CSCI 564 Advanced Computer Architecture

Lecture 03: Amdahl's Law

Dr. Bo Wu (with modifications by Sumner Evans)

March 3, 2021

Colorado School of Mines

Amdahl's Law

- Amdahl's Law is the **fundamental theorem of performance optimization**.
- It was made by Amdahl! (One of the designers of the IBM 360.)

The Key Observation

Optimizations do not (generally) uniformly affect the entire program.

- The more widely applicable a technique is, the more valuable it is
- Conversely, limited applicability can (drastically) reduce the impact of the optimization.

Always heed Amdahl's Law!!!

It is central to many many optimization problems.

Amdahl’s Law in Action

- SuperJPEG-O-Rama2010 ISA extensions **
  - Speeds up JPEG decode by 10x!!!
  - Act now! While Supplies Last!

Amdahl's Law in Action

- SuperJPEG-O-Rama2010 ISA extensions **
  - Speeds up JPEG decode by 10x!!!
  - Act now! While Supplies Last!

** SuperJPEG-O-Rama Inc. makes no claims about the usefulness of this software for any purpose whatsoever. It may not even build. It may cause fatigue, blindness, lethargy, malaise, and irritability. Debugging maybe hazardous. It will almost certainly cause ennui. Do not taunt SuperJPEG-O-Rama. Will not, on grounds of principle, decode images of Justin Beiber. Images of Lady Gaga maybe transposed, and meat dresses may be rendered as tofu. Not covered by US export control laws or the Geneva convention, although it probably should be. Beware of dog. Increases processor cost by 45%. Objects in the rear view mirror may appear closer than they are. Or is it farther? Either way, watch out! If you use SuperJPEG-O-Rama, the cake will not be a lie. All your base are belong to 141L. No whining or complaining. Wingeing is allowed, but only in countries where "wingeing" is a word.

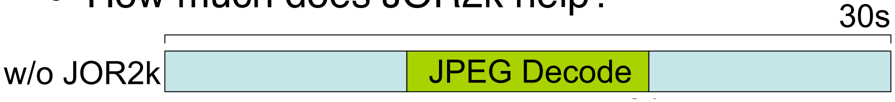
Amdahl's Law in Action

- SuperJPEG-O-Rama2010 ISA extensions **
  - Speeds up JPEG decode by 10x!!!
  - Act now! While Supplies Last!

-Rama. Will not, on grounds of principle, decod
dy Gaga maybe transposed, and meat dresses may b
JS export control laws or the Geneva convention,
e of dog. Increases processor cost by 45%. Objects
er than they are. Or is it farther? Either way, watch
ne cake will not be a lie. All your base are belong to
ingeing is allowed, but only in countries where "wir

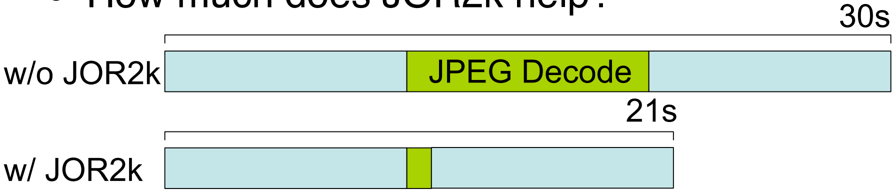
Amdahl's Law in Action

- SuperJPEG-O-Rama2010 in the wild
- PictoBench spends 33% of it's time doing JPEG decode
- How much does JOR2k help?



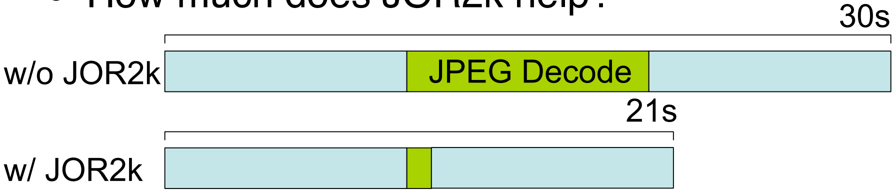
Amdahl's Law in Action

- SuperJPEG-O-Rama2010 in the wild
- PictoBench spends 33% of it's time doing JPEG decode
- How much does JOR2k help?



Amdahl’s Law in Action

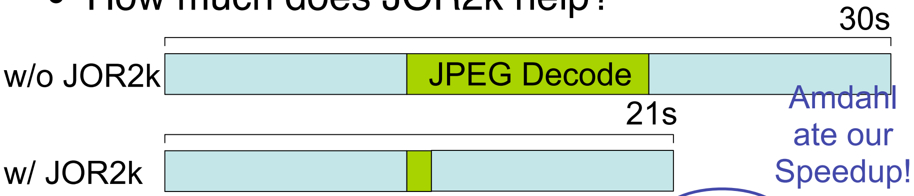
- SuperJPEG-O-Rama2010 in the wild
- PictoBench spends 33% of it’s time doing JPEG decode
- How much does JOR2k help?



Performance: 30/21 = 1.42x Speedup != 10x

Amdahl's Law in Action

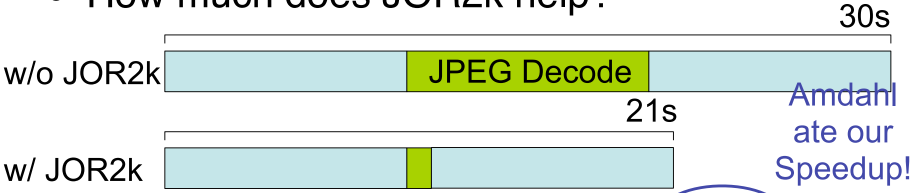
- SuperJPEG-O-Rama2010 in the wild
- PictoBench spends 33% of it's time doing JPEG decode
- How much does JOR2k help?



Performance: 30/21 = 1.42x Speedup != 10x

Amdahl's Law in Action

- SuperJPEG-O-Rama2010 in the wild
- PictoBench spends 33% of it's time doing JPEG decode
- How much does JOR2k help?

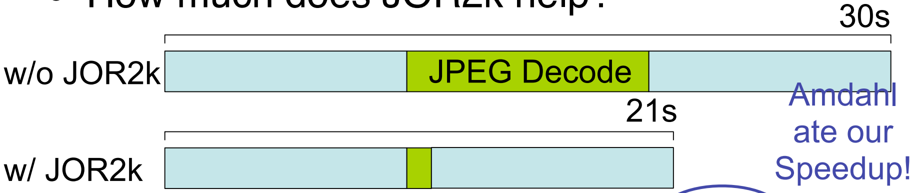


Performance: 30/21 = 1.42x Speedup != 10x

Is this worth the 45% increase in cost?

Amdahl's Law in Action

- SuperJPEG-O-Rama2010 in the wild
- PictoBench spends 33% of it's time doing JPEG decode
- How much does JOR2k help?



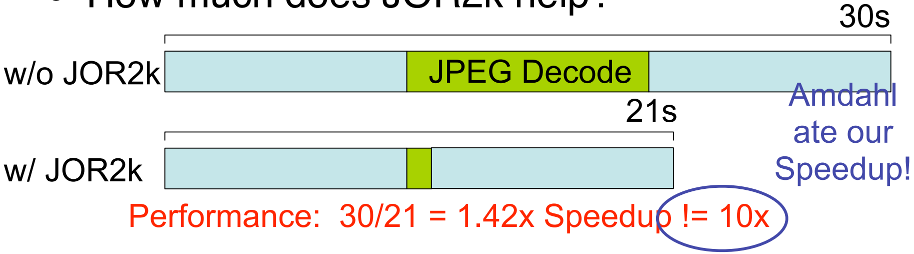
Performance: 30/21 = 1.42x Speedup != 10x

Is this worth the 45% increase in cost?

Metric = Latency * Cost =>

Amdahl's Law in Action

- SuperJPEG-O-Rama2010 in the wild
- PictoBench spends 33% of it's time doing JPEG decode
- How much does JOR2k help?



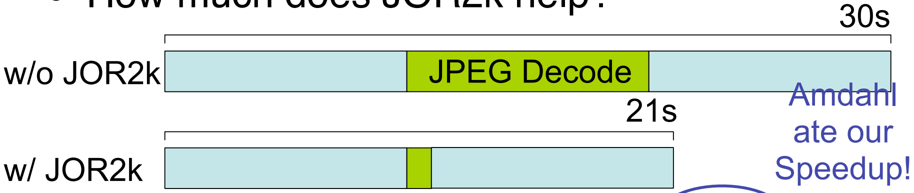
Performance: 30/21 = 1.42x Speedup != 10x

Is this worth the 45% increase in cost?

Metric = Latency * Cost => No

Amdahl's Law in Action

- SuperJPEG-O-Rama2010 in the wild
- PictoBench spends 33% of it's time doing JPEG decode
- How much does JOR2k help?



Performance: 30/21 = 1.42x Speedup != 10x

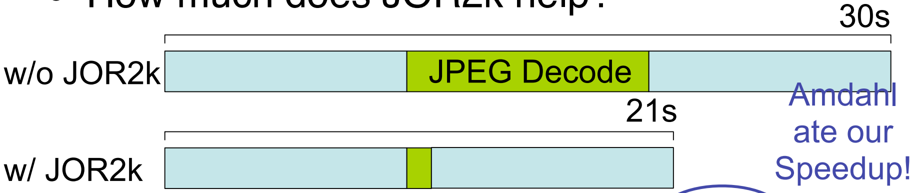
Is this worth the 45% increase in cost?

Metric = Latency * Cost => No

Metric = Latency^2 * Cost =>

Amdahl's Law in Action

- SuperJPEG-O-Rama2010 in the wild
- PictoBench spends 33% of it's time doing JPEG decode
- How much does JOR2k help?



Performance: 30/21 = 1.42x Speedup != 10x

Is this worth the 45% increase in cost?

Metric = Latency * Cost => No

Metric = Latency^2 * Cost => Yes

Explanation

- Latency*Cost and Latency^2*Cost are smaller-is-better metrics.
- Old System: No JOR2k
    - Latency = 30s
    - Cost = C (we don't know exactly, so we assume a constant, C)

Explanation

- Latency*Cost and Latency^2*Cost are smaller-is-better metrics.
- Old System: No JOR2k
  - Latency = 30s
  - Cost = C (we don't know exactly, so we assume a constant, C)
- New System: With JOR2k
  - Latency = 21s
  - Cost = 1.45 * C

Explanation

- Latency*Cost and Latency^2*Cost are smaller-is-better metrics.
- Old System: No JOR2k
  - Latency = 30s
  - Cost = C (we don’t know exactly, so we assume a constant, C)
- New System: With JOR2k
  - Latency = 21s
  - Cost = 1.45 * C
- Latency*Cost
  - Old: 30*C
  - New: 21*1.45*C
  - New/Old = 21*1.45*C/30*C = 1.015
  - New is bigger (worse) than old by 1.015x

Explanation

- Latency*Cost and Latency^2*Cost are smaller-is-better metrics.
- Old System: No JOR2k
  - Latency = 30s
  - Cost = C (we don't know exactly, so we assume a constant, C)
- New System: With JOR2k
  - Latency = 21s
  - Cost = 1.45 * C
- Latency*Cost
  - Old: 30*C
  - New: 21*1.45*C
  - New/Old = 21*1.45*C/30*C = 1.015
  - New is bigger (worse) than old by 1.015x
- Latency^2*Cost
  - Old: 30^2 *C
  - New: 21^2 *1.45*C
  - New/Old = 21^2*1.45*C/30^2*C = 0.71
  - New is smaller (better) than old by 0.71x

Explanation

- Latency*Cost and Latency^2*Cost are smaller-is-better metrics.
- Old System: No JOR2k
  - Latency = 30s
  - Cost = C (we don't know exactly, so we assume a constant, C)
- New System: With JOR2k
  - Latency = 21s
  - Cost = 1.45 * C
- Latency*Cost
  - Old: 30*C
  - New: 21*1.45*C
  - New/Old = 21*1.45*C/30*C = 1.015
  - New is bigger (worse) than old by 1.015x
- Latency^2*Cost
  - Old: 30^2 *C
  - New: 21^2 *1.45*C
  - New/Old = 21^2*1.45*C/30^2*C = 0.71
  - New is smaller (better) than old by 0.71x
- In general, you can make C = 1, and just leave it out.

Amdahl's Law

Amdahl's Law is the **second fundamental theorem of computer architecture**.

Amdahl's Law

If we can speedup x of the program by S times, the total speedup, S_{tot} is given by:

$$S_{tot} = \frac{1}{\left(\frac{x}{S} + (1 - x)\right)}.$$

Sanity check: if $x = 1$ (can speedup entire program)

$$S_{tot} = \frac{1}{\left(\frac{1}{S} + (1 - x)\right)} = \frac{1}{\left(\frac{1}{S}\right)} = S$$

Amdahl's Corollary #1

Maximum Possible Speedup

The maximum possible speedup, S_{max}, from targeting x of the program is given by

$$S_{max} = \frac{1}{1-x}.$$

Why? the maximum possible S is ∞. Take the limit!

$$\lim_{S\to\infty} S_{tot} = \lim_{S\to\infty} \frac{1}{\left(\frac{1}{S} + (1-x)\right)} = \frac{1}{(0 + (1-x))} = \frac{1}{1-x}$$

Practice: Amdahl's Law: Worksheet Problem 1

You have some protein matching code. It completes in 200 hours on your current machine, and spends 20% of the time doing integer operations.

1. How much faster must you make the integer unit to make the code run **10 hours faster**?
2. How much faster must you make the integer unit to make the code run **50 hours faster**?

Practice: Amdahl's Law: Worksheet Problem 2

You have some graph processing code which takes four days to execute on your current machine. Of that time

- 20% of the time is spent performing integer operations, and
- 35% of the time is spent performing I/O operations.

Which of the following is the better tradeoff?

1. A compiler optimization that reduces the number of integer instructions by 25% (assume that each integer operation still takes the same amount of time).
2. A hardware optimization that reduces the latency of each I/O operation from $6\mu s$ to $5\mu s$.

Amdahl's Corollary #2

Make the common case fast (i.e., x should be large)!

- "Start with the largest rocks."
- Common means "most time consuming", not necessarily "most frequent".
- The uncommon case(s) don't make much difference.
- Be sure you know what the common case actually is!
- The common case can change based on inputs, compiler optimizations, optimizations you apply, etc.

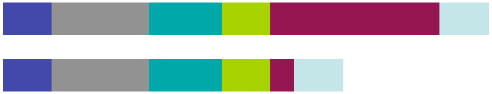
After you've made the common case fast, repeat!

- With optimization, the common becomes uncommon.
- An uncommon case will (hopefully) become the new common case.
- Now target it for optimization!

Amdahl’s Corollary #2: Example

Common case

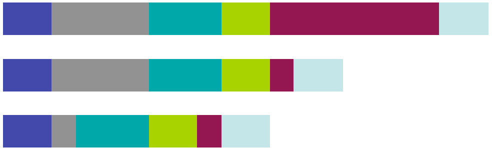
Amdahl's Corollary #2: Example



Common case

🟪 7x => 1.4x

Amdahl’s Corollary #2: Example

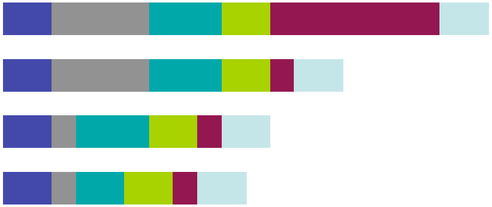


Common case

🟪 7x => 1.4x

⬜ 4x => 1.3x

Amdahl's Corollary #2: Example



Common case

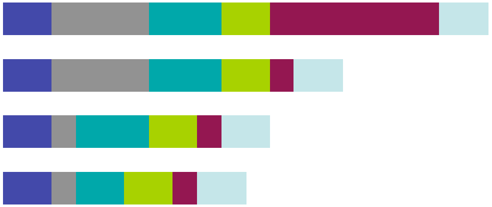
🟪 7x => 1.4x

⬜ 4x => 1.3x

🟦 1.3x => 1.1x

Total = 20/10 = 2x

Amdahl's Corollary #2: Example



Common case

🟥	7x => 1.4x
⬜	4x => 1.3x
🟦	1.3x => 1.1x

Total = 20/10 = 2x

- In the end, there is no common case!
- Options:
  - Global optimizations (faster clock, better compiler)
  - Divide the program up differently
    - e.g. Focus on classes of instructions (maybe memory or FP?), rather than functions.
    - e.g. Focus on function call over heads (which are everywhere).
  - War of attrition
  - Total redesign (You are probably well-prepared for this)

Amdahl's Corollary #3

Amdahl's third corollary bounds the speedup of *parallelizing* a program across p processors.

Maximum Speedup from Parallelization

If x of a program is p-way parallelizable, the maximum possible speedup achievable from parallelizing that part of the application is given by

$$S_{par} = \frac{1}{\frac{x}{p} + (1 - x)}$$

One of the key challenges in parallel programming is increasing x for large p.

- On desktop applications, x is fairly small even for $p = 2$.
- This is why having more cores doesn't necessarily mean you will get better performance!

Practice: Amdahl's Law: Worksheet Problem 3

Recent advances in process technology have quadrupled the number of transistors you can fit on your processor die. Currently, your key customer can use up to 4 processors for 40% of their application.

You have two choices:

1. Increase the number of processors from 1 to 4.
2. Increase the number of processors from 1 to 2, but add features to each processor that will allow the application to use 2 processors for 80% of the execution time.

Which is the best choice?

Amdahl's Corollary #4

By definition:

$$\text{Speedup} = \frac{\text{Latency}_{\text{old}}}{\text{Latency}_{\text{new}}} \Rightarrow \text{Latency}_{\text{new}} = \text{Latency}_{\text{old}} \times \frac{1}{\text{Speedup}}$$

By Amdahl's Law:

$$\text{Latency}_{\text{new}} = \text{Latency}_{\text{old}} \times \frac{1}{\frac{1}{\left(\frac{x}{S}+(1-x)\right)}} = \text{Latency}_{\text{old}} \times \left(\frac{x}{S} + (1-x)\right)$$

Amdahl's Law for Latency

$$\text{Latency}_{\text{new}} = \text{Latency}_{\text{old}} \times \left(\frac{x}{S}\right) + \text{Latency}_{\text{old}} \times (1-x)$$

Amdahl's Non-Corollary

Amdahl's law does not bound slowdown!

$$\mathrm{L}_{\mathrm{new}} = \mathrm{L}_{\mathrm{old}} \times \left(\frac{x}{S}\right) + \mathrm{L}_{\mathrm{old}} \times (1-x) \Rightarrow \boxed{\mathrm{L}_{\mathrm{new}} \propto \frac{x}{S}}$$

Example: $x = 0.01$ of execution. $\mathrm{L}_{old} = 1$.

- $S = 0.001$

$$\mathrm{L}_{new} = \mathrm{L}_{\mathrm{old}} \times (0.01/0.001) + \mathrm{L}_{\mathrm{old}} \times (1-0.01) \approx 10 \times \mathrm{L}_{\mathrm{old}}$$

- $S = 0.00001$

$$\mathrm{L}_{new} = \mathrm{L}_{\mathrm{old}} \times (0.01/0.00001) + \mathrm{L}_{\mathrm{old}} \times (1-0.01) \approx 1000 \times \mathrm{L}_{\mathrm{old}}$$

- Things can only get so fast, but they can get arbitrarily slow.
  - Do not hurt the non-common case too much!

Amdahl's Law: Example

This one is tricky.

In your application, memory operations currently take 30% of the execution time.

A new widget called "cache" speeds up 80% of memory operations by a factor of 4.

A second new widget called "L2 cache" speeds up half of the remaining memory operations by a factor of 2.

What is the total speedup?

Try Applying Each Optimization Independently...

Apply the L1 cache optimization first. $S_{\mathrm{L1}} = 4, x_{\mathrm{L1}} = (0.8)(0.3)$.

$$S_{\mathrm{L1}} = \frac{1}{x_{\mathrm{L1}}/S_{\mathrm{L1}} + (1 - x_{\mathrm{L1}})} = \frac{1}{0.8 \cdot 0.3/4 + (1 - 0.8 \cdot 0.3)} = 1.2195$$

Then apply the L2 cache optimization. $S_{\mathrm{L2}} = 2$,
$x_{\mathrm{L2}} = (0.3)(0.2)(0.5) = 0.03$.

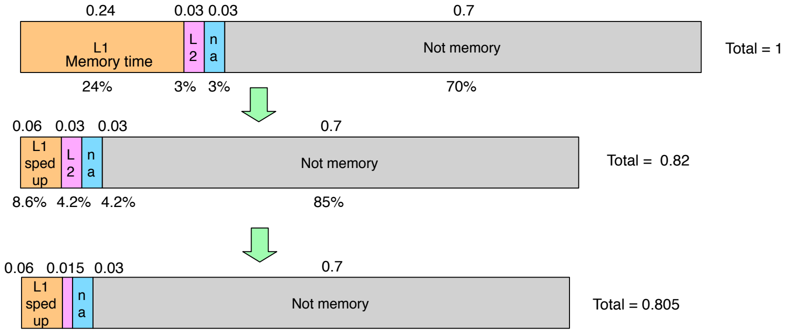
$$S_{\mathrm{L2}} = \frac{1}{x_{\mathrm{L2}}/S_{\mathrm{L2}} + (1 - x_{\mathrm{L2}})} = \frac{1}{0.03/2 + (1 - 0.03)} = 1.0152$$

Combine the speedups:

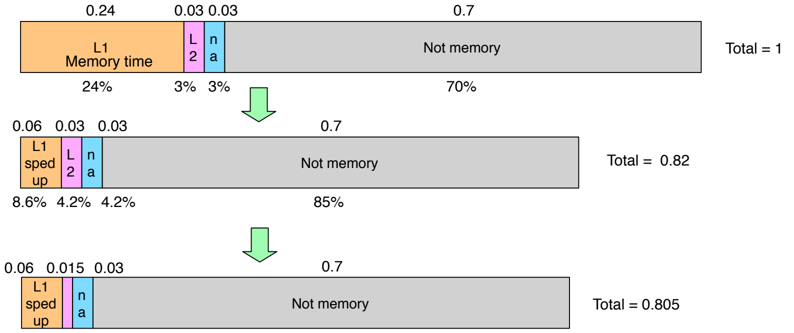
$$S_{\mathrm{tot}} = S_{\mathrm{L1}} \times S_{\mathrm{L2}} = 1.015 \times 1.2195 = 1.2347$$

Is this correct?

Answer in Pictures



Answer in Pictures



OOPS: Speed up = 1.242

Amdahl’s Pitfall

You cannot naïvely apply optimizations one at a time with Amdahl’s Law!

Combining the speedups using multiplication doesn’t work.

Why? After we compute the speedup for one optimization, the execution time changes, so the fraction of execution that the next optimization effects actually grows!

Multiple Optimizations Done Right

Amdahl's Law for Multiple Disjoint Optimizations

Given n **disjoint** optimizations, each of which speed up x_i of the program by S_i, the total speedup is given by

$$S_{\text{tot}} = \frac{1}{\frac{x_1}{S_1} + \frac{x_2}{S_2} + \cdots + \frac{x_i}{S_i} + (1 - x_1 - x_2 - \cdots - x_i)}$$

It is important that the optimizations are disjoint, meaning that for every (x_i, x_j) pair, S_i and S_j must *not* apply to the same portion of the execution.

Practice: Amdahl's Law with Multiple Optimizations: Worksheet Problem 4

In your application, memory operations currently take 30% of the execution time.

A new widget called "cache" speeds up 80% of memory operations by a factor of 4.

A second new widget called "L2 cache" speeds up half of the remaining memory operations by a factor of 2.

What is the total speedup?

Multiple Optimizations Done Right: Overlap

How can we deal with overlapping optimizations? Treat the overlap as a separate portion of the execution and measure its speedup independently.

Example: if we have two overlapping optimizations, then we would have $x_{1\text{only}}$, $x_{2\text{only}}$, and $x_{1\&2}$; and $S_{1\text{only}}$, $S_{2\text{only}}$, and $S_{1\&2}$

$$S_{\text{tot}} = \frac{1}{\frac{x_{1\text{only}}}{S_{1\text{only}}} + \frac{x_{2\text{only}}}{S_{2\text{only}}} + \frac{x_{1\&2}}{S_{1\&2}} + (1 - x_1 - x_2 - x_{1\ \&\ 2})}$$

You can estimate $S_{1\&2} \approx S_{1\text{only}} \times S_{2\text{only}}$, but the real value could be higher or lower.