ECE 574 – Cluster Computing
Lecture 3

Vince Weaver

http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

29 January 2019

Announcements

- HW#1 will be graded
- HW#2 will be assigned Thursday, will be due after a week
will be posted to website (and I will e-mail)

My Performance Analysis Story

- I started out doing Memory Systems/Computer Architecture
  Mostly caches, but lots of other things, attempting to break Memory Wall.
- In academia you can't really design a new chip to test things; everything done in software simulation
- Ended up 95% of time fighting poorly written simulators
  Not much fun. How do you know if right?
  Validation. How do you validate?

Measure on real hardware. How? Performance counters How about those sims? 20% at best. People reporting stuff lost in noise.

Also cheat a bit. Takes weeks to run benchmark, so they just maybe run first million instructions. Often on simulators for things like Dec Alpha

not repeatable. ”heavily modified sim”, ask to see it and they won’t. Or worse, say they will when they clean it up but never do.

Hardware Performance Counters

- Registers that hold architectural performance counts
- Available on all modern CPUs
- Usually 2-8 of them, often 40-64 bits wide
- Possibly up to 100s of events available
- Have registers you set to enable, start, stop, read value, select event type
- Interface varies arch to arch, vendor to vendor, and even chip revisions
- Other useful thing, hardware interrupt can be triggered

when counter overflows. Why?

If you read infrequently, could miss overflows and be off

Also useful for sampling.

- Pure user events, how can you make sure only belongs to your process?

  Operating system can save/restore registers on context switch

Are counter results accurate?

- See my various papers
- Short answer is ususally, but more obscure might not be
- Intel/AMD also tend to overcount on interrupts
- How would you validate the counters themselves? Exact assembly language program.
- Also chip companies care, but counter correctness is not enough to stop a chip from shipping. They might undocument (or errata) if you report a bug.

Linux Version

- perf_event_open() system call. Really complex, see the manpage.
- Old days was perfctr, then perfmon which required patching kernel.
- Slowly looked like was getting merged, but then out of nowhere Molnar introduced perf_event which got in quickly in 2.6.31 kernel
- Has issues but is mostly good enough these days.

perf tool

- perf tool comes with kernel
- Can be used for doing measurement

PAPI

- Layer of abstraction.
- Want to use counters on all kinds of supercomputers without having to change for each?
- Also provides self-monitoring, can add “calipers” to your code to measure things.

Where Performance Info Comes From

- User Level (instrumentation)
- Kernel Level (kernel metrics)
- Hardware Level (performance counters)

Types of Performance Info

- Aggregate counts – total counts of events that happen
- Profiles – periodic snapshots of program behavior, often providing statistical representations of where program hotspots are
- Traces – detailed logs of program behavior over time

Gathering Aggregate Counts

Measuring runtime – using `time`

```
$ time ./dgemm_naive 200
Will need 1280000 bytes of memory, Iterating 10 times

real        0m7.360s
user        0m7.330s
sys        0m0.000s
```

- Real – wallclock time
- User – time the program is actually running (how calculated)
- Sys – time spent in the kernel

- Must USER+SYS = REAL? Not necessarily (what if other things using the kenrel)
- Can USER be greater than REAL? yes, if multiprocessor
- Is the time command deterministic?
  No. Lots of noise in a system. Can write whole papers on why.
- Which do you use in speedup calculations?

perf tool

```
$ perf stat ./dgemm_naive 200
Will need 1280000 bytes of memory, Iterating 10 times

 Performance counter stats for './dgemm_naive 200':

       7239.152263      task-clock (msec)         #    0.992 CPUs utilized
               116      context-switches          #    0.016 K/sec
                 0      cpu-migrations            #    0.000 K/sec
               357      page-faults               #    0.049 K/sec
     6,513,184,942      cycles                    #    0.900 GHz
   <not supported>      stalled-cycles-frontend
   <not supported>      stalled-cycles-backend
     2,592,685,475      instructions              #    0.40  insns per cyc
        91,797,411      branches                  #   12.681 M/sec
           974,817      branch-misses             #    1.06% of all branch

       7.299463710 seconds time elapsed
```

- Many options. Can select events with `-e`

- Use `perf list` to list all available events

- Hundreds of events available on x86, not quite so many on ARM.

- Understanding the results often requires a certain knowledge of computer architecture.

Profiling

- Records summary information during execution
- Usually Low Overhead
- Implemented via **Sampling** (execution periodically interrupted and measures what is happening) or **Measurement** (extra code inserted to take readings)

Profiling Tools

- Low Overhead – Using hardware counters, such as perf
- Small Overhead – Using static instrumentation, such as gprof
- Large Overhead – Using dynamic binary instrumentation, such as valgrind callgrind

Compiler Profiling

- gprof
- gcc -pg
- Adds code to each function to track time spent in each function.
- Run program, gmon.out created. Run “gprof executable” on it.
- Adds overhead, not necessarily fine-tuned, only does time based measurements.
- Pro: available wherever gcc is.

DBI Profiling

- Valgrind / callgrind tool

Perf Profiling

Automatically interrupts program and takes sample every X instructions.

- `perf record`
- `perf annotate`

Skid

- Beware of “skid” in sampled results
- This is what happens when a complex processor cannot stop immediately, so the reported instruction might be off by a few instructions.
- Some processors do not have this problem. Recent Intel processors have special events that can compensate for this.

Tracing

- When and where events of interest took place
- Shows when/where messages sent/received
- Records information on significant events
- Provides timestamps for events
- Trace files are typically *huge*
- When doing multi-processor or multi-machine tracing, hard to line up timestamps

Performance Data Analysis

Manual Analysis

- Visualization, Interactive Exploration, Statistical Analysis
- Examples: TAU, Vampir

Automatic Analysis

- Try to cope with huge amounts of data by automatic analysis
- Examples: Paradyn, KOJAK, Scalasca, Perf-expert