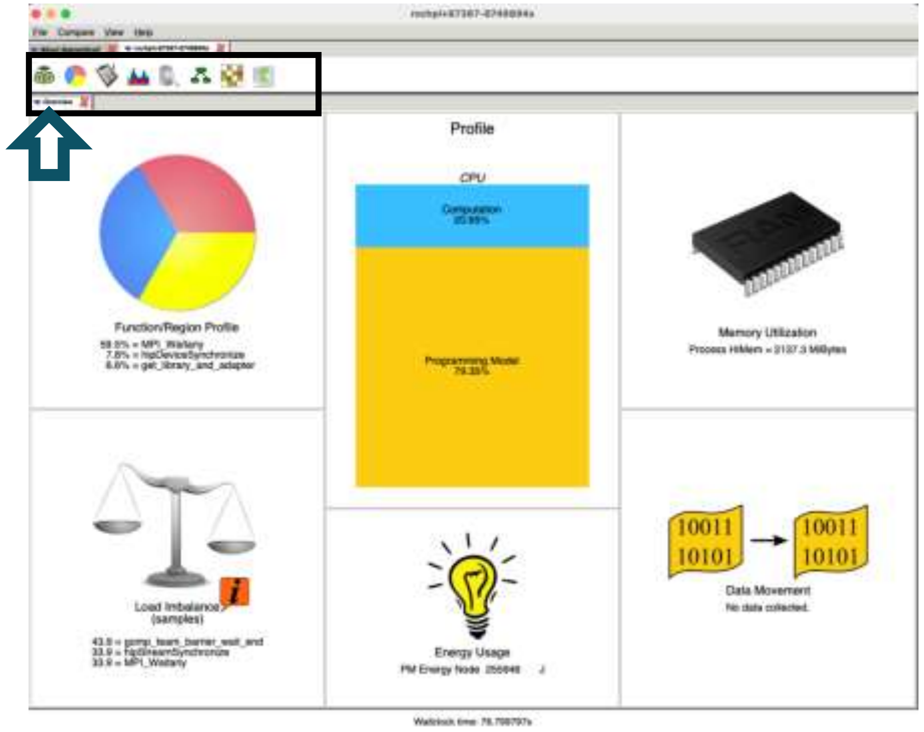
If you run into issues, please include the following log file in the support ticket: output.log

Job was cancelled

For debugging purposes, this card will be retained for 6 more days

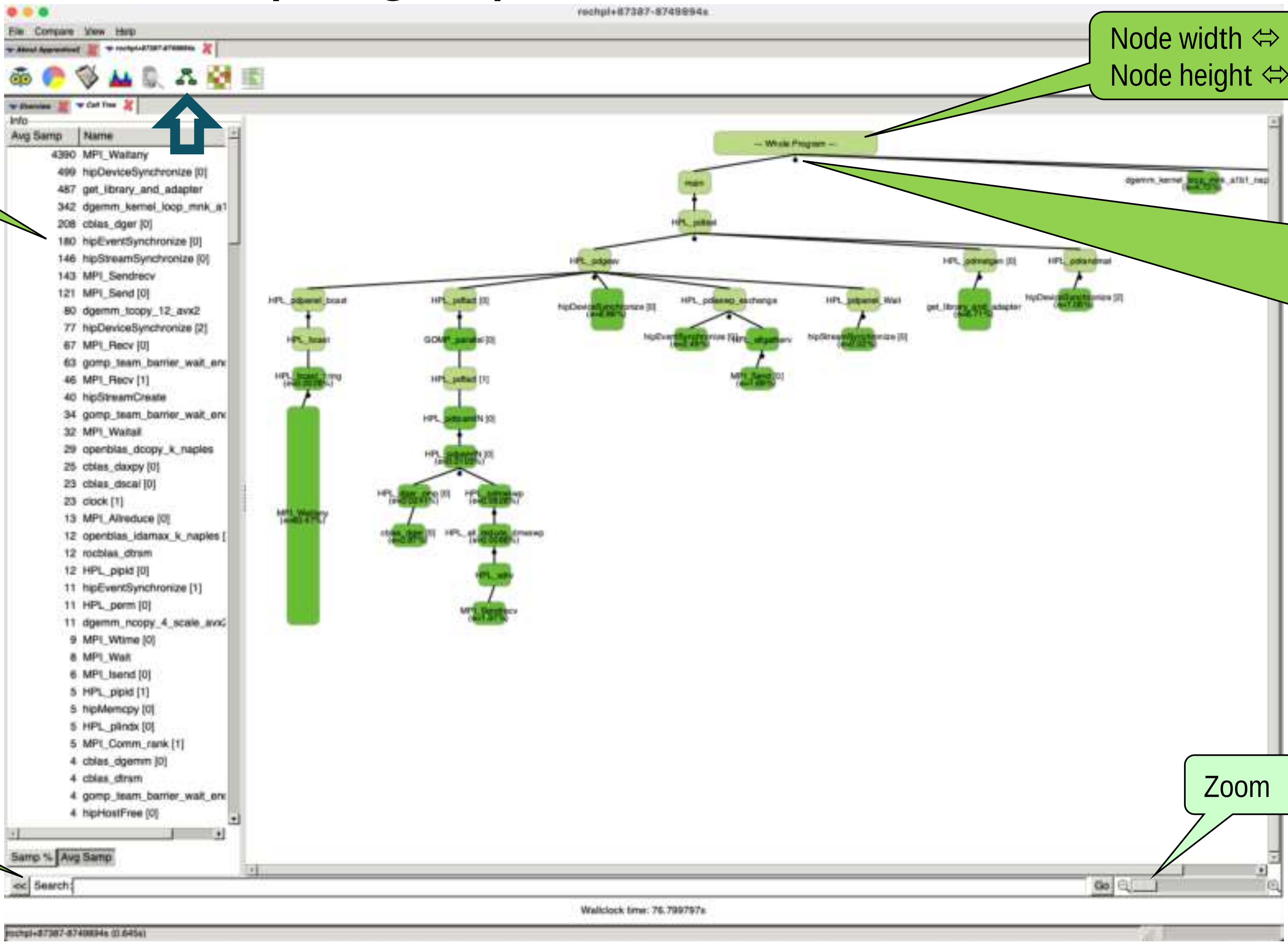
Visualize profiling results

- Can use the Apprentice2 application:
  - `app2 main.x+*/ &`
- The actual information contained in the resulting data file will vary depending on your choice of `pat_build` options and runtime environment variables
  - Need to run pat_report on the experiment directory (it produces the ap2 summary files)



- `Help ->Panel Help` opens a PDF with the documentation on Apprentice2

Call tree view – Sampling experiment



Avg Samp	Name
4390	MPI_Waitany
499	hipDeviceSynchronize [0]
487	get_library_and_adapter
342	dgemm_kernel_loop_mnk_a1
208	cblas_dger [0]
180	hipEventSynchronize [0]
146	hipStreamSynchronize [0]
143	MPI_Sendrecv
121	MPI_Send [0]
80	dgemm_tcopy_12_avx2
77	hipDeviceSynchronize [2]
67	MPI_Recv [0]
63	gomp_team_barrier_wait_en
46	MPI_Recv [1]
40	hipStreamCreate
34	gomp_team_barrier_wait_en
32	MPI_Waitall
29	openblas_dcopy_k_naples
25	cblas_daxpy [0]
23	cblas_dscal [0]
23	clock [1]
13	MPI_Allreduce [0]
12	openblas_idamax_k_naples [
12	rocblas_dtrsm
12	HPL_pipid [0]
11	hipEventSynchronize [1]
11	HPL_perm [0]
11	dgemm_ncopy_4_scale_avx2
9	MPI_Wtime [0]
8	MPI_Wait
6	MPI_Isend [0]
5	HPL_pipid [1]
5	hipMemcpy [0]
5	HPL_plindx [0]
5	MPI_Comm_rank [1]
4	cblas_dgemm [0]
4	cblas_dtrsm
4	gomp_team_barrier_wait_en
4	hipHostFree [0]

Function List

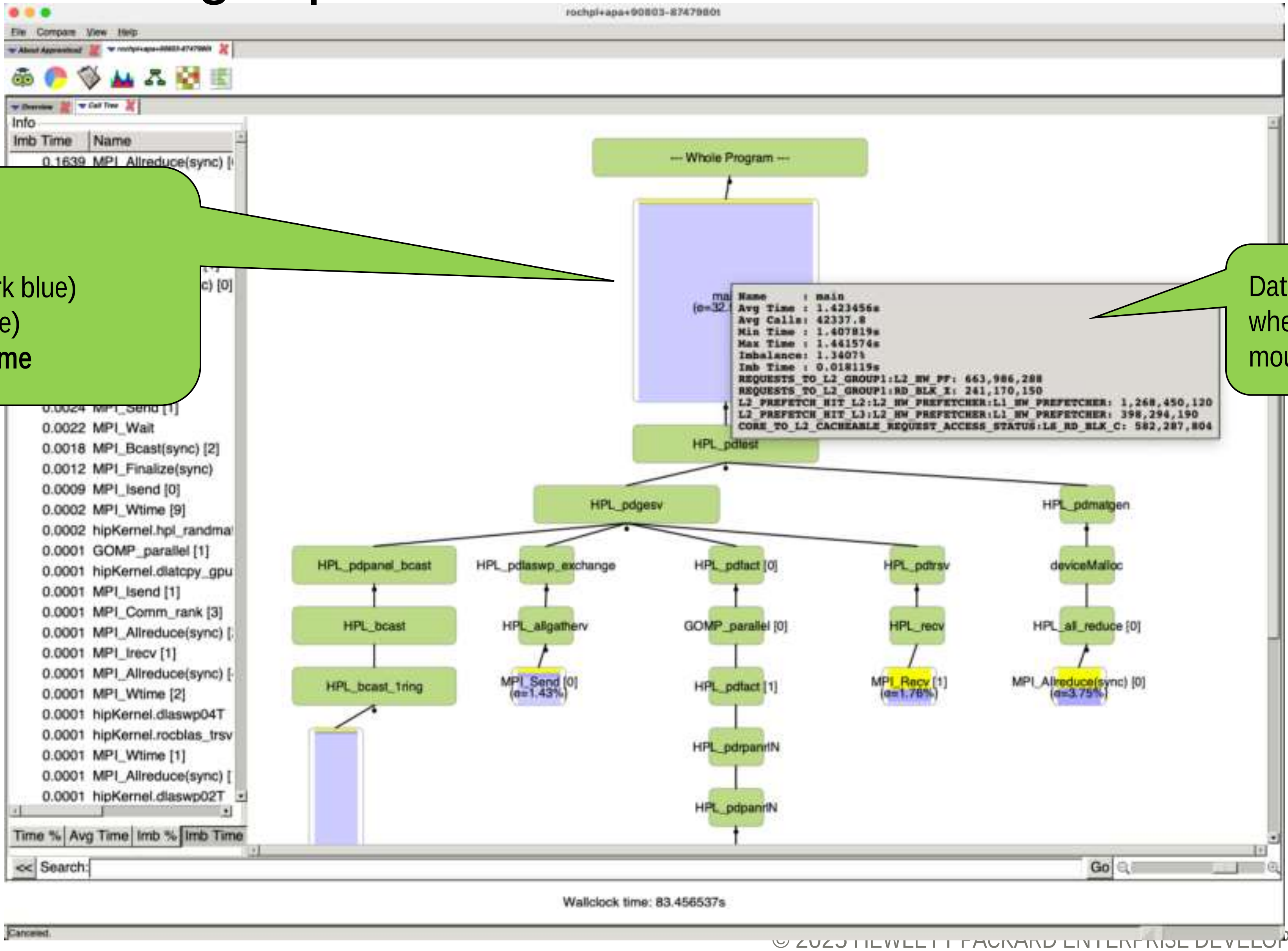
Node width ⇔ inclusive time
Node height ⇔ exclusive time

Filtered node or sub tree.
Right mouse click to get Node menu, e.g. hide/unhide children

Function List display toggle

Zoom

Call tree view – Tracing experiment



Imb Time	Name
0.1639	MPI_Allreduce(sync) [
	c) [0]
0.0024	MPI_Send [1]
0.0022	MPI_Wait
0.0018	MPI_Bcast(sync) [2]
0.0012	MPI_Finalize(sync)
0.0009	MPI_Isend [0]
0.0002	MPI_Wtime [9]
0.0002	hipKernel.hpl_randmat
0.0001	GOMP_parallel [1]
0.0001	hipKernel.dlatcpy_gpu
0.0001	MPI_Isend [1]
0.0001	MPI_Comm_rank [3]
0.0001	MPI_Allreduce(sync) [
0.0001	MPI_Irecv [1]
0.0001	MPI_Allreduce(sync) [
0.0001	MPI_Wtime [2]
0.0001	hipKernel.dlaswp04T
0.0001	hipKernel.rocblas_trsv
0.0001	MPI_Wtime [1]
0.0001	MPI_Allreduce(sync) [
0.0001	hipKernel.dlaswp02T

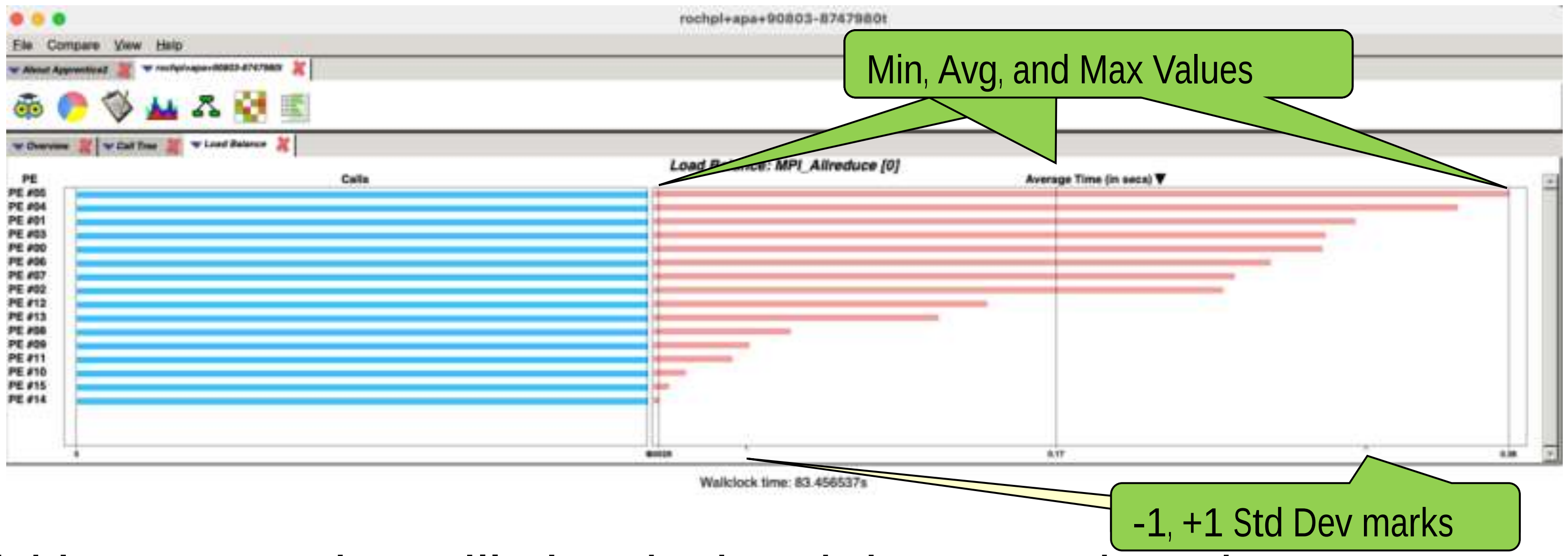
```
main
(e=32.
Name      : main
Avg Time  : 1.423456s
Avg Calls: 42337.8
Min Time  : 1.407819s
Max Time  : 1.441574s
Imbalance: 1.3407%
Imb Time  : 0.018119s
REQUESTS_TO_L2_GROUP1:L2_HW_PF: 663,986,288
REQUESTS_TO_L2_GROUP1:RD_BLK_X: 241,170,150
L2_PREFETCH_HIT_L2:L2_HW_PREFETCHER:L1_HW_PREFETCHER: 1,268,450,120
L2_PREFETCH_HIT_L3:L2_HW_PREFETCHER:L1_HW_PREFETCHER: 398,294,190
CORE_TO_L2_CACHEABLE_REQUEST_ACCESS_STATUS:LS_RD_BLK_C: 582,287,804
```

Load balance overview:
Height ⇔ Max time
Middle bar ⇔ Average time (dark blue)
Lower bar ⇔ Min time (light blue)
Yellow represents imbalance time

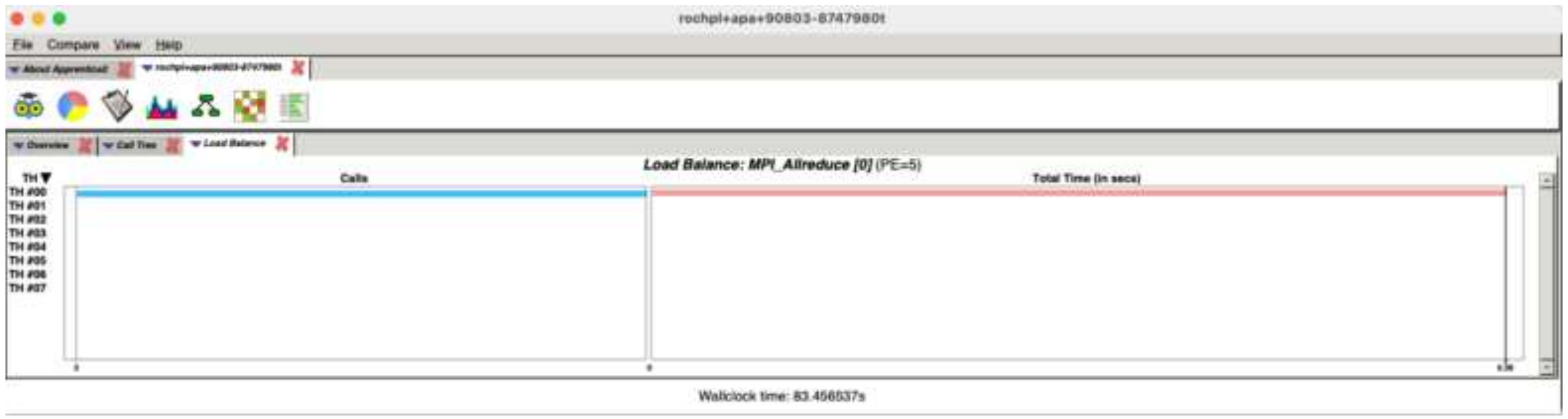
Data displayed when hovering the mouse over nodes

Load Balance

- Clicking on a node of the Call tree view will show the breakdown per each PE, reporting the load balance

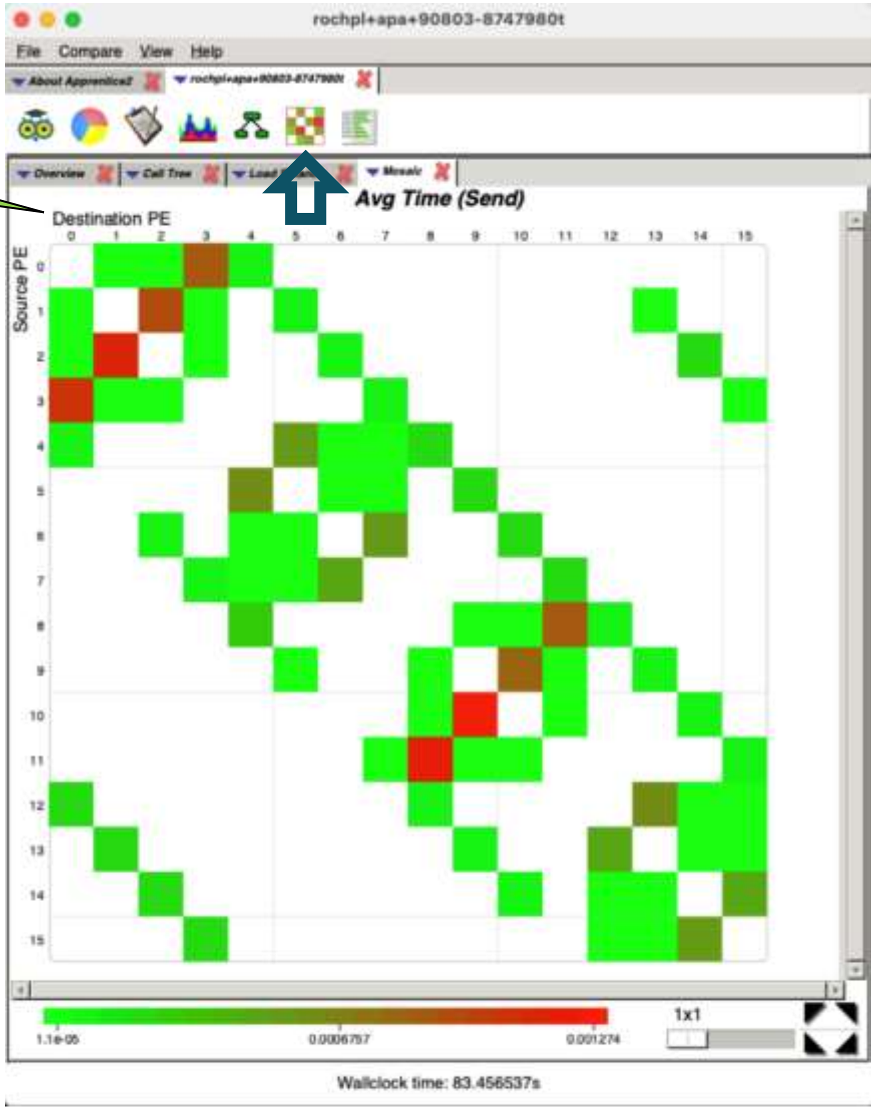


- Further clicking on a PE bar will give the breakdown per threads


rochpl+apa+90803-8747980t
File Compare View Help
Overview Call Tree Load Balance
Load Balance: MPI_Allreduce [0] (PE=5)
TH
Calls
Total Time (in secs)
TH #00
TH #01
TH #02
TH #03
TH #04
TH #05
TH #06
TH #07
Wallclock time: 83.456537s

Mosaic View

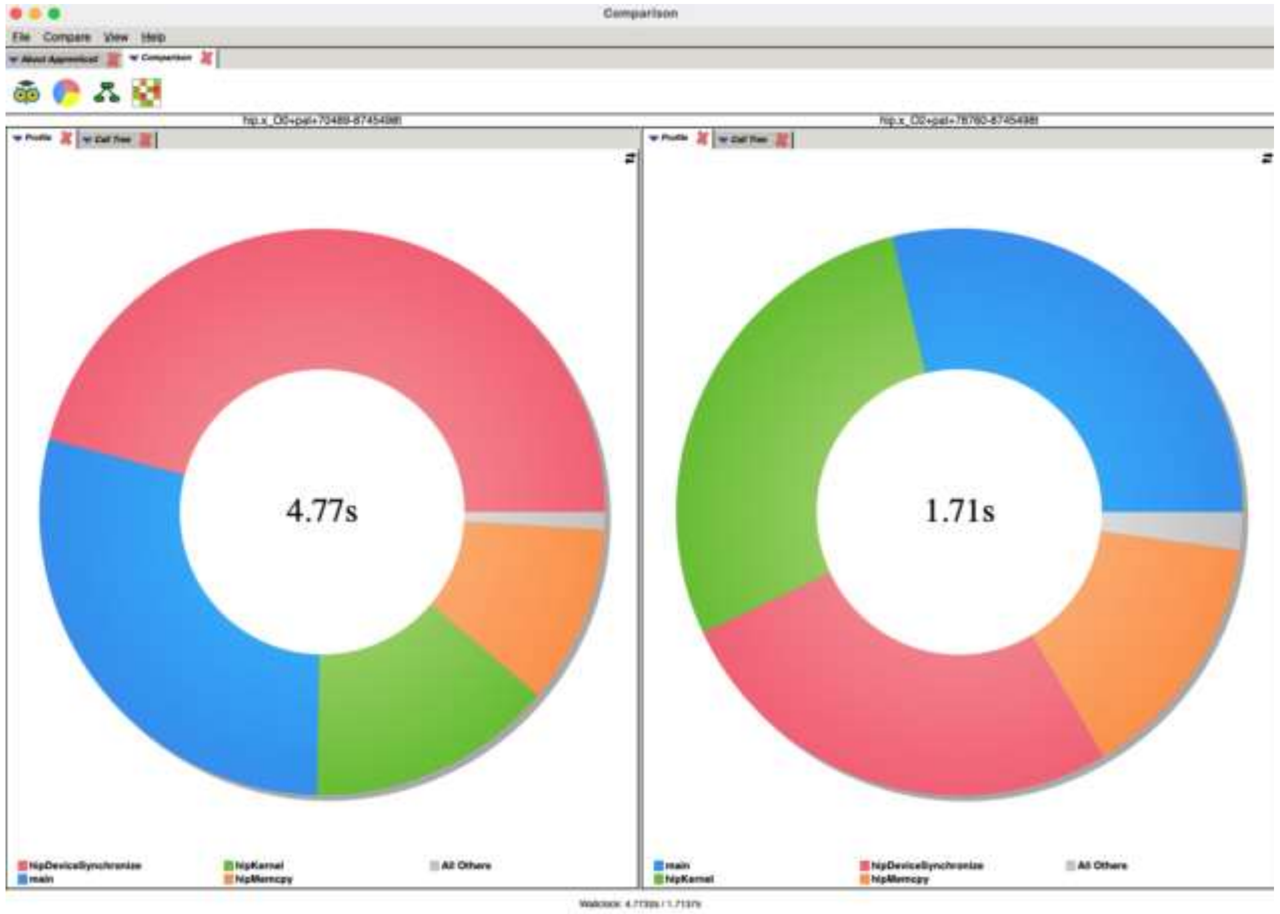
Send/receive of data, useful to check communication patterns



Experiments comparison

- Compare 2 experiments in a side-by-side display
  - Assuming they are the same e experiment and summary types
  - Useful during an application optimization phase
  - Can use CLI or the GUI menu `Compare`

- Check also the `pat_view` utility to compare performance experiments
  - `man pat_view`
  - Can generate pdf or csv outputs

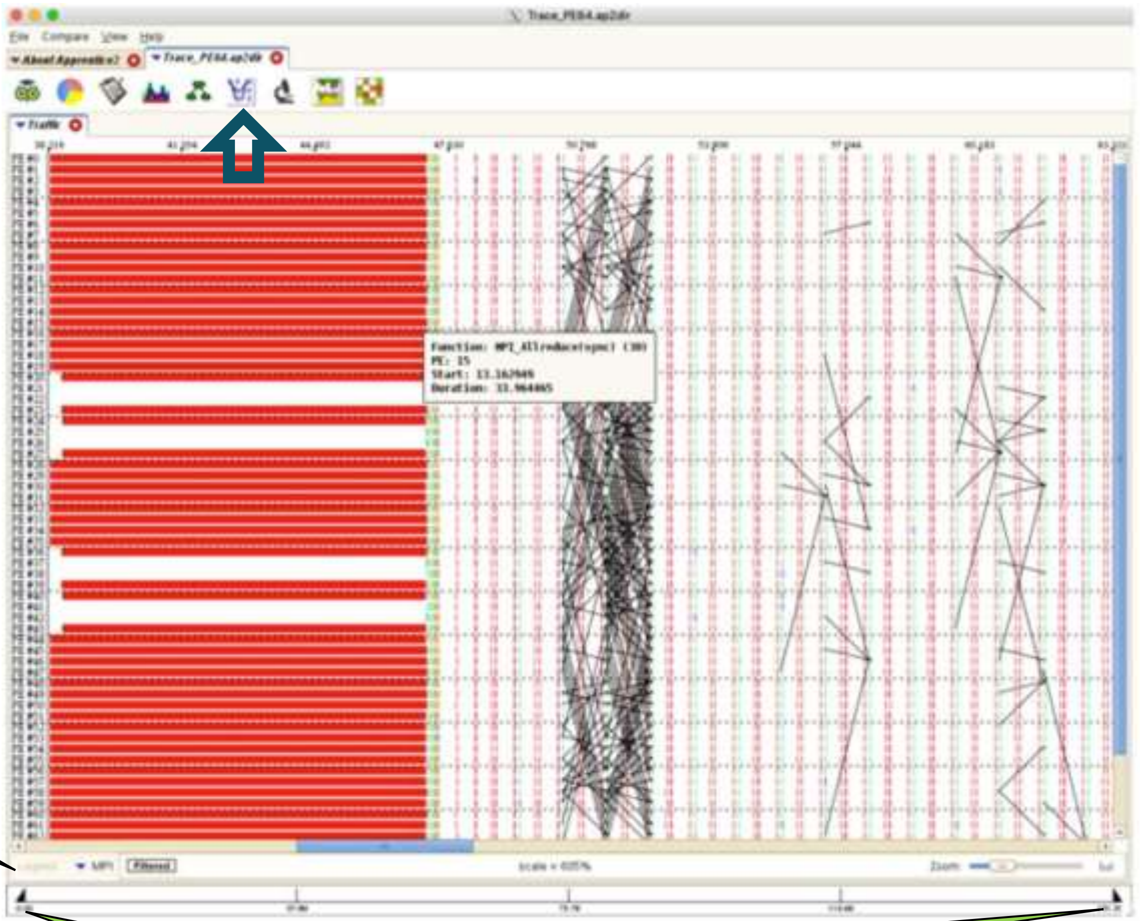


Time Line View

- By default, perftools aggregates data over time
- Use `PAT_RT_SUMMARY=0` to turn off aggregation
  - This will give you data as a function of time in Apprentice2
  - Do not run large experiments with this setting because size of data files!
- It shows:
  - Traffic time line
  - GPU time line with concurrent activity on the CPU (host) and GPU (accelerator)

Traffic Time Line View

- It shows internal PE-to-PE traffic and I/O activity by PE over time
  - Quite dense and typically requires zooming in or filtering to reveal meaningful data



```
Function: MPI_Allreduce(sync) (38)
PE: 15
Start: 13.162946
Duration: 11.964465
```

Legend

Can change time interval of analysis

GPU Time Line View

Hover the cursor over a box to see a tooltip that describes the event.
Colors of the boxes have different meaning:

- Green boxes are transfer CPU<->GPU
- Red boxes are wait-related functions
- Gray boxes are GPU kernels
- Blue boxes are for all the rest

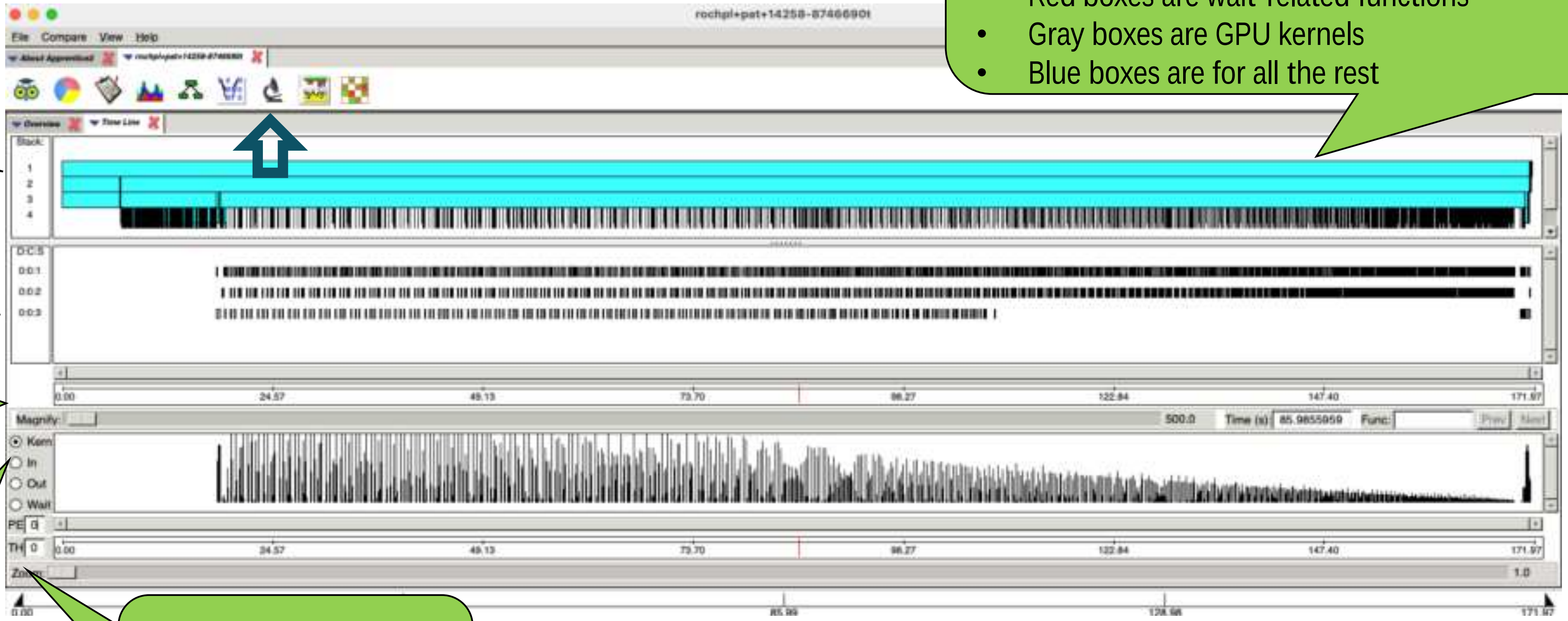
CPU stack, starting with 1 (main) at the top

GPU D:C:S (Device:Context:Stream)

Related time line and magnification controls

Histogram of the GPU activities (Kernel, In, Out, Wait) in percentage (full height of the window representing 100%)

Select data of a PE and thread (both set to zero by default)



Questions?