ECE 574 – Cluster Computing
Lecture 4

Vince Weaver

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

vincent.weaver@maine.edu

9 September 2026

Announcements

- HW#1 will be graded soon
- 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.

Notes from Last Time

- Briefly some things from last time
- Forgot to mention student cluster competition at SC
- Note, if you truly have an interesting parallel project as a student can probably find a place willing to run it for you. The ASC would gladly do this in the old days, not sure if new setup will without some sort of account to charge
- H100/A100 GPUs on campus

Introduction to Performance Analysis

What is Performance?

- Getting results as quickly as possible?
- Getting *correct* results as quickly as possible?
- What about Budget?
- What about Development Time?
- What about Hardware Usage?
- What about Power Consumption?

Motivation for HPC Optimization

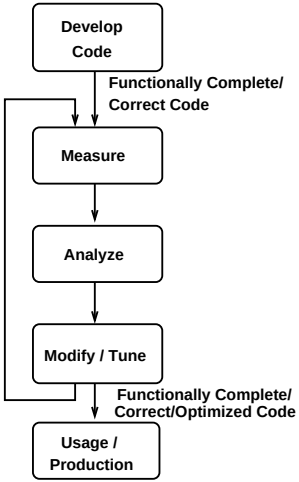
HPC environments are expensive:

- Procurement costs: ∼$40 million
- Operational costs: ∼$5 million/year
- Electricity costs: 1 MW / year ∼$1 million
- Air Conditioning costs: ??

Know Your Limitation

- CPU Constrained
- Memory Constrained (Memory Wall)
- I/O Constrained
- Thermal Constrained
- Energy Constrained

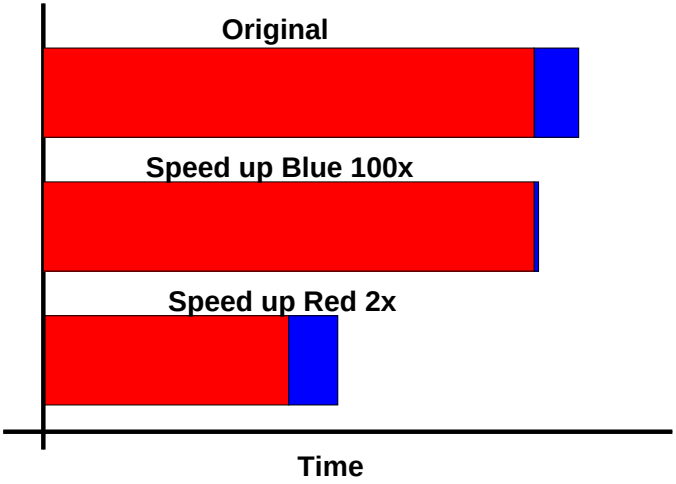
Performance Optimization Cycle



Wisdom from Knuth

"We should forget about small efficiencies, say about 97% of the time:
premature optimization is the root of all evil.
Yet we should not pass up our opportunities in that critical 3%. A good programmer will not be lulled into complacency by such reasoning, he will be wise to look carefully at the critical code; but only after that code has been identified" — Donald Knuth

Amdahl's Law



Speedup

- 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!

Strong Scaling continued

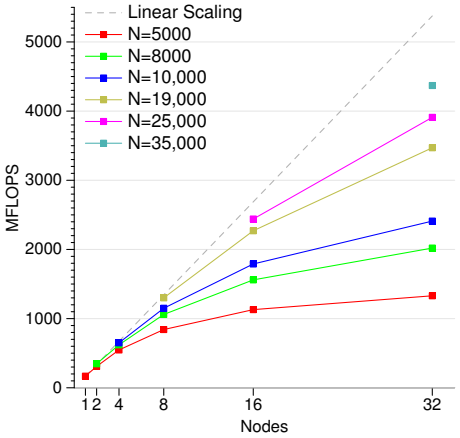
- Improve by throwing CPUs at the problem.
- Often processor bound; adding more processing helps, as communication doesn't dominate
- Hard to achieve for large number of nodes, as many algorithms communication costs get larger the more nodes involved
- Amdahl's Law limits things, as more cores don't help serial code

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 more memory, or faster communication

Scaling Example

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



Scaling Example Explanation

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 time

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

HPCs continued

- 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