Parallel Performance, Speedup and Efficiency

Mark Greenstreet

CpSc 418 – October 3, 2018

Objectives

- Compare and contrast common measures of performance:
  - Latency vs. throughput
  - Wall-clock time vs. operation count
- Evaluate quantitative measures of parallel performance:
  - Speedup.
  - Efficiency.
- Explain common observations about parallel performance
  - Amdahl's and Gustafson's laws: Limitations on parallel performance (and how to evade them).
  - The law of modest returns: High complexity problems are bad, and worse on a parallel machine.
  - Superlinear speed-up: More CPUs means more fast memory, and sometimes you win.
  - Embarrassingly parallel problems: Sometimes you win without even trying.

Measuring Performance

- The main motivation for parallel programming is performance.
  - Time: make a program run faster.
  - Space: allow a program to run with more memory.
- Two common measures of speed:
  - Latency: time from starting a task until it completes.
  - Throughput: the rate at which tasks are completed.
  - Key observation:

$$\begin{aligned} \textit{throughput} &= \frac{1}{\textit{latency}}, \quad \text{sequential programming} \\ \textit{throughput} &\geq \frac{1}{\textit{latency}}, \quad \text{parallel programming} \end{aligned}$$

  - High throughput is achieved through pipelining and/or latency hiding (which often increase latency).

Speed-Up

- Simple definition:

$$SpeedUp = \frac{\text{time(sequential\_execution)}}{\text{time(parallel\_execution)}}$$

- We can also describe speed-up as how many percent faster:

$$\%faster = (SpeedUp - 1) * 100\%$$

- Efficiency is a related measure of what fraction of the P processors are kept busy:

$$Efficiency = \frac{SpeedUp}{P} = \frac{\text{time(sequential\_execution)}}{\text{time(parallel\_execution)}\ P}$$

  - We will focus on speed-up because speed is currently the most common reason for parallelization.
  - Efficiency becomes more important when looking at resources other than time, such as energy or capital cost.

Simple Equation, So Many Interpretations

Simply reporting a speed-up number tells us almost nothing.

- Is “time” latency or (inverse of) throughput?
- How big is the problem? Is the same size used for sequential and parallel version?
- What is the sequential version:
  - The parallel code run on one processor?
  - The fastest possible sequential implementation?
  - Something else?
- *How are we measuring “time”?*

Speed Up – Example

- Let's say that count 3s of a million items takes 10ms on a single processor.
- If I run count 3s with four processes on a four CPU machine, and it takes 3.2ms, what is the speed-up?
- If I run count 3s with 16 processes on a four CPU machine, and it takes 1.8ms, what is the speed-up?
- If I run count 3s with 128 processes on a 32 CPU machine, and it takes 0.28ms, what is the speed-up?

Work and Span

- Describe computation as a graph.
  - Vertices correspond to operations.
    - Which operations should we count?
    - For example, with count 3s, we count the `X==3` test and the adds for the tallies.
    - For parallel count 3s, we also count the send and receive operations.
    - Should we count the details of the recursive calls of the `count3s(List)` function?
    - Equivalently, should we count the operations for setting up and maintaining a loop in an imperative language?
  - Edges represent **dependencies**
    - An edge from V_1 to V_2 if V_2 needs the result from V_1 to perform its operation.
- **Work:** is the total number of vertices. Work corresponds to the sequential execution time.
- **Span:** is the depth of the tree.
  - Span corresponds to the minimum parallel time.
  - Span ignores communication cost – but we could add that by "coloring" vertices.
  - Span still gives us an idea of how parallelizable an algorithm is.

Amdahl's Law

- Given a sequential program where
  - fraction s of the execution time is inherently sequential.
  - fraction $1 - s$ of the execution time benefits perfectly from speed-up.
- The run-time on P processors is:

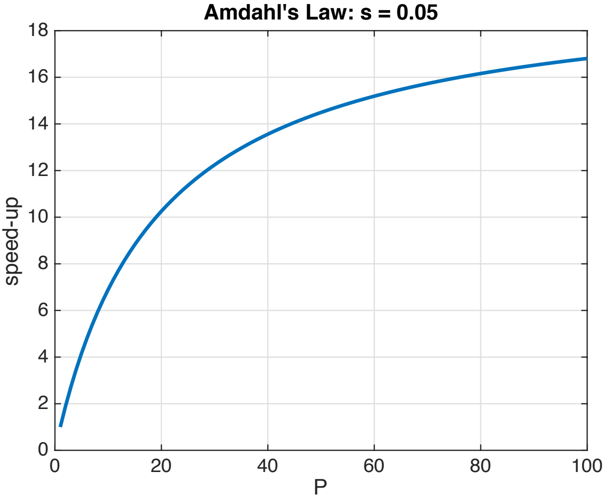
$$T_{parallel} = T_{sequential} * (s + \frac{1-s}{P})$$

- Consequences:
  - Define

$$SpeedUp = \frac{T_{sequential}}{T_{parallel}}$$

  - speed-up on P processors is at most $\frac{1}{s}$.
- Gene Amdahl argued in 1967 that this limit means that parallel computers are only useful for a few special applications where s is very small.

Amdahl's Law



- See also MRR Figure 2.5 (and Figure 2.6 for equivalent efficiency plots).

Amdahl's Law, 50 years later

Amdahl's "law" is a mathematical theorem, not a physical law.

Amdahl's is also an economic observation.

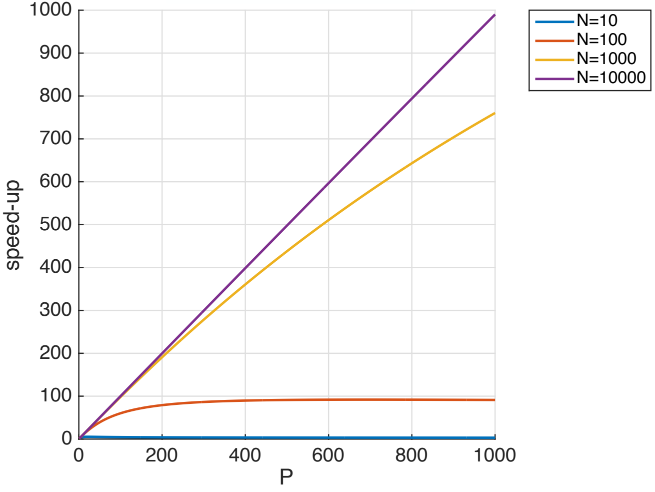
- Amdahl's law was formulated when CPUs were expensive.
- Today, CPUs are cheap!
  - The cost of fabricating eight cores on a die is very little more that the cost of fabricating one.
  - MRR argues that per-core performance grows as $\sqrt{\texttt{\#transistors}}$.
    - Adding cores can be a better use of transistors than trying to improve single processor performance.
- Computer cost is dominated by the rest of the system: memory, disk, network, monitor, ...

Gustafson's Law

Amdahl's law assumes a fixed problem size.

- Gustafson observed in 1988 that when more powerful computers are available, the users solve bigger problems.
  - Many computations have s (sequential fraction) that decreases as N (problem size) increases.
- Examples:
  - Scientific computing.
  - Animation, games and multi-media.
  - Data science and data mining of massive data sets.
- Having lots of cheap CPUs available will
  - Change our ideas of what computations are easy and what are hard.
  - Determine what the next generation of "killer-apps" will be.

Gustafson's Law



- Example: Speedup of a problem where parallel work grows as $N^{3/2}$ and sequential work as $\log P$.

The Law of Modest Returns

More bad news. ☹

- Let's say we have an algorithm with a sequential run-time $T = (12\mathrm{ns})N^4$.
  - If we're willing to wait for one hour for it to run, what's the largest value of N we can use?
  - If we have 10000 machines, and perfect speed-up (i.e. $SpeedUp = 10000$), now what is the largest value of N we can use?
  - What if the run-time is $(5ns)1.2^N$?
- Parallelism offers modest returns unless the problem is of fairly low complexity.
- But:
  - Sometimes, modest returns are good enough: weather forecasting, climate models.
  - Sometimes, problems have huge N and low complexity: data mining, graphics, machine learning.

Super-Linear Speedup

Sometimes, $SpeedUp > P$. ☺

- But if that is true, wouldn't the best sequential algorithm be to simulate P workers by time-sharing a single processor?
  - Probably not: Time-sharing has overhead.
- Memory: a common explanation
  - P machines have more main memory (DRAM)
  - and more cache memory and registers (total)
  - and more I/O bandwidth, . . .
- Multi-threading: another common explanation
  - The sequential algorithm cannot full utilize each CPU's parallel capabilities.
  - A parallel algorithm can make better use through, for example, latency hiding.
- Algorithmic advantages: Some problems are naturally parallel.

BUT: be skeptical, especially if $SpeedUp \gg P$.

Embarrassingly Parallel Problems

Problems that can be solved by a large number of processors with very little communication or coordination.

- Rendering images for computer-animation: each frame is independent of all the others.
- Brute-force searches for cryptography.
- Analyzing large collections of images: astronomy surveys, facial recognition, ...
- Monte-Carlo simulations: same model, run with different random values.
- **Don't be ashamed if your code is embarrassingly parallel:**
  - Embarrassingly parallel problems are great: you can get excellent performance without heroic efforts.
  - The only thing to be embarrassed about is if you don't take advantage of easy parallelism when it is available.

Summary

- Speed-up is sequential time divided by parallel time.
  - Simple definition, can be messy in practice.
  - How do we measure “time” – latency, throughput, throughput under deadline, some other measure?
  - What is the sequential time? Best algorithm? What if the program cannot run on one machine?
- Modeling performance
  - Amdahl’s law: what if some fixed fraction of the computation is non-parallelizable?
  - Gustafson’s law: what if the overhead grows slower that the amount of parallel work as the problem size grows?
  - Work-Span: a graph model that unifies these models.
- Other issues:
  - Super-linear speed-up: usually because more machines have more memory.
  - Embarrassingly parallel problems:
    - Sometimes task are (very nearly) independent.
    - This is great – don’t be ashamed of an embarrassingly parallel problem.
  - The law of modest returns
    - Parallel computing is not a panacea for high computational complexity problems.

Preview

October 3: Parallel Performance		
	PIKA:	pika3 released.
October 5: Performance Loss		
	Reading:	McCool *et al.*, Chapter 2, Sections 2.5 & 2.6.
	PIKA:	pika3 due (1pm).
October 7: HW 2 earlybird (11:59pm).		
October 9: HW 2 due (11:59pm).		
October 10: Parallel Performance: Models		
	Homework:	HW3 released.
October 12: Energy, Power, and Time		
October 15: Sorting Networks		
October 17: The 0-1 Principle		
October 18: HW 3 earlybird (11:59pm).		
October 19: Midterm Review		
	Homework:	HW3 due: **12 noon**.
October 22: Midterm		
October 24-26: Sorting (second half)		
October 29-November 30: Data Parallelism with CUDA		

Review Questions

- What is speed-up? Give an intuitive, English answer and a mathematical formula.
- Why can it be difficult to determine the sequential time for a program when measuring speed-up?
- What is Amdahl's law? Give a mathematical formula. Why is Amdahl's law a concern when developing parallel applications? Why in many cases is it not a show-stopper?
- Is parallelism an effective solution to problems with high big-O complexity? Why or why not?
- What is super-linear speed-up? Describe two causes.
- What is an embarrassingly parallel problem? Give an example.