Analyzing CPU Applications with HPCToolkit

John Mellor-Crummey
Rice University

2025 Energy HPC Conference
February 28, 2025

RICE

STEP SOFTWARE TOOLS ECOSYSTEM PROJECT

U.S. DEPARTMENT OF ENERGY | Office of Science

National Nuclear Security Administration

Topics

- **Events for CPU performance measurement**
- **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

Best for analysis of profile data

- 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

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 modern Linux kernels

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 …

Topics

- Events for CPU performance measurement
- Kernel sampling
- Context recycling for dynamic threads

Topics

- Events for CPU performance measurement
- 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

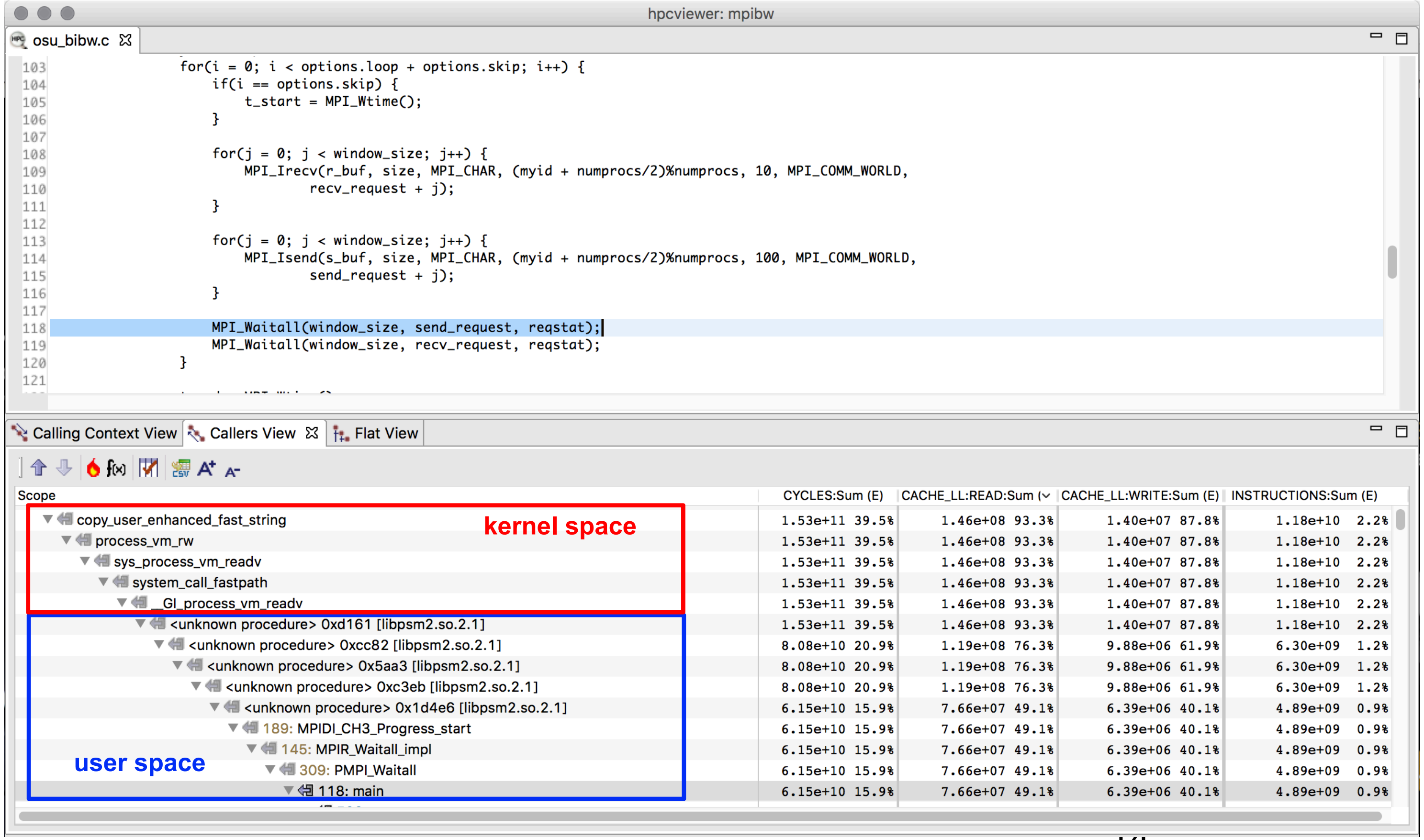
Kernel Sampling Yields Insight

Investigating MPI Performance with Kernel Sampling

Platform
- Intel Broadwell
- Infiniband network

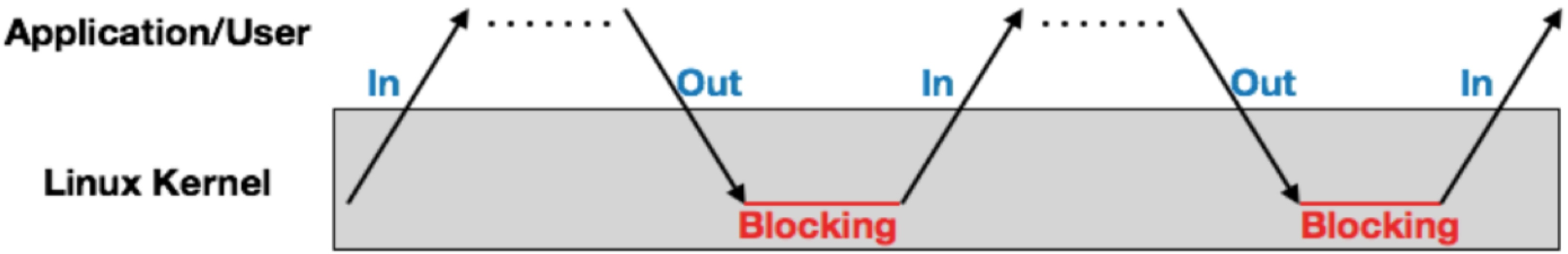
- Q: Why is MPI communication bandwidth so low on node (6-9 GB/s)?

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

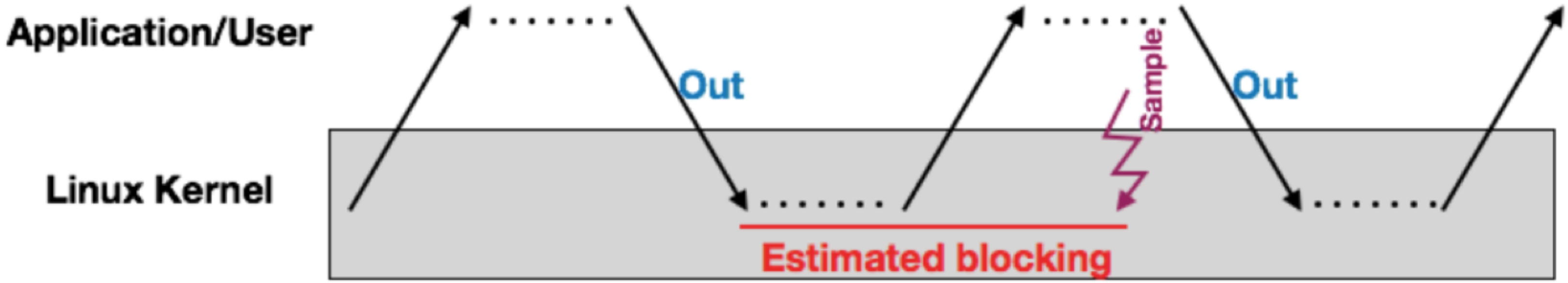


Measure Thread Blocking using perf_events

Original idea: Kernel blocking time

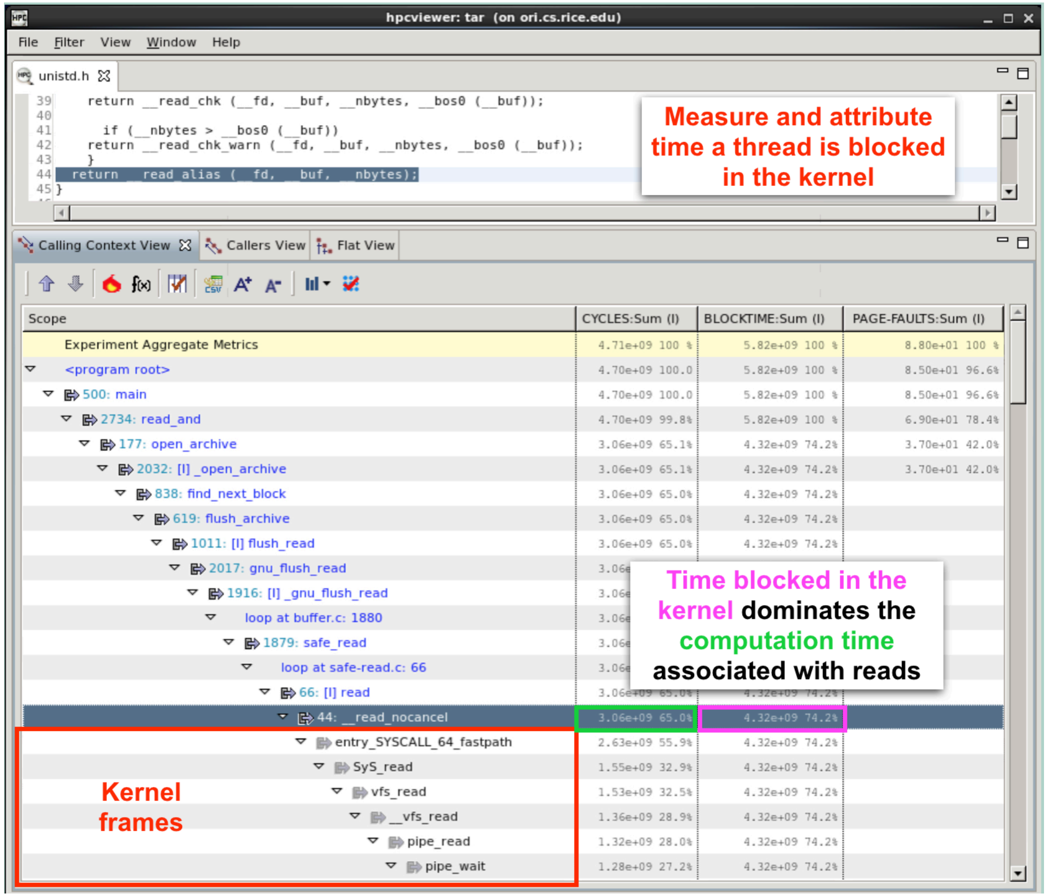


Our approach: Estimated kernel blocking time



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

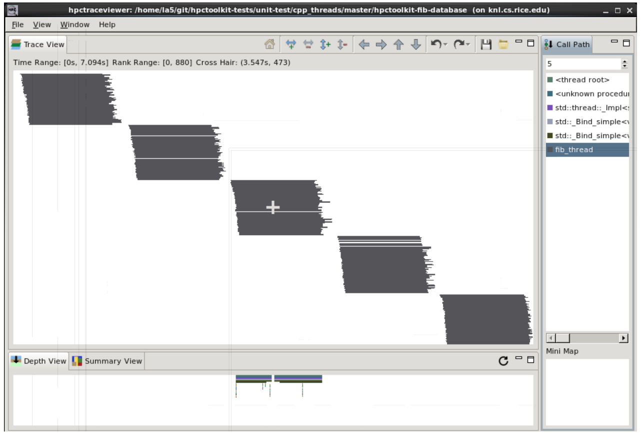


A Few More Things

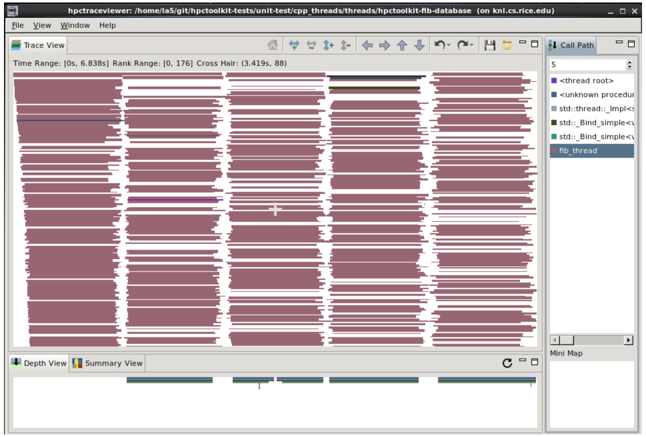
- Events for CPU performance measurement
- Differential performance analysis (useful for CPU and GPU)
- Kernel sampling
- Context recycling for dynamic threads

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



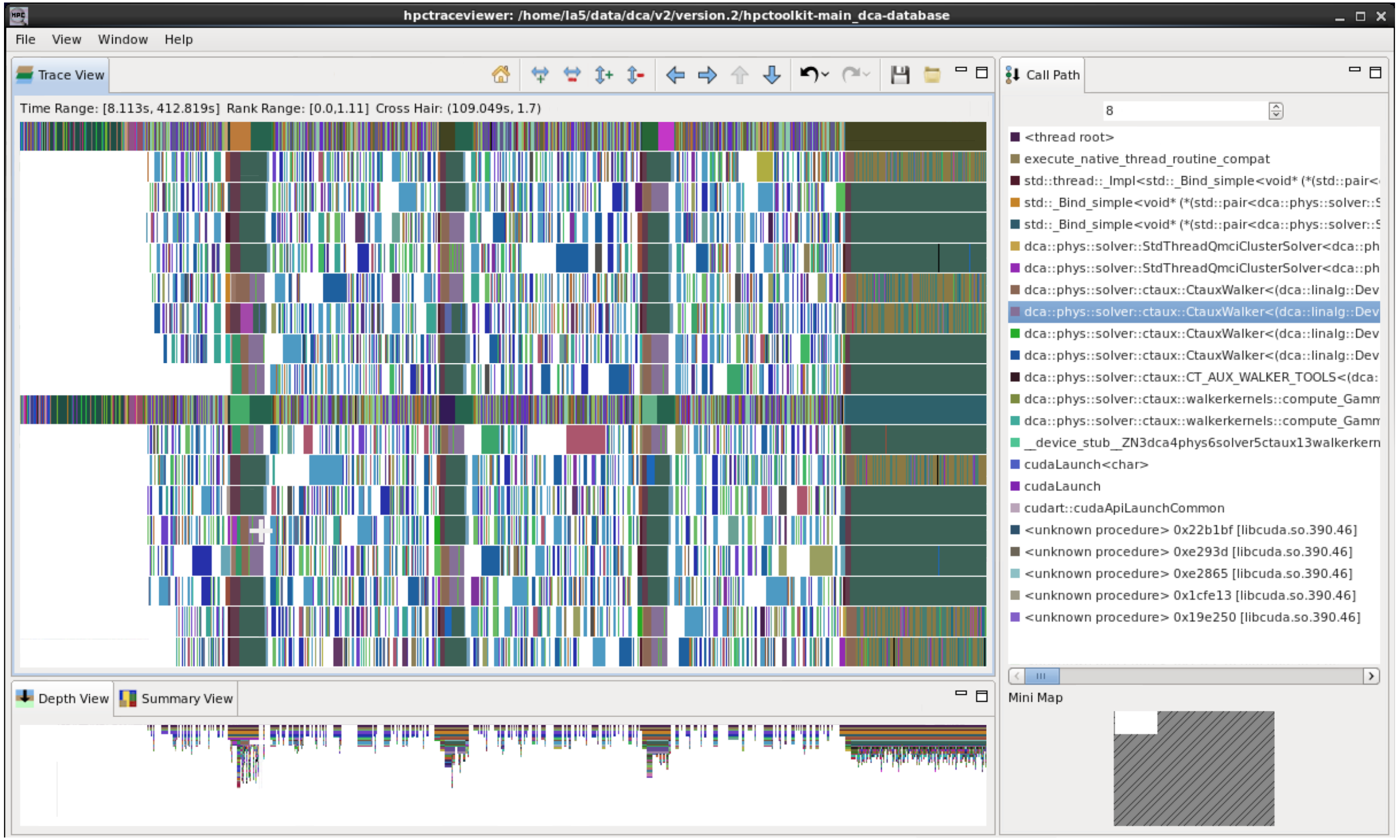
- **Solution**
  - — when a thread completes, put its (CCT, trace) in a free list
  - — when a new thread starts, look for an available (CCT, trace) pair to augment
  - — create a new one only if needed



Credit: Laksono Adhianto

DCA+ using Context Recycling

DCA+ 10 ranks, 12 threads each with context recycling



Credit: Laksono Adhianto