Analyzing CPU Applications with HPCToolkit

EXASCALE COMPUTING PROJECT

John Mellor-Crummey
Rice University

Tutorial
Mar-Apr 2021
NERSC and OLCF (Virtual)

National Nuclear Security Administration
U.S. DEPARTMENT OF ENERGY | Office of Science
RICE
WISCONSIN UNIVERSITY OF WISCONSIN–MADISON

A Few More Things

- **Events for CPU performance measurement**
- **OpenMP tools interface**
- **Differential performance analysis (useful for CPU and GPU)**
- **Kernel sampling**
- **Context recycling for dynamic threads**

Sample Sources - I

Linux thread-centric timers

- CPUTIME (DEFAULT if no sample source is specified)
  - CPU time used by the thread in microseconds
  - does not include time blocked in the kernel
    - disadvantage: completely overlooks time a thread is blocked
    - advantage: a blocked thread is never unblocked by sampling
- REALTIME
  - real time used by the thread in microseconds
  - includes time blocked in the kernel
    - advantage: shows where a thread spends its time, even when blocked
    - disadvantages
      - activates a blocked thread to take a sample
      - a blocked thread appears active even when blocked

Best for analysis of profile data

Produces more intuitive traces

Note: Only use one Linux timer to measure an execution

Sample Sources - II

Linux perf_event monitoring subsystem

- Kernel subsystem for performance monitoring
- Access and manipulate
  - hardware counters: cycles, instructions, …
  - software counters: context switches, page faults, …
- Available in Linux kernels 2.6.31+

A useful explanation about events available through perf
https://sites.google.com/site/lbathen/research/perf

perf_event Hardware Event Counters

- PERF_COUNT_HW_CPU_CYCLES
- PERF_COUNT_HW_INSTRUCTIONS
- PERF_COUNT_HW_CACHE_REFERENCES
- PERF_COUNT_HW_CACHE_MISSES
- PERF_COUNT_HW_BRANCH_INSTRUCTIONS
- PERF_COUNT_HW_BRANCH_MISSES
- PERF_COUNT_HW_BUS_CYCLES
- PERF_COUNT_HW_STALLED_CYCLES_FRONTEND
- PERF_COUNT_HW_STALLED_CYCLES_BACKEND
- PERF_COUNT_HW_REF_CPU_CYCLES

perf_event Hardware Cache Events

- Hardware cache
  - PERF_COUNT_HW_CACHE_L1D
  - PERF_COUNT_HW_CACHE_L1I
  - PERF_COUNT_HW_CACHE_LL
  - PERF_COUNT_HW_CACHE_DTLB
  - PERF_COUNT_HW_CACHE_ITLB
  - PERF_COUNT_HW_CACHE_BPU
  - PERF_COUNT_HW_CACHE_NODE
- Operations
  - PERF_COUNT_HW_CACHE_OP_READ
  - PERF_COUNT_HW_CACHE_OP_WRITE
  - PERF_COUNT_HW_CACHE_OP_PREFETCH
- Results
  - PERF_COUNT_HW_CACHE_RESULT_ACCESS
  - PERF_COUNT_HW_CACHE_RESULT_MISS

perf_event Software Events

- PERF_COUNT_SW_CPU_CLOCK
- PERF_COUNT_SW_TASK_CLOCK
- PERF_COUNT_SW_PAGE_FAULTS
- PERF_COUNT_SW_CONTEXT_SWITCHES
- PERF_COUNT_SW_CPU_MIGRATIONS
- PERF_COUNT_SW_PAGE_FAULTS_MIN
- PERF_COUNT_SW_PAGE_FAULTS_MAJ
- PERF_COUNT_SW_ALIGNMENT_FAULTS
- PERF_COUNT_SW_EMULATION_FAULTS

Measuring Other Hardware Events

- See the full list of available events with
  - hpcrun -L
- Perf events are grouped by categories indicated by a prefix
  - ix86arch::<event> // Intel architecture
  - perf::<event> // perf_event builtin
  - bdw_ep::<event> // Broadwell EP specific
  - …
- For convenience
  - you may omit the category prefix, e.g. “perf::”
  - you may specify counter names using lower case

Multiplexing Events

- In a single execution, you can measure more hardware events than the number of hardware counters available per thread
- If you specify more events than counters available
  - perf_events will automatically multiplex them
- How multiplexing works with Linux perf_event subsystem
  - at any time, the number of events being collected will not exceed the number of hardware counters available per thread
  - the kernel will partition events into sets that can be monitored simultaneously using hardware counter resources
  - the kernel will monitor one set of events for a while then switch to another
  - monitoring of event sets is scheduled in round-robin fashion
  - while multiplexing is convenient, there is some loss of accuracy
    - my advice: multiplexing is fine for casual execution analysis

Controlling perf_event Sampling Frequency

- Automatic Recommended
  - HPCToolkit samples perf_event counters min(300x/second, maximum Linux allows)
    - may be higher than necessary for long executions

      reducing the frequency will reduce measurement overhead
- Specify frequency
  - use the @f<freq> suffix for an event to specify frequency
    - hpcrun -e CYCLES@f100 -e INSTRUCTIONS@f200 …
  - Specify a different default frequency using the -c option
    - example: sample both CYCLES and INSTRUCTION 200x per second

      hpcrun -c f200 -e CYCLES -e INSTRUCTIONS
- Specify period
  - Use the @<period> suffix for an event to specify a period
    - hpcrun -e CYCLES@1000000 -e INSTRUCTIONS@5000000 …

How to Specify What to Measure

- Dynamically-linked executables
  - when you launch your program, use hpcrun, e.g.

```
hpcrun -e perf::CYCLES  -e perf::CS -e snb::PAGE_WALKS:LLC_MISS  ./hello_world
```

- Statically-linked executable
  - at compile time, link your executable with hpclink
  - when you launch your program, set HPCRUN_EVENT_LIST, e.g.

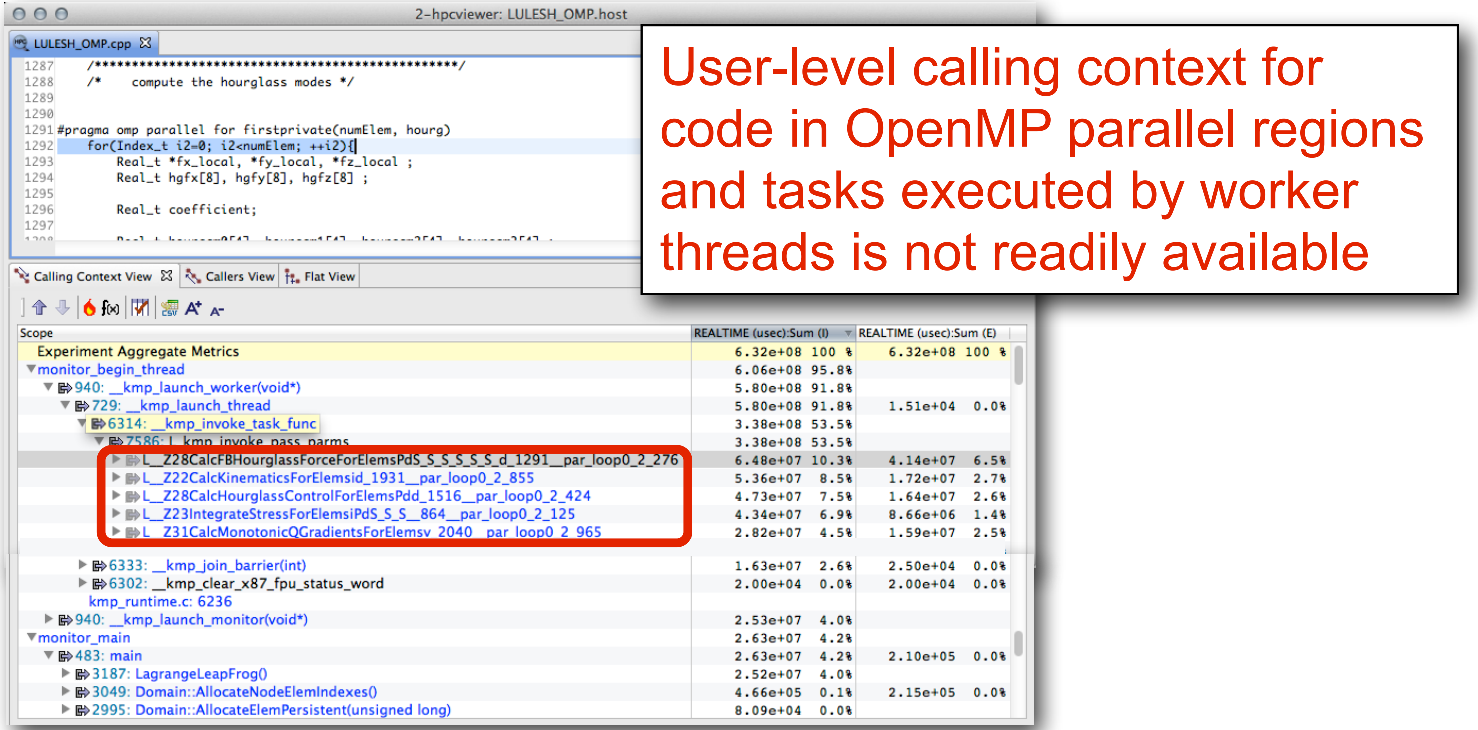
```
export HPCRUN_EVENT_LIST=“perf::CYCLES,perf::CS,snb::PAGE_WALKS:LLC_MISS” ./hello_world
```

A Few More Things

- Events for CPU performance measurement
- OpenMP tools interface
- Differential performance analysis (useful for CPU and GPU)
- Kernel sampling
- Context recycling for dynamic threads

OpenMP: A Challenge for Tools

- **Large gap between between threaded programming models and their implementations**



- **Runtime support is necessary for tools to bridge the gap**

Challenges for OpenMP Node Programs

- Tools provide implementation-level view of OpenMP threads
  - asymmetric threads
    - master thread
    - worker thread
  - run-time frames are interspersed with user code
- Hard to understand causes of idleness
  - long serial sections
  - load imbalance in parallel regions
  - waiting for critical sections or locks

OMPT: An OpenMP Tools API

- Goal: a standardized tool interface for OpenMP
  - prerequisite for portable tools
  - missing piece of the OpenMP language standard
- Design objectives
  - enable tools to measure and attribute costs to application source and runtime system
    - support low-overhead tools based on asynchronous sampling
    - attribute to user-level calling contexts
    - associate a thread's activity at any point with a descriptive state
  - minimize overhead if OMPT interface is not in use
    - features that may increase overhead are optional
  - define interface for trace-based performance tools
  - don't impose an unreasonable development burden
    - runtime implementers
    - tool developers

OpenMP Tool API Status

- Currently HPCToolkit supports OMPT interface based on OpenMP 5.0
- OMPT prototype implementations
  - LLVM
    - interoperable with GNU, Intel compilers
    - still a work in progress
  - IBM LOMP (currently targets OpenMP 5)
- Ongoing work
  - refining OpenMP 5.1 OMPT support in LLVM
  - fine tuning OMPT support in HPCToolkit

OMPT and Tutorial Examples

- On Cori, we are using a copy of the LLVM OpenMP runtime with an OpenMP tools interface developed by Rice
  - "module load openmp/ompt"
  - instead of most time in <thread root>, most time is merged into <program root> - global user-level view
  - software stack was amenable to replacing OpenMP runtime with ours for CPU examples
  - Intel OpenMP runtime in Cray's modules has an issue that triggers an assert in HPCToolkit
    - turn off OMPT to use with Intel OpenMP: export OMP_TOOL=disabled
- On Summit, the OpenMP runtime was not amenable to replacement simply by adding an entry at the front of LD_LIBRARY_PATH
  - only practical choice: use the IBM and PGI implementations that are wired into binary library paths

A Note about OMPT on Cori

- **Thread <omp idle> time is unfortunately reported as <omp barrier> with the LLVM OpenMP + HPCToolkit installed on Cori**
  - **both mean that the thread is idle, so while disconcerting, it still is meaningful**
- **The barrier is reported with the call stack where the thread was last working**
- **This "mistake" comes from too literally reporting the runtime internal state**
  - **we plan to diagnose and fix the problem**
  - **couldn't be done before the workshop**

If you are working on Cori, you can observe the effect of the OpenMP tools interface by editing the run script for amg2013 or hpcg, add

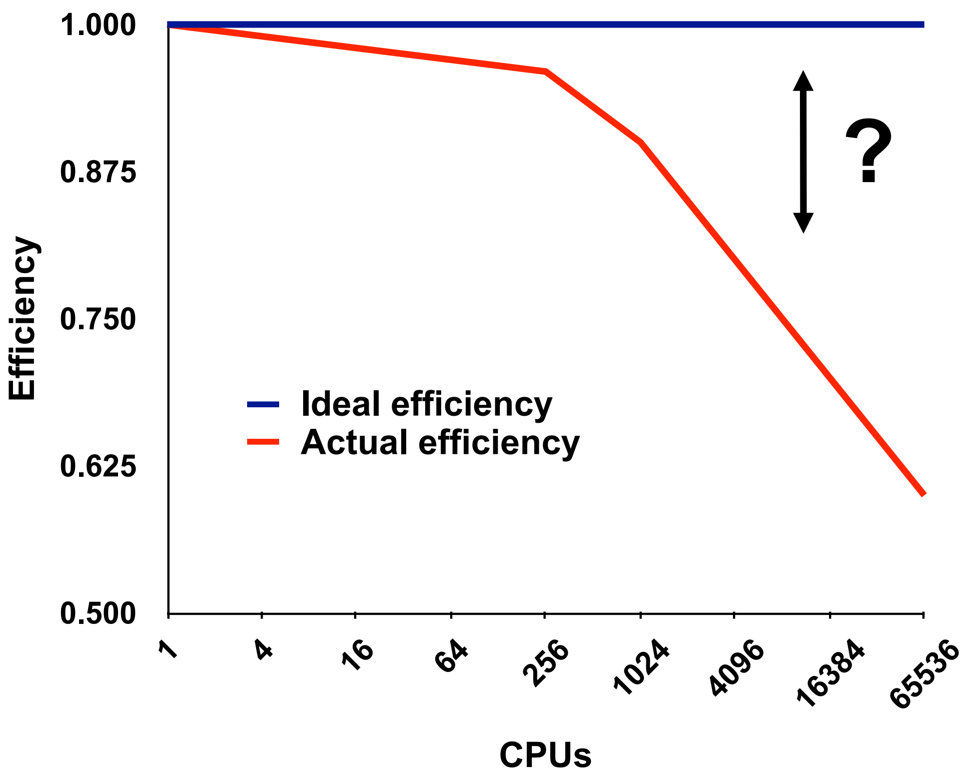
export OMP_TOOL=disabled

in one of the Cori "run" scripts before collecting data with hpcrun

A Few More Things

- Events for CPU performance measurement
- OpenMP tools interface
- Differential performance analysis (useful for CPU and GPU)
- Kernel sampling
- Context recycling for dynamic threads

The Problem of Scaling



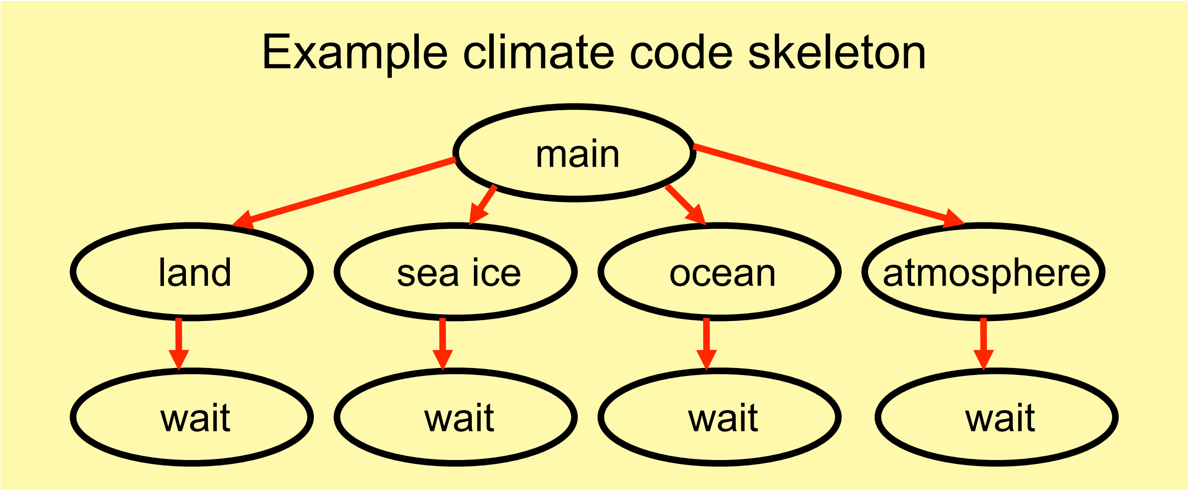
Note: higher is better

Goal: Automatic Scalability Analysis

- **Pinpoint scalability bottlenecks**
- **Guide user to problems**
- **Quantify the magnitude of each problem**
- **Diagnose the nature of the problem**

Challenges for Pinpointing Scalability Bottlenecks

- **Parallel applications**
  - modern software uses layers of libraries
  - performance is often context dependent

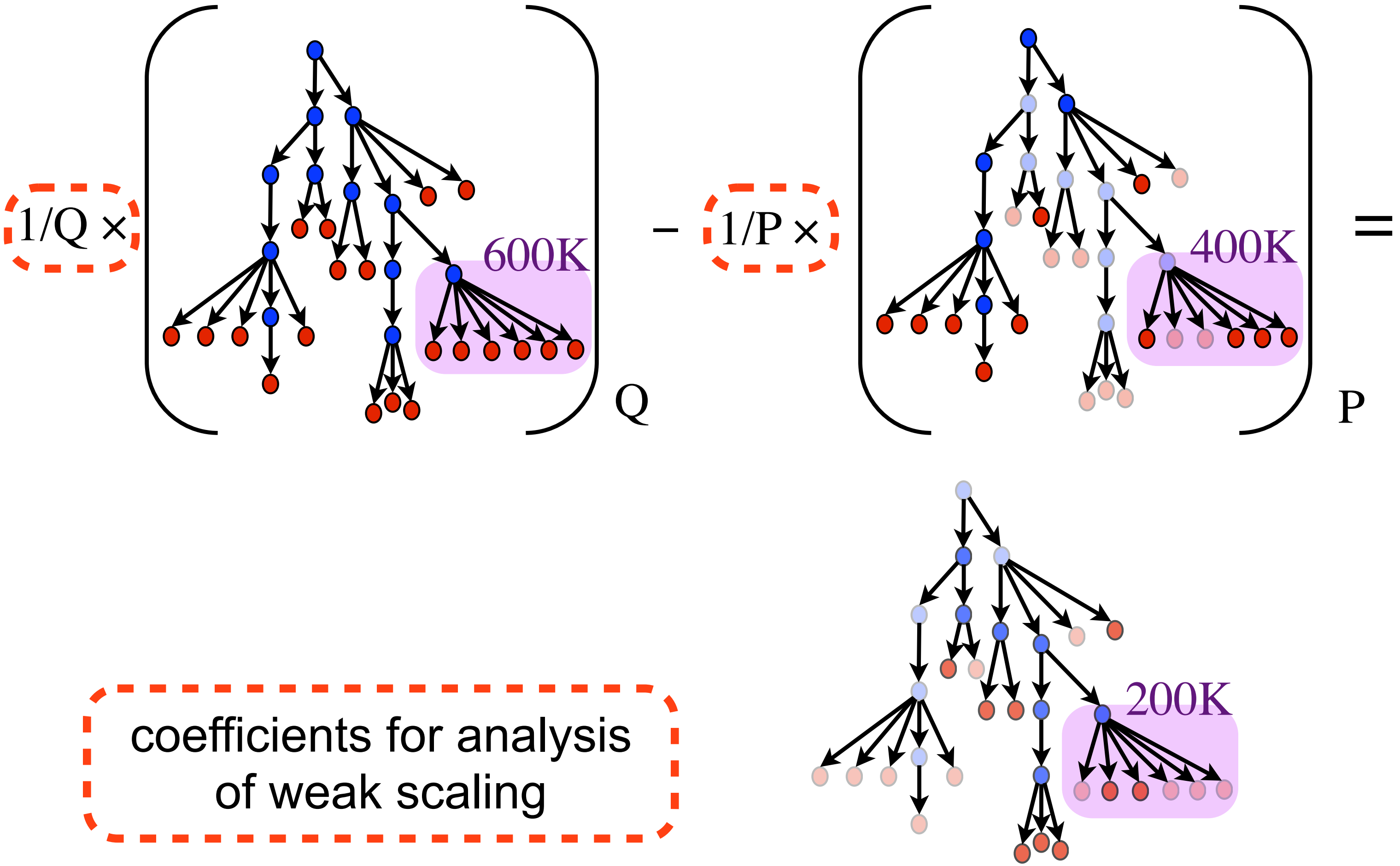


- **Monitoring**
  - bottleneck nature: computation, data movement, synchronization?
  - 2 pragmatic constraints
    - acceptable data volume
    - low perturbation for use in production runs

Performance Analysis with Expectations

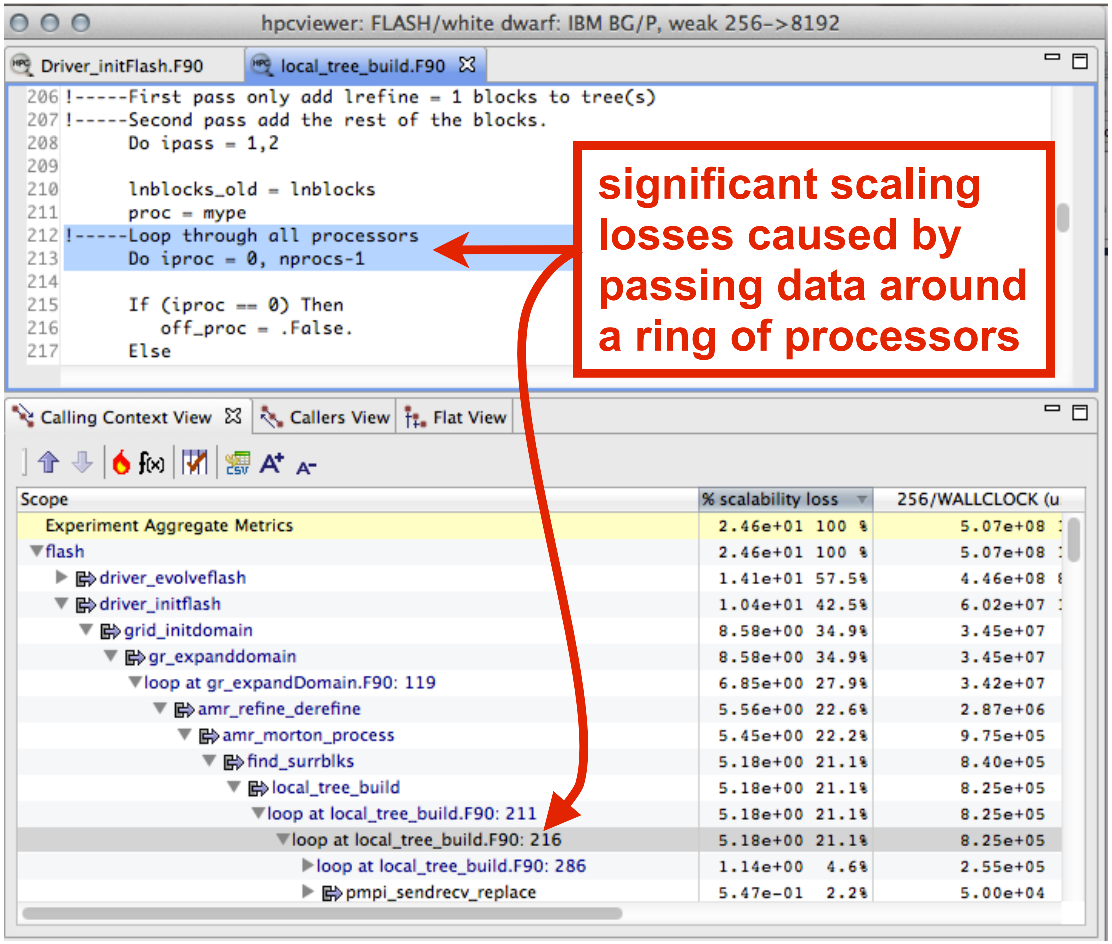
- You have performance expectations for your parallel code
  - — strong scaling: linear speedup
  - — weak scaling: constant execution time
- Put your expectations to work
  - — measure performance under different conditions
    - – e.g. different levels of parallelism or different inputs
  - — express your expectations as an equation
  - — compute the deviation from expectations for each calling context
    - – for both inclusive and exclusive costs
  - — correlate the metrics with the source code
  - — explore the annotated call tree interactively

Pinpointing and Quantifying Scalability Bottlenecks



Scalability Analysis

- **Difference call path profile from two executions**
  - **different number of nodes**
  - **different number of threads**
- **Pinpoint and quantify scalability bottlenecks within and across nodes**



See the HPCToolkit manual for the detailed description of how to do this in practice

Using Differential Performance Analysis

- The example shown was constructed by building a database with a single MPI rank from each of two executions at different scales
  - you can call hpcprof/hpcprof-mpi with .hpcrun files as arguments instead of analyzing a whole measurement directory
- You can do strong or weak scaling analysis on your own by
  - providing two measurement directories to hpcprof/hpcprof-mpi
  - writing an equation to compute the scaling loss from one to the other
- Tree vs. forest?
  - the Flash example shown had a calling context tree
  - when analyzing OpenMP programs without the OpenMP Tools API (OMPT), you get a forest
    - typically two roots for OpenMP codes: <program root>, <thread root>
    - top-down scaling comparisons are problematic for a forest
    - bottom-up scaling comparisons can be informative for a forest

      they focus on WHAT you are doing at the leaves, irrespective of whether the OpenMP master thread or a worker thread did the work

A Few More Things

- Events for CPU performance measurement
- OpenMP tools interface
- Differential performance analysis (useful for CPU and GPU)
- Kernel sampling
- Context recycling for dynamic threads

Kernel Sampling in HPCToolkit

- When sampling using the Linux perf_event subsystem
  - sample user space activity
  - sample kernel space activity
- When a thread is active in the kernel, the user calling context is frozen
- Attribute kernel activity to the point where it occurred in the user calling context
  - form a calling context that has
    - user calling context as the prefix
    - kernel calling context as the suffix

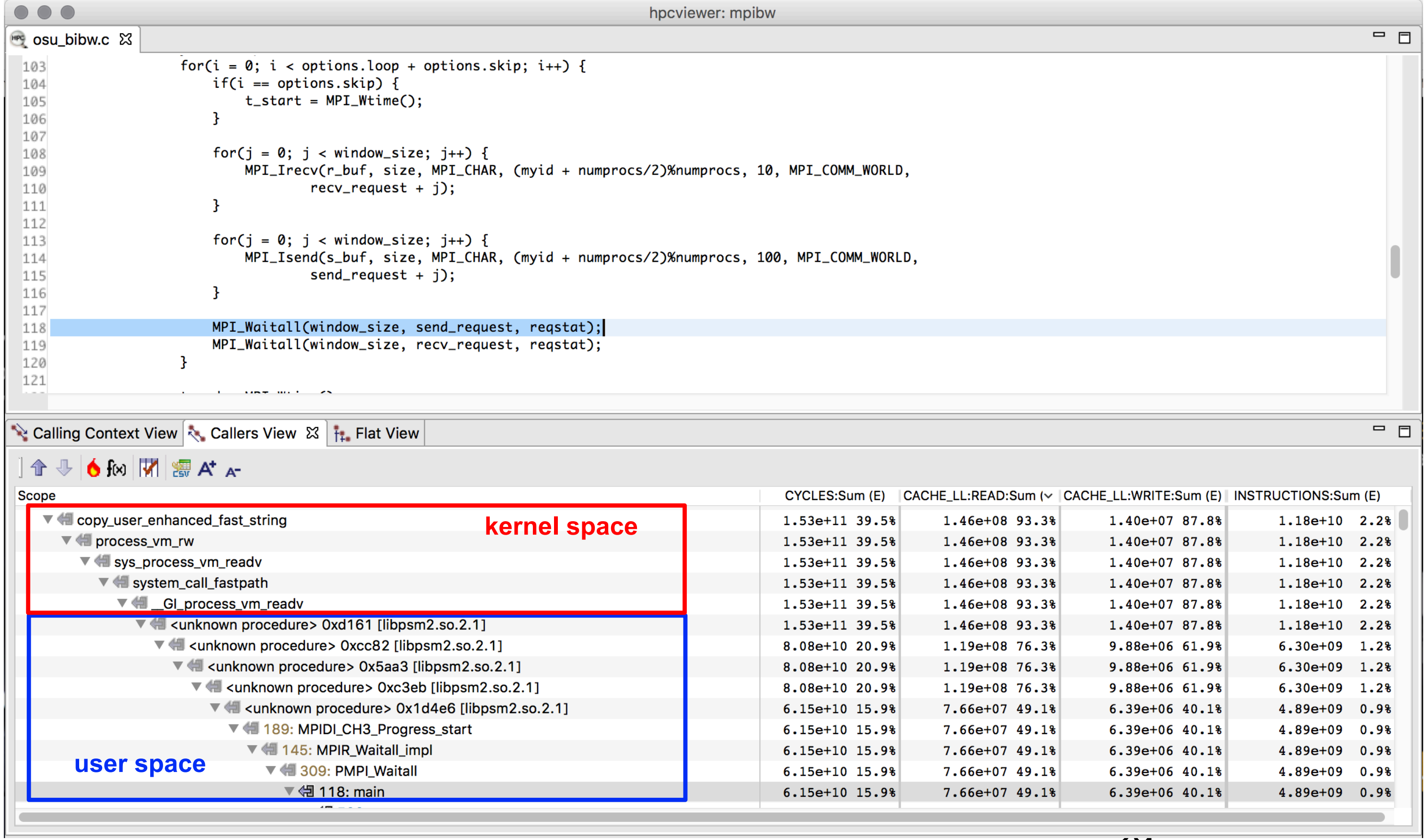
Kernel Sampling Yields Insight

Investigating MPI Performance with Kernel Sampling

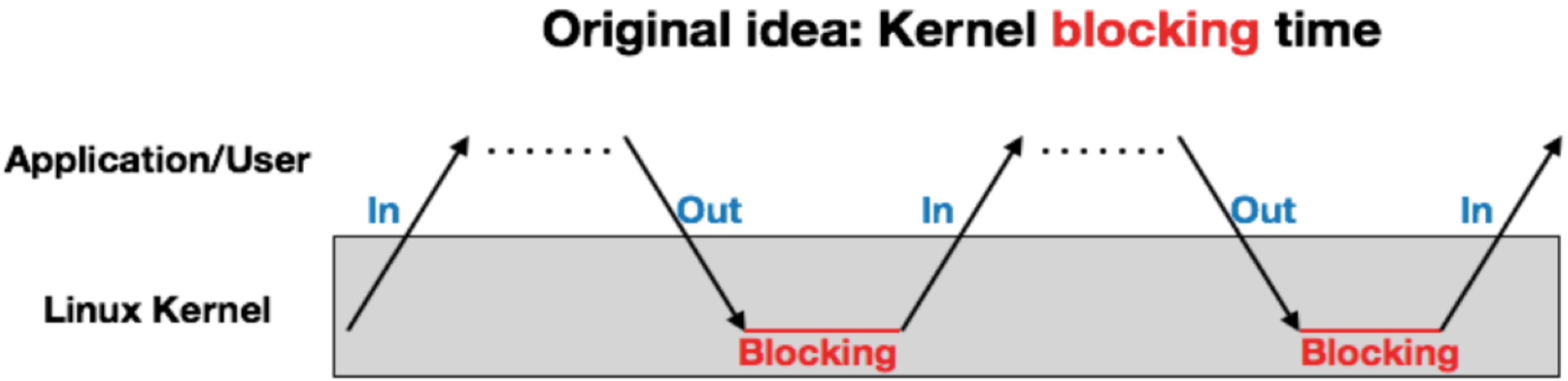
Platform
- Intel Broadwell
- Infiniband network

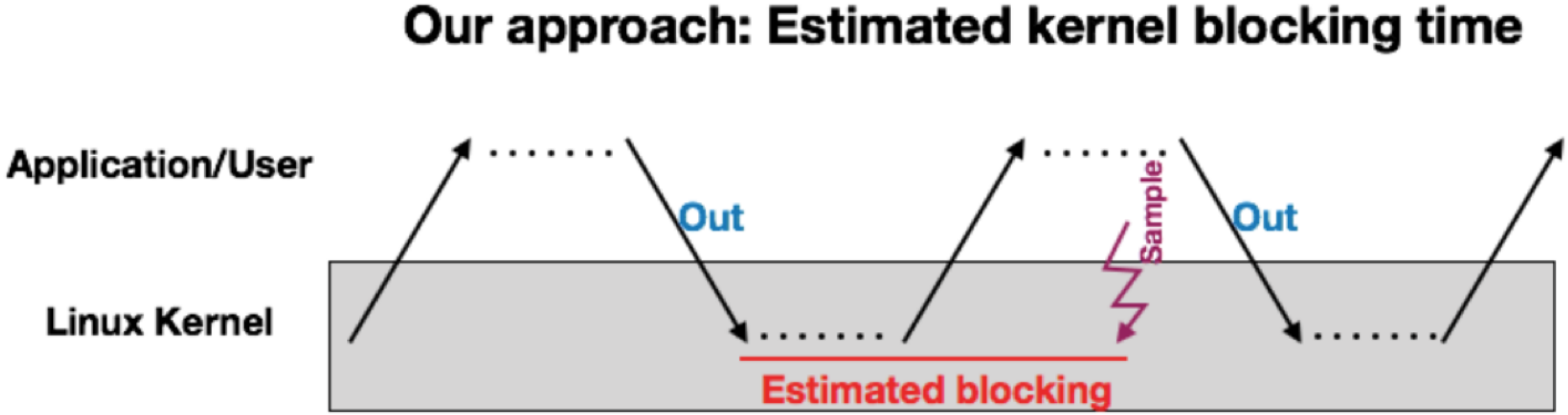
- Q: Why is MPI communication bandwidth so low on node (6-9 GB/s)?

- A: Bounded by single thread memory bandwidth
- Memcpy 12 GB/s
- Stream (1T) 8-9 GB/s
- Stream (OMP) 60 GB/s



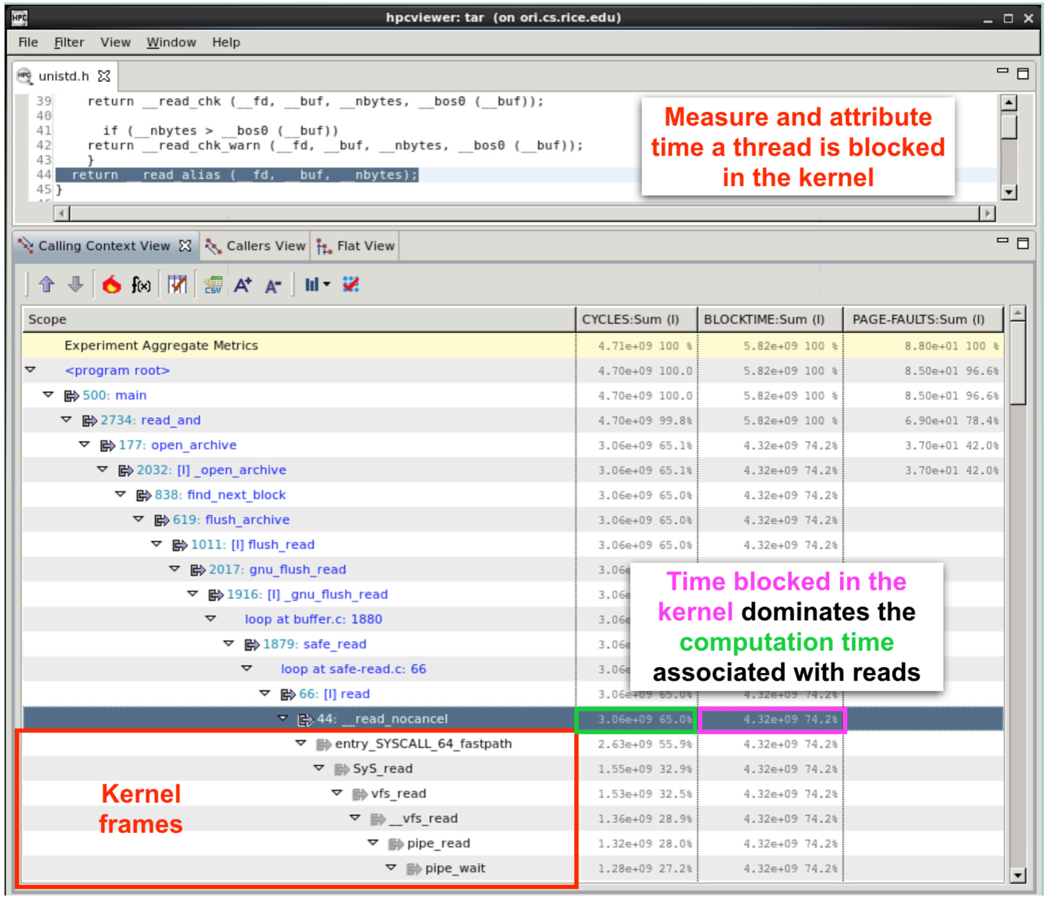
Measure Thread Blocking using perf_events





Example: Thread Blocking in “tar”

hpcrun -e CYCLES -e BLOCKTIME -e PAGE-FAULTS tar xzf \
~/Downloads/eclipse-rcp-indigo-linux-gtk-x86_64.tar.gz

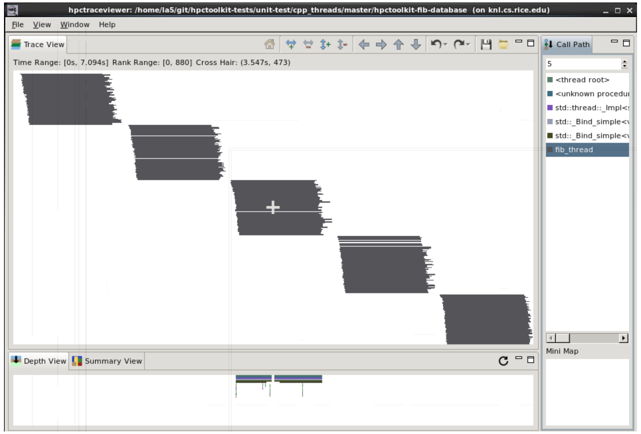


A Few More Things

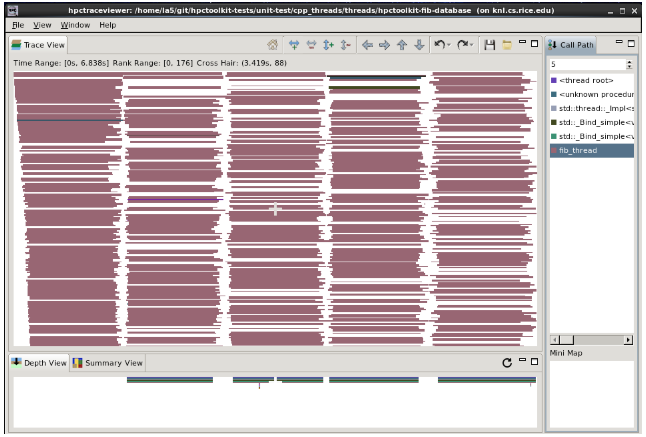
- Events for CPU performance measurement
- OpenMP tools interface
- Differential performance analysis (useful for CPU and GPU)
- Kernel sampling
- Context recycling for dynamic threads

Context Recycling for Short-lived Threads

- **Problem**
  - — some codes create many short-lived threads
    - – DCA+ 160 ranks generated 1.2M thread profiles and traces
  - — time-centric views of such codes are problematic



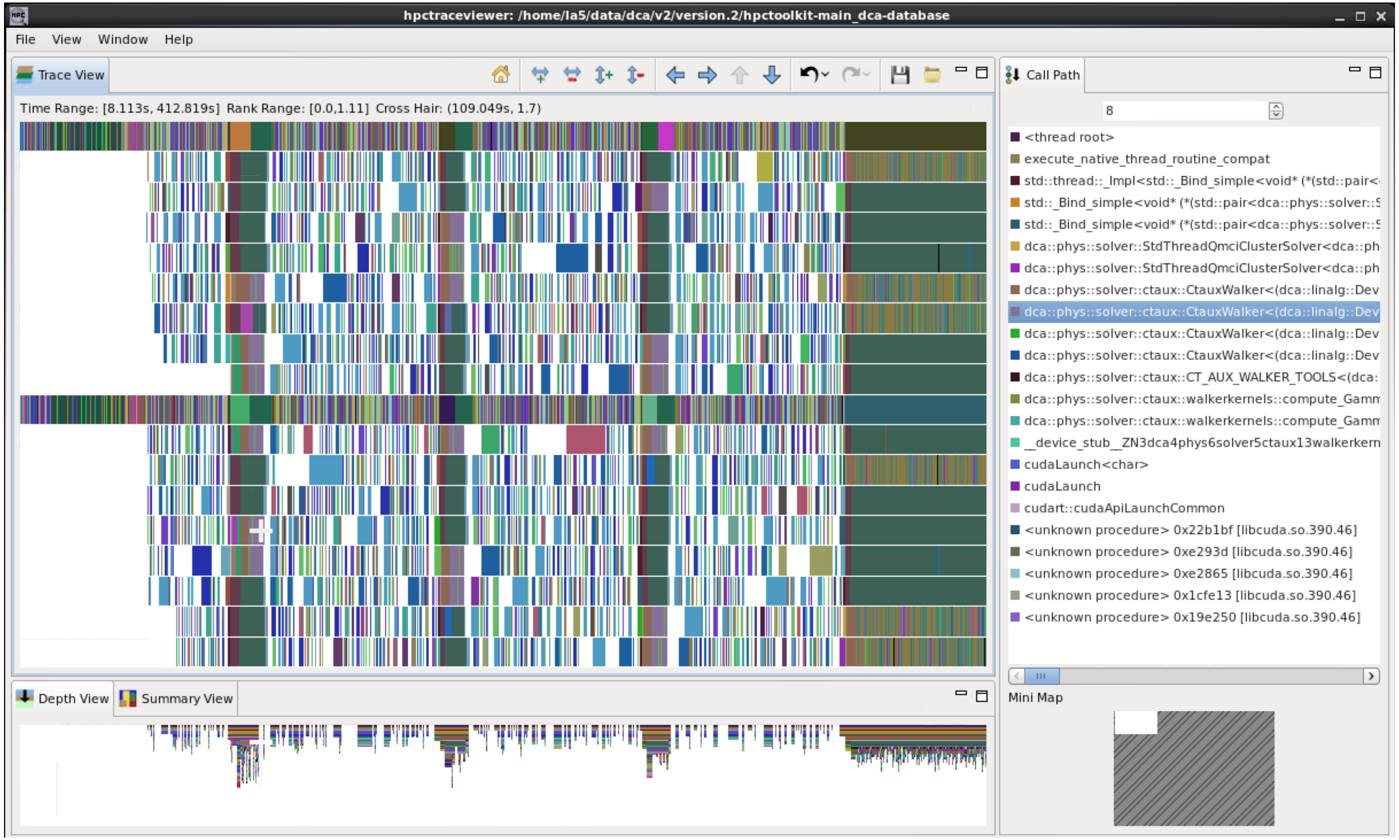
- **Solution**
  - — when a thread completes, put its (CCT, trace) in a free list
  - — when a new thread starts, look for an available (CCT, trace) pair to augment
  - — create a new one only if needed



Credit: Laksono Adhianto

DCA+ using Context Recycling

DCA+ 10 ranks, 12 threads each with context recycling



Credit: Laksono Adhianto

Detailed HPCToolkit Documentation

http://hpctoolkit.org/documentation.html

- **Comprehensive user manual:**
  - **http://hpctoolkit.org/manual/HPCToolkit-users-manual.pdf**
  - **Quick start guide**
    - **essential overview that almost fits on one page**
  - **Using HPCToolkit with statically linked programs**
  - **The hpcviewer's profile and trace views**
  - **Effective strategies for analyzing program performance with HPCToolkit**
    - **analyzing scalability, waste, multicore performance ...**
  - **HPCToolkit and MPI**
  - **HPCToolkit Troubleshooting**
- **Installation guide**
  - **http://hpctoolkit.org/software-instructions.html**

Advice for Using HPCToolkit

Advice and Troubleshooting Tips

- Compile your program with a -g in addition to optimization
  — with -g the compiler records info about line mappings and inlined code for hpcstruct's binary analyzer
- If more than just your program is of interest, use hpcstruct to analyze your libraries of interest as well
  — you can provide more than one structure file to hpcprof/hpcprof-mpi by passing multiple -S options
    – e.g. hpcprof -S my-executable -S my-library1 -S my-library-2 …
- If you lack detailed information about loops in hpcviewer
  — make sure you analyzed your binary with hpcstruct
  — make sure that you provided the structure file to hpcprof/ hpcprof-mpi

Monitoring Large Executions

- Collecting performance data on every node is typically not necessary
- Can improve scalability of data collection by recording data for only a fraction of processes
  - set environment variable HPCRUN_PROCESS_FRACTION
  - e.g. collect data for 10% of your processes
    - set environment variable HPCRUN_PROCESS_FRACTION=0.10

Digesting your Performance Data

- Use hpcstruct to reconstruct program structure
  - e.g. `hpcstruct your_app`
    - creates your_app.hpcstruct
- Correlate measurements to source code with hpcprof and hpcprof-mpi
  - run hpcprof on the front-end to analyze data from small runs
  - run hpcprof-mpi on the compute nodes to analyze data from lots of nodes/threads in parallel
- Digesting performance data in parallel with hpcprof-mpi
  - srun -n 8 hpcprof-mpi \
    -S your_app.hpcstruct \
    -I /path/to/your_app/src/+ \
    hpctoolkit-your_app-measurements.jobid