CSCI 5541: Natural Language Processing

Lecture 1: Overview

computer science
& engineering

UNIVERSITY OF MINNESOTA
Driven to Discover®

Introduction to Parallel Computing (CSCI 5451)

- ❑ **Class Time** : MW 8:15 - 9:30 am
- ❑ **Room** : Lind Hall 302
- ❑ **Instructor**: James Mooney <<moone174@umn.edu>>
- ❑ **TA**: Wenjie Zhang <<zhan7867@umn.edu>>
- ❑ **Course Site**: https://jimtmooney.github.io/Courses/F25/index.html

Course Objectives

❑ Understand what parallel computers are, why they are needed, their architectures and their various modes of processing

❑ Gain proficiency in OpenMP, MPI, CUDA, NCCL

❑ Learn how to think about and design parallel programs

❑ Link the above knowledge with how modern parallel programs work (such as attention variants & matrix multiplication in deep learning)

Course Site

Most course information will be on the course site

❑ Lecture slides/pdfs

❑ Homework descriptions + due dates + code

❑ Code examples from class

❑ Office Hours

❑ etc.

Canvas

The following will be exclusively found on Canvas

❑ Homework Submissions
❑ Project Submissions
❑ Grades

Communications

- We will be using slack for communications. Sometime next week I will be issuing announcements regarding this and all students will be added.

- Please do not use either emails or canvas for communications going forward.

- Direct inquiries on grading to Wenjie (the Graduate TA) in slack if you have them

Texts (Both Optional)

- “Introduction to Parallel Computing, 2nd Edition” by V. Kumar, A. Grama, A. Gupta, and G. Karypis (2003) – ISBN-13: 978-0201648652; ISBN-10: 0201648652 (***Good general resource on parallel computing, algorithms, and methods***)
- “Programming Massively Parallel Processors, Fourth Edition: A Hands-on Approach” by David B. Kirk and Wen-mei W. Hwu. (2022) – ISBN-13: 978-0323912310 (***Good resource for CUDA programming, specifically***)

Evaluation

- Homeworks (5 individual assignments x 15%)
- Group Project (25%)

Evaluation (Homeworks)

- Each will be given ~2 weeks to complete
- All are *individual*
- Each homework writeup will include
  - Homework description (algorithm + parallel framework to use)
  - Unit tests (input & output)
  - Serial version of program
- Submissions must include
  - A zip file containing the program
  - A markdown file describing the submitted program
- Submissions must be turned in by 11:59pm of the due date

Evaluation (Homework Grading)

- Autograded Portion
  - Does the program compile?
  - Does the program pass the published unit tests + our own internal unit tests?
  - Does the program achieve sufficient speedups over the serial version of the program?
- Human-Graded Portion
  - Does the program follow the writeup guidelines (correct APIs, etc.)?
  - Does the markdown summary accurately reflect the program?
- Autograder Testing
  - We will run the autograder at ~midnight for three days before the submission deadline if you wish to test your program

Evaluation (Projects)

- Groups of 3-4
- Work on this will be semester-long
- Think early & often about your project group & potential projects
- Good projects will either
    - Target difficult-to-parallelize algorithms presented in other courses/external work
    - Target open-source projects with un-parallelized sections (ideally those which are a bottleneck to certain functionality within said project)
- I will be meeting with groups during the semester 1-2 times to check in on progress/directions

Evaluation (Late Policy)

For the homeworks, a late penalty of 2.5% will be incurred for every 3 hours the assignment is past due. This more fractional policy is used as we know many students will likely be submitting their work the night of. This will still incur a penalty but it will be more minor. A full day late will result in a 20% penalty, 2 days 40%, etc. Refer to the below equation for determining the exact percentage deducted from your final grade based on how late your assignment is.

Percentage Deducted = Math.ceil(# hours since due time / 3) * 2.5

Late projects will not be accepted unless under extenuating circumstances made clear in advance.

Evaluation (Full Grade)

Grades will be assigned according to the following scale, where T is the total score (out of 100) you have achieved in this course.

A : $100 \geq T \geq 94$

A- : $94 > T \geq 88$

B+ : $88 > T \geq 82$

B : $82 > T \geq 77$

B- : $77 > T \geq 72$

C+ : $72 > T \geq 65$

C : $65 > T \geq 60$

C- : $60 > T \geq 55$

D+ : $55 > T \geq 50$

D : $50 > T \geq 40$

F : $40 > T$

Prerequisites

- This course assumes that you will be comfortable with C syntax, debugging, and algorithms.
- ***This is not an introductory course in programming***, but in applications of programming to the parallel setting.
- We assume that you will be able to incorporate new frameworks and their core ideas quickly.
- We will not be teaching the basics of C programming before diving into the work.
- Nor will we be focusing on the exact workings of some of the algorithms.

Word of Caution

❑ The assignments explored in this course will get more difficult as we move later into the semester

❑ By the end we will be exploring advanced topics requiring comfort with C/exploiting memory/{graph + matrix} algorithms

❑ For assignments, ***we will not be reviewing the algorithms you are to parallelize in detail***, it will be your responsibility to understand them and parallelize them

❑ If you do not have a strong background in programming, it is recommended you consider dropping the course

General Introduction

- ❑ A short history
- ❑ Why parallel computing?
- ❑ Motivation: Scientific applications
- ❑ Levels of parallelism
- ❑ Introduction to parallelism: algorithms, complexity for a simple example
- ❑ Obstacles to efficiency of parallel programs
- ❑ Types of parallel computer organizations

Historical Perspective: 3 Stages of Computing

1. Mechanization of arithmetic (abacus, Blaise Pascal’s adder, Leibnitz’s Stepped Reckoner)
2. Stored Programs (Jacquard Machine, Punched Cards)
3. Merging of Mechanized Arithmetic & Stored Programs
   a. Babbage Analytical Engine
   b. Hollerith Machine (Census)
   c. Mark 1 Computer (First Electromechanical Computer)
   d. ENIAC: Electronic Computer (Again used for Census)

Four (five) Generations:

1. Vacuum tubes
2. Transistors
3. Integrated Circuits
4. Very Large Scale Integration (VLSI)
5. **Some Japanese firms attempted to use AI ideas [LISP] and massive parallelism. In some ways they were too early!

Von Neumann Architecture

❑ Central Processing Unit
  ○ Control Unit
  ○ Arithmetic Logic Unit (ALU)

❑ Memory Unit
  ○ Main Memory
  ○ Cache
  ○ ROM

❑ Peripherals
  ○ External Memories
  ○ I/O Devices
  ○ Terminals
  ○ Printers

Von Neumann Execution

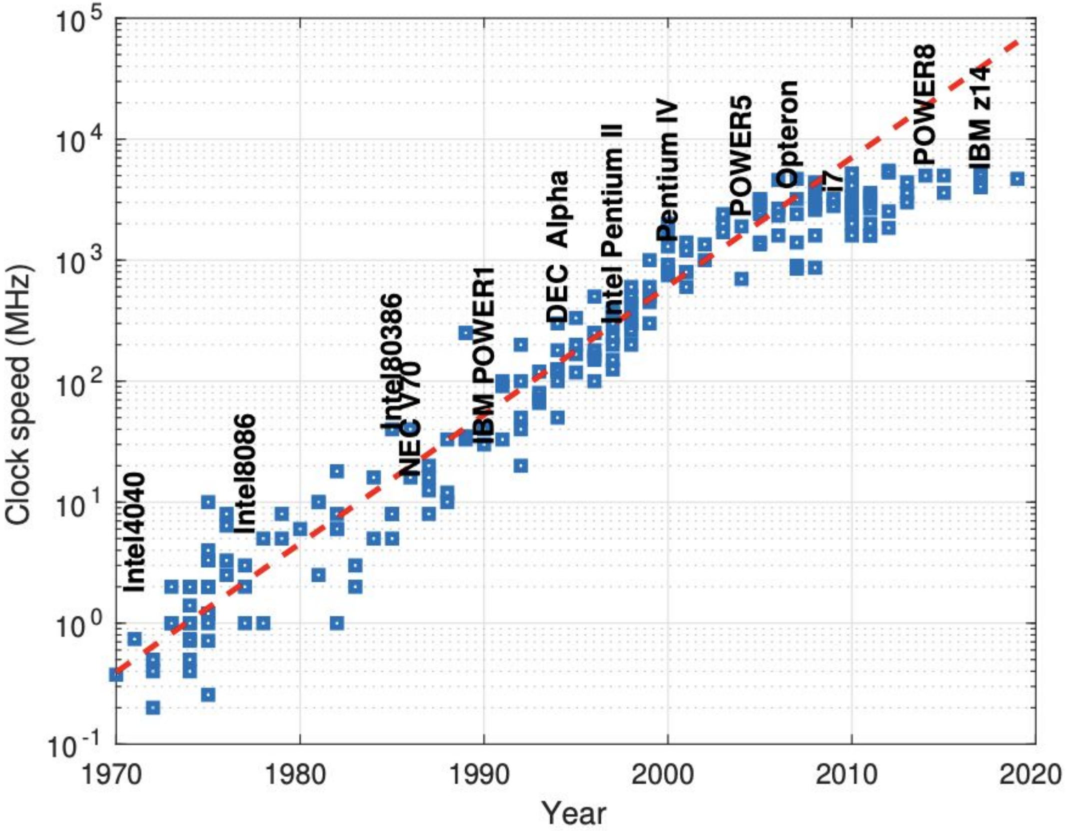
- Fetch next instruction from memory into instruction registers
- Fetch operands from memory into data registers
- Execute instruction
- Load result into memory
- Repeat: fetch next instruction…

→ Parallelism was slowly introduced into this scheme as ICs became more complex/faster

→ Beginning in the late 1980s, parallelism was a major way to get speedups when computing

Why Parallelism?

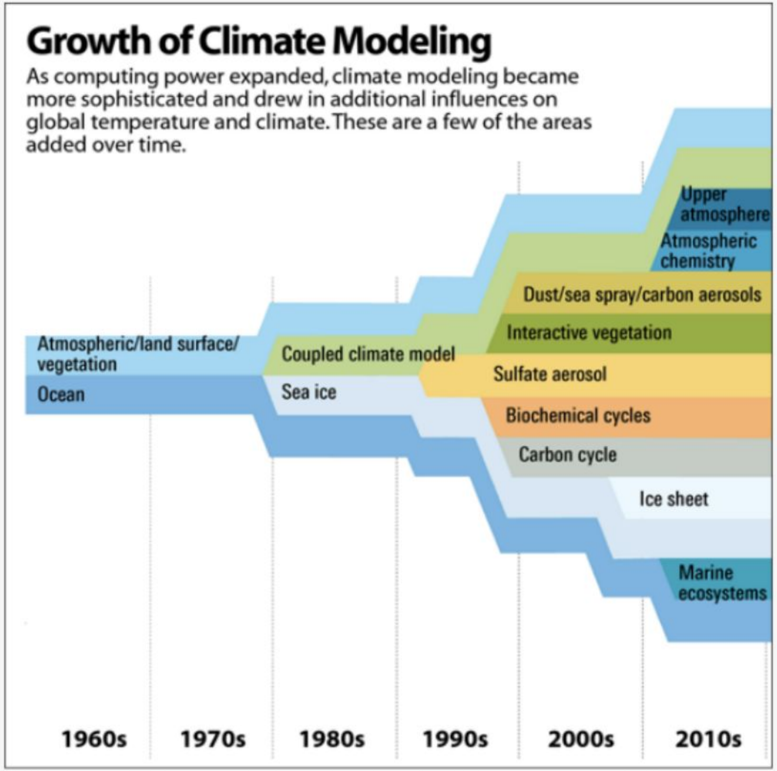
Last 50 years of clock speed improvements



Why Parallelism?

- Main argument: Clock speeds have largely reached their upper bound
- Sustainable gains in speed are only possible through better software, better utilization of hardware and parallelism
- Parallelism is cost-effective: multiplying hardware is cheap, fast components are expensive
- Parallelism helps memory-wise (multiplying memory is cheap, building a system with a single large memory is expensive)

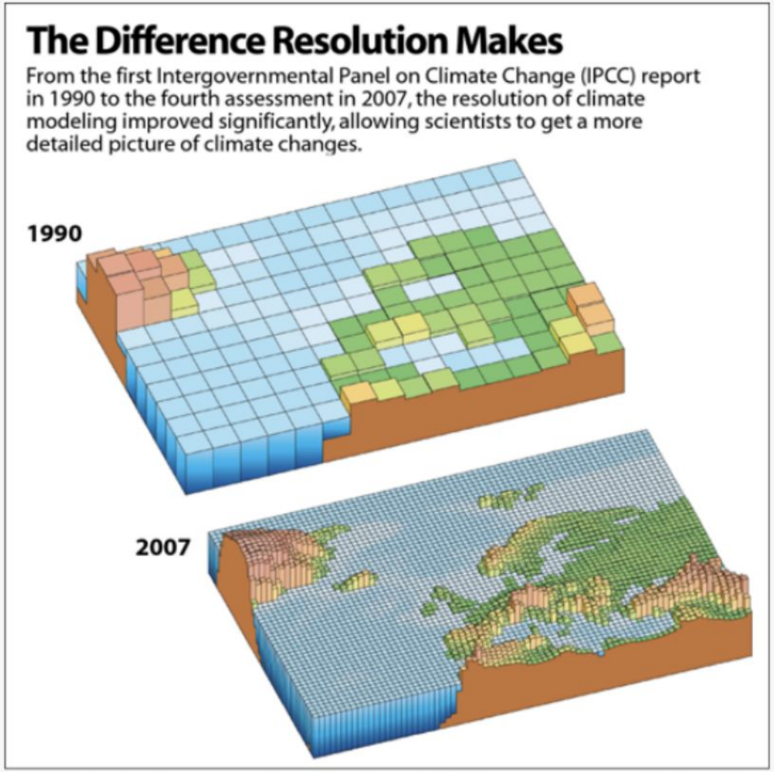
Why Supercomputers? Weather Simulation



Growth of Climate Modeling

As computing power expanded, climate modeling became more sophisticated and drew in additional influences on global temperature and climate. These are a few of the areas added over time.

SOURCE: University Corporation for Atmospheric Research (UCAR) InsideClimate News

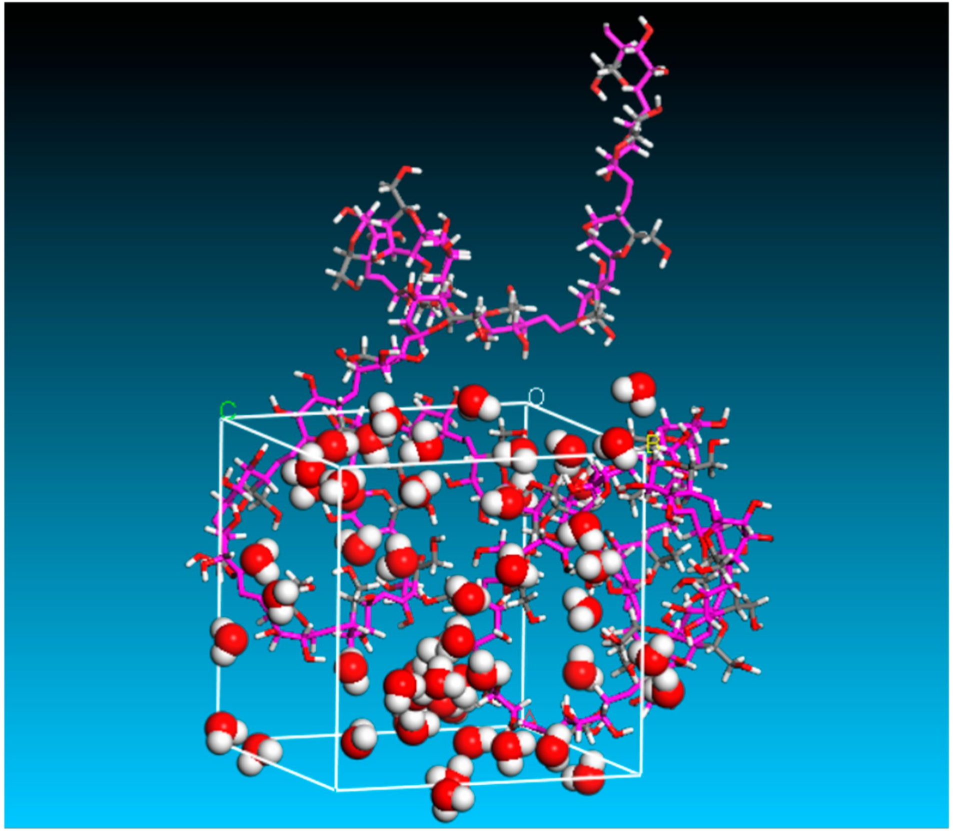


The Difference Resolution Makes

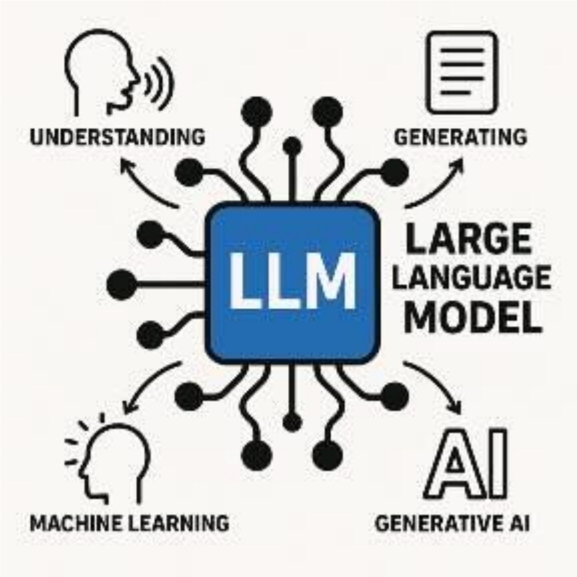
From the first Intergovernmental Panel on Climate Change (IPCC) report in 1990 to the fourth assessment in 2007, the resolution of climate modeling improved significantly, allowing scientists to get a more detailed picture of climate changes.

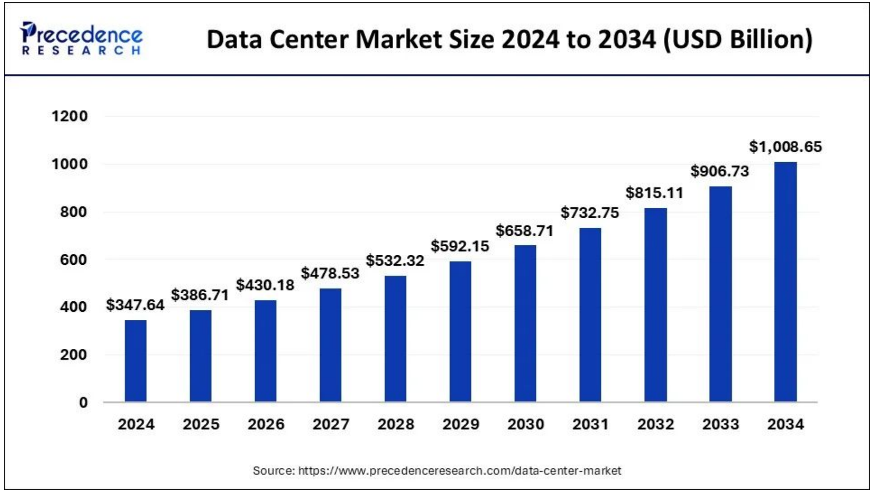
SOURCE: University Corporation for Atmospheric Research (UCAR) InsideClimate News

Why Supercomputers? Molecular Dynamics



Why Supercomputers? AI, LLMs, and Deep Learning





Data Center Market Size 2024 to 2034 (USD Billion)

Source: https://www.precedenceresearch.com/data-center-market

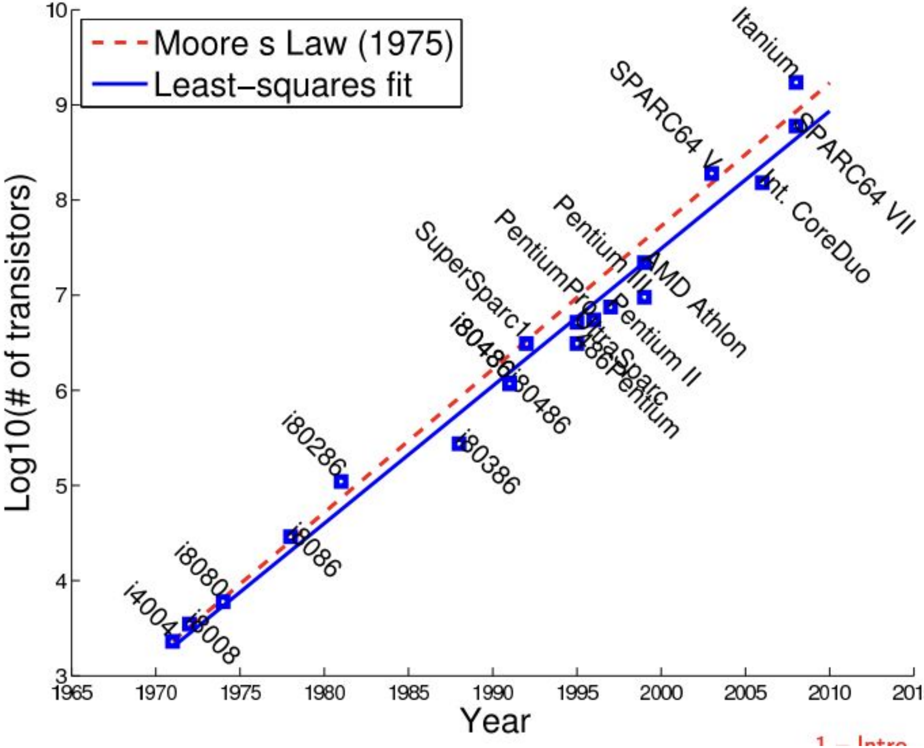
Parallel Computing - Motivation - Moore’s Law

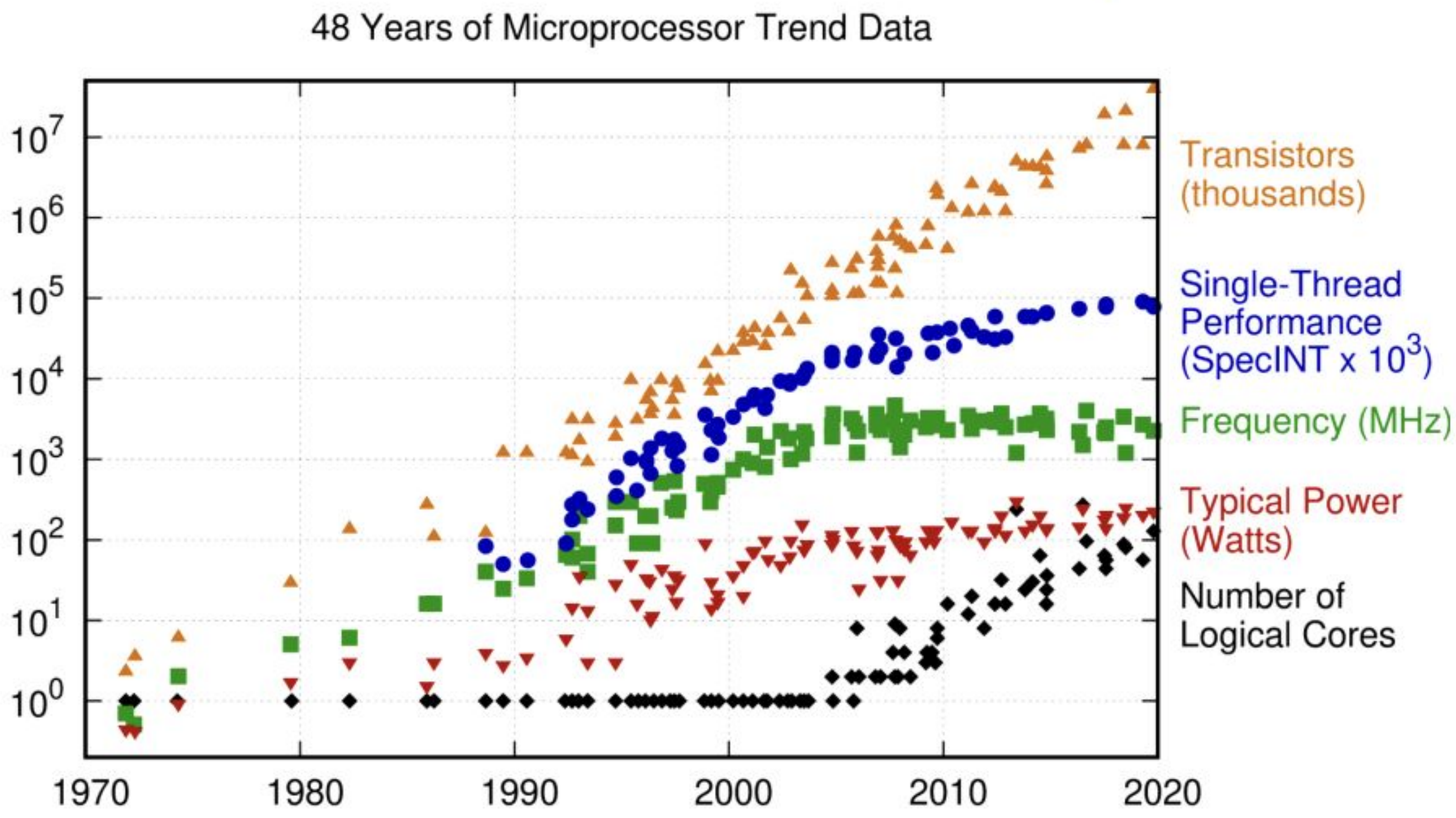
- ❑ Demand for computational speed is always increasing (AI, physics, biology, chemistry)
- ❑ Clock speeds became harder to increase around mid 1970s
- ❑ From 1950 to mid 70s, increase of 100, 000x
- ❑ 3 orders of magnitude from clock speeds. The other 2 design.
- ❑ A factor of 10 every 5 years.
- ❑ From 1970 to 2005: a gain of 5000x
- ❑ From 2005 to present: gain in clock cycle has effectively stopped

Parallel Computing - Motivation - Moore’s Law

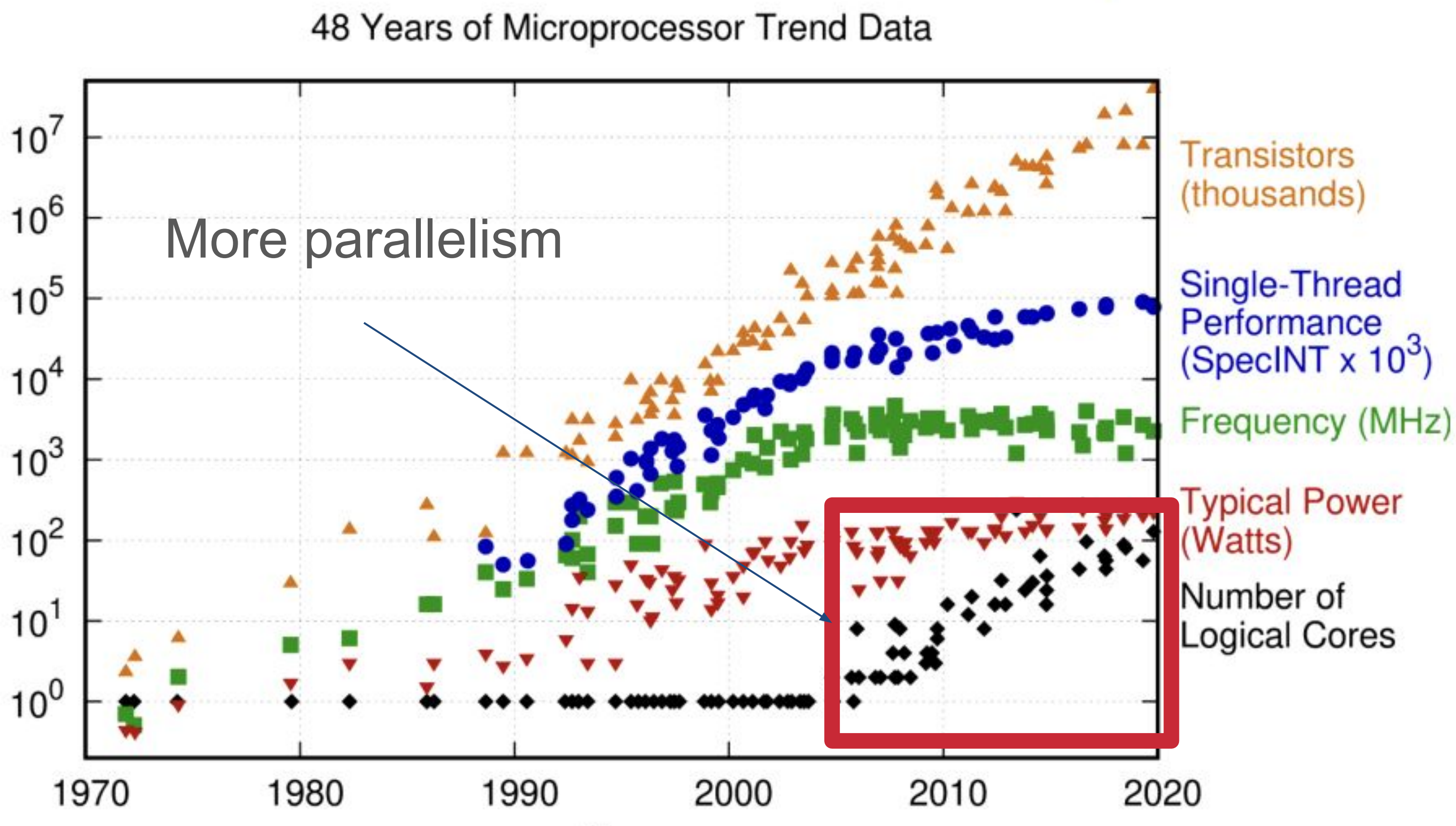
No clock speed improvements, but there are many more transistors on each chip → how to use them?

- More caching/memory
- More parallelism (explicit & implicit)





Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2019 by K. Rupp



More parallelism

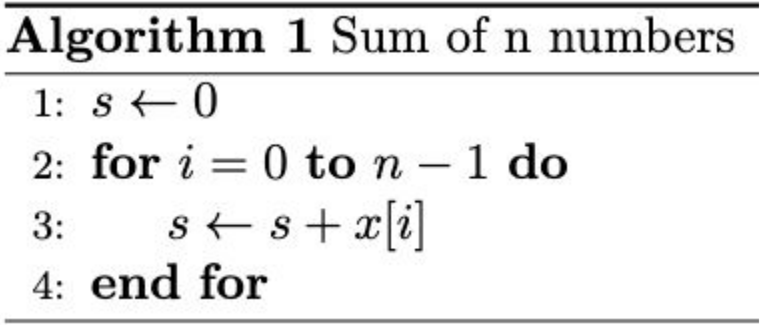
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2019 by K. Rupp

Basics of Parallel Computing

Consider summing n numbers from an array **x**

$$s = \sum_{i=1}^{n} i$$

Represented via its algorithm through n-1 sequential operations below

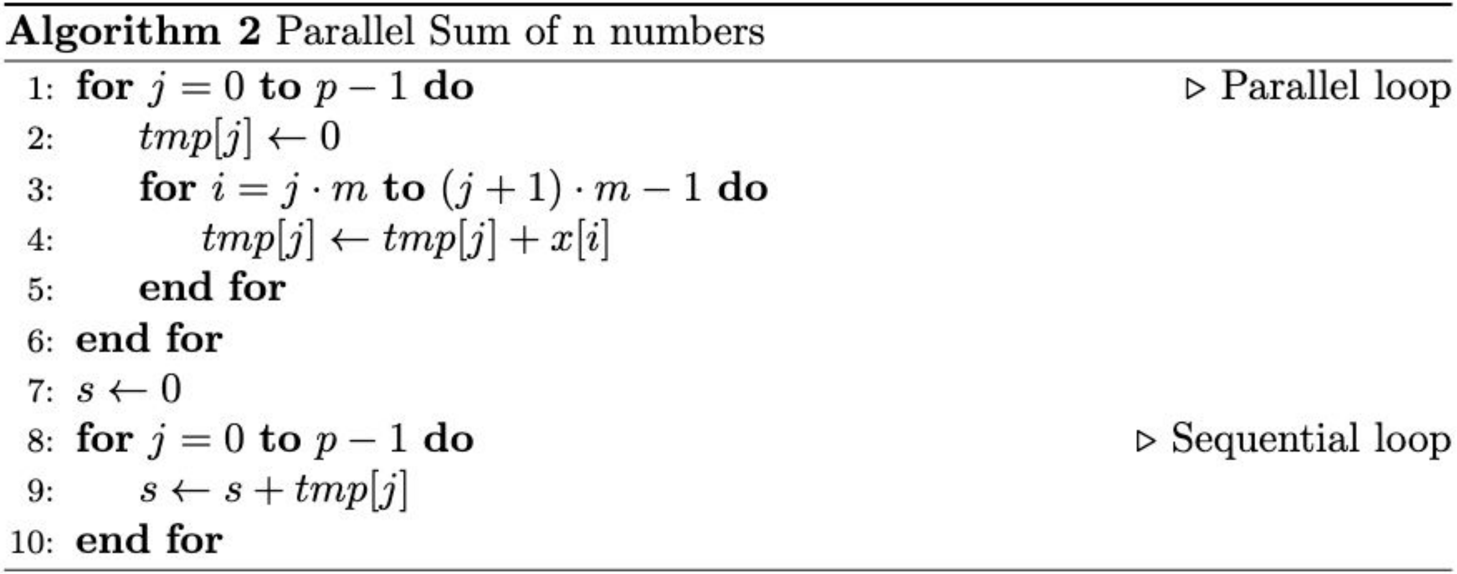


Algorithm 1 Sum of n numbers

```
$s \leftarrow 0$
for $i = 0$ to $n - 1$ do
    $s \leftarrow s + x[i]$
end for
```

Basics of Parallel Computing

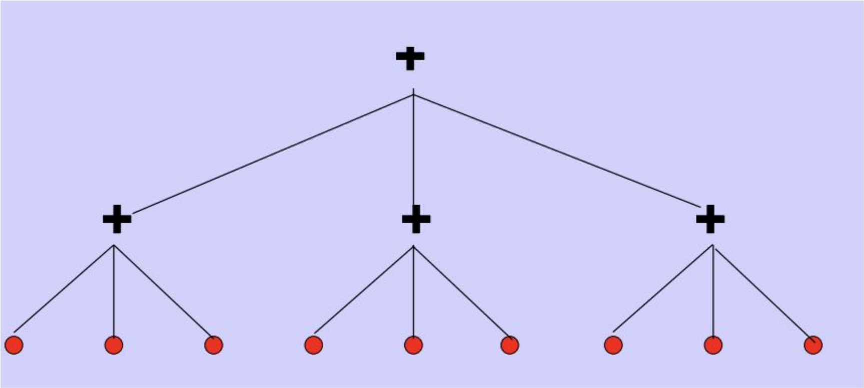
Split the sum into p subsums and set m = n/p. We can parallelize this by summing subsets of the array in parallel, then sum each of those sums.



Algorithm 2 Parallel Sum of n numbers

```
for $j = 0$ to $p - 1$ do                          ▷ Parallel loop
    $tmp[j] \leftarrow 0$
    for $i = j \cdot m$ to $(j + 1) \cdot m - 1$ do
        $tmp[j] \leftarrow tmp[j] + x[i]$
    end for
end for
$s \leftarrow 0$
for $j = 0$ to $p - 1$ do                          ▷ Sequential loop
    $s \leftarrow s + tmp[j]$
end for
```

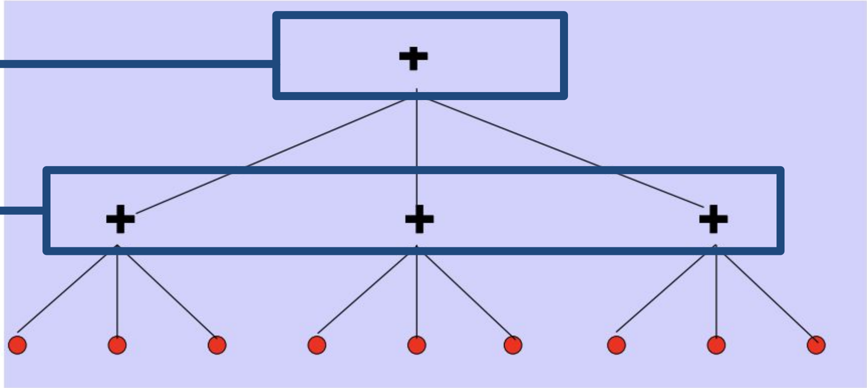
Basics of Parallel Computing



Number of operations:

→ p * (m - 1) + p -1 = n - 1 (unchanged number of total operations)

Basics of Parallel Computing

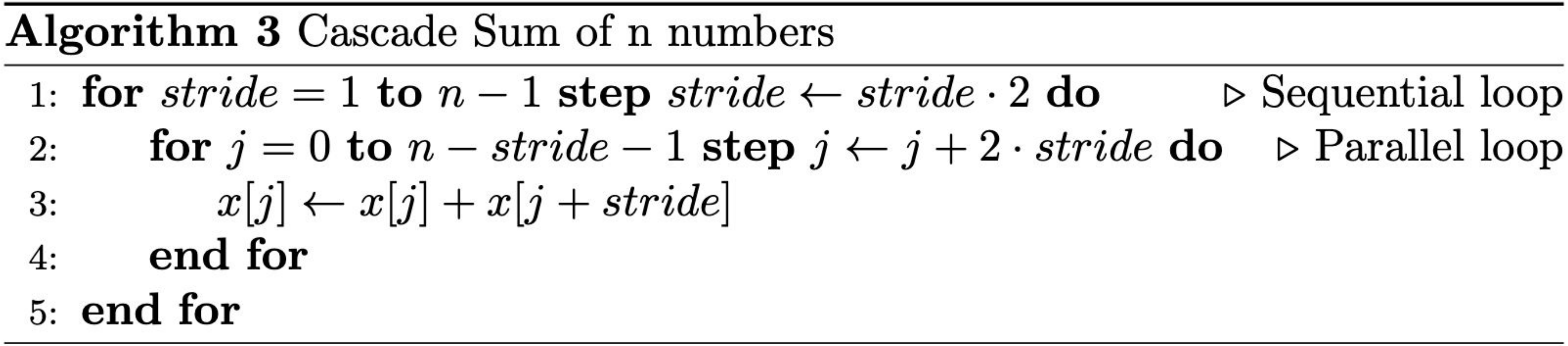


Number of operations:

$$\rightarrow \underbrace{\boxed{p * (m - 1)}}_{\text{Parallel}} + \underbrace{\boxed{p - 1}}_{\text{Serial}} = n - 1 \text{ (unchanged number of total operations)}$$

Basics of Parallel Computing

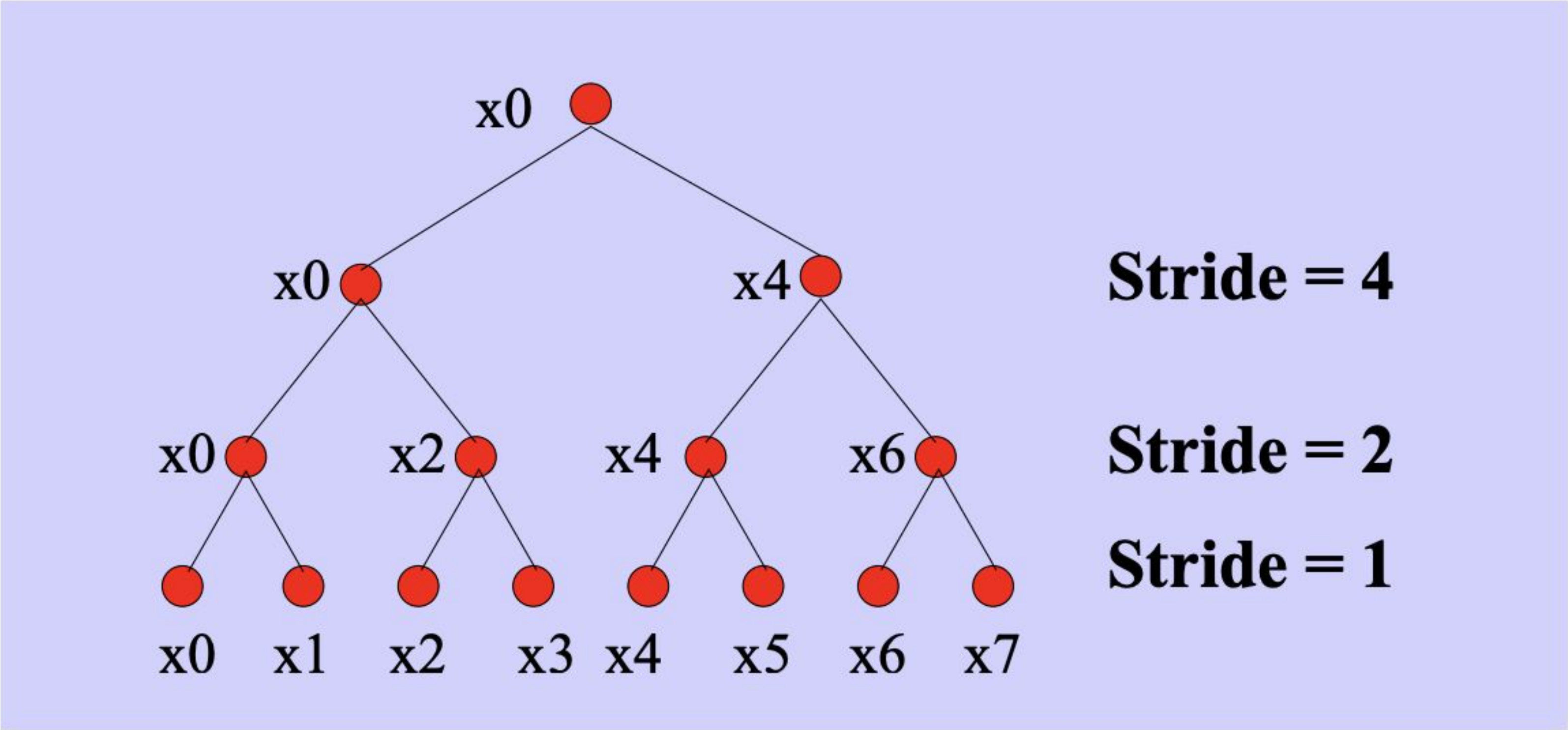
- Assume n = 2^k and n/2 arithmetic units are available
- Divide recursively into two sub-sums. Perform a cascade summation
- Requires $log_2 n$ steps



Algorithm 3 Cascade Sum of n numbers

```
for $stride = 1$ to $n - 1$ step $stride \leftarrow stride \cdot 2$ do        ▷ Sequential loop
    for $j = 0$ to $n - stride - 1$ step $j \leftarrow j + 2 \cdot stride$ do    ▷ Parallel loop
        $x[j] \leftarrow x[j] + x[j + stride]$
    end for
end for
```

Basics of Parallel Computing



Levels of Parallelism

Five different types of parallelism commonly exploited

1. **Job Level:** Running entirely unrelated jobs (e.g. OS-level parallelism)
2. **Macrotask level**: Execution of different sections of a program (e.g. worker-scheduler programs, data pipelines)
3. **Microtask level**: Parallelism within different steps of a loop, may perform different tasks depending on each loop (event worker during data ingestion deciding how to parse each event)
4. **Data Parallelism**: Same operation performed on similar datasets (e.g. adding two vectors)
5. **Arithmetic Level**: Pipelining, additional functional units)

Barriers to Parallel Efficiency

The example of adding n numbers can give an indication to a few of the potential barriers to efficiency:

1. **Data Movement**: Data to be exchanged between processors. Data may have to move through several stages, or other processors, to reach its destination
2. **Poor load balancing**: Ideally: all processors should perform an equal share of work all the time. However, processors are often idle waiting data from others
3. **Synchronization**: Some algorithms require synchronization. This global operation can be costly.

Barriers to Parallel Efficiency

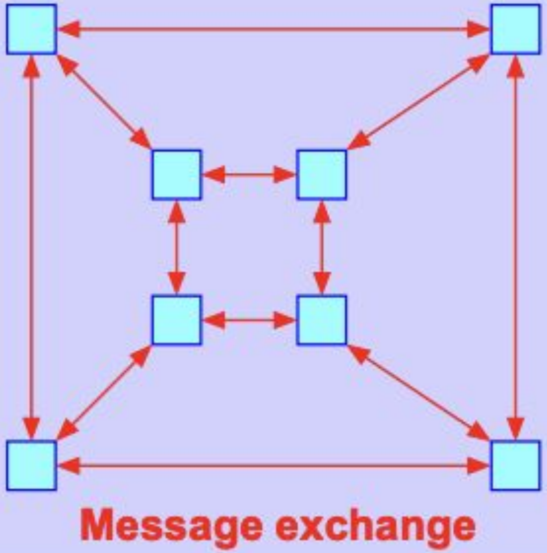
4 **Impact of Architecture**: Often, in parallel processing, processors will be competing for resources, such as access to memory or cache

5 **Inefficient Parallel Algorithm:** Some sequential algorithms do not easily parallelize. New algorithms will be required - which may be inefficient in sequential context

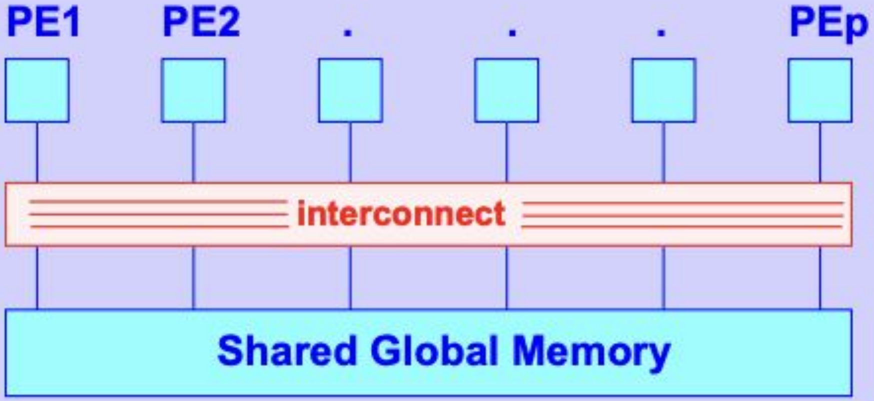
Three common parallel programming paradigms

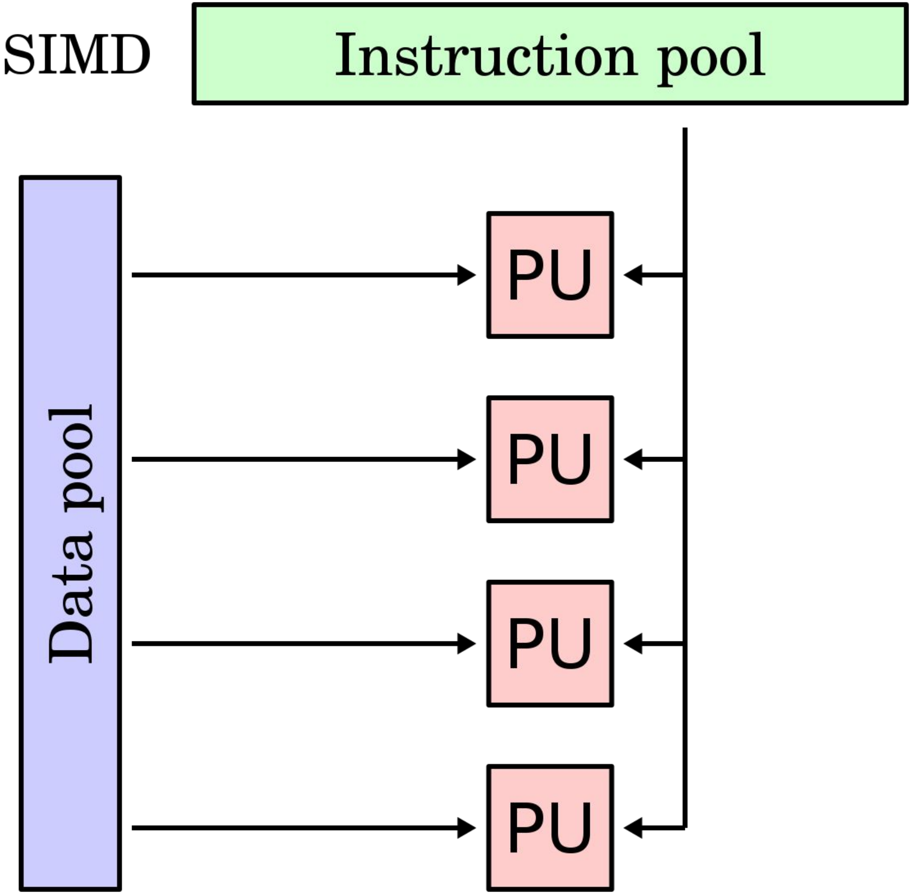
1. **Shared global Memory Viewpoint**: Programs will execute in parallel and access a shared address space [Drawbacks → need to make sure that data is always synchronized] [Example: OpenMP]
2. **Message Passing Viewpoint**: Programs to run in parallel exchange data in explicit ways as determined by the user [Drawbacks: Difficult to program] [Example: MPI]
3. **SIMD/SIMT viewpoint**: Programs execute the same instructions on different data at the same time → extremely fast on specialized hardware [Drawbacks: Difficult to use with irregular computation] [Example: GPUs, Vector Extensions]

Message passing



Shared memory




SIMD
Instruction pool
Data pool
PU
PU
PU
PU