Hewlett Packard
Enterprise

HPE Perftools Application Profiling (PAT)

Marcus Wagner
CORAL-2 Centers of Excellence

March/01/2024

Outline

- This presentation builds on presentations by Stephen Abbott:
  https://www.openmp.org/wp-content/uploads/2022-04-29-ECP-OMP-Telecon-HPE-Compiler.pdf
  Trey White: https://www.olcf.ornl.gov/wp-content/uploads/2-17-22_application_profiling.pdf
  Marcus Wagner: https://www.openmp.org/wp-content/uploads/ecp_sollve_openmp_monthly.offload_perf_ana_craypat.marcus.hpe_.26aug2022.v2.pdf

- Documentation Resources

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

- Build, run and profile miniapp examples on Frontier compute nodes (Bard Peak, AMD Trento CPUs and AMD MI250X GPUs) and show PAT performance analysis using different PAT components.

- Acknowledgements
- The End

Outline (cont.)

- Disclaimer:
  - This is not a comprehensive reference for, or introduction to anything; instead
  - It is meant to be a show-and-tell how to get started with PAT on Frontier
  - For more details, please, refer to the documentation or, as always – OLCF Office Hours, email, call, …

- Claim:
  - I claim ownership of mistakes in this presentation.
  - If you find or suspect a problem, please, let me know.

Documentation Resources

- https://docs.olcf.ornl.gov/systems/frontier_user_guide.html
- https://support.hpe.com # there, search for (including “” if shown here) …
  - “HPE Performance Analysis Tools User Guide (23.12) S-8014”
  - HPE Cray Fortran Reference Manual (17.0) S-3901
  - HPE Cray Clang C and C++ Quick Reference (17.0) S-2179
  - # where: 1st (number) = software version, 2nd S-number = const. document ID independent of SW version
  - Cray Performance Tools manpages https://cpe.ext.hpe.com/docs/performance-tools/index.html#perftools
- https://clang.llvm.org/docs/ClangCommandLineReference.html
- man pages relevant in this context, for the SW env module loaded, e.g., cce/17.0.0
  - 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
- AMD online docs, e.g.,
  - ROCm : https://rocm.docs.amd.com/en/latest/what-is-rocm.html
  - MI250X : https://rocm.docs.amd.com/en/latest/conceptual/gpu-arch/mi250.html
  - https://www.amd.com/content/dam/amd/en/documents/instinct-business-docs/white-papers/amd-cdna2-white-paper.pdf
  - https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/instruction-set-architectures/instinct-mi200-cdna2-instruction-set-architecture.pdf

AM I WASTING MY TIME BY PROFILING A CODE I KNOW?

- How much should you have to bother with detailed profiling for performance optimization if you have experience, domain expertise and good intuition?

- These things will typically help you if they pertain to the currently used (CPU * GPU * Network * Compiler * MPI * Test Case), but …

- Good Judgement comes from Experience, and …
  Experience comes from Bad Judgement.☺

- Case in point…

An “Obvious” Perf. Hotspot and Fix: (crayftn loopmark listing: -h list=a)
Is line 2515 below compute “/” bound? → 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 in this “/” line
                                             elem(ie)%state%Qdp(:,:,k,q,n0_qdp) / dp(:,:,k) ! 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 for array index and offset arithmetic.

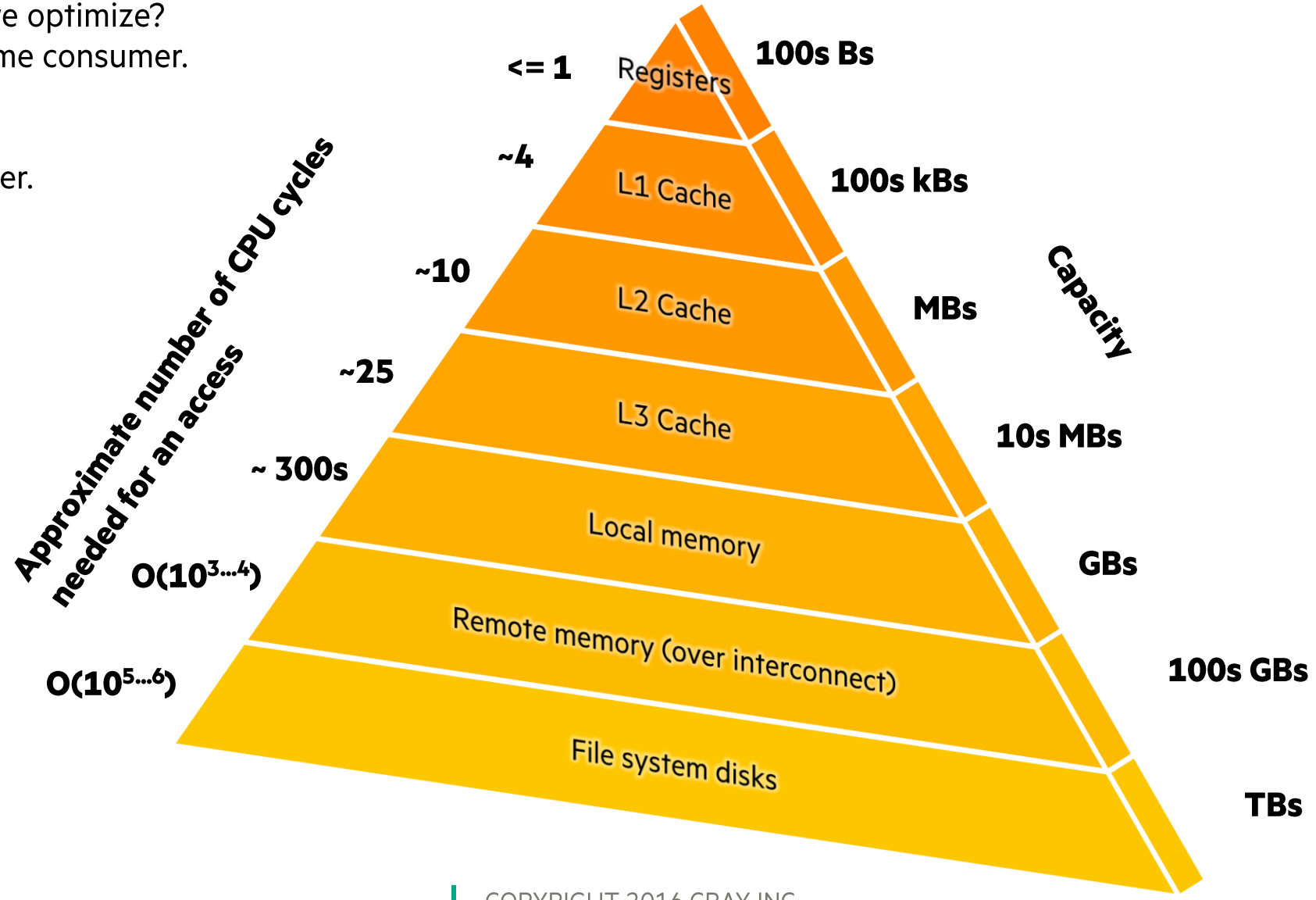
This “/” statement is NOT floating-point bound. Integer Ops and Memory BW dominate time.

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 large 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 optimize**
  - Concentrate your efforts on the thing that slows your code down.
  - This will change as you optimize.
  - So keep on profiling.

A HISTORIC CPU SLIDE: THE IMPORTANCE OF DATA LOCALITY

Q1: What should we optimize?
A1: The biggest time consumer.
…
Q2: Which is?
A2: Ask your profiler.



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 (lustre,gpfs) 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, el. power and specific packages without having to manually instrument your original source code, but you can (and may want to) do some instrumentation, e.g., to exclude the initialization phase of your run from taking data, since that may have different performance characteristics than the steady-state run phase of your code.

- PAT provides detailed customizable analysis in plain text format and a GUI that can run remotely on the compute host as well as locally in an app on your Windows or macOS laptop.

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.
- Not covered here, due to Prep- and Presentation-Time constraints:
  - perftools-preload - Runtime instrumentation version of perftools, eliminates pat_build instrumentation step.
  - pat_run - Launches a program to collect performance information.

During the hackathon, perftools developers are available to help with all the tools, especially pat_run.

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/23.12.0  # or your favorite version
module load perftools               # will match above version
cd $CRAYPAT_ROOT/share/desktop_installers
```

  - download Apprentice2Installer-23.12.0-2.exe to laptop # windows
  - download Apprentice2Installer-23.12.0-2.dmg # 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.
    Not covered here, due to Prep- and Presentation-Time constraints.
    However, there is also a Reveal desktop_installer for apple, RevealInstaller-23.12.0-2.dmg

CAVEATS

- Tightly coupled with specific versions of ROCm
- Check with `module show perftools-base/${VERSION} 2>&1 | grep -i rocm`
- Tip: To minimize problems with SW env module compatibility between (cce, rocm, cray-mpich, perftools, …), start with loading a desired cpe/* module, e.g., cpe/23.12 (=cpe/YY.MM) whose modules are expected to be mutually compatible; then, add/change modules as desired, e.g., a different ROCm module, and then check compatibility with `module show NAME 2>&1 | grep -iE “rocm|cce”`

- No profiling inside AMD GPU kernels*
- Currently limited support for AMD GPU performance counters*
- Timeline visualization for full traces temporarily broken*

* but we are working on it

PAT Overview

- PAT assists the user with application performance analysis and optimization
  - Provides concrete suggestions instead of just reporting data.
  - Works on user codes at realistic core counts with thousands of processes/threads integrate into large codes with millions of lines of code
  - (To optimize VPIC for the BlueWaters/NCSA acceptance, I ran 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:
    1. The 1-hour time limit on interactive sessions was too short to read in all *.xf profile data from that run. (Keep in mind - you don't want those files to fill up your $HOME)
       However, pat_report can be run from within a batch job. ☺
    2. 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 (unless PAT_RT_REPORT_METHOD=0), the other still allocated nodes remain idle.

PAT Overview (cont.)

- 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.2)

- 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 (but with the perftools module loaded) and
  let pat_build do the instrumentation of your optimized code.
- pat_build creates new stand-alone instrumented program while preserving original binary.

- Note 1: For CCE Fortran < 17.0.0, 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".
- Note 2: When you build an app with a perftools module loaded, i.e., perftools or perftools-lite-*
  (but not with perftools-base/* alone, where the following is not the case),
  PAT may keep some temp files in $HOME/.craypat/
  Therefore, you may want to periodically clean out old files under $HOME/.craypat/
  if you have built binaries with module perftools or perftools-lite-* loaded.

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.

BEFORE BUILDING AND RUNNING, ESTABLISH THE DESIRED SW MODULE ENVIRONMENT

```
% cat setup_env.cpe_2312
module load cpe/23.12   # Dec/2023 version
module load rocm/5.7.1  # instead rocm/5.5.1 loaded by cpe/23.12
module load craype-accel-amd-gfx90a
module load perftools   # matches perftools-base/23.12.0
module load cmake/3.23.2  # HIP support in cmake >= 3.21, dflt cmake=3.20.4
module unload darshan-runtime  # don't mix with perftools
```

```
% source ./setup_env.cpe_2312
```

```
% export LD_LIBRARY_PATH=${CRAY_LD_LIBRARY_PATH}:${LD_LIBRARY_PATH}
```

```
# showing only relevant parts of module checking here
```

```
% module show perftools-base/23.12.0 2>&1 | grep -i "supp.*rocm"
   * Add support for ROCm 5.7.0
```

```
% module show cray-libsci/23.12.5 2>&1 | grep -iE "rocm|cce"
   * AMD ROCm 5.0.0 or later
   * CCE 17.x (SLES)
```

```
% module show cray-mpich/8.1.28 2>&1 | grep -iE "rocm|cce"
   * AMD ROCM 5.0 or later
   * CCE 17.0 or later
```

- Note:
  - Loading modules does not change LD_LIBRARY_PATH
  - Remember to use same SW module env and LD_LIBRARY_PATH for running as for building
  - For production jobs built and run without module perftools, i.e., not profiling runs, may want to keep module darshan-runtime loaded to get I/O logs /lustre/orion/darshan/<frontier|crusher>/YYYY/<M|MM>/<D|DD>/*.darshan

WHEN BUILDING WITH HIPCC RATHER THAN CC OF PRGENV-{CRAY,AMD}

```
# establish SW module env as before
# add "CRAY_MPICH_DIR..." for HPE Cray MPI
# add  PE_MPICH_GTL_DIR_amd_gfx90a  and  PE_MPICH_GTL_LIBS_amd_gfx90a
   for GPU-aware MPI with MPICH_GPU_SUPPORT_ENABLED=1
# add "XPMEM... -lxpmem" for zero-copy on-node MPI with CPU buffers
# add “$(pat_opts ...)” when instrumenting with perftools but not using ftn/cc/CC compiler drivers

export CXX=hipcc   # not using CC
export CXXFLAGS="$(pat_opts include hipcc) $(pat_opts pre_compile hipcc) \
                 -g -fopenmp -O3 -std=c++17 --offload-arch=gfx90a -Wall \
                 -I${CRAY_MPICH_DIR}/include $(pat_opts post_compile hipcc)"
export LD=hipcc   # not using CC
export LDFLAGS="$(pat_opts pre_link hipcc)  ${CXXFLAGS} -L${CRAY_MPICH_DIR}/lib \
        ${PE_MPICH_GTL_DIR_amd_gfx90a}  ${CRAY_XPMEM_POST_LINK_OPTS}"
export LIBS="-lmpi ${PE_MPICH_GTL_LIBS_amd_gfx90a} -lxpmem $(pat_opts post_link hipcc)"
make
```

WHEN BUILDING WITH HIPCC, PERFTOOLS AND CMAKE

- # establish SW module env as before
- # Turn on Hip
  -DENABLE_HIP=ON
  -DWITH_HIP=TRUE
- # Target MI250X
  -DWITH-GPU-ARCH=gfx90a
  -DGPU_TARGETS=gfx90a
  -DHIP_ARCH=gfx90a
  -DCMAKE_HIP_ARCHITECTURES="gfx90a“
- # Use hipcc
  -DCMAKE_CXX_COMPILER=“hipcc”
  -DMPI_CXX_COMPILER="hipcc -I${MPICH_DIR}/include“
- # Add Perftools (and other) arguments
  -DCMAKE_CXX_FLAGS="$(pat_opts include hipcc) $(pat_opts pre_compile hipcc) \
  --offload-arch=gfx90a -munsafe-fp-atomics $(pat_opts post_compile hipcc)“
  -DCMAKE_EXE_LINKER_FLAGS="$(pat_opts pre_link hipcc) $(pat_opts post_link hipcc)"

WHEN BUILDING WITH HIPCC, PERFTOOLS AND CMAKE (CONT.)

- If you get errors such as …

```
readelf: Warning: Unrecognized form: 0x22
readelf: Warning: Unrecognized form: 0x23
```

then that indicates an issue with readelf from binutils-2.38 and earlier.
This is fixed in binutils-2.39.

- As a temporary workaround (if using binutils < 2.39)
add to the above CMAKE_CXX_FLAGS also -gdwarf-4 as in
-DCMAKE_CXX_FLAGS="$(pat_opts include hipcc) $(pat_opts pre_compile hipcc) \
--offload-arch=gfx90a -gdwarf-4 -munsafe-fp-atomics $(pat_opts post_compile hipcc)“
to use an older debug format.

- If cmake gives you the error

```
Cannot determine location of 'ld.lld' in PATH
```

then `export PATH="${PATH}:${ROCM_PATH}/llvm/bin“`
(see https://docs.olcf.ornl.gov/systems/crusher_quick_start_guide.html#olcfdev-513-error-with-perftools-lite-gpu (I did some of my prep work on crusher))

USING PAT_BUILD TO GENERATE AN INSTRUMENTED BINARY

```
# With the desired SW module env set up on the previous slide
# while still having module perftools loaded now, just as it was for compiling and linking …
# `pat_build a.out` generates an instrumented binary executable named
# a.out+pat to be run to generate profiling data as a.out would be;

# Build binary instrumented for sampling,
# where the default PAT_RT_EXPERIMENT is samp_pc_time;
# see `man cray_pat` for PAT_RT_EXPERIMENT;
pat_build -S a.out

# or, build binary instrumented for tracing (-w),
# see `man pat_build` for the many supported trace groups:
pat_build -w -g mpi,omp,io,hip  a.out
```

USING PAT_BUILD TO GENERATE AN INSTRUMENTED BINARY (CONT.)

```
# ... Or by default, build binary instrumented for
# "automatic profiling analysis" (apa) implying 2 build+run cycles:
make clean ; make ; pat_build a.out  # generates a.out+pat
export MPICH_ENV_DISPLAY=1
export MPICH_GPU_SUPPORT_ENABLED=1
export MPICH_OFI_NIC_POLICY=GPU
export MPICH_VERSION_DISPLAY=1
export OMP_PLACES=threads
export OMP_PROC_BIND=spread
CORES_PER_RANK=7  # if the Frontier default of "#SBATCH -S 8" is used;
# you have 8 GCDs/node and want 8 MPI-ranks/node && 1 MPI-rank/GCD;
# want/node: 64 cores = (8 MPI) * $CORES_PER_RANK + $SPECIALIZED_CORES;
# want the 8 MPI-ranks/node spaced 8 cores apart to optimally map to GCDs;
export OMP_NUM_THREADS=7  # <= CORES_PER_RANK
# run generates a.out+pat+<PID>-<node>s experiment data directory.
srun -N $NODES -n $RANKS -c $CORES_PER_RANK \
        --gpus-per-task=1 --gpu-bind=closest a.out+pat [a.out args]
# see compute node diagram at https://docs.olcf.ornl.gov/systems/frontier_user_guide.html
```

USING PAT_BUILD TO GENERATE AN INSTRUMENTED BINARY (CONT2.)

```
# run pat_report to make report from experiment data and produce a text file,
pat_report  -i a.out+pat   a.out+pa+<PID>-<node>s \
              > pat_report.out.dflt.1   # capture default sampling report
# the above pat_report generated 1 or more *.ap2 files from *.xt files,
# i.e., a.out+pat+<PID>-<node>s/ap2-files/*.ap2;
# the *.xt files can now be deleted, but not the *.ap2 files;

# build the 2nd instrumented binary based pat_build's recommendations;
# this is the "automatic" part of "automatic profiling analysis“, using the plain text file build-options.apa
# generated in the previous run, which contains instrumentation instructions for PAT to be used now:
# this 2nd pat_build invocation generates a.out+apa
pat_build -O <my_program>+pat+<PID>-<node>s/build-options.apa

srun -N $NODES -n $RANKS -c $CORES_PER_RANK   \
       --gpus-per-task=1 --gpu-bind=closest a.out+apa [a.out args]
# run generates <my_program>+apa+<PID2>-<node>t profiling data dir

pat_report -i a.out+apa <my_program>+apa+<PID2>-<node>t   \
          > pat_report.out.dflt.2  # capture default tracing report
                  # from automatic instrumentation
```

USING PAT_BUILD TO GENERATE AN INSTRUMENTED BINARY (CONT3.)

- If, during `srun … a.out+pat …`, you get a warning like
  pat[WARNING][0]: 19543 CID to EID records were dropped.
  Try increasing the value of PAT_RT_ACC_CID_TO_EID_BUFFER_SIZE
  then … Try increasing that☺, e.g.,
  export PAT_RT_ACC_CID_TO_EID_BUFFER_SIZE=128 MB # default = 1 MB

- A description of the many PAT_RT_* environment parameters can be found in `man cray_pat` under "Runtime Environment Variables Summary"

- If you want other reports from pat_report than the default one, such as
  pat_report -O acc_time -s show_ca=fu,so,li experiment-data-dir > pat_report.out.show_ca.fu.so.li
  (for accelerator kernels also showing callers by function, source, line);
- you can get the pre-defined options pat_report supports listed with a short description from `pat_report -O -h`

DEFAULT REPORT

Table 1: Profile by Function Group and Function

```
  Time% |      Time |      Imb. |   Imb. |         Calls | Group
        |           |      Time | Time% |               |  Function
        |           |           |       |               |   PE=HIDE
        |           |           |       |               |    Thread=HIDE
 100.0% | 5.136167 |        -- |    -- | 1,327,756.0 | Total
|-------------------------------------------------------------------------
|  64.7% | 3.321289 |        -- |    -- |   727,589.0 | HIP
||------------------------------------------------------------------------
||  26.2% | 1.346300 | 0.284986 | 20.0% |    33,000.0 | hipStreamSynchronize
||  11.6% | 0.595530 | 0.000128 |  0.0% |       253.0 | hipMemset
||   7.9% | 0.406597 | 0.006219 |  1.7% |        20.0 | hipStreamDestroy
||   5.6% | 0.286828 | 0.018657 |  7.0% |    66,000.0 | hipKernel.gpuRun3x1<>
||   2.5% | 0.129979 | 0.003099 |  2.7% |   198,232.0 | hipLaunchKernel
||   2.4% | 0.122329 | 0.001136 |  1.1% |        20.0 | hipStreamCreate
||   2.3% | 0.118819 | 0.003993 |  3.7% |       110.0 | hipKernel.init
...
```

// Host times spent on Hip API calls,
// not kernel execution times on accelerator.

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 (Max – Avg)
- 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)]`

DEFAULT REPORT (CONT.1)

Table 6: MPI Message Stats by Caller

MPI Msg Bytes%	MPI Msg Bytes	MPI Msg Count	MsgSz <16 Count	256<= MsgSz <4KiB Count	64KiB<= MsgSz <1MiB Count	Function Caller PE=[mmm] Thread=HIDE
100.0%	7,550,930,468.0	77,714.0	11,114.0	33,300.0	33,300.0	Total
100.0%	7,550,930,400.0	77,700.0	11,100.0	33,300.0	33,300.0	MPI_Isend
99.1%	7,482,904,000.0	77,000.0	11,000.0	33,000.0	33,000.0	Faces::share 3 main
99.1%	7,482,904,000.0	77,000.0	11,000.0	33,000.0	33,000.0	pe.0
99.1%	7,482,904,000.0	77,000.0	11,000.0	33,000.0	33,000.0	pe.4
99.1%	7,482,904,000.0	77,000.0	11,000.0	33,000.0	33,000.0	pe.7

DEFAULT REPORT (CONT.2)

Table 7: File Input Stats by Filename

Avg Read Time per Reader Rank	Avg Read MiBytes per Reader Rank	Read Rate MiBytes/sec	Number of Reader Ranks	Avg Reads per Reader Rank	Bytes/ Call	File Name=!x/^/(proc|sys)/ PE=HIDE
0.000020	0.000097	4.926054	1	102.0	1.00	stdin
0.000008	0.004499	556.516441	1	1.0	4,718.00	/tmp/comgr-538d7e/input/CompileSource
0.000008	0.004499	572.156081	1	1.0	4,718.00	/tmp/comgr-b039d7/input/CompileSource
0.000007	0.000690	92.833138	8	3.0	241.00	/opt/rocm-5.3.0/bin/.hipVersion
0.000006	0.004499	729.125818	1	1.0	4,718.00	/tmp/comgr-9788a0/input/CompileSource
0.000006	0.008984	1,487.106780	1	1.0	9,420.00	/tmp/comgr-9788a0/output/CompileSource.bc
0.000006	0.004499	777.104564	1	1.0	4,718.00	/tmp/comgr-ebc47e/input/CompileSource
0.000006	0.004499	794.953255	1	1.0	4,718.00	/tmp/comgr-1ff8d6/input/CompileSource
0.000005	0.004499	831.688618	1	1.0	4,718.00	/tmp/comgr-b1721a/input/CompileSource
0.000005	0.004499	847.351304	1	1.0	4,718.00	/tmp/comgr-5c35ff/input/CompileSource
0.000005	0.008984	1,782.816444	1	1.0	9,420.00	/tmp/comgr-b1721a/output/CompileSource.bc
0.000005	0.008984	1,818.913153	1	1.0	9,420.00	/tmp/comgr-b039d7/output/CompileSource.bc
0.000005	0.004499	922.204432	1	1.0	4,718.00	/tmp/comgr-dda55e/input/CompileSource
0.000005	0.008984	1,928.641490	1	1.0	9,420.00	/tmp/comgr-538d7e/output/CompileSource.bc
0.000004	0.008984	1,997.245901	1	1.0	9,420.00	/tmp/comgr-ebc47e/output/CompileSource.bc

DEFAULT REPORT (CONT.3)

Table 9: Time and Bytes Transferred for Accelerator Regions

Time%	Time	Acc Time%	Acc Time	Acc Copy Out (MiBytes)	Events	Calltree Accelerator ID PE=HIDE Thread=HIDE
100.0%	5.136167	100.0%	3.77	36.00	331,016	Total
100.0%	5.135812	100.0%	3.77	36.00	331,016	main
61.3%	3.147641	99.8%	3.77	--	330,000	Faces::share
26.2%	1.346300	--	--	--	33,000	hipStreamSynchronize
						acc.0
15.9%	0.815399	--	--	--	0	MPI_Waitall
						acc.0
13.8%	0.707482	99.8%	3.77	--	297,000	gpuFor<>

DEFAULT REPORT (CONT.5)

```
Notes for table 9:                                          // this is from a different app
  This table shows energy and power usage for the nodes with the
    maximum, mean, and minimum usage, as well as the sum of usage over
    all nodes.
    Energy and power for accelerators is also shown, if available.
  For further explanation, see the "General table notes" below, or
    use:  pat_report -v -O program_energy ...
Table 9:  Program Energy and Power Usage from Cray PM
Node Id / PE=HIDE / Thread=HIDE
=============================================================
  Total
-------------------------------------------------------------
  PM Energy Node  3,158 W   194,060 J
  PM Energy Cpu     316 W    19,437 J
  PM Energy Memory  319 W    19,589 J
  PM Energy Acc0    530 W    32,565 J
  PM Energy Acc1    534 W    32,825 J
  PM Energy Acc2    506 W    31,073 J
  PM Energy Acc3    520 W    31,981 J
  Process Time             61.452468 secs
```

DEFAULT REPORT (CONT.6)

```
Notes for table 10:
  This table shows values shown for HiMem calculated from information
    in the /proc/self/numa_maps files captured near the end of the
    program. It is the total size of all pages, including huge pages,
    that were actually mapped into physical memory from both private
    and shared memory segments.
  For further explanation, see the "General table notes" below, or
    use:  pat_report -v -O himem ...

Table 10:  Memory High Water Mark by Numa Node
   Process |      HiMem |      HiMem |      HiMem |      HiMem | Numanode
     HiMem | Numa Node | Numa Node | Numa Node | Numa Node |   PE=HIDE
 (MiBytes) |         0 |         1 |         2 |         3 |
           | (MiBytes) | (MiBytes) | (MiBytes) | (MiBytes) |
|-----------------------------------------------------------------------
|     514.2 |     427.7 |      29.6 |      28.9 |      28.0 | numanode.0
|     521.5 |      29.1 |     435.6 |      28.8 |      27.9 | numanode.1
|     521.1 |      29.1 |      29.6 |     434.5 |      27.9 | numanode.2
|     514.5 |      29.0 |      29.5 |      28.9 |     427.1 | numanode.3
|=======================================================================
```

WHAT ELSE CAN PAT_REPORT TELL YOU?

A lot.

```
% pat_report -O -h
pat_report: Help for -O option:
D1_D2_observation         D1 + D2 cache utilization
D1_D2_util                Functions with low D1+D2 cache hit ratio
D1_observation            D1 cache utilization
D1_util                   Functions with low D1 cache hit ratio
...
samp_profile              Sample Profile by Function
samp_profile+hwpc         Sample Profile by Function with Counters
samp_profile+src          Sample Profile by Group, Function, and Line
samp_profile+src+hwpc     Sample Profile by Group, Function, and Line with Counters
...
% pat_report -O -h | wc -l
158

% pat_report -O OPTION -h  # more details on option OPTION

% man pat_report
```

TEMPLATES – CAN MAKE PROFILES MORE OBSCURE

- C++ codes often launch kernels from template functions that take lambda arguments
- "Real" kernel code is in the user-provided lambdas
- Think "portability layers", Kokkos, Raja, Alpaka, Yakl, *etc.*
- Templates get inlined into user code
- Profiles show line numbers somewhere inside portability layers
  instead of line numbers in user code where lambdas appear

- WORKAROUND - Turn off compiler inlining of top layer of portability-layer templates:

```
template <typename F>
#ifdef CRAYPAT
  __attribute__((noinline))
#endif
void gpuFor(const std::initializer_list<int> il0, F f,
            const hipStream_t stream = 0)
{ ...
```

  Minimal impact on runtime
  - Kernel launches are much more expensive than host function calls
- But changes are in portability layer, not user code

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

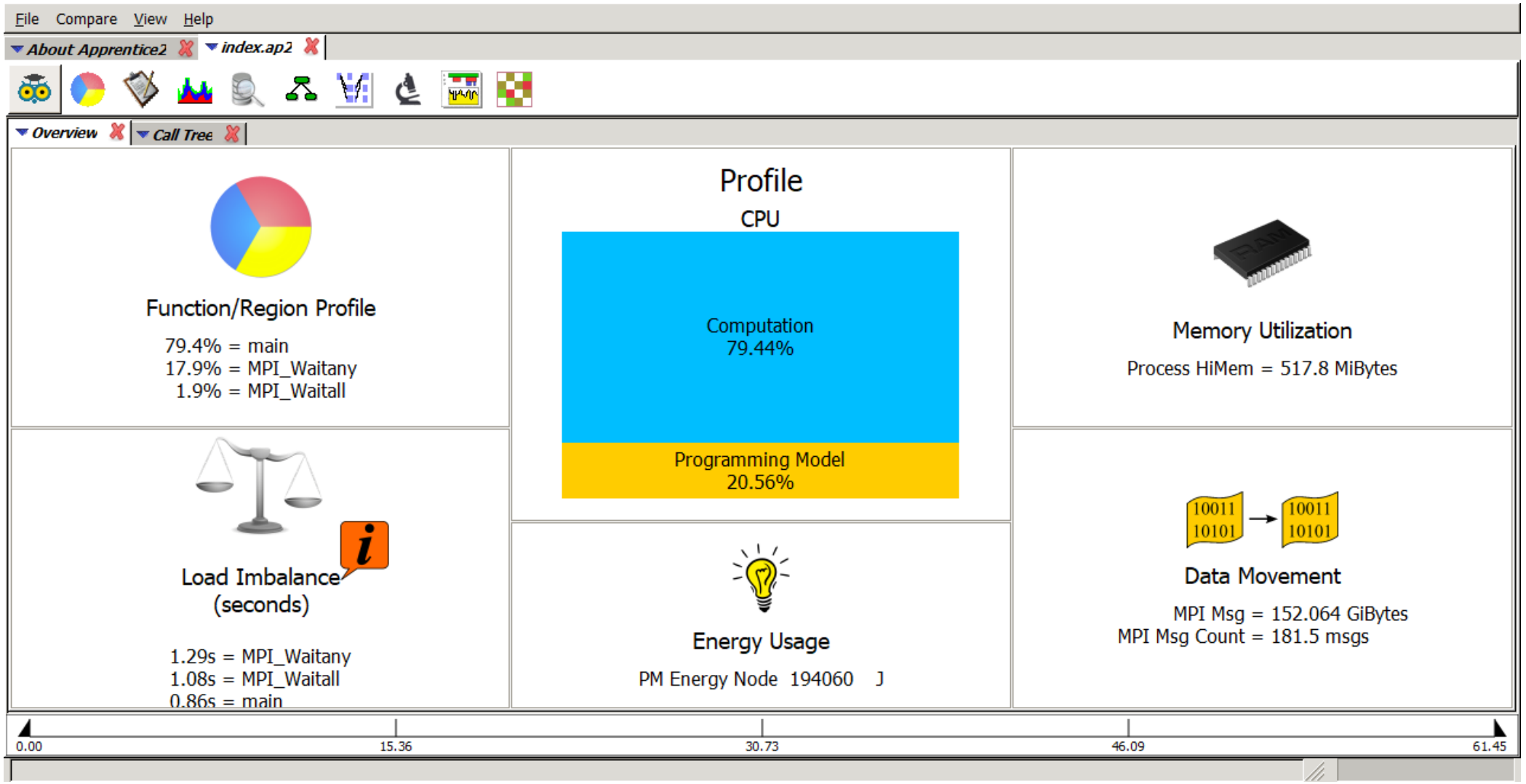
Apprentice2 – Graphic Representation of Performance Data (cont.)

- module load perftools-base
- Must run pat_report to generate the *.ap2 file(s) from the *.xf file(s), after that, can remove the *.xf file(s)

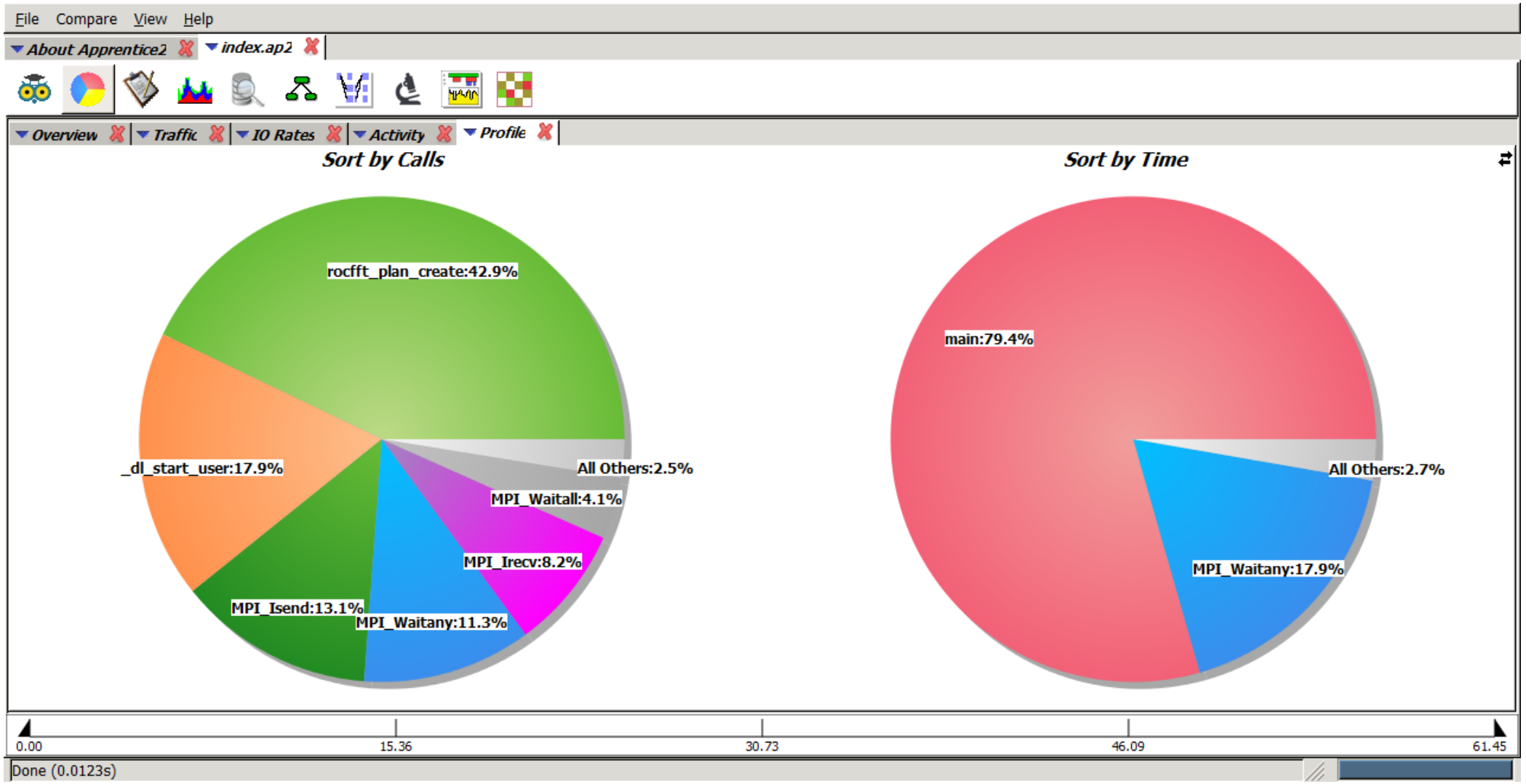
- From the linux command prompt:
  - `app2 <pat-exp-data-dir>/index.ap2` or just
    `app2 index.ap2` # opens an X-window

- On your laptop (win/mac):
  - Can run app2 on frontier and display the GUI on your laptop, but that's slow.
  - MUCH faster to run app2 on your laptop and
    File -> Open Remote... -> enter user@host -> enter PIN+OTP -> cp+paste full path to PAT-expt-dir -> click index.ap2
  - Alternatively, scp PAT-expt-dir to laptop (pfew!) and then
    File -> Open -> ...
    but that's not much faster and gave me errors for some app2 functions.

Apprentice2 overview comes up after having clicked on index.ap2



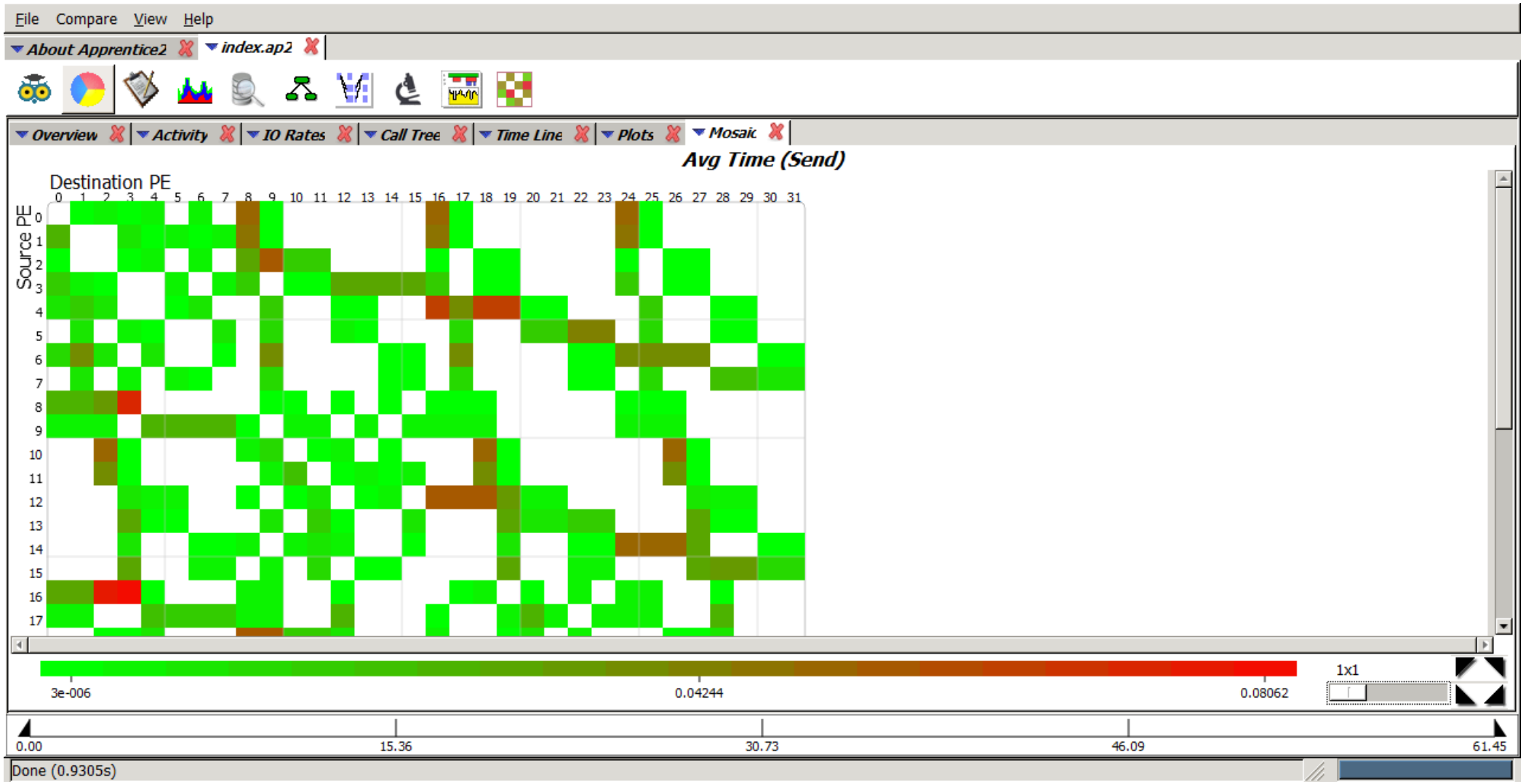
FUNCTION / REGION PROFILE



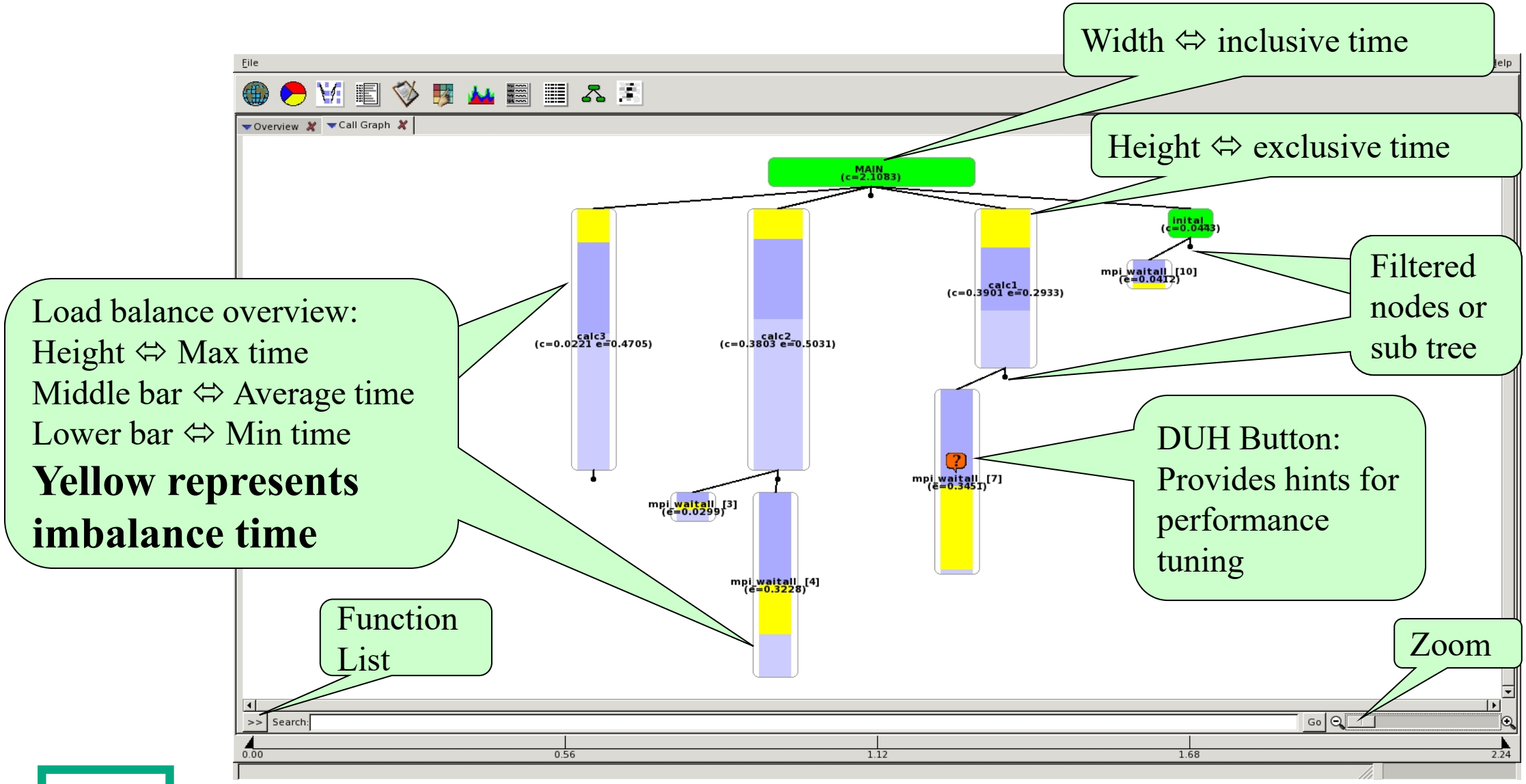
After clicking on desired function in “Sort by Calls”: Load Balance

MinLine , Avg-SdevMark , AvgLine , Avg+SdevMark , MaxLine

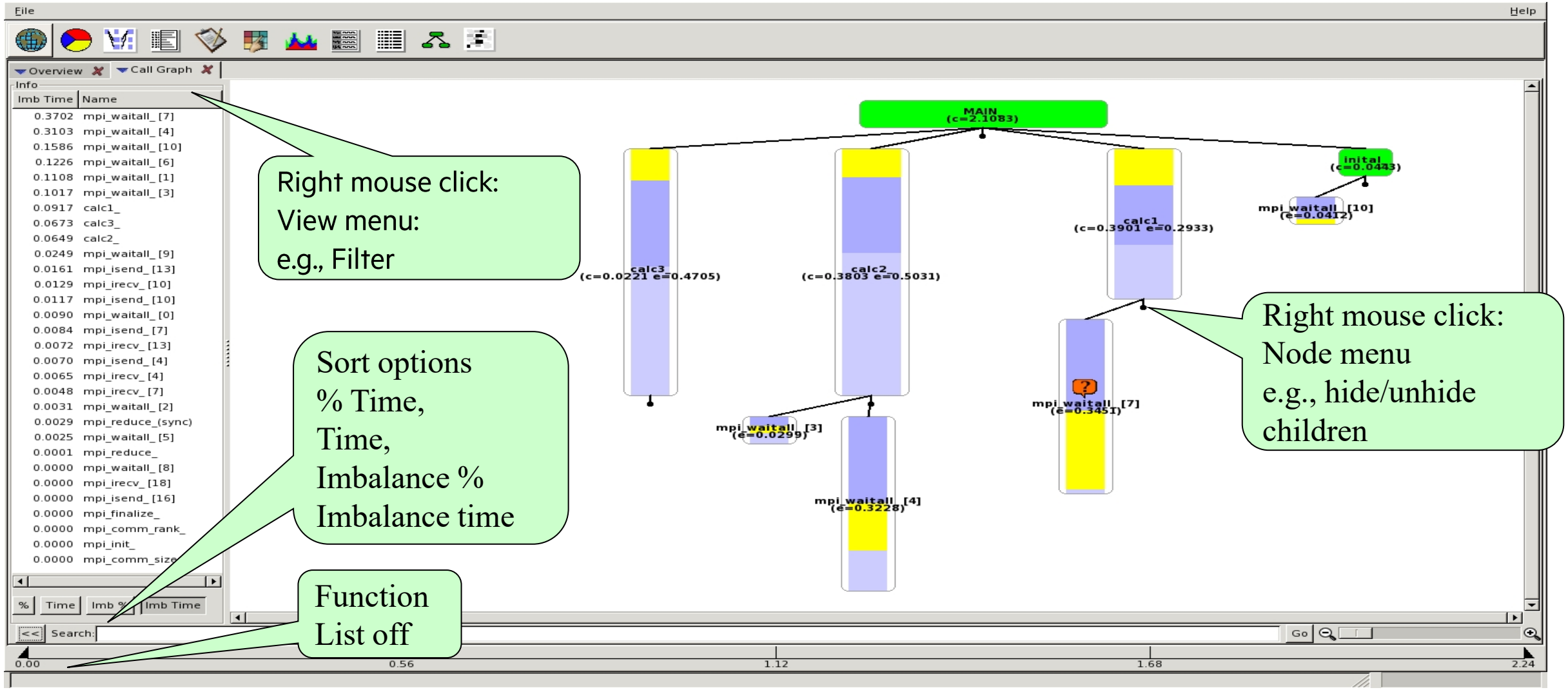
MPI COMMUNICATION HEAT MAP



CALL TREE VIEW

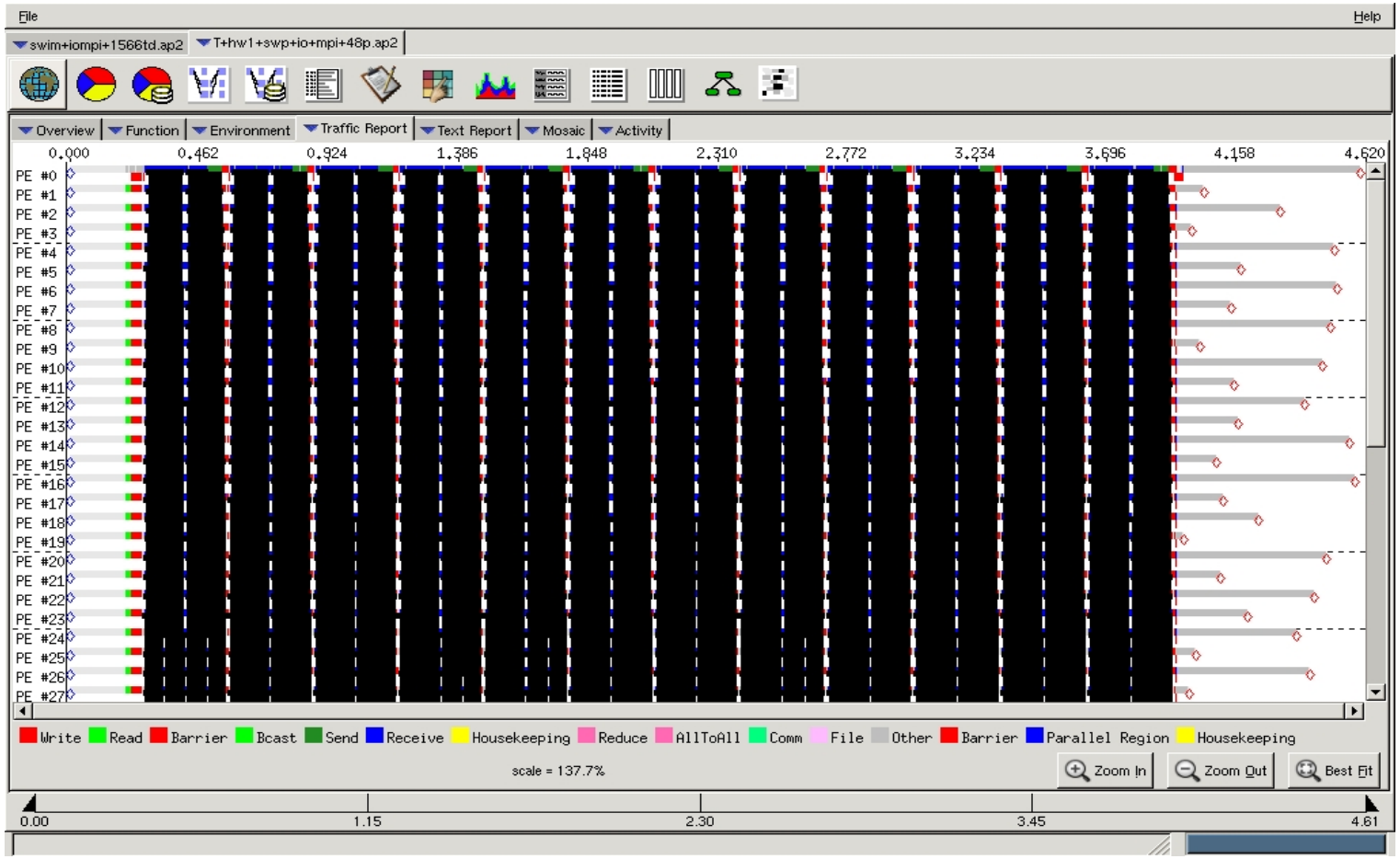


CALL TREE VIEW – FUNCTION LIST

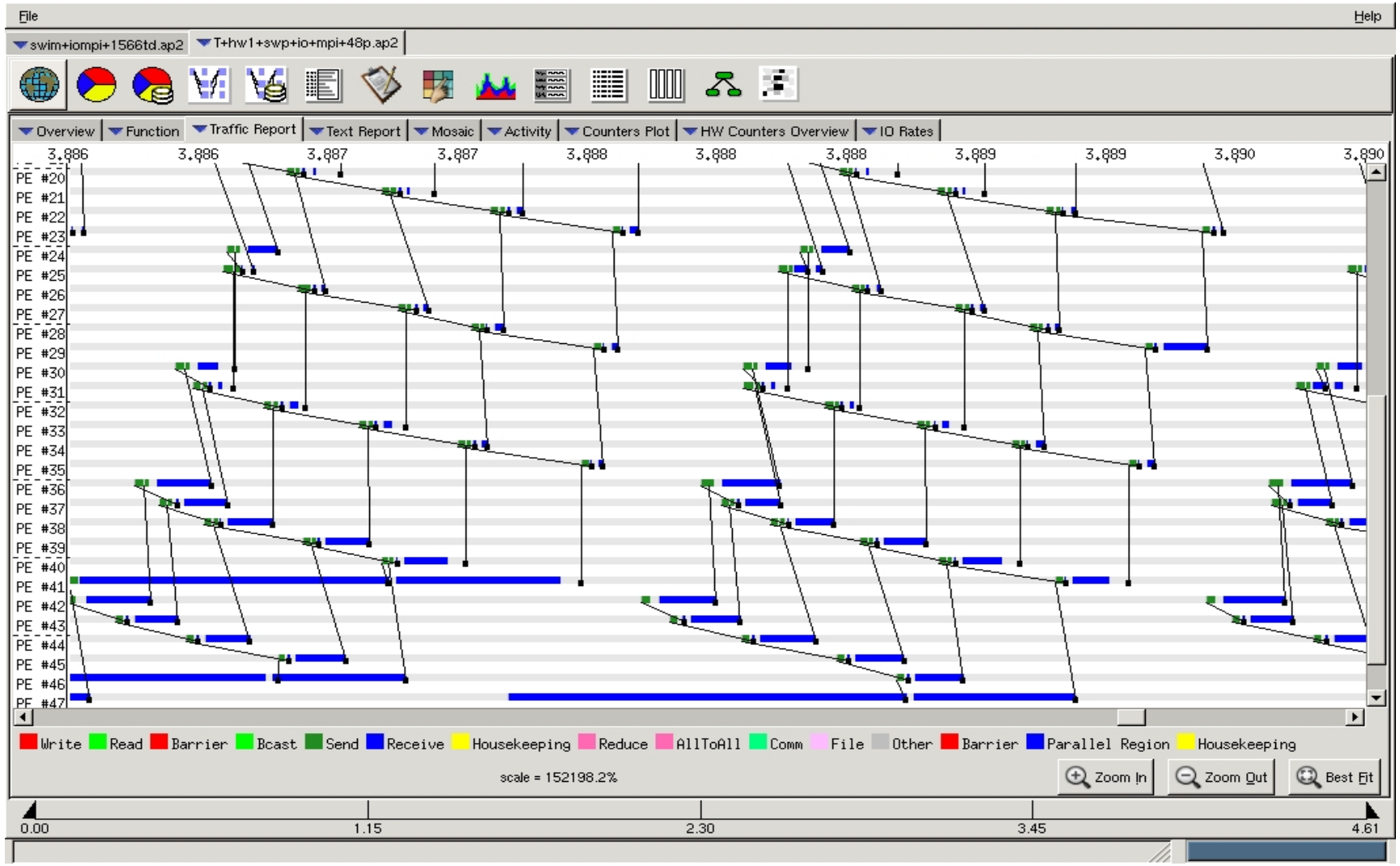


TIME LINE VIEW (communication) Traffic Tab

- 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.

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_NON_USER
  - based on ownership of compilation dir of function def
    to turn off: export PAT_REPORT_PRUNE_NON_USER=0 # while *.ap2 file is being created
- 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="“ # while *.ap2 file is being created

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 rc0 = 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 rc1 = 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 rc2 = 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'

TIP FOR BETTER SCALING

How can we improve strong scaling in the number of MPI-ranks without touching scaling in the number of MPI-ranks?

- (You are probably already doing this, but just in case you don't:)
- At https://docs.olcf.ornl.gov/systems/frontier_user_guide.html look for "SBCASTing a binary with libraries stored on shared file systems"
- Using the bare-bones test case of

```
#include <stddef.h>
#include <mpi.h>
int main()   // cc -o mpi_blip mpi_blip.cc  # stddef.h for NULL
{  MPI_Init_thread(NULL, NULL, MPI_THREAD_MULTIPLE, NULL);
   MPI_Finalize();
}
```

- **#SBATCH --constraint=nvme**

```
cd $SLURM_SUBMIT_DIR
x=./mpi_blip
export LD_LIBRARY_PATH="/mnt/bb/$USER/${x}_libs:${LD_LIBRARY_PATH}"
time sbcast --send-libs -pf $x /mnt/bb/$USER/$x
if [[ $? != "0" ]]; then echo "error: sbcast failed with '$?'"; exit 1; fi
time srun -N $NUM_NODES -n $NUM_RANKS -c 7 $x
```

TIP FOR BETTER SCALING (CONT.)

- srun wallclock times in seconds (sbcast was always <= 1 second)

```
  submit from: #   $HOME   # /lustre/orion
   use sbcast: #   N   |   Y   #   N   |   Y
Nodes |  MPI   # t/s | t/s # t/s | t/s
 512  |  4096  #  66 |  14 #  14 |  11 // some noise here
1024  |  8192  # 119 |  14 #  11 |   7
2048  | 16384  # 218 |  17 #  13 |   9
4096  | 32768  # 434 |  19 # 155 |  16

==> Without using sbcast, your jobs' strong scaling behavior
    may appear undeservedly slow, because
    as the jobs' wall clock time shrinks with more MPI-ranks,
    the startup time increases due to loading shared libs to more nodes.

==> Consider developing the habit of using that sbcast approach.
    (For production jobs built and run without module perftools,
     i.e., not profiling runs, may want to trim LD_LIBRARY_PATH
     after using the sbcast trick, see Frontier user guide.)
```

TIP: SAVE (SOME) SW ENV MODULE VERSION COMPATIBILITY HEADACHE

- To minimize potential version compatibility issues between loaded SW env modules such as (cce , rocm , cray-mpich , cray-libsci , perftools-base , …)
  start out with loading a desired cpe/* module, e.g., cpe/23.12
  (=cpe/YY.MM) whose modules are expected to be mutually compatible.
- Then, add/change modules as desired, e.g., a different rocm module and then
  check compatibility with `module show MODNAME 2>&1 | grep -iE "rocm|cce"`
- Also, here are is an incomplete/patch-work compatibility chart.
  If you see relevant omissions, please add. If you see mistakes, definitely correct.

cray-mpich	ROCm	CCE	GCC
8.1.27	5.0-5.4	14.0-16.0	9.1-12.2
8.1.28	5.5-5.7	17.0-?	>= 12.3
8.1.29	6.0-6.?	17.0-?	>= 12.3

ROCm	clang	CCE
5.3.0-5.4.3	15.0.0	15.*
5.5.0-5.6.1	16.0.0	16.*
5.7.0-6.0.0	17.0.0	17.*

Acknowledgements

- Useful input from Stephen Abbott, Mark Stock, Tanner Firl, Kostas Makrides, Luke Roskop (all HPE) and Trey White (formerly HPE, now ORNL) is gratefully acknowledged
- as is 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 OLCF Office Hours
  - Email help@olcf.ornl.gov

THANK YOU

Marcus Wagner
marcus.wagner@hpe.com