Hewlett Packard
Enterprise

INTRODUCTION TO PERFTOOLS

HPE/Cray tools

June 10, 2024

AGENDA

- Pat_run and pat_report
- Apprentice2
- Perftools-lite
- Perftools
- Perftools API
- Load balance analysis
- Perftools Timeline
- Hardware performance counters
- Performance analysis on GPUs
- GPU timeline
- Reveal

COURSE MATERIALS

- /home/ta159/ta159/shared/HPE_June10_demos directory
- /home/ta159/ta159/shared/HPE_June10_perftools slides
- /home/ta159/ta159/shared/HPE_June10_debug slides

PERFTOOLS LANDSCAPE

perftools-base

perftools-lite

perftools-lite-events

perftools-lite-loops

pat_run

(*-hbm, *-gpu)

For novices or convenience

pat_build
pat_report
API

For in-depth performance analysis. Includes everything from `lite`.

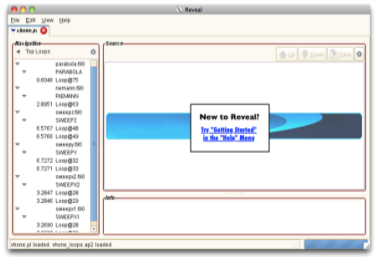
perftools

grid_order

MPI rank order list generator

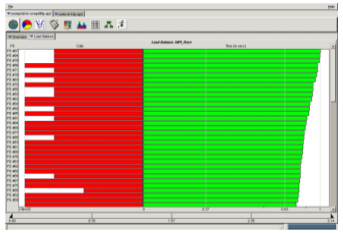
pat_help
pat_info

Advanced help system



Reveal

Apprentice2



Analysis, visualization, and parallelization.

PAT_RUN AND PAT_REPORT

Launch a dynamically-linked program instrumented for performance analysis

DEMO 1A_PAT_RUN_PS
1B_PAT_RUN_MPIPI

PROFILE EXISTING BINARIES

```
$> srun -n 16 pat_run ./app.exe
$> pat_report app.exe+*/ > my_report
```

- Insert `pat_run` before executable in run command. No instrumentation needed.
- Useful if source code is not accessible but program is dynamically linked

```
$> export PAT_RT_PERFCTR=1
$> srun -n 8 pat_run ./app.exe
```

- Use existing perftools capability.
- Optionally collect a different group of performance counters.

```
$> pat_report -P -O callers+src app.exe+*/ > my_callers_report
```

- Create additional views of the data with `pat_report` options.
- If at least object files and libraries are available, load the `perftools-preload` module before relinking. It provides more information like line number hot spots.

PERFTOOLS FOR PYTHON (EXPERIMENTAL SUPPORT)

```
$> module load cray-python
$> srun -n 4 pat_run `which python` hello.py
```

- Load the cray-python module.
- Use `pat_run` and the absolute path to python.

```
$> pat_report -o myrep <exp-dir>
$> pat_report -O ct+src -o myrep.ct <exp-dir>
```

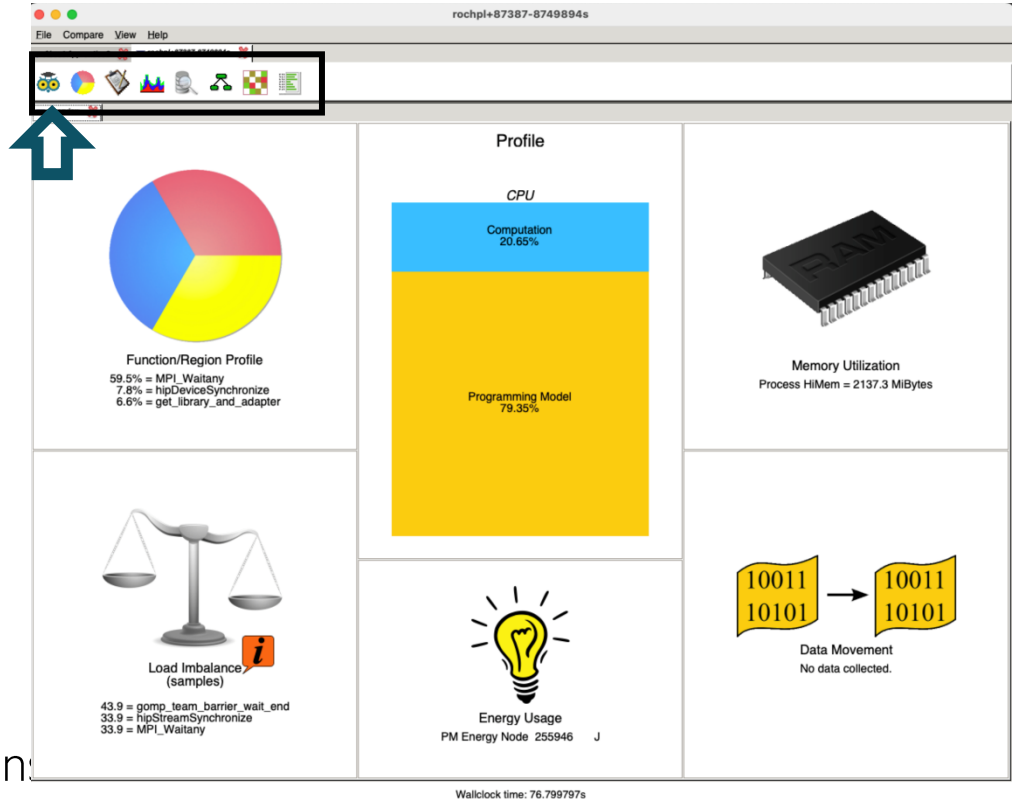
- Generate call tree report in addition to default one.
- Python module must be loaded when `pat_report` is invocated.
- Python methods from the source code are prepended with `python.*`
- Check the `pat_run` man page for more details.

```
Table 1:  Calltree View with Callsite Line Numbers

  Samp% |  Samp | Calltree
        |       |  PE=HIDE

 100.0% | 323.5 | Total
|-----------------------------------------------------------------
|  32.7% | 105.8 | read
|  24.2% |  78.2 | python.main:heat-p2p.py:line.125
||----------------------------------------------------------------
||  18.7% |  60.5 | python.iterate:heat-p2p.py:line.82
|||---------------------------------------------------------------
3||   7.7% |  25.0 | python.evolve:heat-p2p.py:line.51
||||--------------------------------------------------------------
4|||   5.8% |  18.8 | python.evolve:heat-p2p.py:line.51(exclusive)
4|||   1.8% |   5.8 | __memcpy_avx_unaligned_erms
||||==============================================================
3||   5.9% |  19.0 | python.evolve:heat-p2p.py:line.53
||||--------------------------------------------------------------
4|||   4.6% |  15.0 | python.evolve:heat-p2p.py:line.53(exclusive)
4|||   1.2% |   4.0 | __memcpy_avx_unaligned_erms
||||==============================================================
3||   3.6% |  11.5 | python.evolve:heat-p2p.py:line.50
||||--------------------------------------------------------------
4|||   2.5% |   8.0 | __memcpy_avx_unaligned_erms
4|||   1.1% |   3.5 | python.evolve:heat-p2p.py:line.50(exclusive)
||||==============================================================
3||   1.5% |   5.0 | python.evolve:heat-p2p.py:line.55
4||       |       |  __memcpy_avx_unaligned_erms
|||==============================================================
||   5.5% |  17.8 | python.iterate:heat-p2p.py:line.81
|||---------------------------------------------------------------
3||   3.9% |  12.5 | python.exchange:heat-p2p.py:line.73
4||       |       |  MPI_Sendrecv
3||   1.6% |   5.2 | python.exchange:heat-p2p.py:line.77
4||       |       |  MPI_Sendrecv
||==============================================================
```

APPRENTICE2

APPRENTICE2

- 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)

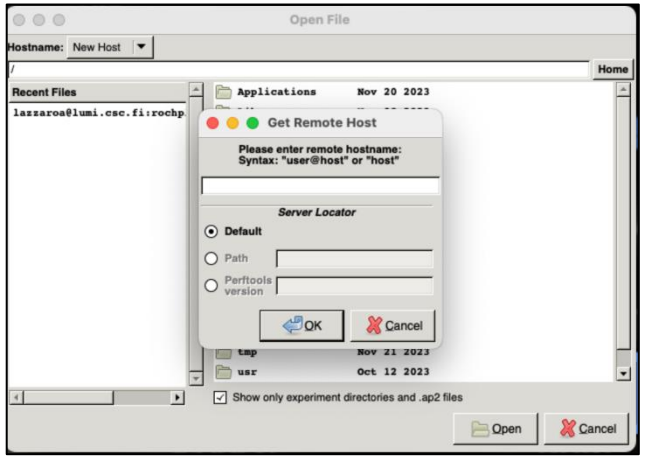


- `Help ->Panel Help` open

APPRENTICE2 – DESKTOP INSTALLER WITH REMOTE ACCESS

- Remote visualisation 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
- MacOS application startup window:

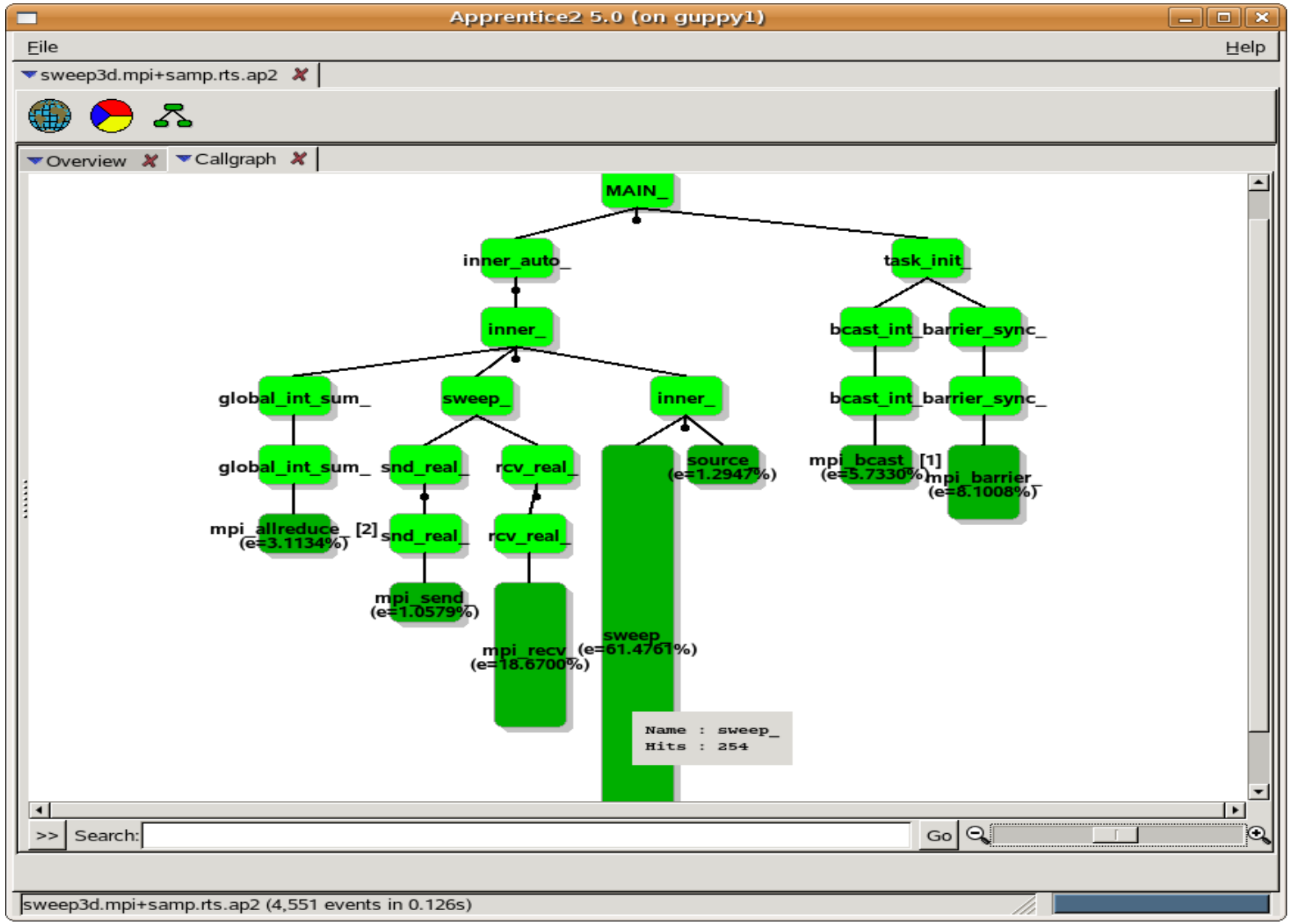
- Open PAT data via `File -> Open Remote…`
  - Insert the `<login>` with `Default` path
  - Navigate to your PAT data directories
  - NB does not work on ARCHER2 (due to ssh-key)



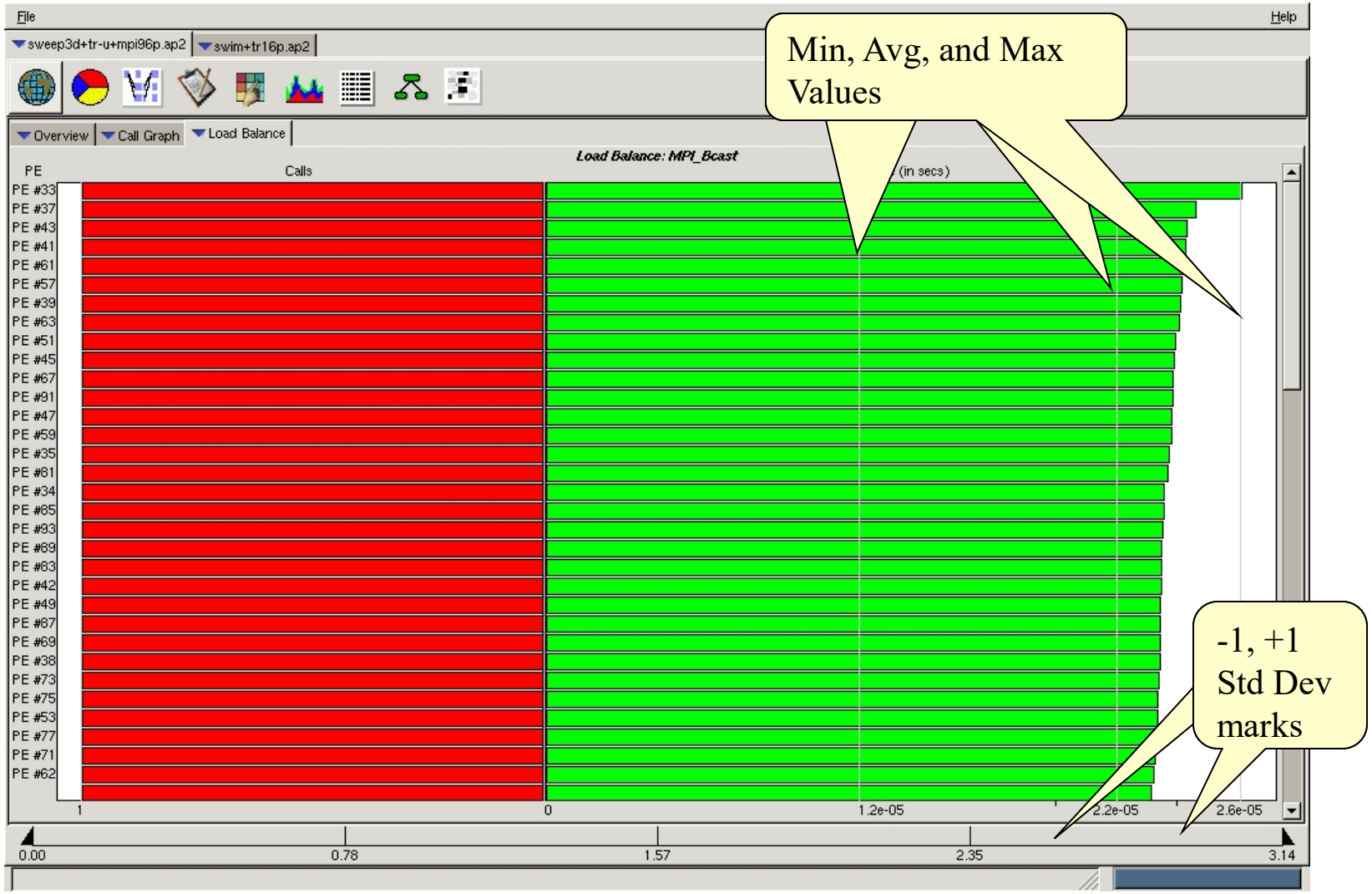
APPRENTICE2 – LOCAL DESKTOP WITH COPY OF THE PAT DATA

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

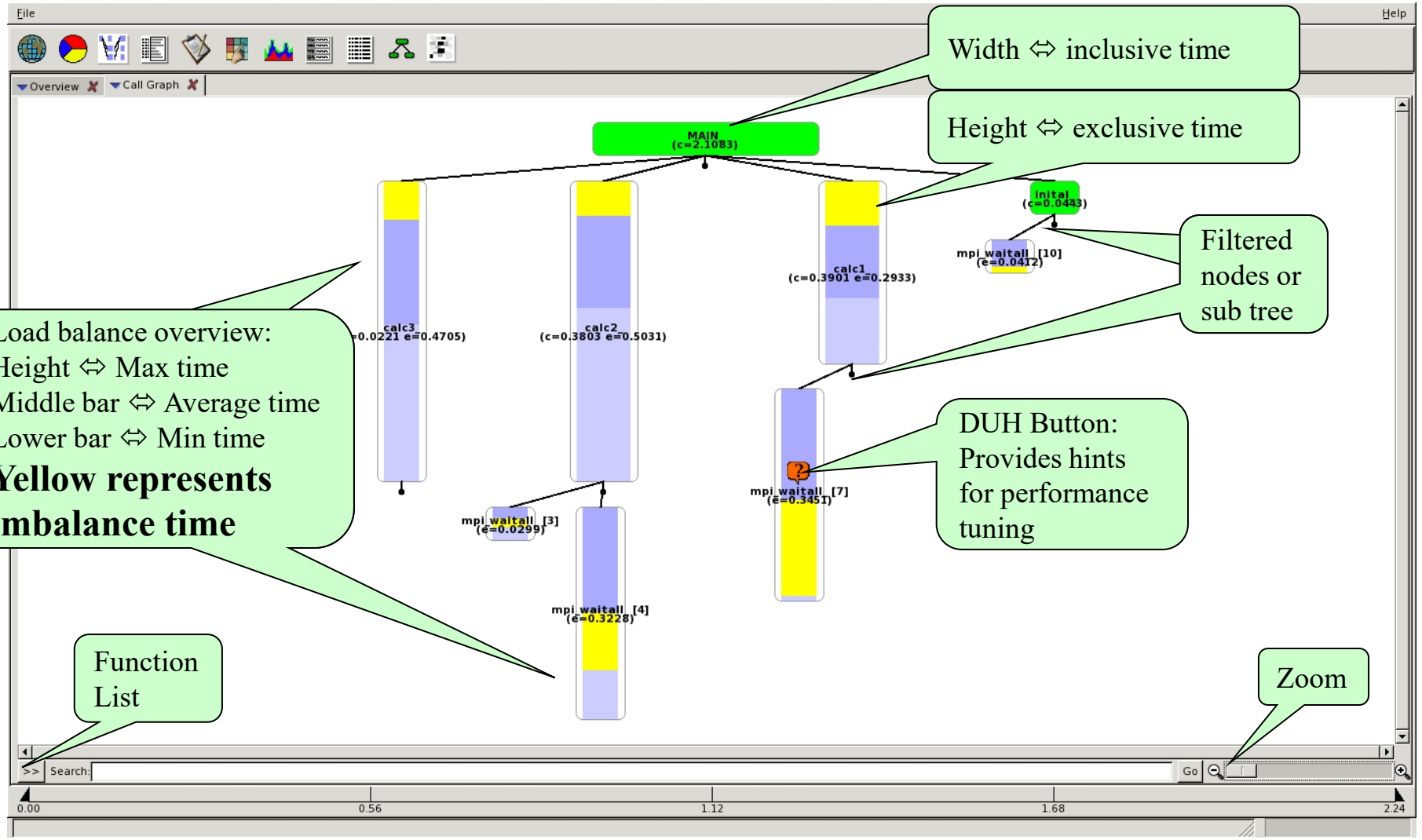
CALL TREE VIEW OF SAMPLED DATA



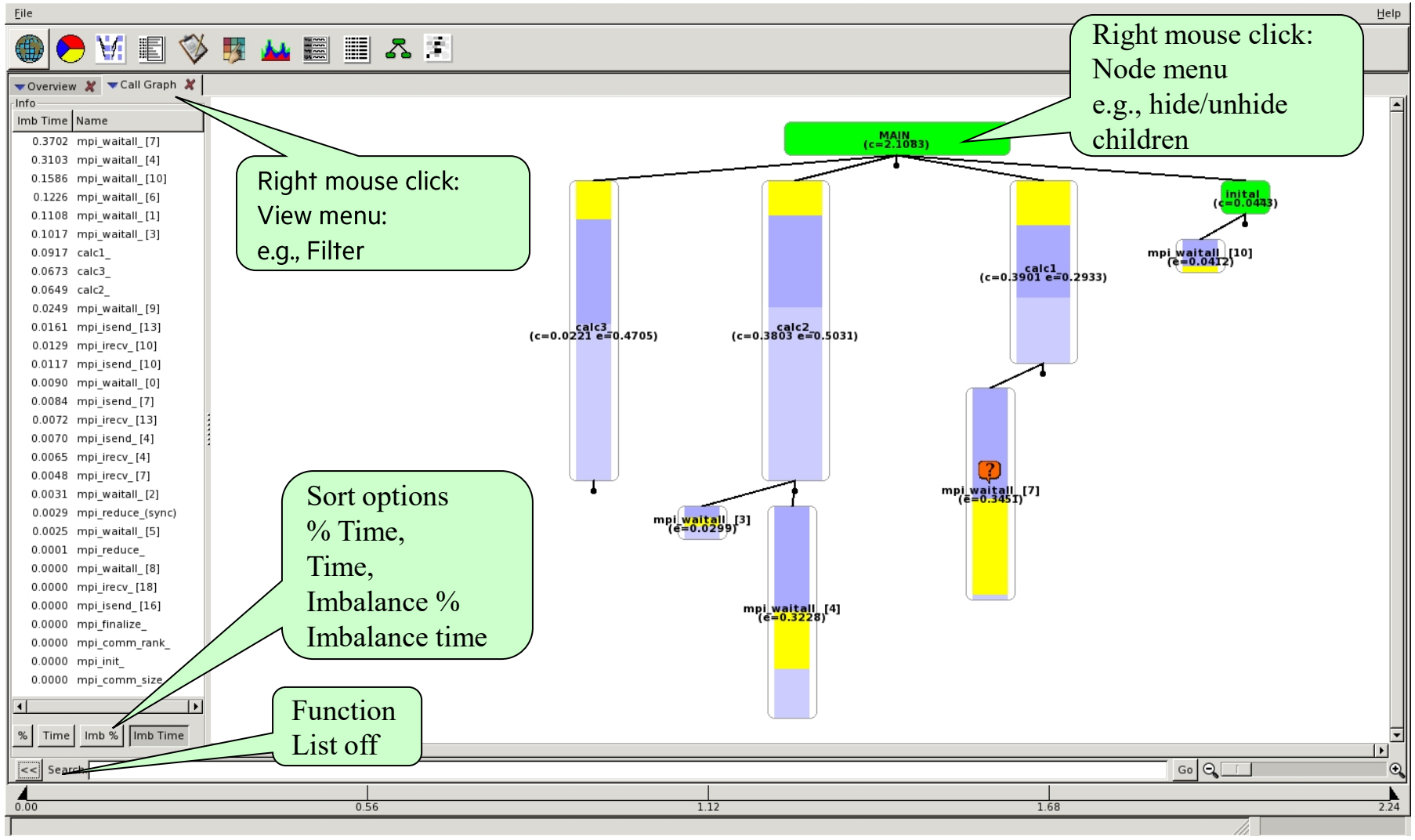
LOAD BALANCE



CALL TREE VIEW

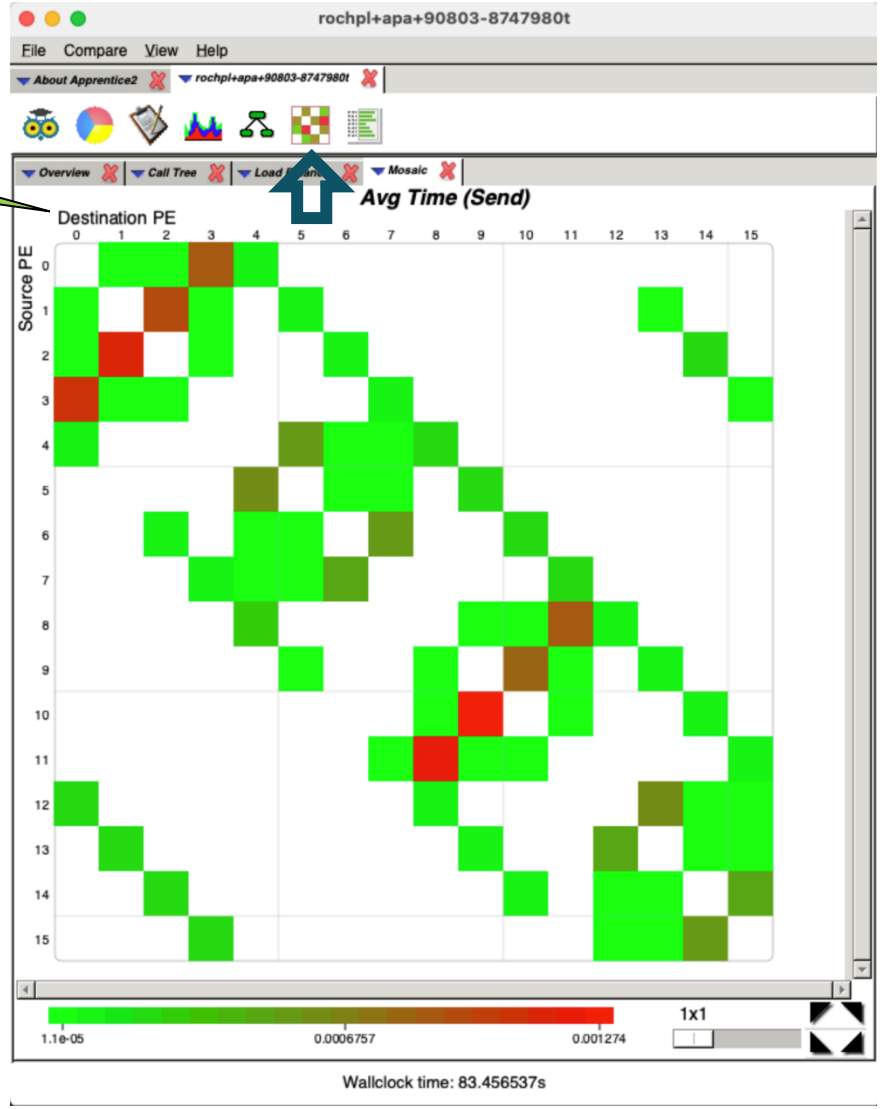


CALL TREE VIEW - FUNCTION LIST



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`**

PERFTOOLS-LITE

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

DEMO 1C_PERFTOOLS_LITE

TWO FUNDAMENTAL WAYS OF PROFILING

1. Sampling

- By taking regular snapshots of the applications call stack we can create a statistical profile of where the application spends most time.
- Snapshots can be taken at regular intervals in time or when some other external event occurs, like a hardware counter overflowing

Advantages

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

Disadvantages

- Only statistical averages available
- Limited information from performance counters

2. Event Tracing

- Alternatively, we can record performance information every time a specific program event occurs, e.g. entering or exiting a function.
- We can get accurate information about specific areas of the code every time the event occurs
- Event tracing code can be added automatically or included manually through API calls.

Advantages

- More accurate and more detailed information
- Data collected from every traced function call not statistical averages

Disadvantages

- Increased overheads as number of function calls increases
- Huge volumes of data generated

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.

OUTPUT: SAMPLING

```
CrayPat/X:  Version 6.4.1 Revision 6a6694f  06/27/16 17:24:11
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: 38241.909182 time(s): 23.633639 4.157338e-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 : 11.851884 sec.
Loop executed for 50 times
Gosa : 4.109140e-04
MFLOPS measured : 38128.768748
Score based on Pentium III 600MHz : 460.270024


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

CrayPat/X:  Version 6.4.1 Revision 6a6694f  06/27/16 17:24:11
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:  Mon Mar  6 14:10:34 2017
System name and speed:  nid00036  2301 MHz (approx)
Intel haswell CPU  Family:  6  Model: 63  Stepping:  2


Avg Process Time:        36.82 secs
High Memory:         14,927.0 MBytes 1,865.9 MBytes per PE
MFLOPS:         Not supported (see observation below)
I/O Write Rate:       0.308860 MBytes/sec
Avg CPU Energy:          8,923 joules   8,923 joules per node
Avg CPU Power:         242.31 watts   242.31 watts per node
```

Regular program output

General job information

Sampling summary

Further analysis

```
Table 1:  Profile by Function Group and Function (top 4 functions shown)

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

 100.0% | 3,670.8 |  --    |  --     |Total
|-----------------------------------------------
| 100.0% | 3,670.8 |  --    |  --     |ETC
||----------------------------------------------
||  82.2% | 3,018.5 | 62.5   |  2.3%   | jacobi
||  12.2% |  447.8  |  5.2   |  1.3%   | __cray_scopy_HSW
||   2.3% |   84.5  |  4.5   |  5.8%   | initmt
||   2.2% |   79.0  | 48.0   | 43.2%   | sendp
||   1.1% |   38.8  |  2.2   |  6.3%   | __cray_sset_HSW
|================================================

================== Observations and suggestions ==================

MFLOPS not available on Intel Haswell:

   The document that specifies performance monitoring events for Intel
   processors does not include events that could be used to compute a
   count of floating point operations for Haswell processors: Intel 64
   and IA-32 Architectures Software Developer's Manual, Order Number
   253665-050US, February 2014.

MPI utilization:

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

======================== End Observations ========================
```

```
Table 2:  File Output Stats by Filename

   Write |   Write | Write Rate | Writes | Bytes/ |File Name[max10]
    Time |   MBytes | MBytes/sec |      |  Call | PE=HIDE

 0.001908 | 0.000589 |  0.308860 |  18.0 |  34.33 |Total
|-------------------------------------------------------------------
| 0.001908 | 0.000589 |  0.308860 |  18.0 |  34.33 |stdout
|===================================================================

Program invocation:  himeno.exe

For a complete report with expanded tables and notes, run:
  pat_report /lustre/scratch/x_esposia/test/himeno.exe+10827-36s.ap2

For help identifying callers of particular functions:
  pat_report -O callers+src /lustre/scratch/x_esposia/test/himeno.exe+10827-36s.ap2
To see the entire call tree:
  pat_report -O calltree+src /lustre/scratch/x_esposia/test/himeno.exe+10827-36s.ap2

For interactive, graphical performance analysis, run:
  app2 /lustre/scratch/x_esposia/test/himeno.exe+10827-36s.ap2

================ 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

```
CrayPat/X:  Version 6.4.1 Revision 6a6694f  06/27/16 17:24:11
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: 38452.612767 time(s): 23.504137 4.157338e-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 : 11.744095 sec.
Loop executed for 50 times
Gosa : 4.109140e-04
MFLOPS measured : 38478.720227
Score based on Pentium III 600MHz : 464.494450


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


CrayPat/X:  Version 6.4.1 Revision 6a6694f  06/27/16 17:24:11
Experiment:               lite  lite/event_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:  Mon Mar  6 16:38:03 2017
System name and speed:  nid00036  2301 MHz (approx)
Intel haswell CPU  Family:  6  Model: 63  Stepping:  2



Avg Process Time:    36.57 secs
High Memory:      14,918.6 MBytes     1,864.8 MBytes per PE
I/O Write Rate:   0.347860 MBytes/sec
```

Regular program output

Event tracing summary. Note difference to sampling

Further analysis

```
Table 1:  Profile by Function Group and Function

  Time% |     Time     |  Imb.       | Imb.    |  Calls    | Group
        |              |  Time       | Time%   |           | Function
        |              |             |         |           |  PE=HIDE

 100.0% | 37.169472 |     --        |   --    | 2,589.2 | Total
|-----------------------------------------------------------
|  96.9% | 36.02032   |     --        |   --    |    6.0    | USER
||----------------------------------------------------------
||  93.5% | 34.76561   | 0.764943 |  2.5%    |    2.0    | jacobi
||   3.4% |  1.253708  | 0.006834 |  0.6%    |    1.0    | initmt
||==========================================================
|   3.1%  |  1.135804 |     --        |   --    | 2,423.0 |MPI
||----------------------------------------------------------
||   2.6%  |  0.950663 | 1.135159  | 62.2%    |   450.0 | MPI_Waitall
|==========================================================
```

```
Table 2:  File Output Stats by Filename

    Write |   Write | Write Rate | Writes | Bytes/ |File Name[max10]
     Time |   MBytes | MBytes/sec |        |  Call | PE=HIDE

 0.001694 | 0.000589 |   0.347860 |   18.0 | 34.33 |Total
|-------------------------------------------------------------------
| 0.001694 | 0.000589 |   0.347860 |   18.0 | 34.33 |stdout
|===================================================================
Program invocation:  himeno.exe

For a complete report with expanded tables and notes, run:
  pat_report /lustre/scratch/x_esposia/test/himeno.exe+11229-36t.ap2


For help identifying callers of particular functions:
  pat_report -O callers+src /lustre/scratch/x_esposia/test/himeno.exe+11229-36t.ap2
To see the entire call tree:
  pat_report -O calltree+src /lustre/scratch/x_esposia/test/himeno.exe+11229-36t.ap2


For interactive, graphical performance analysis, run:
  app2 /lustre/scratch/x_esposia/test/himeno.exe+11229-36t.ap2


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

General job information

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

```
CrayPat/X:  Version 6.4.1 Revision 6a6694f  06/27/16 17:24:11
PGO data version:  L.14.1:B.3.1
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: 34448.127778 time(s): 26.236418 4.157541e-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 : 13.261153 sec.
Loop executed for 50 times
Gosa : 4.108945e-04
MFLOPS measured : 34076.806186
Score based on Pentium III 600MHz : 411.356907


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

CrayPat/X:  Version 6.4.1 Revision 6a6694f  06/27/16 17:24:11
Experiment:                  lite  lite/loop_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:  Mon Mar  6 17:49:00 2017
System name and speed:  nid00036  2301 MHz (approx)
Intel haswell CPU  Family:  6  Model: 63  Stepping:  2



Avg Process Time:    40.82 secs
High Memory:      14,919.3 MBytes 1,864.9 MBytes per PE
```

Regular program output

General job information

Loop Statistics by function

Table 1: Inclusive and Exclusive Time in Loops (from -hprofile_generate)

Loop Incl Time%	Loop Incl Time	Time (Loop Adj.)	Loop Hit	Loop Trips Avg	Loop Trips Min	Loop Trips Max	Function=/.LOOP[.] PE=HIDE
97.0%	39.497551	7.219257	2	75.0	50	100	jacobi.LOOP.1.li.236
67.8%	27.620370	0.003188	150	255.0	255	255	jacobi.LOOP.2.li.240
67.8%	27.617181	0.230981	38,250	255.0	255	255	jacobi.LOOP.3.li.241
67.2%	27.386201	27.386201	9,753,750	511.0	511	511	jacobi.LOOP.4.li.242
11.4%	4.657924	0.001652	150	255.0	255	255	jacobi.LOOP.5.li.263
11.4%	4.656272	0.959343	38,250	255.0	255	255	jacobi.LOOP.6.li.264
9.1%	3.696929	3.696929	9,753,750	511.0	511	511	jacobi.LOOP.7.li.265
2.1%	0.847145	0.000004	1	259.0	259	259	initmt.LOOP.1.li.191
2.1%	0.847141	0.000605	259	259.0	259	259	initmt.LOOP.2.li.192
2.1%	0.846535	0.846535	67,081	515.0	515	515	initmt.LOOP.3.li.193

Program invocation: himeno.exe

For a complete report with expanded tables and notes, run:
 pat_report /lustre/scratch/x_esposia/test/himeno.exe+11333-36t.ap2

Subroutine

Line number

Nested Loops

PERFTOOLS-LITE 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 create 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
```

- 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

```
 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
|===============================================
```

FURTHER ANALYSIS (WITHOUT RE-RUNNING)

- Generate report from specific subset of ranks

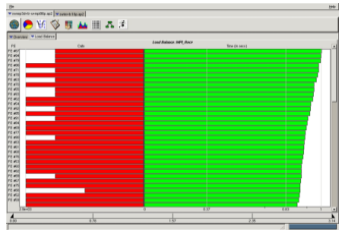
```
$> pat_report -s filter_input='pe==0'
```

  - 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

- Example traffic from an MPI+OpenMP run.

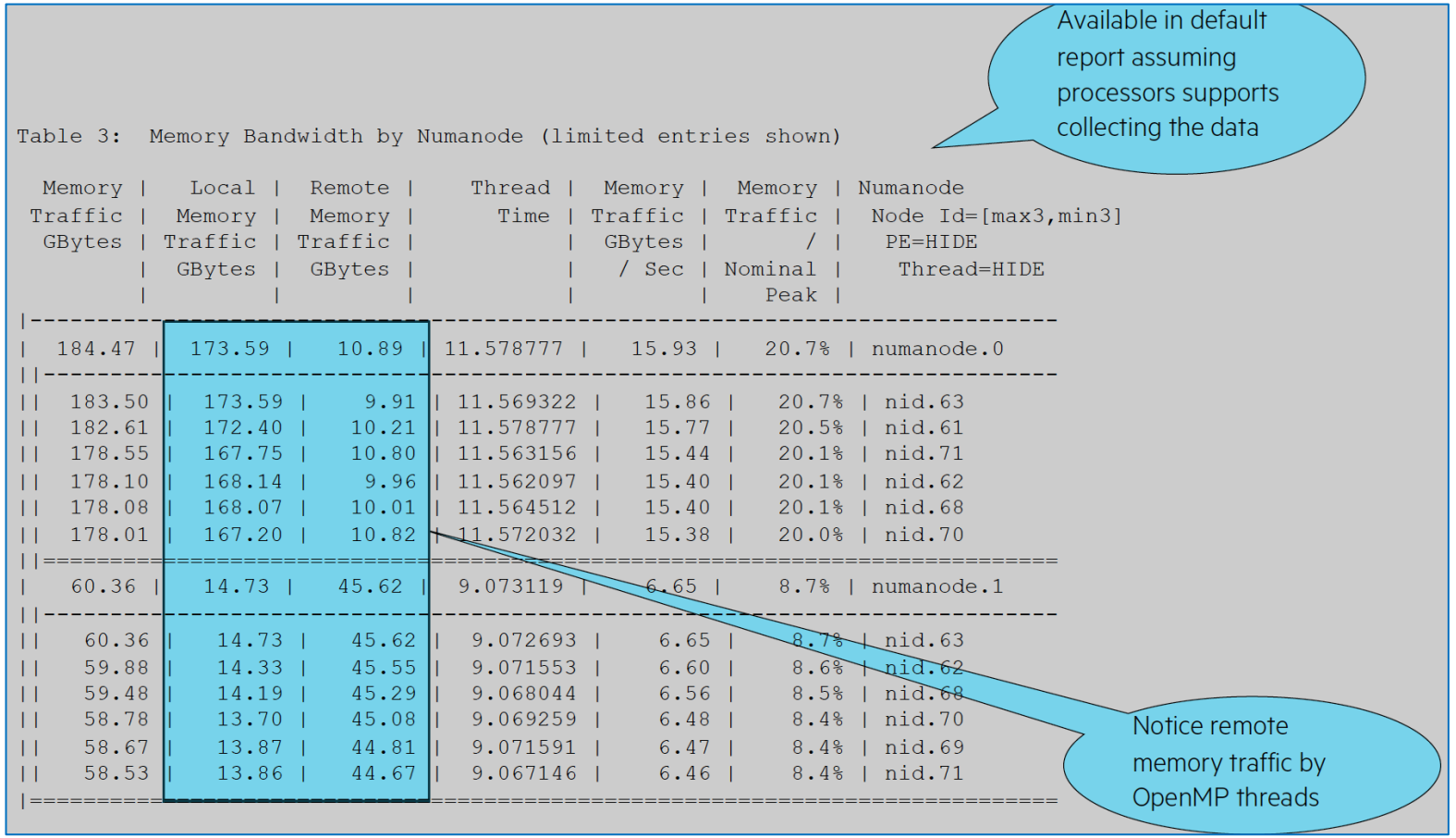


Table 3: Memory Bandwidth by Numanode (limited entries shown)

Memory Traffic GBytes	Local Memory Traffic GBytes	Remote Memory Traffic GBytes	Thread Time	Memory Traffic GBytes / Sec	Memory Traffic / Nominal Peak	Numanode Node Id=[max3,min3] PE=HIDE Thread=HIDE
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

OBSERVATION EXAMPLES

```
Observation:  Stall cycles are 66.4% of total cycles,
which exceeds the guideline of 40.0%. This can be caused
by issues in the program such as saturation of memory
bandwidth.
```

```
Observation: Program sensitivity to vectorization: The
low proportion of packed instructions and low rate of
stalls in the program suggest that its performance
could be improved by increased vectorization. Use
compiler optimization messages to identify loops in
functions high in the profile that were not vectorized
and try to use directives or restructure the loops to
enable them to be vectorized.
```

OBSERVATION EXAMPLES

```
Observation: Metric-Based Rank Order No rank order was
suggested based on the USER Samp metric because that
metric was already well balanced across the nodes.
```

```
Observation:  Program sensitivity to memory latency The
relatively low memory bandwidth utilization but
significant rate of stalls in the program suggest that
its performance is limited by memory latency. It could
be beneficial to improve prefetching in loops in
functions high in the profile, by modifying compiler-
generated prefetches or inserting directives into the
source code.
```

PERFTOOLS

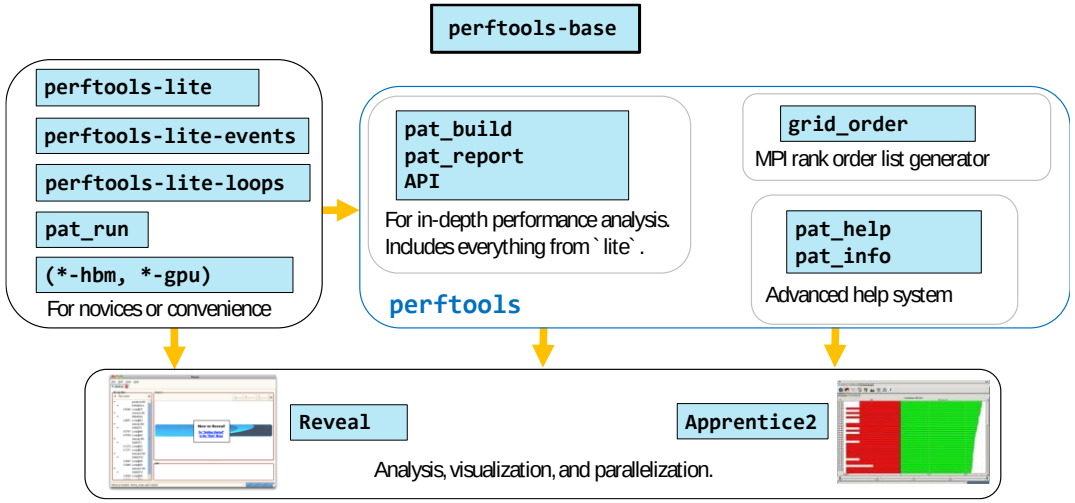
In-depth Performance Analysis

DEMO PAT_HELP 1D_PERFTOOLS_APA

OVERVIEW

The `perftools` components:

- Instrumentation and report generation must be done manually.
- `pat_build` — instruments the program to be analyzed.
- `pat_report` — generates text reports from the performance data captured during program execution and exports data for use in other programs.
- These tools together with Reveal, Apprentice2, `pat_help`, and `pat_info` are in principle already available with `perftools-base`



AUTOMATIC PROFILING ANALYSIS (1/2)

```
> module load perftools-base
> module load perftools
```

- Setup the perftools environment

```
> make clean; make
> pat_build app.exe
```

- The APA (`-O apa`) is the default experiment. No option needed.
- The `pat_build` generates a binary instrumented for sampling (different from the pure sampling shown before.)

```
> srun -n 8 app.exe+pat
> pat_report -o myrep.txt app+pat+*/
```

- Running the "`+pat`" binary creates an experiment directory.
- Applying `pat_report` to the `app+pat+*/` directory generates an `*.apa` file therein.

AUTOMATIC PROFILING ANALYSIS (2/2)

```
> vi app.exe+pat+*/*.apa
```

- The `*.apa` file contains instructions for the next step, i.e. tracing. Modify it according to your needs.

```
> pat_build -O app.exe+pat+*/*.apa
```

- Generates an instrumented binary `app.exe+apa` for tracing according to the instructions in the `app.exe+pat+*/*.apa` file.

```
> srun -n 8 ./app.exe+apa
> pat_report -o myrep.txt app.exe+apa+*/
```

- Running the `app.exe+apa` binary creates a new data file or directory.
- Applying `pat_report` to the `app+apa+*/` directory generates a new report.

OUTPUT: APA WITH PERFTOOLS

Sampling

```
CrayPat/X:  Version 6.4.1 Revision 6a6694f  06/27/16 17:24:11

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:  Wed Mar  8 14:24:42 2017
System name and speed:  nid00036  2301 MHz (approx)
Intel haswell CPU  Family:  6  Model: 63  Stepping:  2
...

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

 100.0% | 3,685.0 |  -- |   -- |Total
|-----------------------------------------------------
| 84.2% | 3,104.5 |  -- |   -- |USER
||----------------------------------------------------
|| 81.9% | 3,018.1 | 57.9 |  2.2% |jacobi
||  2.3% |   86.1 |  2.9 |  3.7% |initmt
||====================================================
| 13.3% |  489.9 |  -- |   -- |ETC
||----------------------------------------------------
|| 12.2% |  451.2 |  6.8 |  1.7% |__cray_scopy_HSW
||  1.0% |   38.6 |  2.4 |  6.6% |__cray_sset_HSW
||====================================================
|  2.5% |   90.6 |  -- |   -- |MPI
||----------------------------------------------------
||  1.9% |   69.6 | 46.4 | 45.7% |MPI_Waitall
|====================================================
...
 Total
-------------------------------------------------------------------------------
 CPU_CLK_THREAD_UNHALTED:THREAD_P          109,408,212,031
 CPU_CLK_THREAD_UNHALTED:REF_XCLK            3,588,344,079
 DTLB_LOAD_MISSES:MISS_CAUSES_A_WALK            53,188,056
 PM_ENERGY:NODE              30.382 /sec          1,112 J
 CPU_CLK                     3.05GHz
 TLB utilization           268.83 refs/miss       0.53 avg uses
 D1 cache hit,miss ratios   70.8% hits          29.2% misses
 D1 cache utilization (misses)   3.43 refs/miss       0.43 avg hits
 D2 cache hit,miss ratio    67.8% hits          32.2% misses
 D1+D2 cache hit,miss ratio   90.6% hits          9.4% misses
 D1+D2 cache utilization     10.65 refs/miss       1.33 avg hits
 D2 to D1 bandwidth      6,149.405MiB/sec  236,055,684,984 bytes
```

Suggestion to collect Performance counters

APA file

```
# ----------------------------------------------------------------------
# Collect the default PERFCTR group.

-Drtenv=PAT_RT_PERFCTR=default

...

# Libraries to trace.

-g mpi

# ----------------------------------------------------------------------

# User-defined functions to trace, so
...

  -w  # Enable tracing of user-defined functions.
      # Note: -u should NOT be specified as an additional option.

# 81.90% 3735 bytes
      -T jacobi

# 2.34% 2450 bytes
      -T initmt

# ----------------------------------------------------------------------

-o himeno.exe+apa # New instrumented program.

...
```

Augment this list if needed, i.e. `-g mpi,io`

Add or remove functions as needed.

Create the binary for tracing

Event Tracing

```
CrayPat/X:  Version 6.4.1 Revision 6a6694f  06/27/16 17:24:11

Number of PEs (MPI ranks):   8
Numbers of PEs per Node:     8
Numbers of Threads per PE:   1
Number of Cores per Socket:  16
...
Table 1:  Profile by Function Group and Function


 Time% |     Time |     Imb. |  Imb. |   Calls |Group
       |          |     Time | Time% |         | Function
       |          |          |       |         | PE=HIDE

 100.0% | 37.748701 |      -- |   -- | 2,584.0 |Total
|-------------------------------------------------------------
|  95.9% | 36.184768 |      -- |   -- |     5.0 |USER
||------------------------------------------------------------
||  92.5% | 34.935081 | 1.178364 |  3.7% |     2.0 |jacobi
||   3.3% |  1.248950 | 0.016487 |  1.5% |     1.0 |initmt
|||===========================================================
||   4.1% |  1.537996 |      -- |   -- | 2,423.0 |MPI
||------------------------------------------------------------
||   3.6% |  1.344349 | 1.096798 | 51.3% |   450.0 |MPI_Waitall

...
==============================================================
USER / jacobi
--------------------------------------------------------------
Time%                                        92.5%
Time                                   34.935081 secs
Imb. Time                               1.178364 secs
Imb. Time%                                  3.7%
Calls                  0.057 /sec           2.0 calls
CPU_CLK_THREAD_UNHALTED:THREAD_P     104,712,083,433

CPU_CLK_THREAD_UNHALTED:REF_XCLK       3,465,232,627
DTLB_LOAD_MISSES:MISS_CAUSES_A_WALK       44,033,261
DTLB_STORE_MISSES:MISS_CAUSES_A_WALK      26,116,487

L1D:REPLACEMENT                      6,955,645,277
L2_RQSTS:ALL_DEMAND_DATA_RD          3,675,034,522
L2_RQSTS:DEMAND_DATA_RD_HIT          1,408,925,969
MEM_UOPS_RETIRED:ALL_LOADS          22,406,881,264
CPU_CLK                     3.02GHz
TLB utilization           319.41 refs/miss       0.62 avg uses
D1 cache hit,miss ratios   69.0% hits          31.0% misses
D1 cache utilization (misses)   3.22 refs/miss       0.40 avg hits
```

SAMPLING WITH PERFTOOLS

```
> module load perftools-base
> module load perftools
```

- No automatic instrumentation takes place as with `perftools-lite*`.

```
> make clean; make
> pat_build -S app.exe
```

- This generates a new executable `app.exe+pat` and preserves `app.exe`.
- Object files must be present during this stage.

```
> srun -n 8 ./app.exe+pat
> pat_report -o myrep.trace.rpt app.exe+pat+*/
```

- Running the `app.exe+pat` creates an `app.exe+pat+*/` directory.
- `pat_report` reads that directory and prints a lot of human-readable performance data.
- If worthwhile, a MPI rank reordering file will be produced.

EVENT TRACING USING PERFTOOLS

```
> module load perftools-base
> module load perftools
```

- No automatic instrumentation takes place.

```
> make clean; make
> pat_build -u -g mpi app.exe
```

- This generates a new executable `app.exe+pat` and preserves `app.exe`.
- If object files and user libraries have already been compiled with perftools enabled, just relink the application with `rm app.exe; make` instead.
- Traces MPI functions calls and functions defined in the program source files.

```
> srun -n 8 ./app.exe+pat
> pat_report -o myrep.trace.rpt app.exe+pat+*/
```

- Running the `app.exe+pat` creates a `app.exe+pat+*/` directory.
- `pat_report` reads that directory and prints a lot of human-readable performance data.

OUTPUT: EVENT TRACING WITH PERFTOOLS

```
CrayPat/X:  Version 6.4.1 Revision 6a6694f  06/27/16 17:24:11

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:  Tue Mar  7 21:28:21 2017
System name and speed:  nid00036  2301 MHz (approx)
Intel haswell CPU  Family:  6  Model: 63  Stepping:  2

....

Table 1:  Profile by Function Group and Function


  Time% |      Time |     Imb. |  Imb. |    Calls |Group
        |           |     Time | Time% |          | Function
        |           |          |       |          | PE=HIDE

 100.0% | 36.703706 |       -- |    -- | 3,036.0 |Total
|-----------------------------------------------------------------------
| 97.2% | 35.666406 |       -- |    -- |   457.0 |USER
||----------------------------------------------------------------------
|| 93.8% | 34.432069 | 0.683771 |  2.2% |     2.0 |jacobi
||  3.4% |  1.232095 | 0.012880 |  1.2% |     1.0 |initmt
||  0.0% |  0.000941 | 0.001077 | 61.0% |     1.0 |main
||  0.0% |  0.000486 | 0.000232 | 36.9% |   150.0 |sendp
||  0.0% |  0.000356 | 0.000445 | 63.5% |   150.0 |sendp3
||  0.0% |  0.000266 | 0.000024 |  9.3% |     1.0 |initcomm
||  0.0% |  0.000172 | 0.000192 | 60.3% |   150.0 |sendp2
||  0.0% |  0.000020 | 0.000008 | 31.8% |     1.0 |exit
||  0.0% |  0.000002 | 0.000001 | 29.9% |     1.0 |initmaxclone_12853_1
||=======================================================================
==========
|  2.8% |  1.019139 |       -- |    -- | 2,423.0 |MPI
||----------------------------------------------------------------------
||  2.3% |  0.833848 | 0.688832 | 51.7% |   450.0 |MPI_Waitall
||  0.5% |  0.169196 | 0.025124 | 14.8% |   900.0 |MPI_Isend
||  0.0% |  0.012631 | 0.016372 | 64.5% |   900.0 |MPI_Irecv
||  0.0% |  0.001162 | 0.000014 |  1.3% |   152.0 |MPI_Allreduce
||  0.0% |  0.001077 | 0.000043 |  4.4% |     1.0 |MPI_Cart_create
||  0.0% |  0.000703 | 0.000038 |  5.8% |     3.0 |MPI_Type_commit
...
||=======================================================|
0.0% |  0.018161 |       -- |    -- |   156.0 |MPI_SYNC
||----------------------------------------------------------------------
||  0.0% |  0.014154 | 0.014143 | 99.9% |     2.0 |MPI_Barrier(sync)
||  0.0% |  0.003969 | 0.003266 | 82.3% |   152.0 |MPI_Allreduce(sync)
||  0.0% |  0.000029 | 0.000027 | 92.5% |     1.0 |MPI_Init(sync)
||  0.0% |  0.000009 | 0.000009 | 99.6% |     1.0 |MPI_Finalize(sync)
|=======================================================
```

General job information

Load balance with MPI message statistics

More on MPI with message sizes

```
Table 2:  Load Balance with MPI Message Stats (limited entries shown)

 Time% |      Time |   MPI | MPI Msg Bytes |    Avg MPI |Group
       |           |   Msg |               |   Msg Size | PE=[mmm]
       |           | Count |               |            |

100.0% | 36.703700 | 602.0 | 197,839,216.0 | 328,636.57 |Total
|-------------------------------------------------------------------
| 97.2% | 35.666406 |   0.0 |         0.0 |        -- |USER
||------------------------------------------------------------------
|| 99.1% | 36.361896 |   0.0 |         0.0 |        -- |pe.3
|| 96.9% | 35.581600 |   0.0 |         0.0 |        -- |pe.7
|| 95.3% | 34.961048 |   0.0 |         0.0 |        -- |pe.5
|===================================================================
|  2.8% |  1.019132 | 602.0 | 197,839,216.0 | 328,636.57 |MPI
||------------------------------------------------------------------
||  4.7% |  1.722054 | 602.0 | 197,839,216.0 | 328,636.57 |pe.5
||  2.4% |  0.881184 | 602.0 | 197,839,216.0 | 328,636.57 |pe.1
||  0.9% |  0.332689 | 602.0 | 197,839,216.0 | 328,636.57 |pe.3
|===================================================================
|  0.0% |  0.018161 |   0.0 |         0.0 |        -- |MPI_SYNC
||------------------------------------------------------------------
||  0.1% |  0.029523 |   0.0 |         0.0 |        -- |pe.6
||  0.1% |  0.019500 |   0.0 |         0.0 |        -- |pe.1
||  0.0% |  0.001622 |   0.0 |         0.0 |        -- |pe.0
|===================================================================
```

```
Table 3:  MPI Message Stats by Caller (limited entries shown)

  MPI | MPI Msg Bytes |   MPI | MsgSz | 64KiB<= |Function
  Msg |               |   Msg |  <16  |  MsgSz  | Caller
Bytes% |               | Count | Count |  <1MiB  | PE=[mmm]
       |               |       |       |  Count  |

100.0% | 197,839,216.0 | 602.0 | 152.0 |   450.0 |Total
|-----------------------------------------------------------------
| 100.0% | 197,838,600.0 | 450.0 |   0.0 |   450.0 |MPI_Isend
||----------------------------------------------------------------
||  40.0% |  79,104,600.0 | 150.0 |   0.0 |   150.0 |sendp2
3||       |               |       |       |         | sendp
4||       |               |       |       |         | jacobi
5||       |               |       |       |         |  main
|||||-------------------------------------------------------------
6|||||  40.0% |  79,104,600.0 | 150.0 |   0.0 |   150.0 |pe.0
6|||||  40.0% |  79,104,600.0 | 150.0 |   0.0 |   150.0 |pe.4
6|||||  40.0% |  79,104,600.0 | 150.0 |   0.0 |   150.0 |pe.7
|||||=============================================================
||  40.0% |  79,104,600.0 | 150.0 |   0.0 |   150.0 |sendp1
3||       |               |       |       |         |jacobi
4||       |               |       |       |         | main
|||||-------------------------------------------------------------
5|||||  40.0% |  79,104,600.0 | 150.0 |   0.0 |   150.0 |pe.0
5|||||  40.0% |  79,104,600.0 | 150.0 |   0.0 |   150.0 |pe.4
5|||||  40.0% |  79,104,600.0 | 150.0 |   0.0 |   150.0 |pe.7
|||||=============================================================
||  20.0% |  39,629,400.0 | 150.0 |   0.0 |   150.0 |sendp3
```

All user defined routines traced and shown (-T option).

OPTIONS FOR TRACING

- More information is given in the `pat_build` man page
  - **-u** Create new trace intercept routines for those functions that are defined in the respective source file owned by the user.
  - **-w** Make tracing the default experiment and create new trace intercept routines for those functions for which no trace intercept routine already exists. If -t, -T, or the trace build directive are not specified, only those functions necessary to support the CrayPat runtime library are traced.
  - **-T tracefunc** Instrument program to trace the function references to tracefunc. This option applies to all user-defined entry points as well as to those that appear in the predefined function groups listed under the -g option. Use the nm or readelf command to determine function names to specify for tracing. If tracefunc begins with an exclamation point (!) character, references to tracefunc are not traced.
  - **-t tracefile** Instrument program to trace all function references listed in tracefile.
- Only true function calls can be traced. Functions that are inlined by the compiler or that have local scope in a compilation unit cannot be traced.

OPTIONS FOR TRACING

- More information is given in the `pat_build` man page
  - **-g tracegroup** Instrument the program to trace all function references belonging to the trace function group tracegroup. Only those functions actually executed by the program at runtime are traced. A selection of tracegroup values is:

    - **blas** Basic Linear Algebra subprograms
    - **netcdf** Network Common Data Form
    - **hdf5** HDF5 I/O library
    - **heap** dynamic heap
    - **io** includes stdio and sysio groups
    - **lapack** Linear Algebra Package
    - **mpi** MPI
    - **omp** OpenMP API
    - **sysio** I/O system calls
    - **syscall** system calls

- More information on the various tracegroup values is given in `$CRAYPAT_ROOT/share/traces` after loading the `perftools-base` module.

USING PAT_REPORT

- Always need to run `pat_report` at least once to perform data conversion
  - Experiment directory contains *.xf files in the xf-files/ subdirectory.
  - Perftools combines information from `xf` output (raw performance data optimized for writing to disk) and executable to produce `ap2` files (optimized for visualization analysis) stored in the ap2-files/ subdirectory.
  - Instrumented binary must still exist when data is converted!
  - Resulting `ap2` files are the input for subsequent `pat_report` calls and Reveal or Apprentice2
    - Always use the entire experiment directory as input and not single ap2 files.
  - `xf` files and instrumented binary files could be removed once `ap2` file is generated.

- Generates a text report of performance results
  - Data laid out in tables
  - Many options for sorting, slicing or dicing data in the tables.

```
> pat_report -O <table option> app.exe+pat+*/
> pat_report -O help (list of available profiles)
```

  - Volume and type of information depends upon sampling vs tracing.

USING PAT_REPORT

- The performance numbers reported are in general an average over all tasks (also explains non-integer values)

```
 Time% |      Time |     Imb. |  Imb. |  Calls |Group
       |           |     Time | Time% |        | Function
       |           |          |       |        |  PE=HIDE

 100.0% | 20.643909 |       -- |    -- | 1149.0 |Total
|-----------------------------------------------------------
|  98.8% | 20.395989 |       -- |    -- |  219.0 |USER
||----------------------------------------------------------
||  91.1% | 18.797060 | 0.115535 |  0.7% |    2.0 |jacobi
||   7.7% |  1.597866 | 0.006647 |  0.5% |    1.0 |initmt
||   0.0% |  0.000402 | 0.000167 | 33.5% |   53.0 |sendp3
```

- Not always meaningful
  - Master-slave schemes
  - MPMD
- To solve this you can filter the *.ap2 file

```
> pat_report -sfilter_input='condition' …
```

  - The 'condition' should be an expression involving 'pe' such as 'pe<1024' or 'pe%2==0'.
  - This option is also useful when the size of the full data file makes a report incorporating data from all PEs take too long or exceed the available memory
- More help:
  - `pat_report -h` => usage
  - `pat_report -O -h` => available report tables
  - `pat_report -s -h` => options for content and format
  - `pat_report -d -h` => options for data columns

GENERAL REMARKS

- Always check that the instrumenting binary has not affected the run time significantly compared to the original executable.
- Collecting event traces on large numbers of frequently called functions, or setting the sampling interval very low can introduce a lot of overhead (check `trace-text-size` option to `pat_build`)
- Highly recommended to run on Lustre !
- The runtime analysis can be modified using environment variables of the form `PAT_RT_*`
  - Check the `PAT_LD_OBJECT_TMPDIR` variable if you cannot preserve the original build tree.
- `pat_build` may recognize for instance MPI in your application and trace `MPI_Init` and `MPI_Finalize`. Not the same as `-g mpi` which traces all MPI calls.
- Processing the `app+*/` directory from `perftools-lite` yields all the options of `pat_build` to reproduce the experiment with regular `perftools`.

PERFTOOLS API

Customized performance analysis.

DEMO 1E_PERFTOOLS_API

API FOR ADDING USER INSTRUMENTATION

- The Perftools API calls enable you to insert functions into your source code that write special tracing records into the experiment data file at runtime. Useful for large routines
  - API calls are supported in both Fortran and C.
  - After the `perftools` module is loaded, the include files that define the Perftools API can be found in the `$CRAYPAT_ROOT/include` directory and consist of the C header file, `pat_api.h`, and the Fortran and Fortran 77 header files, `pat_apif.h` and `pat_apif77.h`, respectively.
  - `int PAT_region_begin (int id, char *label)`
    - `id` is a unique identifier for the region,
    - Label is the description that will appear in profiling output.
  - `int PAT_region_end (int id)`
    - `id` must match begin call.
  - Fortran equivalents, are subroutines with extra final integer argument for return value similar to MPI.
  - Data collection through API can be controlled with `PAT_RT_TRACE_API`

API FOR ADDING USER INSTRUMENTATION

- Main information is given in the `pat_build` man page in the section APPLICATION PROGRAM INTERFACE.
  - For further examples of Pertools API calls in source code, see the topic "API" in the `pat_help` system.
- When one of the `perftools` instrumentation modules is loaded, it defines a compiler macro called `CRAYPAT` which can be used to wrap headers and API calls.

C:

```
#ifdef CRAYPAT
PAT_region_begin ( 1, "loop" );
#endif
```

Fortran:

```
#ifdef CRAYPAT
call PAT_region_begin ( 1, "loop", istat );
#endif
```

- HWPC are collected for individual regions.
- The API works for sampling, tracing, and loop profiling as well as with the `perftools-lite*` modules.

PAT REGIONS EXAMPLE IN C AND FORTRAN

The #ifdef CRAYPAT is not mandatory but helps code handling.

C:

```
#include "pat_api.h“
...
PAT_region_begin( 1, ”jacobi_part1");
// the execution of this code segment will
// appear in CrayPAT output as “jacobi_part1”
 ...
PAT_region_end( 1);
 ...
PAT_region_begin( 2, ”jacobi_part2”);
// the execution of this code segment will
// appear in CrayPAT output as “jacobi_part2”
 ...
PAT_region_end( 2);
 ...
```

Fortran:

```
include "pat_apif.h“
...
call PAT_region_begin( 1, ”jacobi_part1", istat )
! the execution of this code segment will
! appear in CrayPAT output as “jacobi_part1”
...
call PAT_region_end( 1, istat )
...
call PAT_region_begin( 2, ”jacobi_part2", istat )
! the execution of this code segment will
! appear in CrayPAT output as “jacobi_part2”
...
call PAT_region_end( 2, istat )
...
```

EVENT TRACING W/ AND W/O PERFTOOLS API (C)

Regions

With API

Time%	Time	Imb. Time	Imb. Time%	Calls	Group Function PE=HIDE
100.0%	45.429778	--	--	3,336.0	Total
82.6%	37.513233	--	--	757.0	USER
69.9%	31.777804	7.855317	22.7%	150.0	#1.jacobi_part1
9.9%	4.480465	0.484320	11.1%	150.0	#2.jacobi_part2
2.8%	1.252451	0.066733	5.8%	1.0	initmt
0.0%	0.001015	0.002053	76.5%	1.0	main
0.0%	0.000479	0.000493	58.0%	150.0	sendp
0.0%	0.000298	0.000075	22.9%	2.0	jacobi
0.0%	0.000297	0.000026	9.2%	1.0	initcomm
0.0%	0.000271	0.000211	50.0%	150.0	sendp3
0.0%	0.000132	0.000255	75.2%	150.0	sendp2
0.0%	0.000019	0.000000	2.8%	1.0	exit
17.3%	7.843884	--	--	2,423.0	MPI
16.8%	7.652373	2.858017	31.1%	450.0	MPI_Waitall
0.4%	0.170341	0.033417	18.7%	900.0	MPI_Isend
0.0%	0.016676	0.018946	60.8%	900.0	MPI_Irecv
0.0%	0.001239	0.000039	3.4%	1.0	MPI_Cart_create
0.0%	0.001214	0.000015	1.4%	152.0	MPI_Allreduce
0.0%	0.000933	0.000022	2.6%	3.0	MPI_Type_commit
0.0%	0.000644	0.000031	5.2%	3.0	MPI_Cart_shift
0.0%	0.000459	0.000027	6.5%	3.0	MPI_Type_vector
0.0%	0.000004	0.000002	34.5%	2.0	MPI_Barrier

Without API

Time%	Time	Imb. Time	Imb. Time%	Calls	Group Function PE=HIDE
100.0%	45.649773	--	--	3,036.0	Total
83.4%	38.065287	--	--	457.0	USER
80.6%	36.771025	7.132552	18.6%	2.0	jacobi
2.8%	1.291914	0.077825	6.5%	1.0	initmt
0.0%	0.001031	0.000933	54.3%	1.0	main
0.0%	0.000520	0.000342	45.3%	150.0	sendp
0.0%	0.000337	0.000028	8.7%	1.0	initcomm
0.0%	0.000320	0.000264	51.6%	150.0	sendp3
0.0%	0.000121	0.000322	83.1%	150.0	sendp2
0.0%	0.000017	0.000001	3.9%	1.0	exit
16.4%	7.500291	--	--	2,423.0	MPI
16.0%	7.298155	4.192142	41.7%	450.0	MPI_Waitall
0.4%	0.183874	0.048775	24.0%	900.0	MPI_Isend
0.0%	0.014285	0.020122	66.8%	900.0	MPI_Irecv
0.0%	0.001146	0.000013	1.3%	1.0	MPI_Cart_create
0.0%	0.001003	0.000012	1.4%	152.0	MPI_Allreduce
0.0%	0.000756	0.000012	1.7%	3.0	MPI_Type_commit
0.0%	0.000566	0.000009	1.8%	3.0	MPI_Type_vector
0.0%	0.000501	0.000015	3.3%	3.0	MPI_Cart_shift
0.0%	0.000005	0.000002	34.1%	2.0	MPI_Barrier
0.0%	0.000000	0.000000	15.4%	4.0	MPI_Wtime
0.0%	0.000000	0.000000	18.7%	1.0	MPI_Cart_get

LOAD IMBALANCE ANALYSIS

LOAD IMBALANCE

- Common cause for performance bottlenecks when running applications at scale
- Look for high imbalance time and percentage.
  - User functions

    Imbalance time = Maximum time – Average time
  - Synchronization (Collective communication and barriers)

    Imbalance time = Average time – Minimum time
  - Imbalance percentage of time that the rest of the team is not engaged in useful work on the given function
- Also look for MPI (SYNC) time.
  - Measure for time spent waiting in collectives.
  - Only available with event tracing experiments.
- Guidance on rank reordering for better load balance might appear in report (MPI Utilization: …)

```
Table 1:  Profile by Function Group and Function

  Time% |      Time |      Imb. |    Imb. |     Calls |Group
        |           |     Time  |   Time% |           | Function
        |           |           |         |           |  PE=HIDE

 100.0% | 1.957703 |       -- |      -- | 42,970.8 |Total
|-----------------------------------------------------------------
|  60.0% | 1.174021 |       -- |      -- |  3,602.0 |USER
||----------------------------------------------------------------
||  30.8% | 0.603850 | 0.176924 |  23.0% |  1,198.0 |function3_
||  19.2% | 0.375117 | 0.128748 |  26.0% |  1,200.0 |function2_
||   9.1% | 0.178111 | 0.081880 |  32.0% |  1,200.0 |function1_
||================================================================
|  36.0% | 0.704928 |       -- |      -- |  9,613.0 |MPI_SYNC
||----------------------------------------------------------------
||  25.8% | 0.505174 | 0.385130 |  76.2% |  9,596.0 |mpi_barrier_(sync)
||  10.2% | 0.199537 | 0.199518 | 100.0% |      1.0 |mpi_init_(sync)
||================================================================
|   4.0% | 0.078736 |       -- |      -- | 29,754.8 |MPI
||----------------------------------------------------------------
||   2.3% | 0.045351 | 0.003531 |   7.3% |  9,596.0 |MPI_BARRIER
||   1.1% | 0.021520 | 0.051295 |  71.6% |  8,756.9 |MPI_ISEND
|================================================================
```

COMMUNICATION BOTTLENECKS

- Sort messages by caller
  - Available in default report for tracing (lite-events) experiments.
- Analyze message sizes
  - Useful when tuning MPI env vars (eager, rendez-vous, …)
  - Found in default report.
- Put barrier in front of collectives to filter sync times.
- Use rank reordering for max. on-node communication.
  - Look for guidance and instructions in report.
  - `grid_order` utility.
- Visualizing data
  - Apprentice2 (`app2 app.exe+*/`) imbalance or activity report.
  - `pat_report -s pe=ALL` shows data by MPI rank

MPI Msg Bytes%	MPI Msg Bytes	MPI Msg Count	MsgSz <16 Count	4KiB<= MsgSz <64KiB Count	Function Caller PE=[mmm]
100.0%	34,940,767.4	8,771.9	258.6	8,513.3	Total
100.0%	34,940,647.4	8,756.9	243.6	8,513.3	MPI_ISEND
56.2%	19,622,700.0	4,837.5	56.2	4,781.2	calc2_3 shalow_
56.4%	19,718,400.0	7,200.0	2,400.0	4,800.0	pe.0
56.4%	19,699,200.0	4,800.0	0.0	4,800.0	pe.32
42.3%	14,784,000.0	4,800.0	1,200.0	3,600.0	pe.63

Total

MPI Msg Bytes%	100.0%
MPI Msg Bytes	4,465,684,125.8
MPI Msg Count	13,057.0 msgs
MsgSz <16 Count	719.0 msgs
16<= MsgSz <256 Count	28.0 msgs
256<= MsgSz <4KiB Count	0.7 msgs
4KiB<= MsgSz <64KiB Count	279.8 msgs
64KiB<= MsgSz <1MiB Count	12,029.6 msgs

MPI_Send

MPI Msg Bytes%	100.0%
MPI Msg Bytes	4,465,680,353.8
MPI Msg Count	12,318.0 msgs
MsgSz <16 Count	8.0 msgs
16<= MsgSz <256 Count	0.0 msgs
256<= MsgSz <4KiB Count	0.7 msgs
4KiB<= MsgSz <64KiB Count	279.8 msgs
64KiB<= MsgSz <1MiB Count	12,029.6 msgs

MPI_Send / LAMMPS_NS::Comm::reverse_comm

MPI Msg Bytes%	48.6%
MPI Msg Bytes	2,171,466,150.3
MPI Msg Count	6,006.0 msgs
MsgSz <16 Count	0.0 msgs
16<= MsgSz <256 Count	0.0 msgs
256<= MsgSz <4KiB Count	0.0 msgs
4KiB<= MsgSz <64KiB Count	0.0 msgs
64KiB<= MsgSz <1MiB Count	6,006.0 msgs

MPI_Send / LAMMPS_NS::Comm::reverse_comm / LAMMPS_NS::Verlet::run

MPI Msg Bytes%	48.6%
MPI Msg Bytes	2,169,218,110.3
MPI Msg Count	6,000.0 msgs
MsgSz <16 Count	0.0 msgs
16<= MsgSz <256 Count	0.0 msgs
256<= MsgSz <4KiB Count	0.0 msgs
4KiB<= MsgSz <64KiB Count	0.0 msgs
64KiB<= MsgSz <1MiB Count	6,000.0 msgs

PERFTOOLS TIME LINE

DEMO 1F_PERFTOOLS_TIMELINE

DATA AGGREGATION / TIMELINE

- By default, perftools aggregates data over time.

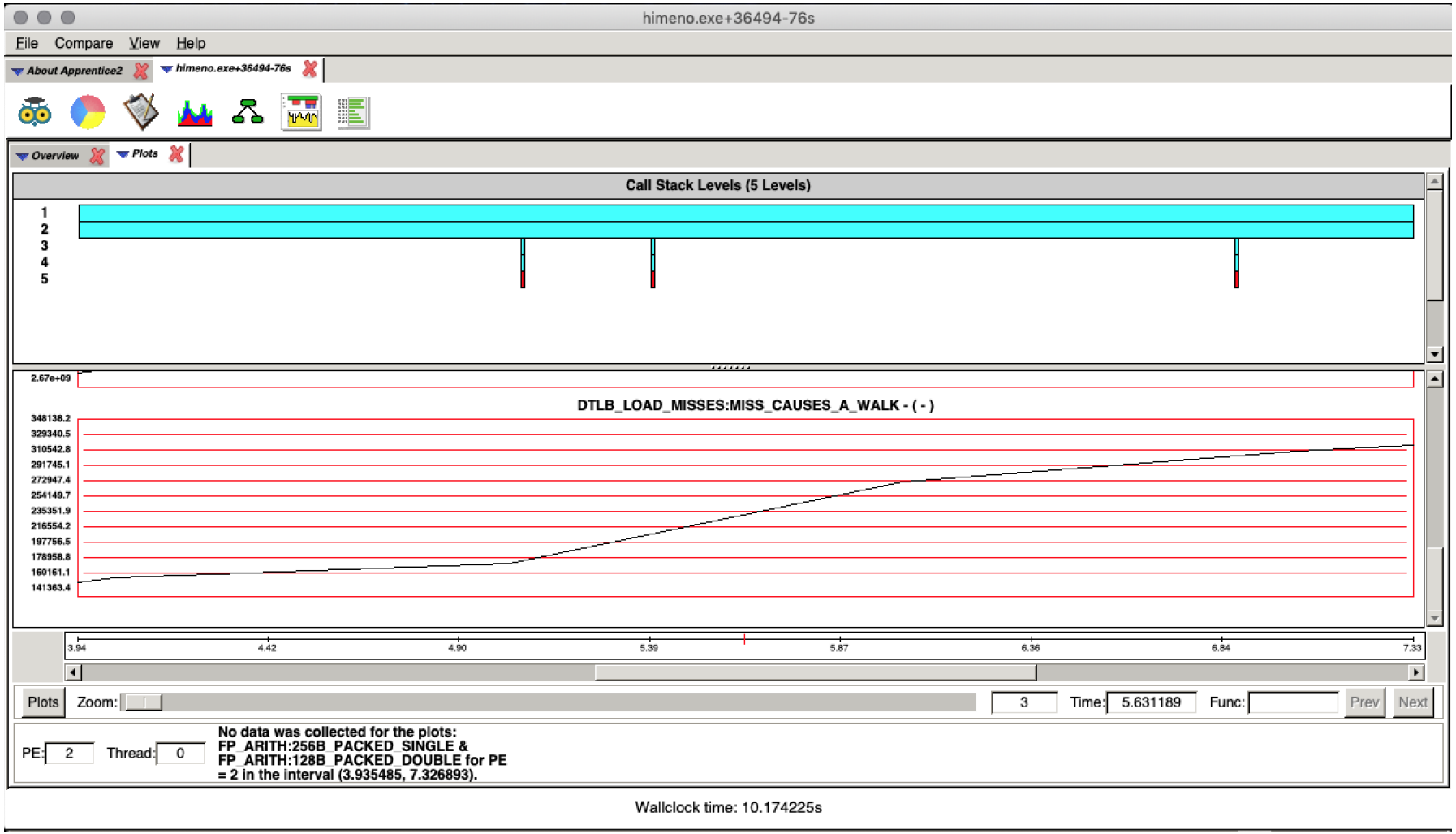
- Use `PAT_RT_SUMMARY=0` to turn off aggregation.
  - This will give you data as a function of time in apprentice2
  - Do not run large experiments with this setting because size of data files!

- Timelines can be visualized for every rank separately.

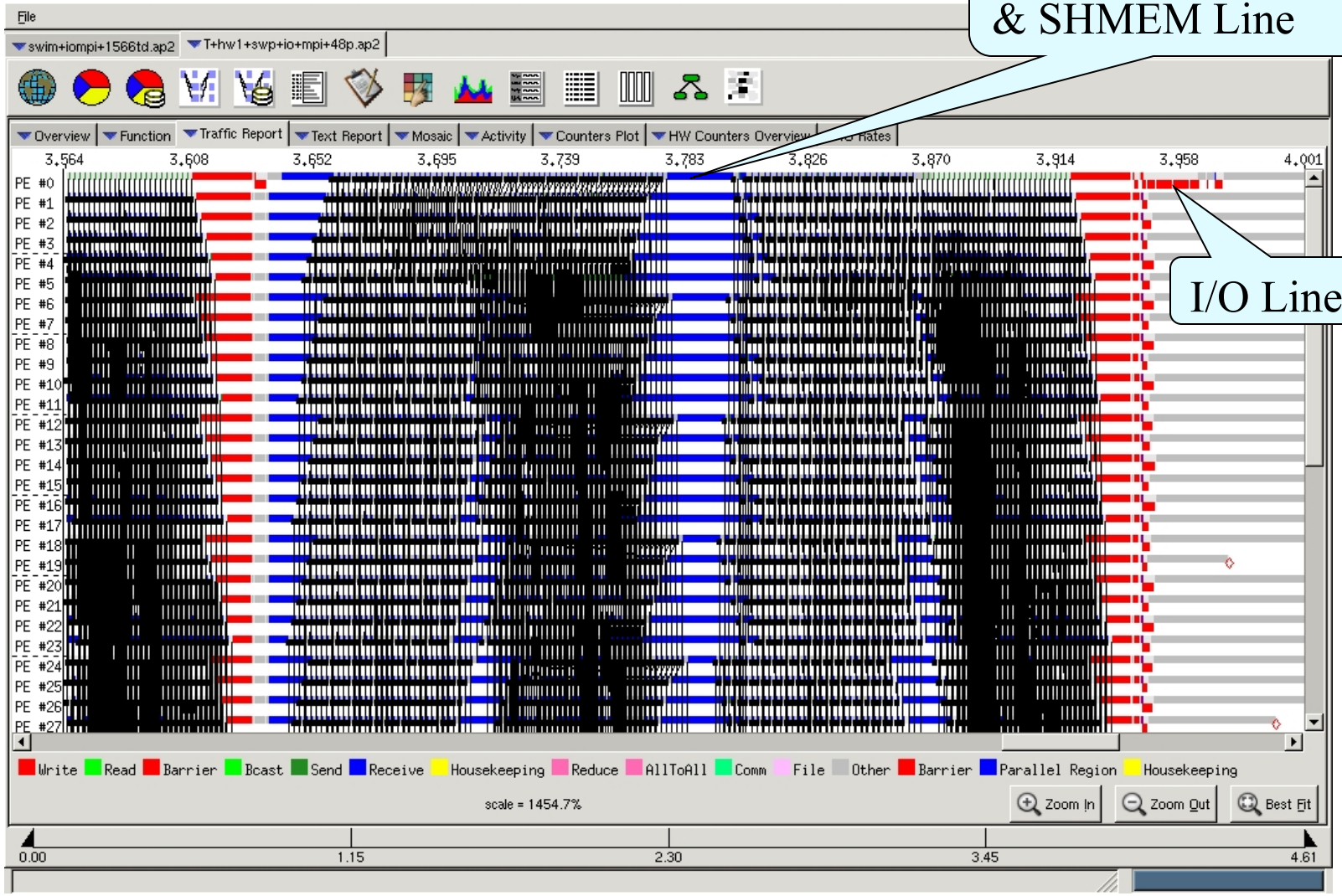
TIMELINE HINTS

- export PAT_RT_EXPFILE_PES=0,4
  - to only trace listed ranks
- pat_build -w -T sendp_ -T sendp1_ -T sendp2_ -T sendp3_ -T initcomm_ -T initmt_ -r himeno.exe
  - Trace top time-consuming user routines using –T
  - The -r flag on pat_build is useful for reporting on tracing issues
- -D trace-text-size=900 and -D varargs=y may be required to include some functions in the timeline
- pat_report -s traced_functions=show
- nm -a <exe> shows available user routine names

TIMELINE VIEW



TIMELINE VIEW



HARDWARE PERFORMANCE COUNTERS

HARDWARE PERFORMANCE COUNTERS

- Perftools supports the use of hardware counters to collect hardware events
  - All counters accessed through the PAPI interface.
  - Predefined sets of hardware counters are specified that can be instrumented for performance analysis experiment.
  - Number of simultaneous counters limited by hardware.
- Perftools provides information at the function call level on hardware features like caches, vectorization and memory bandwidth.
  - Very useful feature for deep understanding of application performance bottlenecks.
  - Impact of compiler options and code optimization.
- HWPC collection can slow down the execution notably.
  - Should be used within a tracing experiment only for a small set of functions or ideally through an automatic performance analysis.

WHEN TO COLLECT HARDWARE PERFORMANCE COUNTERS

- Use to understand the causes of a bottleneck.
- Default set of CPU counters are already collected for whole program
  - Used to present memory and vector summary metrics
- To collect performance counters
  - Set `PAT_RT_PERFCTR` environment variable to list of events or group prior to execution.
    (Or use `-Drtenv=PAT_RT_PERFCTR=<event list> | <group>` for `pat_build`.
    Environment variable has priority.)
- Run the following utility on a compute node to get list of events for a processor:
  - `papi_native_avail`
  - `papi_avail`
- Use pat_help to see counter groups and derived metrics
  - `$> pat_help counters processor_type deriv`
  - Example: `$> pat_help counters rome deriv`

```
Total
-------------------------------------------------------------------------------
Thread Time                                                  9.951056 secs
UNHALTED_REFERENCE_CYCLES                               20,395,627,382
CPU_CLK_THREAD_UNHALTED:THREAD_P                        32,834,506,946
INST_RETIRED:ANY_P                                     105,953,203,644
RESOURCE_STALLS:ANY                                      4,369,240,877
OFFCORE_RESPONSE_0:ANY_REQUEST:L3_MISS_LOCAL               274,949,826
OFFCORE_RESPONSE_1:ANY_REQUEST:L3_MISS_MISS_REMOTE_HOP1_DRAM   114,681
CPU_CLK                                  3.33GHz
CPU CLK Boost                                               1.61 X
Resource stall cycles / Cycles                             13.3%
Memory traffic GBytes                    1.769G/sec        17.60 GB
Local Memory traffic GBytes              1.768G/sec        17.60 GB
Remote Memory traffic GBytes             0.738M/sec         0.01 GB
Memory Traffic / Nominal Peak                               1.4%
Retired Inst per Clock                                      3.23
```

HARDWARE PERFORMANCE COUNTERS

- For AMD Rome `PAT_RT_PERFCTR` note the following counters:
  - `PAPI_L2_DCM (L2 data cache misses),`
  - `PAPI_L2_DCR (L2 data cache read attempts),`
  - `PAPI_L2_DCH (L2 data cache hits)`
  - `PAPI_FP_OPS (floating point operations)`
  - `PAPI_TOT_CYC (total cycles)`
  - `stalls (group of stalls counters)`

PERFORMANCE ANALYSIS ON GPUS

Performance Analysis Tools (PAT)

PERFORMANCE ANALYSIS ON GPUS

- You can use the cross-platform perftools capabilities you are familiar with on ARCHER2 to get whole-program analysis at scale with some GPU information

- For deep-dive of individual GPU behaviour (one rank) look at ROCm tools (rocprof)

GENERATE AN EVENT PROFILE FOR GPU EXPERIMENTS

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

- Assuming `perftools-base is loaded`, load `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.

EVENT TRACING USING PERFTOOLS FOR GPU (HIP)

```
> module load PrgEnv-gnu
> module load craype-x86-trento craype-accel-amd-gfx90a rocm/5.2.3
> module load perftools-base
> module load perftools
```

```
> make clean; make
> pat_build -u -g hip app.exe
```

- This generates a new executable `app.exe+pat` and preserves `app.exe`.
- Traces HIP functions calls.

```
> srun -n 8 ./app.exe+pat
> pat_report -v -T -o myrep.trace.rpt app.exe+pat+*/
```

- Running the `app.exe+pat` creates a `app.exe+pat+*/` directory.
- `pat_report` reads that directory and prints human-readable performance data.

OUTPUT: EVENT TRACING, PERFTOOLS FOR GPU (HIP)

pat_report –v app.exe+pat+*/
(or add `–O accelerator` for only HIP statistics)

HIP routines

```
Table 1:  Profile by Function Group and Function
 Time% |     Time | Imb. |  Imb. |   Calls | Group
       |          | Time | Time% |         |  Function
       |          |    |     |       |   Thread=HIDE
 100.0% | 1.055954 |  -- |   -- | 1,255.0 | Total
|-----------------------------------------------------------------------
|  99.9% | 1.054614 |  -- |   -- |   808.0 | HIP
||----------------------------------------------------------------------
||  88.1% | 0.930584 |  -- |   -- |   110.0 | hipDeviceSynchronize
||  11.7% | 0.123213 |  -- |   -- |   644.0 | hipMalloc
||   0.1% | 0.000684 |  -- |   -- |    18.0 | hipLaunchKernel
||   0.0% | 0.000057 |  -- |   -- |    10.0 | hipMemcpy
||   0.0% | 0.000039 |  -- |   -- |    10.0 | hipMemsetAsync
||   0.0% | 0.000012 |  -- |   -- |     4.0 | hipFree
||   0.0% | 0.000011 |  -- |   -- |     1.0 | hipKernel.count_kernel
||   0.0% | 0.000008 |  -- |   -- |     3.0 | hipGetDeviceProperties
||   0.0% | 0.000004 |  -- |   -- |     4.0 | __hipPushCallConfiguration
||   0.0% | 0.000003 |  -- |   -- |     4.0 | __hipPopCallConfiguration
||=====================================================================
|   0.1% | 0.001300 |  -- |   -- |   446.0 | ETC
||---------------------------------------------------------------------
||   0.1% | 0.001225 |  -- |   -- |     3.0 | __hip_module_dtor
||   0.0% | 0.000044 |  -- |   -- |    41.0 | hiprtcCreateProgram
||   0.0% | 0.000025 |  -- |   -- |   400.0 | call_init.part.0
||   0.0% | 0.000004 |  -- |   -- |     1.0 | __hip_module_ctor
||   0.0% | 0.000002 |  -- |   -- |     1.0 | __hip_register_globals
||=====================================================================
|   0.0% | 0.000040 |  -- |   -- |     1.0 | USER
||---------------------------------------------------------------------
||   0.0% | 0.000040 |  -- |   -- |     1.0 | main
|=======================================================================
|=============================================================================
```

```
Table 2:  Profile of maximum function times
 Time% |     Time | Imb. |  Imb. | Function
       |          | Time | Time% |  Thread=HIDE
|-------------------------------------------------------------
| 100.0% | 0.930584 |  -- |   -- | hipDeviceSynchronize
|  13.2% | 0.123213 |  -- |   -- | hipMalloc
|   0.1% | 0.001225 |  -- |   -- | __hip_module_dtor
|   0.1% | 0.000684 |  -- |   -- | hipLaunchKernel
|   0.0% | 0.000057 |  -- |   -- | hipMemcpy
|   0.0% | 0.000044 |  -- |   -- | hiprtcCreateProgram
|   0.0% | 0.000040 |  -- |   -- | main
|   0.0% | 0.000039 |  -- |   -- | hipMemsetAsync
|   0.0% | 0.000025 |  -- |   -- | call_init.part.0
|   0.0% | 0.000012 |  -- |   -- | hipFree
|   0.0% | 0.000011 |  -- |   -- | hipKernel.count_kernel
|   0.0% | 0.000008 |  -- |   -- | hipGetDeviceProperties
|   0.0% | 0.000004 |  -- |   -- | __hip_module_ctor
|   0.0% | 0.000004 |  -- |   -- | __hipPushCallConfiguration
|   0.0% | 0.000003 |  -- |   -- | __hipPopCallConfiguration
|   0.0% | 0.000002 |  -- |   -- | __hip_register_globals
|=============================================================
```

```
CrayPat/X:  Version 22.06.0 Revision 4b5ab6256  05/21/22 02:03:49
Number of PEs (MPI ranks):    1
Numbers of PEs per Node:      1
Numbers of Threads per PE:    2
Number of Cores per Socket:  64
Execution start time:  Sun Feb 12 10:34:24 2023
System name and speed:  nid007295  2.010 GHz (nominal)
AMD   Trento             CPU  Family: 25  Model: 48  Stepping:  1
```

General job information

EVENT TRACING USING PERFTOOLS FOR GPU (MPI+OPENMP OFFLOAD)

```
> module load PrgEnv-cray
> module load craype-x86-trento craype-accel-amd-gfx90a rocm
> module load perftools-base
> module load perftools
```

```
> make clean; make
> pat_build -u -g mpi,omp app.exe
```

- This generates a new executable `app.exe+pat` and preserves `app.exe`.
- Traces MPI functions calls, OpenMP offload directives and functions defined in the program source files.

```
> srun -n 8 ./app.exe+pat
> pat_report -T -o myrep.trace.rpt app.exe+pat+*/
```

- Running the `app.exe+pat` creates a `app.exe+pat+*/` directory.
- `pat_report` reads that directory and prints human-readable performance data.

EVENT TRACING WITH PERFTOOLS FOR GPU (MPI+OPENMP OFFLOAD)

GPU TIMELINE

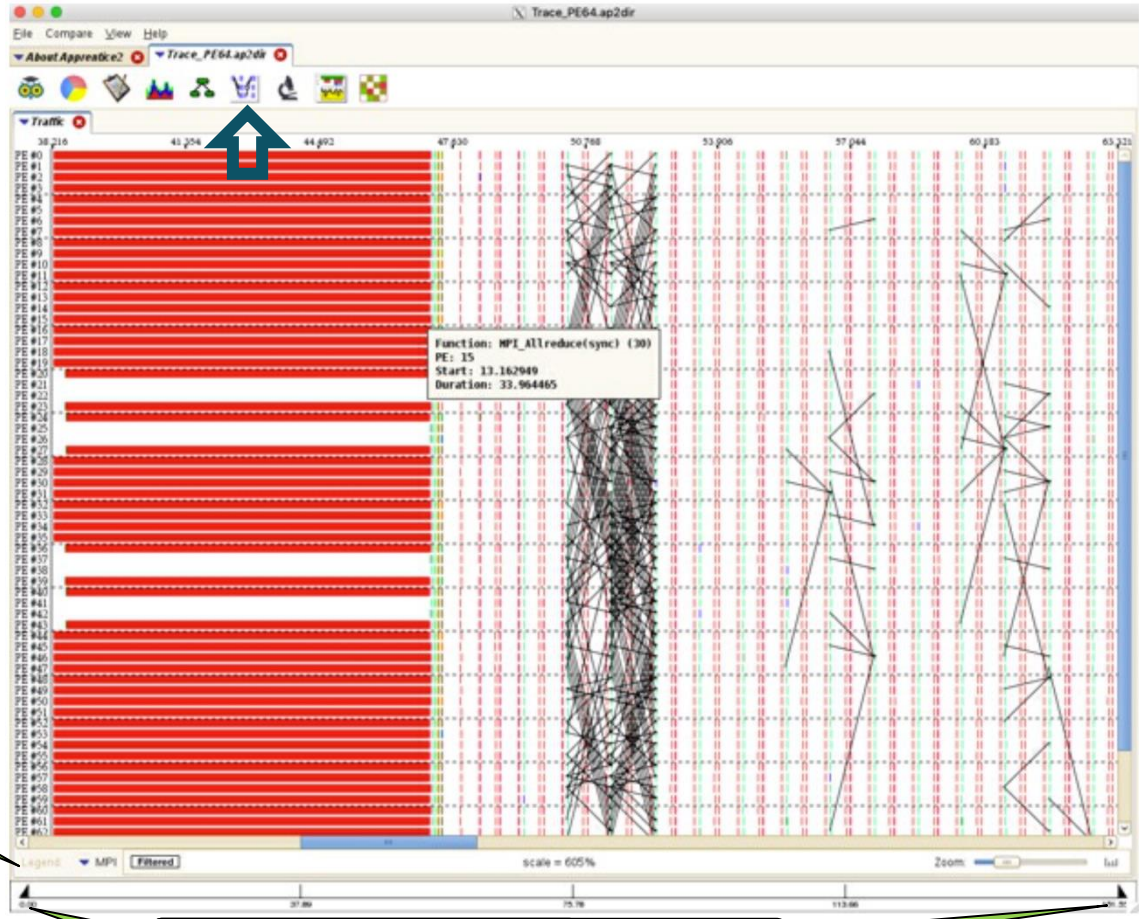
Perftools timeline

TIMELINE 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 timeline
  - GPU timeline with concurrent activity on the CPU (host) and GPU (accelerator)
- NOTE: GPU timeline requires a new ROCM (5.5.1+)
  - `module use /pfs/lustrep2/projappl/project_462000125/samantao-public/mymodules`
  - `module load rocm/5.5.3`
  - Make sure you include and link the correct ROCM during the building of the application, e.g. by adding `-I${ROCM_PATH}/include` and `-L${ROCM_PATH}/lib -lamdhip64`

TRAFFIC TIMELINE 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



Legend

Can change time interval of analysis

GPU TIMELINE 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

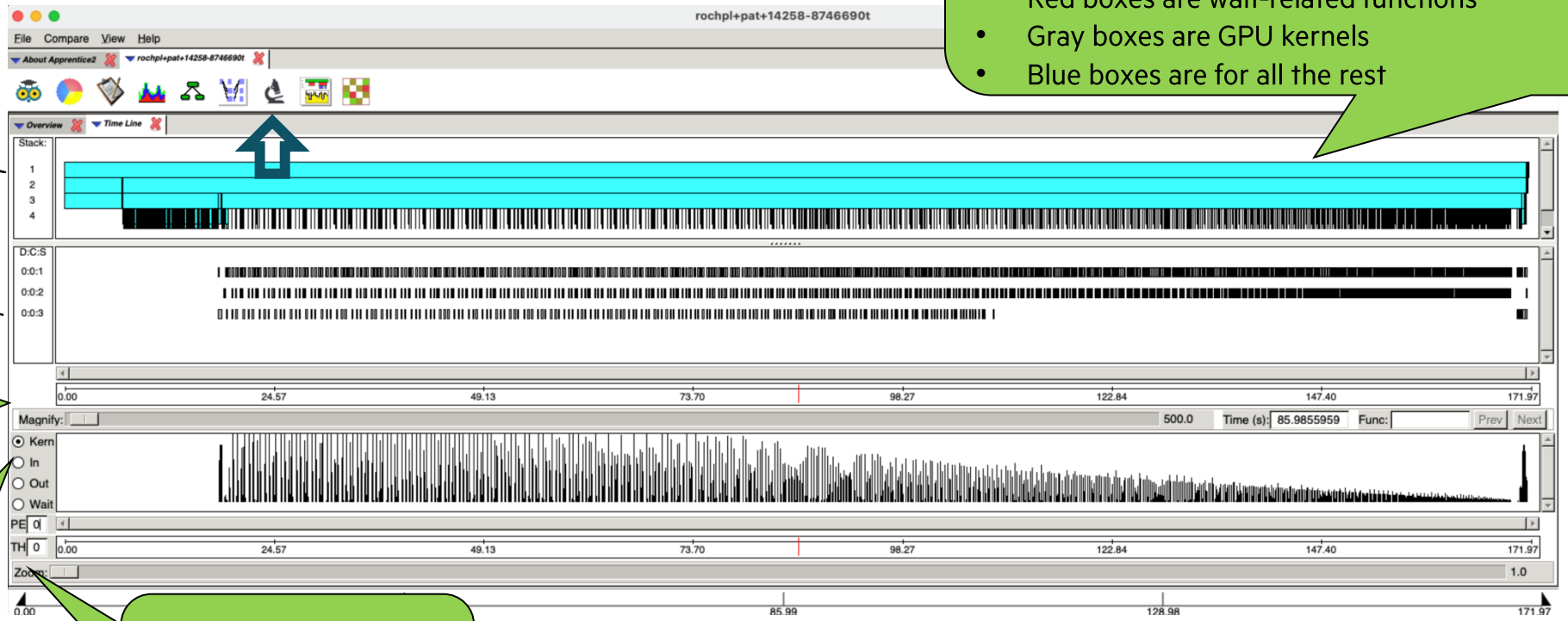
CPU stack, starting with 1 (main) at the top

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)



REVEAL

Compiler feedback and code restructuring assistant

DEMO 2_REVEAL+PERFTOOLS

MOTIVATION: COMPILER LISTINGS (CCE)

- Much information produced by compiler.
  - Can be listed together with source code for better clarity.
  - Use `-fsave-loopmark` for C and `-rmd` for Fortran (`-hlist=a` for C/C++ from cce/*classic modules.) generates an `*.lst` file for every source code file.
  - A + sign indicates that more information can be found after the routine definition.
  - Can also be inspected in Reveal with a corresponding program library.

```
%%%    L o o p m a r k    L e g e n d    %%%

Primary Loop Type        Modifiers
------- ---- ----        ---------
A - Pattern matched      a - atomic memory operation
                         b - blocked
C - Collapsed            c - conditional and/or computed
D - Deleted
E - Cloned
F - Flat - No calls      f - fused
G - Accelerated          g - partitioned
I - Inlined              i - interchanged
M - Multithreaded        m - partitioned
                         n - non-blocking remote transfer
                         p - partial
R - Rerolling            r - unrolled
                         s - shortloop
V - Vectorized           w - unwound

+ - More messages listed at end of listing
-----------------------------------------
```

```
210.  + 1----------------<   for(i=0 ; i<imax ; ++i)
211.  + 1 2--------------<     for(j=0 ; j<jmax ; ++j)
212.    1 2 Vr2---------<       for(k=0 ; k<kmax ; ++k){
213.    1 2 Vr2                   a[0][i][j][k]=1.0;
214.    1 2 Vr2                   a[1][i][j][k]=1.0;
215.    1 2 Vr2                   a[2][i][j][k]=1.0;
216.    1 2 Vr2                   a[3][i][j][k]=1.0/6.0;
217.    1 2 Vr2                   b[0][i][j][k]=0.0;
218.    1 2 Vr2                   b[1][i][j][k]=0.0;
219.    1 2 Vr2                   b[2][i][j][k]=0.0;
220.    1 2 Vr2                   c[0][i][j][k]=1.0;
221.    1 2 Vr2                   c[1][i][j][k]=1.0;
222.    1 2 Vr2                   c[2][i][j][k]=1.0;
223.    1 2 Vr2 A------<>         p[i][j][k]=(float)((i+it)*(i+it))/(float)((mx-1)*(mx-1));
224.    1 2 Vr2 A------<>         wrk1[i][j][k]=0.0;
225.    1 2 Vr2 A------<>         wrk2[i][j][k]=0.0;
226.    1 2 Vr2 A------<>         bnd[i][j][k]=1.0;
227.    1 2 Vr2--------->>>     }
228.                          }
```

```
CC-6294 CC: VECTOR initmt, File = himeno.c, Line = 210
  A loop was not vectorized because a better candidate was found at line 212.

CC-6294 CC: VECTOR initmt, File = himeno.c, Line = 211
  A loop was not vectorized because a better candidate was found at line 212.

CC-6005 CC: SCALAR initmt, File = himeno.c, Line = 212
  A loop was unrolled 2 times.

CC-6204 CC: VECTOR initmt, File = himeno.c, Line = 212
  A loop was vectorized.
```

INPUT TO REVEAL: PROGRAM LIBRARY

```
> ftn -O3 -hpl=my_program.pl -c my_program_file1.f90
> cc -O3 -fcray-program-library-path=my_program.pl -c my_program_file2.c
```

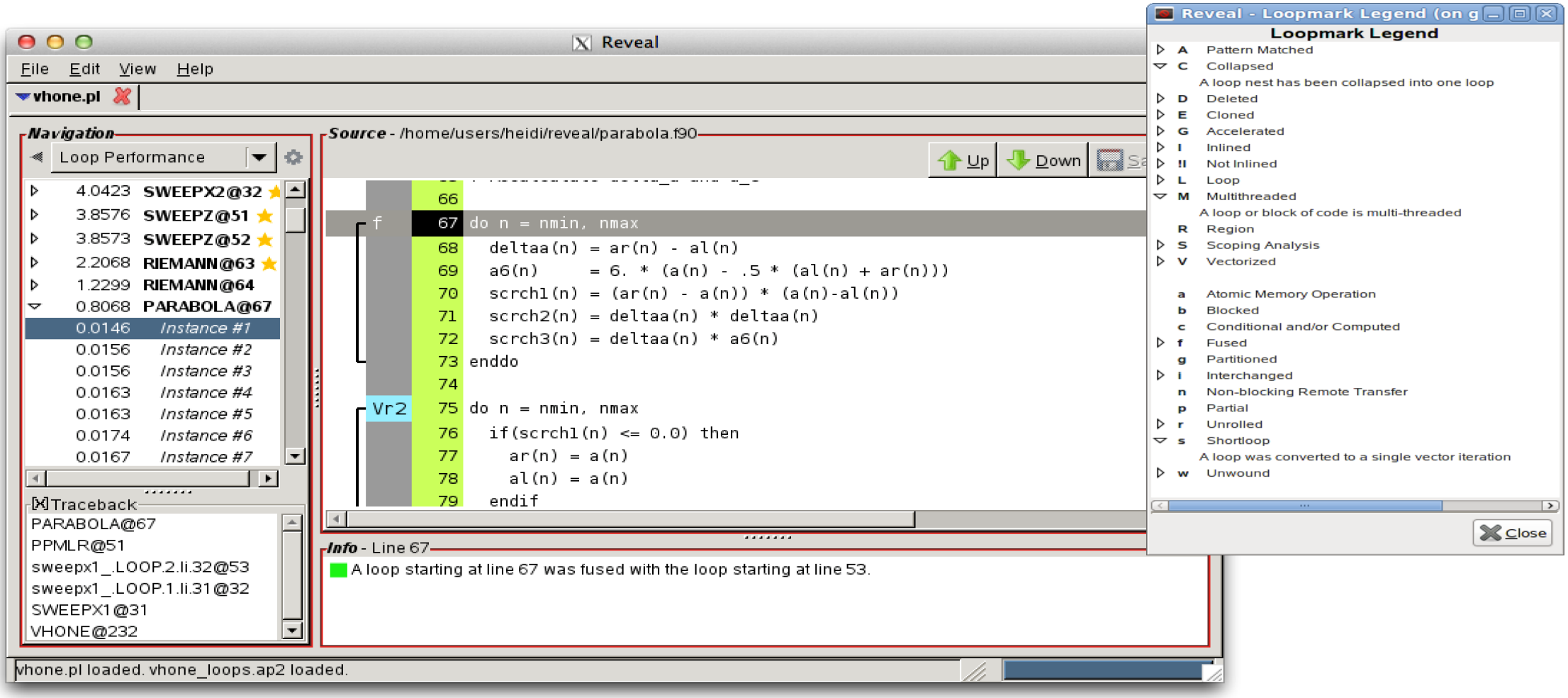
- Recompile only sources to generate program library `my_program.pl`
- The program library is most useful when generated from fully optimized code.
- Use absolute paths to specify the program library if necessary.
- Use `-hpl=my_program.pl` for C/C++ for cce/*classic modules.

```
> reveal my_program.pl &
```

- Optionally, collect loop work estimates in a separate experiment and load it with the program library or directly from the tool.
- Note that `-hprofile_generate` option disables OpenMP and significant compiler loop restructuring optimizations except for vectorization, which is why Cray recommends generating this data separately from generating the program library.
- Desktop client installer: **/opt/cray/pe/perftools/<version>/share/desktop_installers/**

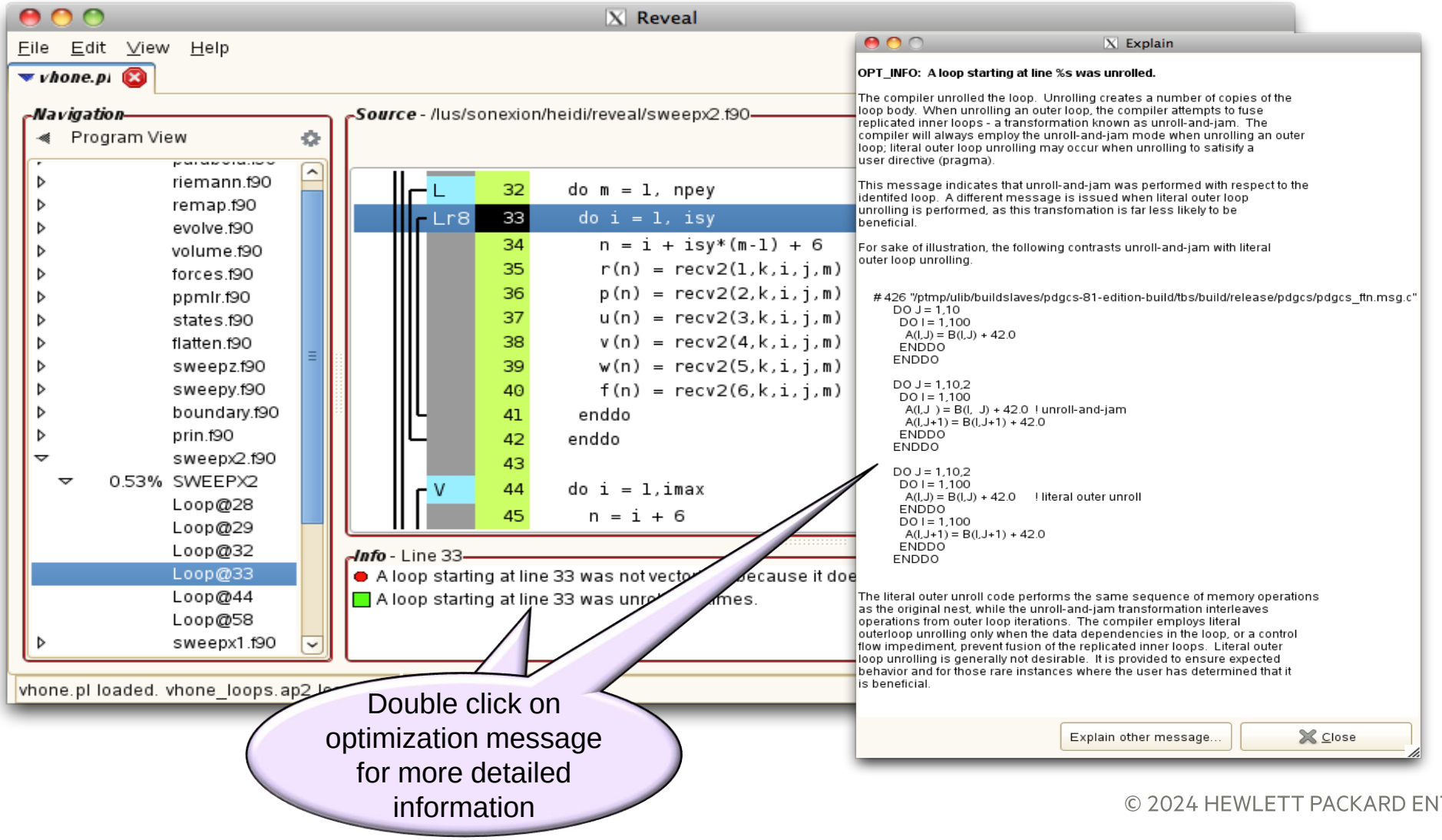
REVEAL

- View source, performance, and optimization information at the same time.



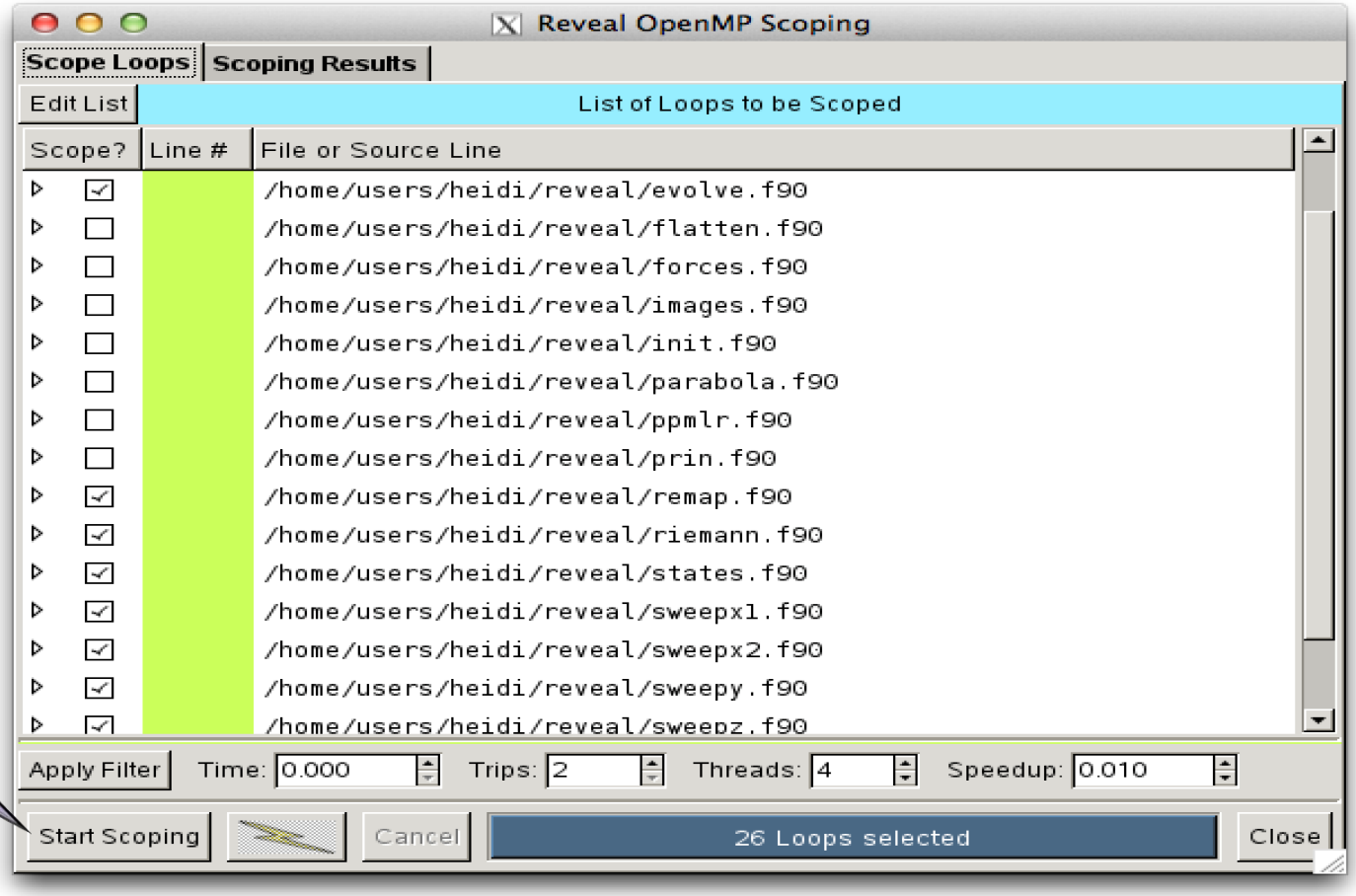
REVEAL

- Access Cray compiler message information



REVEAL

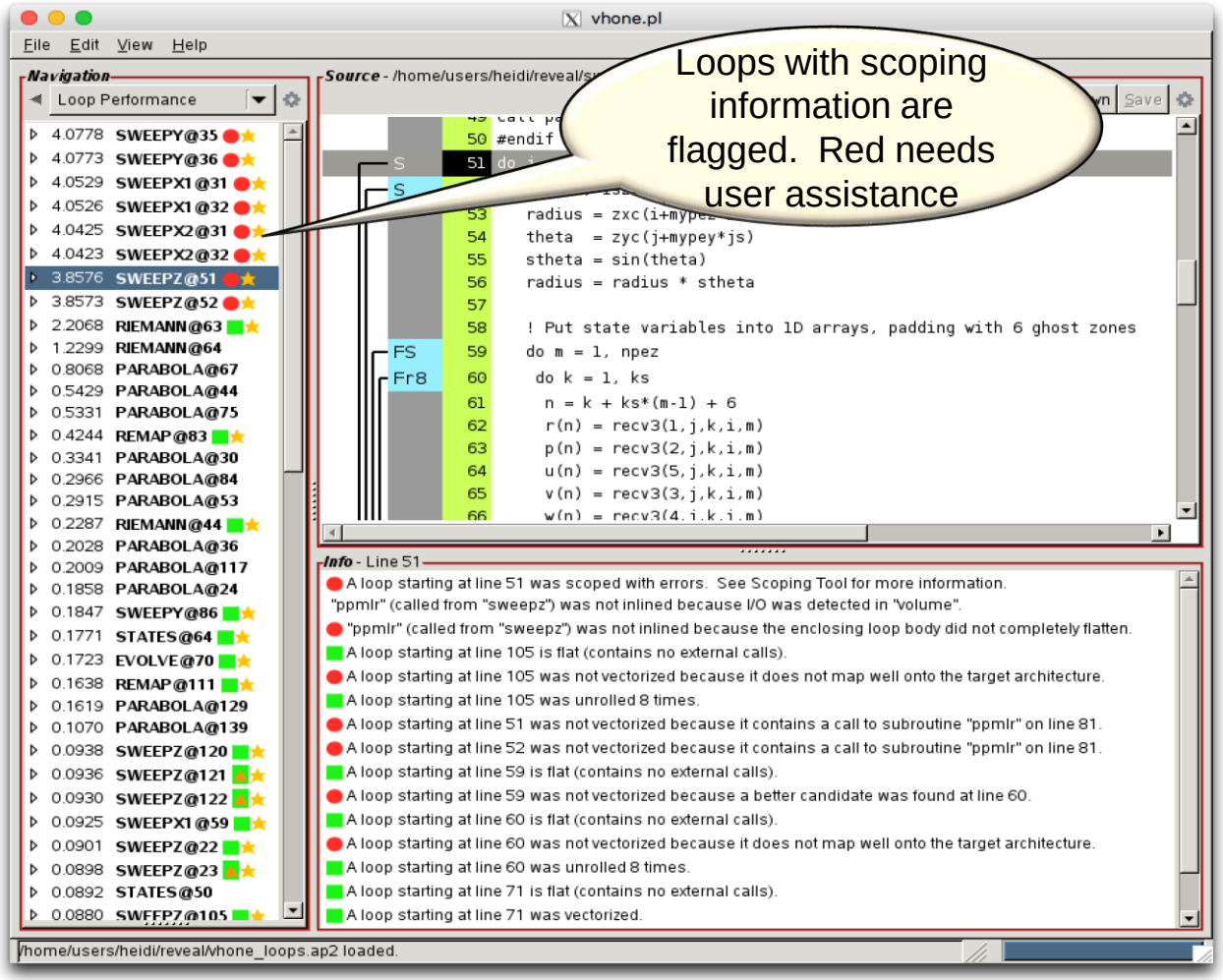
- Scope selected loop(s)

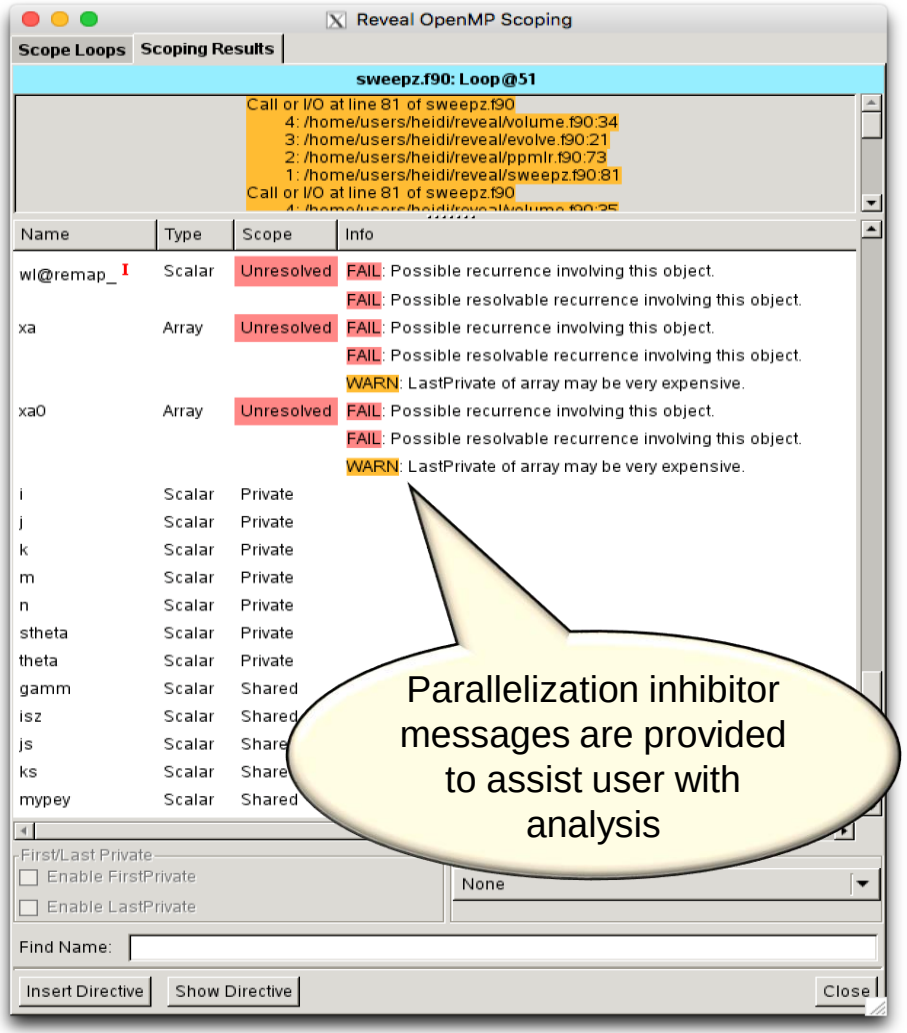


- Trigger dependence analysis
- scope loops above given threshold

REVEAL

- Review scoping results.





REVEAL

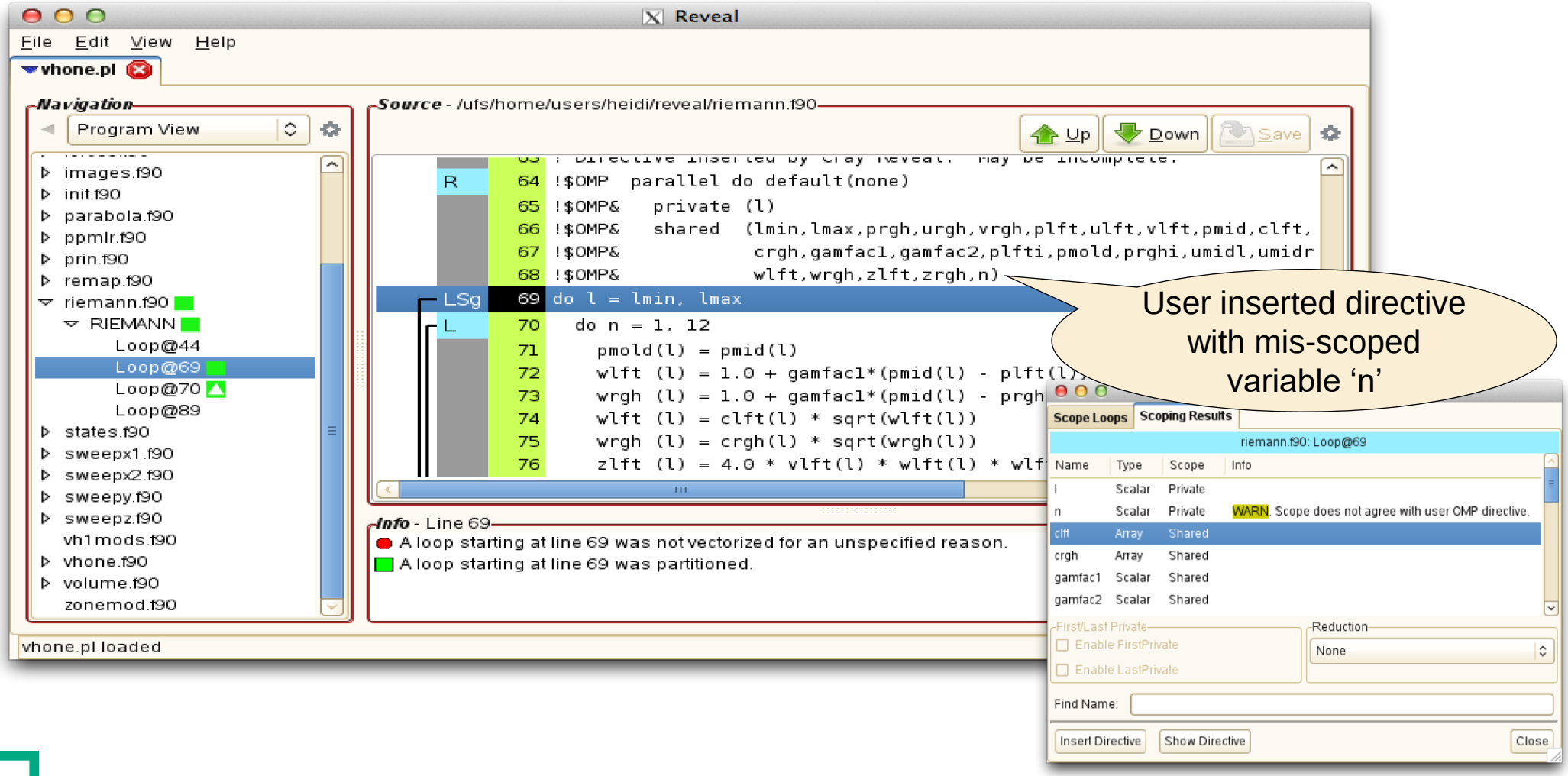
- Generate OpenMP directives.

```
! Directive inserted by Cray Reveal.  May be incomplete.
!$OMP  parallel do default(none)                                          &
!$OMP&   unresolved (dvol,dx,dx0,e,f,flat,p,para,q,r,radius,svel,u,v,w,   &
!$OMP&              xa,xa0)                                               &
!$OMP&   private (i,j,k,m,n,$$_n,delp2,delp1,shock,temp2,old_flat,        &
!$OMP&              onemfl,hdt,sinxf0,gamfac1,gamfac2,dtheta,deltx,fractn, &
!$OMP&              ekin)                                                 &
!$OMP&   shared  (gamm,isy,js,ks,mypey,ndim,ngeomy,nlefty,npey,nrighty, &
!$OMP&              recv1,send2,zdy,zxc,zya)
do k = 1, ks
 do i = 1, isy
   radius = zxc(i+mypey*isy)

   ! Put state variables into 1D arrays, padding with 6 ghost zones
   do m = 1, npey
    do j = 1, js
     n = j + js*(m-1) + 6
     r(n) = recv1(1,k,j,i,m)
     p(n) = recv1(2,k,j,i,m)
     u(n) = recv1(4,k,j,i,m)
     v(n) = recv1(5,k,j,i,m)
     w(n) = recv1(3,k,j,i,m)
     f(n) = recv1(6,k,j,i,m)
    enddo
   enddo

   do j = 1, jmax
     n = j + 6
```

Reveal generates OpenMP directive with illegal clause marking variables that need addressing

REVEAL

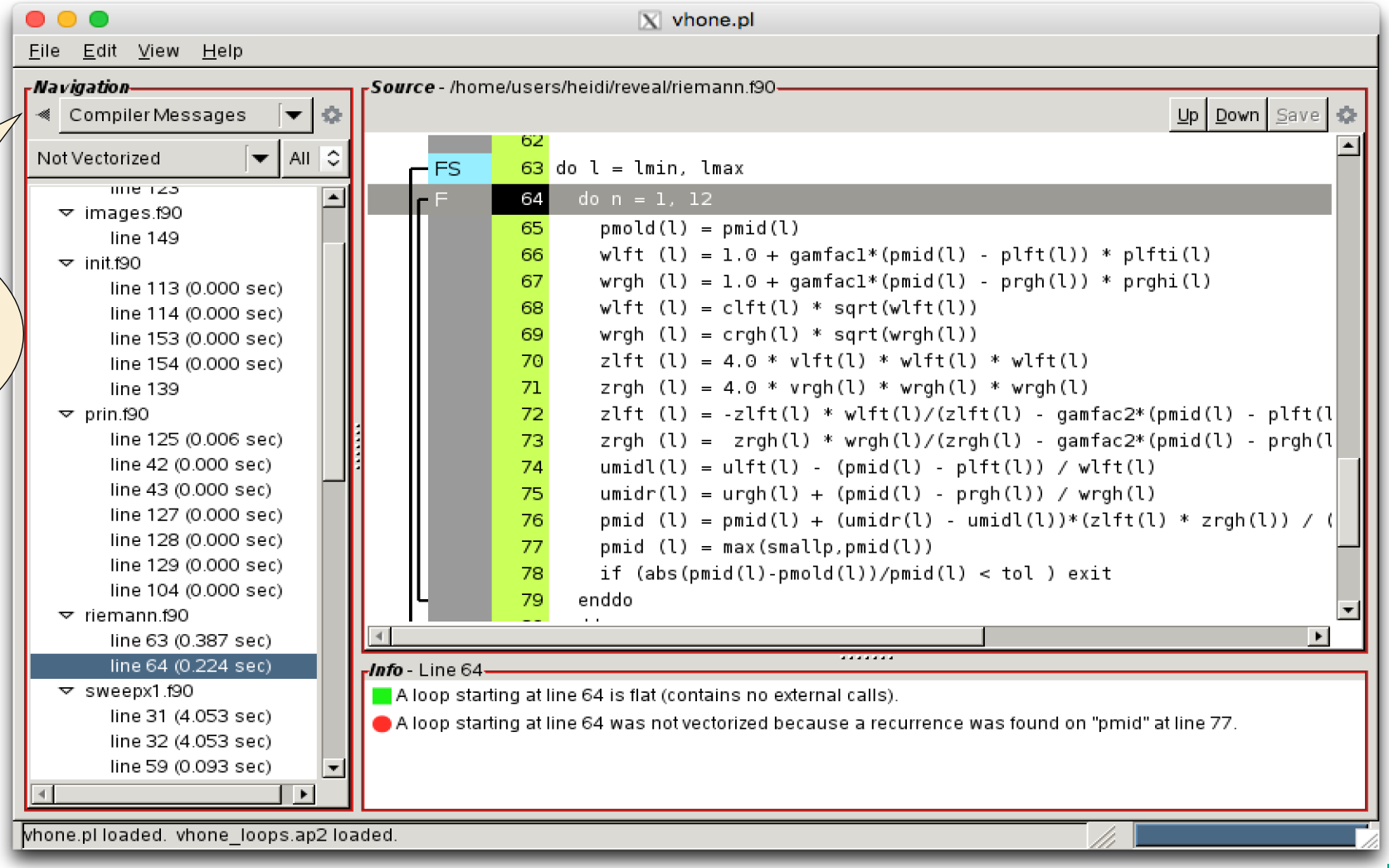
- Validate user inserted directive.



REVEAL

- Look for vectorization opportunities.

Choose “Compiler Messages” view to access message filtering, then select desired type of message



OPENMP DATA COLLECTION AND REPORTING

- For OpenMP programs
  - Perftools can measure the overhead incurred by entering and leaving parallel regions and work-sharing constructs within parallel regions
  - Show per-thread timings in addition to other data.
  - Calculate the load balance across threads for such constructs.
- For programs that use both MPI and OpenMP
  - Profiles by default show the load balance over PEs of the average time in the threads for each PE
  - But you can also see load balances for each programming model separately.
- Options for `pat_report`
  - `profile_pe.th`
    - Imbalance based on the set of all threads in the program
  - `profile_pe_th` (default view)
    - Highlights imbalance across MPI ranks
    - Uses max for thread aggregation to avoid showing under-performers
    - Aggregated thread data merged into MPI rank data
  - `profile_th_pe`
    - For each thread, show imbalance over MPI ranks
    - Example: Load imbalance shown where thread 4 in each MPI rank didn't get much work

OPENMP DATA COLLECTION AND REPORTING

- OpenMP support needs to be enabled during compilation.
- OpenMP tracing calls inserted by default when perftools is loaded.

```
Table 1:  Profile by Function Group and Function

  Time% |      Time |      Imb. |  Imb. |  Calls |Group
        |           |      Time | Time% |        | Function
        |           |           |       |        |  PE=HIDE
        |           |           |       |        |   Thread=HIDE

 100.0% | 2.452453 |       -- |    -- | 1426.8 |Total
|-------------------------------------------------------------------------
|  96.9% | 2.377154 |       -- |    -- |  309.8 |USER
||------------------------------------------------------------------------
||  82.1% | 2.013394 | 0.027282 |  1.8% |  100.0 |work.LOOP@li.533
||  10.6% | 0.259470 | 0.000282 |  0.1% |    1.0 |exit
||   2.4% | 0.057711 | 0.000562 |  1.3% |    1.0 |initializeMatrix
||   1.0% | 0.024130 | 0.000313 |  1.7% |    1.0 |setPEsParams.SINGLE@li.355
||========================================================================
|   1.6% | 0.039963 |       -- |    -- |  909.0 |MPI
||------------------------------------------------------------------------
||   1.6% | 0.039247 | 0.079519 | 89.3% |  301.5 |MPI_Wait
||========================================================================
|   1.2% | 0.029108 |       -- |    -- |  101.0 |OMP
||------------------------------------------------------------------------
||   1.2% | 0.029058 | 0.012000 | 39.0% |  100.0 |work.REGION@li.492(ovhd)
|=========================================================================
```

Work sharing construct

Region

Overhead

```
Table 2:  Load Imbalance by Thread

     Max. |      Imb. |  Imb. |Thread
     Time |      Time | Time% | PE=HIDE

 2.452470 | 0.316486 | 17.2% |Total
|-----------------------------------
| 2.453287 | 0.000817 |  0.0% |thread.0
| 2.078727 | 0.036293 |  2.3% |thread.2
| 2.074969 | 0.048712 |  3.1% |thread.1
| 2.066243 | 0.043468 |  2.8% |thread.3
|===================================
```

QUESTIONS?

Ian.Cockshott@hpe.com

