CS302

Exercise Session 1

Intro to course structure & Amdahl's law examples

A reminder on who's who

- Professors:

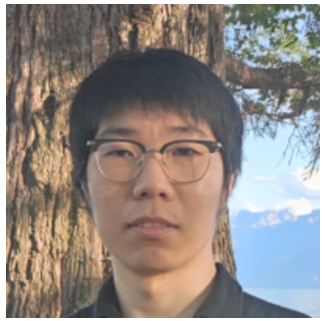

Arka


Babak

- TAs:


Ayan


Pooria


Yuanlong

- SAs:


Patrick


Rayan


Simon


William

A reminder on who's who

- Course offered by PARSA lab @ EPFL IC
- If you are interested in any topics for projects, come talk to us!

What are exercise sessions?

- Held in the same classroom after lecture on Thursdays
- Exercise sessions have three purposes:
  - Practical demo of concepts taught in class
  - Solving example problems
  - Time for you to work on homeworks and assignments
- Will be conducted by either a professor or TA
- TAs and SAs will be around for you to ask for help
  - If there is time remaining after main content of the exercise session
  - Lab sessions are the primary sessions for this purpose

What are lab sessions?

- Held every friday 10:15 am-12:00 pm in CM 011 & CM 012
- First lab session NEXT Friday

- Free time for you to work on assignments and homeworks

- Ask questions about the assignment/homework/course content

Evaluation Components

- Programming assignments (30%)
  - Three assignments: 1x OpenMP, 1x MPI, 1x GPU
- Homeworks (20%)
  - Lecture-based take home exercises
  - Given on Tuesdays, solutions to be submitted by Sunday
  - Total eight exercises over the whole semester
  - 1st homework is a sample (non-graded)
  - Best 6 out of 7 exercises will be used for final grade
  - Usually biweekly, sometimes weekly, will be announced on Moodle
- Midterm (20%) and Final exam (30%)

Course Readings

- Available here:
  - https://parsa.epfl.ch/course-info/cs302/index.php?page=readings.php
  - Will also be posted on moodle
- Taken from textbooks, research papers, online resources
  - Meant to expand material taught in lecture
  - Highly recommended, but not mandatory
- This week:
  - Culler and Singh, Parallel Computer Architecture, Ch.1
- Some readings are lengthy, so you are encouraged to start early

Assignment and Exercise Submissions

- Exercise and assignment solutions to be submitted on Moodle
- Each assignment must be submitted with a report
  - Format will be mentioned on the assignment PDF
- No late submissions or deadline exceptions!
- Take the opportunity to improve your writing skills
  - Not specific to English grammar!
  - Learn how to write precisely
  - No need to write long answers to questions

Groups for Assignments

- Assignments have to be done in groups of two
  - Shares the load while building cooperation/collaboration
- Single student groups are not encouraged
  - Unless you are the last person remaining
  - Your partner leaves the course, and you cannot find anyone else
- Group form available on Moodle already
  - Please fill before next Monday
  - People who have not filled the form will be paired randomly
- Notify us if your partner leaves the course

Examples of Amdahl’s Law

Problem 1

- Assume a program takes 100 secs to run on a single core
- The program has two parts:
  - Serial part (10 secs)
  - Parallel part (90 secs)
- What is the speedup when the program is executed on 5 cores?

Problem 1

- Serial part (10 secs) remains unchanged
- Parallel part can now be distributed amongst 5 cores
  - New execution time = (90 / 5) secs = 18 secs
- So, new total execution time = 10 + 18 secs = 28 secs
- Speedup = 100 secs / 28 secs = 3.5x

Problem 2

- For the same problem as before, how many cores would you need for 5x speedup?

Problem 2

- For the same problem as before, how many cores would you need for 5x speedup?

- Old execution time = 100 secs
- New execution time = Old execution time / speedup
  = 100 / 5 = 20 secs
- Serial phase takes 10 secs (remains unchanged)
- Time for parallel phase = (20 – 10) secs = 10 secs

Problem 2

- For the same problem as before, how many cores would you need for 5x speedup?

- Time for parallel phase = 10 secs
- So, number of cores required = 90/10 = 9

Problem 3

- For the same problem as before, what is the maximum speedup you can get?

Problem 3

- For the same problem as before, what is the maximum speedup you can get?

- Again, serial phase remains unchanged (10 secs)
- Maximum speedup occurs with infinite cores
  - Parallel phase takes (90 / infinity = 0 secs) to finish

- Maximum speedup = 100 / 10 = 10x

Problem 4

- Assume you now run your program on a heterogeneous system
  - 2x high performance cores
  - 8x normal cores (same as in previous problems)
- Each high performance core is 2x as powerful as a normal core
- What is the maximum speedup you can get from this system?

Problem 4

- Serial phase can now run on a high performance core
  - Note: it can only use one of the two high performance cores

- High performance core is 2x faster
- So, new serial phase execution time = 10 / 2 = 5 secs

Problem 4

- Parallel phase can now run on both high performance and normal cores

- 2x high performance core and 8x normal cores
- Speedup factor = 8 + 2*2 = 12
- New execution time of parallel phase = 90 / 12 = 7.5 secs

Problem 4

- Total execution time = 5 + 7.5 secs = 12.5 secs
- Overall speedup of the program = 100 / 12.5 secs = 8x

Problem 5

- Amdahl's law is not limited to serial and parallel phases
- Can also be used to evaluate tradeoffs and oppurtinities

- Assume you have a program with:
  - 500 memory instructions
  - 1000 arithmetic instructions

- Assume memory instructions take 5 clock cycle and arithmetic instructions take 2 clock cycles to execute

Problem 5

- Assume you have a program with:
  - 500 memory instructions
  - 1000 arithmetic instructions
- Assume memory instructions take 6 clock cycle and arithmetic instructions take 2 clock cycles to execute
- You want to design a new core such that it can either:
  - Speedup memory instructions by 2x
  - Speedup arithmetic instructions by 2x
  - Which one do you choose?

Problem 5

- Memory instructions take (500 * 6) = 3000 clock cycles
- Arithmetic instructions take (1000 * 2) = 2000 clock cycles
- Total time = 5000 clock cycles

- Option 1: Speedup memory instructions by 2x
  - New time = 3000 / 2 + 2000 = 3500 clock cycles

- Option 2: Speedup arithmetic instructions by 2x
  - New time = 3000 + 2000 / 2 = 4000 clock cycles

- Option 1 is better!

Problem 6

- Consider the same program as before:
  - Serial part (10 secs)
  - Parallel part (90 secs)

- What is its parallel efficiency?

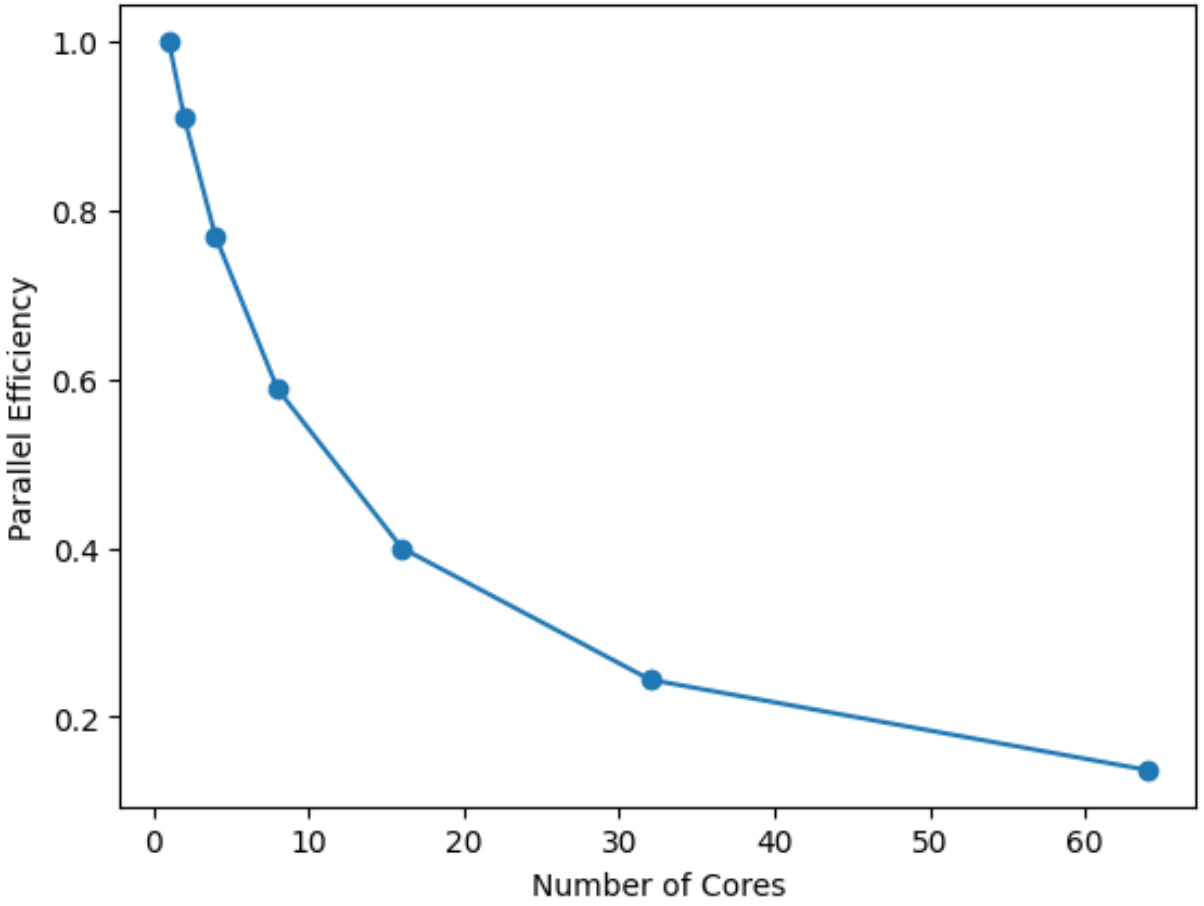
Problem 6

- What is its parallel efficiency?

- Speedup on n cores = $\frac{100}{\left(\frac{90}{n}+10\right)} = \frac{1}{0.1+\frac{0.9}{n}}$
- Parallel efficiency = Speedup / Number of cores
- Parallel efficiency = $\frac{\frac{1}{0.1+\frac{0.9}{n}}}{n} = \frac{1}{n(0.1+\frac{0.9}{n})} = \frac{1}{0.1n+0.9}$

Problem 6

- What is its parallel efficiency?



Problem 6

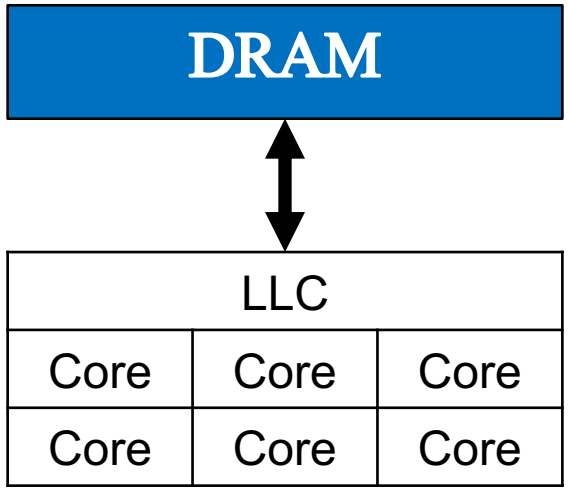
- Parallel efficiency < 1 for all number of cores > 1
- This is true for many programs, not just our toy program

- Does this mean we should not build multi-core CPUs?

Problem 7

- Calculate the costup as a function of number of cores

- Consider a CPU with 32MB LLC (~17mm^2) and 4GB DRAM
- A single core is 1.4mm^2
- Example CPU with 6 cores:



Problem 7

- Calculate the costup as a function of number of cores

- Consider a CPU with 32MB LLC (~17mm^2) and 4GB DRAM
- A single core is 1.4mm^2

- Assume, cost of fabrication: 1 CHF/mm^2
- Assume, cost of DRAM: 5 CHF/GB
- Assume that LLC and DRAM is fixed for all core counts
  - Not always realistic but assume for this toy example

Problem 7

- Cost of building a CPU with n cores = Cost of DRAM + Cost of chip

- Cost of DRAM = 4 * 5 CHF = 20 CHF
- Cost of LLC = 17 * 1 CHF = 17 CHF
- Cost of 1 core = 1.4 * 1 CHF = 1.4 CHF

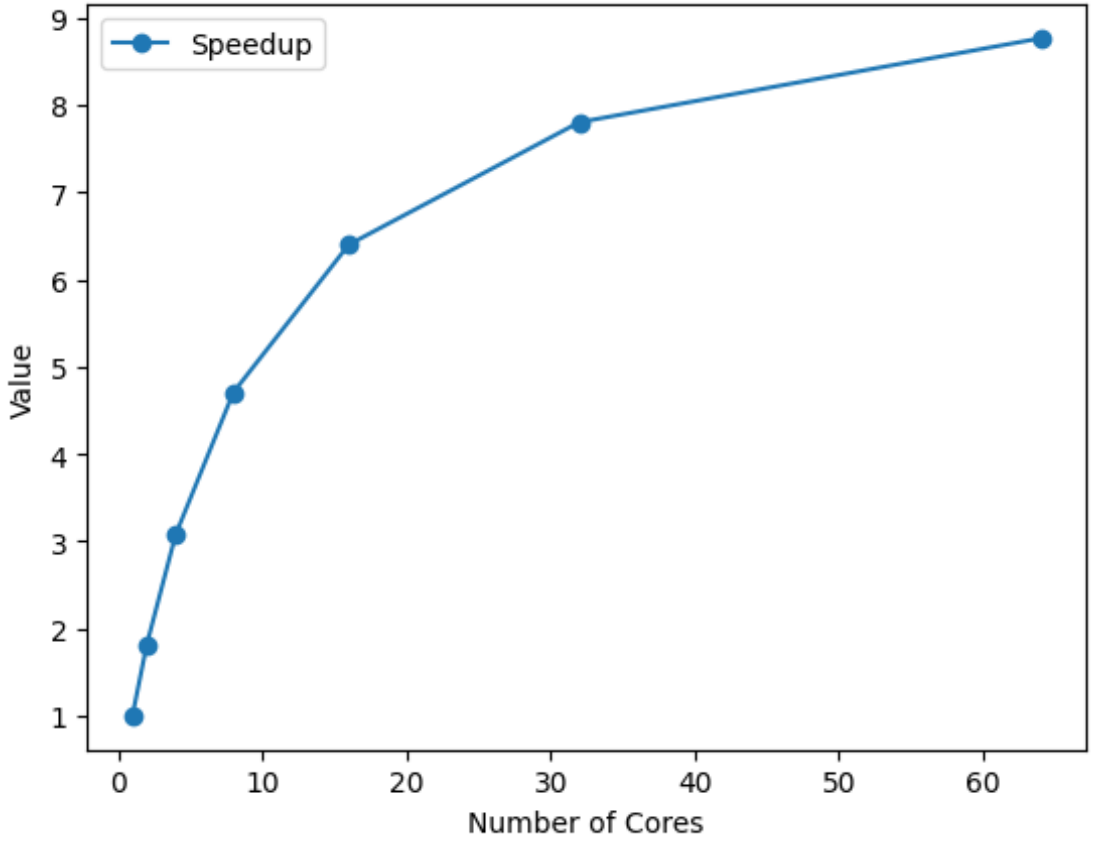
- So, cost = 1.4n + 17 + 20 = 1.4n + 37
- Costup (n) = cost(n) / cost(1) = (1.4n + 37) / 38.4

Problem 7

- Computing is cost-effective if speedup(n) > costup(n)
- i.e., speedup(n) / costup(n) > 1

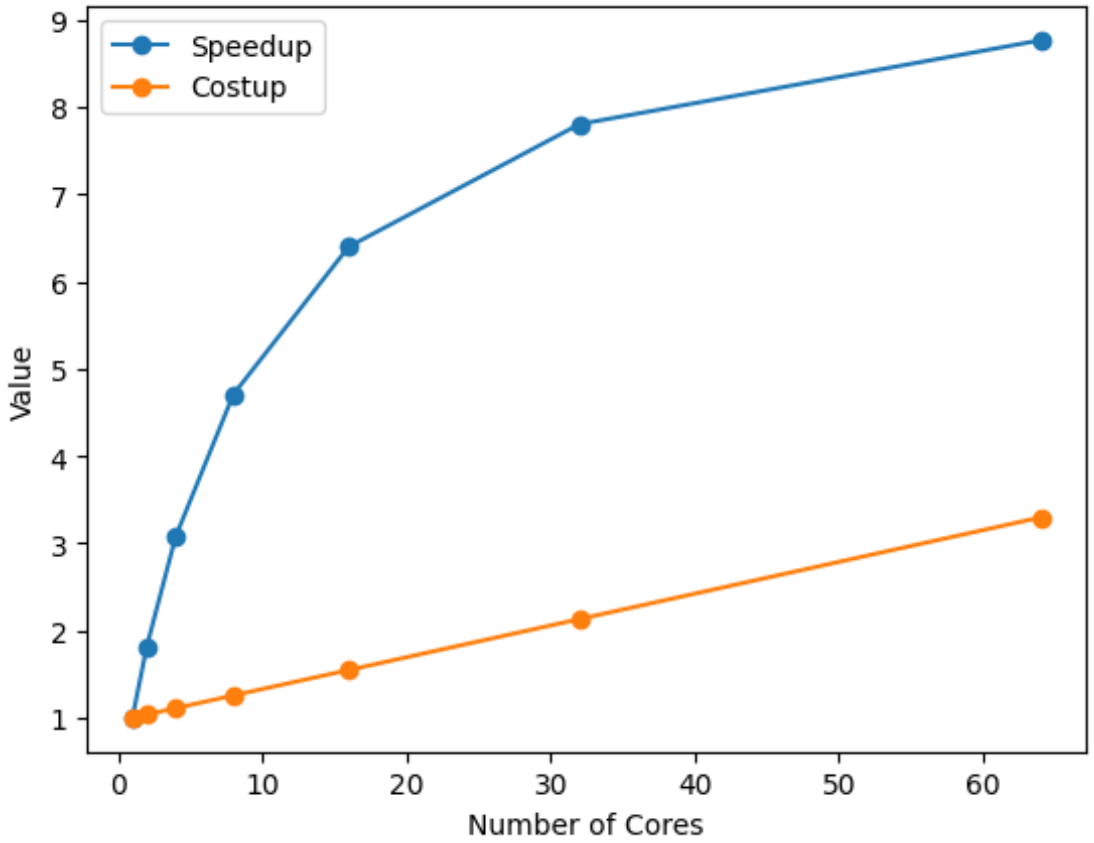
Problem 7

- Computing is cost-effective if speedup(n) > costup(n)
- i.e., Speedup(n) / costup(n) > 1



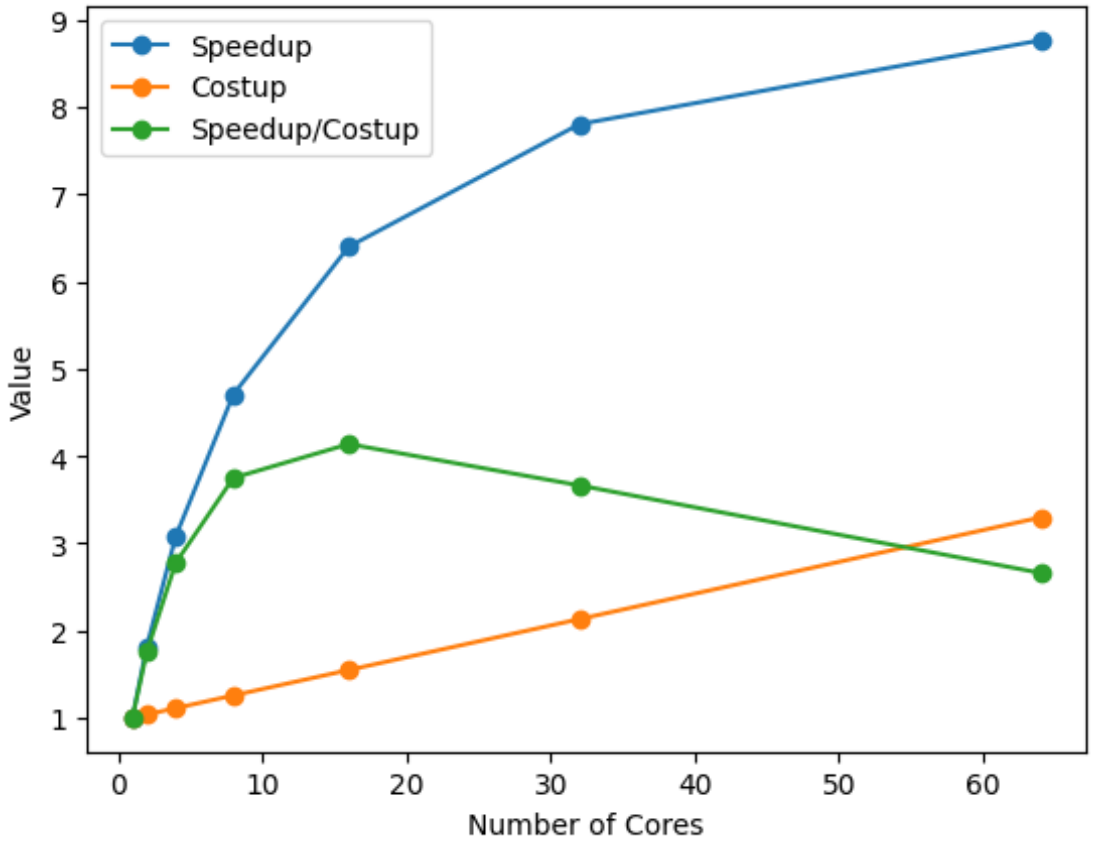
Problem 7

- Computing is cost-effective if speedup(n) > costup(n)
- i.e., Speedup(n) / costup(n) > 1



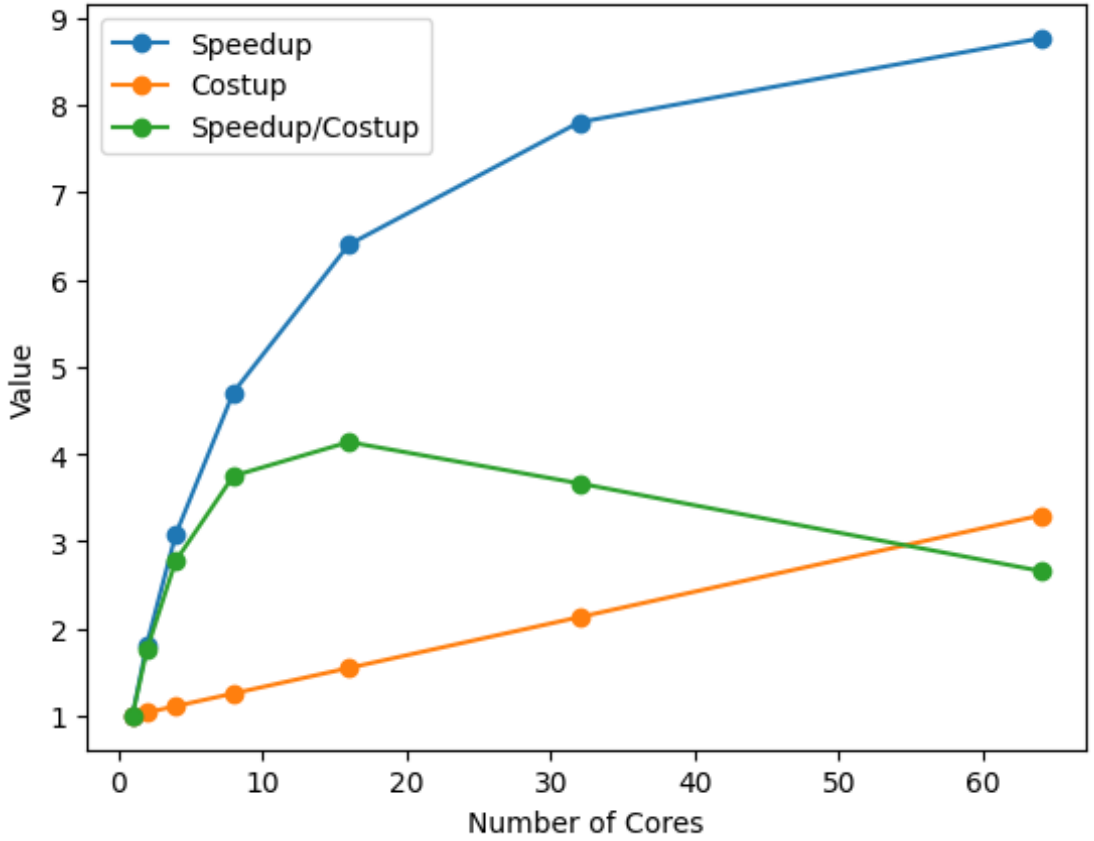
Problem 7

- Computing is cost-effective if speedup(n) > costup(n)
- i.e., Speedup(n) / costup(n) > 1



Problem 7

- Our CPU is cost-effective for all core counts
- Core count = 16 offers the best speedup relative to cost



Problem 8

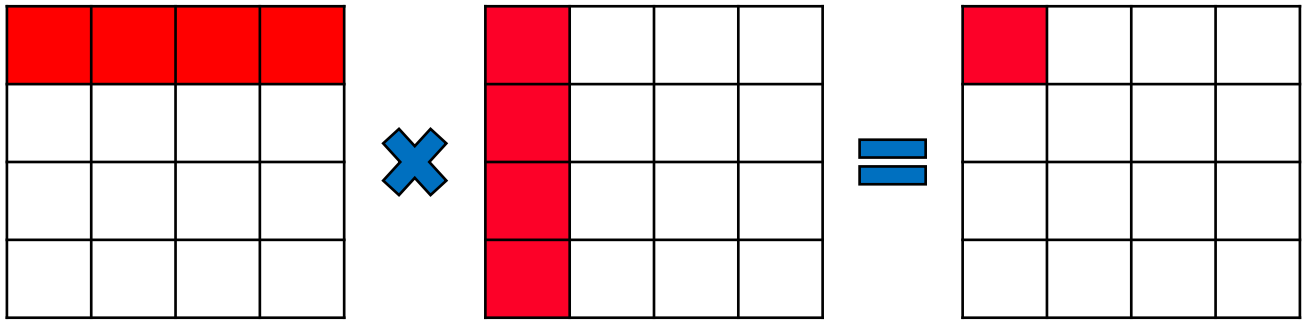
- Consider a simple matrix multiplication:

$$C = AB$$

- Where A, B and C are N x N matrices

- Assume all elements of A, B and C are in single precision floating point
- The system has a peak floating-point performance of 10 GFLOPs/s
- The memory bandwidth is 5 GB/s
- Is this operation memory bound or compute bound?

Problem 8



```
for(int i = 0; i < N; i++) {
      for(int j = 0; j < N; j++) {
            for(int k = 0; k < N; k++) {
                  C[i][j] += A[i][k] * B[k][j];
            }
      }
}
```

Problem 8: Worst case analysis

```
for(int i = 0; i < N; i++) {
     for(int j = 0; j < N; j++) {
          for(int k = 0; k < N; k++) {
               C[i][j] += A[i][k] * B[k][j];
          }
     }
}
```

- There are no registers or caches in the system
- For every calculation:
  - Read A[i][k], B[k][j] and C[i][j] from memory
  - Do two operations (one addition and one multiplication)
  - Write C[i][j] into memory

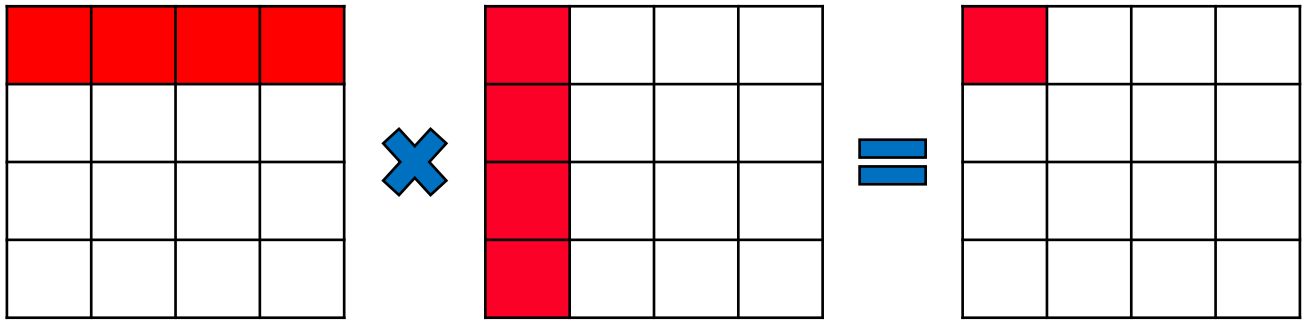
Problem 8: Worst case analysis

- Total number of memory operations = 4
- Each element is 4 bytes, so data transferred = 4 * 4 = 16 bytes

- Operational intensity (OI) = # operations / amount of data
- OI = 2 / 16 = 0.125 floating point ops / byte (very low!)

- Memory bound performance = OI * BW = 0.125 * 5 = 0.625 GFLOPs/s
- Peak possible performance = 10 GFLOPs/s
- Application is severely memory bound!

Problem 8: Best case analysis



- What if there is an ideal cache?
  - Matrix A and B only need to be read once, C written once
  - Maximum data reuse for the entire operation

- Total number of memory operations = $3N^2$
- Each element is 4 bytes, so amount of data transferred = $12N^2$ bytes

Problem 8: Best case analysis

- Total amount of data transferred = $12N^2$ bytes
- Total number of operations = $2N^3$ (remains unchanged)

- Operational intensity (OI) = $2N^3$ / $12N^2$ = N / 6 floating point ops/byte
- OI is a function of N!

- Memory bandwidth performance = OI * BW = 0.8N GFLOPs/s
- Peak possible performance = 10 GFLOPs/s
- Application is memory bound for N <= 12
- Application is compute bound for N >= 13