ECE 574 – Cluster Computing Lecture 3

Vince Weaver

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

vincent.weaver@maine.edu

11am Barrows 133

23 January 2024

Announcements

- HW#1 was graded, you should have gotten an e-mail
- If you did not do HW#1 for any reason (for example, added the class last-minute) let me know, it it still possible to do the assignment. Homeworks are 50% in this class so you don't want to miss any if you can avoid it.
- Weather is a reminder that even complex models backed by large HPC systems are often (always?) wrong

Speedup Review

- Speedup is the improvement in latency (time to run)

$$S = \frac{t_{old}}{t_{new}}$$

  So if originally took 10s, new took 5s, then speedup=2.

- Good metric for serial code

Scalability

- How a workload behaves as more processors are added
- Parallel efficiency: $E_p = \frac{S_p}{p} = \frac{T_s}{pT_p}$
  p=number of processes (threads)
  T_s is execution time of serial code
  T_p is execution time with p processes
- Linear scaling, ideal: $S_p = p$, $E_p = 100\%$
- Real world it's usually less. Why?
- Super-linear scaling – possible but unusual

Strong vs Weak Scaling

- Strong Scaling –for fixed program size, how does adding more processors help

- Weak Scaling – how does adding processors help with the same per-processor workload

Strong Scaling

- Have a problem of a certain size, want it to get done faster.
- Ideally with problem size N, with 2 cores it runs twice as fast as with 1 core (linear speedup)
- *NOTE* can still be some amount of strong scaling even if it's not linear!
- Often processor bound; adding more processing helps, as communication doesn't dominate
- Hard to achieve for large number of nodes, as many

algorithms communication costs get larger the more nodes involved

- Amdahl's Law limits things, as more cores don't help serial code
- Improve by throwing CPUs at the problem.

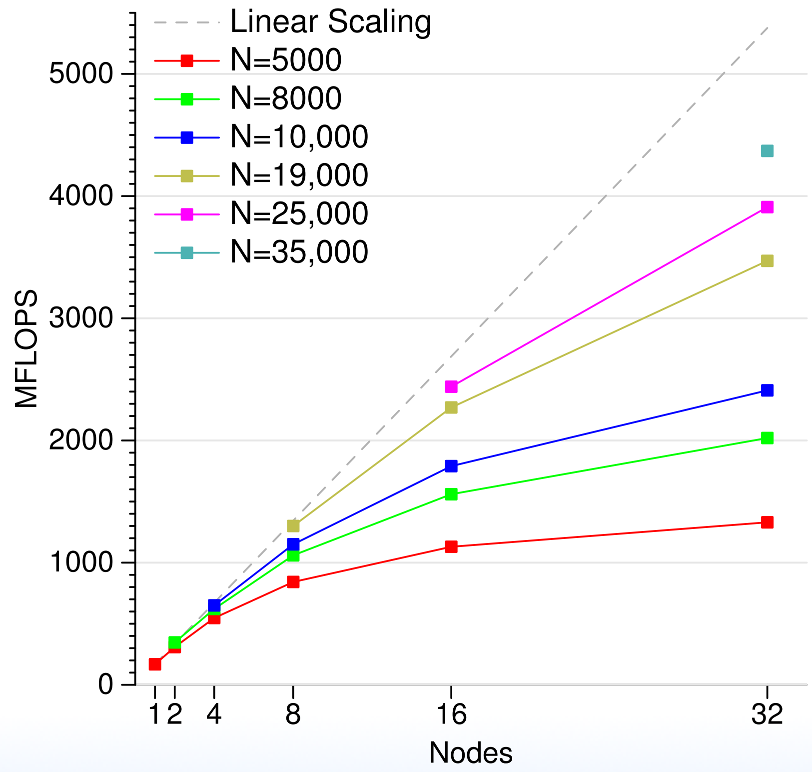
Weak Scaling

- Have a problem, want to increase problem size without slowing down.
- Ideally with problem size N with 1 core, a problem of size 2*N just as fast with 2 cores.
- Often memory or communication bound.
- Gustafson's Law (rough paraphrase)
  No matter how much you parallelize your code, there will be serial sections that just can't be made parallel
- Improve by adding memory, or improving

communication?

Scaling Example

LINPACK on Rasp-pi cluster. What kind of scaling is here?



There is some strong scaling but it quickly fades after 8 nodes.
There is much more prominent weak scaling, in order to approach a linear speedup you have to increase the workload as you add cores.
This is most likely due to the very slow network connections in this particular cluster.

Common Performance Analysis Methods

- Aggregate/Overall Measurements
  - Wall clock time
  - Hardware Performance Counters
- Profiling
- Tracing

Where Performance Info Comes From

- User Level (instrumentation)
  Add timing measurements to own code

- Kernel Level (kernel metrics)
  Kernel tracks metrics on context switch

- Hardware Level (performance counters)
  CPU hardware tracks performance independent of software

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

`time` Corner Cases

- Can REAL be much larger than USER?
  Related: Must USER+SYS = REAL?
  On a heavily loaded multitasking system your program might only get a fraction on the CPU power
- 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?

`time` Related Note on Measurements

- Ideally try to measure on an idle system
- Can turn off various features (ASLR, bind to cores)
- Even then, expect sometimes up to 5% variation run to run
- Ideally take *many* measurements and do things like calculate standard deviation
- Be wary making changes to your code and reporting speedups of under 10% because they might be noise

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
- Will give a demo next class, but you can do something like

  ```
  perf stat ./xhpl
  ```
- Might be disabled by default for security reasons, at least partly it is my fault.

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.

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
- Exterme Overhead – full system simulator

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

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

Using Perf

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.

Perf Profiling

Automatically interrupts program and takes sample every X instructions.

- `perf record`
- `perf report`
- `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.

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