Performance (III)

Hung-Wei Tseng
Recap: von Neumann Architecture

Processor

Program

Memory

Storage
Recap: CPU Performance Equation

**Performance** = \( \frac{1}{\text{Execution Time}} \)

**Execution Time** = \( \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}} \)

\[ ET = IC \times CPI \times CT \]

1 GHz = \( 10^9 Hz = \frac{1}{10^9} \text{ sec per cycle} = 1 \text{ ns per cycle} \)

Frequency (i.e., clock rate)
Recap: Speedup

- The relative performance between two machines, X and Y. Y is \( n \) times faster than X

\[
n = \frac{\text{Execution Time}_X}{\text{Execution Time}_Y}
\]

- The speedup of Y over X

\[
\text{Speedup} = \frac{\text{Execution Time}_X}{\text{Execution Time}_Y}
\]
Recap: Summary of CPU Performance Equation

\[
\text{Performance} = \frac{1}{\text{Execution Time}}
\]

\[
\text{Execution Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}
\]

\[
ET = IC \times CPI \times CT
\]

- IC (Instruction Count)
  - ISA, Compiler, algorithm, programming language, \textit{programmer}
- CPI (Cycles Per Instruction)
  - Machine Implementation, microarchitecture, compiler, application, algorithm, programming language, \textit{programmer}
- Cycle Time (Seconds Per Cycle)
  - Process Technology, microarchitecture, \textit{programmer}
Recap: Amdahl’s Law

\[ Speedup_{\text{enhanced}}(f, s) = \frac{1}{(1-f) + \frac{f}{s}} \]

- \( f \) — The fraction of time in the original program
- \( s \) — The speedup we can achieve on \( f \)

\[ Speedup_{\text{enhanced}} = \frac{\text{Execution Time}_{\text{baseline}}}{\text{Execution Time}_{\text{enhanced}}} \]
Recap: Amdahl’s Law

\[
\text{Speedup}_{\text{enhanced}}(f, s) = \frac{1}{(1 - f) + \frac{f}{s}}
\]

Execution Time_{baseline} = 1

Execution Time_{enhanced} = (1-f) + \frac{f}{s}

\[
\text{Speedup}_{\text{enhanced}} = \frac{\text{Execution Time}_{baseline}}{\text{Execution Time}_{enhanced}} = \frac{1}{(1 - f) + \frac{f}{s}}
\]
Recap: Amdahl’s Law on Multiple Optimizations

- We can apply Amdahl’s law for multiple optimizations
- These optimizations must be dis-joint!
  - If optimization #1 and optimization #2 are dis-joint:

\[
\text{Speedup}_{\text{enhanced}}(f_{\text{Opt1}}, f_{\text{Opt2}}, s_{\text{Opt1}}, s_{\text{Opt2}}) = \frac{1}{(1 - f_{\text{Opt1}} - f_{\text{Opt2}}) + \frac{f_{\text{Opt1}}}{s_{\text{Opt1}}} + \frac{f_{\text{Opt2}}}{s_{\text{Opt2}}}}
\]

- If optimization #1 and optimization #2 are not dis-joint:
Outline

• Amdahl’s Law and Performance
• “Fair” comparison
Amdahl’s Law (cont.)
Practicing Amdahl’s Law (2)

- Final Fantasy XV spends lots of time loading a map — within which period that 95% of the time on the accessing the H.D.D., the rest in the operating system, file system and the I/O protocol. If we replace the H.D.D. with a flash drive, which provides 100x faster access time and a better processor to accelerate the software overhead by 2x. By how much can we speed up the map loading process?
  
  A. ~7x  
  B. ~10x  
  C. ~17x  
  D. ~29x  
  E. ~100x
Practicing Amdahl’s Law (2)

- Final Fantasy XV spends lots of time loading a map — within which period that 95% of the time on the accessing the H.D.D., the rest in the operating system, file system and the I/O protocol. If we replace the H.D.D. with a flash drive, which provides 100x faster access time and a better processor to accelerate the software overhead by 2x. By how much can we speed up the map loading process?
  
  A. ~7x  
  B. ~10x  
  C. ~17x  
  D. ~29x  
  E. ~100x
Final Fantasy XV spends lots of time loading a map — within which period that 95% of the time on the accessing the H.D.D., the rest in the operating system, file system and the I/O protocol. If we replace the H.D.D. with a flash drive, which provides 100x faster access time and a better processor to accelerate the software overhead by 2x. By how much can we speed up the map loading process?

A. ~7x  
B. ~10x  
C. ~17x  
D. ~29x  
E. ~100x

\[ \text{Speedup}_{\text{enhanced}}(95\%, 5\%, 100, 2) = \frac{1}{(1 - 95\% - 5\%) + \frac{95\%}{100} + \frac{5\%}{2}} = 28.98 \times \]
• With the latest flash memory technologies, the system spends 16% of time on accessing the flash, and the software overhead is now 84%. If we want to adopt a new memory technology to replace flash to achieve 2x speedup on loading maps, how much faster the new technology needs to be?

A. \(~5x\)
B. \(~10x\)
C. \(~20x\)
D. \(~100x\)
E. None of the above
Speedup further!

• With the latest flash memory technologies, the system spends 16% of time on accessing the flash, and the software overhead is now 84%. If we want to adopt a new memory technology to replace flash to achieve 2x speedup on loading maps, how much faster the new technology needs to be?

A. ~5x  
B. ~10x  
C. ~20x  
D. ~100x  
E. None of the above
Speedup further!

- With the latest flash memory technologies, the system spends 16% of time on accessing the flash, and the software overhead is now 84%. If we want to adopt a new memory technology to replace flash to achieve 2x speedup on loading maps, how much faster the new technology needs to be?

A. ~5x  
B. ~10x  
C. ~20x  
D. ~100x  
E. None of the above

\[
\text{Speedup}_{\text{enhanced}}(16\%, x) = \frac{1}{(1 - 16\%) + \frac{16\%}{x}} = 2
\]

\[
x = 0.47
\]

Does this make sense?
Amdahl’s Law Corollary #1

• The maximum speedup is bounded by

\[
\text{Speedup}_{\text{max}}(f, \infty) = \frac{1}{(1 - f) + \frac{f}{\infty}}
\]

\[
\text{Speedup}_{\text{max}}(f, \infty) = \frac{1}{(1 - f)}
\]
With the latest flash memory technologies, the system spends 16% of time on accessing the flash, and the software overhead is now 84%. If we want to adopt a new memory technology to replace flash to achieve 2x speedup on loading maps, how much faster the new technology needs to be?

A. ~5x
B. ~10x
C. ~20x
D. ~100x
E. None of the above

\[
\text{Speedup}_{\text{max}}(16\%, \infty) = \frac{1}{(1 - 16\%)} = 1.19
\]

2x is not possible
Corollary #1 on Multiple Optimizations

- If we can pick just one thing to work on/optimize

\[
\begin{align*}
\text{Speedup}_{\text{max}}(f_1, \infty) &= \frac{1}{1 - f_1} \\
\text{Speedup}_{\text{max}}(f_2, \infty) &= \frac{1}{1 - f_2} \\
\text{Speedup}_{\text{max}}(f_3, \infty) &= \frac{1}{1 - f_3} \\
\text{Speedup}_{\text{max}}(f_4, \infty) &= \frac{1}{1 - f_4}
\end{align*}
\]

The biggest \( f_x \) would lead to the largest \( \text{Speedup}_{\text{max}} \)!
Corollary #2 — make the common case fast!

- When f is small, optimizations will have little effect.
- Common == **most time consuming** not necessarily the most frequent
- The uncommon case doesn’t make much difference
- The common case can change based on inputs, compiler options, optimizations you’ve applied, etc.
Identify the most time consuming part

- Compile your program with -pg flag
- Run the program
  - It will generate a gmon.out
  - gprof your_program gmon.out > your_program.prof
- It will give you the profiled result in your_program.prof
If we repeatedly optimizing our design based on Amdahl’s law...

- With optimization, the common becomes uncommon.
- An uncommon case will (hopefully) become the new common case.
- Now you have a new target for optimization.
- You have to revisit “Amdahl’s Law” every time you applied some optimization.
Don’t hurt non-common part too mach

- If the program spend 90% in A, 10% in B. Assume that an optimization can accelerate A by 9x, by hurts B by 10x...
- Assume the original execution time is $T$. The new execution time

$$ET_{new} = \frac{ET_{old} \times 90\%}{9} + ET_{old} \times 10\% \times 10$$

$$ET_{new} = 1.1 \times ET_{old}$$

$$Speedup = \frac{ET_{old}}{ET_{new}} = \frac{ET_{old}}{1.1 \times ET_{old}} = 0.91 \times \ldots \text{slowdown!}$$

You may not use Amdahl’s Law for this case as Amdahl’s Law does NOT
(1) consider overhead
(2) bound to slowdown
Amdahl’s Law on Multicore Architectures

- Symmetric multicore processor with \( n \) cores (if we assume the processor performance scales perfectly)

\[

\text{Speedup}_{\text{parallel}}(f_{\text{parallelizable}}, n) = \frac{1}{(1 - f_{\text{parallelizable}}) + \frac{f_{\text{parallelizable}}}{n}}

\]

Regarding Amdahl’s Law on multicore architectures, how many of the following statements is/are correct?

1. If we have unlimited parallelism, the performance of each parallel piece does not matter as long as the performance slowdown in each piece is bounded
2. With unlimited amount of parallel hardware units, single-core performance does not matter anymore
3. With unlimited amount of parallel hardware units, the maximum speedup will be bounded by the fraction of parallel parts
4. With unlimited amount of parallel hardware units, the effect of scheduling and data exchange overhead is minor

A. 0  
B. 1  
C. 2  
D. 3  
E. 4
Regarding Amdahl’s Law on multicore architectures, how many of the following statements is/are correct?

① If we have unlimited parallelism, the performance of each parallel piece does not matter as long as the performance slowdown in each piece is bounded

② With unlimited amount of parallel hardware units, single-core performance does not matter anymore

③ With unlimited amount of parallel hardware units, the maximum speedup will be bounded by the fraction of parallel parts

④ With unlimited amount of parallel hardware units, the effect of scheduling and data exchange overhead is minor

A. 0  
B. 1  
C. 2  
D. 3  
E. 4
Regarding Amdahl’s Law on multicore architectures, how many of the following statements is/are correct?

① If we have unlimited parallelism, the performance of each parallel piece does not matter as long as the performance slowdown in each piece is bounded

② With unlimited amount of parallel hardware units, single-core performance does not matter anymore

③ With unlimited amount of parallel hardware units, the maximum speedup will be bounded by the fraction of parallel parts

④ With unlimited amount of parallel hardware units, the effect of scheduling and data exchange overhead is minor

A. 0
B. 1
C. 2
D. 3
E. 4
Corollary #3 and Corollary #4

\[
\text{Speedup}_{\text{parallel}}(f_{\text{parallelizable}}, \infty) = \frac{1}{(1 - f_{\text{parallelizable}} + f_{\text{parallelizable}, \infty}}
\]

- Single-core performance still matters
  - It will eventually dominate the performance
  - If we cannot improve single-core performance further, finding more “parallelizable” parts is more important
- If we can build a processor with unlimited parallelism
  - The complexity doesn’t matter as long as the algorithm can utilize all parallelism
  - That’s why bitonic sort or MapReduce works!
- The future trend of software/application design is seeking for more parallelism rather than lower the computational complexity
An Overview of Kepler GK110 and GK210 Architecture

Kepler GK110 was built first and foremost for Tesla, and its goal was to be the highest performing parallel computing microprocessor in the world. GK110 not only greatly exceeds the raw compute horsepower delivered by previous generation GPUs, but it does so efficiently, consuming significantly less power and generating much less heat output.

GK110 and GK210 are both designed to provide fast double precision computing performance to accelerate professional HPC compute workloads; this is a key difference from the NVIDIA Maxwell GPU architecture, which is designed primarily for fast graphics performance and single precision consumer compute tasks.

While the Maxwell architecture performs double precision calculations at a rate of 1/32 that of single precision calculations, the GK110 and GK210 Kepler-based GPUs are capable of performing double precision calculations at a rate of up to 1/3 of single precision compute performance.

Full Kepler GK110 and GK210 implementations include SMX units and basic memory controllers. Different products will use different configurations. For example, some products may deploy 13 or 14 SMXs.

Key features of the architecture that will be discussed below in more depth include:

- The new SMX processor architecture
- An enhanced memory subsystem, offering additional caching capabilities, more bandwidth at each level of the hierarchy, and a fully redesigned and substantially faster DRAM I/O implementation.
- Hardware support throughout the design to enable new programming model capabilities
- GK210 expands on GK110 on-chip resources, doubling the available register file and shared memory capacities per SMX.
Streaming Multiprocessor (SMX) Architecture

The Kepler GK110/GK210 SMX unit features several architectural innovations that make it the most powerful multiprocessor ever built for double precision compute workloads.

SMX: 192 single-precision CUDA cores, 64 double-precision units, 32 special function units (SFU), and 32 load/store units (LD/ST).

Each of these performs the same operation, but each of these is also a “thread”.

A total of 16*12 = 192 cores!
“Fair” Comparisons
TFLOPS (Tera Floating-point Operations Per Second)
## TFLOPS (Tera FLoating-point Operations Per Second)

<table>
<thead>
<tr>
<th>Device</th>
<th>TFLOPS</th>
<th>Clock Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch</td>
<td>1</td>
<td>921 MHz</td>
</tr>
<tr>
<td>XBOX One X</td>
<td>6</td>
<td>1.75 GHz</td>
</tr>
<tr>
<td>PS4 Pro</td>
<td>4</td>
<td>1.6 GHz</td>
</tr>
<tr>
<td>GeForce GTX 2080</td>
<td>14.2</td>
<td>1.95 GHz</td>
</tr>
</tbody>
</table>
Is TFLOPS (Tera FLoating-point Operations Per Second) a good metric?

\[
TFLOPS = \frac{\text{# of floating point instructions} \times 10^{-12}}{\text{Execution Time}}
\]

\[
= \frac{\text{IC} \times \% \text{ of floating point instructions} \times 10^{-12}}{\text{IC} \times \text{CPI} \times \text{CT}}
\]

\[
= \frac{\% \text{ of floating point instructions} \times 10^{-12}}{\text{CPI} \times \text{CT}}
\]

IC is gone!

• Cannot compare different ISA/compiler
  • What if the compiler can generate code with fewer instructions?
  • What if new architecture has more IC but also lower CPI?
• Does not make sense if the application is not floating point intensive
Deep Learning Training in Less Than a Workday

8X Tesla V100

5.1 Hours

15.3 Hours

Time to Solution in Hours—Lower is Better

Server Config: Dual Xeon E5-2695 v4 2.6 GHz | 8X NVIDIA® Tesla® V100 or V100 | ResNet-50 Training on ImageNet Dataset for 90 Epochs with 1.28M ImageNet Dataset.

AI TRAINING

From recognizing speech to training virtual personal assistants and teaching autonomous cars to drive, data scientists are taking on increasingly complex challenges with AI. Solving these kinds of problems requires training deep learning models that are exponentially growing in complexity, in a practical amount of time.

With 640 Tensor Cores, Tesla V100 is the world’s first GPU to break the 100 teraFLOPS (TFLOPS) barrier of deep learning performance. The next generation of NVIDIA NVLink™ connects multiple V100 GPUs at up to 300 GB/s to create the world’s most powerful computing servers. AI models that would consume weeks of computing resources on previous systems can now be trained in a few days. With this dramatic reduction in training time, a whole new world of problems will now be solvable with AI.
NVIDIA® Tesla® V100 is the world’s most advanced data center GPU ever built to accelerate AI, HPC, and graphics. Powered by NVIDIA Volta, the latest GPU architecture, Tesla V100 offers the performance of up to 100 CPUs in a single GPU—enabling data scientists, researchers, and engineers to tackle challenges that were once thought impossible.

### SPECIFICATIONS

<table>
<thead>
<tr>
<th></th>
<th>Tesla V100 PCIe</th>
<th>Tesla V100 SXM2</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU Architecture</td>
<td>NVIDIA Volta</td>
<td></td>
</tr>
<tr>
<td>NVIDIA Tensor Cores</td>
<td>640</td>
<td></td>
</tr>
<tr>
<td>NVIDIA CUDA Cores</td>
<td>5,120</td>
<td></td>
</tr>
<tr>
<td>Double-Precision</td>
<td>7 TFLOPS</td>
<td>7.8 TFLOPS</td>
</tr>
<tr>
<td>Performance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single-Precision</td>
<td>14 TFLOPS</td>
<td>15.7 TFLOPS</td>
</tr>
<tr>
<td>Performance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tensor Performance</td>
<td>112 TFLOPS</td>
<td>125 TFLOPS</td>
</tr>
<tr>
<td>Performance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU Memory</td>
<td>32GB / 16GB HBM2</td>
<td></td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>900GB/sec</td>
<td></td>
</tr>
<tr>
<td>ECC</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Interconnect Bandwidth</td>
<td>32GB/sec</td>
<td>300GB/sec</td>
</tr>
<tr>
<td>System Interface</td>
<td>PCIe Gen3</td>
<td>NVIDIA NVLink</td>
</tr>
<tr>
<td>Form Factor</td>
<td>PCIe Full Height/Length</td>
<td>SXM2</td>
</tr>
</tbody>
</table>

125 TFLOPS Only @ 16-bit floating point

1 GPU Node Replaces Up to 64 GPU Nodes
Node Replacement: HPC Mixed Workload
They try to tell it’s the better AI hardware


<table>
<thead>
<tr>
<th></th>
<th>K80 2012</th>
<th>TPU 2015</th>
<th>P40 2016</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inferences/Sec &lt;10ms latency</td>
<td>$1/_{13}X$</td>
<td>1X</td>
<td>2X</td>
</tr>
<tr>
<td>Training TOPS</td>
<td>6 FP32</td>
<td>NA</td>
<td>12 FP32</td>
</tr>
<tr>
<td>Inference TOPS</td>
<td>6 FP32</td>
<td>90 INT8</td>
<td>48 INT8</td>
</tr>
<tr>
<td>On-chip Memory</td>
<td>16 MB</td>
<td>24 MB</td>
<td>11 MB</td>
</tr>
<tr>
<td>Power</td>
<td>300W</td>
<td>75W</td>
<td>250W</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>320 GB/S</td>
<td>34 GB/S</td>
<td>350 GB/S</td>
</tr>
</tbody>
</table>
What’s wrong with inferences per second?

- There is no standard on how they inference
  - What model?
  - What dataset?
- That’s why Facebook is trying to promote an AI benchmark — MLPerf

---

**Pitfall:** For NN hardware, Inferences Per Second (IPS) is an inaccurate summary performance metric.

Our results show that IPS is a poor overall performance summary for NN hardware, as it’s simply the inverse of the complexity of the typical inference in the application (e.g., the number, size, and type of NN layers). For example, the TPU runs the 4-layer MLP1 at 360,000 IPS but the 89-layer CNN1 at only 4,700 IPS, so TPU IPS vary by 75X! Thus, using IPS as the single-speed summary is even more misleading for NN accelerators than MIPS or FLOPS are for regular processors [23], so IPS should be even more disparaged. To compare NN machines better, we need a benchmark suite written at a high-level to port it to the wide variety of NN architectures. Fathom is a promising new attempt at such a benchmark suite [3].
Extreme Multitasking Performance

- Dual 4K external monitors
- 1080p device display
- 7 applications
What’s missing in this video clip?

• The ISA of the “competitor”
• Clock rate, CPU architecture, cache size, how many cores
• How big the RAM?
• How fast the disk?
Choose the right metric — Latency v.s. Throughput/Bandwidth
Latency v.s. Bandwidth/Throughput

- Latency — the amount of time to finish an operation
  - access time
  - response time
- Throughput — the amount of work can be done within a given period of time
  - bandwidth (MB/Sec, GB/Sec, Mbps, Gbps)
  - IOPs
  - MFLOPs
RAID — Improving throughput

Access time: 10 ms
Bandwidth: 125 MB/sec

Aggregated Bandwidth: 500 MB/sec

MORE SPECS

Model Code (Capacity)

General

DIMENSION (WxHxD)
100 X 22.85 X 6.8 (mm)

TRIM SUPPORT
Yes

ENCRYPTION SUPPORT
AES 256-bit Encryption (Class 0), TCG/Opal
IEEE 1677 (Encrypted Drive)

Performance

SEQUENTIAL READ
Up to 550 MB/s

RANDOM WRITE (4KB, QD=32)
Up to 60,000 IOPS

Environment

AVERAGE POWER CONSUMPTION
(SYSTEM LEVEL)
1,000 GB: Average 2.2 W, Maximum 4.0 W
2,000 GB: Average 3.1 W, Maximum 4.2 W
4,000 GB: Average 3.1 W, Maximum 5.4 W
(Burst mode)
The performance between RAID and SSD

• Compare (X) RAID consists of 4x H.D.D. where each has 10 ms access time and 125 MB/sec bandwidth — aggregated bandwidth at 500 MB/Sec (Y) a single SSD with 100 us access time and 550MB/Sec bandwidth. Both accept 4KB data as the smallest request size. If we want to load a program with 100KB code size, how much faster is Y over X at least?

  A. 1x — no speedup
  B. 1.1x
  C. 4x
  D. 4.4x
  E. 100x
The performance between RAID and SSD

- Compare (X) RAID consists of 4x H.D.D. where each has 10 ms access time and 125 MB/sec bandwidth — aggregated bandwidth at 500 MB/Sec. (Y) a single SSD with 100 us access time and 550 MB/Sec bandwidth. Both accept 4KB data as the smallest request size. If we want to load a program with 100KB code size, how much faster is Y over X at least?

A. 1x — no speedup
B. 1.1x
C. 4x
D. 4.4x
E. 100x
The performance between RAID and SSD

- Compare (X) RAID consists of 4x H.D.D. where each has 10 ms access time and 125 MB/sec bandwidth — aggregated bandwidth at 500 MB/Sec (Y) a single SSD with 100 us access time and 550MB/Sec bandwidth. Both accept 4KB data as the smallest request size. If we want to load a program with 100KB code size, how much faster is Y over X at least?
  A. 1x — no speedup
  B. 1.1x
  C. 4x
  D. 4.4x
  E. 100x

\[
ET_{HDD_{BestCase}} = 10\ ms
\]

\[
ET_{SSD_{worst}} = \frac{100KB}{4K} \times 100\ us = 2.5\ ms
\]
Latency/Delay v.s. Throughput

**Toyota Prius**
- 100 miles (161 km) from UCSD
- 75 MPH on highway!
- Max load: 374 kg = 2,770 hard drives (2TB per drive)

**100 Gb Network**
- 100 miles (161 km) from UCSD
- Lightspeed! — $3 \times 10^8$ m/sec
- Max load: 4 lanes operating at 25GHz

<table>
<thead>
<tr>
<th>Bandwidth</th>
<th>Toyota Prius</th>
<th>100 Gb Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>290GB/sec</td>
<td></td>
<td>100 Gb/s or 12.5GB/sec</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Latency</th>
<th>Toyota Prius</th>
<th>100 Gb Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5 hours</td>
<td>You see nothing in the first 3.5 hours</td>
<td>2 Peta-byte over 167772 seconds = 1.94 Days</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Response Time</th>
<th>Toyota Prius</th>
<th>100 Gb Network</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>You can start watching the movie as soon as you get a frame!</td>
<td></td>
</tr>
</tbody>
</table>
Announcements

• Reading quizzes
  • Due tomorrow

• Resources
  • Ask questions — piazza
  • Reading quizzes, turning in assignments — Canvas
  • Slides, schedule, assignment questions — Check our website
  • Video archive — Prof. Usagi’s Youtube channel
Computer Science & Engineering

Now playing — I'm so tired (Lauv & Troye Sivan)