ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Swiss National Supercomputing Centre

DAY 3: Performance Measurement and Tuning

Multi-threaded Programming, Tuning and Optimization on Multi-core MPP Platforms

15-17 February 2011

CSCS, Manno

Using perftools for threaded and hybrid codes

Roberto Ansaloni
roberto.ansaloni@cray.com

~course02/slides/day2/CrayPerf.pdf

Agenda

- Craypat basics
- Craypat Automatic Performance Analysis
- Craypat Analysis: a slightly different approach
- Detecting load imbalance
- Apprentice^2 basics

Craypat basics

Craypat components

- Craypat module
  - The code to be analyzed must be compiled with perftools module loaded
- pat_build
  - Utility that instruments the application
  - No source code modification required
- Run-time library (transparent to the user)
  - API for performance collection at finer granularity
  - Collects performance data during execution
  - Writes data file

Craypat components

- pat_report
  - Utility to create the performance report
- pat_help
  - Provides craypat info

pat_build: program instrumentation

- pat_build is a stand-alone utility that automatically instruments the application for performance collection
- CrayPat 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
  - trace 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

Sampling experiment

```
Table 1:  Profile by Function Group and Function
 Time % |      Time |Imb. Time |   Imb. |    Calls |Experiment=1
 100.0% | 32.495064 |       -- |     -- | 101546.9 |Total
|-------------------------------------------------------------------
|  90.0% | 29.236753 |       -- |     -- |  44170.0 |USER
||------------------------------------------------------------------
||   9.7% |  3.161488 | 0.886909 |  22.0% |   1002.0 |z_solve_.LOOP@li.39
||   9.6% |  3.119553 | 0.822960 |  20.9% |   1002.0 |y_solve_.LOOP@li.33
||   9.3% |  3.013514 | 0.423609 |  12.3% |   1002.0 |x_solve_.LOOP@li.32
||   8.9% |  2.896714 | 0.898060 |  23.7% |   1004.0 |compute_rhs_.LOOP@li.310
||   8.4% |  2.732843 | 0.578612 |  17.5% |   1004.0 |compute_rhs_.LOOP@li.87
||   8.2% |  2.652253 | 0.698125 |  20.9% |   1004.0 |compute_rhs_.LOOP@li.199
||   6.0% |  1.934034 | 0.414945 |  17.7% |   1004.0 |compute_rhs_.LOOP@li.29
||   5.8% |  1.874099 | 0.416301 |  18.2% |   1002.0 |tzetar_.LOOP@li.30
||   4.9% |  1.595281 | 0.314736 |  16.5% |   1004.0 |compute_rhs_.LOOP@li.61
||   4.3% |  1.397905 | 0.228231 |  14.1% |   1004.0 |compute_rhs_.LOOP@li.392
||   3.8% |  1.231308 | 0.427311 |  25.8% |   1002.0 |txinvr_.LOOP@li.30
||   3.6% |  1.175925 | 0.270148 |  18.7% |   1002.0 |add_.LOOP@li.22
||   1.9% |  0.619119 | 0.289468 |  31.9% |   1004.0 |compute_rhs_.LOOP@li.437
||   1.6% |  0.522802 | 0.168957 |  24.5% |   1002.0 |pinvr_.LOOP@li.22
||   1.6% |  0.514470 | 0.283484 |  35.6% |   1002.0 |ninvr_.LOOP@li.22
||==================================================================
|   8.9% |  2.885000 |       -- |     -- |   6221.9 |MPI
||------------------------------------------------------------------
||   8.6% |  2.791468 | 6.250720 |  69.3% |   2066.6 |mpi_waitall
||==================================================================
|   1.0% |  0.311773 |       -- |     -- |  51140.0 |OMP
|===================================================================
```

Tracing experiment (PAT_RT_HWPC=1)

```
Table 2:  Profile by Function Group and Function

=============================================================================
USER
-----------------------------------------------------------------------------
  Time%                                            88.5%
  Time                                          29.123221 secs
  Imb.Time                                             -- secs
  Imb.Time%                                            --
  Calls                          0.002M/sec        53190.0 calls
  PAPI_L1_DCM                   10.894M/sec      317209739 misses
  PAPI_TLB_DM                    0.343M/sec        9982544 misses
  PAPI_L1_DCA                  848.617M/sec    24709968236 refs
  PAPI_FP_OPS                  784.181M/sec    22833719645 ops
  User time (approx)            29.118 secs    61147745916 cycles  100.0%Time
  Average Time per Call                           0.000548 secs
  CrayPat Overhead : Time          0.2%
  HW FP Ops / User time        784.181M/sec    22833719645 ops   9.3%peak(DP)
  HW FP Ops / WCT              784.038M/sec
  Computational intensity         0.37 ops/cycle    0.92 ops/ref
  MFLOPS (aggregate)         401500.50M/sec
  TLB utilization              2475.32 refs/miss   4.835 avg uses
  D1 cache hit,miss ratios       98.7% hits         1.3% misses
  D1 cache utilization (misses)  77.90 refs/miss   9.737 avg hits
=============================================================================
```

pat_build: options

Option	Description
-w	Make tracing the default experiment and create new trace intercept routines for those function entry points for which no trace intercept routine already exists.
-u	Create new trace intercept routines for those function entry points that are defined in the source file
-o instr_program	The resulting instrumented program. If the -o option is not specified the instrumented program is named program+pat
-t tracefile -T tracefunc	List all function entry points that must be traced or not traced (!function)
-g tracegroup	Instrument the program to trace all function entry point references belonging to the trace function group tracegroup Examples: mpi, libsci, lapack, scalapack, heap
-O apa	Instrument the program for automatic program analysis.

Craypat Automatic Performance Analysis (APA)

APA phase1: generate .apa file

- Load perftools module files and build the app
  - module load perftools
  - make clean; make
- Instrument application for automatic profiling analysis
  - pat_build –O apa a.out => get an instrumented program a.out+pat
- Run the application to get top time consuming routines
  - aprun … a.out+pat
  - You should get a performance file (“<sdatafile>.xf”) or multiple files in a directory <sdatadir>

APA phase1: generate .apa file

- Generate .apa file
  - pat_report –o my_sampling_report [<sdatafile>.xf | <sdatadir>]
  - creates a report file & an automatic profile analysis file <apafile>.apa

APA File Example

```
#  You can edit this file, if desired, and use it
#  to reinstrument the program for tracing like this:
#
#          pat_build -O mhd3d.Oapa.x+4125-401sdt.apa
#
#  These suggested trace options are based on data from:
#
#       /home/crayadm/ldr/mhd3d/run/mhd3d.Oapa.x+4125-
          401sdt.ap2,
        /home/crayadm/ldr/mhd3d/run/mhd3d.Oapa.x+4125-
          401sdt.xf
# ----------------------------------------------------------------------
#       HWPC group to collect by default.
  -Drtenv=PAT_RT_HWPC=1  # Summary with instructions
          metrics.
# ----------------------------------------------------------------------
#       Libraries to trace.
  -g mpi
# ----------------------------------------------------------------------
#       User-defined functions to trace, sorted by % of samples.
#       Limited to top 200. A function is commented out if it has <
          1%
#       of samples, or if a cumulative threshold of 90% has been
          reached,
#       or if it has size < 200 bytes.

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

```
# 43.37%  99659 bytes
        -T mlwxyz_
# 16.09%  17615 bytes
        -T half_
#  7.98%  12666 bytes
        -T full_
#  6.82%  6846 bytes
        -T artv_
. . .
#  1.29%  5352 bytes
        -T currenh_
#  1.03%  25294 bytes
        -T bndbo_
#  Functions below this point account for less than 10% of
         samples.

#  1.03%  31240 bytes
#        -T bndto_
#  0.51%  11169 bytes
#        -T bncto_
. . .
# ----------------------------------------------------------------------
  -o mhd3d.x+apa                # New instrumented program.
  /work/crayadm/ldr/mhd3d/mhd3d.x  # Original program.
```

APA phase2: use.apa file to collect perf data

- Look at <apafile>.apa file
  - Verify if additional instrumentation is wanted
- Instrument application for further analysis (a.out+apa)
  - pat_build –O <apafile>.apa
  - You should get an instrumented program a.out+apa
- Run the application
  - Remember to modify <script> to run a.out+apa
  - aprun … a.out+apa
  - You should get a performance file (“<datafile>.xf”) or multiple files in a directory <datadir>

APA phase2: use.apa file to collect perf data

- Create text report
  - pat_report –o my_text_report.txt [<datafile>.xf | <datadir>]
  - Will generate a compressed performance file (<datafile>.ap2)
- View results in text (my_text_report.txt) and/or with Cray Apprentice2
  - app2 <datafile>.ap2

Craypat Analysis: a slightly different approach

Step1: generate sampling profile

- Load CrayPat & Cray Apprentice2 module files
  - module load xt-craypat apprentice2
  - make clean; make
- Instrument application for sampling
  - pat_build a.out
- Run application to get top time consuming routines
  - aprun … a.out+pat (or qsub <pat script>)
  - You should get a performance file (“<sdatafile>.xf”) or multiple files in a directory <sdatadir>
- Generate sampling report file
  - pat_report –o my_sampling_report [<sdatafile>.xf | <sdatadir>]

Sampling experiment

```
Table 1:  Profile by Function Group and Function
 Time % |      Time |Imb. Time |   Imb. |    Calls |Experiment=1
 100.0% | 32.495064 |       -- |    -- | 101546.9 |Total
|-------------------------------------------------------------------
|  90.0% | 29.236753 |       -- |    -- |  44170.0 |USER
||------------------------------------------------------------------
||   9.7% |  3.161488 | 0.886909 |  22.0% |   1002.0 |z_solve_.LOOP@li.39
||   9.6% |  3.119553 | 0.822960 |  20.9% |   1002.0 |y_solve_.LOOP@li.33
||   9.3% |  3.013514 | 0.423609 |  12.3% |   1002.0 |x_solve_.LOOP@li.32
||   8.9% |  2.896714 | 0.898060 |  23.7% |   1004.0 |compute_rhs_.LOOP@li.310
||   8.4% |  2.732843 | 0.578612 |  17.5% |   1004.0 |compute_rhs_.LOOP@li.87
||   8.2% |  2.652253 | 0.698125 |  20.9% |   1004.0 |compute_rhs_.LOOP@li.199
||   6.0% |  1.934034 | 0.414945 |  17.7% |   1004.0 |compute_rhs_.LOOP@li.29
||   5.8% |  1.874099 | 0.416301 |  18.2% |   1002.0 |tzetar_.LOOP@li.30
||   4.9% |  1.595281 | 0.314736 |  16.5% |   1004.0 |compute_rhs_.LOOP@li.61
||   4.3% |  1.397905 | 0.228231 |  14.1% |   1004.0 |compute_rhs_.LOOP@li.392
||   3.8% |  1.231308 | 0.427311 |  25.8% |   1002.0 |txinvr_.LOOP@li.30
||   3.6% |  1.175925 | 0.270148 |  18.7% |   1002.0 |add_.LOOP@li.22
||   1.9% |  0.619119 | 0.289468 |  31.9% |   1004.0 |compute_rhs_.LOOP@li.437
||   1.6% |  0.522802 | 0.168957 |  24.5% |   1002.0 |pinvr_.LOOP@li.22
||   1.6% |  0.514470 | 0.283484 |  35.6% |   1002.0 |ninvr_.LOOP@li.22
||===================================================================
|   8.9% |  2.885000 |       -- |    -- |   6221.9 |MPI
||------------------------------------------------------------------
||   8.6% |  2.791468 | 6.250720 |  69.3% |   2066.6 |mpi_waitall
||===================================================================
|   1.0% |  0.311773 |       -- |    -- |  51140.0 |OMP
|===================================================================
```

```
$ cat tracefile
z_solve_
y_solve_
x_solve_
compute_rhs_
tzetar_
txinvr_
add_
```

Step2: tracing experiments

- Select user functions from sampling report and create tracefile
- Instrument application for tracing with mpi
  - pat_build –w –t tracefile –g omp –g mpi
- Run application to get HW counter info on specific set of routines
  - export PAT_RT_HWPC=1
  - aprun … a.out+pat

Step3: generating short report

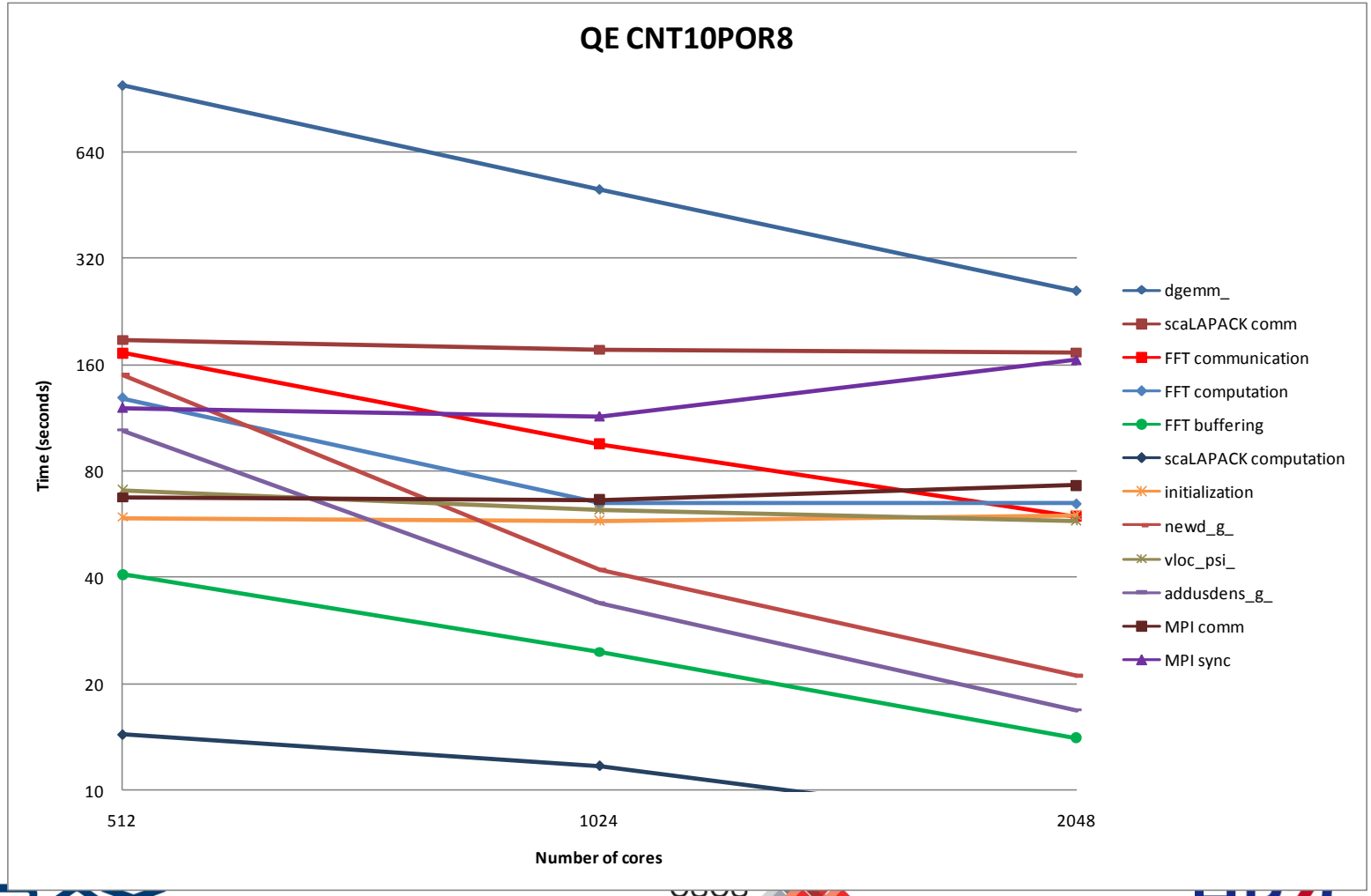
- Generate tracing report files
  - Standard report is ok
- Generate short version
  - Short version options

pat_report -b fu -s show_data="cols" -d time%,cum_time%,time,mflops,flops,PAPI_L1_DCA

Time %	Cum. Time %	Time	MFLOPS (aggregate)	FLOPs	PAPI_L1_DCA	Function
100.0%	100.0%	31474.270087	0.03	46762902584966	60839512818120	Total
10.3%	10.3%	3254.149808	903.15	2942976540672	4406980573687	z_solve_.LOOP@li.39
10.2%	20.6%	3217.860677	911.84	2937747800064	4406987651604	y_solve_.LOOP@li.33
9.8%	30.3%	3078.864701	949.51	2927241068544	4212311246293	x_solve_.LOOP@li.32
9.4%	39.8%	2968.254033	908.27	2700355829760	1727105574764	compute_rhs_.LOOP@li.310
8.8%	48.5%	2756.828633	1369.92	3783459078144	3275994815339	compute_rhs_.LOOP@li.87
8.6%	57.2%	2714.999221	1460.47	3973878906880	2742041536928	compute_rhs_.LOOP@li.199
6.0%	63.1%	1881.378693	251.25	474203111424	435472018512	compute_rhs_.LOOP@li.29
5.9%	69.0%	1854.367212	404.95	752938647552	492226885000	tzetar_.LOOP@li.30
5.3%	74.3%	1665.196359	0.00	0	3268085813848	mpi_waitall
5.0%	79.3%	1559.028802	0.00	0	342460961885	compute_rhs_.LOOP@li.61
4.4%	83.6%	1373.995877	828.76	1142543646720	907423821463	compute_rhs_.LOOP@li.392

Step3: get scalability information

- Repeat step2 varying the number of cores to get scalability info



Further considerations

Pat_build tracegroups (-g)

adios	Adaptable I/O System API
aio	Functions that perform asynchronous I/O.
armci	Aggregate Remote Memory Copy
blacs	**Basic Linear Algebra communication subprograms**
blas	**Basic Linear Algebra subprograms**
caf	**Co-Array Fortran (Cray CCE compiler only)**
chapel	Chapel language compile and runtime library API
dmapp	Distributed Memory Application API for Gemini
ffio	functions that perform Flexible File I/O (Cray CCE compiler only)
fftw	**Fast Fourier Transform library**
ga	Global Arrays API
hdf5	**Hierarchical Data Format library**
heap	**dynamic heap**
io	functions and system calls that perform I/O
lapack	**Linear Algebra Package**
lustre	Lustre File System
math	POSIX.1 math functions
mpi	**MPI**
omp	**OpenMP API**

Pat_build tracegroups (-g) continued

netcdf	Network Common Ddata Form (manages array-oriented scientific data)
pblas	Parallel Basic Linear Algebra Subroutines
petsc	Portable Extensible Toolkit for Scientific Computation. Supported for "real" computations only.
pgas	**Parallel Global Address Space**
pthreads	POSIX threads
realtime	POSIX realtime extensions
scalapack	**Scalable LAPACK**
shmem	SHMEM
stdio	all library functions that accept or return the FILE* construct
syscall	system calls
sysio	system calls that perform I/O
upc	**Unified Parallel C (Cray CCE compiler, versions 7.2 and later only)**

Using Craypat on OpenMP programs

- For OpenMP programs, CrayPat can
  - measure the overhead incurred by entering and leaving parallel regions and work-sharing constructs within parallel regions
  - show per-thread timings and other data
  - calculate the load balance across threads for such constructs.
- For programs that use both MPI and OpenMP,
  - by default compute load balance across all threads in all ranks
  - but one can see load balances for each programming model separately.
- The PGI and Cray compilers automatically insert calls to trace points in the run-time library that support the CrayPat measurements.
  - With other compilers, it is currently the user's responsibility to insert API calls.

Using Craypat MPI statistics

```
MPI Msg Bytes |  MPI Msg |   MsgSz | 16B<= |  256B<= |   4KB<= |Experiment=1
              |    Count |    <16B | MsgSz |   MsgSz |   MsgSz |Function
              |          |   Count | <256B |    <4KB |   <64KB | Caller
              |          |         | Count |   Count |   Count |  PE[mmm]
 3062457144.0 | 144952.0 | 15022.0 |  39.0 | 64522.0 | 65369.0 |Total
|-------------------------------------------------------------------------
| 3059984152.0 | 129926.0 |      -- |  36.0 | 64522.0 | 65368.0 |mpi_isend_
||------------------------------------------------------------------------
|| 1727628971.0 |  63645.1 |      -- |   4.0 | 31817.1 | 31824.0 |MPP_DO_UPDATE_R8_3DV.in.MPP_DOMAINS_MOD
3|              |          |         |       |         |         | MPP_UPDATE_DOMAIN2D_R8_3DV.in.MPP_DOMAINS_MOD
||||----------------------------------------------------------------------
4||| 1680716892.0 |  61909.4 |      -- |    -- | 30949.4 | 30960.0 |DYN_CORE.in.DYN_CORE_MOD
5|||              |          |         |       |         |         | FV_DYNAMICS.in.FV_DYNAMICS_MOD
6|||              |          |         |       |         |         |  ATMOSPHERE.in.ATMOSPHERE_MOD
7|||              |          |         |       |         |         |   MAIN__
8|||              |          |         |       |         |         |    main
|||||||||-------------------------------------------------------------------
9|||||||| 1680756480.0 |  61920.0 |      -- |    -- | 30960.0 | 30960.0 |pe.13666
9|||||||| 1680756480.0 |  61920.0 |      -- |    -- | 30960.0 | 30960.0 |pe.8949
9|||||||| 1651777920.0 |  54180.0 |      -- |    -- | 23220.0 | 30960.0 |pe.12549
|||||||||===================================================================
```

Memory allocation data from Craypat

```
Table 7:  Heap Leaks during Main Program

 Tracked | Tracked | Tracked |Experiment=1
  MBytes |  MBytes | Objects |Caller
     Not |     Not |     Not | PE[mmm]
 Freed % |   Freed |   Freed |

  100.0% | 593.479 |   43673 |Total
|-----------------------------------------
|   97.7% | 579.580 |   43493 |_F90_ALLOCATE
||----------------------------------------
||   61.4% | 364.394 |     106 |SET_DOMAIN2D.in.MPP_DOMAINS_MOD
3|         |         |         | MPP_DEFINE_DOMAINS2D.in.MPP_DOMAINS_MOD
4|         |         |         |  MPP_DEFINE_MOSAIC.in.MPP_DOMAINS_MOD
5|         |         |         |   DOMAIN_DECOMP.in.FV_MP_MOD
6|         |         |         |    RUN_SETUP.in.FV_CONTROL_MOD
7|         |         |         |     FV_INIT.in.FV_CONTROL_MOD
8|         |         |         |      ATMOSPHERE_INIT.in.ATMOSPHERE_MOD
9|         |         |         |       ATMOS_MODEL_INIT.in.ATMOS_MODEL
10         |         |         |        MAIN__
11         |         |         |         main
||||||||||||-----------------------------
12|||||||||    0.0% | 364.395 |     110 |pe.43
12|||||||||    0.0% | 364.394 |     107 |pe.8181
12|||||||||    0.0% | 364.391 |      88 |pe.1047
```

Using Craypat on large numbers of processors

- Pat_report can use an inordinate amount of time on the front-end system
  - Try submitting the pat_report as a batch job
  - Only give Pat_report a subset of the .xf files
    - Pat_report fms_cs_test13.x+apa+25430-12755tdt/*3.xf

Detecting load imbalance

Motivation for Load Imbalance Analysis

- Increasing system software and architecture complexity
  - Current trend in high end computing is to have systems with tens of thousands of processors
  - This is being accentuated with multi-core processors
- Applications have to be very well balanced In order to perform at scale on these MPP systems
  - Efficient application scaling includes a balanced use of requested computing resources
- Desire to minimize computing resource “waste”
  - Identify slower paths through code
  - Identify inefficient “stalls” within an application

Cray Tools Load Imbalance Support

- Very few performance tools focus on load imbalance
  - Need standard metrics
  - Need intuitive way of presentation
- CrayPat support:
  - Imbalance time and %
  - MPI sync time
  - OpenMP Performance Metrics
  - MPI rank placement suggestions
- Cray Apprentice^2 support:
  - Load imbalance visualization

Imbalance Time

- Metric based on execution time
- It is dependent on the type of activity:
  - User functions

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

    Imbalance time = Average time – Minimum time
- Identifies computational code regions and synchronization calls that could benefit most from load balance optimization
- Estimates how much overall program time could be saved if corresponding section of code had a perfect balance
  - Represents upper bound on “potential savings”
  - Assumes other processes are waiting, not doing useful work while slowest member finishes

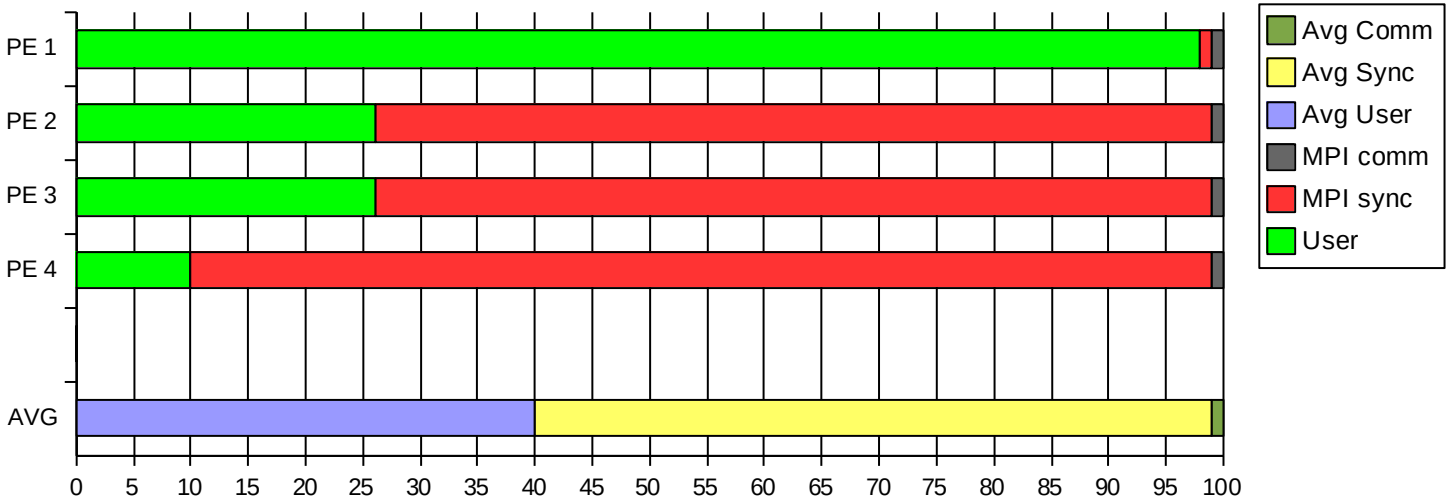
Load balance metric - rationale

Between two barriers

User: Imb = Max-Avg = 99-40 = 59

MPI Sync: Avg = 59

MPI Sync+Comm: Avg-Min = 60-1 = 59



Imbalance %

$$\text{Imbalance\%} = 100 \times \frac{\text{Imbalance time}}{\text{Max Time}} \times \frac{N}{N - 1}$$

- Represents % of resources available for parallelism that is "wasted"
- Corresponds to % of time that rest of team is not engaged in useful work on the given function
- Perfectly balanced code segment has imbalance of 0%
- Serial code segment has imbalance of 100%

Craypat load-imbalance data

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

 Time % |        Time |  Imb. Time |   Imb. |     Calls |Experiment=1
        |             |            | Time % |           |Group
        |             |            |        |           | Function
        |             |            |        |           |  PE='HIDE'

 100.0% | 1061.141647 |         -- |     -- | 3454195.8 |Total
|-------------------------------------------------------------------
|  70.7% |  750.564025 |         -- |     -- |  280169.0 |MPI_SYNC
||------------------------------------------------------------------
||  45.3% |  480.828018 | 163.575446 |  25.4% |   14653.0 |mpi_barrier_(sync)
||  18.4% |  195.548030 |  33.071062 |  14.5% |  257546.0 |mpi_allreduce_(sync)
||   7.0% |   74.187977 |   5.261545 |   6.6% |    7970.0 |mpi_bcast_(sync)
||==================================================================
|  15.2% |  161.166842 |         -- |     -- | 3174022.8 |MPI
||------------------------------------------------------------------
||  10.1% |  106.808182 |   8.237162 |   7.2% |  257546.0 |mpi_allreduce_
||   3.2% |   33.841961 | 342.085777 |  91.0% |  755495.8 |mpi_waitall_
||==================================================================
|  14.1% |  149.410781 |         -- |     -- |       4.0 |USER
||------------------------------------------------------------------
||  14.0% |  148.048597 | 446.124165 |  75.1% |       1.0 |main
|===================================================================
```

MPI Sync Time

- Measure load imbalance in programs instrumented to trace MPI functions to determine if MPI ranks arrive at collectives together
- Separates potential load imbalance from data transfer
- Sync times reported by default if MPI functions traced
- If desired, PAT_RT_MPI_SYNC=0 deactivated this feature

MPI Sync Time Statistics

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

 100.0% | 7.193714 |       -- |     -- | 17604 |Total
|-----------------------------------------------------------
|  76.5% | 5.500078 |       -- |     -- |  4752 |USER
||----------------------------------------------------------
||  96.0% | 5.277791 | 0.171848 |   3.3% |    12 |sweep_
||   3.2% | 0.177352 | 0.005482 |   3.1% |    12 |source_
||   0.3% | 0.018588 | 0.000527 |   2.9% |    12 |flux_err_
||   0.2% | 0.010866 | 0.003033 |  22.8% |  2280 |snd_real_
||   0.1% | 0.005032 | 0.000144 |   2.9% |     1 |initialize_
||   0.1% | 0.004933 | 0.000154 |   3.2% |     1 |initxs_
||   0.1% | 0.002819 | 0.001773 |  40.3% |  2280 |rcv_real_
||==========================================================
|  16.6% | 1.197321 |       -- |     -- |  4603 |MPI
||----------------------------------------------------------
||  93.9% | 1.124227 | 0.277878 |  20.7% |  2280 |mpi_recv_
||   5.9% | 0.070481 | 0.014437 |  17.7% |  2280 |mpi_send_
||   0.2% | 0.002210 | 0.001088 |  34.4% |    32 |mpi_allreduce_
||==========================================================
|   6.3% | 0.453091 |       -- |     -- |    39 |MPI_SYNC
||----------------------------------------------------------
||  61.1% | 0.277012 | 0.215608 |  45.7% |     4 |mpi_bcast_(sync)
||  38.7% | 0.175564 | 0.270049 |  63.2% |    32 |mpi_allreduce_(sync)
||   0.1% | 0.000515 | 0.000265 |  35.5% |     3 |mpi_barrier_(sync)
|==========================================================
```

Apprentice² basics

Roberto Ansaloni
roberto.ansaloni@cray.com

CRAY THE SUPERCOMPUTER COMPANY

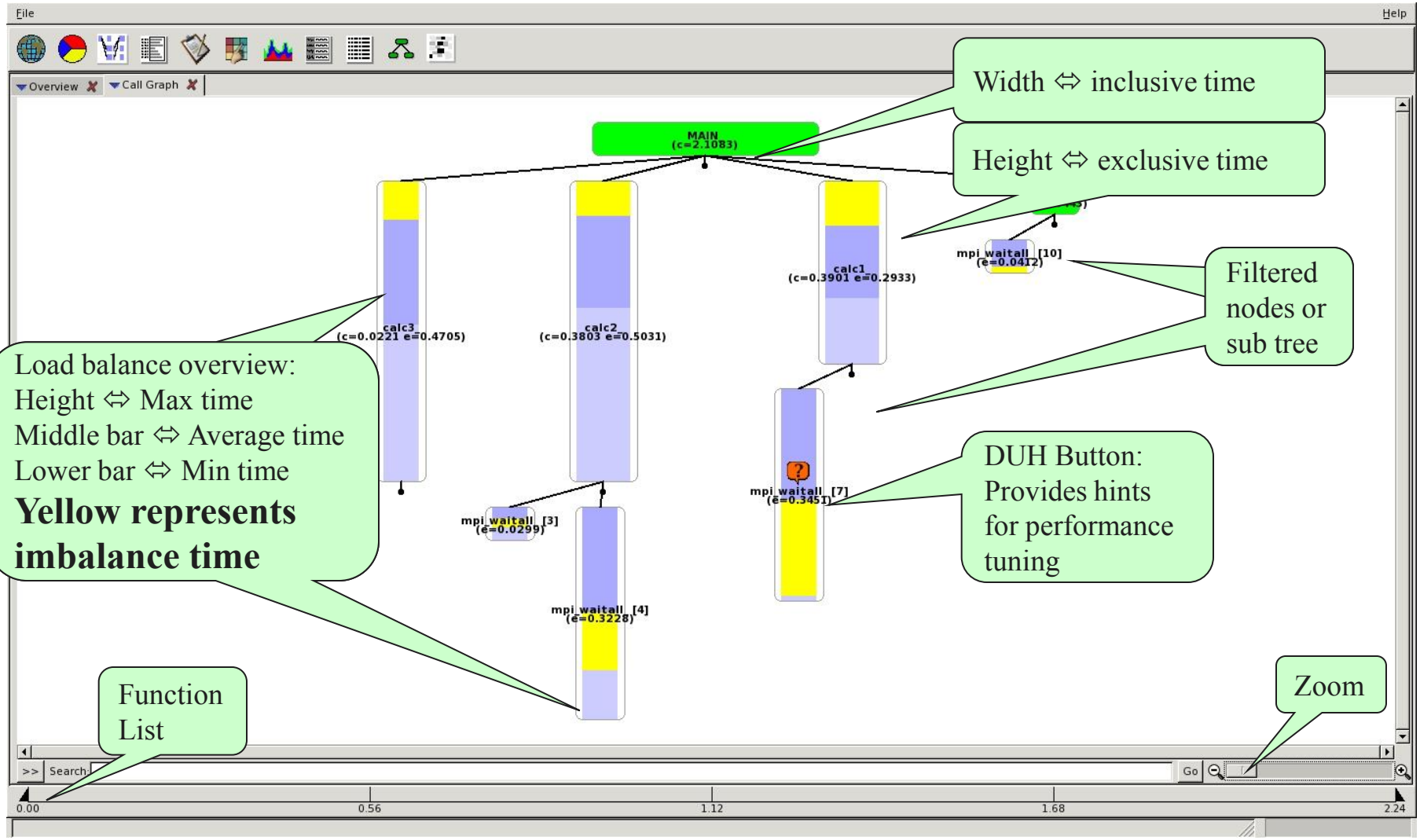
CSCS Swiss National Supercomputing Centre

HP2C

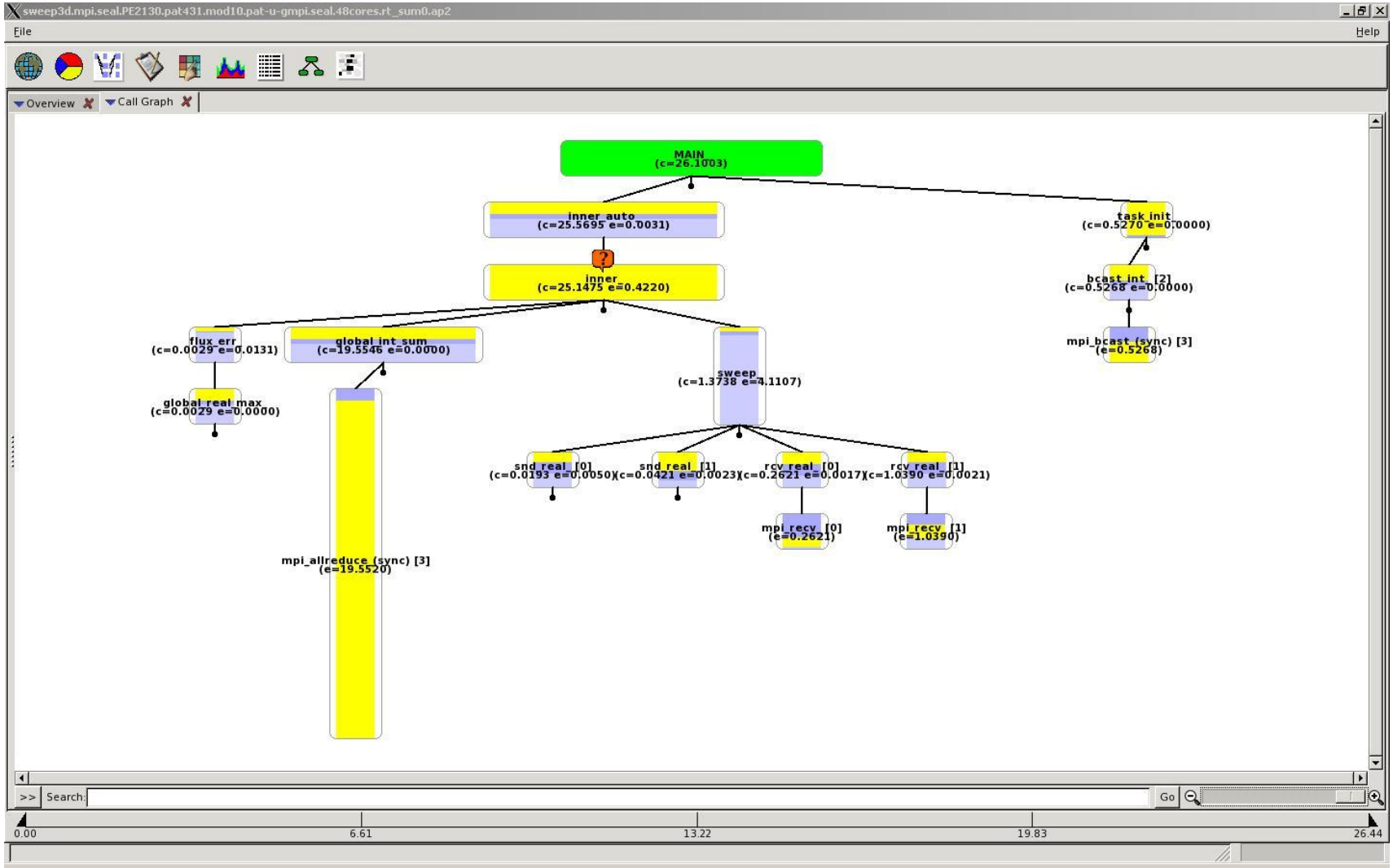
Apprentice^2 howto

- Run the craypat instrumented code
- Generate the .ap2 file
  - First pat_report generates this automatically
  - Otherwise use `pat-report -f ap2`
  - .ap2 file is portable
- Load Apprentice^2 module
  - Automatically loaded by perftools
  - module load perftools
- Start Apprentice^2
  - app2 <file.ap2>

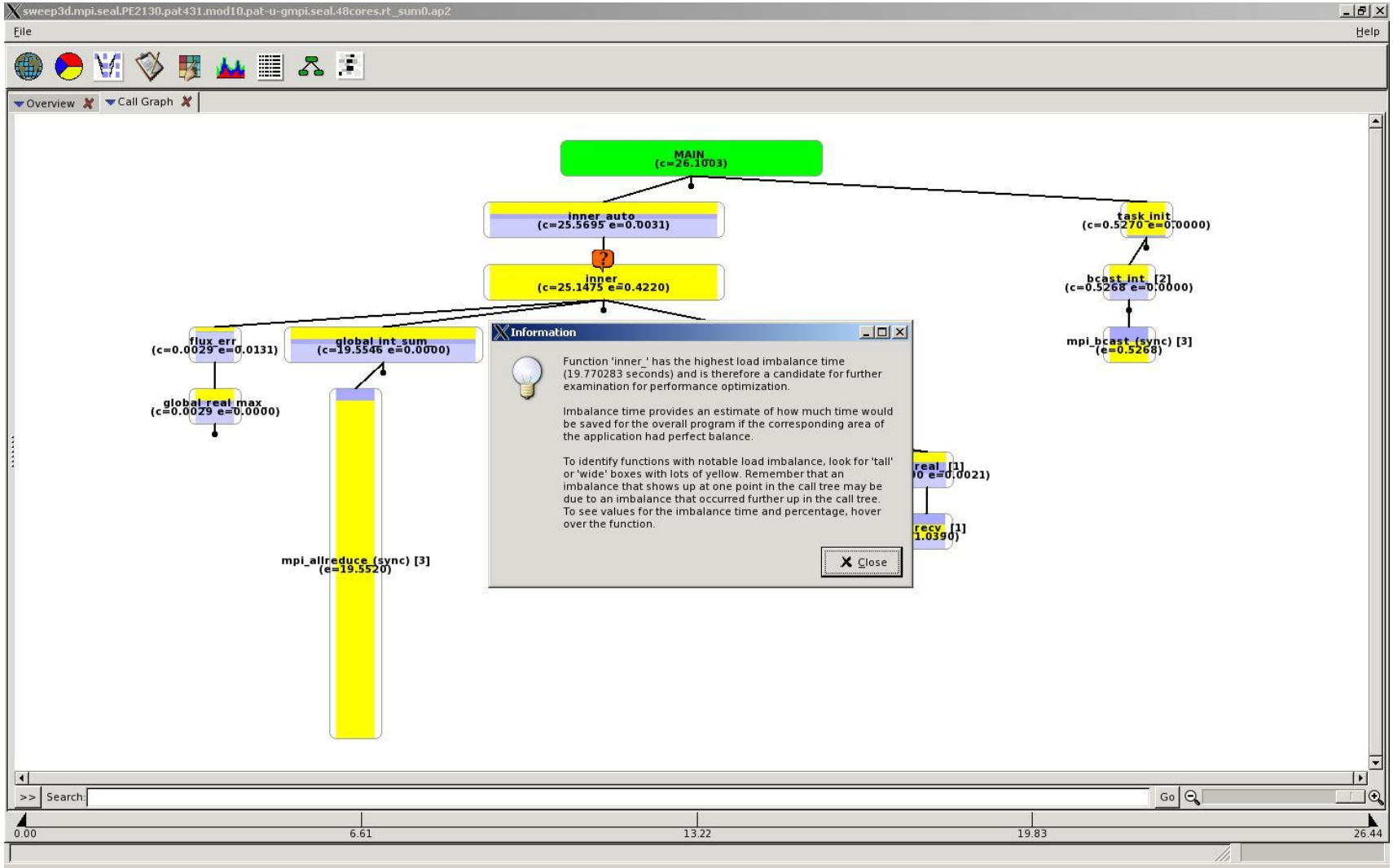
Example Cray Apprentice² Call Tree Display



Apprentice: Call Tree Visualization (Swim3d)



Discrete Unit of Help (DUH Button)

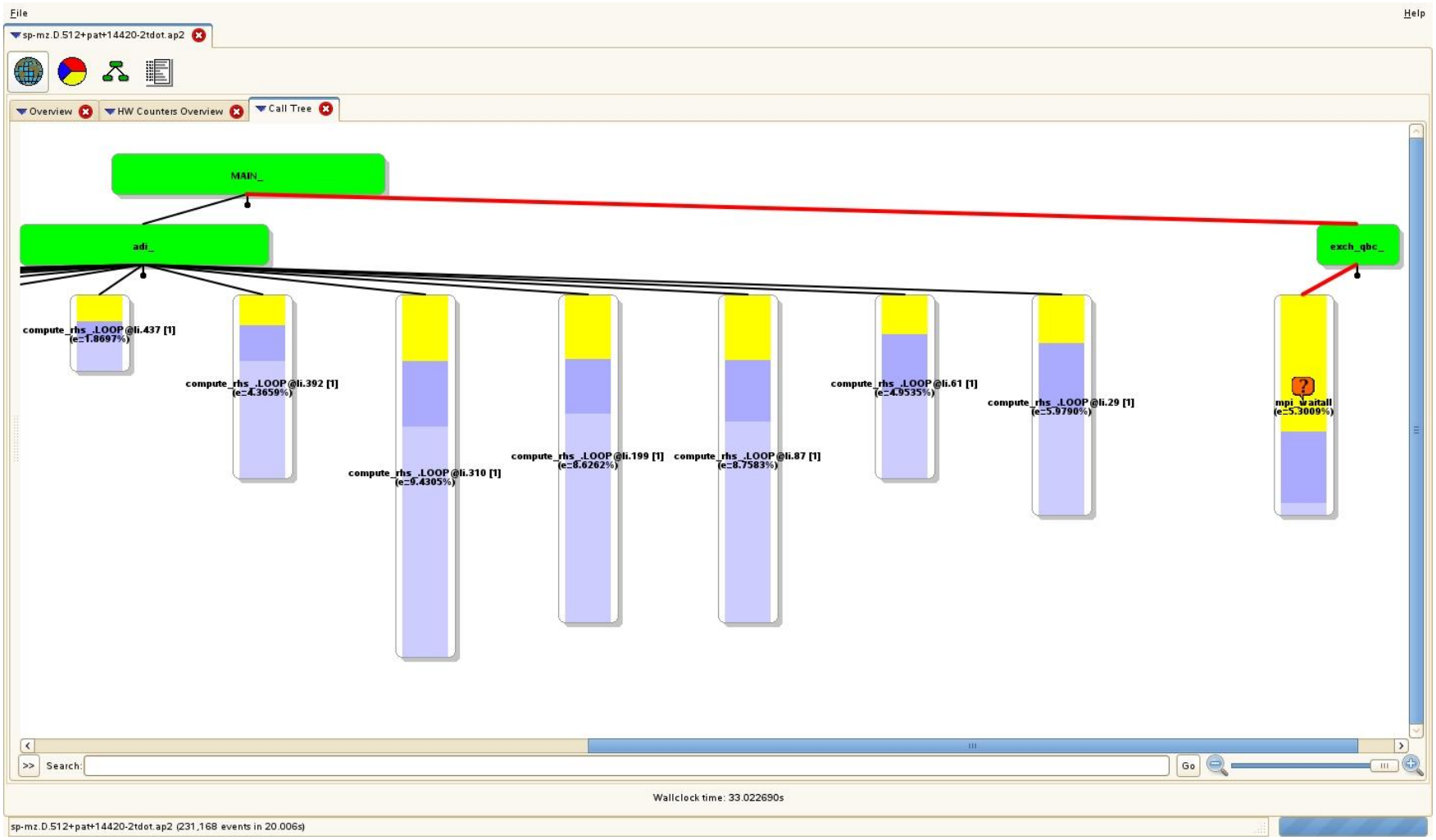


Load Distribution

Apprentice^2 hidden features

- Tools available depend on the experiment
  - For tracing experiments a HW Counter Overview panel is available
- Moving the mouse around or clicking it randomly can produce surprisingly interesting info
- It is possible to get a screendump to JPEG files
  - Look for it in the File menu

Apprentice² screedump



Docs

- Man pages
  - pat_build
  - pat_report
  - hwpc: Predefined hardware performance counter groups
- pat_help
  - Online help system with examples and FAQs
- Apprentice^2 help panel
- Manuals
  - Performance Analysis section on CrayDocs
  - S-2376-51: Using Cray Performance Analysis Tools

Thank you !