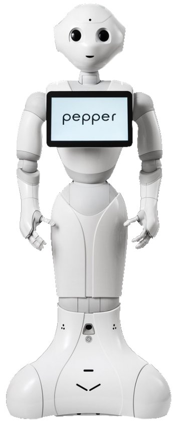
CS302

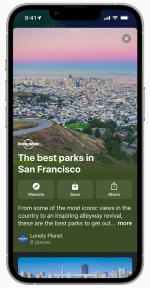
Performance, power and metrics

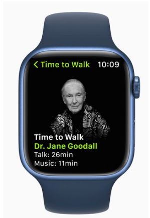


Spring 2025

Arkaprava Basu & Babak Falsafi

parsa.epfl.ch/course-info/cs302





Schedule for this semester

M	T	W	T	F
17-Feb	18-Feb		20-Feb	21-Feb
24-Feb	25-Feb	26-Feb	27-Feb	28-Feb
3-Mar	4-Mar	5-Mar	6-Mar	7-Mar
10-Mar	11-Mar	12-Mar	13-Mar	14-Mar
17-Mar	18-Mar	19-Mar	20-Mar	21-Mar
24-Mar	25-Mar	26-Mar	27-Mar	28-Mar
31-Mar	1-Apr	2-Apr	3-Apr	4-Apr
7-Apr	8-Apr	9-Apr	10-Apr	11-Apr
14-Apr	15-Apr	16-Apr	17-Apr	18-Apr
21-Apr	22-Apr	23-Apr	24-Apr	25-Apr
28-Apr	29-Apr	30-Apr	1-May	2-May
5-May	6-May	7-May	8-May	9-May
12-May	13-May	14-May	15-May	16-May
19-May	20-May	21-May	22-May	23-May
26-May	27-May	28-May	29-May	30-May

- Class intro
  - Performance
  - Power
  - Metrics
- Exercise session
  - Intro and examples of Amdahl’s law
  - Roofline model example
- Next Tuesday
  - Parallel programming

Grading/Regrading [Please note the update]

- Grades (curved)
  - Assignments 30%
  - Homework 20%
  - Midterm 20%
  - Final 30%
- Regrading for up to a week after the grade release
  - Please contact TAs to ask for a regrade

Homeworks & Assignments

- Biweekly (sometimes weekly) homeworks
  - Homework one (non-grade) + solution posted
  - Check Moodle announcements for next homework

- Assignment one will be posted next Monday

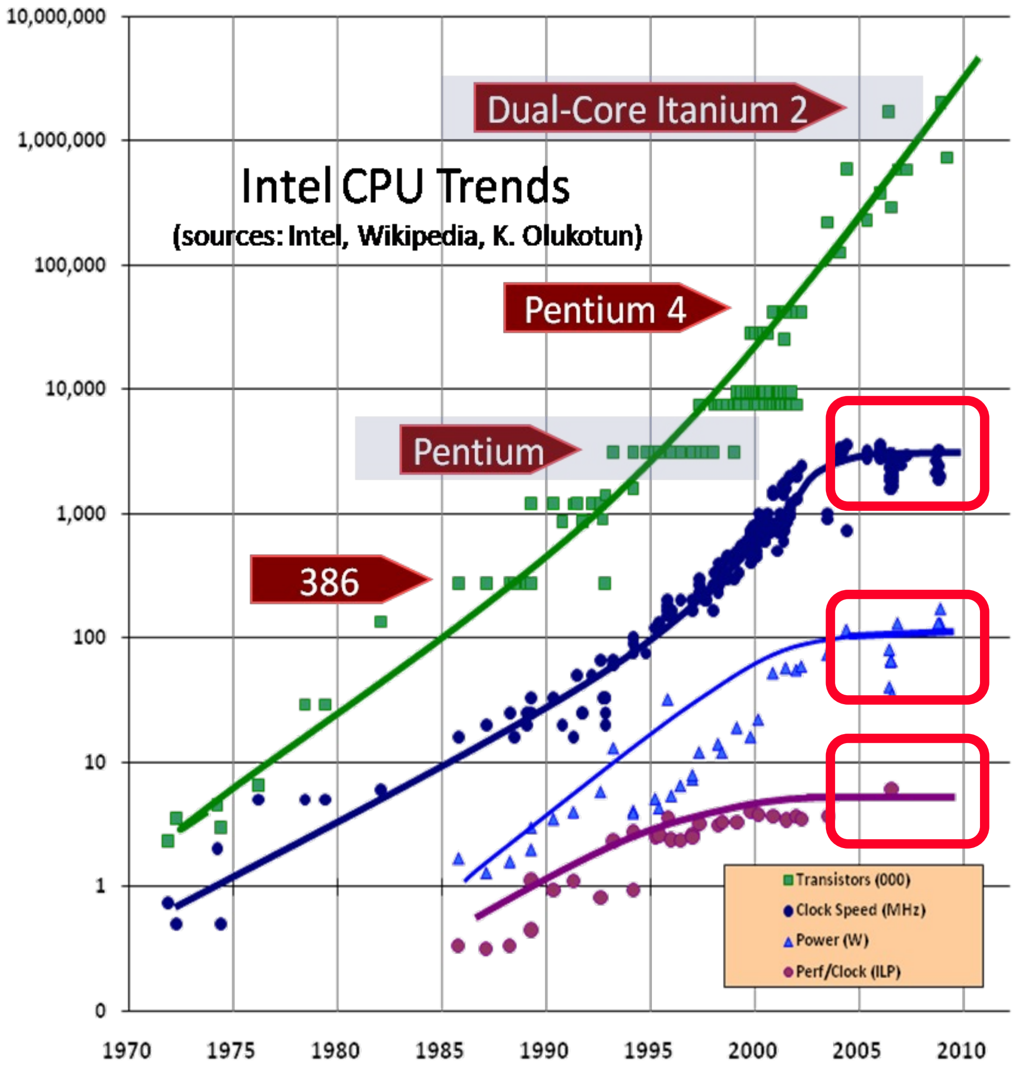
- Two more assignments with the following tentative schedule:
  - March 25
  - May 6

Review: Why Parallelism?

- Ran out of free lunch (c.a. ~2005)
  - Power Wall
    - End of frequency scaling
  - ILP tapped out
    - Little hidden parallelism is left

- Moore's Law reinterpreted
  - Chip density increases slowly
  - Clock speed does not

- Parallelism is a key solution to achieving higher performance



When is parallel computing effective?

- Need performance metrics
  - How fast is a program running on multiple processors?
  - What do we measure to help improve the speed?
- Need cost metrics
  - How much are we paying for a particular speed
- How do we balance cost with performance?

The "Iron Law" of Processor Performance

$$Processor\ Performance =$$

$$\frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

Compiler Architecture Circuit

Instructions/Program

$$Processor\ Performance =$$

$$\frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

Compiler

Instructions/Program

- Compilers
  - # of machine instructions per line of code

- Runtime system
  - Interpreted languages (e.g., Python)
  - Interpretation + dynamic compilation + runtime overhead (e.g., garbage collection)
  - Number of machine instructions per line of application code

- Example: Multiplying two 1000x1000 matrices
  - 50B x86 instructions in C
  - 2.3T x86 instructions in Python (47x higher!)

Cycles/Instruction

$$Processor\ Performance =$$

$$\frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

Compiler Architecture Circuit

Cycles/Instruction

CPI = 1/ (pipeline width) + front-end stalls + back-end stalls

Example:

- 4-way superscalar with no stalls → CPI = 0.25

- Pipeline width: Maximum number of instructions fetch, executed, and retired
- Instruction-level parallelism (ILP): the measure of parallelism in the binary (mix of instructions)
- Memory-level parallelism (MLP): the measure of parallelism available in memory instructions

Cycles/Instruction (Cont.)

- Front-end cycles
  - control-flow hazards: branch predictor, BTB
  - instruction cache
  - Instruction TLB
- Back-end cycles
  - structural hazards: arithmetic units, pipeline buffers
  - data cache
  - data TLB

Cycles/Instruction (Cont.)

- Branch predictors
  - Branch (condition) misprediction rate
  - BTB miss rate
- Cache hierarchies & TLBs
  - Hit latency
  - Miss rate
  - Miss penalty
  - Bandwidth (how many accesses per cycle)
- Memory
  - Latency
  - Bandwidth

Seconds/Cycle

$$Processor\ Performance =$$

$$\frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

Compiler Architecture Circuit

Seconds/Cycle

- Also, known as frequency (F)

- Measure of how fast the circuits are
  - # of gates traversed per clock tick

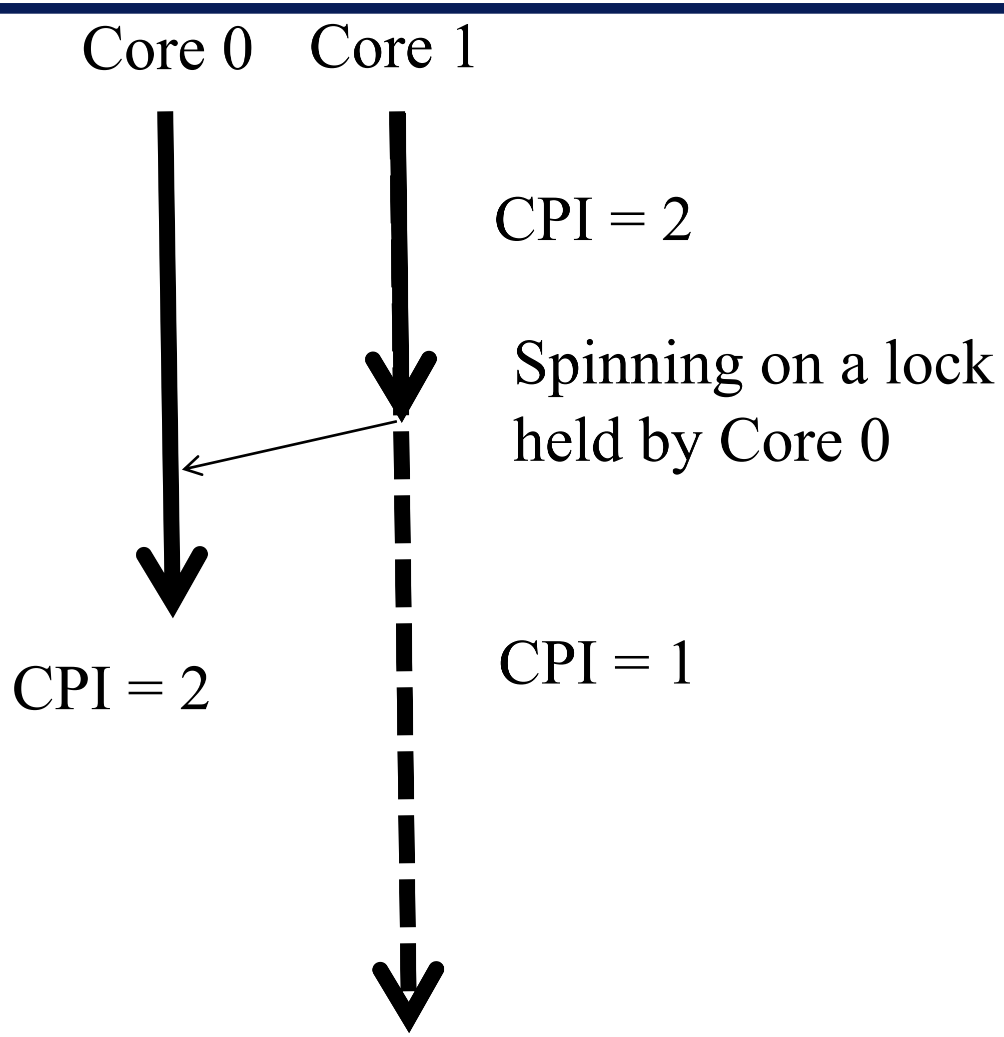
- Dictates power ($P = CV^2F$)

- Frequencies
  - went up from 1970s to 2005 but stopped due to power
  - are going up again since 2019 because there is no other way of improving chip performance

Beware of pitfalls of using CPI in evaluating parallel processors

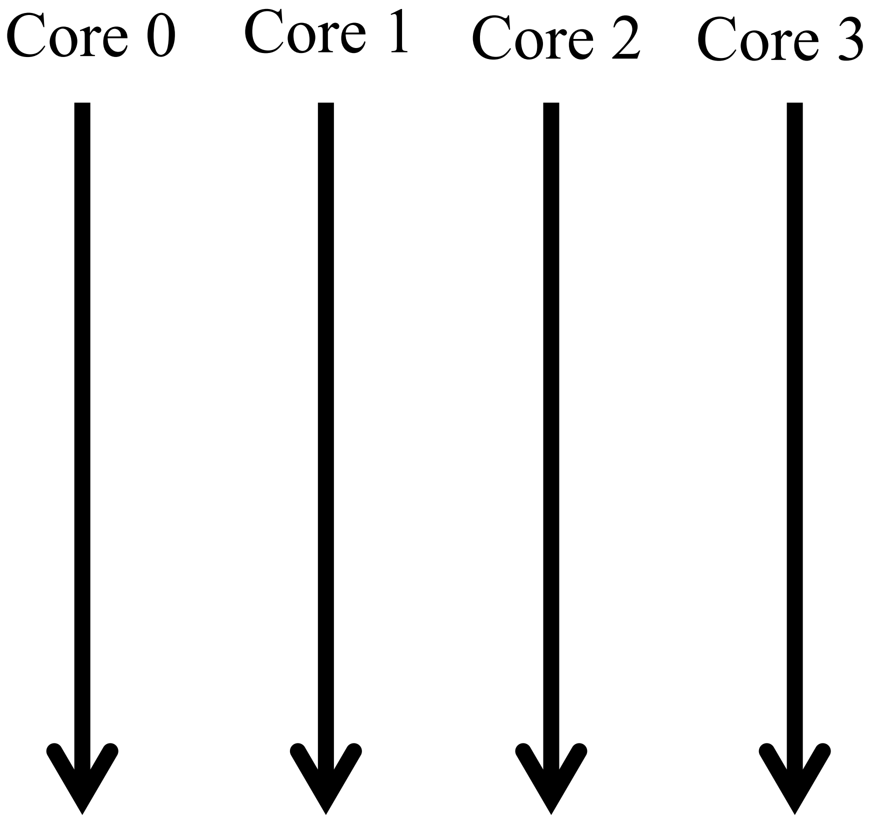
- Uniprocessor: Cycles per instruction (CPI) is a good metric
- Multiprocessor: CPI is not necessarily proportional to perf.
  - Threads often synchronize through spinning
  - Waiting for others to catch up
  - Or allowing access to shared data one at a time
- Spinning threads have low CPI (i.e., better) !!!!
  - Averaging across threads would lead to misleading results
- In the limit, if all threads are waiting all the time
  - Average CPI is really low
  - But not doing useful work
  - Not all instructions make forward progress; User cares about execution time
  - Frequent spins on I/O and locks

Example: CPI may not be useful for Parallel Programs



Example: CPI may not be useful for Parallel Programs

If every thread is doing independent useful work, i.e., not waiting for others then average CPI is fine. But how do you know?



Example: CPI may not be useful for Parallel Programs

- Bottom-line that users see is execution time
- If spinning (useless work) can be counted out, then CPI works
- Need metric for “useful work”
  - Could depend on the program semantics

- Example: Transactions completed per second
  - Widely used in databases
- Example: Requests serviced per second
  - Widely used in webservcies

Review: Latency vs. Bandwidth

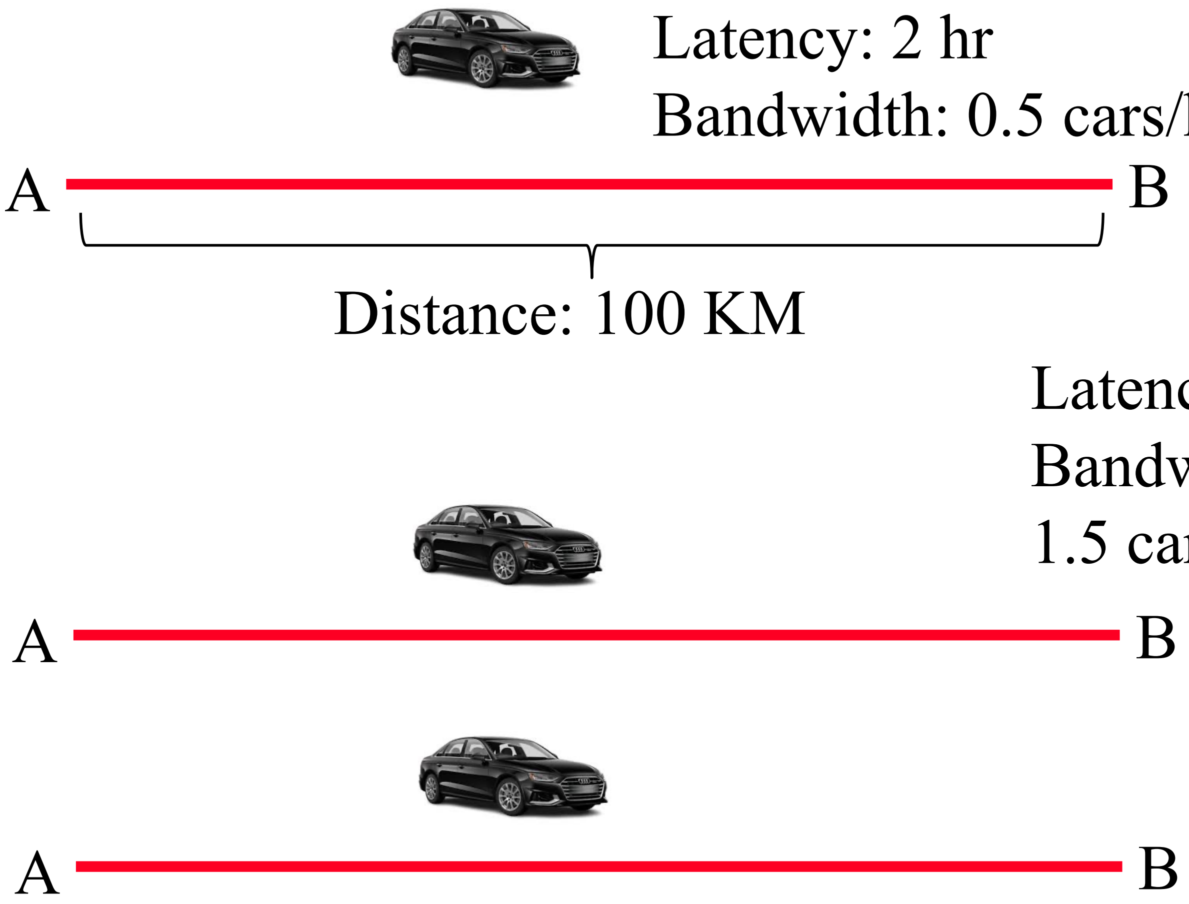
- Latency = time it takes for an event to complete
  - Time to complete execution of a program
  - Memory/disk access time: time for an access to complete
  - Network traversal: time for a message to arrive at the destination

- Bandwidth = how many events per unit time (a throughput metric)
  - Instructions per cycle, Transactions per seconds, Requests per seconds
  - How many bits per second from memory, disk, network
  - How many access per cycle in the cache hierarchy/TLBs (e.g., L1D, L2)

- Reducing latency increases bandwidth but not vice versa

Review: Latency vs. Bandwidth

Speed: 50 KM/hr
Latency: 2 hr
Bandwidth: 0.5 cars/hr



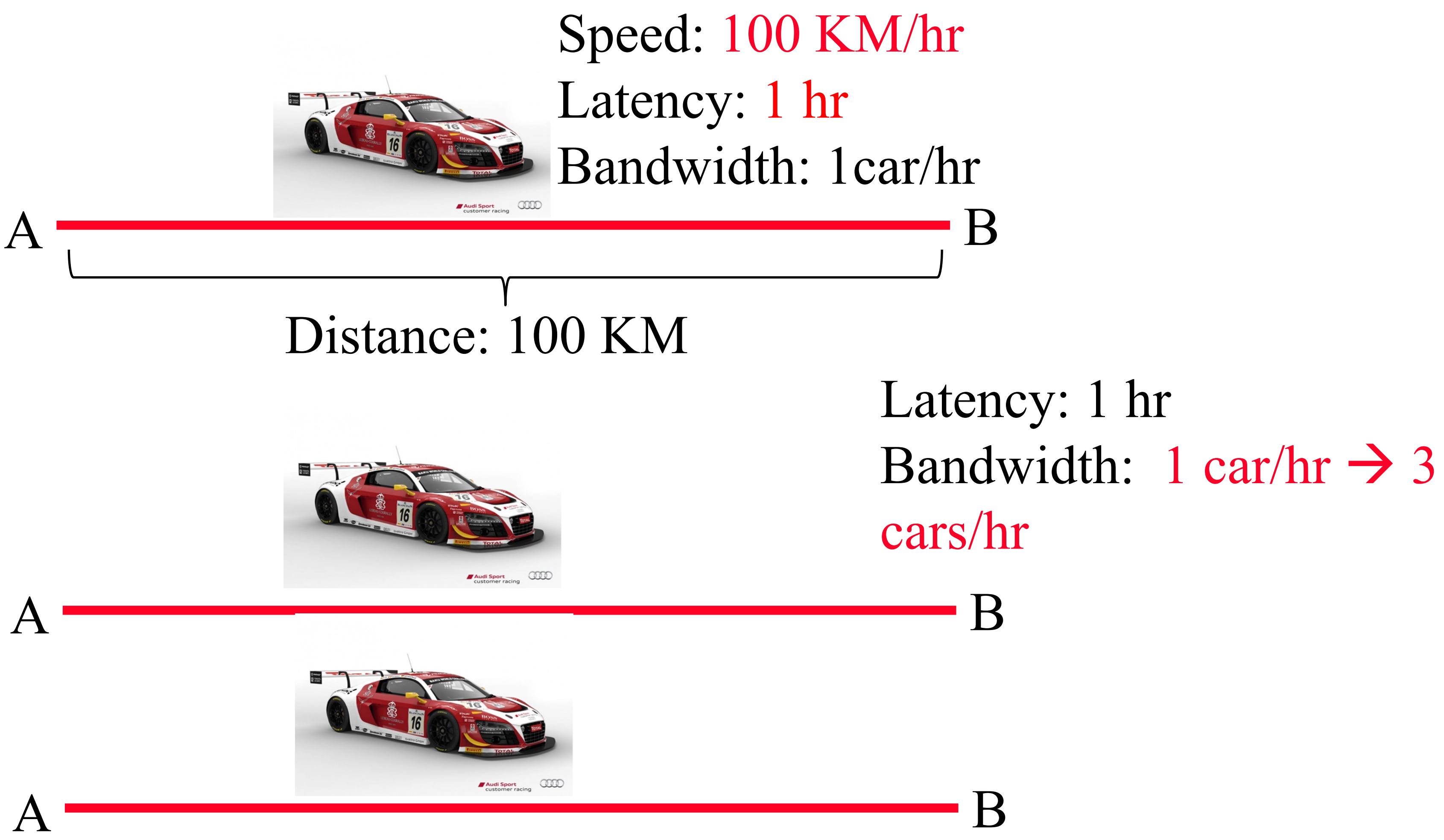
Add two more lanes → Bandwidth improves

Latency: 2 hr
Bandwidth: 0.5 cars/hr → 1.5 cars/hr

Review: Latency vs. Bandwidth

Reducing latency improves bandwidth. But not other way around.

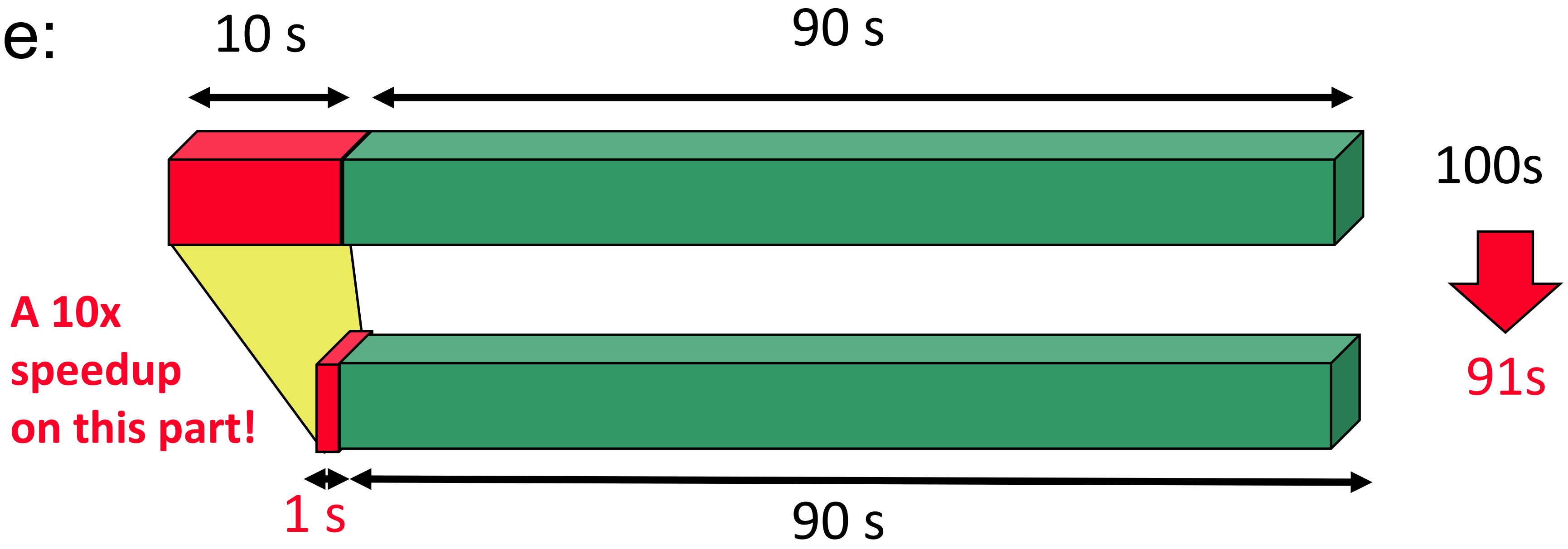
Caveat: Improving bandwidth can sometime reduce latency by reducing congestion (queueing)



Finding Enough Parallelism

- Amdahl's law
  - In English: if you speed up only a small fraction of the execution time of a computation, the speedup you achieve on the whole computation is limited!

- Example:



Amdahl's Law

$$Speedup = \frac{1}{\frac{Fraction_{enhanced}}{Speedup_{enhanced}} + (1 - Fraction_{enhanced})}$$

Example:

Program runs for 100 seconds on a uniprocessor

50% of the program can be parallelized on a multiprocessor.

Assume a multiprocessor with 10 processors:

Amdahl's Law

$$Speedup = \frac{1}{\frac{Fraction_{enhanced}}{Speedup_{enhanced}} + (1 - Fraction_{enhanced})}$$

Example:

Program runs for 100 seconds on a uniprocessor

50% of the program can be parallelized on a multiprocessor.

Assume a multiprocessor with 10 processors:

$$Speedup = \frac{1}{\frac{0.5}{10} + (1 - 0.5)} = \frac{1}{0.05 + 0.5} = \frac{1}{0.55} = 1.82$$

Amdahl's Law

$$Speedup = \frac{1}{\frac{Fraction_{enhanced}}{Speedup_{enhanced}} + (1 - Fraction_{enhanced})}$$

Example:

Assume that 10% of the program cannot be parallelized. What is the maximum achievable speedup?

Amdahl's Law

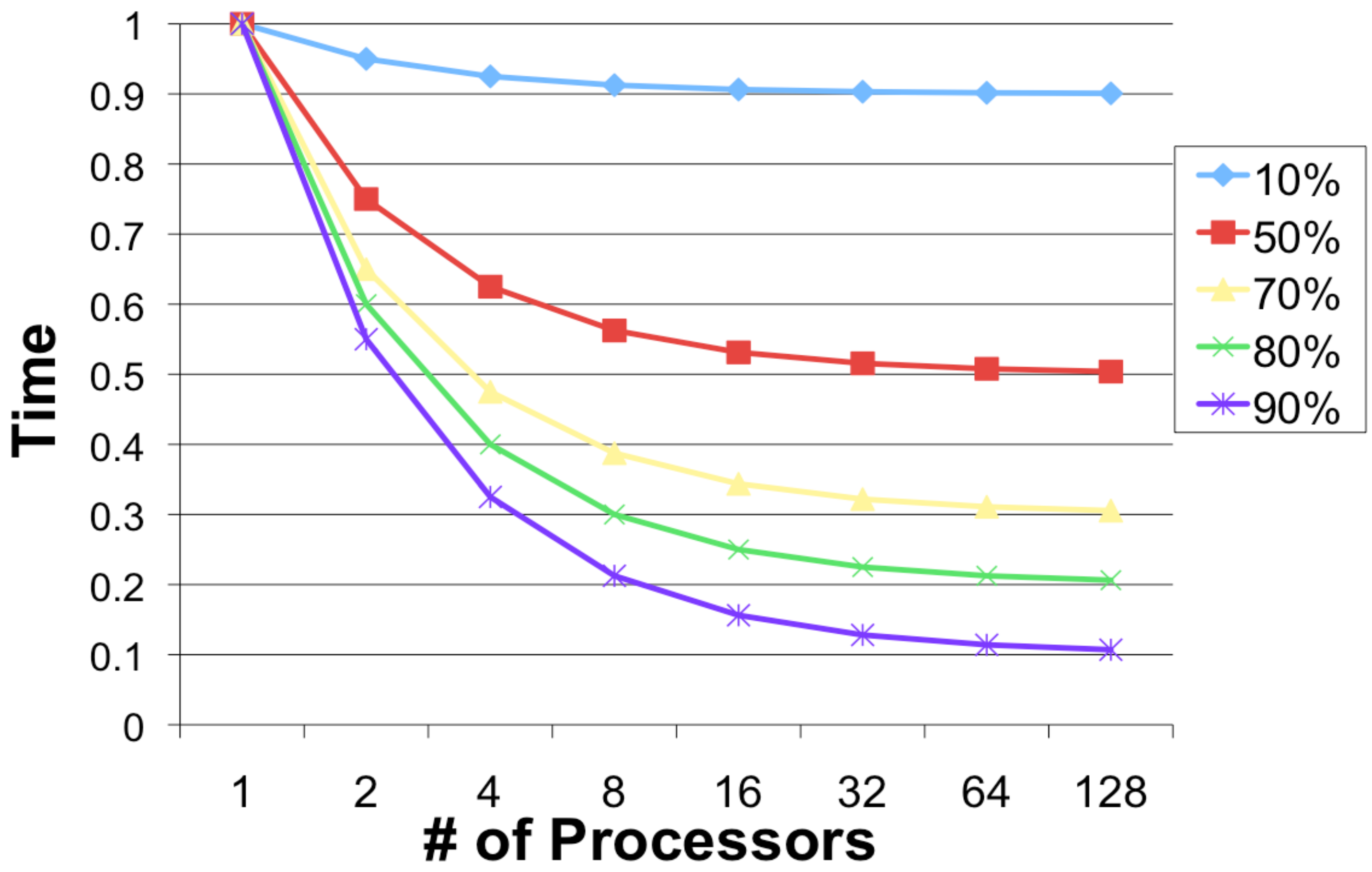
$$Speedup = \frac{1}{\frac{Fraction_{enhanced}}{Speedup_{enhanced}} + (1 - Fraction_{enhanced})}$$

Example:

Assume that 10% of the program cannot be parallelized. What is the maximum achievable speedup?

$$Speedup = \lim_{s \to \infty} \frac{1}{\frac{0.9}{s} + (1 - 0.9)} = \frac{1}{0 + 0.1} = 10$$

Visualize Implications of Amdahl's Law



If an application does not have enough parallelism, using many processors will not help speed it up!

Metrics of Computing and Memory Capabilities

- Processor advertises their compute throughput in max. FLOP/sec
  - Floating Point Operations/sec
  - For example, NVIDIA's H100 GPU has max. compute throughput of 60 teraflops (10^{12})

- Max. compute throughput may not be attainable in practice

- Factors that affect achievable compute throughput:
  - Memory system throughput, expressed in (Giga) bytes/sec
  - For example, NVIDIA H100 GPU's max. memory bandwidth ~ 2TB/sec
  - Program/application characteristics: How much computation is performed for every byte brought from memory?

Roofline Model for Attainable Max. FLOPs/sec

- A simple analytical model and visualization of attainable FLOPs/sec
  - Help identify key system bottlenecks for different
- Operational (Arithmetic) intensity: FLOPs/bytes
  - Property of the program/algorithm
  - For example, sparse matrix-vector multiplication (SpMV) has low operational intensity
  - FFT (Fast Fourier Transformation) has a relatively high operational intensity

```
for (i = 0; i < N; ++i)

    z[i] = x[i]+y[i]
```

Arith. Intensity →

1 Add
3 * 8 bytes of Load/Store
Intensity = 1/24

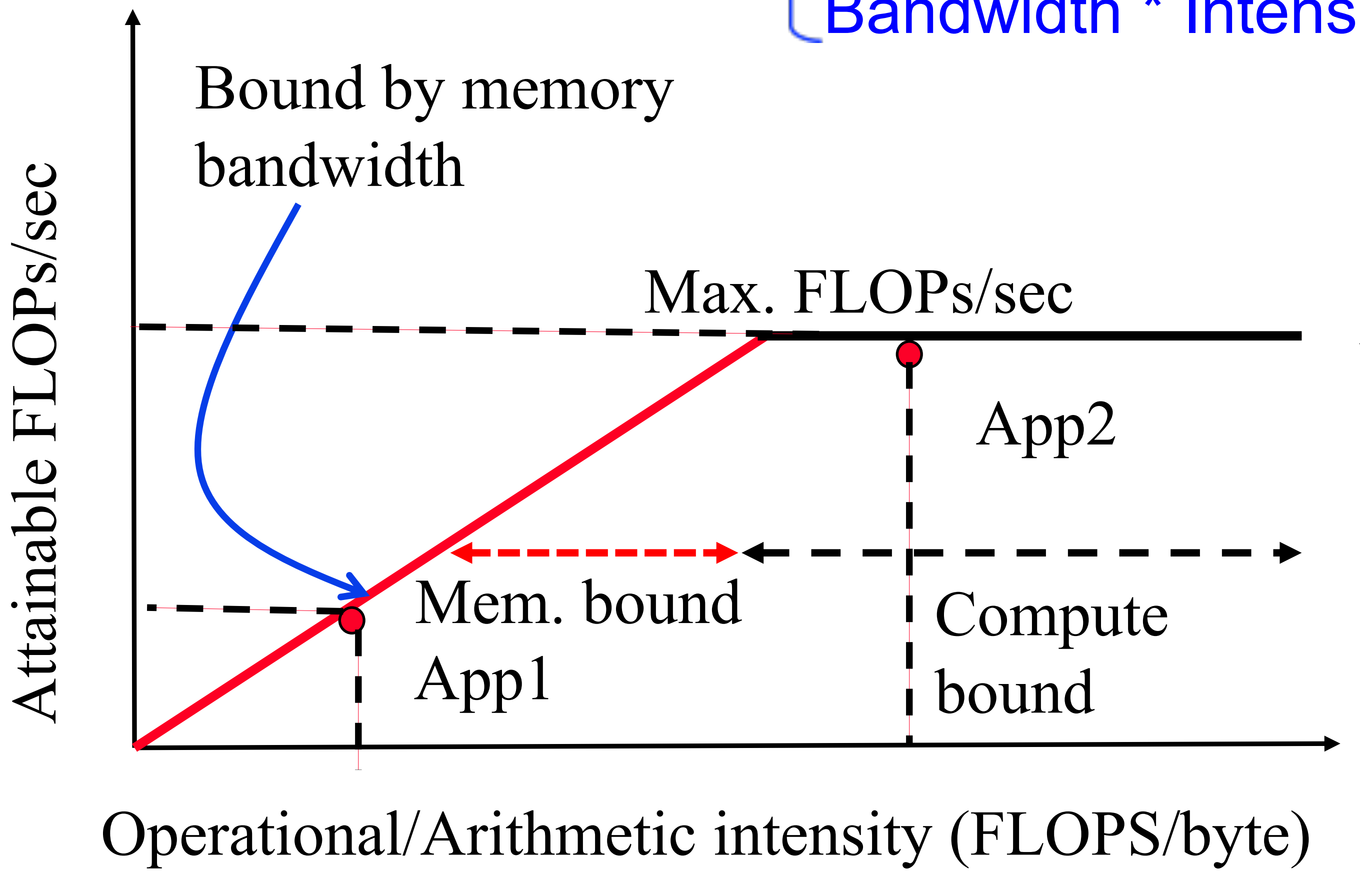
```
for (i = 0; i < N; ++i)
    z[i] = x[i]+y[i]*x[i]
```

Arith. Intensity →

1 Add + 1 Mult
3 * 8 bytes of Load/Store
Intensity = 2/24 = 1/12

Roofline Model for Attainable Max. FLOPs/sec

Performance = **min** { Peak GFLOPS ; Bandwidth * Intensity }



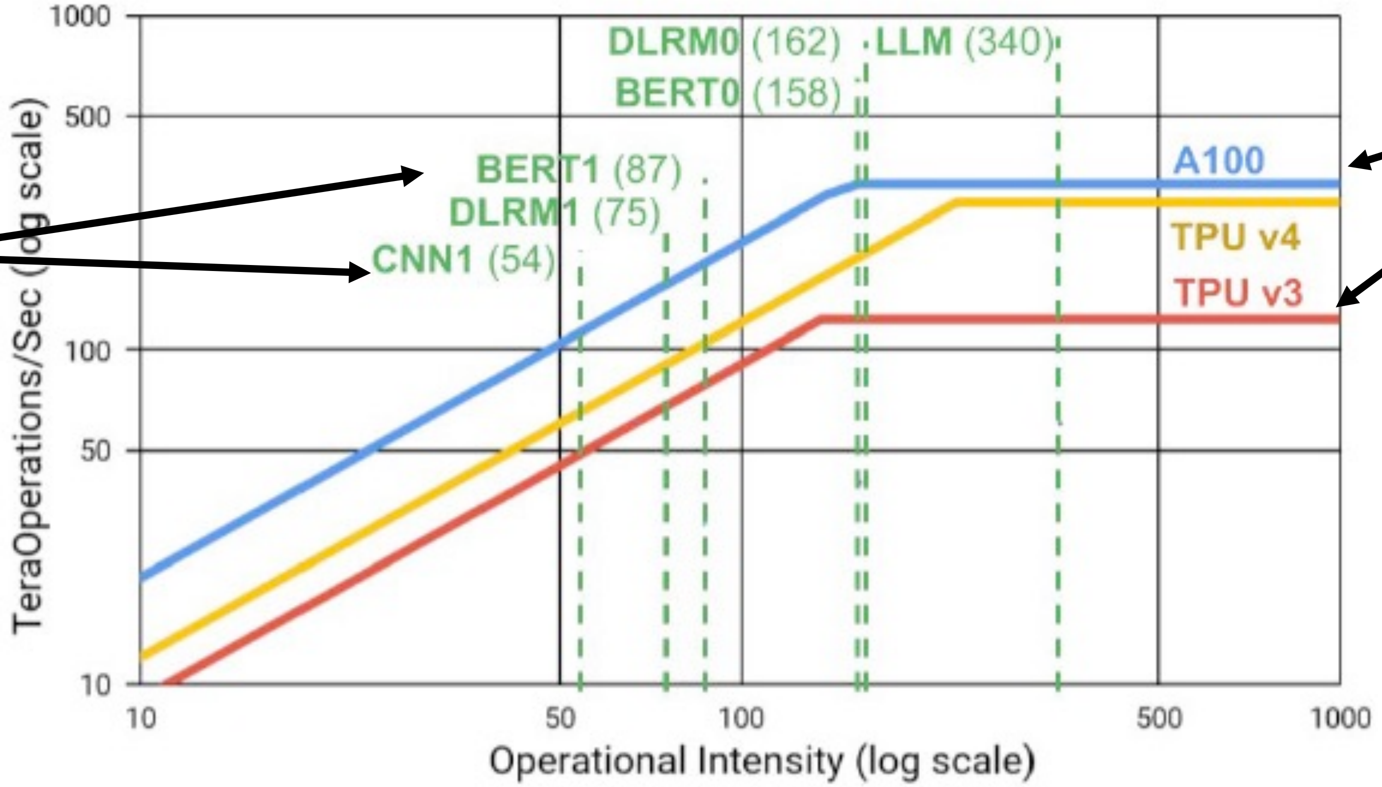
App1 → Memory bandwidth bound

App2 → Compute bound

Practical Usefulness of Roofline

Provides a high-level visualization of potential bottlenecks and optimization opportunities for both systems and applications

Different Applications



Different hardware

Picture from: TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings

Parallel Efficiency

- How much performance do we get from additional cores?

$$Parallel\ Efficiency = \frac{Speedup}{\#\ of\ cores}$$

Power vs. Energy

- Power is measured in Watts
- Power efficiency is measured as Performance/Watt
- Energy is measured in Joules (in smaller platforms) and kWh (in servers)

Cost

- Economic cost:
  - How much does it cost to make a parallel computer (IoT, phone, laptop, server)?
    - 100 CHF (IoT) to 100,000 CHF (GPU server)
  - How much does it cost to operate a parallel computer (electricity)?
    - Residential electricity to commercial electricity cost 35 cents – 1 CHF/kWh

- Environmental cost:
  - Measured in CO_2 or in CO_2-eq (normalized CO_2 across many gasses)
  - Emissions from building a platform
  - Emissions from operating the platform

Cost-Effective Parallel Computing

- What is the incremental cost of a core?
  - Your cell phone has an 8-core CPU?
  - A core (with all additional resources needed including memory, network, I/O) is only a fraction of a server cost

- Speedup (n) = Performance improvement of n cores over 1 core
- Costup (n) = Cost increase of n cores over 1 core

- Computing is cost-effective if Speedup (n) > Costup (n) for a given n

Averaging

- Programs often come various inputs
- What is the average performance for a program?
- How do we average metrics?
  - Example: You drove first 60km at 60km/h and next 60km at 120km/h.
  - What is your average speed?

Arithmetic and Harmonic Mean

- Arithmetic mean:
  - An average of individual times that tracks total execution time

$$\frac{1}{n}\sum_{i=1}^{n} Time_i$$

This is the definition for "average" you are most familiar with

- Harmonic mean:
  - An average of individual rates that tracks total execution time

$$\frac{n}{\sum_{i=1}^{n} \frac{1}{Rate_i}}$$

This is a different definition for "average" you are probably less familiar with

Geometric Mean

- Used for relative rate or normalized performance

$$Relative\_Rate = \frac{Rate}{Rate_{ref}} = \frac{Time_{ref}}{Time}$$

- Geometric mean

$$\sqrt[n]{\prod_{i=1}^{n} Relative\_Rate_i} = \frac{\sqrt[n]{\prod_{i=1}^{n} Rate_i}}{Rate_{ref}}$$

Why does the choice of the mean matter?

Benchmark	FP ops (millions)	Computer 1	Computer 2	Speedup (C2 vs C1)
Absolute performance (Time)				
Program 1	100	1	20	
Program 2	100	1000	20	
Total time		1001	40	25
Arithmetic mean		500	20	25

Why does the choice of the mean matter?

Benchmark	FP ops (millions)	Computer 1	Computer 2	Speedup (C2 vs C1)
Absolute performance (Time)				
Program 1	100	1	20	
Program 2	100	1000	20	
Total time		1001	40	25
Arithmetic mean		500	20	25
Performance in MFLOPS (Rate)				
Program 1		100	5	
Program 2		0.1	5	
Arithmetic mean		50.1	5	0.1
Geometric mean		3.2	5	1.6
Harmonic mean		0.2	5	25

Measuring Metrics

- Practical demo in exercise session on March 6

- Example metrics of interest on real programs
  - Both hardware and software metrics
  - Their impact on program performance

- Tools and techniques to capture several metrics

- Impact of optimizations on the metrics of a parallel program

Summary

- The “Iron Law”: Performance depends on various factors from compiler, architecture to circuit
- Amdahl’s Law captures the impact of the serial portion of a program on speedups from parallelization
- The Roofline model is useful in visualizing attainable FLOPs/sec
- Choosing the “right” average is important in summarizing results