Analyzing CPU Applications with HPCToolkit

John Mellor-Crummey Rice University

2025 Energy HPC Conference February 28, 2025









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
- 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 modern Linux kernels

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

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

perf_event Hardware Event Counters

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

perf_event Software Events

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>
 - perf::<event>
 - bdw_ep::<event>

// Intel architecture
// perf_event builtin
// 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

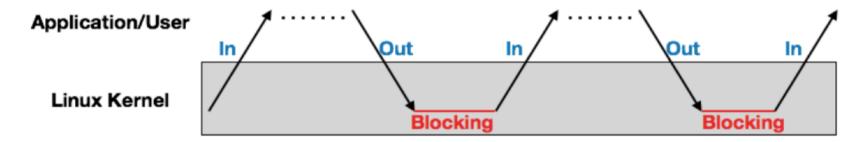
 Q: Why is MPI communication bandwidth so low on node (6-9 GB/s)?

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

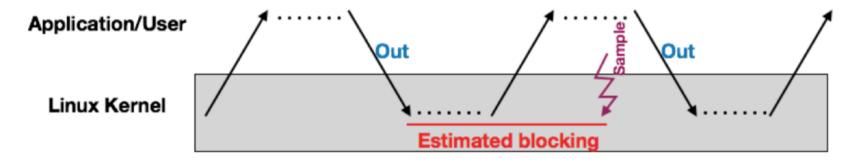
		er: mpibw				
🦻 osu_bibw.c Σ	X					
103	<pre>for(i = 0; i < options.loop + options.skip; i++) {</pre>					
104	<pre>if(i == options.skip) {</pre>					
105 106	<pre>t_start = MPI_Wtime(); }</pre>					
100	}					
108	for(j = 0; j < window_size; j++) {					
109	<pre>MPI_Irecv(r_buf, size, MPI_CHAR, (myid + numprocs/2)%numpr</pre>	rocs, 10, MPI_COMM_WORLD,				
110	<pre>recv_request + j);</pre>					
111 112	}					
112	<pre>for(j = 0; j < window_size; j++) {</pre>					
114	MPI_Isend(s_buf, size, MPI_CHAR, (myid + numprocs/2)%numpr	rocs, 100, MPI_COMM_WORLD,				
115	<pre>send_request + j);</pre>					
116	}					
117	<pre>MPI_Waitall(window_size, send_request, regstat);</pre>					
118 119	MPI_Waitall(window_size, recv_request, reqstat);					
120	}					
121	•					
Calling Conte	ext View 🔨 Callers View 🕄 🏪 Flat View					
] 🛧 🕂 💧 fox	ext View 🔦 Callers View 83 🏗 Flat View	0Y01E2.Sum (5) 00				
] 🛧 🕹 🤞 fx Scope				CACHE_LL:WRITE:Sum (E) II		ım (E
] 1 🕀 🍐 🏠 fixe Scope 💌 🕊 copy_us	ser_enhanced_fast_string	1.53e+11 39.5%	1.46e+08 93.3%	1.40e+07 87.8%	1.18e+10	ım (E 2.
Cope ▼	ser_enhanced_fast_string kernel space	1.53e+11 39.5% 1.53e+11 39.5%	1.46e+08 93.3% 1.46e+08 93.3%	1.40e+07 87.8% 1.40e+07 87.8%	1.18e+10 1.18e+10	ım (E 2. 2.
] ☆ ↓ 6 fox Scope ▼	Image: Ser_enhanced_fast_string kernel space ses_vm_rw ys_process_vm_readv	1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5%	1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3%	1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8%	1.18e+10 1.18e+10 1.18e+10	ım (E 2. 2. 2.
Cope Scope ▼ € copy_us ▼ € proc ▼ € sy ▼ €	b IV	1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5%	1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3%	1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8%	1.18e+10 1.18e+10 1.18e+10 1.18e+10	ım (E 2. 2. 2. 2.
Cope Cope Copy_us	ser_enhanced_fast_string kernel space ses_vm_rw ys_process_vm_readv system_call_fastpath G_process_vm_readv	1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5%	1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3%	1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8%	1.18e+10 1.18e+10 1.18e+10 1.18e+10 1.18e+10	ım (E 2. 2. 2. 2. 2.
Cope Scope ▼ € copy_us ▼ € proc ▼ € sy ▼ €	ser_enhanced_fast_string kernel space yses_vm_rw ys_process_vm_readv system_call_fastpath dGLprocess_vm_readv vdsuknown procedure> 0xd161 [libpsm2.so.2.1]	1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5%	1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3%	1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8%	1.18e+10 1.18e+10 1.18e+10 1.18e+10 1.18e+10 1.18e+10	IM (E 2. 2. 2. 2. 2. 2.
Cope Scope ▼ € copy_us ▼ € proc ▼ € sy ▼ €	system_call_fast_string	1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 8.08e+10 20.9%	1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.19e+08 76.3%	1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 9.88e+06 61.9%	1.18e+10 1.18e+10 1.18e+10 1.18e+10 1.18e+10 1.18e+10 6.30e+09	IM (E 2. 2. 2. 2. 2. 2. 1.
1 1 1 6 f(x) Scope ▼ 4 copy_us ▼ 4 proc ▼ 4 proc ▼ 4 sy ▼ 4 sy ▼ 4 sy	Image: Series and Series	1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 8.08e+10 20.9% 8.08e+10 20.9%	1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.19e+08 76.3%	1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 9.88e+06 61.9%	1.18e+10 1.18e+10 1.18e+10 1.18e+10 1.18e+10 1.18e+10 6.30e+09 6.30e+09	Im (E 2. 2. 2. 2. 2. 1.
1 1 1 6 f(x) Scope ▼ 4 copy_us ▼ 4 proc ▼ 4 proc ▼ 4 sy ▼ 4 sy ▼ 4 sy	Image: Series of the series	1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 8.08e+10 20.9% 8.08e+10 20.9%	1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.19e+08 76.3% 1.19e+08 76.3%	1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 9.88e+06 61.9% 9.88e+06 61.9% 9.88e+06 61.9%	1.18e+10 1.18e+10 1.18e+10 1.18e+10 1.18e+10 1.18e+10 6.30e+09 6.30e+09 6.30e+09	IM (E 2. 2. 2. 2. 2. 1. 1. 1.
Cope Scope ▼ € copy_us ▼ € proc ▼ € sy ▼ €	<pre>kernel space kernel space</pre>	1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 8.08e+10 20.9% 8.08e+10 20.9% 6.15e+10 15.9%	1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.19e+08 76.3% 1.19e+08 76.3% 1.19e+08 76.3%	1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 9.88e+06 61.9% 9.88e+06 61.9% 9.88e+06 61.9% 6.39e+06 40.1%	1.18e+10 1.18e+10 1.18e+10 1.18e+10 1.18e+10 6.30e+09 6.30e+09 6.30e+09 4.89e+09	Im (E 2 2 2 1 1 0
Ì ✿ ♣ ♠ ♠ Scope ▼	Image: Section 2016 Image: Section 2016 section 2016 Section 2016 section 2017 Section 2017 section 2017 Section 2017 <	1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 8.08e+10 20.9% 8.08e+10 20.9% 8.08e+10 20.9% 6.15e+10 15.9%	1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.19e+08 76.3% 1.19e+08 76.3% 1.19e+08 76.3% 7.66e+07 49.1%	1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 9.88e+06 61.9% 9.88e+06 61.9% 9.88e+06 61.9% 6.39e+06 40.1%	1.18e+10 1.18e+10 1.18e+10 1.18e+10 1.18e+10 6.30e+09 6.30e+09 6.30e+09 4.89e+09	Im (E 2. 2. 2. 2. 1. 1. 0.
Scope	<pre>ser_enhanced_fast_string kernel space ses_vm_rw ys_process_vm_readv system_call_fastpath GI process_vm_readv system_call_fastpath GI procedure> 0xc82 [libpsm2.so.2.1] state(unknown procedure> 0xc82 [libpsm2.so.2.1] st</pre>	1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 8.08e+10 20.9% 8.08e+10 20.9% 8.08e+10 20.9% 6.15e+10 15.9% 6.15e+10 15.9%	1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.19e+08 76.3% 1.19e+08 76.3% 7.66e+07 49.1% 7.66e+07 49.1%	1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 9.88e+06 61.9% 9.88e+06 61.9% 9.88e+06 61.9% 6.39e+06 40.1% 6.39e+06 40.1%	1.18e+10 1.18e+10 1.18e+10 1.18e+10 1.18e+10 6.30e+09 6.30e+09 6.30e+09 4.89e+09 4.89e+09	Im (E 2 2 2 1 1 0 0
Scope	Image: Section 2016 Image: Section 2016 section 2016 Section 2016 section 2017 Section 2017 section 2017 Section 2017 <	1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 1.53e+11 39.5% 8.08e+10 20.9% 8.08e+10 20.9% 8.08e+10 20.9% 6.15e+10 15.9%	1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.46e+08 93.3% 1.19e+08 76.3% 1.19e+08 76.3% 1.19e+08 76.3% 7.66e+07 49.1%	1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 1.40e+07 87.8% 9.88e+06 61.9% 9.88e+06 61.9% 9.88e+06 61.9% 6.39e+06 40.1%	1.18e+10 1.18e+10 1.18e+10 1.18e+10 1.18e+10 6.30e+09 6.30e+09 6.30e+09 4.89e+09	2.: 2.: 2.: 2.: 1.: 1.: 0.: 0.: 0.:

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

IPC	hpcviewer: tar (on ori.cs	.rice.edu)				_ = :		
File <u>Filter</u> View <u>W</u> indow Help)							
unistd.h 🔀						- 6		
	fd,buf,nbytes,bos0 (buf));							
40 41 if (nbytes > 1			Measure and attribute					
<pre>42 returnread_chk_wa 43 }</pre>	arn (fd,buf,nbytes,bos0 (bu	f));	time	e a thread is	blocked			
<pre>44 returnread_alias (45 }</pre>	fd,buf,nbytes);		in the kernel					
🔧 Calling Context View 🔀 🔖 Ca	allers View 👯 Flat View					- 6		
1 1 4 6 fx 17 5				I				
Scope		CYCLES:	Sum (l)	BLOCKTIME:Sum (I)	PAGE-FAULTS:Sum	ωĺ≏		
Experiment Aggregate Metr	rics		09 100 %	5.82e+09 100 %	8,80e+01 1			
<program root=""></program>		4.70e+	09 100.0	5.82e+09 100 %	8.50e+01 9	6.6%		
▽ 🖹 500: main		4.70e+	09 100.0	5.82e+09 100 %	8.50e+01 9	6.6%		
✓ ➡ 2734: read_and		4.70e+	-09 99.8%	5.82e+09 100 %	9 100 % 6.90e+01 78.4			
✓ ➡ 177: open_archive		3.06e+	09 65.1%	4.32e+09 74.2% 3.70e+01		2.0%		
▼ 🖹 2032: [I] _open_archive		3.06e+	09 65.1%	4.32e+09 74.2% 3.70e+0		2.0%		
▼ 🖹 838: find_nex	t_block	3.06e+	09 65.0%	4.32e+09 74.2%				
		3.06e+	09 65.0%	4.32e+09 74.2%				
		3.06e-	09 65.0%	4.32e+09 74.2%				
マ ➡ 2017: gnu_flush_read		3.06e	Tir	ne blocked i	in the			
▽ 🖹 19	916: [I] _gnu_flush_read	3.06e		nel dominat				
▽	loop at buffer.c: 1880	3.06e						
	► 1879: safe_read	3.06e		omputation		_		
	✓ loop at safe-read.c: 66	3.06e		ociated with	reads			
	▼ ➡ 66: [I] read ▼ ➡ 44:read nocancel		09 65.08	4.32e+09 74.2%				
	I → Har System Cancel I → Har Syst		09 65.0%	4.32e+09 74.2%	.			
	✓ SyS read		-09 32.9%	4.32e+09 74.24				
Kernel	✓ ■ vfs read		-09 32.5%	4.32e+09 74.2%				
frames	✓ IN vfs read		09 28.9%	4.32e+09 74.2%				
Irames	▼ pipe_read		-09 28.0%	4.32e+09 74.2%				
	✓ pipe wait	1.28e+	09 27.2%	4.32e+09 74.2%				

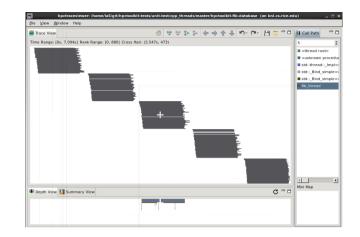
16

A Few More Things

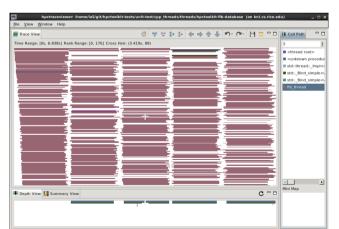
- Events for CPU performance measurement
- Differential performance analysis (useful for CPU and GPU)
- Kernel sampling
- Context recycling for dynamic threads

Context Recycling for Short-lived Threads

- Problem
 - some codes create many shortlived 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



DCA+ using Context Recycling

DCA+ 10 ranks, 12 threads each with context recycling

