Hewlett Packard
Enterprise

OpenMP Offload Performance Analysis with “HPE Performance Analysis Tools (PAT)” (formerly CrayPAT)

Marcus Wagner
CORAL-2 Centers of Excellence

August/26/2022

Outline

- This presentation builds on Steven Abbott's presentation "HPE COMPILER GPU OFFLOADING“ https://www.openmp.org/wp-content/uploads/2022-04-29-ECP-OMP-Telecon-HPE-Compiler.pdf
- Documentation Resources
- Disclaimer:
  - This is not a comprehensive reference for, or introduction to anything; instead
  - It is meant to be an easy show-and-tell how to get started with PAT for OpenMP offloading
  - For more details, please, refer to the documentation or, as always - ask, email, discuss
- Claim:
  - I claim ownership of mistakes in this presentation. If you find or suspect a problem, please, let me know.

- What are the "HPE Performance Analysis Tools (PAT)", formerly CrayPAT?
- Build and run an example code (miniQMC - the QMCpack miniapp) with OpenMP offloading to AMD MI250X and show PAT performance analysis using different PAT components.
- Acknowledgements
- The End

Documentation Resources

- https://docs.olcf.ornl.gov/systems/crusher_quick_start_guide.html
- https://www.openmp.org/wp-content/uploads/2022-04-29-ECP-OMP-Telecon-HPE-Compiler.pdf
- https://support.hpe.com # look for:
  - HPE Performance Analysis Tools User Guide (22.06) (S-8014)
  - HPE Cray Fortran Reference Manual (14.0) (S-3901)
  - HPE Cray Clang C and C++ Quick Reference (14.0) (S-2179)
  - # where: 1st (number) = software version, 2nd (S-number) = const. document ID independent of SW version
- man pages relevant in this context:
  - cc, CC, ftn : CCE compiler drivers
  - craycc, crayCC, crayftn : CCE compilers
  - intro_openmp, intro_directives, intro_mpi
  - intro_craypat, pat_build, pat_opts, pat_help, pat_report, pat_run, grid_order, app2, reveal

The Golden Rules of Profiling

- **Profile your code**
  - The compiler/runtime will not do all the optimisation for you.
- **Profile your code yourself**
  - Don't believe what anyone tells you. They're wrong.
- **Profile on the hardware you want to run on**
  - Don't profile on your laptop if you plan to run on a Cray/HPE system;
- **Profile your code running the full-sized problem**
  - The profile will almost certainly be qualitatively different for a test case.
- **Keep profiling your code as you optimise**
  - Concentrate your efforts on the thing that slows your code down.
  - This will change as you optimise.
  - So keep on profiling.

Side-Note on "Obvious" Perf. Hotspot: (crayftn loopmark listing: -h list=a)
Is line 2515 below compute "/" bound? → No, not that obvious after all.☺

```
2508.  + 1--------------------<     do ie = nets, nete
2509.    1                          ! add hyperviscosity to RHS.  apply to Q at timelevel n0, Qdp(n0)/dp
2510.  + 1 2------------------<      do k = kbeg, kend
2511.    1 2 Vps-------------<>        dp(:,:,k) = elem(ie)%derived%dp(:,:,k) - &
                                                   rhs_multiplier*dt*elem(ie)%derived%divdp_proj(:,:,k)
                                        ! Changing "dp = (...)" to "dp = 1./(...)" does not help below
2512.    1 2------------------>      enddo
2513.  + 1 b------------------<      do q = qbeg, qend
2514.  + 1 b b----------------<        do k= kbeg, kend
2515.    1 b b Vps-----------<>          Qtens_biharmonic(:,:,k,q,ie) = &  ! 4.0% of wall time goes into the "/" line
                                           elem(ie)%state%Qdp(:,:,k,q,n0_qdp)/ dp(:,:,k) ! ... but ...
                                                                         ! ... changing "/" to "*" does not help
2516.    1 b b                          if ( rhs_multiplier == 1 ) then
2517.  + 1 b b fVCw----------<>             qmin(k,q,ie)= &
                                              min(qmin(k,q,ie),minval(Qtens_biharmonic(:,:,k,q,ie)))
2518.    1 b b f-------------<>             qmax(k,q,ie)= &
                                              max(qmax(k,q,ie),maxval(Qtens_biharmonic(:,:,k,q,ie)))
2519.    1 b b                          else
2520.  + 1 b b fVCw----------<>             qmin(k,q,ie)=minval(Qtens_biharmonic(:,:,k,q,ie))
2521.    1 b b f-------------<>             qmax(k,q,ie)=maxval(Qtens_biharmonic(:,:,k,q,ie))
2522.    1 b b                          endif
2523.    1 b b---------------->        enddo
2524.    1 b------------------>      enddo
2525.    1-------------------->    enddo
```

Why does "/" → "*" not help? See $FILE_NAME.opt CCE compiler listing, line 2515 (-hlist=d)

```
%kbeg + $I_L2514_1031 + 16 * $I_L2514_1079, hybrid%qbeg + $I_L2513_1060 + 16 * $I_L2513_1085, nets + $I_L2508_1089)
= ( ((elem%base_addr)(nets + $I_L2508_1089, 0)%state%qdp)(1 + $I_L2515_910, 1 + $I_L2515_972, hybrid%kbeg +
$I_L2514_1031 + 16 * $I_L2514_1079, hybrid%qbeg + $I_L2513_1060 + 16 * $I_L2513_1085, n0_qdp) * 1.0 / dp(1 +
$I_L2515_910, 1 + $I_L2515_972, hybrid%kbeg + $I_L2514_1031 + 16 * $I_L2514_1079) )
```

One "/" and many integer operations, incl. some multiplications - array index and offset arithmetic.

This "/" statement is NOT floating-point bound. Memory BW and Integer Ops dominate time.

What are the "HPE Performance Analysis Tools (PAT)", formerly CrayPAT?

- The Performance Analysis Tools (Perftools, PAT, formerly CrayPAT) are a suite of utilities to capture performance data during program execution, and to analyze and visualize those data afterwards.

- PLEASE, use them when running in a parallel file system (gpfs, lustre) but not in $HOME, because a lot of data can be generated, and the system administrator and the other users may smite you with their wrath if you fill up $HOME ☺ ; besides, running in $HOME is also slower.

- PAT can tell you about many things about performance in much detail on- and off-node, such as CPU and GPU performance, cache, OpenMP, MPI, IO and specific packages without having to manually instrument your original source code, but you can (and may want to) do a little instrumentation, e.g., to bypass the initialization phase of your run for taking data, since that may have different performance characteristics than the steady-state run phase of your code.

PAT programming interfaces

- Perftools-lite: Simple interface that produces reports to stdout. There are five Perftools-lite submodules:
  - perftools-lite - Lowest overhead sampling experiment identifies key program bottlenecks.
  - perftools-lite-events - Produces a summarized trace, detailed MPI statistics, including sync. overhead.
  - perftools-lite-loops - Provides loop work estimates (must be used with CCE).
  - perftools-lite-gpu - Focuses on the program's use of GPU accelerators.
  - perftools-lite-hbm - Reports memory traffic info. (must be used with CCE and only for Intel procs).

- Perftools - (Traditional CrayPAT) Advanced interface that provides full data collection and analysis capability, including full traces with timeline displays. It includes the following components:
  - pat_build - Utility that instruments programs for performance data collection.
  - pat_report – After the instrumented program generated by pat_built was run, pat_report can generate text reports from the collected profile data and export the data to other apps.

- Perftools-preload - Runtime instrumentation version of perftools, which eliminates the instrumentation step by pat_build. Not covered here, due to Prep- and Presentation-Time constraints.

And then, there are …

- Apprentice2 (app2)
  - An interactive X Window System tool for visualizing and manipulating performance analysis data captured during program execution. Mac and Windows clients are also available.

  - Installing Apprentice2 on Laptop

```
module load perftools-base/22.06.0  # or your favorite version
module load perftools               # will match above version
cd $CRAYPAT_ROOT/share/desktop_installers
```

  - download Apprentice2Installer-22.06.0-3.exe to laptop # windows
  - download Apprentice2Installer-22.06.0-3.dmg to laptop # apple
  - double-click on installer on laptop and follow directions to install

- Reveal
  - Source code visualization and analysis tool, good to point out where to add what OpenMP directives, but so far focused on CPU. Not covered here, due to Prep- and Presentation-Time constraints.

PAT Overview

- PAT assist the user with application performance analysis and optimization
  - Provides concrete suggestions instead of just reporting data.
  - Work on user codes at realistic core counts with thousands of processes/threads integrate into large codes with millions of lines of code
  - (To optimize for the BlueWaters/NCSA acceptance, I ran VPIC with CrayPAT in 2013 on 180224 MPI-ranks with 4 OpenMP-threads on 720896 cores, 22528 nodes. This is a data point which shows that PAT can scale.
    Fine-print: The 1-hour time limit on interactive sessions was too short to read in all *.xf profile data from that run.
    But pat_report can be run from within a batch job. ☺
    Keep that in mind, though, if you run a job on many nodes with perftools-lite-XXX where the equiv. of pat_report is automatically done at the end on the first allocated node only – while the other still allocated nodes remain idle.)

- PAT is a universal tool (different compilers, hardware, performance aspects – io, communication, compute, memory, on-node, inter-node)
  - Basic functionality available to all compilers on the system
  - Additional functionality available for the Cray compiler (loop profiling)
  - Requires no source code or Makefile modification
  - Automatic instrumentation at group (function) level such as mpi, io, omp (see `man pat_build` -g trace-group)
  - Requires object files and archives for instrumentation and to be compiled with the {cc, CC, ftn} drivers while a perftools module was loaded. (If you really don't want to use {cc,CC,ftn} - a workaround shown later.)

PAT Overview (cont.)

- PAT is able to generate instrumentation on optimized code.
- It’s not necessary or helpful to add extra "-G/-g" flags for performance analysis with PAT,
  because then you would profile a code you normally don't run;
  you want to know the hot-spots in the optimized code.
- Instead, build your code as usual and let pat_build do the instrumentation of your optimized code.
- pat_build creates new stand-alone instrumented program while preserving original binary.

- Side note 1: In general for CCE, be careful with the "-g" compilation flag which corresponds to "-G0",
  because –g is heavy-handed (more so than with other compilers) and will turn off optimization.
  Instead, try "-G2" or, if that's not enough, "-G1".

- Side note 2: When you build an application with a perftools module loaded
  (except for the perftools-base/* modules where this is not the case and this note does not apply)
  PAT will keep some temp files in $HOME/.craypat/
  Therefore, you may want to periodically clean out old files under $HOME/.craypat/

Sampling and Event Tracing

- CrayPAT provides two fundamental ways of profiling:

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

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

Sampling vs. Tracing

- Sampling
  - Advantages
    - Only need to instrument main routine
    - Low Overhead - depends only on sampling frequency
    - Smaller volumes of data produced
  - Disadvantages
    - Only statistical averages available
    - Limited information from performance counters

- Event Tracing
  - Advantages
    - More accurate and more detailed information
    - Data collected from every traced function call not statistical averages
  - Disadvantages
    - Increased overheads as number of function calls increases
    - Potentially huge volumes of data generated

- Automatic Profile Analysis (APA) combines the two approaches.

Examples of Doing a Sampling vs. Tracing Experiment

- module load perftools-base
- module load perftools
- make clean
- make
- Sampling:
  - pat_build a.out # generates a.out+pat
- Tracing:
  - pat_build -w -g mpi,omp,io,hip a.out # generates a.out+pat
- srun [srun opts] a.out+pat [a.out opts]
- pat_report -i a.out+pat experiment-data-dir > pat_report.out.dflt
  - where experiment-data-dir looks like <my_prog>+pat+<PID>-<node>[s|t]
  - first pat_report invocation generates *.ap2 files from *.xf files – don't need to keep those *.xf files anymore
- or, e.g.,
- pat_report -O acc_time -s show_ca=fu,so,li experiment-data-dir
  - for accelerator kernels also showing callers by function, source, line
- `pat_report -O -h` displays a list of all pre-defined report options that pat_report provides with a short description

An Example of some Output from a (dated) Sampling Experiment

```
$> make
INFO: A maximum of 51 functions from group 'io' will be traced.
INFO: A maximum of 208 functions from group 'mpi' will be traced.
INFO: A maximum of 20 functions from group 'realtime' will be traced.
INFO: A maximum of 56 functions from group 'syscall' will be traced.
INFO: creating the CrayPat-instrumented executable '/a/certain/dir/cp2k.pdbg'
  (sample_profile) ...OK

> cat job.out
#########################################################################
#                                                                       #
#            CrayPat-lite Performance Statistics                        #
#                                                                       #
#########################################################################

CrayPat/X:  Version 6.3.0 Revision 14378 (xf 14041)  09/15/15 10:48:06
Experiment:                  lite  lite/sample_profile
Number of PEs (MPI ranks):     48
Numbers of PEs per Node:       24  PEs on each of  2  Nodes
Numbers of Threads per PE:      1
Number of Cores per Socket:    12
Execution start time:  Wed Oct 14 14:07:17 2015
System name and speed:  mom11 2501 MHz

Avg Process Time:          5.14 secs
High Memory:               2,070 MBytes  43.13 MBytes per PE
MFLOPS:           Not supported (see observation below)
I/O Read Rate:          4.803892 MBytes/sec
I/O Write Rate:        88.963763 MBytes/sec
Avg CPU Energy:            1,499 joules 749.50 joules per node
Avg CPU Power:           291.59 watts  145.80 watts per node
```

```
...
Table 1:  Profile by Function Group and Function (top 8 functions shown)

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

 100.0% | 263.4 |    -- |    -- |Total
|-------------------------------------------------------------------------
|  78.0% | 205.3 |    -- |    -- |MPI
||------------------------------------------------------------------------
||  62.4% | 164.4 | 115.6 | 42.2% |mpi_bcast
||  10.4% |  27.4 | 186.6 | 89.1% |MPI_ALLREDUCE
||   4.7% |  12.4 |  86.6 | 89.3% |MPI_IPROBE
||========================================================================
|  13.1% |  34.5 |    -- |    -- |USER
||------------------------------------------------------------------------
||   3.3% |   8.6 |  61.4 | 89.5% |__message_passing_MOD_mp_probe
||   2.8% |   7.5 |   8.5 | 54.4% |__fist_nonbond_force_MOD_force_nonbond
||   2.0% |   5.2 |   5.8 | 53.6% |__ewalds_MOD_ewald_evaluate
||   1.1% |   2.9 |   3.1 | 52.5% |__splines_methods_MOD_potential_s
||========================================================================
|   8.2% |  21.5 |    -- |    -- |ETC
||------------------------------------------------------------------------
||   2.5% |   6.6 |   9.4 | 59.7% |__memmove_ssse3
||   1.7% |   4.4 |   4.6 | 52.7% |__memset_sse2
|=========================================================================.....
```

An Example of some Output from a (dated) Trace Experiment

```
> cat job.out
...
Table 1: Profile by Function Group and Function (top 4 functions shown)

  Time% |     Time |     Imb. |   Imb. |     Calls |Group
        |          |     Time |  Time% |           | Function
        |          |          |        |           |  PE=HIDE
 100.0% | 3.075490 |       -- |     -- | 562,739.2 |Total
|-------------------------------------------------------------------
|  74.2% | 2.282250 |       -- |     -- |   9,855.8 |MPI_SYNC
||------------------------------------------------------------------
||  50.8% | 1.562708 | 1.551026 |  99.3% |   3,131.2 |mpi_bcast_(sync)
||  12.9% | 0.396947 | 0.396920 | 100.0% |       1.0 |mpi_init_(sync)
||  10.5% | 0.322147 | 0.293341 |  91.1% |   6,721.6 |mpi_allred_(sync)
||==================================================================
|  19.2% | 0.590622 |       -- |     -- |       2.0 |USER
||------------------------------------------------------------------
||  19.2% | 0.590584 | 0.661898 |  54.0% |       1.0 |main
||==================================================================
|   5.4% | 0.166062 |       -- |     -- | 552,576.7 |MPI
||------------------------------------------------------------------
||   4.1% | 0.126472 | 0.779788 |  87.9% | 541,104.1 |MPI_IPROBE
|===================================================================
...
```

Load-Imbalance – What is it? Do we care?

- It’s the fraction of time that rest of team is not engaged in useful work on the given function.
- Identifies computational code regions and synchronization calls that could benefit most from load balance optimization.
- Estimates how much overall time could be saved if corresponding code had a perfect balance.
- Represents an upper bound on "potential savings“ due to better load-balancing.
- Assumes other processes are waiting, not doing useful work while slowest member finishes.
- Perfectly balanced code segment has imbalance of zero.

- `Imbalance time = (Maximum - Average) time  # user functions`
- `Imbalance time = (Average - Minimum) time  # MPI sync + barrier`

- `Imbalance% = 100% * [(Imbal/Max)time] * [ntasks/(ntasks-1)]`

Automatic Profile Analysis (APA)

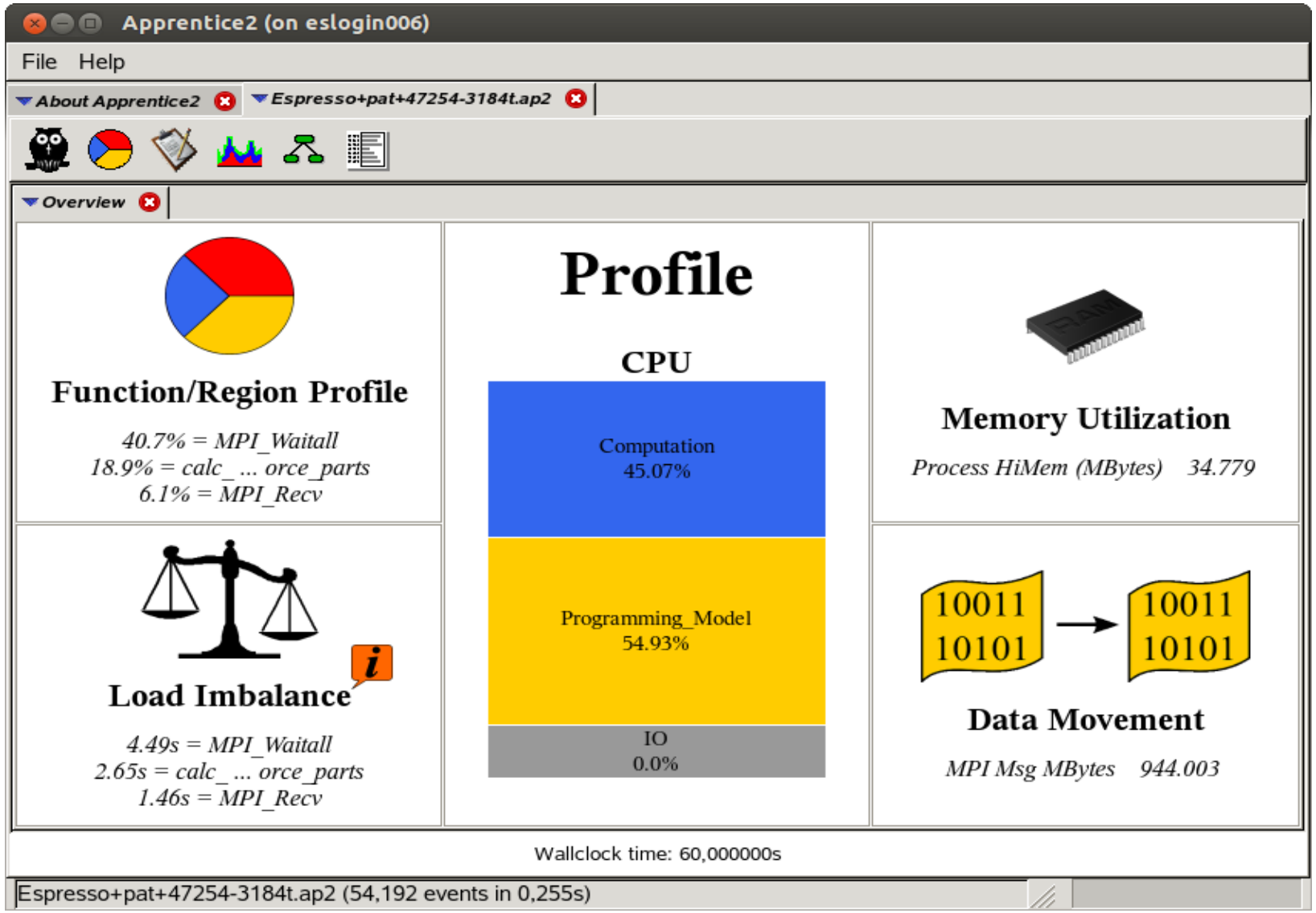
- What if you have to optimize a code you don't know in less time than you have and you don't even know enough about it to ask productive questions?
  - Let PAT tell you what it thinks you should look at.

```
module load perftools-base ; module load perftools
make clean ; make
pat_build a.out
srun a.out+pat
pat_report –i a.out+pat exp-data-dir > pat_report.out.dflt
pat_build –O exp-data-dir/build-options.apa
# The text file *.apa contains instructions for pat_build
# what to instrument and how, based on the first profile run with a.out+pat.
# The 1st sampling run tells PAT what it should instrument for tracing in the 2nd run.
srun a.out+apa
pat_report exp+apa+data-dir > pat_report.out.apa.dflt
```

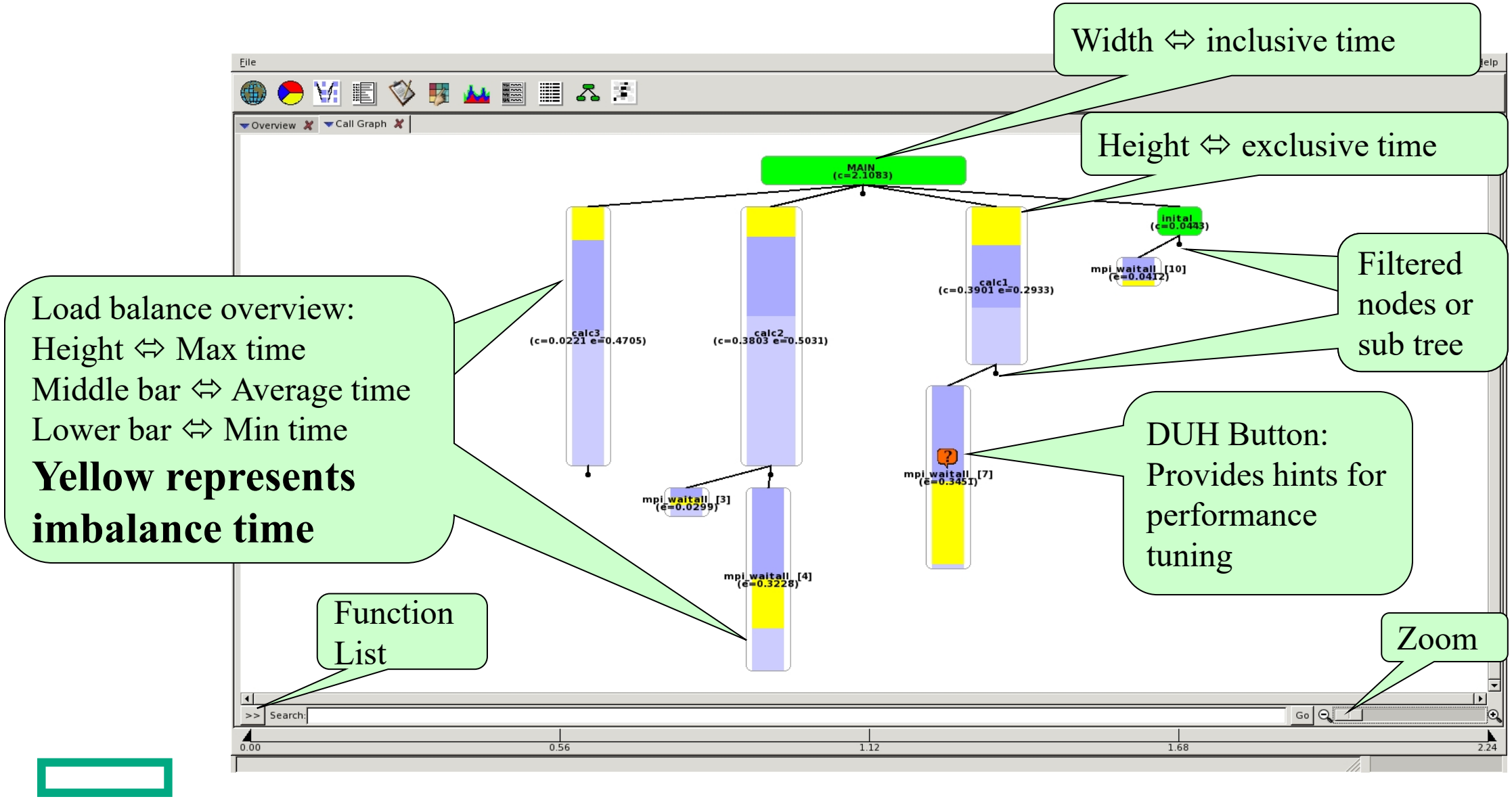
Apprentice2 – Graphic Representation of Performance Data

- Apprentice2 is a post-processing performance data visualization tool; takes *.ap2 files as input.
- Main features:
  - Call graph profile
  - Communication statistics
  - Time-line view for Communication and IO.
  - Activity view
  - Pair-wise communication statistics
  - Text reports
- Helps identify:
  - Load imbalance
  - Excessive communication
  - Network contention
  - Excessive serialization
  - I/O Problems
- module load perftools-base
  - App2 my_program.ap2 # opens an X-window ; if no X-forwarding then
    install app2 on laptop, download *.ap2 to laptop and open *.ap2 in app2-GUI

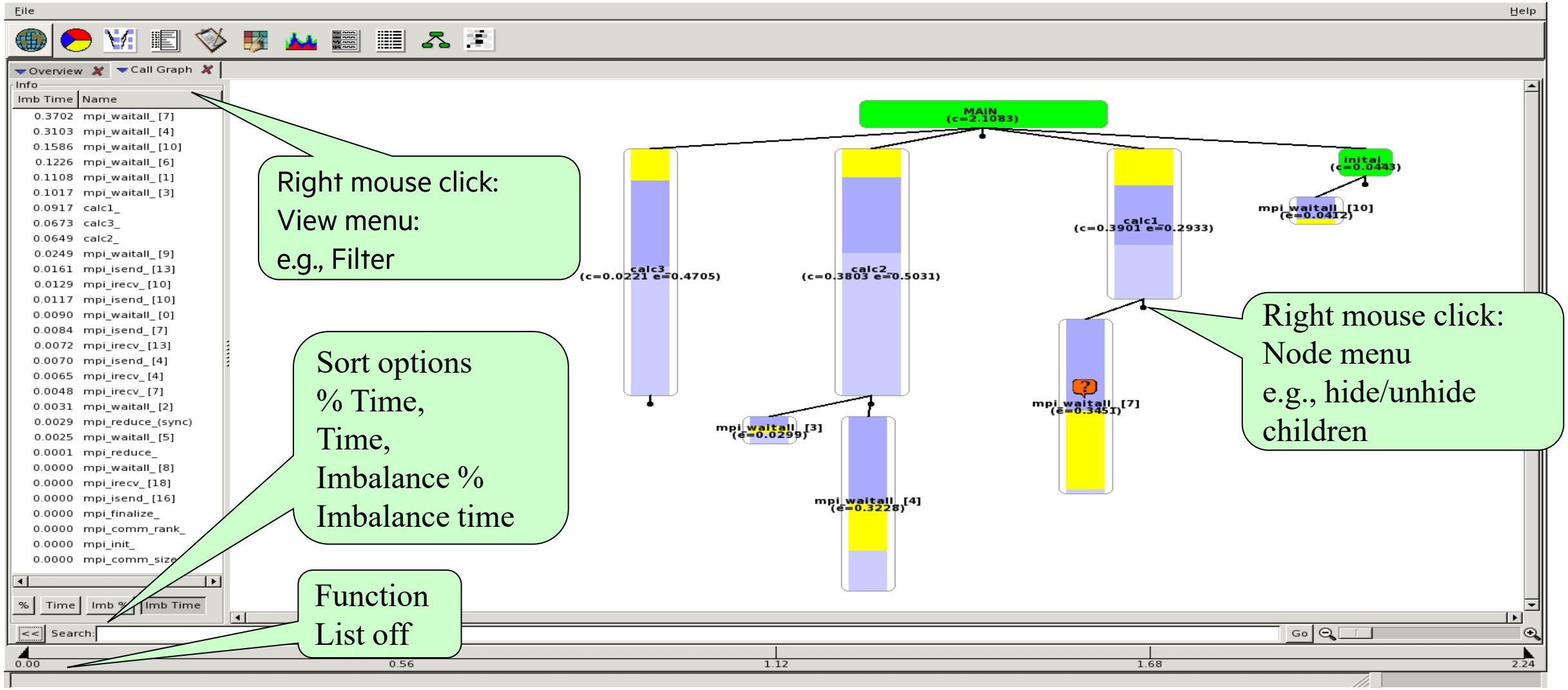
CRAY APPRENTICE²



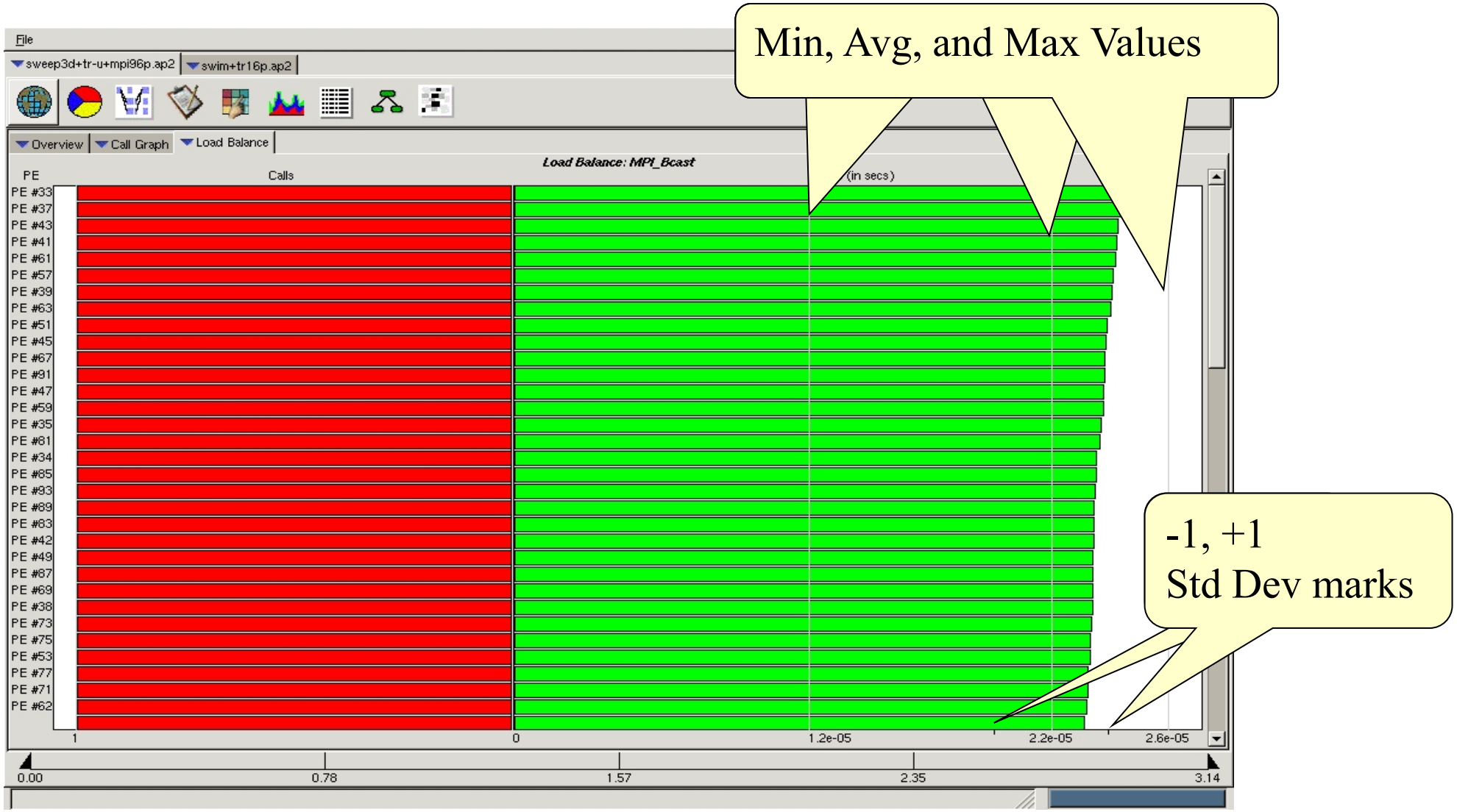
CALL TREE VIEW



CALL TREE VIEW – FUNCTION LIST

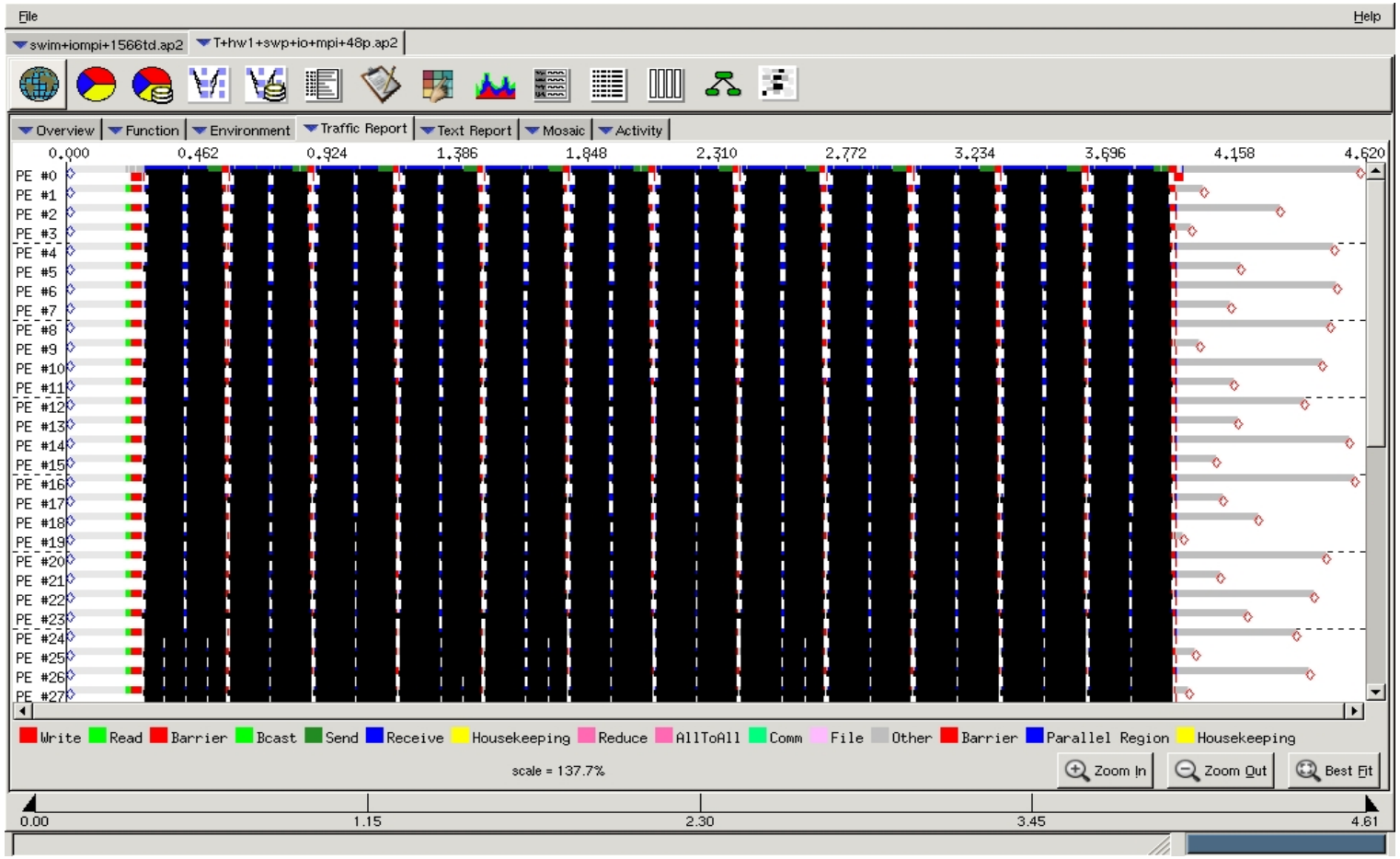


LOAD BALANCE VIEW (FROM CALL TREE)

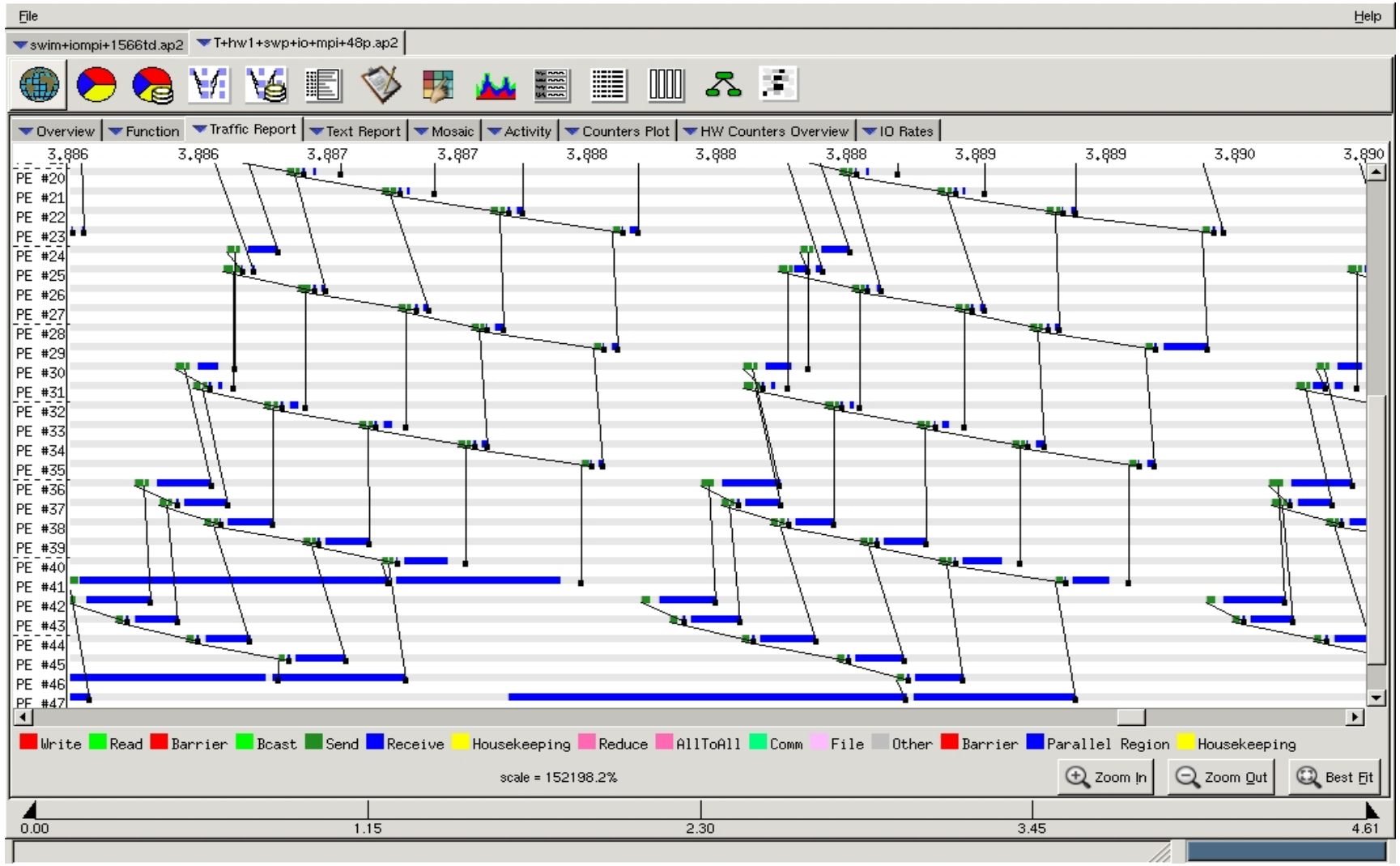


TIME LINE VIEW

- Full trace (sequence of events) enabled by setting `PAT_RT_SUMMARY=0`
- Helpful to see communication bottlenecks.
- Use it only for small experiments !



TIME LINE VIEW (FINE GRAIN ZOOM)



Misc. PAT Topics: More MPI-rank specific info, e.g., for Apprentice2 Timeline

- If you are doing tracing but don't get as much detailed information per MPI-rank as expected, consider export PAT_RT_SUMMARY=0
- However, this will collect SIGNIFICANTLY more data during the run.
- E.g., from the "same" 8-MPI-rank ~2min perftools-lite-gpu run of miniqmc:
  - % du -sh miniqmc+*
  - 18G miniqmc+104730-6078151t.run_18 # export PAT_RT_SUMMARY=0
  - YES, that’s 18 GB counter data from 1 compute-node in 2 minutes code wallclock time.
  - 15M miniqmc+62750-6079239t.run_17   # default PAT_RT_SUMMMARY=1
  - Again, please, don't try that at $HOME ☺ - use gpfs/lustres instead.

Get and Build miniqmc for offloading

- Download and install miniqmc-OMP_offload.zip from /* git clone did not work for me */ https://github.com/QMCPACK/miniqmc/wiki/OpenMP-offload#build-recipes

- Set up module environment on crusher, such as
- ```
  % . set_mod_env.cce.uninst
  % module -t list 2>&1 | pr -T --columns=2
  craype-x86-trento                  PrgEnv-cray/8.3.3
  libfabric/1.15.0.0                 cce/14.0.2
  craype-network-ofi                 DefApps/default
  xpmem/2.4.4-2.3_2.12__gff0e1d9.shas craype-accel-amd-gfx90a
  cray-pmi/6.1.2                     cray-mpich/8.1.18
  craype/2.7.15                      rocm/5.1.0
  cray-dsmml/0.2.2                   cmake/3.23.2
  cray-libsci/21.08.1.2              perftools-base/22.06.0
  ```
- Then add either module perftools-lite-gpu xor perftools (depending on what you want to do) but not both modules at once


  ```

Build and Run miniqmc with perftools-lite-gpu

- Acc. to https://github.com/QMCPACK/miniqmc/wiki/OpenMP-offload#build-recipes building with Cray Clang has been done, but only up to v9.0 (now we have 14.0) and with craype-accel-nvidia60, not with craype-accel-amd-gfx90a .
- How hard can that porting be? If OpenMP offloading works - not very.
- After extraction of miniqmc-OMP_offload, cd miniqmc-OMP_offload

1. Change CMakeLists.txt to allow for adding some flags:

```
% diff CMakeLists.txt.orig CMakeLists.txt.loopmark.add_cxx_flags
603a604,613
> if(DEFINED LOOPMARK)
>   if("${LOOPMARK}" STREQUAL "ON")
>     set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fsave-loopmark")
>   endif()
> endif()
> if(DEFINED ADD_CXX_FLAGS)
>   set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${ADD_CXX_FLAGS}")
> endif()
```

Build and Run miniqmc with perftools-lite-gpu (cont.)

- 2. then add some stuff to build.sh to enable offloading add loopmarks and in general to allow for adding some CXX flags if desired:

```
% diff build.sh.orig build.sh.3.cce.perftools-lite-gpu
6c6,22
< cd build; cmake ..; make; cd ..
---
> echo Using module env
> module -t list 2>&1 | pr -T --columns=2
>
> # cd build; cmake ..; make; cd ..
> cd build
> t0=$SECONDS
> cmake -DCMAKE_SYSTEM_NAME=CrayLinuxEnvironment -DCMAKE_CXX_COMPILER=CC -DQMC_MPI=ON
DENABLE_OFFLOAD=ON -DOFFLOAD_TARGET=amdgcn-amd-amdhsa -DOFFLOAD_ARCH=gfx90a -DLOOPMARK=ON ..
> t1=$SECONDS
> echo "cmake took $(( t1 - t0 )) wallclock seconds."

> t0=$SECONDS
> make VERBOSE=1 -j 1
> t1=$SECONDS
> echo "make took $(( t1 - t0 )) wallclock seconds.
> cd ..

3. then build the binary which contains PAT instrumentation due to module perftools-lite-gpu :s
module load perftools-lite-gpu; build.sh > build.sh.3.out.cce.perftools-lite-gpu 2>&1
```

sbatch.miniqmc.crusher.bash

```
#!/usr/bin/bash
#SBATCH --account=...
#SBATCH --partition=batch
#SBATCH --nodes=1
#SBATCH --time=00:05:00
#SBATCH --output=sbatch.miniqmc.crusher.out
#SBATCH --exclusive
#SBATCH --job-name=miniqmc
#SBATCH --cpu-freq=High
#SBATCH --export=ALL
#SBATCH --verbose

cd $SLURM_SUBMIT_DIR
ulimit -c unlimited
ulimit -s unlimited
echo "info: $0 running in $(pwd) on the following $SLURM_JOB_NUM_NODES nodes:"
echo $SLURM_JOB_NODELIST
echo "with the following modules loaded:"
module list

# export CRAY_ACC_DEBUG=1
export MPICH_GPU_SUPPORT_ENABLED=1
# export OMP_DISPLAY_AFFINITY=TRUE
export OMP_NUM_THREADS=1
export OMP_PLACES=threads
export OMP_PROC_BIND=spread
# export PAT_RT_SUMMARY=0
```

sbatch.miniqmc.crusher.bash (cont.)

```
GBIND="--gpus-per-task=1 --gpu-bind=closest"
CBIND="--cpus-per-task=8 --mem-bind=local --cpu-
bind=mask_cpu:4,400,40000,4000000,400000000,40000000000,4000000000000,400000
000000000"
TASKS="--nodes=1 --ntasks-per-node=8 --ntasks=8"
x=./bin/hello_jobstep
c="OMP_NUM_THREADS=$OMP_NUM_THREADS srun $TASKS $CBIND $GBIND $x | sort"
echo $c
eval $c
echo "###"

x=./bin/miniqmc
a="-g \"2 2 2\" -r 0.999 -n 32"
c="OMP_NUM_THREADS=$OMP_NUM_THREADS srun $TASKS $CBIND $GBIND $x $a"
echo $c
t0=$SECONDS
eval $c
t1=$SECONDS
echo "Running '$x $a' took $(( t1 - t0 )) wallclock seconds."
```

Build and Run miniqmc with perftools-lite-gpu (cont2.)

- cd build # now in miniqmc-OMP_offload/build
- sbatch sbatch.miniqmc.crusher.bash ; … in file sbatch.miniqmc.crusher.out

- The uninstrumented run (built and run without module perftools-lite-gpu) had from miniqmc-internal timers (showing only selected lines here)
- Timer Inclusive_time Exclusive_time Calls Time_per_call
- Total 88.2685 0.0005 1 88.268485593
- Running './bin/miniqmc -g "2 2 2" -r 0.999 -n 32' took 93 wallclock seconds.

- The otherwise same run instrumented with perftools-lite-gpu had corresponding timings:
- Timer Inclusive_time Exclusive_time Calls Time_per_call
- Total 124.0438 0.0007 1 124.043783622
- Running './bin/miniqmc -g "2 2 2" -r 0.999 -n 32' took 128 wallclock seconds.

- This profiling overhead from tracing is larger than what one normally sees from sampling.

Build and Run miniqmc with perftools-lite-gpu (cont3.)

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

CrayPat/X:  Version 22.06.0 Revision 4b5ab6256  05/21/22 02:03:49
Experiment:                  lite  lite-gpu
Number of PEs (MPI ranks):      8
Numbers of PEs per Node:        8
Numbers of Threads per PE:      1
Number of Cores per Socket:    64
Accelerator Model: AMD MI200 Memory: 32.00 GB Frequency: 1.09 GHz  // What? That should be 64GB/GCD, 1.7GHz
                                     // I will need to follow-up HPE-internally on those GB and GHz values ...
Execution start time:  Thu Aug 25 03:46:29 2022
System name and speed:  crusher181  2.820 GHz (nominal)
AMD   Trento               CPU  Family: 25  Model: 48  Stepping:  1
Core Performance Boost:  All 8 PEs have CPB capability
Avg Process Time:       124.55 secs
High Memory:          12,556.8 MiBytes     1,569.6 MiBytes per PE
I/O Read Rate:       23.485416 MiBytes/sec
I/O Write Rate:   2,670.770056 MiBytes/sec
```

Build and Run miniqmc with perftools-lite-gpu (cont4.)

```
Table 1:  Profile by Function Group and Function
  Time% |       Time |     Imb. |   Imb. |        Calls | Group
        |            |     Time |  Time% |              |  Function=[MAX10]  PE=HIDE
 100.0% | 123.212179 |       -- |     -- | 11,768,092.5 | Total
|-------------------------------------------------------------------------
|  58.2% |  71.652569 |       -- |     -- |  7,484,042.0 | OACC
||------------------------------------------------------------------------
||  26.0% |  31.989059 | 0.600431 |   2.1% |    709,641.0 | qmcplusplus::einspline_spo_omp<>::evaluate_vgh.ACC_COPY@li.310
||  17.4% |  21.397965 | 1.802478 |   8.9% |  2,782,845.0 | qmcplusplus::einspline_spo_omp<>::evaluate_v.ACC_COPY@li.162
||  10.5% |  12.959512 | 1.661557 |  13.0% |  2,385,288.0 | qmcplusplus::einspline_spo_omp<>::evaluate_v.ACC_COPY@li.179
||   3.6% |   4.484895 | 0.476412 |  11.0% |    608,256.0 | qmcplusplus::qmc_allocator_traits<>::updateFrom.ACC_COPY@li.158
||========================================================================
|  36.3% |  44.718211 |       -- |     -- |  3,783,338.9 | USER
||------------------------------------------------------------------------
||  19.4% |  23.916608 | 0.190435 |   0.9% |      9,230.0 | qmcplusplus::DelayedUpdate<>::updateInvMat
||   7.3% |   8.951502 | 0.032032 |   0.4% |    495,852.0 | qmcplusplus::DistanceTableAA<>::move
||   2.2% |   2.749999 | 0.265698 |  10.1% |         32.0 | main.LOOP@li.404
||   1.5% |   1.789667 | 0.007626 |   0.5% |     98,304.0 | qmcplusplus::DistanceTableAA<>::evaluate
||   1.0% |   1.282584 | 0.036625 |   3.2% |    397,548.0 | qmcplusplus::TwoBodyJastrow<>::ratio
||========================================================================
|   4.5% |   5.536736 |       -- |     -- |          2.0 | MPI_SYNC
||------------------------------------------------------------------------
||   4.5% |   5.536608 | 5.536604 | 100.0% |          1.0 | MPI_Finalize(sync)
||========================================================================
|   1.0% |   1.239408 |       -- |     -- |    498,928.0 | HIP
|========================================================================
```

Build and Run miniqmc with perftools-lite-gpu (cont5.)

```
Observation:  MPI utilization
    No suggestions were made because all ranks are on one node.

Notes for table 2:
  This table shows functions that have significant exclusive host or accelerator time,
  averaged across ranks, and also data copied in and out, and event counts.
  For further explanation, use:  pat_report -v -O acc_fu ...
Table 2:  Time and Bytes Transferred for Accelerator Regions
  Time% |       Time |    Acc |   Acc |  Acc Copy |  Acc Copy |    Events | Function=[max10]
        |            |  Time% |  Time |        In |       Out |           |  PE=HIDE
        |            |        |       | (MiBytes) | (MiBytes) |           |
 100.0% | 123.212179 | 100.0% | 79.11 |     1,500 |    33,126 | 8,481,928 | Total
|-------------------------------------------------------------------------------
|  26.0% |  31.989059 |     -- |    -- |        -- |        -- |   709,641 |
qmcplusplus::einspline_spo_omp<>::evaluate_vgh.ACC_COPY@li.310
|  19.4% |  23.916608 |     -- |    -- |        -- |        -- |         0 | qmcplusplus::DelayedUpdate<>::updateInvMat
|  17.4% |  21.397965 |     -- |    -- |        -- |        -- | 2,782,845 |
qmcplusplus::einspline_spo_omp<>::evaluate_v.ACC_COPY@li.162
|  10.5% |  12.959512 |  12.4% |  9.82 |        -- |     9,318 | 2,385,288 |
qmcplusplus::einspline_spo_omp<>::evaluate_v.ACC_COPY@li.179
|   7.3% |   8.951502 |     -- |    -- |        -- |        -- |         0 | qmcplusplus::DistanceTableAA<>::move
|   4.5% |   5.536608 |     -- |    -- |        -- |        -- |         0 | MPI_Finalize(sync)
|   3.6% |   4.484895 |   5.2% |  4.13 |        -- |    23,760 |   608,256 |
qmcplusplus::qmc_allocator_traits<>::updateFrom.ACC_COPY@li.158
|   2.2% |   2.749999 |     -- |    -- |        -- |        -- |         0 | main.LOOP@li.404
|   1.5% |   1.789667 |     -- |    -- |        -- |        -- |         0 | qmcplusplus::DistanceTableAA<>::evaluate
|   1.0% |   1.282584 |     -- |    -- |        -- |        -- |         0 | qmcplusplus::TwoBodyJastrow<>::ratio
|===============================================================================
```

Build and Run miniqmc with perftools-lite-gpu (cont6.)

```
Notes for table 4:
  This table show the average time and number of bytes read from each input file,
  taking the average over the number of ranks that read from the file.
  It also shows the number of read operations, and average rates.
  For further explanation, use:  pat_report -v -O read_stats ...
Table 4:  File Input Stats by Filename
 Avg Read | Avg Read |    Read Rate | Number |    Avg |   Bytes/ | File Name=!x/^/(proc|sys)/
 Time per |  MiBytes |  MiBytes/sec |     of |  Reads |     Call |  PE=HIDE
   Reader |per Reader|              | Reader |per Reader|        | /* my bad edit to save a line */
     Rank |    Rank  |              |  Ranks |   Rank  |          |
|-------------------------------------------------------------------------------
| 0.000005 | 0.000675 |   144.927996 |      8 |    3.0 |   236.00 | /opt/rocm-5.1.0/bin/.hipVersion
| 0.000004 | 0.003991 | 1,076.646079 |      1 |    1.0 | 4,185.00 | /tmp/comgr-17c3f1/input/CompileSource
| 0.000004 | 0.000004 |     1.042516 |      8 |    1.0 |     4.00 | /dev/urandom
| 0.000003 | 0.003991 | 1,237.175144 |      1 |    1.0 | 4,185.00 | /tmp/comgr-2ef6a9/input/CompileSource
| 0.000003 | 0.003991 | 1,284.973282 |      1 |    1.0 | 4,185.00 | /tmp/comgr-71e3ad/input/CompileSource
| 0.000003 | 0.003991 | 1,306.424555 |      1 |    1.0 | 4,185.00 | /tmp/comgr-419585/input/CompileSource
| 0.000003 | 0.003991 | 1,319.380831 |      1 |    1.0 | 4,185.00 | /tmp/comgr-7376c1/input/CompileSource
| 0.000003 | 0.000216 |    73.161227 |      8 |    1.0 |   227.00 | /etc/os-release
| 0.000003 | 0.003991 | 1,378.627639 |      1 |    1.0 | 4,185.00 | /tmp/comgr-6d216e/input/CompileSource
| 0.000003 | 0.003991 | 1,407.804943 |      1 |    1.0 | 4,185.00 | /tmp/comgr-fc4ac7/input/CompileSource
| 0.000003 | 0.003991 | 1,438.243969 |      1 |    1.0 | 4,185.00 | /tmp/comgr-a786fd/input/CompileSource
| 0.000003 | 0.008366 | 3,014.641839 |      1 |    1.0 | 8,772.00 | /tmp/comgr-6d216e/output/CompileSource.bc
| 0.000003 | 0.008366 | 3,250.050934 |      1 |    1.0 | 8,772.00 | /tmp/comgr-419585/output/CompileSource.bc
| 0.000003 | 0.008366 | 3,275.501607 |      1 |    1.0 | 8,772.00 | /tmp/comgr-17c3f1/output/CompileSource.bc
| 0.000002 | 0.008366 | 3,553.794012 |      1 |    1.0 | 8,772.00 | /tmp/comgr-2ef6a9/output/CompileSource.bc
|===============================================================================
```

Even without profiling, initial sanity checks …

- Keeping in mind that all runtime checks cost some time and data overhead …
- Do your MPI-ranks and OpenMP-threads have the desired core-affinity?
  - `export OMP_DISPLAY_AFFINITY=TRUE`
- Do transfer data between CPU and GPU? How much? How often? What? If with PrgEnv-cray then
  - Runtime environment variable `CRAY_ACC_DEBUG`:
  - Write accelerator-related activity to stdout for debugging purposes.
  - Valid output values in increasing verbosity in {0,3} where 0 means "none" (default) and 1-3 range from terse to verbose. 2 is reasonable for debugging. But - use for debugging only, because, e.g.,

```
% wc -l sbatch.miniqmc.crusher.out.CRAY_ACC_DEBUG_?
      95 sbatch.miniqmc.crusher.out.CRAY_ACC_DEBUG_0
   64722 sbatch.miniqmc.crusher.out.CRAY_ACC_DEBUG_1
  194006 sbatch.miniqmc.crusher.out.CRAY_ACC_DEBUG_2
 1871974 sbatch.miniqmc.crusher.out.CRAY_ACC_DEBUG_3
```

- Even at CRAY_ACC_DEBUG=1 you already get, e.g.,
  - ACC: Transfer 1 items (to acc 98304000 bytes, to host 0 bytes) from einspline_spo_omp.cpp:127
  - ACC: Transfer 4 items (to acc 4 bytes, to host 0 bytes) from einspline_spo_omp.cpp:129
  - ACC: Execute kernel __omp_offloading_70_d8d73b92__ZN11qmcplusplus17einspline_spo_ompIdE3setEiiiiib_l129 from einspline_spo_omp.cpp:129

How about a build and run of miniqmc with perftools (CrayPAT) classic?

```
cd miniqmc-OMP_offload
mv build build.perftools-lite-gpu.your_favorite_tag
mkdir build
module swap perftools-lite-gpu perftools
cp build.sh.3.cce.perftools-lite-gpu build.sh.5.cce.perftools
build.sh.5.cce.perftools > build.sh.5.cce.perftools.out 2>&1
# 339 seconds later (due to make -j 1) ... the build is done
cd bin ; patbuild miniqmc
cd .. # now in miniqmc-OMP_offload/build
sbatch sbatch.miniqmc.crusher.bash
# since miniqmc+pat+23080-6078120s/build-options.apa wanted to
# trace just ‘-g mpi’ I changed that to ‘-g hip,io,mpi,omp’
pat_build -O miniqmc+pat+23080-6078120s/build-options.apa
# since miniqmc+apa shows up in build/ (not in build/bin/ )
# change sbatch.miniqmc.crusher.bash accordingly and submit
# and this is where I run out of time, since my job is
# waiting in the queue ...
```

What if pat_report …

- Says “No APA data file was generated because no samples occurred in USER functions.” …
  - This usually means either that the program spent so little time in user-defined functions that no samples were taken there, or that CrayPat failed to identify the user-defined functions (and instead showed them under the group ETC).
- … Or shows a lot of time spent under ETC in Automatic Profiling Analysis (APA).
  - While APA instruments some libs, such as MPI, it does not instrument some others, such as libsci. A lot of time spent in ETC suggests that PAT was not able to identify where those calls under ETC originated. For sampling, the classification of functions as USER versus ETC is based on whether the user running pat_report had write access to the directory that contained the source file of the call.
- To remedy this:
  - Move aside or delete the *.ap2 files created by pat_report from the *.xf files that were generated by running the PAT-instrumented binary
  - export PAT_REPORT_PRUNE_NON_USER=0 # search for PAT_RT_ in `man intro_craypat`
  - repeat pat_report invocation
  - … because that pruning is done when the *.ap2 files are generated from the *.xf files; if pat_report finds *.ap2 files, it will not attempt to regenerate those from *.xf files

If you want to limit or completely turn off PAT's pruning

- Search `man pat_report` for _PRUNE
- PAT_REPORT_PRUNE_USER
  - based on ownership of compilation dir of function def
    to turn off: export PAT_REPORT_PRUNE_NON_USER=0
- PAT_REPORT_PRUNE_NAME
  - based on function name
    to turn off: export PAT_REPORT_PRUNE_NAME=""
- PAT_REPORT_PRUNE_NAME_FILE
  - based on name of file that contains function
    to turn off: export PAT_REPORT_PRUNE_NAME_FILE=""
- PAT_REPORT_PRUNE_SRC
  - based on path of file that contains function
    to turn off: export PAT_REPORT_PRUNE_SRC=""

Misc. PAT Topics: More MPI stats

- To get more MPI stats out of CrayPAT, if you did
  - pat_build -w -g mpi a.out ; srun [srun-opts] a.out+pat
- try
  - pat_report -i a.out+pat -O opt{,opt,...} experiment_data_dir > pat_report.out.mpi
- with {opt} in
  - mpi_callers
    Show MPI sent- and collective-message statistics
  - mpi_sm_callers
    Show MPI sent-message statistics
  - mpi_coll_callers
    Show MPI collective-message statistics
  - mpi_dest_bytes
    Show MPI bin statistics as total bytes
  - mpi_dest_counts
    Show MPI bin statistics as counts of messages

Misc. PAT Topics: More OpenMP stats

- To get more OpenMP info from CrayPAT, try (after pat_build –w –g omp …)
- pat_report -i a.out+pat -O opt{,opt,…} experiment_data_dir > pat_report.out.mpi
- with {opt} in
  - profile_pe.th
    Show the imbalance over the set of all threads in the program.
  - profile_pe_th
    Show the imbalance over PEs of maximum thread times.
  - profile_th_pe
    For each thread, show the imbalance over PEs.
  - thread_times
    For each thread number, show the average of all PE times
    and the PEs with the minimum, maximum, and median times.

Misc. PAT Topics: Avoid “contamination” of profiles by initialization phase

- Minor code changes are necessary for that. Whenever you make PAT calls:

```
#if defined(CRAYPAT)
#include pat_api.h    // C/C++
include  pat_apif.h    ! F90
include  pat_apif77.h  ! F77
#endif
```

- Method 1:

```
// At the beginning of main
#if defined(CRAYPAT)
int PAT_record(PAT_STATE_OFF)  // returns PAT_API_OK(1) or PAT_API_FAIL(0)
#endif
// initialization - not to be PAT-recorded
#if defined(CRAYPAT)
int PAT_record(PAT_STATE_ON)
#endif
// code of interest
```

- Method 2:

```
Start your job with export PAT_RT_RECORD=PAT_STATE_OFF
// in the source code, just above the region of interest
#if defined(CRAYPAT)
int PAT_record(PAT_STATE_ON)
#endif
```

Misc. PAT Topics: What if you want pat_report only from selected MPI-ranks?

- `pat_report -sfilter_input='condition'`
  - The 'condition' could be an expression involving 'pe' such as 'pe<1024' or 'pe%2==0'

Misc. PAT Topics: If you want to build with hipcc instead of using {cc,CC,ftn}?

- Load all necessary software environment modules including perftools-base and perftools.
- Add the following flags generated by pat_opts to the appropriate places in your build procedure before building your uninstrumented binary a.out followed by `pat_build a.out` to get the PAT-instrumented binary a.out+pat:
- % pat_opts include hipcc
  -I/opt/cray/pe/perftools/22.06.0/include
  % pat_opts pre_compile hipcc
  -DCRAYPAT -g –gpubnames
  % pat_opts post_compile hipcc
  -fno-omit-frame-pointer -mno-omit-leaf-frame-pointer -fno-optimize-sibling-calls
  % pat_opts pre_link hipcc
  -B/opt/cray/pe/perftools/22.06.0/libexec64 -L/opt/cray/pe/perftools/22.06.0/lib64 \
  -L/opt/rocm-5.1.0/lib -lroctracer64
  % pat_opts post_link hipcc
  # empty here, but not guaranteed always to be empty
- And finally: make clean ; make ; pat_build [your PAT options] a.out ; srun [your srun opts] a.out+pat

Acknowledgements

- Useful input from Steve Abbott, Tanner Firl, Kostas Makrides, Luke Roskop and Trey White (all HPE) is gratefully acknowledged, as is
- the opportunity provided to speak here, by Dossay Oryspayev (BNL)
- and your patience having made it to this point.☺

The End

- Questions, Comments, Concerns, Corrections?
  - PAT is vast – nobody knows everything about it and forgetting details, even important ones, is normal.
  - Don’t be shy to ask! We may not know either, but we will find out.☺
  - And if you found or suspect a bug in any PAT utility, please, report it.
  - Discuss now
  - Email me
  - Contact any CORAL-2 CoE member
  - Take advantage of Crusher Office Hours: https://www.olcf.ornl.gov/crusher-office-hours
  - Email help@olcf.ornl.gov

THANK YOU

Marcus Wagner
marcus.wagner@hpe.com