Introduction to Software Performance & Tuning

Noah Mendelsohn
Tufts University
Email: noah@cs.tufts.edu
Web: http://www.cs.tufts.edu/~noah
Today

- Why performance matters
- What do we mean by performance?
- Things to understand before you tune your program

Later we will explore tools for measuring performance and techniques for writing high performance programs.
Performance Concepts and Terminology
Performance, Scalability, Availability, Reliability

- **Performance**
  - Get a lot done quickly
  - Preferably at low cost

- **Scalability**
  - Low barriers to growth
  - A scalable system isn’t necessarily fast…
  - …but it can grow without slowing down

- **Availability**
  - Always there when you need it

- **Reliability**
  - Never does the wrong thing, never loses or corrupts data
Throughput vs. Response Time vs. Latency

- **Throughput**: the aggregate rate at which a system does work
- **Response time**: the time to get a response to a request
- **Latency**: time spent waiting (e.g. for disk or network)

We can **improve throughput** by:
- Minimizing work done per request
- Doing enough work at once to keep all hardware resources busy…
- …and when some work is delayed (latency) find other work to do

We can **improve response time** by:
- Minimizing total work and delay (latency) on critical path to a response
- Applying parallel resources *to an individual response*…*including streaming*
- Precomputing response values
Where does the time go?
Know how fast things are

- **CPU Speed:** register – register instruction: ~ 1 cycle (maybe less)
  - In 2014-2016: our machines do perhaps 2 Billion cycles/sec (0.5 nsec / cycle)
  - more /proc/cpuinfo

- **Memory:**
  - L1 Cache hit: a few cycles
  - Memory access (cache miss): 100-200 cycles

- **Hard disk**
  - Depends on where data is on disk
  - Seek time: 5 – 10 ms Rotational delay: 5ms
  - Bandwidth from magentic media: 1 Gbit/sec
  - Streaming: several hundred Mbytes/sec (bandwidth limited)
  - Random access: several msec/seek (latency limited)
  - Watch for contention from other programs or accessing multiple files.

- **SSD**
  - Setup time: 100usec
  - Bandwidth: 2 Gbit/sec (typical) ←note: SSD wins big on latency, some on bandwidth
    Local area network (Gbit Ethernet)
  - Latency: 50-100usec (note microseconds)
  - Bandwidth: 1 Gb/sec (100mbytes/sec)

- **Long distance network**
  - Latency (ping time): 10-100ms
  - Bandwidth: 5 – 100 Mb/sec
Hard disks are slow
Performance Modeling
What is a performance model?

- A model you develop to predict where time will be spent by your program
- To build a model, you need:
  - A clear understanding of how computer hardware performs
  - A clear understanding (or good guess) as to how compilers will translate your programs
  - A high level picture of the flow of your program and where it will spend time
- Your model will also depend on the size and nature of the data your program processes
- When you care about performance, build the best mental model you can, then measure to learn and update your model

The first few times you do this you’ll find that your predictive powers aren’t good...keep at it, and you’ll develop a good intuitive sense of why your programs perform as they do
What is a performance model?

- A model you develop to predict where time will be spent by your program.
- To build a model, you need:
  - A clear understanding of how computer hardware performs
  - A clear understanding (or guess) as to how compilers will translate your programs
  - A high level picture of the flow of your program and where it will spend time
- Your model will also depend on the size and nature of the data your program will process
- When you care about performance, build the best mental model you can, then measure to learn and update your model.

The first few times you do this you’ll find that your predictive powers aren’t good... keep at it, and you’ll develop a good intuitive sense of why your programs perform as they do.

...even programmers who are skilled at modeling and prediction can make the mistake of optimizing prematurely.
Making Systems Faster
Understanding Bottlenecks
“We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.”
Hot spots

The bank robbery model of code tuning:

Q: Mr. Sutton, why do you rob banks?

A: That's where the money is.

Remember: 80% of the time is usually spent in 20% of the code. Tune that 20% only!
Aside: more on Willie Sutton

In his partly ghostwritten autobiography, Where the Money Was: The Memoirs of a Bank Robber (Viking Press, New York, 1976), Sutton dismissed this story, saying:

“The irony of using a bank robber's maxim [...] is compounded, I will now confess, by the fact that I never said it. The credit belongs to some enterprising reporter who apparently felt a need to fill out his copy...

“If anybody had asked me, I'd have probably said it. That's what almost anybody would say...it couldn't be more obvious.

Aside: more on Willie Sutton

In his partly ghostwritten autobiography, Where the Money Was: The Memoirs of a Bank Robber (Viking Press, New York, 1976), Sutton dismissed this story, saying:

“The irony of using a bank robber's maxim [...] is compounded, I will now confess, by the fact that I never said it. The credit belongs to some enterprising reporter who apparently felt a need to fill out his copy...

“If anybody had asked me, I'd have probably said it. That's what almost anybody would say...it couldn't be more obvious.

Or could it?

Why did I rob banks? Because I enjoyed it. I loved it. I was more alive when I was inside a bank, robbing it, than at any other time in my life. I enjoyed everything about it so much that one or two weeks later I'd be out looking for the next job. But to me the money was the chips, that's all.

Go where the money is...and go there often.

Amdahl’s claim: parallel processing won’t scale

1967: Major controversy … will parallel computers work?

“Demonstration is made of the continued validity of the single processor approach and of the weaknesses of the multiple processor approach in terms of application to real problems and their attendant irregularities. Gene Amdahl*”

* Gene Amdahl, *Validity of the single processor approach to achieving large scale computing capabilities*, AFIPS spring joint computer conference, 1967

http://www-inst.eecs.berkeley.edu/~n252/paper/Amdahl.pdf
Amdahl: why no parallel scaling?

“The first characteristic of interest is the fraction of the computational load which is associated with data management housekeeping. This fraction [...might eventually be reduced to 20%...]. The nature of this overhead appears to be sequential so that it is unlikely to be amenable to parallel processing techniques. Overhead alone would then place an upper limit on throughput of five to seven times the sequential processing rate. Gene Amdahl (Ibid)
Amdahl: why no parallel scaling?

“The first characteristic of interest is the fraction of the computational load which is associated with data management housekeeping. This fraction [...might eventually be reduced to 20%...]. The nature of this overhead appears to be sequential so that it is unlikely to be amenable to parallel processing techniques. Overhead alone would then place an upper limit on throughput of five to seven times the sequential processing rate. Gene Amdahl (Ibid)

In short: even if the part you’re optimizing went to zero time, the speedup would be only 5x ... unoptimized code would remain!

\[
\text{Speedup} = \frac{1}{(r_s + (r_p/n))}
\]

where \( r_s \) and \( r_p \) are sequential/parallel fractions of computation

As \( r_p/n \to 0 \), Speedup \( \to 1/r_s \)
Amdahl: why no parallel scaling?

“The first characteristic of interest is the fraction of the computational load which is associated with data management housekeeping. This fraction might eventually be reduced to 20%. The nature of this overhead appears to be sequential so that it is unlikely to be amenable to parallel processing techniques. Overhead alone would then place an upper limit on throughput of five to seven times the sequential processing rate.

Gene Amdahl (Ibid)

In short: even if the part you're optimizing went to zero time, the speedup would be only 5x.

Speedup = $1/(r_s + (r_p/n))$

where $r_s$ and $r_p$ are sequential/parallel fractions of computation

As $r_p/n \to 0$, Speedup $\to 1/r_s$
Don't do premature optimization. Write the program well. Choose a good algorithm. Then *measure* performance, then improve the parts that most affect performance.
Don't do premature optimization. Write the program well. Choose a good algorithm. Then *measure* performance, then *improve* the parts that most affect performance.
Summary

- Understand where the time goes in your program:
  - Model: predict where the time will go
  - Measure then tune

- Understand what takes time in a computer
  - Speed of each component
  - Latency: time spent waiting (e.g. for disk, network or memory)

- Avoid premature optimization
  - Keep most of your program clean
  - Most of the overhead probably comes from a few “hot spots” in your program, tune those

Next: we will explore tools for measuring performance and techniques for writing high performance programs.