



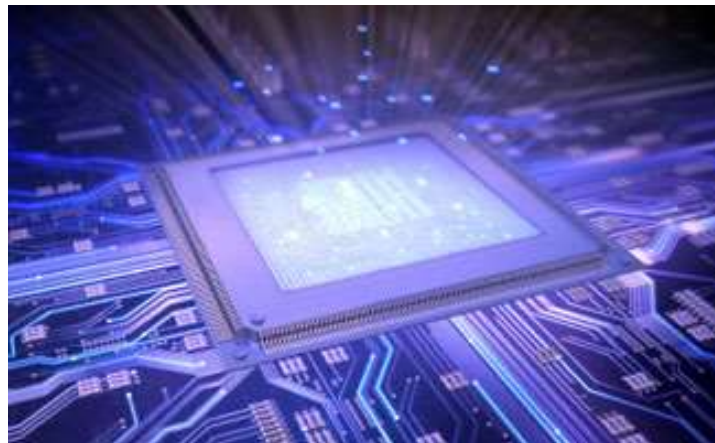
Parallel Computing

Why Parallel Computing?

Mohamed Zahran (aka Z)

mzahran@cs.nyu.edu

<http://www.mzahran.com>



Who Am I?



- Mohamed Zahran (aka Z)
- Computer architecture/OS/Compilers Interaction
- <http://www.mzahran.com>
- Office hours:
 - Online: zoom link provided in course website and Brightspace
 - Wed 2-3:30pm
 - If you cannot make it to office hours
 - We can set an online appointment.

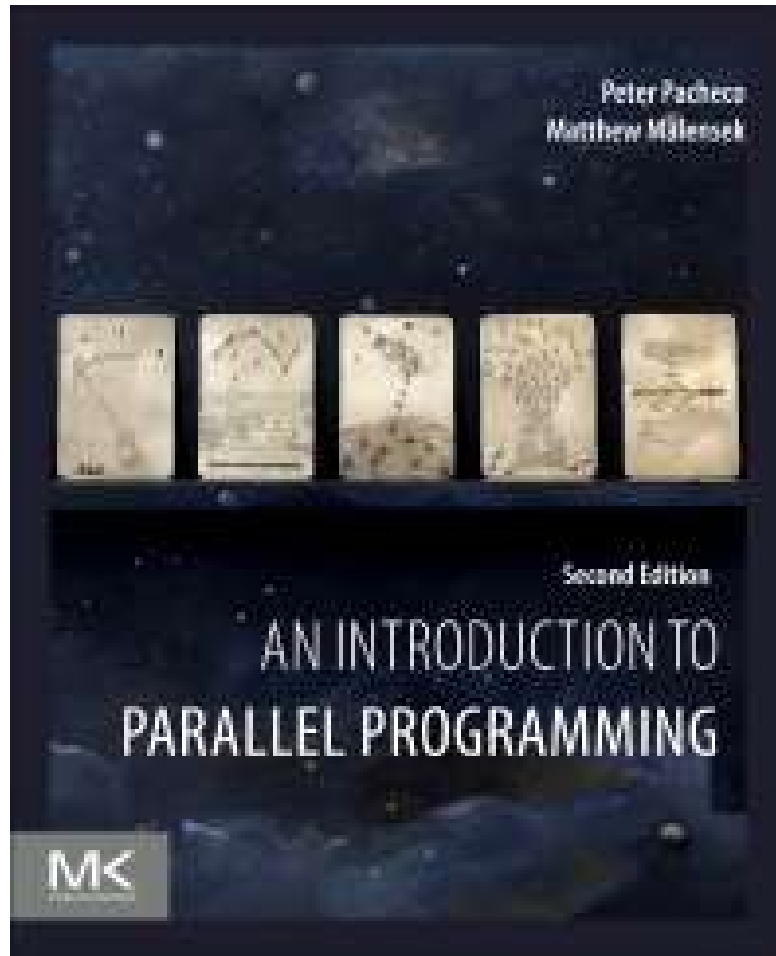
Main Goals of this Course

- Why did parallel computing become the **current** norm? And why is it here to stay?
- How does the parallel hardware look like?
- What are the challenges of parallel computing?
- How to write parallel programs and **make the best use of the underlying hardware?**

My wish list for this course

- Learn to think in parallel
- Make the best choice of hardware configuration and software tools/languages
- Be ready for the competitive market or for your next step in the academic/research ladder
- Learn how to progress way beyond the course!
- Enjoy the course!

Textbook



- Authors:
 - Peter Pacheco
 - Matthew Malensek
- Release Date: 2021
- Publisher: Elsevier
- Print Book ISBN :
9780128046050

Course Components

- Lectures
 - Higher level concepts (slides + reading material)
- Homework assignments (15%)
 - The theoretical part
- Programming labs (30%)
 - Learning by doing
- Midterm Exam (25%)
- Final Exam (cumulative) (30%)

Web Presence

Brightspace

- Announcements from instructor and grader
- Discussion forums
- Download assignments and labs
- Submissions (of assignments and labs)
- Getting grades

Course Website

- Lecture slides
- Reading material
- Useful links

Policies: Assignments

- You must work **alone** on all assignments/labs
 - Post all questions on Brightspace discussion forums.
 - You are encouraged to answer others' questions but refrain from explicitly giving away solutions.
- Hand-ins
 - Submission through Brightspace by due date and time
 - You can submit several times till the deadline. We will grade only the last submission.

Policies:

The following excuses are not accepted and result in penalty:

- I tried to submit one minute after the deadline but I couldn't.
- I submitted the wrong file.
- My machine crashed the night before the deadline
 - It is highly encouraged to submit each time you finish part of the lab.
 - We will grade only the last submission.
- I spent 100 hrs/week studying for this course, why didn't I get a high grade?
 - Do you really think that your grade is just a function of how much you study?
- What do I concentrate on when studying for the exam?
 - Do you really mean that some parts of the material are not important?

Policies:

Arguing a grade of an assignment, lab, or exam.

You have **one week** from the time you receive your assignment/exam grade to argue about it if you want.

- If lab/homework, first discuss the issue with the grader.
- If issue is not resolved, then come to me.
- For exams, come to me directly.

After that, no arguments are allowed.

Now, what is this story of
parallel computing, multicore,
multiprocessing, multi-this and
multi-that?

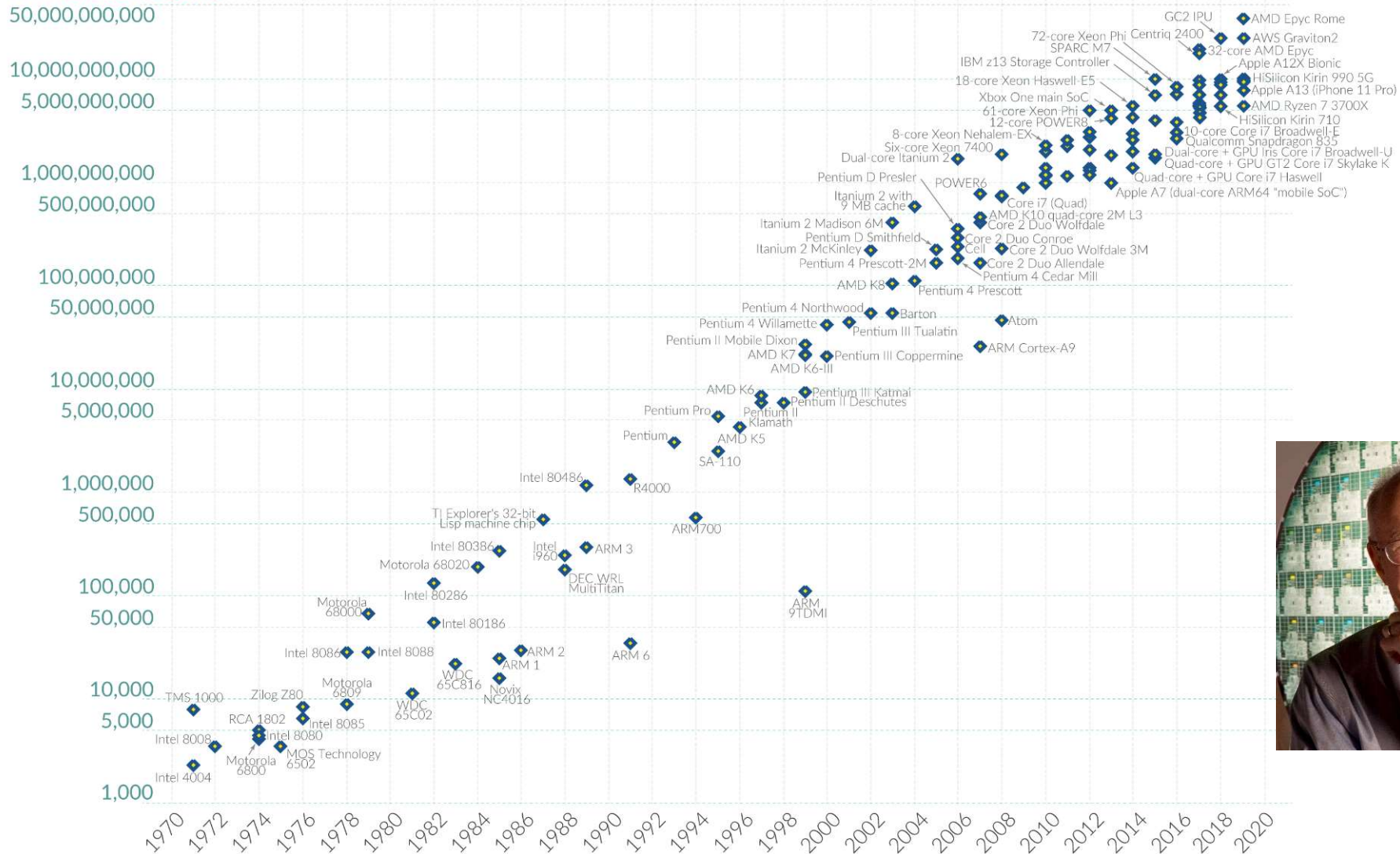
The Famous Moore's Law

Moore's Law: The number of transistors on microchips doubles every two years

Our World
in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.

Transistor count



Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)

OurWorldinData.org – Research and data to make progress against the world's largest problems.

Licensed under CC-BY by the authors Hannah Ritchie and Max Roser.

It was implicitly assumed that more transistors per chip = more performance. BUT ...

Effect of Moore's law

Performance increase per year:

- ~1986 - 2002 → 50% performance increase
- Since 2002 → ~20% performance increase

Hmmm ...

- Why do we care? 20%/year is still nice.
- What happened at around 2002?
- Can't we have auto-parallelizing programs?

Why do we care?

- More realistic games
- Decoding the human genome
- More accurate medical applications

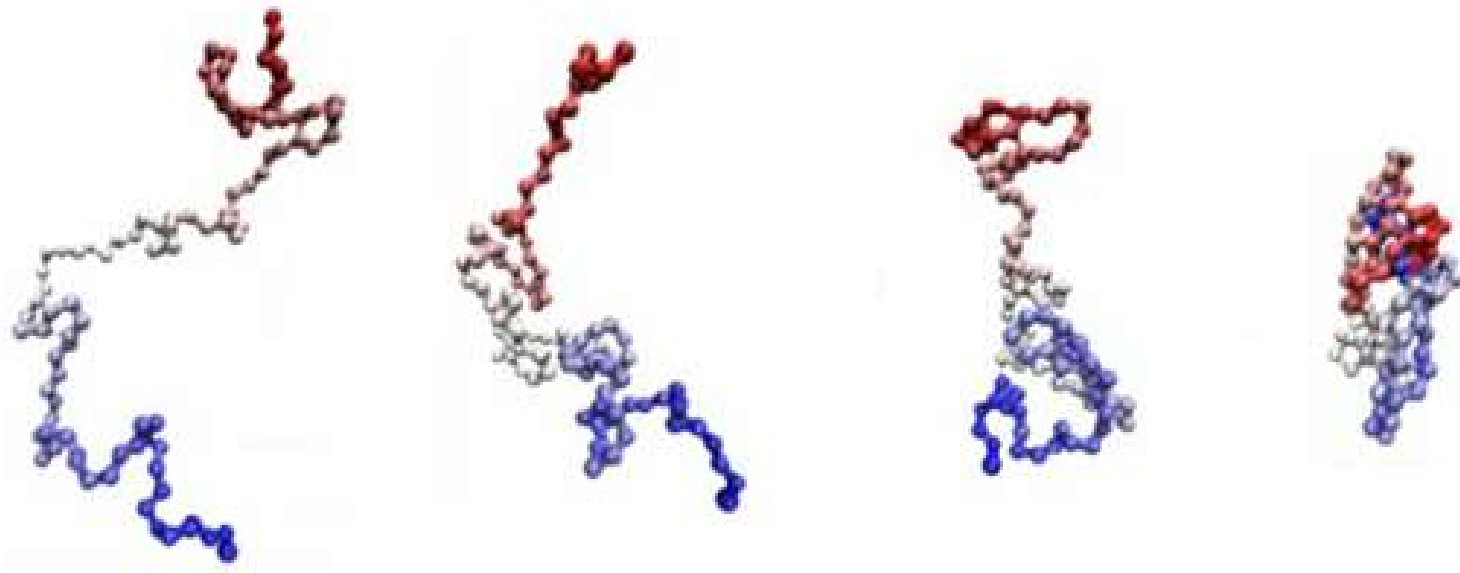
The list goes on and on

As our computational power increases → the number of problems we can seriously consider also increases.

Climate modeling



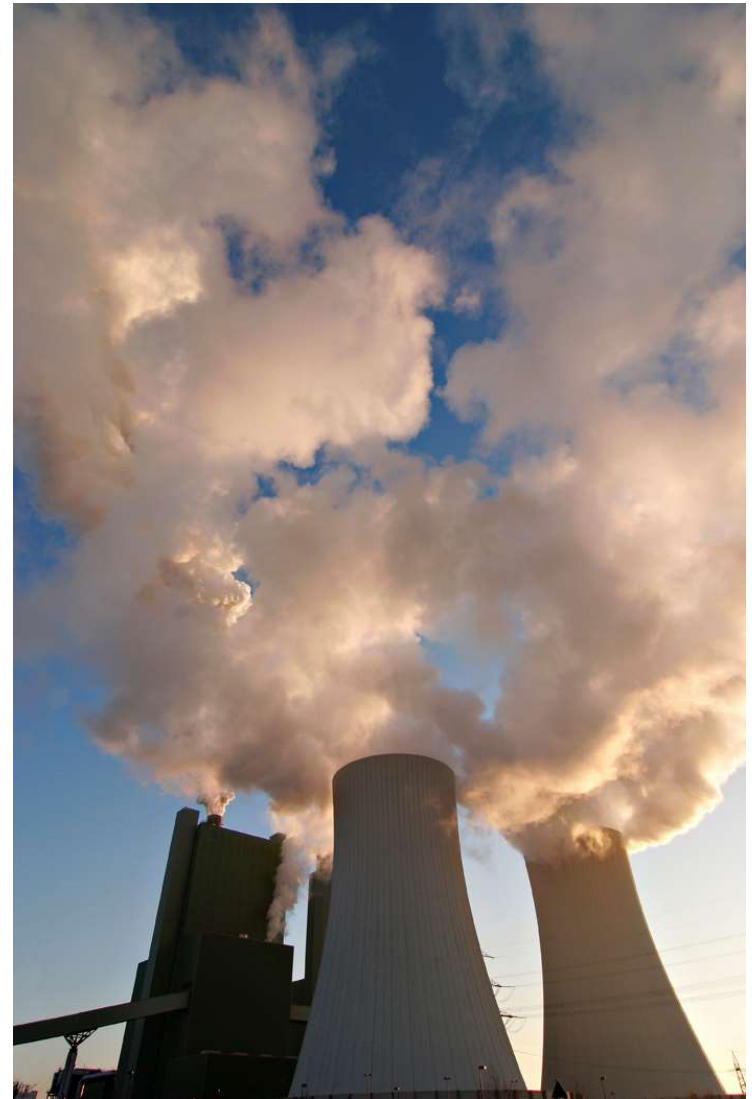
Protein folding



Drug discovery



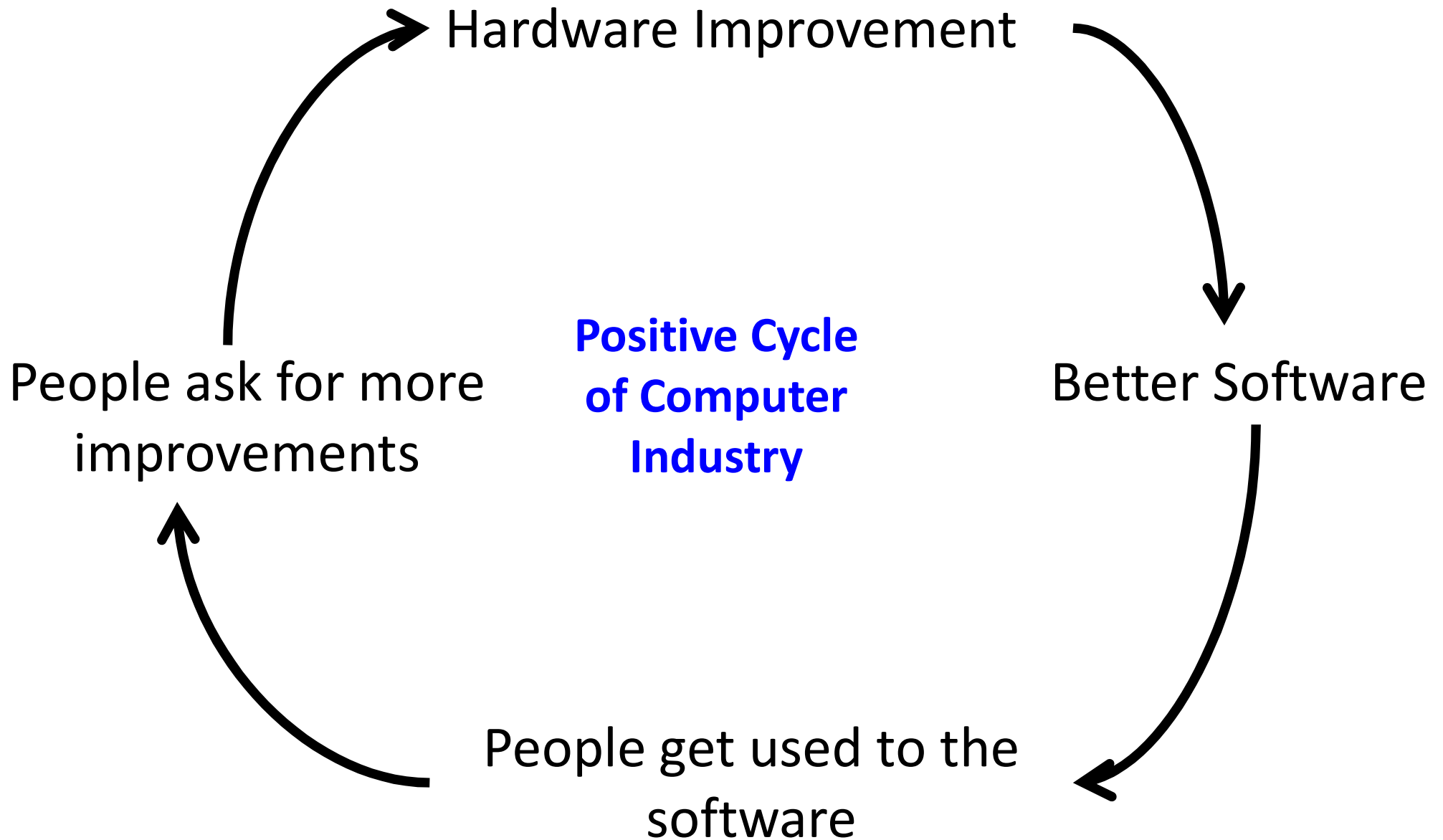
Energy research



Data analysis



+2.688
+5.000
+1.500
+1.125
+1.062



Why did we build parallel machines
(and continue to do so)?

(multicore, multiprocessors,
multi-anything!)

What happened around 2002 that made performance enhancement
go down from 50%/year to only 20%/year?

Moore's law works because of ...

Dennard scaling

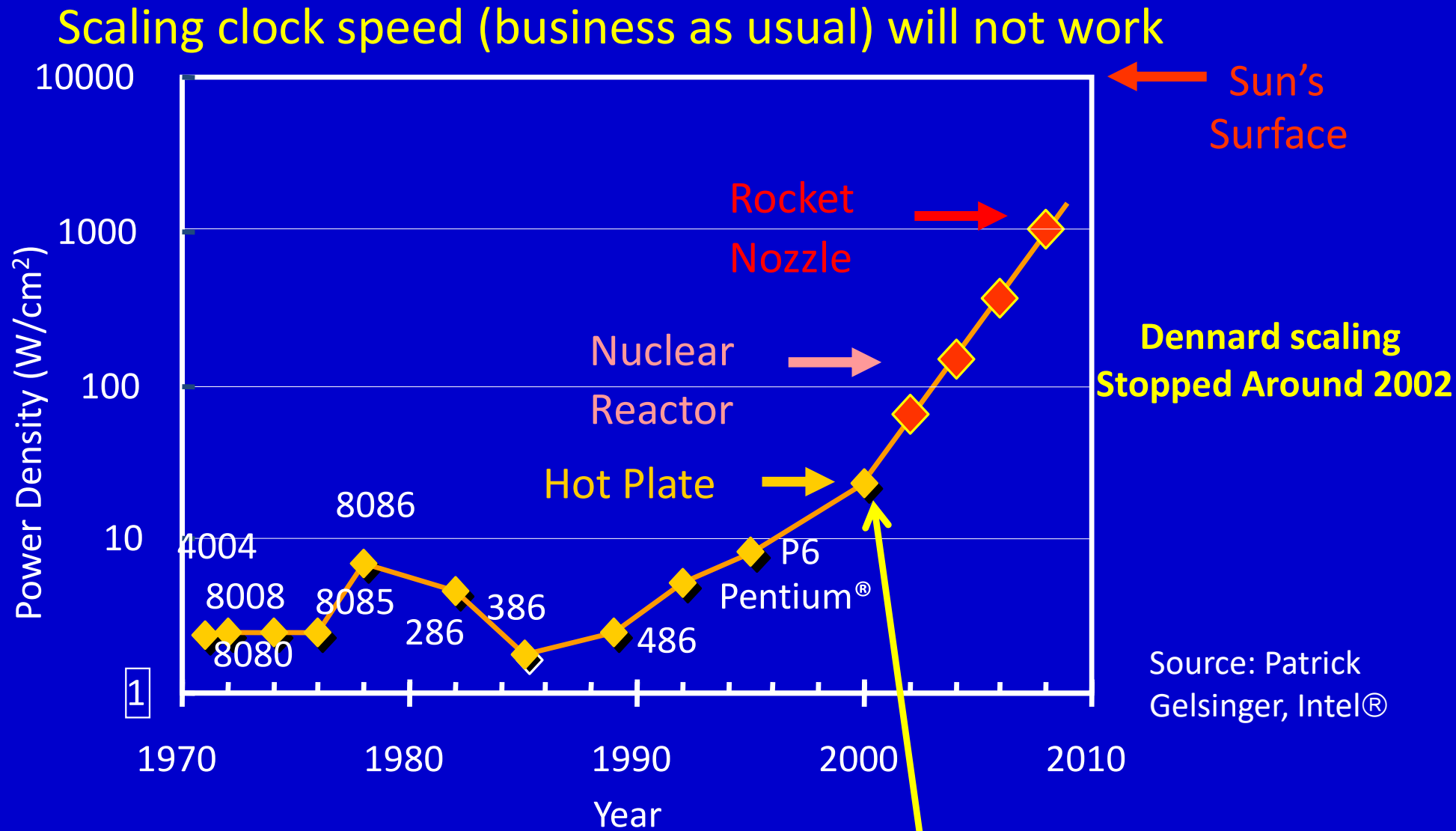
Simply speaking scaling down transistor dimensions leads to:

- Reduce circuit delay
- Increase operating frequency
- Operating voltage is reduced → reducing power

Named after Robert Dennard

Power Density

Moore's law is giving us more transistors than we can afford!



This is what happened at around 2002!

Why Did Power Become a Problem?

- Power needed per chip is:
 - Power consumed: to do the calculations the processor does to execute your programs.
 - Power dissipated: in the form of temperature.
- More power means:
 - Exponential increase in the cost of packaging
 - Electricity bill for big machines (or shorter battery life for portable machines)
- Power per chip increases due to two factors:
 - Increasing the number of transistors per chip
 - After Dennard scaling stopped.
 - Increasing the frequency (i.e. clock scaling)

Multicore Processors Save Power

$$\text{Power} = C * V^2 * F$$

(C = capacitance V = voltage F = frequency)

$$\text{Performance} = \text{Cores} * F$$

Let's have two cores

$$\text{Power} = 2 * C * V^2 * F$$

$$\text{Performance} = 2 * \text{Cores} * F$$

But decrease frequency by 50% (Note that $V \propto F$)

$$\text{Power} = 2 * C * V^2/4 * F/2$$

$$\text{Performance} = 2 * \text{Cores} * F/2$$



$$\text{Power} = C * V^2/4 * F$$

$$\text{Performance} = \text{Cores} * F$$

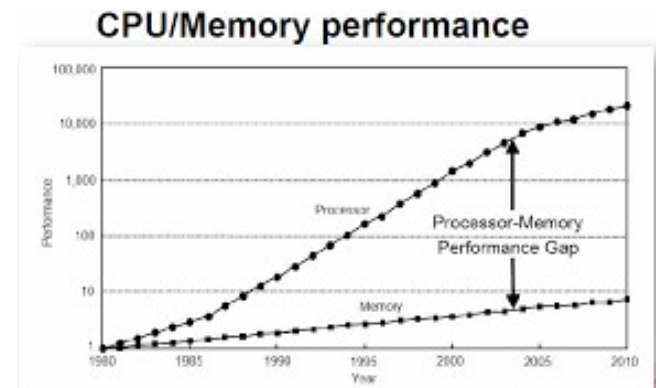
What Do we Conclude from Previous Slide?

- Using the transistors per chip to build more cores + lowering the frequency (i.e. clock) of each core → better performance with lower power

More cores with slower frequency is better than one big fat core with higher frequency, if we can make good use of them.

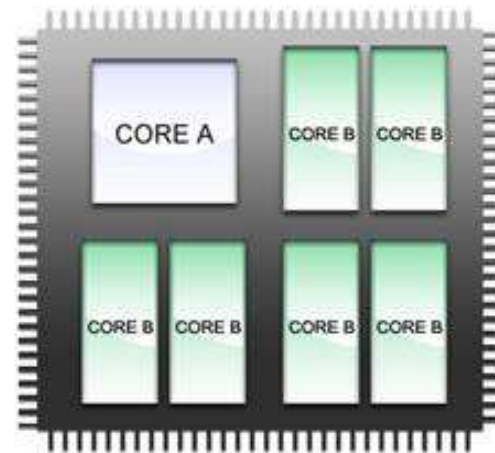
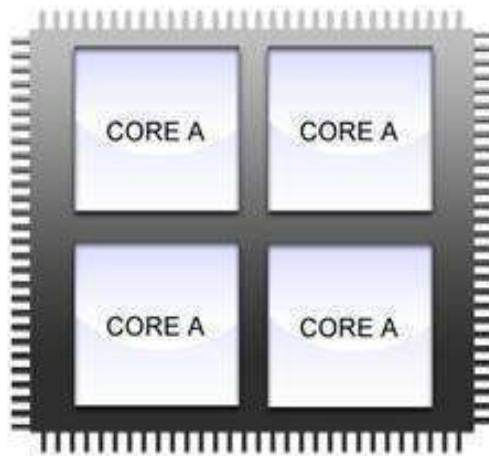
A Case for Multiple Processors

- Can exploit **different types of parallelism**
 - Parallelism among instructions
 - Parallelism among tasks/threads/processes
 - Processing different data at the same time
- Reduces power
- An effective way to **hide memory latency**
- Simpler cores
 - = easier to design and test
 - = higher yield
 - = lower cost



An intelligent solution

- Instead of designing and building faster microprocessors, put multiple cores on a single integrated circuit.



Now it's up to the programmers

- Adding more cores doesn't help much if programmers aren't aware of them...
- ... or don't know how to use them.
- Serial programs don't benefit from this approach (in most cases).



The Need for Parallel Programming

Parallel computing: using multiple processors in parallel to solve problems more quickly than with a single processor

Examples of parallel machines:

- A cluster computer** that contains multiple PCs combined together with a high speed network
- A shared memory multiprocessor (SMP)** by connecting multiple processors to a single memory system
- A Chip Multi-Processor (i.e. multicore) (CMP)** contains multiple processors (called cores) on a single chip

Attempts to Make Multicore Programming Easy

- **1st idea:** The right computer language would make parallel programming straightforward
 - **Result so far:** Some languages made parallel programming easier, but none has made it as fast, efficient, and flexible as traditional sequential programming.

Attempts to Make Multicore Programming Easy

- **2nd idea:** If you just design the hardware properly, parallel programming would become easy.
 - **Result so far:** no one has yet succeeded!

Attempts to Make Multicore Programming Easy

- **3rd idea:** Write software that will automatically parallelize existing sequential programs.
 - **Result so far:** Success here is inversely proportional to the number of cores!

Parallelizing a sequential program is not very easy!

- It is not about parallelizing every step of the sequential program.
- Maybe we need a totally new algorithm.
- Our parallelization strategy also depends on the software we try to parallelize!

Example


- Compute n values and add them together.
- Serial solution:

```
sum = 0;
for (i = 0; i < n; i++) {
    x = Compute_next_value(. . .);
    sum += x;
}
```

Example (cont.)

- We have p cores, p much smaller than n .
- Each core performs a partial sum of approximately n/p values.

```
my_sum = 0;  
my_first_i = . . . ;  
my_last_i = . . . ;  
for (my_i = my_first_i; my_i < my_last_i; my_i++) {  
    my_x = Compute_next_value( . . . );  
    my_sum += my_x;  
}
```



Each core uses its own private variables and executes this block of code independently of the other cores.

Example (cont.)

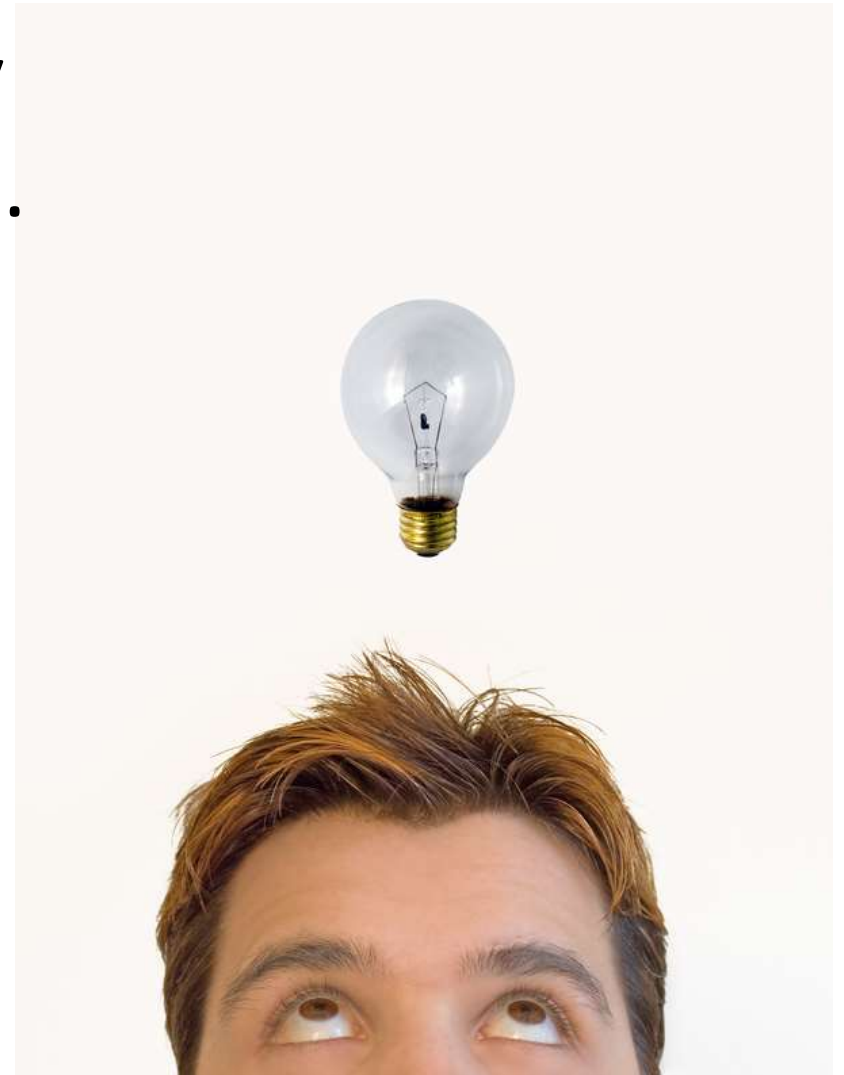
- Once all the cores are done computing their private `my_sum`, they form a global sum by sending results to a designated "master" core which adds the final result.

Example (cont.)

```
if (I'm the master core) {  
    sum = my_x;  
    for each core other than myself {  
        receive value from core;  
        sum += value;  
    }  
} else {  
    send my_x to the master;  
}
```

But wait!

There's a much better way
to compute the global sum.



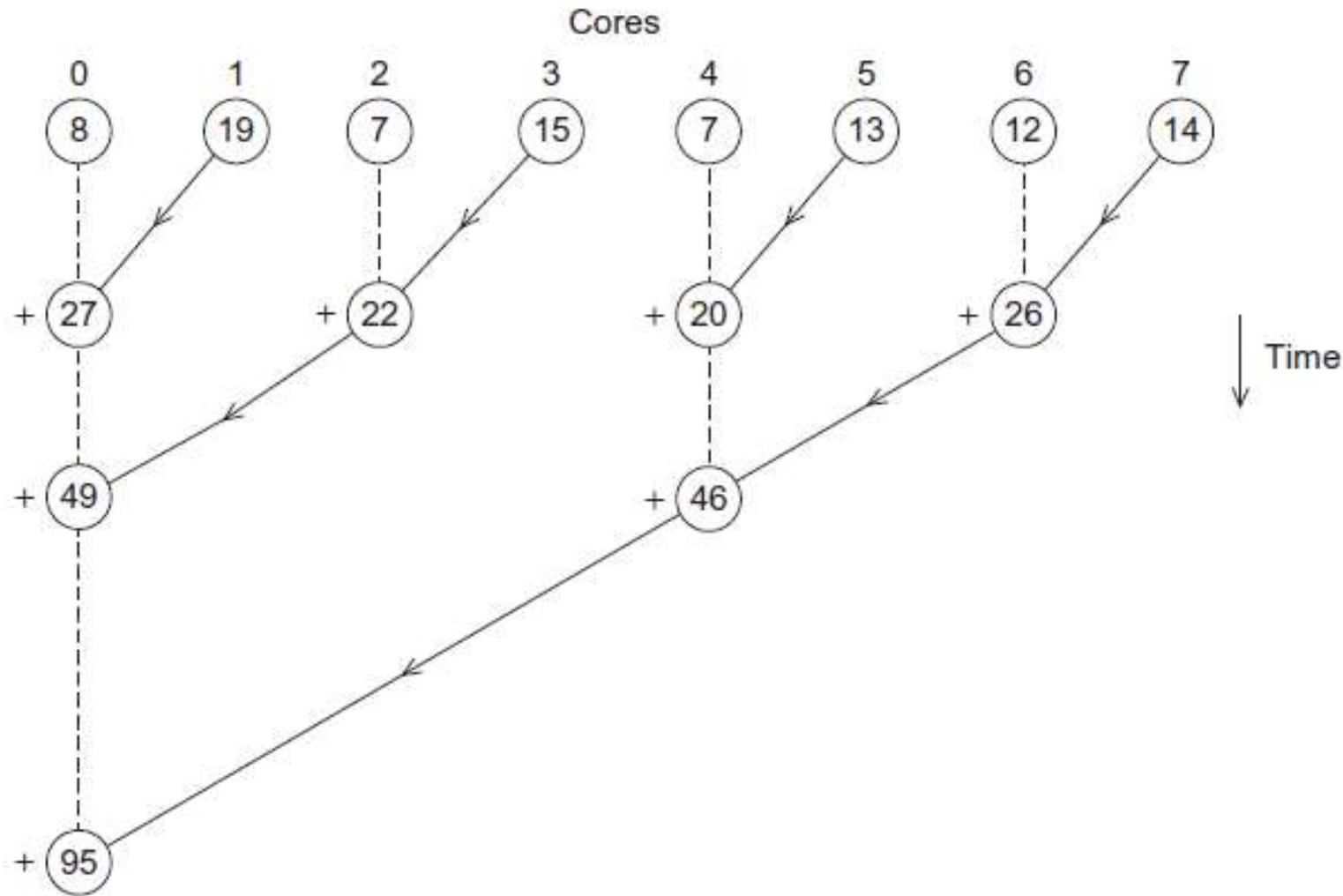
Better parallel algorithm

- Don't make the master core do all the work.
- Share it among the other cores.
- Pair the cores so that core 0 adds its result with core 1's result.
- Core 2 adds its result with core 3's result, etc.
- Work with odd and even numbered pairs of cores.

Better parallel algorithm (cont.)

- Repeat the process now with only the evenly ranked cores.
- Core 0 adds result from core 2.
- Core 4 adds the result from core 6, etc.
- Now cores divisible by 4 repeat the process, and so forth, until core 0 has the final result.

Multiple cores forming a global sum



Analysis

- In the first version, the master core performs 7 receives and 7 additions.
- In the second version, the master core performs 3 receives and 3 additions.
- The improvement is more than a factor of 2.

Analysis (cont.)

- The difference is more dramatic with a larger number of cores.
- If we have 1000 cores:
 - The first example would require the master to perform 999 receives and 999 additions.
 - The second example would only require 10 receives and 10 additions.
- That's an improvement of almost a factor of 100!!

Two Ways Of Thinking ... And one Strategy!

- Strategy: **Partitioning!**
- Two ways of thinking:
 - **Task-parallelism**
 - **Data-parallelism**
- Some constraints:
 - communication
 - memory access
 - load balancing
 - synchronization

Conclusions

- Due to technology constraints, we moved to multicore processors.
- Parallel programming is now a must → The free lunch is over!
- There are different flavors of parallel hardware that we will discuss and also many flavors of parallel programming languages that we will deal with.