Interpreting perftools Performance Data

Heidi Poxon

Sr. Principal Engineer

Cray Inc.

April 2019

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publicly announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA and YARCDATA. The following are trademarks of Cray Inc.: CHAPEL, CLUSTER CONNECT, CLUSTERSTOR, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE, REVEAL. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used on this website are the property of their respective owners.

Some Useful Experiments

- Identify slowest areas and notable bottlenecks of a program
  - Use `perftools-lite`
  - Good for examining performance characteristics of a program and for scaling analysis

- Focus on MPI communication
  - Use `perftools-lite` first to determine if MPI time is dominant or if there is a load imbalance between ranks
  - Use `perftools (pat_build –g mpi)` to collect more detailed MPI-specific information including MPI_SYNC time to detect late arrivers to collectives
  - Good for identifying source of imbalance and scaling analysis at targeted final job size

- Focus on loop optimization
  - Use `perftools-lite-loops`
  - Use `perftools-lite-hbm` for memory traffic analysis
  - Good for vectorizing, parallelizing and cache optimization

Cray Compiler Listings

- CCE provides loopmark, cross-references, compile options, and optimization messages in easy-to-read text files
- Just add the following flag to the application's Makefile:

```
-h list=a …
```

- Tip: For additional information on restructuring and optimization changes made by CCE, try `–h list=d` for decompiled code
- Use with perftools to understand top time consuming loop optimization information

Get Additional Information Without Re-running

- Generate full report
  - `user@login> pat_report my_data_directory+12s/ > rpt`

- Generate report with call tree (or by callers)
  - `user@login> pat_report –O calltree+src`

- Generate report without pruning
  - `user@login> pat_report –P`

- Show each MPI rank or each OpenMP thread in report
  - `user@login> pat_report –s pe=ALL`
  - `user@login> pat_report –s th=ALL`

Don’t See an Expected Function?

- To make the profile easier to interpret, samples are attributed to a caller that is either a user defined function, or a library function called directly by a user defined function

- To disable this adjustment, and show functions actually sampled, 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`’

Don’t See an Expected Function? (continued)

Why don’t I see a particular function in a report?

- Cray tools filter out data that may distract you
  - 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

What is ETC Group in a Report?

- When a function is called that cannot be attributed to a user-defined parent function, it gets placed in ETC
- Try '`pat_report –P`'
- Note: pat_report depends on the accuracy of the DWARF issued by the compiler

How Do I See per-Rank or per-Thread Data?

- `$ pat_report –s pe=ALL`

- `$ pat_report –s th=ALL`

How Was Data Aggregated?

- Check the Notes before each table in the text report

```
Notes for table 1:

  This table shows functions that have significant exclusive sample
    hits, averaged across ranks.

  For further explanation, see the "General table notes" below,
    or use:  pat_report -v -O samp_profile ...

Table 1:  Profile by Function
```

When to Collect Counters

- Use to understand the “why” of a bottleneck

- Default set of counters are collected for whole program
  - Used to present memory and vector summary metrics

- User can choose to collect per function or per region of code with PAT_region API

Performance Counters Overview

- Cray supports raw counters, derived metrics and thresholds for:
  - Processor (core and uncore)
  - Network
  - Accelerator
  - Power

- Predefined groups
  - Groups together counters for experiments

- See *hwpc, nwpc, accpc, and rapl* man pages

CrayPat Runtime Options

- Runtime controlled through PAT_RT_XXX environment variables

- See intro_craypat(1) man page

- Examples of control
  - Enable full trace
  - Change number of data files created
  - Enable collection of HW, network or power counter events
  - Enable tracing filters to control trace file size (max threads, max call stack depth, etc.)

How to Get List of Events for a Processor

- Run the following utility on a compute node:
  - papi_native_avail
  - papi_avail

- Use pat_help to see counter groups and derived metrics
  - user@login> `pat_help counters` *`processor_type`* `deriv`

- To collect performance counters
  - Set `PAT_RT_PERFCTR` environment variable to list of events or group prior to execution

Focus on Loop Optimization – Find Top Loops

- `$ module load perftools-lite-loops`
- Build program with CCE
  - Should see messages from CrayPat during build saying that it created an instrumented executable
  - Remember to add `-hlist=a` to build with CCE listing
  - Add `-hpl=/path_to_program_library/my_program.pl` if you want to use Reveal
- Run program
- Performance data sent to `STDOUT` and to directory with unique name

Example Loop Statistics

```
Inclusive and Exclusive Time in Loops
  Loop |  Loop Incl |  Loop Hit |  Loop |  Loop |  Loop |Function=/.LOOP[.]
  Incl |       Time |           | Trips | Trips | Trips | PE=HIDE
 Time% |            |           |   Avg |   Min |   Max |
|-----------------------------------------------------------------------------
| 99.4% | 333.895923 |         1 | 500.0 |   500 |   500 | les3d_.LOOP.3.li.216
| 98.7% | 331.721721 |       500 |   2.0 |     2 |     2 | les3d_.LOOP.4.li.272
| 26.5% |  89.032566 |     1,000 |  96.0 |    96 |    96 | fluxk_.LOOP.1.li.28
| 24.6% |  82.681435 |    96,000 |  97.0 |    97 |    97 | fluxk_.LOOP.2.li.29
| 24.2% |  81.356609 |     1,000 |  96.0 |    96 |    96 | fluxj_.LOOP.1.li.28
| 22.5% |  75.770180 |    96,000 |  97.0 |    97 |    97 | fluxj_.LOOP.2.li.29
| 22.5% |  75.458432 |     1,000 |  96.0 |    96 |    96 | fluxi_.LOOP.1.li.21
| 22.5% |  75.453469 |    96,000 |  96.0 |    96 |    96 | fluxi_.LOOP.2.li.22
| 18.9% |  63.574836 | 9,312,000 |  96.0 |    96 |    96 | visck_.LOOP.1.li.344
| 17.1% |  57.529187 | 9,312,000 |  96.0 |    96 |    96 | viscj_.LOOP.1.li.340
| 15.7% |  52.794857 | 9,216,000 |  97.0 |    97 |    97 | visci_.LOOP.1.li.782
|  5.0% |  16.924522 |     1,000 |  99.0 |    99 |    99 | extrapi_.LOOP.1.li.128
```

How to Identify Important Loops

- High inclusive time
- Create call tree with loops:
  - `user@login> pat_report –O calltree`

What About Memory Bandwidth?

- Phased in over perftools 7.0.0, 7.0.1, and 7.0.2 for Intel Xeon processors
- New default counter group with perftools-lite and perftools experiments
- New table for memory bandwidth by NUMA node in default lite and full reports
- Separate functionality from perftools-lite-hbm experiment which uses CCE, CrayPat, and Reveal to track memory traffic and associate with allocation sites

Example: Memory Bandwidth per NUMA

8 MPI ranks, 4 on each of 2 nodes

```
Table 1: Memory Bandwidth by Numanode

  Memory |    Local |   Remote |      Thread |   Memory |   Memory | Numanode
 Traffic |   Memory |   Memory |        Time |  Traffic |  Traffic | Node Id
  GBytes |  Traffic |  Traffic |             |   GBytes |        / | PE=HIDE
         |   GBytes |   GBytes |             |    / Sec |  Nominal |   Thread=HIDE
         |          |          |             |          |     Peak |
  39,429 |   39,429 |        0 |  990.218871 |    39.82 |    33.4% | Max of Numanode values
 |-----------------------------------------------------------------------------
 | 39,429 |   39,429 |        0 | 990.217439 |    39.82 |    33.4% | numanode.0
 ||----------------------------------------------------------------------------
 ||   39,429 | 39,429 |        0 | 990.217439 |    39.82 |    33.4% | nid.200
 ||   38,389 | 38,389 |        0 | 990.224163 |    38.77 |    32.5% | nid.205
 ||============================================================================
 | 38,857 |   38,857 |        0 | 990.218903 |    39.24 |    32.9% | numanode.1
 ||----------------------------------------------------------------------------
 ||   38,857 | 38,857 |        0 | 990.211194 |     39.24 |    32.9% | nid.200
 ||   38,528 | 38,528 |        0 | 990.226690 |     38.91 |    32.6% | nid.205
```

Examine NUMA Traffic

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
172.95	171.48	1.48	19.755654	8.75	11.4%	numanode.0
172.77	171.48	1.30	19.414237	8.90	11.6%	nid.68
172.09	170.61	1.48	19.071340	9.02	11.7%	nid.63
171.20	169.93	1.27	17.631761	9.71	12.6%	nid.62
162.51	161.07	1.43	19.675857	8.26	10.8%	nid.71
162.28	160.82	1.46	19.730793	8.22	10.7%	nid.72
161.75	160.29	1.46	19.755654	8.19	10.7%	nid.70
168.69	166.81	1.89	19.781479	8.53	11.1%	numanode.1
168.69	166.81	1.89	19.454144	8.67	11.3%	nid.62
167.74	166.03	1.71	19.476164	8.61	11.2%	nid.63
166.66	164.88	1.78	19.225409	8.67	11.3%	nid.61
161.68	160.07	1.61	19.781479	8.17	10.6%	nid.71
161.60	159.99	1.62	19.642791	8.23	10.7%	nid.70
157.32	156.01	1.31	18.036118	8.72	11.4%	nid.72

Example traffic from pure MPI run

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

Controls for Report Generation

perftools-lite:

- Optionally run pat_report on the data directory from login node
  - `export PAT_RT_REPORT_CMD=pat_report,-Q0`
  - Reduces job execution time, but disables parallel pat_report execution

perftools-lite or perftools:

- For a quick preview of performance data, use subset of data to generate a report
  - `user@login> pat_report -Q1` ← report from 1st ap2 file
  - `user@login> pat_report –Q3` ← report from 1st, middle, and last file

Controlling Instrumentation

- Record Subset of PEs during execution
  - It works again! (we found that it was broken last year)
  - Example: export PAT_RT_EXPFILE_PES=0,4,5,10

- Don’t instrument select binaries when using perftools-lite
  - Good for applications that generate test or intermediate binaries with CMake and GNU Autotools
  - Use CRAYPAT_LITE_WHITELIST for binaries you DO want instrumented

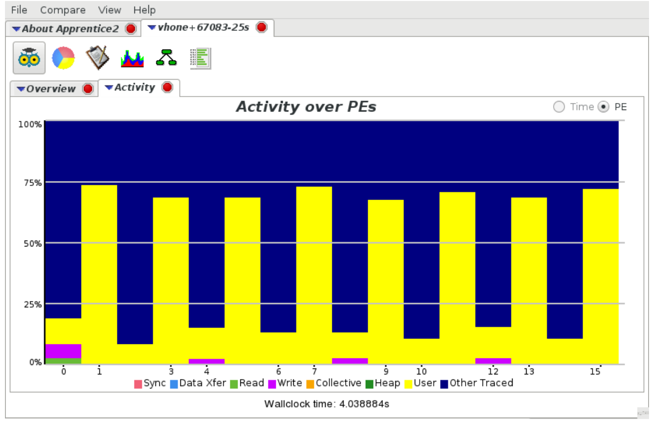
pat_run New!

Utility that allows you to profile un-instrumented, dynamically linked binaries with CrayPat!

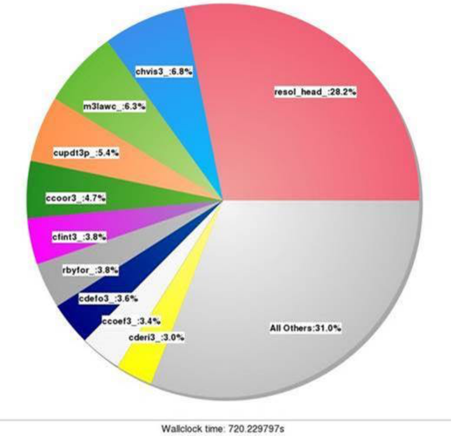
- Delivers Cray performance tools profiling information for codes that cannot easily be rebuilt
- Makes profiling possible for a wider set of HPC applications
- Available starting with perftools 7.0.1
- Initially targets Cray XC systems running CLE 6 or later

Using pat_run

- Insert before executable in run command
  - `user@login> srun –n 16 pat_run ./my_program`
  - `user@login> pat_report expdir > my_report`



- Use existing perftools capability
  - Optionally collect a different group of performance counters
    - `user@login> export PAT_RT_PERFCTR=1`
    - `user@login> aprun -n 16 pat_run ./my_program`
  - Perform other experiments, for example trace MPI routines
    - `user@login> pat_run -g mpi ./my_program`
  - Create additional views of the data with pat_report options, such as
    - `user@login> pat_report –P –O callers+src`



Example Advanced Capability

- Use pat_region API to mark regions in your code to profile
- Enable or disable specific function tracing based on name, size, …
- Choose additional predefined trace groups (blas, hdf5, ...)
- Adjust sampling rate (collect counters per sample, sample 1000 times a second, ….)
- Collect different performance counters (cache hierarchy, TLB misses, …)
- Customize data sorting, aggregation, or collection

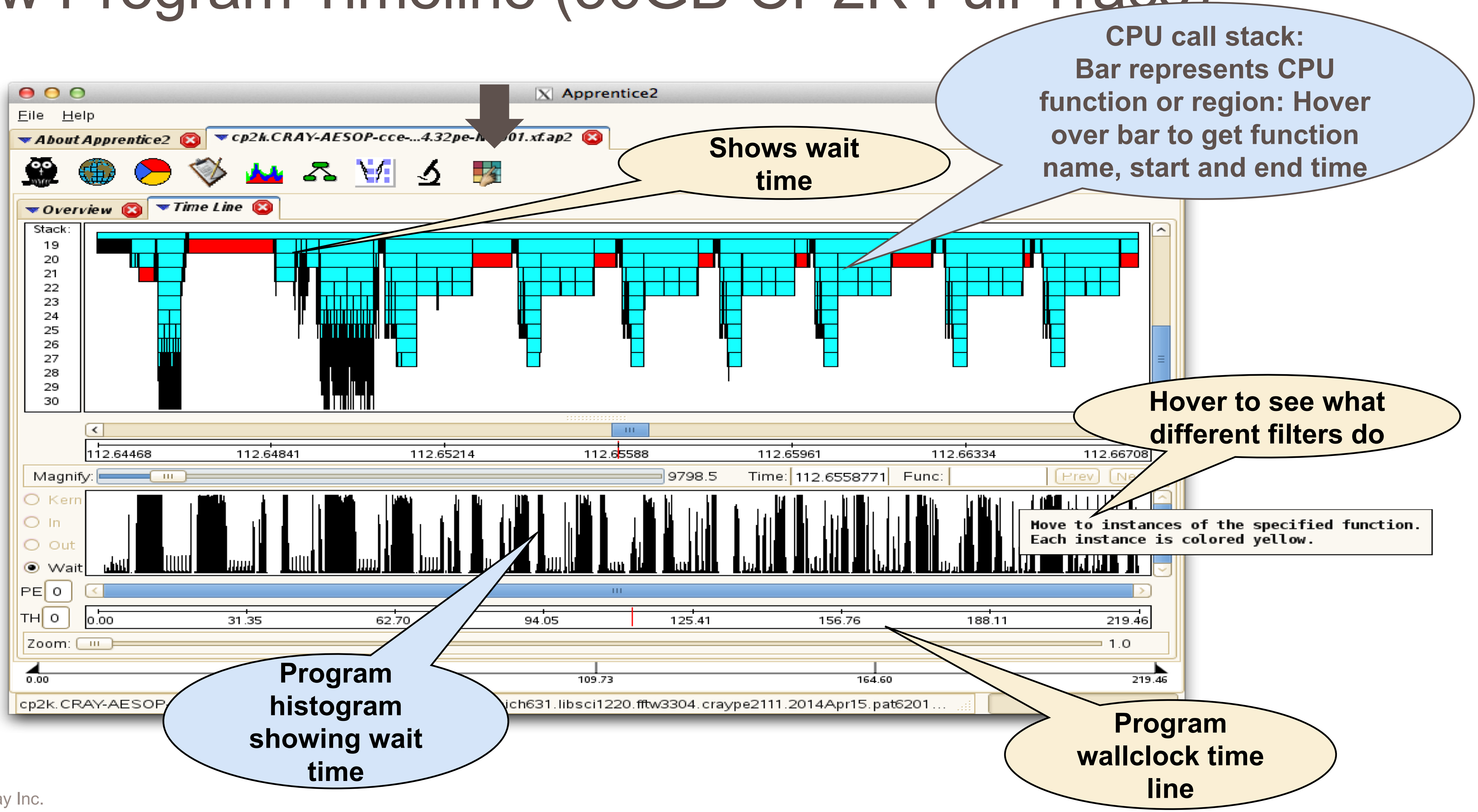
Creating a Timeline

- Can produce huge amounts of performance data
  - Adjust job size for shorter and smaller runs

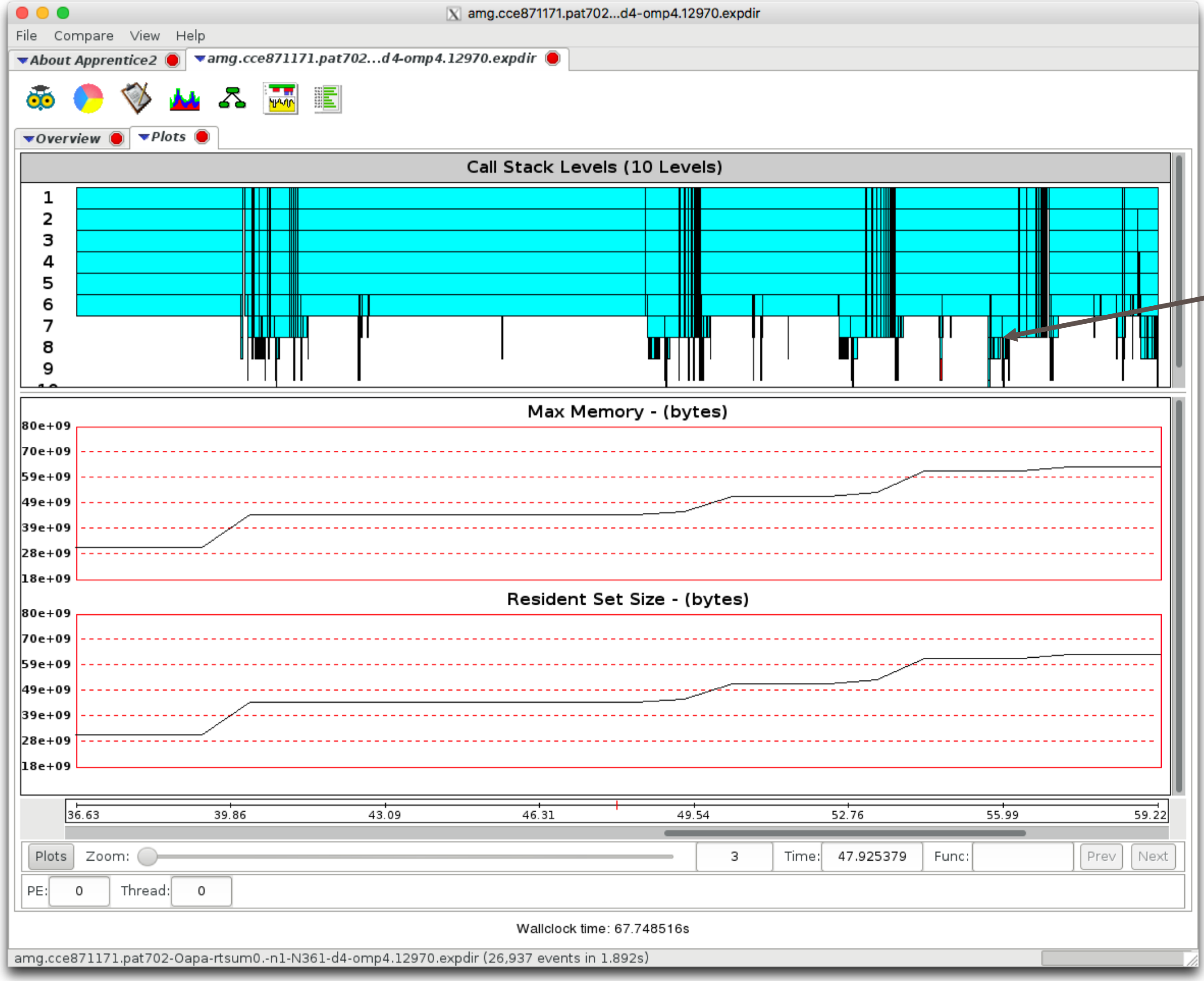
- Can be used with sampling and tracing (tracing is most common)
  - `perftools-lite-events` module
  - `pat_build –g mpi`
  - `pat_build –u –g mpi`

- To enable, set `PAT_RT_SUMMARY=0` `environment` variable at runtime

View Program Timeline (36GB CP2K Full Trace)



Memory High Water Over Time (Apprentice2)



Name:	MPI_Waitall
Start Time:	54.666761
End Time:	54.716777
Group:	MPI

Produced with:

```
pat_build ./my_program
PAT_RT_SAMPLING_DATA=memory
PAT_RT_SUMMARY=0
```

Summary of Cray Performance Tools

- Focus on whole program analysis
- Reduce the time investment associated with porting and tuning applications on new and existing Cray systems
- Provide easy-to-use interfaces complimented with a wealth of capability when you need it for analyzing the most critical production codes
- Offer analysis and recommendations that focus on areas that impact performance and scaling, such as
  - Imbalance
  - Communication overhead and inefficiencies
  - Vectorization and memory utilization efficiency