Cray Programming Environment Hack-a-Thon

Luiz DeRose, Heidi Poxon, & John Levesque
Cray Inc.

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 publically 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.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and 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 names and brands may be claimed as the property of others. Other product and service names mentioned herein are the trademarks of their respective owners.

Agenda (Tentative)

09:00 – 09:15 Introductions and goals

09:15 – 09:45 Update on Cori

09:45 – 10:45 Using CCE and Cray Performance Tools

10:45 – 11:00 Break

11:00 – 12:00 Using Reveal to add OpenMP and find vectorization opportunities

12:00 – 13:00 Lunch

13:00 – 13:15 Where to find documentation

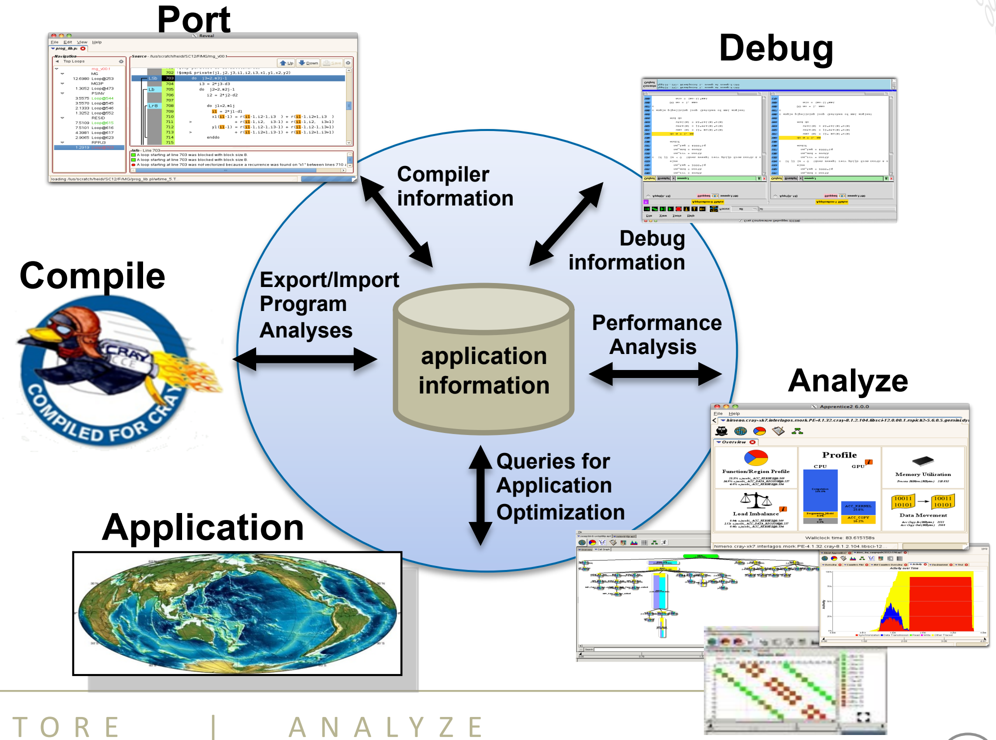
13:15 – 16:15 Profiling and tuning with Cray software

16:15 – 16:30 PE Roadmap for KNL

16:30 – 17:00 Questions / Recap

The Cray Programming Environment Mission

- Focus on **Performance** and **Programmability**
  - It is the role of the Programming Environment to **close the gap** between observed performance and achievable performance
- Support the **application development life cycle** by providing a **tightly coupled** environment with compilers, libraries, and tools that will **hide the complexity** of the system
  - Address issues of scale and complexity of HPC systems
  - Target **ease of use** with extended **functionality** and increased **automation**
  - Close **interaction with users**
    - ➢ For feedback targeting functionality enhancements



The Cray Compiling Environment

- **Cray technology focused on scientific applications**
  - Takes advantage of **automatic vectorization**
  - Takes advantage of **automatic shared memory parallelization**

- **Automatic optimizations for Cray architectures to deliver performance of a new target through simple recompile**
  - Hide system complexity

- **PGAS languages (UPC & Fortran Coarrays) fully optimized and integrated into the compiler**
  - No preprocessor involved
  - Target the network appropriately
  - Full debugger support with Allinea's DDT

- **Focus on standards for application portability and investment protection**
  - Fortran 2008 standard compliant
  - C++11 compliant (working on C++14)
  - OpenMP 4.0 compliant (working on OpenMP 4.5)
  - OpenACC 2.0
  - UPC 1.3

CCE Highlights

- **Arguably the most complete vectorization capabilities in the industry**
  - **Fully automatic loop vectorization** without the need of directives and source code modification
    - This includes **automatic outer loop vectorization**, which is unique in the industry
- **Focus on real applications, instead of just benchmarks**
- **Compiler feedback with annotated listing of source code indicating important optimizations**
- **The Program Library (PL), an application wide repository**
  - Allows **whole application analysis**
  - Allows exchange of information between tools and the compiler
- **Automatic shared memory parallelization with whole program analysis**
- **Bit reproducibility while maintaining high performance is a key example; critical for our climate modeling customers**
- **Fully integrated heterogeneous optimization capability**

CCE 8.3 Highlights (June 2014)

- **The new option –h develop selects compiler optimization levels to balance compile time against application execution time.**
  - This option is intended for use during application development, when quick turnaround is desired.
  - It minimizes compile time at the cost of some execution time performance.
- **-h flex_mp=strict provides a level repeatability of between the conservative and intolerant levels.**
  - Other general improvements have also been made for -h flex_mp.
- **New UPC extensions**
  - cray_upc_sheap_info() call provides symmetric heap usage information
  - cray_upc_shared_cast() call creates a pointer-to-shared from a pointer-to-local.
- **For Fortran applications, a string identifying MPI rank and OpenMP thread ID begins each line written to stdout and stderr.**
- **Performance….**

CCE 8.4 Highlights (September 2015)

- **Support for the C++11 language standard**
  - To enable C++11 features, use the -h std=c++11 command line option
- **Support for the OpenMP 4.0 specification**
- **Support for the inline assembly ASM construct for x86 processor targets**
- **Support for GNU extensions by default (-h gnu option)**
- **Fortran option to initialize floating point arrays to NaNs**
- **Performance….**

Production Quality

- **Functional regression testing done nightly**
  - Roughly 35,000 nightly regression tests run for Fortran (14,000), C (7,000), and C++ (14,000)
  - Default optimization, but for multiple targets (X86, X86+AVX+FMA, X2, X86+NVIDIA), plus “debug” and “production” compiler versions
  - Additionally, cycle through “options testing” with the same test base
    - Fortran: -G0, -G1, -G2, -O0, -Oipa0, -Oipa5 -hpic, “-O3,fp3” –e0
    - C and C++: -Gn, -O0, -hipa0, -hipa5, -hpic, “-O3 –hfp3” -hzero
    - Additional tests and suites have been added for GPU testing
    - And some “stress test” option sets to create worse-case scenarios for the compiler
    - Other combinations as necessary and by request
- **Performance regression testing done weekly using important applications and benchmarks**
- **Automated tools quickly isolate a test change to a specific compiler or library mod**

Some Cray Compilation Environment Basics

- **CCE-specific features:**
  - Optimization: **-O2** is the default and you should usually use this
  - CCE only gives minimal information to stderr when compiling
    - To see more information, you should request a compiler listing file
      - flag **-hlist=a**
      - writes a file with extension .lst
      - contains annotated source listing, followed by explanatory messages
    - Each message is tagged with an identifier, e.g.: **ftn-6430**
      - to get more information on this, type: **explain <identifier>**
    - Cray Reveal can display all this information (and more)

Example: Cray loopmark Messages

- –hlist=m …

```
29.  b-------<   do i3=2,n3-1
30.  b b-----<      do i2=2,n2-1
31.  b b Vr--<         do i1=1,n1
32.  b b Vr             u1(i1) = u(i1,i2-1,i3) + u(i1,i2+1,i3)
33.  b b Vr     *          + u(i1,i2,i3-1) + u(i1,i2,i3+1)
34.  b b Vr             u2(i1) = u(i1,i2-1,i3-1) + u(i1,i2+1,i3-1)
35.  b b Vr     *          + u(i1,i2-1,i3+1) + u(i1,i2+1,i3+1)
36.  b b Vr-->         enddo
37.  b b Vr--<         do i1=2,n1-1
38.  b b Vr             r(i1,i2,i3) = v(i1,i2,i3)
39.  b b Vr     *         - a(0) * u(i1,i2,i3)
40.  b b Vr     *         - a(2) * ( u2(i1) + u1(i1-1) + u1(i1+1) )
41.  b b Vr     *         - a(3) * ( u2(i1-1) + u2(i1+1) )
42.  b b Vr-->         enddo
43.  b b----->      enddo
44.  b------->   enddo
```

Example: Cray loopmark messages (cont)

```
ftn-6289 ftn: VECTOR File = resid.f, Line = 29
  A loop starting at line 29 was not vectorized because a recurrence was found on "U1" between
lines 32 and 38.
ftn-6049 ftn: SCALAR File = resid.f, Line = 29
  A loop starting at line 29 was blocked with block size 4.
ftn-6289 ftn: VECTOR File = resid.f, Line = 30
  A loop starting at line 30 was not vectorized because a recurrence was found on "U1" between
lines 32 and 38.
ftn-6049 ftn: SCALAR File = resid.f, Line = 30
  A loop starting at line 30 was blocked with block size 4.
ftn-6005 ftn: SCALAR File = resid.f, Line = 31
  A loop starting at line 31 was unrolled 4 times.
ftn-6204 ftn: VECTOR File = resid.f, Line = 31
  A loop starting at line 31 was vectorized.
ftn-6005 ftn: SCALAR File = resid.f, Line = 37
  A loop starting at line 37 was unrolled 4 times.
ftn-6204 ftn: VECTOR File = resid.f, Line = 37
  A loop starting at line 37 was vectorized.
```

Example of Explain Utility

users/ldr> explain ftn-6289

VECTOR: A loop starting at line %s was not vectorized because a recurrence

was found on "var" between lines num and num.

Scalar code was generated for the loop because it contains a linear recurrence. The following loop would cause this message to be issued:

```
DO I = 2,100
   B(I) = A(I-1)
   A(I) = B(I)
ENDDO
```

Recommended CCE Compilation Options

- **Use default optimization levels**
  - It's the equivalent of most other compilers –O3 or –fast
  - It is also our most thoroughly tested configuration

- **Using –O3,fp3 (or –O3 –hfp3, or some variation)**
  - -O3 only gives you slightly more than –O2
  - We also test this thoroughly
  - -hfp3 gives you a lot more floating point optimization, esp. 32-bit
  - Higher numbers are not always correlated with better performance

- **Optimizing for compile time rather than execution time**
  - Compile time can sometimes be improved by disabling certain features/ optimizations
    - Some common things to try: -hnodwarf, -hipa0, -hunroll0

OpenMP

- **OpenMP is ON by default**
  - Optimizations controlled by **–hthread#**
- **Autothreading is NOT on by default;**
  - -hautothread to turn on
  - Modernized version of Cray X1 streaming capability
  - **Interacts with OpenMP directives**
- **If you do not want to use OpenMP and have OMP directives in the code, make sure to shut off OpenMP at compile time**
  - **To shut off** use **–hthread0** or **–xomp** or **–hnoomp**

Cray Performance Measurement and Analysis Tools

Cray Performance Tools Strengths

- Whole program analysis across many nodes
- New and advanced user interfaces
- Support for MPI, SHMEM, OpenMP, UPC, CAF, OpenACC, CUDA
- Load Imbalance detection
- HW counter derived metrics
- Performance statistics for libraries called by program (BLAS, LAPACK, PETSc, NetCDF, HDF5, etc.)
- Observations of inefficient performance
- Data correlation to user source (line number, function)
- Energy consumption
- Minimal program perturbation

Two Interfaces to the Performance Tools

- **Support traditional post-mortem performance analysis**
  - Indication of causes of problems
  - Suggestions of modifications for performance improvement
- **CrayPat-lite for first time users**
- **CrayPat for in-depth performance investigation and tuning assistance**

New perftools-base and Instrumentation Modules

Access perftools Software

- **Load perftools-base module and leave it loaded**
  - Provides access to man pages, Reveal, Cray Apprentice2, and the new instrumentation modules
  - Can keep loaded with no impact to applications
- **Available starting in perftools/6.3.0 in September 2015**
- **Prior to perftools/6.3.0:**
  - Load perftools module

Program Instrumentation Modules

Instrumentation modules available after perftools-base is loaded:

- **perftools**
- **perftools-lite**
- perftools-lite-events
- perftools-lite-gpu
- **perftools-lite-loops**

What Do the Instrumentation Modules Do?

perftools

- Full access to CrayPat functionality
- Use pat_build to instrument, pat_report to process data and collect reports
- Equivalent to loading perftools module in earlier releases

perftools-lite

- Default CrayPat-lite profiling
- Load before building and running program to get a basic performance profile sent to stdout
- Equivalent to loading perftools-lite module in earlier releases

Tips

- Loading perftools without loading perftools-base first will continue to work as in pre-6.3.0 releases until perftools/6.4.0
- Sites can consider loading the default perftools-base for all users. Cray will look at automatically loading this module in a future release.
- Instrumentation modules can be loaded and unloaded for different performance experiments
- Use the ‘module list’ command to easily see which type of instrumentation is currently active
- Unload the instrumentation module after performance analysis experiments are complete

CrayPat-lite

How to Use CrayPat-lite

Access performance tools software & instrumentation module

```
> module load perftools-lite
```

Assumes default perftools-base is loaded in .login and kept loaded

Build program

```
> make
```

→ a.out (instrumented program)

Run program (no modification to batch script)

```
aprun a.out
```

→ Condensed report to stdout
a.out*.rpt (same as stdout)
a.out*.ap2
files

Example CrayPat-lite Output

```
CrayPat/X:  Version 6.1.4.12457 Revision 12457 (xf 12277)  02/26/14 13:58:24
Experiment:                  lite  lite/sample_profile
Number of PEs (MPI ranks):   8164
Numbers of PEs per Node:       16  PEs on each of  510  Nodes
                                4  PEs on     1  Node
Numbers of Threads per PE:      1
Number of Cores per Socket:     8
Execution start time:  Fri Feb 28 23:06:31 2014
System name and speed:  hera2 2100 MHz

Wall Clock Time:    999.595275 secs
High Memory:            475.52 MBytes
MFLOPS (aggregate):  806112.33 M/sec
I/O Read Rate:           33.57 MBytes/Sec
I/O Write Rate:         215.40 MBytes/Sec
```

Example CrayPat-lite Output (2)

```
Table 1:  Profile by Function Group and Function (top 7 functions shown)
 Time%   |       Time   |     Imb. |  Imb. |      Calls  |Group
         |              |     Time | Time% |             | Function
         |              |          |       |             |  PE=HIDE

 100.0% | 101.961423 |       -- |     -- | 5315211.9 |Total
|-------------------------------------------------------------------------------
|  92.5% |  94.267451 |       -- |     -- | 5272245.9 |USER
||------------------------------------------------------------------------------
||  75.8% |  77.248585 | 2.356249 |   3.0% |    1001.0 |LAMMPS_NS::PairLJCut::compute
||   6.5% |   6.644545 | 0.105246 |   1.6% |      51.0 |LAMMPS_NS::Neighbor::half_bin_newton
||   4.1% |   4.131842 | 0.634032 |  13.5% |       1.0 |LAMMPS_NS::Verlet::run
||   3.8% |   3.841349 | 1.241434 |  24.8% | 5262868.9 |LAMMPS_NS::Pair::ev_tally
||   1.3% |   1.288463 | 0.181268 |  12.5% |    1000.0 |LAMMPS_NS::FixNVE::final_integrate
||==============================================================================
|   7.0% |   7.110931 |       -- |     -- |   42637.0 |MPI
||------------------------------------------------------------------------------
||   4.8% |   4.851309 | 3.371093 |  41.6% |   12267.0 |MPI_Send
||   1.5% |   1.536106 | 2.592504 |  63.8% |   12267.0 |MPI_Wait
|===============================================================================
```

Example CrayPat-lite Output (3)

```
Table 2:  File Input Stats by Filename

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

 387.432937 | 13006.522781 |  33.571030 | 41596900.0 |       327.87 |Total
|------------------------------------------------------------------------------------
| 331.691801 |  1395.829828 |   4.208213 | 13153931.0 |       111.27 |/proc/self/maps
|  13.129507 |  4075.682968 | 310.421627 |      868.0 |   4923575.28 |regional.grid.a
|  12.654338 |  2000.329418 | 158.074605 | 26892862.0 |        77.99 |./patch.input
|   3.924810 |   679.265625 | 173.069704 |        3.0 | 237420544.00 |./forcing.radflx.a
. . .
```

More Information from Same Profile

- You don't need to run again for the following:

```
For a complete report with expanded tables and notes, run:
  pat_report /lus/scratch/heidi/lab/craypat-lite/run/sweep3d.mpi.ap2

For help identifying callers of particular functions:
  pat_report -O callers+src /lus/scratch/heidi/lab/craypat-lite/run/
sweep3d.mpi.ap2

To see the entire call tree:
  pat_report -O calltree+src /lus/scratch/heidi/lab/craypat-lite/run/
sweep3d.mpi.ap2
```

Sampling with Line Number information

```
heidi@limited: /h/heidi — ssh — 81×26
Table 2:  Profile by Group, Function, and Line

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

 100.0% | 8376.9 |     -- |      -- |Total
|-----------------------------------------------------------------------------
|  93.2% | 7804.0 |     -- |      -- |USER
||----------------------------------------------------------------------------
||  51.7% | 4328.7 |     -- |      -- |calc3_
3|        |        |        |         | heidi/DARPA/cache_util/calc3.do300-ijswap.F
||||--------------------------------------------------------------------------
4|||  15.7% | 1314.4 |  93.6 |    6.8% |line.78
4|||  13.9% | 1167.7 |  98.3 |    7.9% |line.79
4|||  14.5% | 1211.6 |  97.4 |    7.6% |line.80
4|||   1.2% |  103.1 |  26.9 |   21.2% |line.93
4|||   1.1% |   88.4 |  22.6 |   20.8% |line.94
4|||   1.0% |   84.5 |  17.5 |   17.6% |line.95
4|||   1.0% |   86.8 |  33.2 |   28.2% |line.96
4|||   1.3% |  105.0 |  23.0 |   18.4% |line.97
4|||   1.4% |  116.5 |  24.5 |   17.7% |line.98
||||==========================================================================
                                                               144,1      38%
```

CrayPat

The Cray Performance Analysis Framework

- **Supports traditional post-mortem performance analysis**
  - Indication of causes of problems
  - Suggestions of modifications for performance improvement
- **pat_build: provides automatic instrumentation**
- **CrayPat run-time library collects measurements (transparent to the user)**
- **pat_region API**
  - Provides mechanism to control collection of performance data within source code
- **pat_report performs analysis and generates text reports**
- **pat_help: online help utility**
- **Cray Apprentice2: graphical visualization tool**

Application Instrumentation with pat_build

- **Supports two categories of experiments**
  - asynchronous experiments (sampling) which capture values from the call stack or the program counter at specified intervals or when a specified counter overflows
  - Event-based experiments (tracing) which count some events such as the number of times a specific system call is executed
- **While tracing provides most useful information, it can be very heavy if the application runs on a large number of cores for a long period of time**
- **Sampling can be useful as a starting point, to provide a first overview of the work distribution**

How to Use CrayPat

- **Make sure the following modules are loaded:**
  - PrgEnv-cray module
  - perftools module (perftools-base is already loaded)
- **Instrument binary for tracing user functions and MPI**
  - > pat_build –u –g mpi my_program
  - OpenMP is on by default when tracing is enabled
- **Run application**
- **Create report with GPU statistics**
  - > pat_report my_program.xf > my_report

pat_report

- Combines information from binary with raw performance data
- Performs analysis on data
- Generates text report of performance results
- Generates customized instrumentation template for automatic profiling analysis
- Formats data for input into Cray Apprentice²

MPI Messages By Caller

heidi@limited: /h/heidi — ssh — 81×26

```
Table 4:  MPI Message Stats by Caller

    MPI Msg   |     MPI   | MsgSz   | 4KB<=   |Function
      Bytes   |     Msg   |  <16B   | MsgSz   | Caller
              |   Count   | Count   | <64KB   |  PE=[mmm]
              |           |         | Count   |

 140166953.8 |  8890.6 |  339.8 | 8550.8 |Total
|-----------------------------------------------------
| 140166833.8 |  8875.6 |  324.8 | 8550.8 |MPI_ISEND
||----------------------------------------------------
||   78272400.0 |   4850.0 |    75.0 | 4775.0 |calc2_
3|              |          |         |        | shalow_
||||--------------------------------------------------
4|||   78700800.0 |   7200.0 | 2400.0 | 4800.0 |pe.0
4|||   78681600.0 |   4800.0 |    0.0 | 4800.0 |pe.1
4|||   59020800.0 |   4800.0 | 1200.0 | 3600.0 |pe.47
||||==================================================
||   59421800.0 |   3725.0 |   100.0 | 3625.0 |calc1_
3|              |          |         |        | shalow_
||||--------------------------------------------------
4|||   78700800.0 |   7200.0 | 2400.0 | 4800.0 |pe.0
4|||   59011200.0 |   3600.0 |    0.0 | 3600.0 |pe.1
4|||   59011200.0 |   3600.0 |    0.0 | 3600.0 |pe.24
||||==================================================
                                                         624,3          79%
```

Collecting Performance Counter Information

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.)

Performance Counters

- **Cray supports raw counters, derived metrics and thresholds for:**
  - Processor (core and uncore)
  - Network
  - Accelerator
  - Power
- **Predefined groups**
  - Groups together suggested counters for experiments
- **Single interface to access counters**
  - **PAT_RT_PERFCTR** environment variable
- **See *hwpc, nwpc, accpc, and rapl* man pages**

How to Get List of Events for a Processor

- **Run the following utilities on a compute node:**
  - papi_avail
  - papi_native_avail

- **Use pat_help on login node**
  - > pat_help counters haswell
    - deriv
    - Groups
    - Native
    - papi

- **Set PAT_RT_PERFCTR environment variable to list of events or group prior to execution**

Performance Counters via PAPI

- **Common set of events deemed relevant and useful for application performance tuning**
  - Accesses to the memory hierarchy, cycle and instruction counts, functional units, pipeline status, etc.
  - The “papi_avail” utility shows which predefined events are available on the system – execute on compute node
- **PAPI also provides access to native events**
  - The “papi_native_avail” utility lists all native events available on the system – execute on compute node
- **PAPI uses perf_events Linux subsystem**

Example Performance Counter Groups - SNB

> pat_help counters sandybridge groups

```
There are 14 predefined hardware performance counter event groups that can be specified by
setting PAT_RT_PERFCTR to the group id. Some groups contain the keyword "mpx" to enable
multiplexing.

  Additional topics:

     0: D1 with instruction counts
     1: Summary -- FP and cache metrics
     2: D1, D2, L3 Metrics
     6: Micro-op queue stalls
     7: Back end stalls
     8: Instructions and branches
     9: Instruction cache
    10: Cache Hierarchy
    11: Floating point operations dispatched
    12: AVX floating point operations
    13: SSE and AVX floating point operations SP
    14: SSE and AVX floating point operations DP
    23: FP and cache metrics (same as 1)
    default: group 1
```

Example: HW counter data and Derived Metrics

```
  PAPI_TLB_DM  Data translation lookaside buffer misses
  PAPI_L1_DCA  Level 1 data cache accesses
  PAPI_FP_OPS  Floating point operations
  DC_MISS      Data Cache Miss
  User_Cycles  Virtual Cycles
========================================================================
USER
------------------------------------------------------------------------
  Time%                                              98.3%
  Time                                           4.434402 secs
  Imb.Time                                             -- secs
  Imb.Time%                                            --
  Calls                          0.001M/sec        4500.0 calls
  PAPI_L1_DCM                   14.820M/sec      65712197 misses
  PAPI_TLB_DM                    0.902M/sec       3998928 misses
  PAPI_L1_DCA                  333.331M/sec    1477996162 refs
  PAPI_FP_OPS                  445.571M/sec    1975672594 ops
  User time (approx)             4.434 secs   11971868993 cycles  100.0%Time
  Average Time per Call                          0.000985 sec
  CrayPat Overhead : Time         0.1%
  HW FP Ops / User time        445.571M/sec    1975672594 ops   4.1%peak(DP)
  HW FP Ops / WCT              445.533M/sec
  Computational intensity         0.17 ops/cycle     1.34 ops/ref
  MFLOPS (aggregate)           1782.28M/sec
  TLB utilization               369.60 refs/miss    0.722 avg uses
  D1 cache hit,miss ratios       95.6% hits          4.4% misses
  D1 cache utilization (misses)  22.49 refs/miss    2.811 avg hits
========================================================================
```

PAT_RT_PERFCTR=1
Flat profile data
Raw counts
Derived metrics

Maximize On-node Communication by Reordering MPI ranks

When Is Rank Re-ordering Useful?

- Maximize on-node communication between MPI ranks
- Physical system topology agnostic
- Grid detection and rank re-ordering is helpful for programs with significant point-to-point communication
- Relieve on-node shared resource contention by pairing threads or processes that perform different work (for example computation with off-node communication) on the same node

MPI Rank Reorder – Two Interfaces Available

- **CrayPat**
  - Include –g mpi when instrumenting program
  - Run program and let CrayPat determine if communication is dominant, detect communication pattern and suggest MPI rank order if applicable
- **grid_order utility**
  - User knows communication pattern in application and wants to quickly create a new MPI rank placement file
  - Available when perftools module is loaded

Automatic Communication Grid Detection

- Cray performance tools produce a custom rank order if it's beneficial based on grid size, grid order and cost metric
- Summarized findings in report
- Available with sampling or tracing
- Describe how to re-run with custom rank order

MPI Rank Order Observations

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

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

 100.0% | 463.147240 |         -- |      -- | 21621.0 |Total
|--------------------------------------------------------------------------
|  52.0% | 240.974379 |         -- |      -- | 21523.0 |MPI
||-------------------------------------------------------------------------
||   47.7% | 221.142266 | 36.214468 |   14.1% | 10740.0 |mpi_recv
||    4.3% |  19.829001 | 25.849906 |   56.7% | 10740.0 |MPI_SEND
||=========================================================================
|  43.3% | 200.474690 |         -- |      -- |    32.0 |USER
||-------------------------------------------------------------------------
||   41.0% | 189.897060 | 58.716197 |   23.6% |    12.0 |sweep_
||    1.6% |   7.579876 |  1.899097 |   20.1% |    12.0 |source_
||=========================================================================
|   4.7% |  21.698147 |         -- |      -- |    39.0 |MPI_SYNC
||-------------------------------------------------------------------------
|   4.3% |  20.091165 | 20.005424 |   99.6% |    32.0 | mpi_allreduce_(sync)
||=========================================================================
|   0.0% |   0.000024 |         -- |      -- |    27.0 |SYSCALL
|==========================================================================
```

MPI Rank Order Observations (2)

```
MPI Grid Detection:

    There appears to be point-to-point MPI communication in a 96 X 8
    grid pattern. The 52% of the total execution time spent in MPI
    functions might be reduced with a rank order that maximizes
    communication between ranks on the same node. The effect of several
    rank orders is estimated below.

    A file named MPICH_RANK_ORDER.Grid was generated along with this
    report and contains usage instructions and the Custom rank order
    from the following table.

         Rank    On-Node     On-Node   MPICH_RANK_REORDER_METHOD
        Order   Bytes/PE   Bytes/PE%
                            of Total
                            Bytes/PE

        Custom  2.385e+09      95.55%   3
           SMP  1.880e+09      75.30%   1
          Fold  1.373e+06       0.06%   2
    RoundRobin  0.000e+00       0.00%   0
```

Auto-Generated MPI Rank Order File

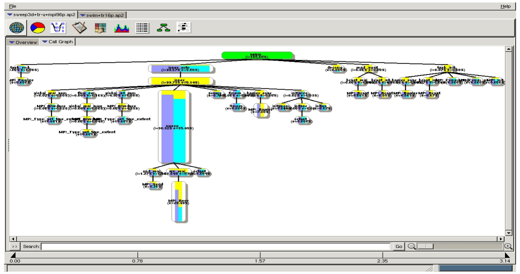
```
# The 'USER_Time_hybrid'
rank order in this file
targets nodes with multi-
core
# processors, based on Sent
Msg Total Bytes collected
for:
#
# Program:      /lus/
nid00023/malice/craypat/
WORKSHOP/bh2o-demo/Rank/
sweep3d/src/sweep3d
# Ap2 File:
sweep3d.gmpi-u.ap2
# Number PEs:   768
# Max PEs/Node: 16
#
# To use this file, make a
copy named MPICH_RANK_ORDER,
and set the
# environment variable
MPICH_RANK_REORDER_METHOD to
3 prior to
# executing the program.
#
0,532,64,564,32,572,96,540,8
,596,72,524,40,604,24,588
104,556,16,628,80,636,56,620
,48,516,112,580,88,548,120,6
12
1,403,65,435,33,411,97,443,9
,467,25,499,105,507,41,475
73,395,81,427,57,459,17,419,
113,491,49,387,89,451,121,48
3
6,436,102,468,70,404,38,412,
14,444,46,476,110,508,78,500
86,396,30,428,62,460,54,492,
118,420,22,452,94,388,126,48
4
129,563,193,531,161,571,225,
539,241,595,233,523,249,603,
185,555
153,587,169,627,137,635,201,
619,177,515,145,579,209,547,
217,611
7,405,71,469,39,437,103,413,
47,445,15,509,79,477,31,501
111,397,63,461,55,429,87,421
,23,493,119,389,95,453,127,4
85
134,402,198,434,166,410,230,
442,238,466,174,506,158,394,
246,474
190,498,254,426,142,458,150,
386,182,418,206,490,214,450,
222,482
128,533,192,541,160,565,232,
525,224,573,240,597,184,557,
248,605
168,589,200,517,152,629,136,
549,176,637,144,621,208,581,
216,613
5,439,37,407,69,447,101,415,
13,471,45,503,29,479,77,511
53,399,85,431,21,463,61,391,
109,423,93,455,117,495,125,4
87
2,530,34,562,66,538,98,522,1
0,570,42,554,26,594,50,602
18,514,74,586,58,626,82,546,
106,634,90,578,114,618,122,6
10
135,315,167,339,199,347,259,
307,231,371,239,379,191,331,
247,299
175,363,159,323,143,355,255,
291,207,275,183,283,151,267,
215,223
133,406,197,438,165,470,229,
414,245,446,141,478,237,502,
253,398
157,510,189,462,173,430,205,
390,149,422,213,454,181,494,
221,486
130,316,260,340,194,372,162,
348,226,308,234,380,242,332,
250,300
202,364,186,324,154,356,138,
292,170,276,178,284,210,218,
268,146
4,535,36,543,68,567,100,527,
12,599,44,575,28,559,76,607
52,591,20,631,60,639,84,519,
108,623,92,551,116,583,124,6
15
3,440,35,432,67,400,99,408,1
1,464,43,496,27,472,51,504
19,392,75,424,59,456,83,384,
107,416,91,488,115,448,123,4
80
132,401,196,441,164,409,228,
433,236,465,204,473,244,393,
188,497
252,505,140,425,212,457,156,
385,172,417,180,449,148,489,
220,481
131,534,195,542,163,566,227,
526,235,574,203,598,243,558,
187,606
251,590,211,630,179,638,139,
622,155,550,171,518,219,582,
147,614
761,660,737,652,705,668,745,
692,673,700,641,684,713,644,
753,724
729,732,681,756,721,716,764,
676,697,748,689,657,740,665,
649,708
760,528,736,536,704,560,744,
520,672,568,712,592,752,552,
640,600
728,584,680,624,720,512,696,
632,688,616,664,544,608,656,
648,576
762,659,738,651,706,667,746,
643,714,691,674,699,754,683,
730,723
722,731,763,658,642,755,739,
675,707,650,682,715,698,666,
690,747
257,345,265,313,281,305,273,
337,609,369,577,377,617,329,
513,529
545,297,633,361,625,321,585,
537,601,289,553,353,593,521,
569,561
256,373,261,341,264,349,280,
317,272,381,269,309,285,333,
277,365
352,301,320,325,288,357,328,
304,360,312,376,293,296,368,
336,344
258,338,266,346,282,314,274,
370,766,306,710,378,742,330,
678,362
646,298,750,322,718,354,758,
290,734,662,686,670,726,702,
694,654
262,375,263,343,270,311,271,
351,286,319,278,342,287,350,
279,374
294,318,358,383,359,310,295,
382,326,303,327,367,366,335,
302,334
765,661,709,663,741,653,711,
669,767,655,743,671,749,695,
679,703
677,727,751,693,647,701,717,
687,757,685,733,725,719,735,
645,759
```

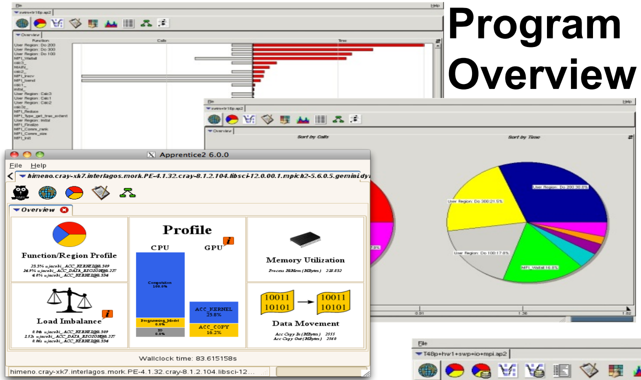
Using New Rank Order

- Save grid_order output to file called MPICH_RANK_ORDER
- Export MPICH_RANK_REORDER_METHOD=3
- Run non-instrumented binary with and without new rank order to check overall wallclock time for performance improvements
- Can be used for all subsequent executions of same job size

Visualizing Performance of Your Application Through Cray Apprentice2

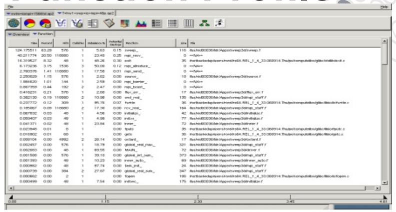
Cray Apprentice²



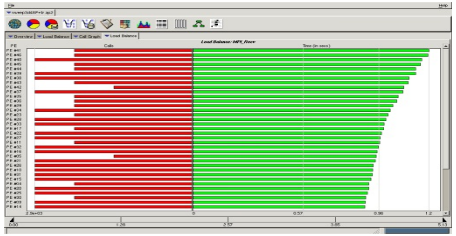


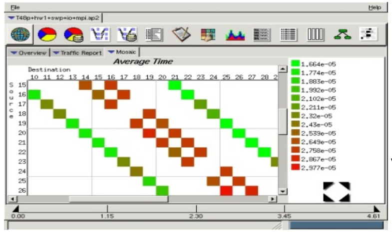
Program Overview

Function Profile



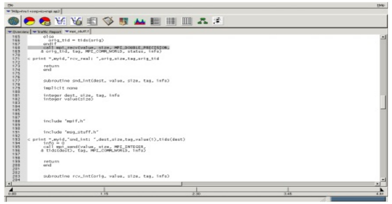
Load balance views

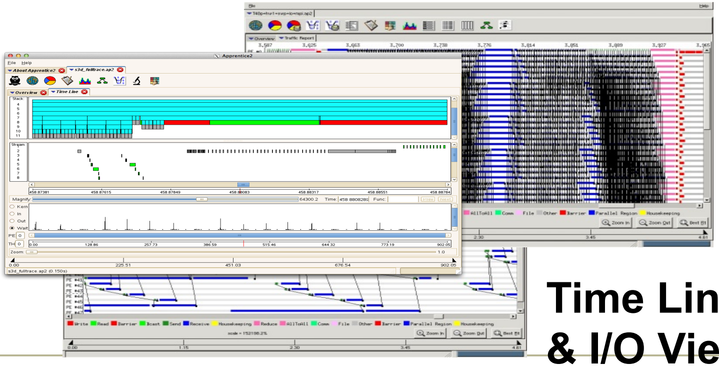


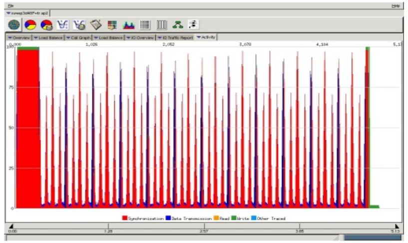


Pair-wise Communication View

Source code mapping







Time Line & I/O Views

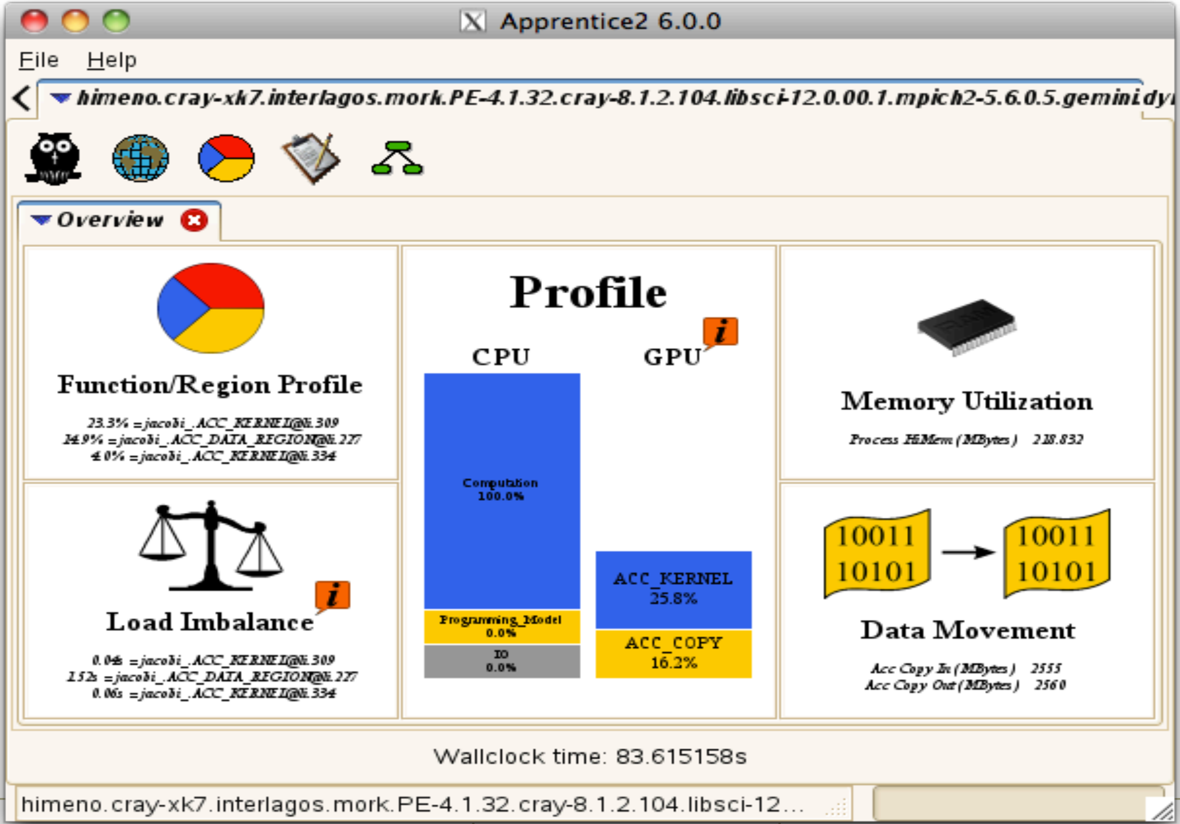
Communication & I/O Activity View

Installing Apprentice2 on Laptop

From a Cray login node

- `> module load perftools`
- **Go to:**
  - $CRAYPAT_ROOT/share/desktop_installers/
- **Download .dmg or .exe installer to laptop**
- **Double click on installer and follow directions to install**

Apprentice2 Overview with GPU Data



Call Tree View

Node width ⇔ inclusive time
Node height ⇔ exclusive time

Load balance overview:
Height ⇔ Max time
Upper bar ⇔ Average time
Lower bar ⇔ Min time
Yellow represents imbalance time

Function List

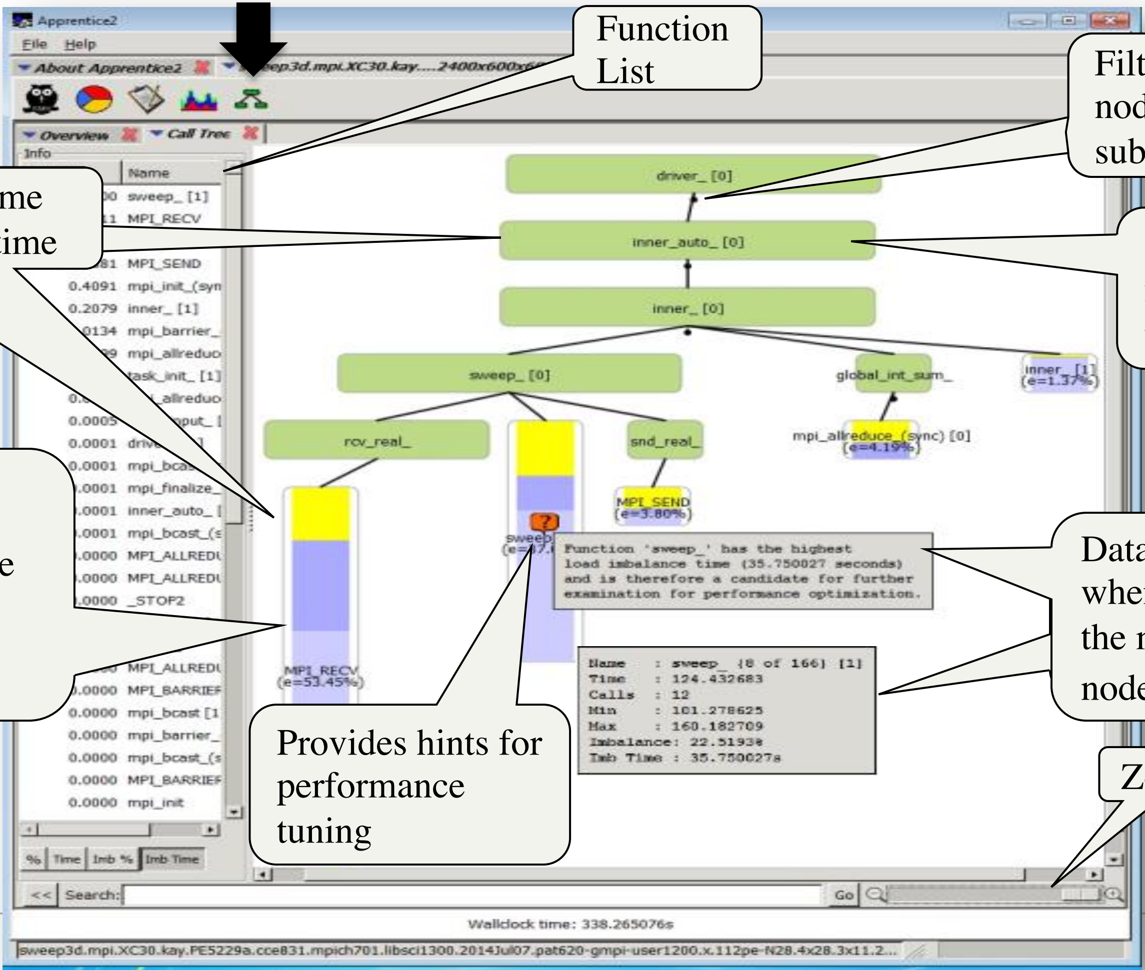
Filtered node or sub tree

Green colored nodes are not traced.

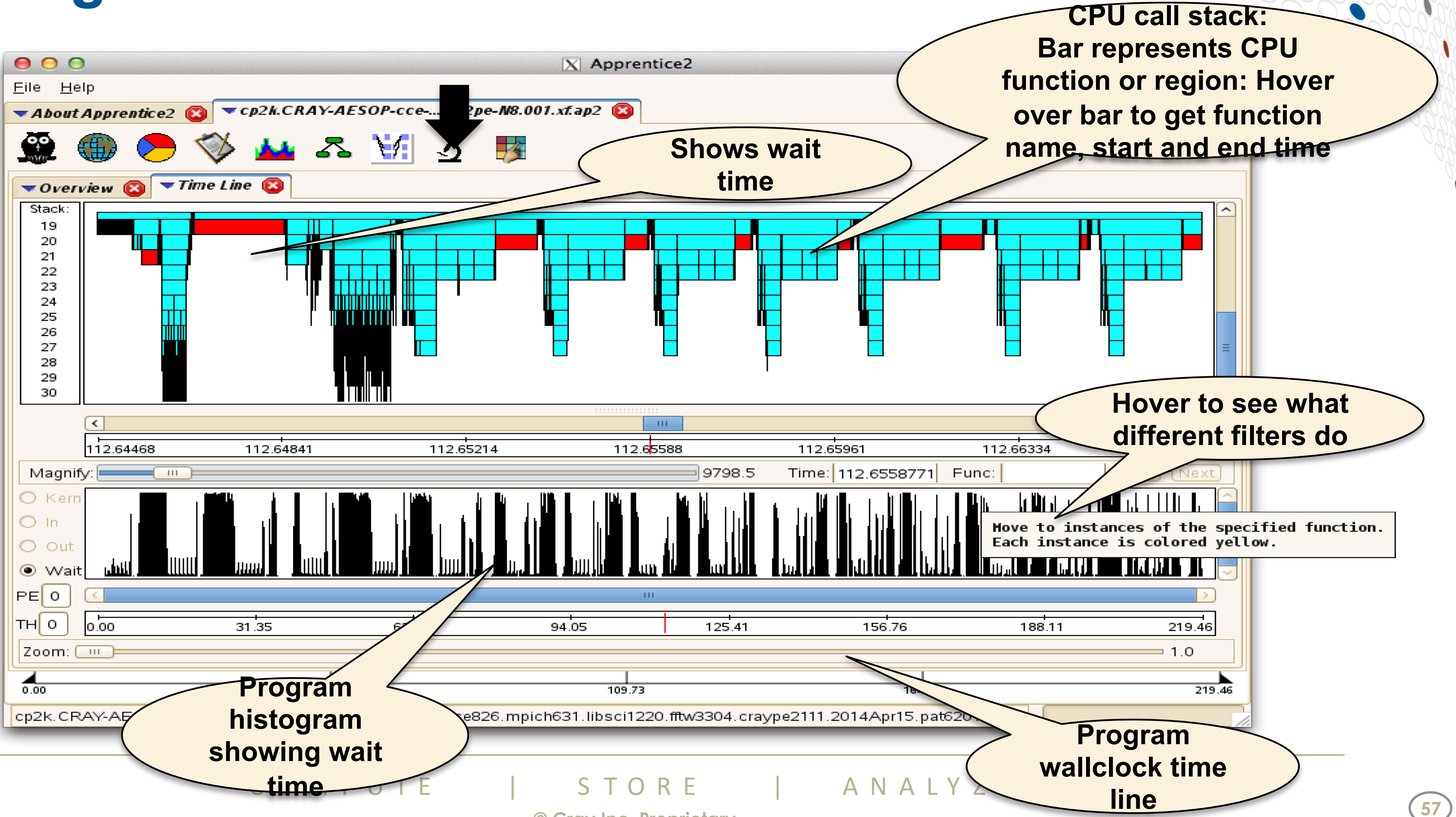
Data displayed when hovering the mouse over nodes or "?".

Provides hints for performance tuning

Zoom



CPU Program Timeline: 36GB CP2K Full Trace

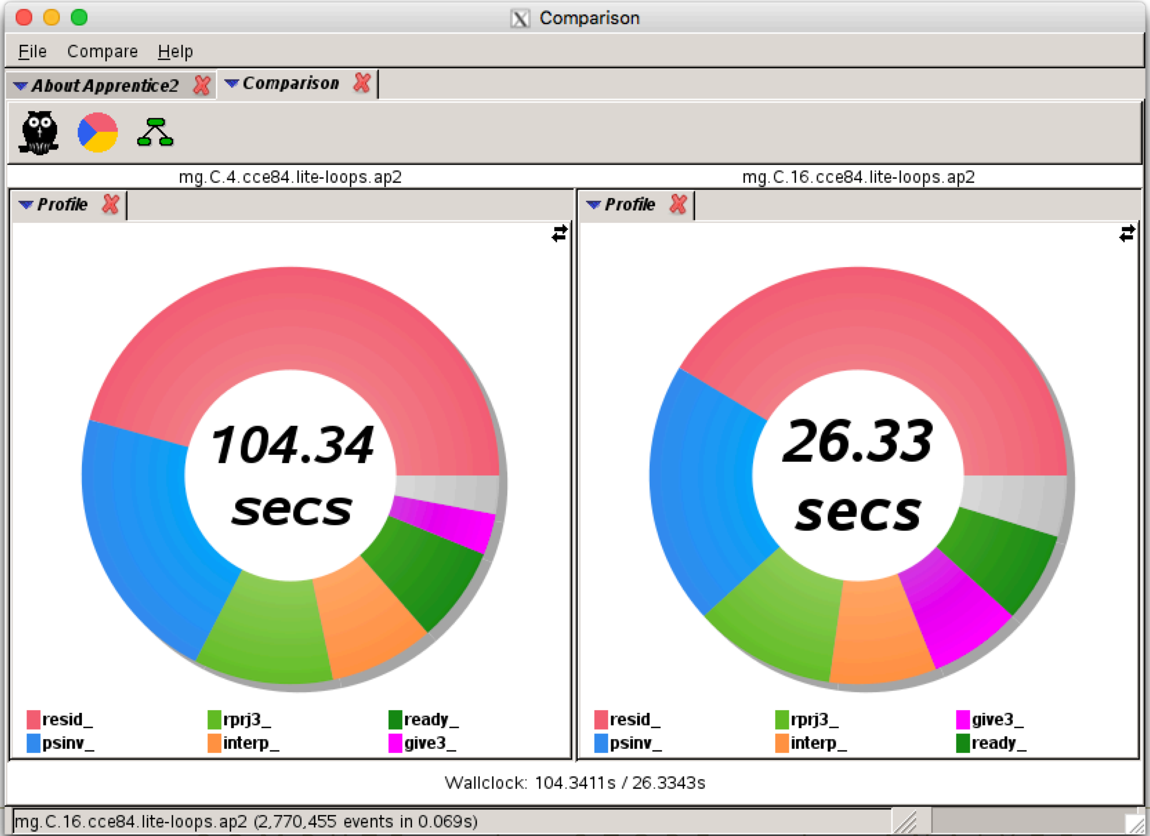


What's New?

Recent Enhancements

- **Improved ease of use:**
  - perftools-base module, pat_info utility
- **Profile comparison in Cray Apprentice2**
  - Useful for comparing MPI vs MPI+OpenMP, scaling bottlenecks, etc.
- **2D communication heat map (Cray Apprentice2 Mosaic) in summarized mode**
- **Visualize sampling data over time with associated call stack**

Apprentice2 Comparison



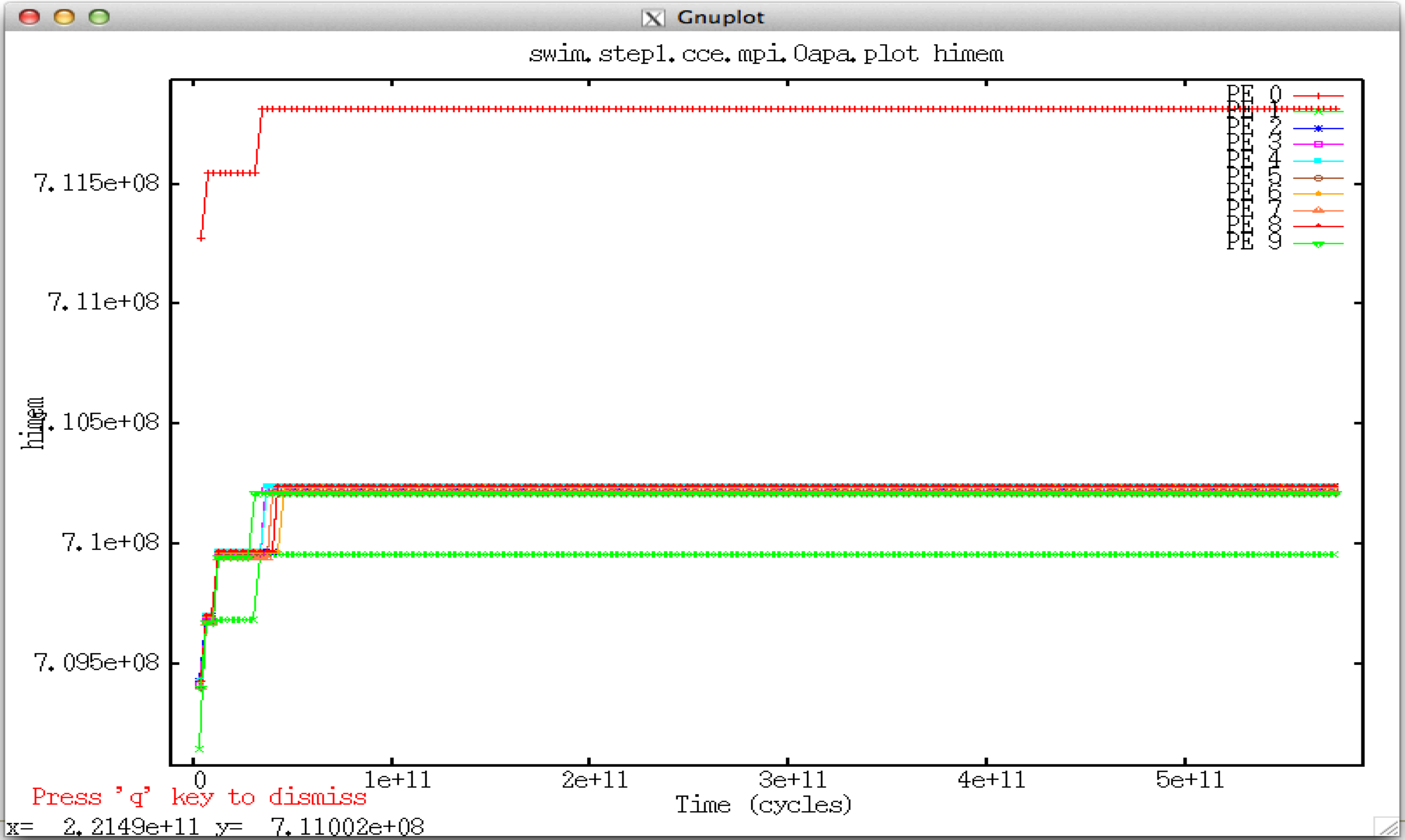
Sampling Over Time

- Available in perftools/6.2.3 (available in April 2015)
- Intended for collecting higher overhead performance data
- Sampling experiment in non-summary mode
  - `PAT_RT_SUMMARY=0`
  - `PAT_RT_SAMPLING_DATA=cray_pm`
- Records data every 100 Program Counter addresses by default (user can adjust)
- Examples:
  - Heap, shared heap
  - Perfctr (selected performance counters)
  - Rusage (resource usage (getrusage)
  - Cray PM, RAPL

Visualize Samples Over Time

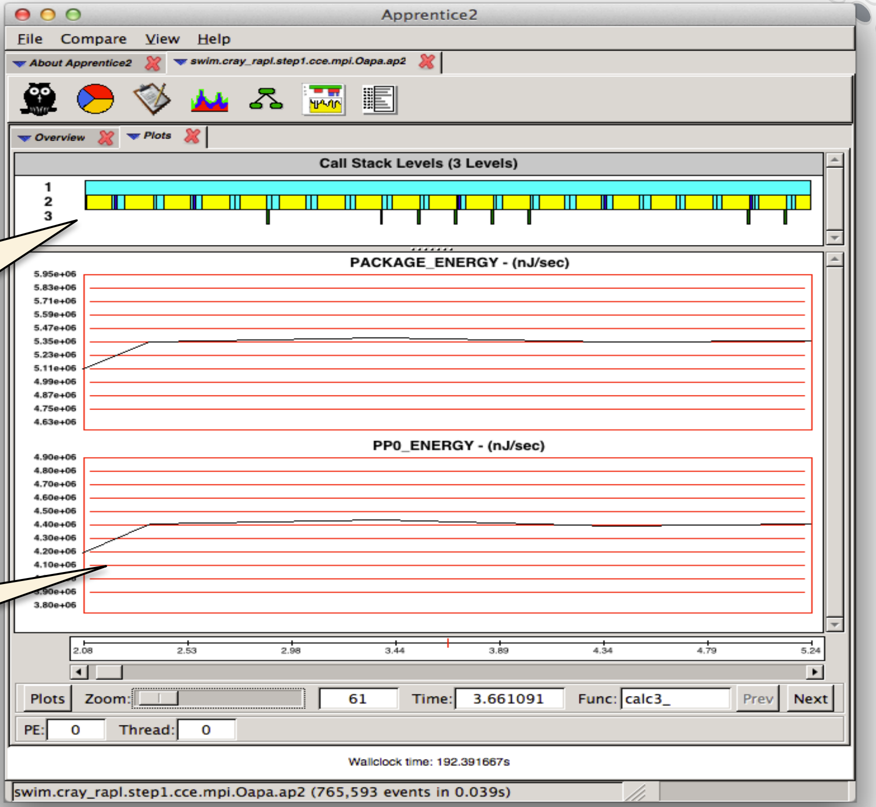
- **Plots show activity over time**
- **pat_report generates gnuplot files**
  - > pat_report [-r] -f plot $some.xf
  - > pat_report [-r] -f plot $some.ap2
- **Visualize (pat_report launches gnuplot)**
  - > pat_report $some.plot
  - > pat_report $some.plot/himem.gp
  - > pat_report -s pe=N
    - plot data only for pe N
  - > pat_report -s filter_input='pe<10'
    - specify a subset of pe values for which to plot data
- **Run "pat_help plots" or see craypat(1) man page for more info**

Memory High Water Mark with Gnuplot



Energy Consumption Over Time in Apprentice2

- Associates counter data with program call stack



Call stack:
Bar represents function or region: Hover over bar to get function name, start and end time

Plots of energy consumed by the socket and by the cores within a socket over time. Can also show memory high water mark, etc.

Performance Tools Documentation and Tips

Cray PE Documentation Available

- Release Notes
  - `> module help` *`product/product_version`*
- User Guides
  - http://docs.cray.com
- Man pages, for example:
  - cc
  - crayftn
  - intro_directives
  - Intro_openacc

How to Access Perftools

- **> module load perftools-base (can go in .login)**
- **Then:**
  - To do light profile: > module load perftools-lite
  - To get loop statistics: > module load perftools-lite-loops

Perftools Documentation Available

- Release Notes
  - `> module help perftools/`*`version_number`*
- User manual "Using the Cray Performance Measurement and Analysis Tools" available at http://docs.cray.com
- pat_help – interactive help utility on the Cray Performance toolset
- Man pages

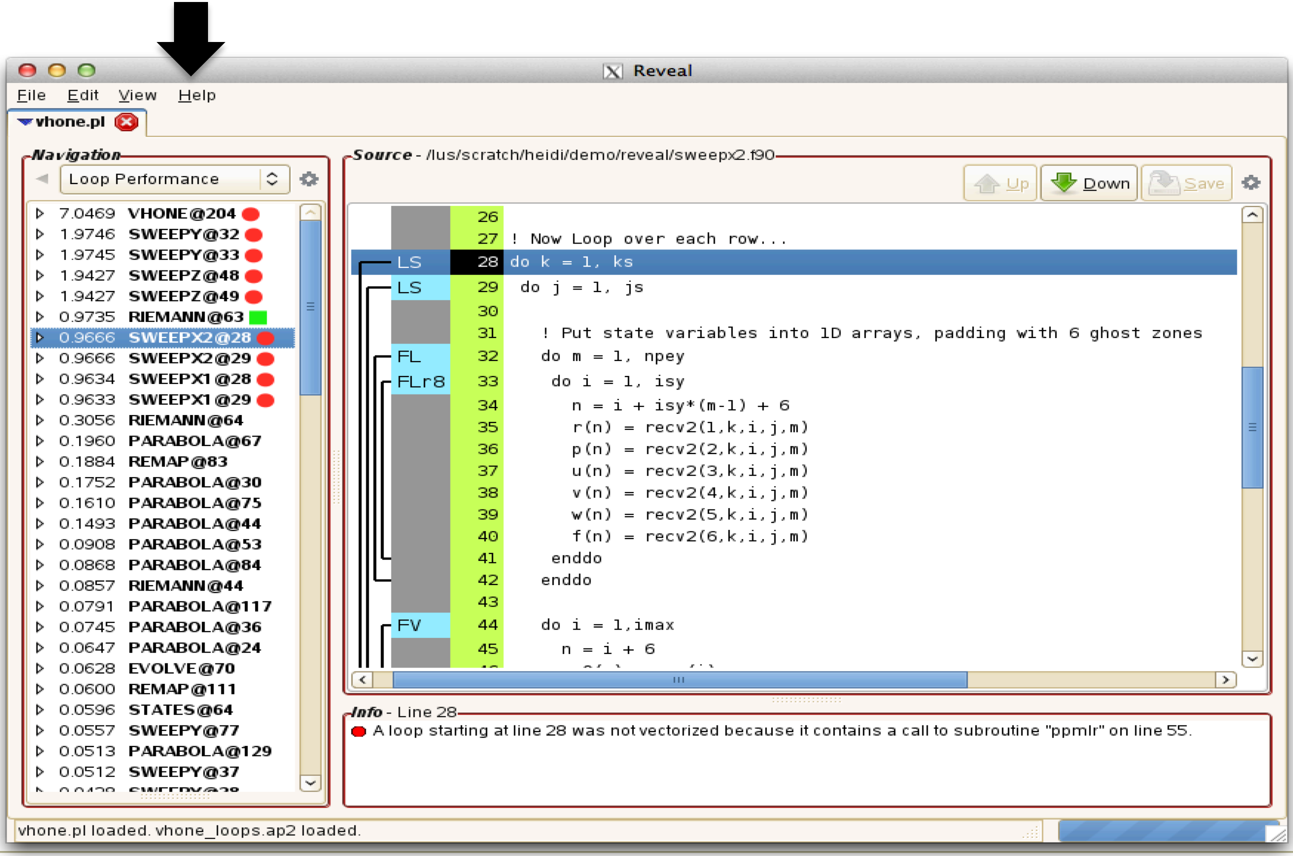
Man pages

- **intro_craypat(1)**
  - Introduces the craypat performance tool
  - Runtime environment variables (enable full trace, etc.)
- **pat_build(1)**
  - Instrument a program for performance analysis
- **pat_help(1)**
  - Interactive online help utility
- **pat_report(1)**
  - Generate performance report in both text and for use with GUI
- **app2 (1)**
  - Describes how to launch Cray Apprentice2 to visualize performance data

Man pages (2)

- **hwpc(5)**
  - describes predefined hardware performance counter groups
- **nwpc(5)**
  - Describes predefined network performance counter groups
- **accpc(5) / accpc_k20(5), etc.**
  - Describes predefined GPU performance counter groups
- **intro_papi(3)**
  - Lists PAPI event counters
  - Use papi_avail or papi_native_avail utilities to get list of events when running on a specific architecture

Reveal Help



Reveal
File Edit View Help
vhone.pl
Navigation
Loop Performance
7.0469 VHONE@204
1.9746 SWEEPY@32
1.9745 SWEEPY@33
1.9427 SWEEPZ@48
1.9427 SWEEPZ@49
0.9735 RIEMANN@63
0.9666 SWEEPX2@28
0.9666 SWEEPX2@29
0.9634 SWEEPX1@28
0.9633 SWEEPX1@29
0.3056 RIEMANN@64
0.1960 PARABOLA@67
0.1884 REMAP@83
0.1752 PARABOLA@30
0.1610 PARABOLA@75
0.1493 PARABOLA@44
0.0908 PARABOLA@53
0.0868 PARABOLA@84
0.0857 RIEMANN@44
0.0791 PARABOLA@117
0.0745 PARABOLA@36
0.0647 PARABOLA@24
0.0628 EVOLVE@70
0.0600 REMAP@111
0.0596 STATES@64
0.0557 SWEEPY@77
0.0513 PARABOLA@129
0.0512 SWEEPY@37
Source - /lus/scratch/heidi/demo/reveal/sweepx2.f90
Up Down Save
! Now Loop over each row...
LS do k = 1, ks
LS do j = 1, js
! Put state variables into 1D arrays, padding with 6 ghost zones
FL do m = 1, npey
FLr8 do i = 1, isy
n = i + isy*(m-1) + 6
r(n) = recv2(1,k,i,j,m)
p(n) = recv2(2,k,i,j,m)
u(n) = recv2(3,k,i,j,m)
v(n) = recv2(4,k,i,j,m)
w(n) = recv2(5,k,i,j,m)
f(n) = recv2(6,k,i,j,m)
enddo
enddo
FV do i = 1,imax
n = i + 6
Info - Line 28
A loop starting at line 28 was not vectorized because it contains a call to subroutine "ppmlr" on line 55.
vhone.pl loaded. vhone_loops.ap2 loaded.

Reveal Usage Recipe

- **Access Cray compiler**
  - > module load PrgEnv-cray
- **Access perftools**
  - > module load perftools-base
- **Enable loop work estimates program instrumentation**
  - > module load perftools-lite-loops
- **Build program (make)**
- **Run program to get loop work estimates in file with .ap2 suffix**

Reveal Usage Recipe (2)

- **Disable loop work estimates program instrumentation so we can get fully optimized program now**
  - > module unload perftools-lite-loops

- **Create program library with CCE:**
  - Add –h pl=/*full_path*/my_program.pl to program's Makefile

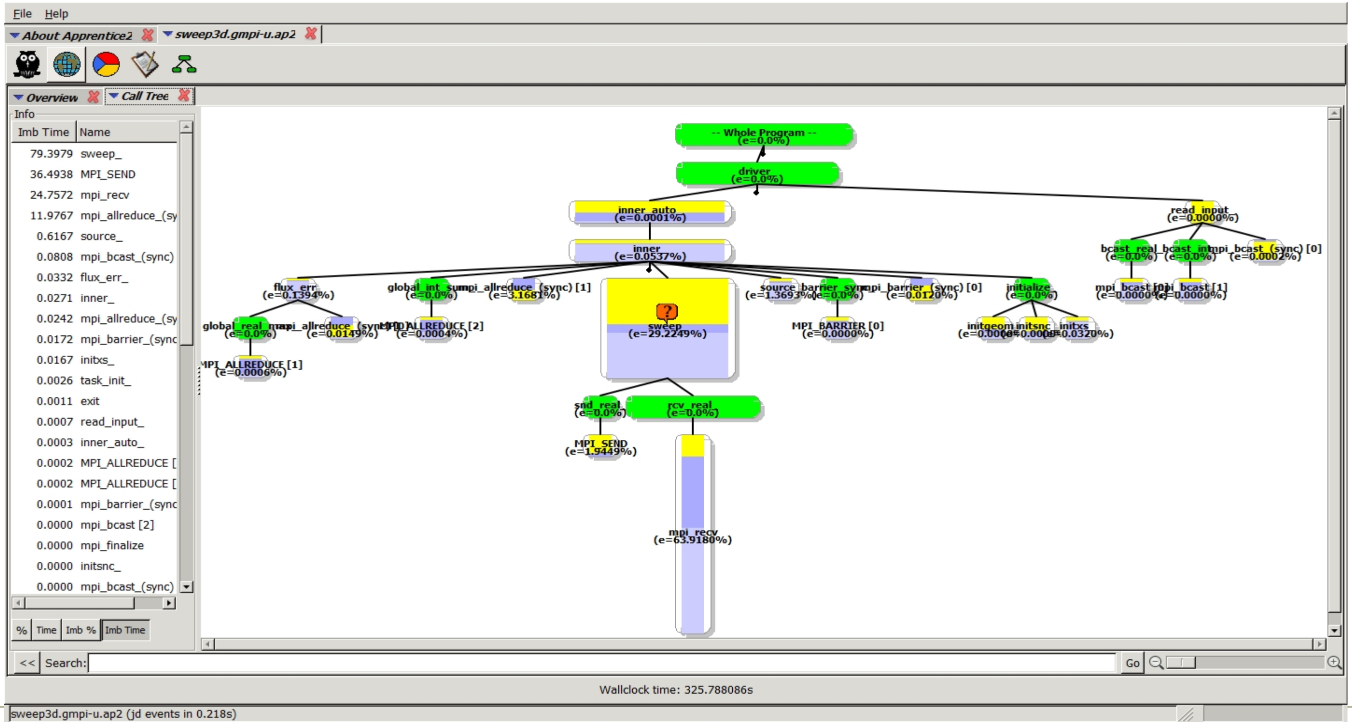
- **Rebuild application with full optimization**
  - > make clean
  - > make

- **Launch Reveal**
  - > reveal /*full_path*/my_program.pl loop_work_estimates.ap2

How to Install Apprentice2 on Your Laptop

- > module load perftools
- Go to:
  - $CRAYPAT_ROOT/share/desktop_installers/
- Download .dmg or .exe installer
- Double click on installer and follow directions to install

Apprentice2 Help



COMPUTE | STORE | ANALYZE

Why Should I generate a ".ap2" file?

- The ".ap2" file is a self contained compressed performance file
- Normally it is about 5 times smaller than the ".xf" file
- Contains the information needed from the application binary
  - Can be reused, even if the application binary is no longer available or if it was rebuilt
- It is the only input format accepted by Cray Apprentice^2

Files Generated and the Naming Convention

File Suffix	Description
a.out+pat	Program instrumented for data collection
a.out…s.xf	Raw data for sampling experiment, available after application execution
a.out…t.xf	Raw data for trace (summarized or full) experiment, available after application execution
a.out…st.ap2	Processed data, generated by pat_report, contains application symbol information
a.out…s.apa	Automatic profiling pnalysis template, generated by pat_report (based on pat_build –O apa experiment)
a.out+apa	Program instrumented using .apa file
MPICH_RANK_ORDER.Custom	Rank reorder file generated by pat_report from automatic grid detection an reorder suggestions

More on pat_report Data

Data from pat_report

- **Default reports are intended to be useful for most applications**
- **Don’t need to rerun program to get more detailed data**
- **Different aggregations, or levels of information available**
  - Get fined-grained thread-imbalance information for OpenMP program
- **Get list of tables available:**
  - > pat_report –O –h
- **Other formats available (txt, html, csv, xml)**

A Useful Tip. . .

If you don’t see the function you are looking for in a report:

- **Disable pruning: “pat_report –P . . .”**
  - Pruning hides path from sample or event to user source so data is better correlated to user source code
  - For example, hides low level ugni network protocol calls and instead points to MPI call in user source

- **Disable thresholding: “pat_report –T . . .”**
  - Adds back in functions that took insignificant amount of time

Questions About the Data?

- See Job summary information at top of report
- See Details section at bottom of report (may include warnings from CrayPat)
- Check pat_help
- Check man pages

Notes Section

- Check the Notes before each table in the text report

```
Notes for table 5:

  The Total value for Process HiMem (MBytes), Process Time is the avg
  for the PE values.

  The value shown for Process HiMem is calculated from information in
  the /proc/self/numa_maps files captured near the end of the program.
  It is the total size of all pages, including huge pages, that were
  actually mapped into physical memory from both private and shared
  memory segments.

  This table shows only the maximum, median, minimum PE entries,
    sorted by Process Time.
```

Pat_help

- > pat_help environment . . .

```
pat_help environment (.=quit ,=back ^=up /=top ~=search)
=> PAT_RT_SAMPLING_DATA

    Specifies additional data to collect during a sampling
    experiment. The valid values are shown below.

    The value may be followed by '@ratio' which indicates the
    frequency at which the data is sampled. By default the data is
    sampled once for every 100 sampled program counter addresses. For
    example, if 'ratio' is '1', the additional data requested would
    be collected each time the program counter is sampled.
    If the 'ratio' is '1000', the additional data requested would
    be collected once every 1000 program counter samples.

    Collecting additional data during sampling is only supported in
    full-trace mode (see PAT_RT_SUMMARY).

  Additional topics that may follow "PAT_RT_SAMPLING_DATA":

    cray_pm      perfctr
    cray_rapl    rusage
    heap         sheap
    memory
```

Pat_help (2)

- > pat_help environment PAT_RT_SAMPLING_DATA memory

```
pat_help environment PAT_RT_SAMPLING_DATA
(.=quit ,=back ^=up /=top ~=search) => memory

    memory    collect data about the current state of memory

      himem         -  memory high water mark
      rss           -  resident set size
      peak          -  maximum virtual memory used
      priv          -  private resident memory
      shared        -  shared resident memory
      proportional  -  proportional resident memory
```