Performance Analysis on the Cray XT5

Roberto Ansaloni

(stealing slides from Luiz DeRose presentations)

roberto.ansaloni@cray.com

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 xt-craypat 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
- 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 performance analysis experiments
  - trace experiments (tracing) which count some event such as the number of times a specific system call is executed
  - asynchronous experiments (sampling) which capture values from the call stack or the program counter at specified intervals or when a specified counter overflows
- 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

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 CrayPat & Cray Apprentice2 module files
  - module load xt-craypat apprentice2
- Build application
  - make clean; make
- Instrument application for automatic profiling analysis
  - pat_build –O apa a.out => get an instrumented program a.out+pat
- 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 .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 performance 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 application
  - Remember to modify <script> to run a.out+apa
  - aprun … a.out+apa (or qsub <apa script>)
  - You should get a performance file ("<datafile>.xf") or multiple files in a directory <datadir>
- 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

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

Craypat Hardware Counters statistics

```
USER / MPP_DO_UPDATE_R8_3DV.in.MPP_DOMAINS_MOD
-----------------------------------------------------------------------
  Time%                                              10.2%
  Time                                           49.386043 secs
  Imb.Time                                        1.359548 secs
  Imb.Time%                                           2.7%
  Calls                        167.1 /sec           8176.0 calls
  PAPI_L1_DCM                 10.512M/sec        514376509 misses
  PAPI_TLB_DM                  2.104M/sec        102970863 misses
  PAPI_L1_DCA                155.710M/sec       7619492785 refs
  PAPI_FP_OPS                                            0 ops
  User time (approx)          48.934 secs     112547914072 cycles  99.1%Time
  Average Time per Call                           0.006040 sec
  CrayPat Overhead : Time     0.0%
  HW FP Ops / User time                                  0 ops   0.0%peak(DP)
  HW FP Ops / WCT
  Computational intensity     0.00 ops/cycle          0.00 ops/ref
  MFLOPS (aggregate)          0.00M/sec
  TLB utilization            74.00 refs/miss         0.145 avg uses
  D1 cache hit,miss ratios   93.2% hits               6.8% misses
  D1 cache utilization (M)   14.81 refs/miss         1.852 avg uses
```

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
```

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

Craypat Analysis:
a slightly different approach

Step1: generate sampling profile

- Load CrayPat & Cray Apprentice2 module files
  - module load xt-craypat apprentice2
- Build application
  - 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 output: portals functions

```
 Samp % |     Samp | Imb. |    Imb. |Experiment=1
        |          | Samp | Samp % |Group
        |          |      |         | Function
 100.0% | 8979964 |   -- |      -- |Total
|----------------------------------------------
|  99.6% | 8944835 |   -- |      -- |ETC
||---------------------------------------------
||  12.0% | 1077412 |   -- |      -- |PtlEQPeek
||   6.5% |  580893 |   -- |      -- |MPIDI_CRAY_smpdev_progress
||   6.4% |  575762 |   -- |      -- |fast_nal_poll
||   4.5% |  402805 |   -- |      -- |PtlEQGet
||   4.5% |  400989 |   -- |      -- |ioctl
||   3.9% |  349327 |   -- |      -- |old_ndim_
||   3.8% |  339380 |   -- |      -- |slcomm_
||   3.6% |  319789 |   -- |      -- |PtlEQGet_internal
||   3.2% |  290447 |   -- |      -- |laitri_
||   2.7% |  245147 |   -- |      -- |apl_arome_
```

Step2: tracing experiments

- Select user functions from sampling report and create tracefile
- Instrument application for tracing with mpi
  - pat_build –w –t tracefile –g blas –g lapack –g mpi –o a.out+patt
- Instrument application for tracing without mpi
  - pat_build –w –t tracefile –g blas –g lapack –o a.out+patu
- Run application twice to get HW counter info with or w/o MPI
  - export PAT_RT_HWPC=1
  - aprun … a.out+patt / a.out_patu
- Generate tracing report files
  - Short version options

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

    pat_report -b fu,ca -s show_callers=fu,source,lines -t 95

Tracing output: short report with MPI

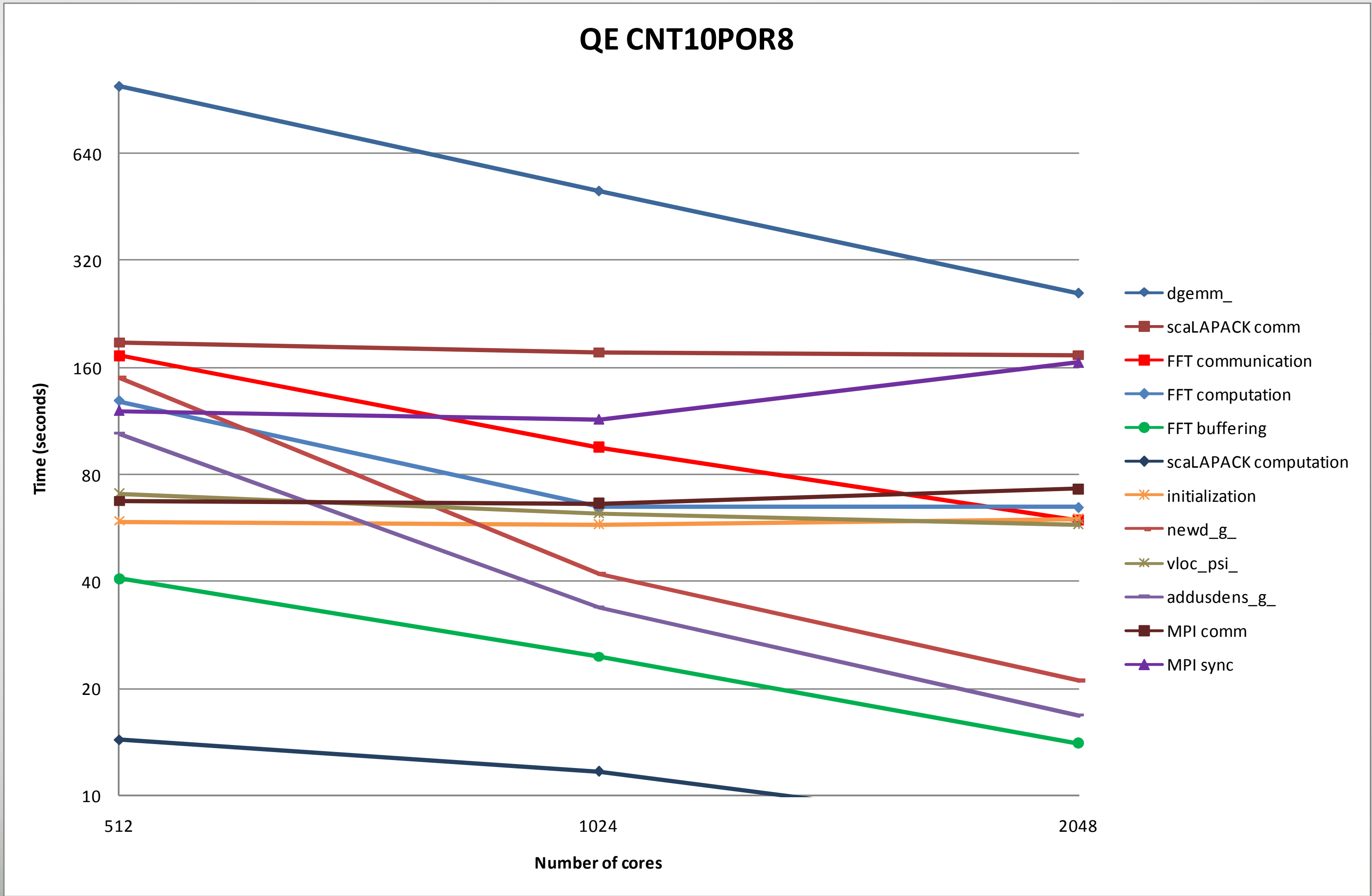
Time %	Cum. Time %	Time	MFLOPS (aggregate)	FLOPs	PAPI_L1_DCA	Function
100.0%	100.0%	171419.527553	122.21	16843617975648	145240582357424	Total
22.6%	22.6%	38715.820831	0.00	0	46219120666276	mpi_recv_
19.7%	42.3%	33830.594736	0.00	0	48651260478731	mpi_barrier_(sync)
8.7%	51.0%	14852.430075	232.98	3445128115608	8030481855315	apl_arome_
4.0%	55.0%	6867.822695	0.00	0	1972321551298	lfildo_
3.8%	58.8%	6586.566755	213.30	771137004076	3354307220006	cnt4_
3.7%	62.6%	6412.979550	0.00	2888000	773616214499	slcomm_
3.7%	66.3%	6386.157229	8.98	18110872000	820770509832	old_ndim_
3.6%	69.9%	6099.787559	293.71	1789103968279	4732650704586	turb_
3.5%	73.3%	5971.108624	251.63	1229696390496	2693086877417	cpg_
2.7%	76.0%	4617.809613	488.84	2255689728000	3215881203507	laitri_
1.8%	77.9%	3164.023323	0.04	111104	1215702680	lfiouv_
1.8%	79.7%	3049.667864	441.22	974558694615	2225536635237	cnt0_
1.8%	81.4%	3032.090787	0.00	0	1365314961	main
1.6%	83.0%	2671.788838	0.03	36792718	750597327487	mpi_bsend_
1.4%	84.4%	2371.023386	703.05	1406044535088	2148927204342	gp_model_
1.4%	85.7%	2353.416490	219.74	516427955045	833040750459	rrtm_rtrn1a_140gp_
1.3%	87.1%	2262.257882	238.74	514871950080	1237593940722	shallow_convection_
1.2%	88.3%	2101.712566	0.00	0	4623295	aroclose_write_cover_tex_
1.1%	89.4%	1920.125393	12.68	14977380000	854640582797	EFTINV_CTL.in.EFTINV_CTL_MOD

Tracing output: short report without MPI

Time %	Cum. Time %	Time	MFLOPS (aggregate)	FLOPs	PAPI_L1_DCA	Function
100.0%	100.0%	160162.835711	131.85	16843579136047	131692317923416	Total
24.0%	24.0%	38454.220908	24.25	771137051518	43553152124665	cnt4_
12.4%	36.4%	19855.691220	0.00	2888000	16835665596483	slcomm_
9.7%	46.1%	15603.169273	0.00	0	20661263228122	etransinv_mdl_
9.3%	55.4%	14836.805645	233.32	3445128115608	8029842410798	apl_arome_
4.8%	60.2%	7708.941506	142.06	974558695900	8977666454554	cnt0_
4.6%	64.8%	7396.158296	2.64	14977380000	6799757758300	EFTINV_CTL.in.EFTINV_CTL_MOD
4.2%	69.1%	6796.044091	0.00	0	1972329619074	lfildo_
4.0%	73.1%	6382.078176	8.96	18110872000	820775933775	old_ndim_
3.8%	76.9%	6098.058552	294.18	1789103968279	4732614214181	turb_
3.7%	80.6%	5962.973039	251.44	1229696390496	2692567585426	cpg_
2.9%	83.5%	4613.248690	490.65	2255689728000	3219332183393	laitri_
2.0%	85.5%	3249.678934	0.00	0	398115722415	main
1.5%	87.0%	2380.079465	0.00	0	4623626	aroclose_write_cover_tex_
1.5%	88.5%	2372.842931	703.19	1406044535088	2149408833341	gp_model_
1.5%	90.0%	2356.639713	219.58	516427955045	833086853295	rrtm_rtrn1a_140gp_
1.4%	91.4%	2319.680864	0.04	111104	1211727777	lfiouv_
1.4%	92.8%	2249.869208	238.76	514871950080	1236511873621	shallow_convection_
1.1%	93.9%	1805.557353	0.00	0	458397849159	fm_read_
0.9%	94.9%	1490.354499	0.00	0	398032346456	fmreadx2_

Step3: get scalability information

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



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

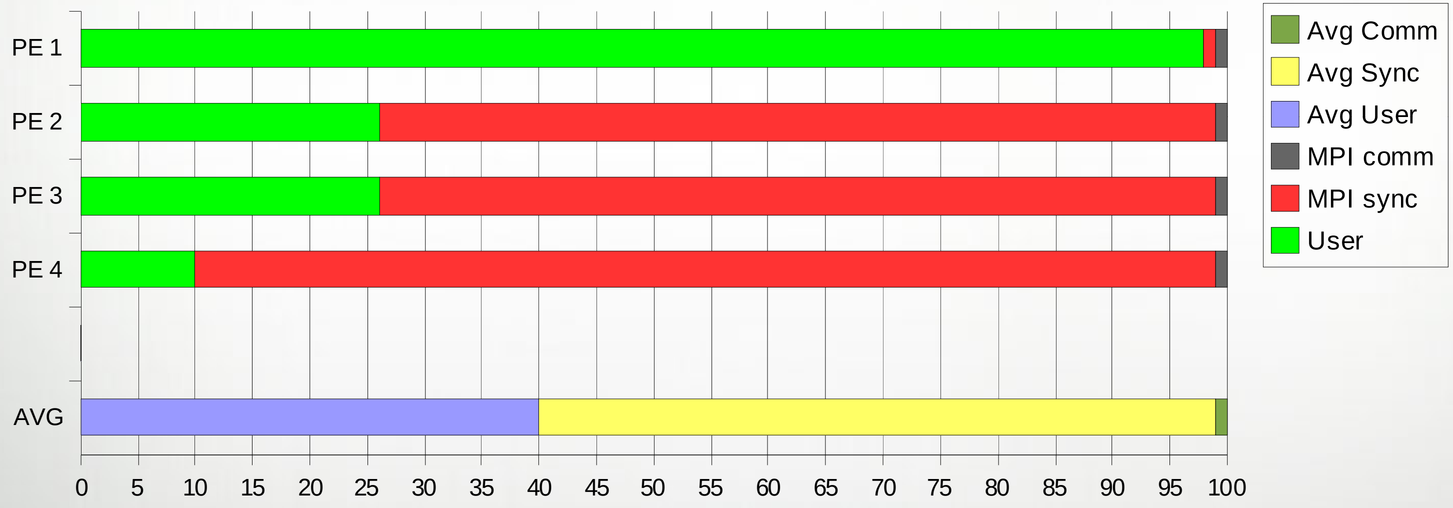
Load balance metric - rationale

Between two barriers

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

MPI Sync: Avg = 59

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

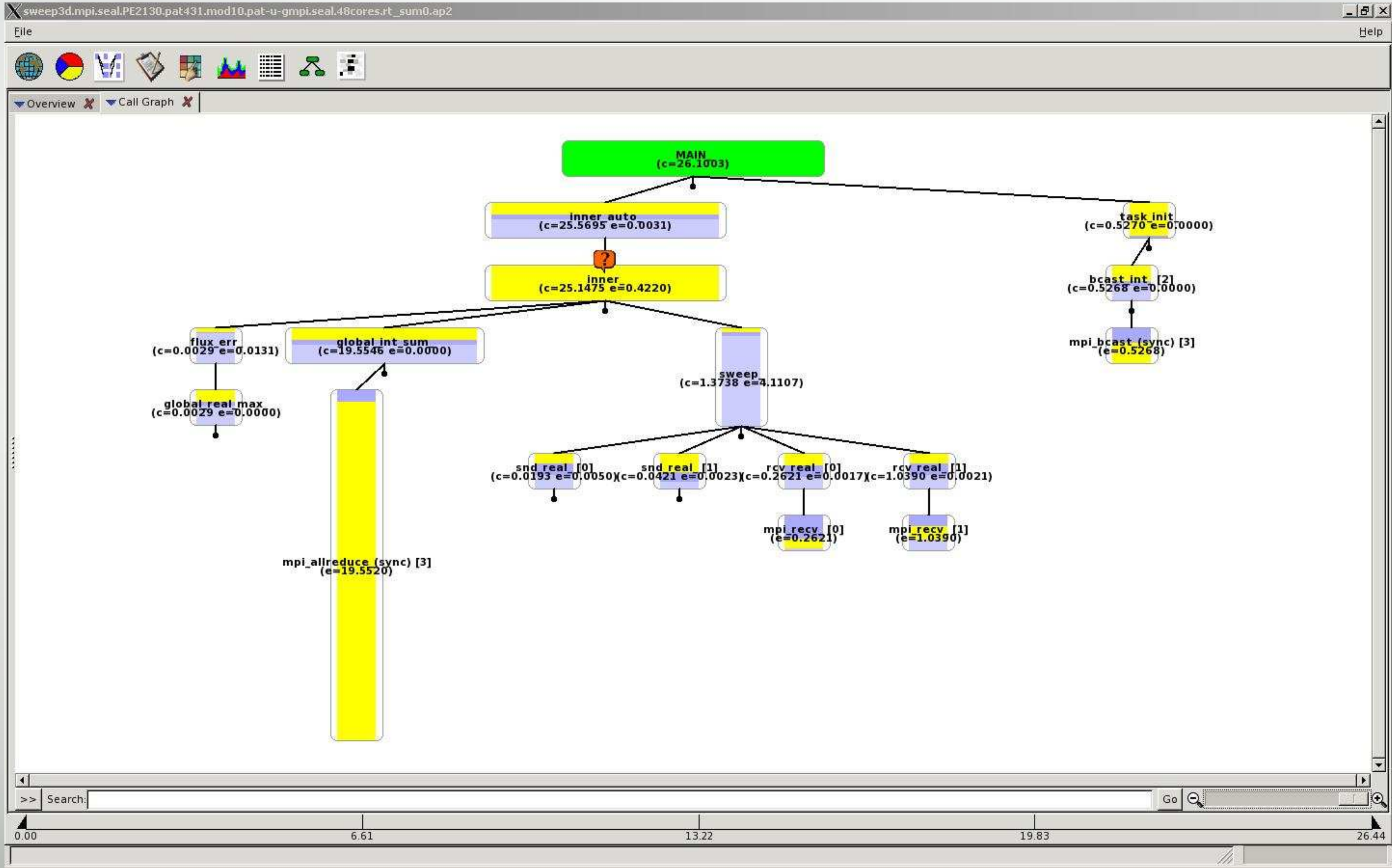


Imbalance %

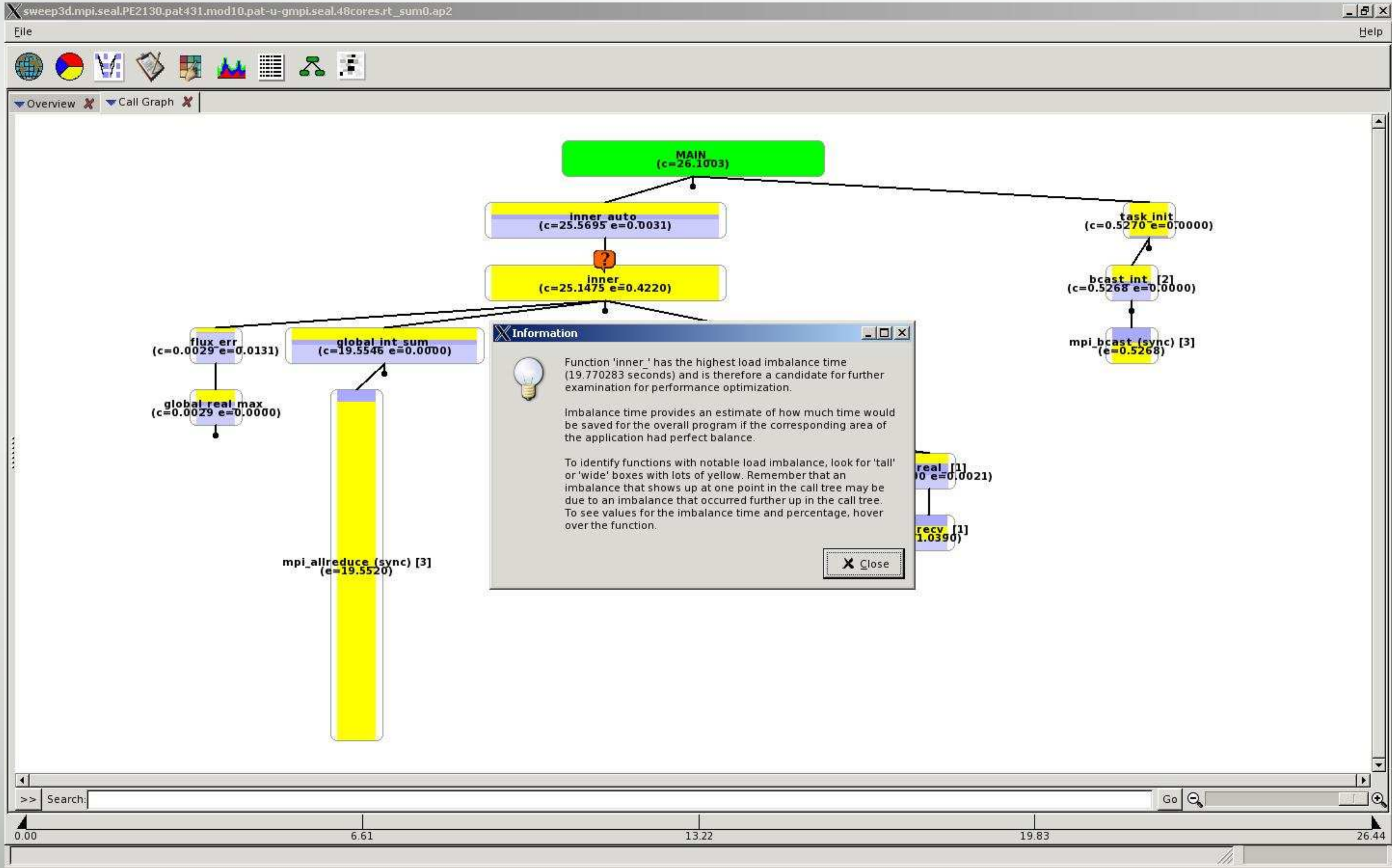
$$\text{Imbalance\%} = 100 \times \frac{\text{Imbalance time}}{\text{Max Time}} \times \frac{N}{N-1}$$

- 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%

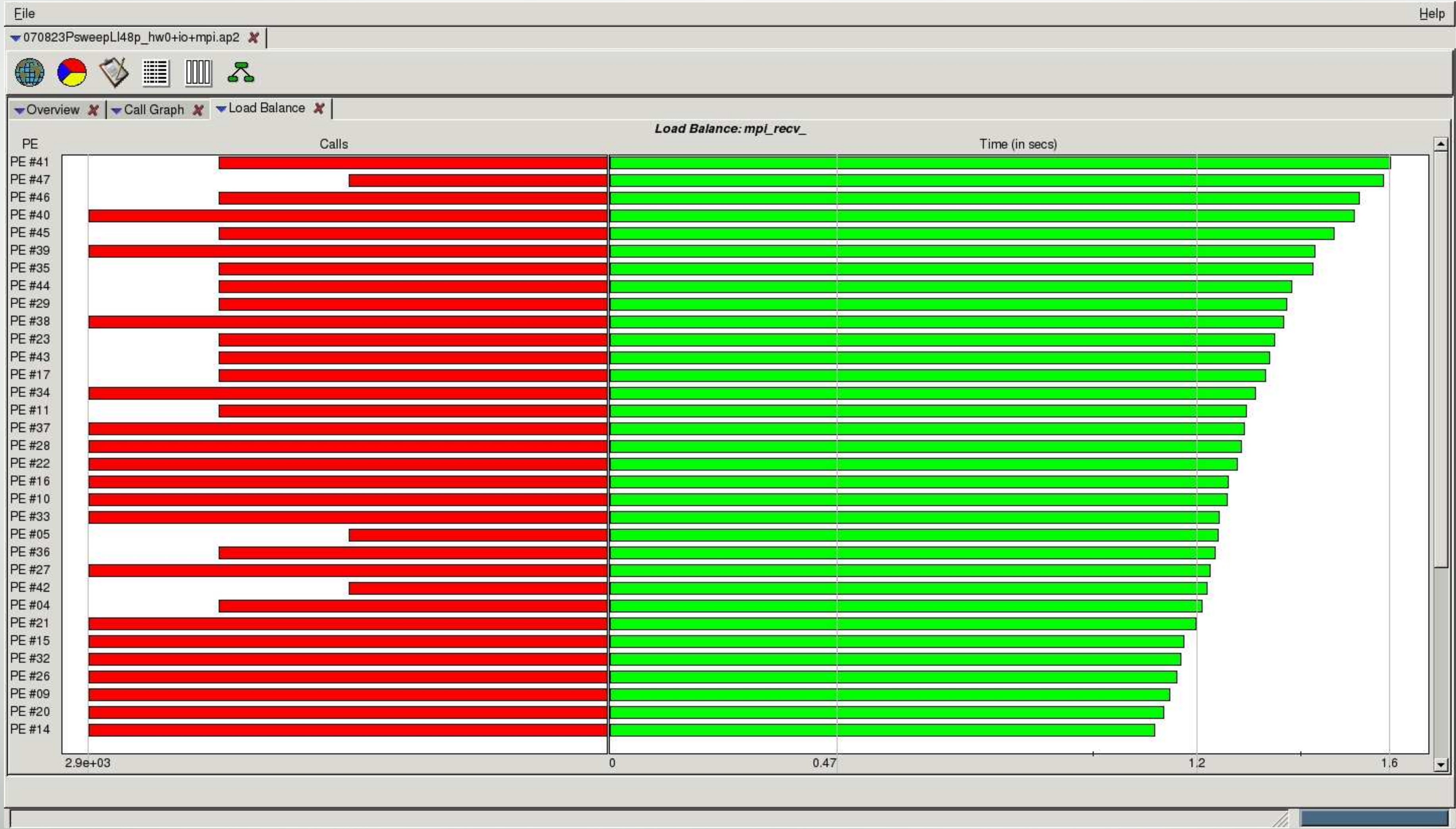
Apprentice: Call Tree Visualization (Swim3d)



Discrete Unit of Help (DUH Button)



Load Distribution



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)
|===========================================================
```

Apprentice^2 basics

Apprentice^2 howto

- Run the craypat instrumented code
- Generate the .ap2 file
  - First pat_report generates this automatically
  - Otherwise use **`pat-report –f ap2`**
- Load Apprentice^2 module
  - module load apprentice2
- Start Apprentice^2
  - app2 <file.ap2>
- Live DEMO

THE SUPERCOMPUTER COMPANY