上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

CS4302-01
Parallel and Distributed Computing

Lecture 1 Introduction

Zhuoran Song

2023/9/12

Course Details

- **Time**: *Tues 14:00-15:40am, Fri 10:00-11:40am*
- **Location:** *东中院3-206*
- **Course Website:** *Canvas*
- **Instructor:** *Zhuoran Song, songzhuoran@sjtu.edu.cn*
- **TA: TBD**
- **Textbook:** *"An Introduction to Parallel Programming" - by Peter Pacheco*

  "Programming Massively Parallel Processors, A Hands-on Approach" –by David Kirk and Wen-mei Hwu

  "通用图形处理器设计—GPGPU编程模型和架构原理" - by 景乃锋、柯晶、梁晓峣
- **Grades:**

Homework (40%), Project (55%), Presence (5%)

Self Introduction

宋卓然，上海交通大学计算机科学与工程系助理教授。博士毕业于上海交通大学。目前发表会议/期刊论文共29篇，其中包括9篇CCF-A论文：ISCA、MICRO、TCAD、DAC。申请发明专利共10项，以第一作者申请专利4项，授权专利共3项

梁晓峣，上海交通大学计算机科学与工程系教授。博士毕业于美国哈佛大学。研究方向包括计算机体系结构，AI芯片设计以及GPU架构等。发表论文100余篇，引用超1600次，包括国际顶级学术会议ISCA, HPCA, MICRO, ISSCC，DAC，ICCAD等，在计算机体系结构三大顶会共发表12篇论文，两次入选计算机体系结构年度最佳论文（IEEE MICRO TOP PICKS）。

Course Objectives

- **Study the state-of-art multicore processor architectures**
  - Why are the latest processors turning into multicore
  - What is the basic computer architecture to support multicore
- **Learn how to program parallel processors and systems**
  - Learn how to think in parallel and write correct parallel programs
  - Achieve performance and scalability through understanding of architecture and software mapping
- **Significant hands-on programming experience**
  - Develop real applications on real hardware
- **Discuss the current parallel computing context**
  - What are the drivers that make this course timely
  - Contemporary programming models and architectures, and where is the field going

Course Importance

- **Multi-core and many-core era is here to stay**
  - Why? Technology Trends
- **Many programmers will be developing parallel software**
  - But still not everyone is trained in parallel programming
  - Learn how to put all these vast machine resources to the best use!
- **Useful for**
  - Joining the work force
  - Graduate school
- **Our focus**
  - Teach core concepts
  - Use common and novel programming models
  - Discuss broader spectrum of parallel computing

Course Arrangement

In class:

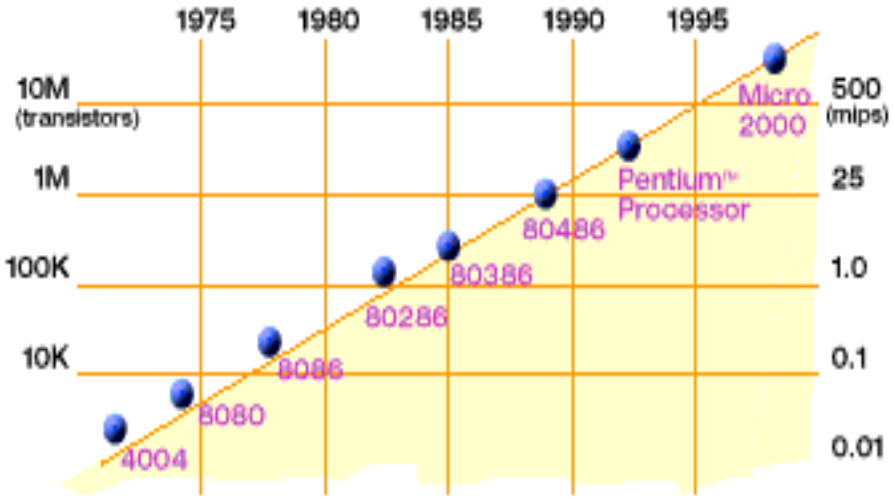
- **1 Lecture for Introduction**
- **2 Lectures for Parallel Computer Architecture/Distributed System**
- **3 Lectures for OpenMP**
- **3 Lectures for GPU Architecture**
- **3 Lectures for CUDA**
- **1 lecture for AI Processors and Programming**
- **1 Lecture for Project Introduction**

There are no in-class lectures after the 8th week. All students are expected to finish their course projects at home of in the lab.

What is Parallel Computing

- **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 servers 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 (CMP) contains multiple processors (called cores) on a single chip
- **Concurrent execution comes from desire for performance; unlike the inherent concurrency in a multi-user distributed system**
- **Why are we adding an undergraduate course now?**
  - Because the entire computing industry has bet on parallelism
  - There is a desperate need for parallel programmers

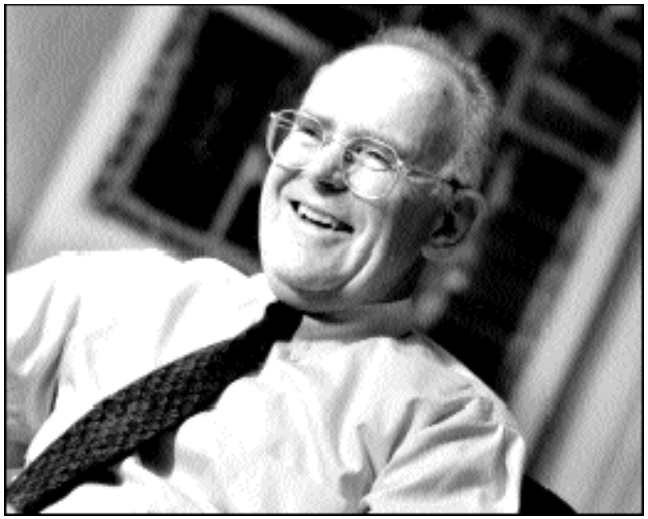
Microprocessor Capacity



2X transistors/Chip Every 1.5 years

Called "Moore's Law"

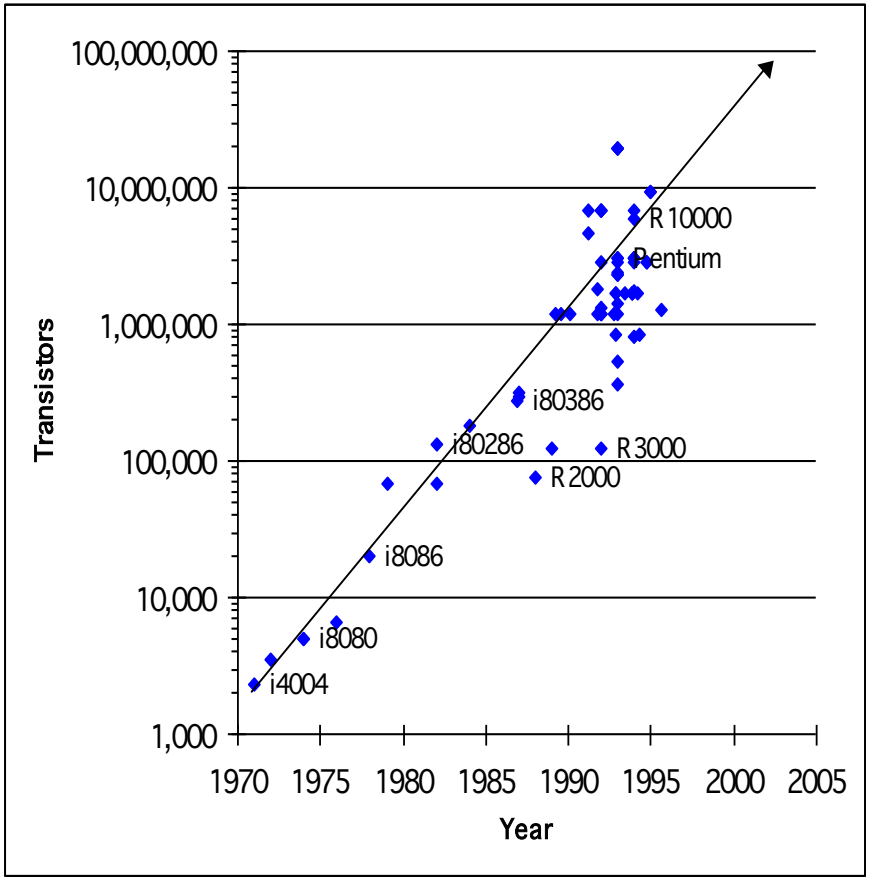
Microprocessors have become smaller, denser, and more powerful.



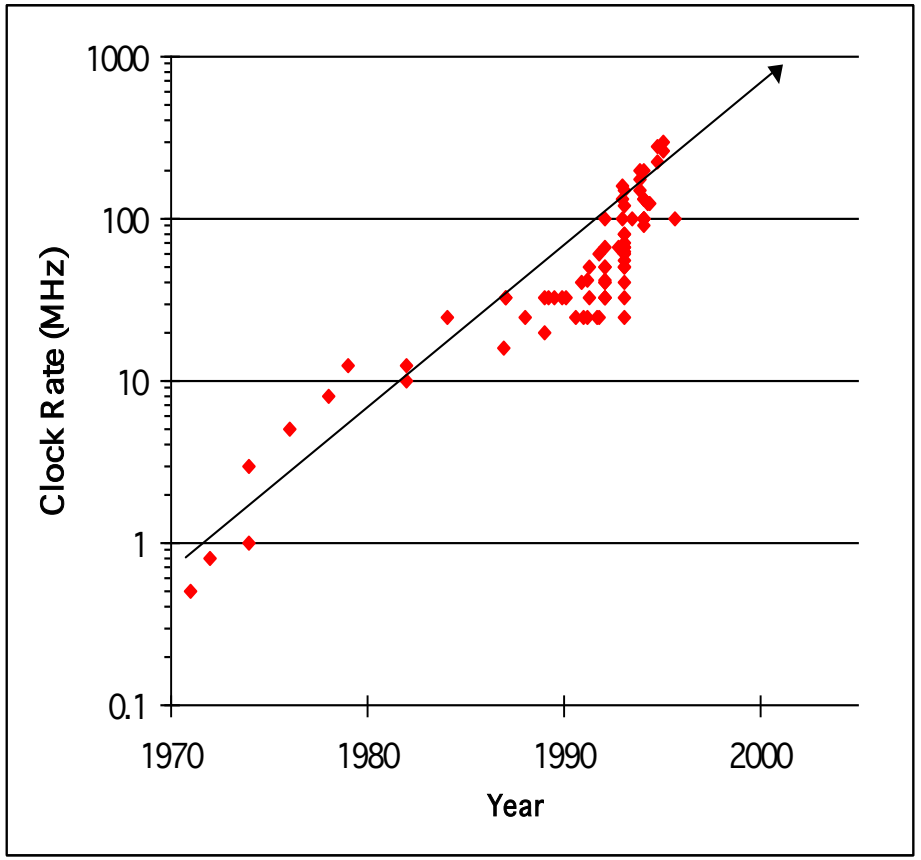
Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Microprocessor Speed

Growth in transistors per chip



Increase in clock rate

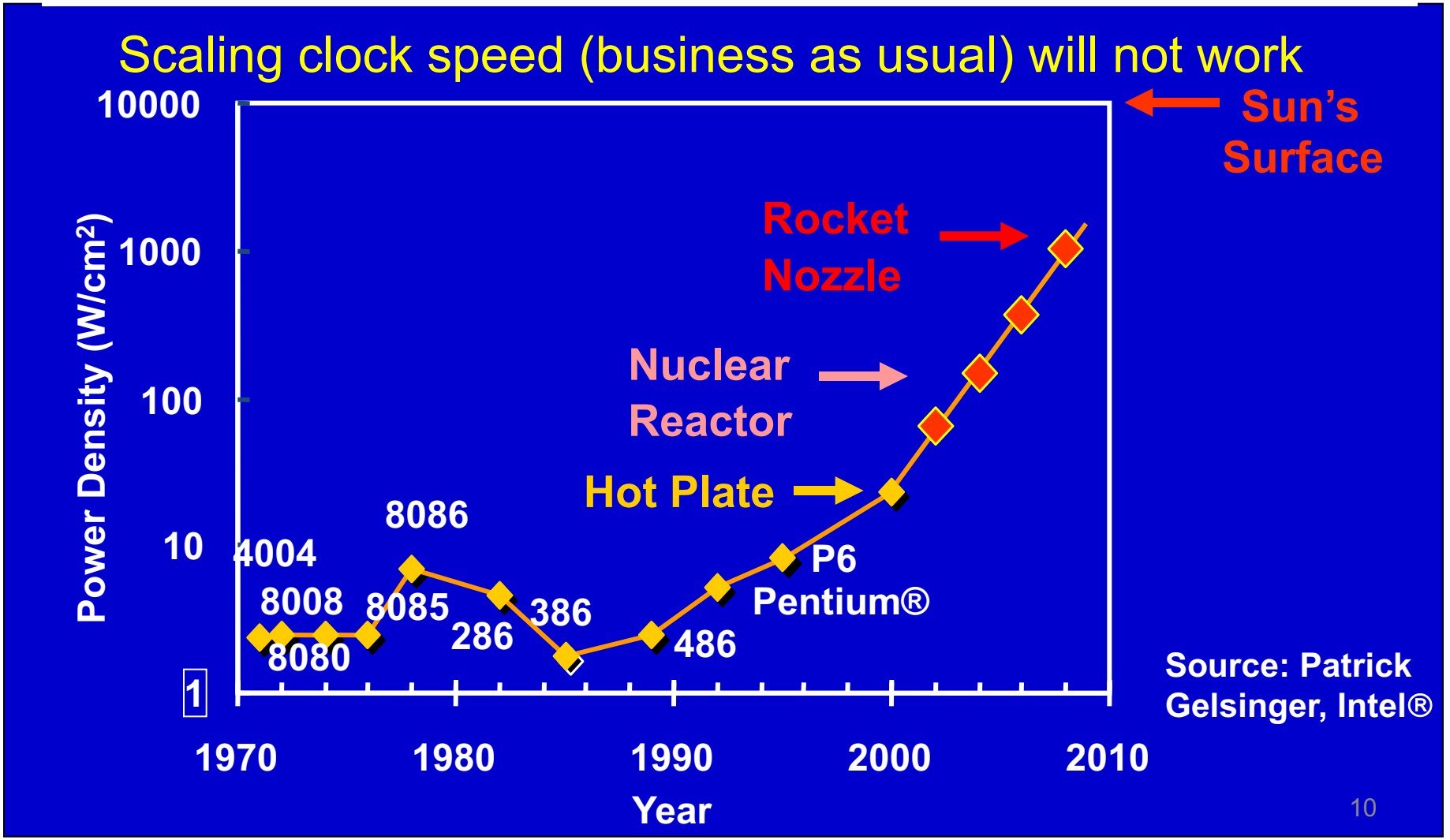


Why bother with parallel programming? Just wait a year or two…

Limit #1: Power Density

Can soon put more transistors on a chip than can afford to turn on.

-- Patterson '07



Parallelism Saves Power

- **Exploit explicit parallelism for reducing power**

  Power = (C * V^2 * F) Performance = Cores * F

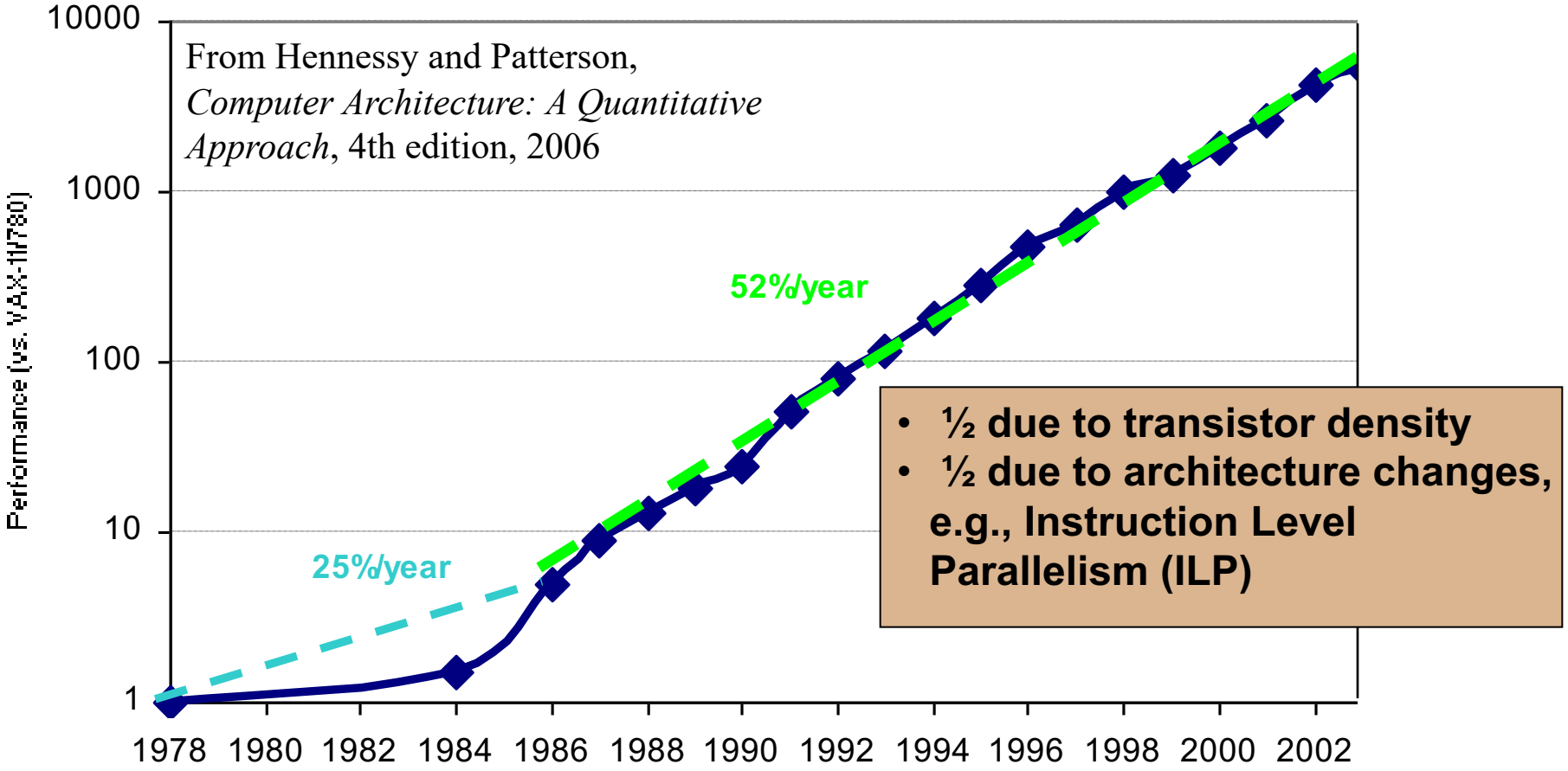
  Capacitance Voltage Frequency

- **Using additional cores**
  - *Increase density (= more transistors = more capacitance)*
  - *Increase cores (2x), but decrease frequency (1/2): same performance at (1/4) the power*

- **Additional benefits**
  - *Small/simple cores → more predictable performance*

Limit #2: ILP Tapped Out

Application performance was increasing by 52% per year as measured by the SpecInt benchmarks here



- **VAX : 25%/year 1978 to 1986**
- **RISC + x86: 52%/year 1986 to 2002**

Limit #2: ILP Tapped Out

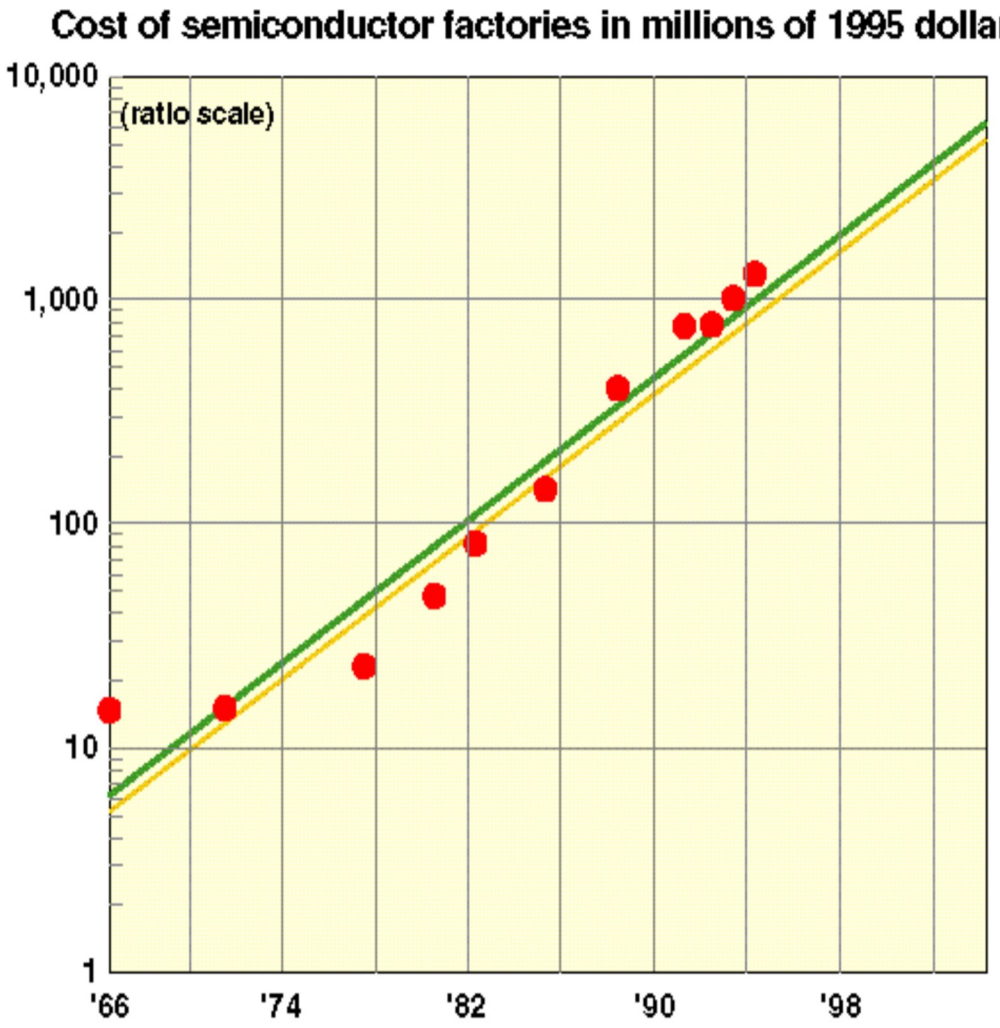
- **Superscalar (SS) designs were the state of the art; many forms of parallelism not visible to programmer**
  - *multiple instruction issue*
  - *dynamic scheduling: hardware discovers parallelism between instructions*
  - *speculative execution: look past predicted branches*
  - *non-blocking caches: multiple outstanding memory ops*

- **You may have heard of these before, but you haven't needed to know about them to write software**

- **Unfortunately, these sources have been used up**

Limit #3: Chip Yield

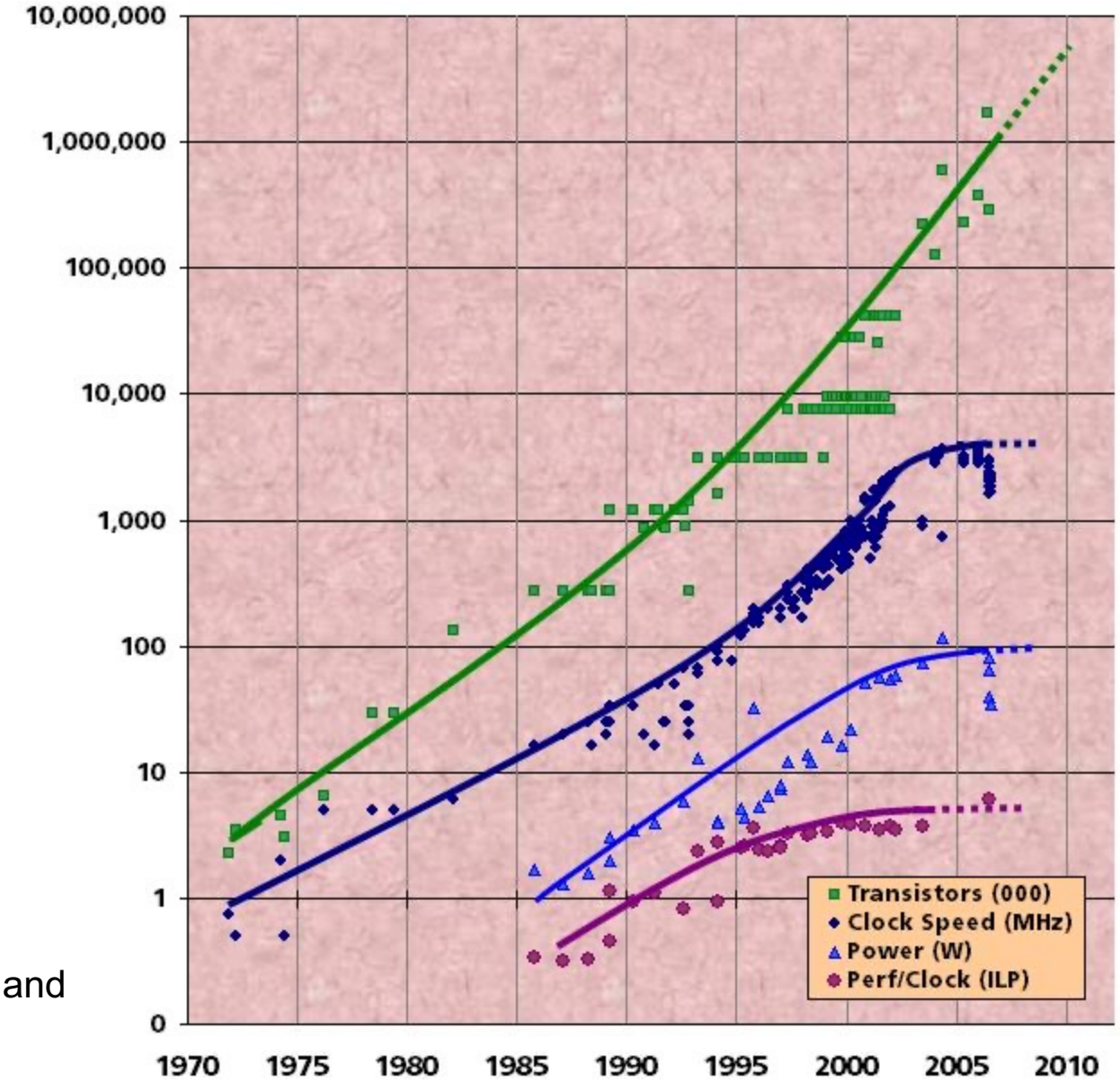
Manufacturing costs and yield problems limit use of density



- **Moore's (Rock's) 2nd law: fabrication costs go up**
- **Yield (% usable chips) drops**
- **Parallelism can help**
  - *More smaller, simpler processors are easier to design and validate*
  - *Can use partially working chips:*
  - *E.g., Cell processor (PS3) is sold with 7 out of 8 "on" to improve yield*

Current Situation

- Chip density is continuing increasing
  - *Clock speed is not*
  - *Number of processor cores may double instead*
- There is little or no hidden parallelism (ILP) to be found
- Parallelism must be exposed to and managed by software



Source: Intel, Microsoft (Sutter) and Stanford (Olukotun, Hammond)

Multicore In Products

- All microprocessor companies switch to MP (2X CPUs / 2 yrs)

Manufacturer/Year	AMD/'05	Intel/'06	IBM/'04	Sun/'07
Processors/chip	2	2	2	8
Threads/Processor	1	2	2	16
Threads/chip	2	4	4	128

And at the same time,

- The STI Cell processor (PS3) has 8 cores
- The NVidia Graphics Processing Unit (GPU) has 1024 cores
- Intel has demonstrated an Xeon-Phi chip with 60 cores

Paradigm Shift

- What do we do with all the transistors?
  - *Movement away from increasingly complex processor design and faster clocks*
  - *Replicated functionality (i.e., parallel) is simpler to design*
  - *Resources more efficiently utilized*
  - *Huge power management advantages*

All Computers are Parallel Computers.

Why Parallelism

- **These arguments are no long theoretical**
- **All major processor vendors are producing multicore chips**
  - *Every machine will soon be a parallel machine*
  - *All programmers will be parallel programmers???*
- **New software model**
  - *Want a new feature? Hide the "cost" by speeding up the code first*
  - *All programmers will be performance programmers???*
- **Some may eventually be hidden in libraries, compilers, and high level languages**
  - *But a lot of work is needed to get there*
- **Big open questions**
  - *What will be the killer apps for multicore machines*
  - *How should the chips be designed, and how will they be programmed?*

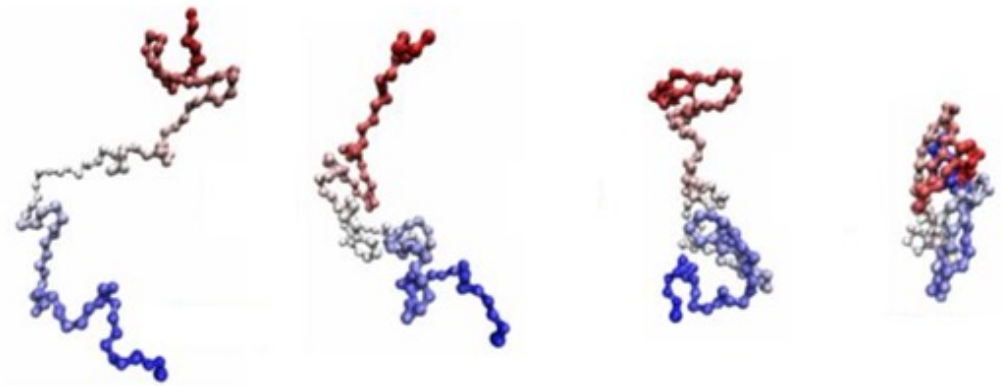
Scientific Simulation

- **Traditional scientific and engineering paradigm**
  - *Do theory or paper design.*
  - *Perform experiments or build system.*

- **Limitations:**
  - *Too difficult -- build large wind tunnels.*
  - *Too expensive -- build a throw-away passenger jet.*
  - *Too slow -- wait for climate or galactic evolution.*
  - *Too dangerous -- weapons, drug design, climate experimentation.*

- **Computational science paradigm**
  - *Use high performance computer systems to simulate the phenomenon*
  - *Base on known physical laws and efficient numerical methods.*

Scientific Simulation

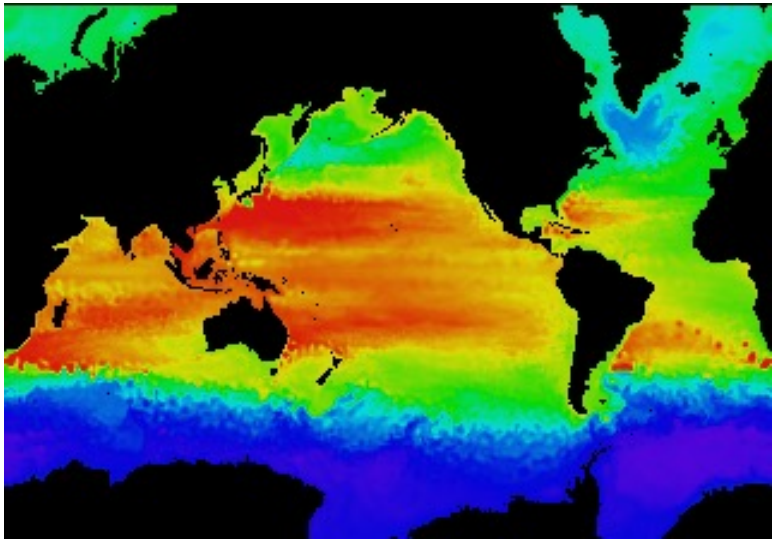


Example

- **Problem is to compute**

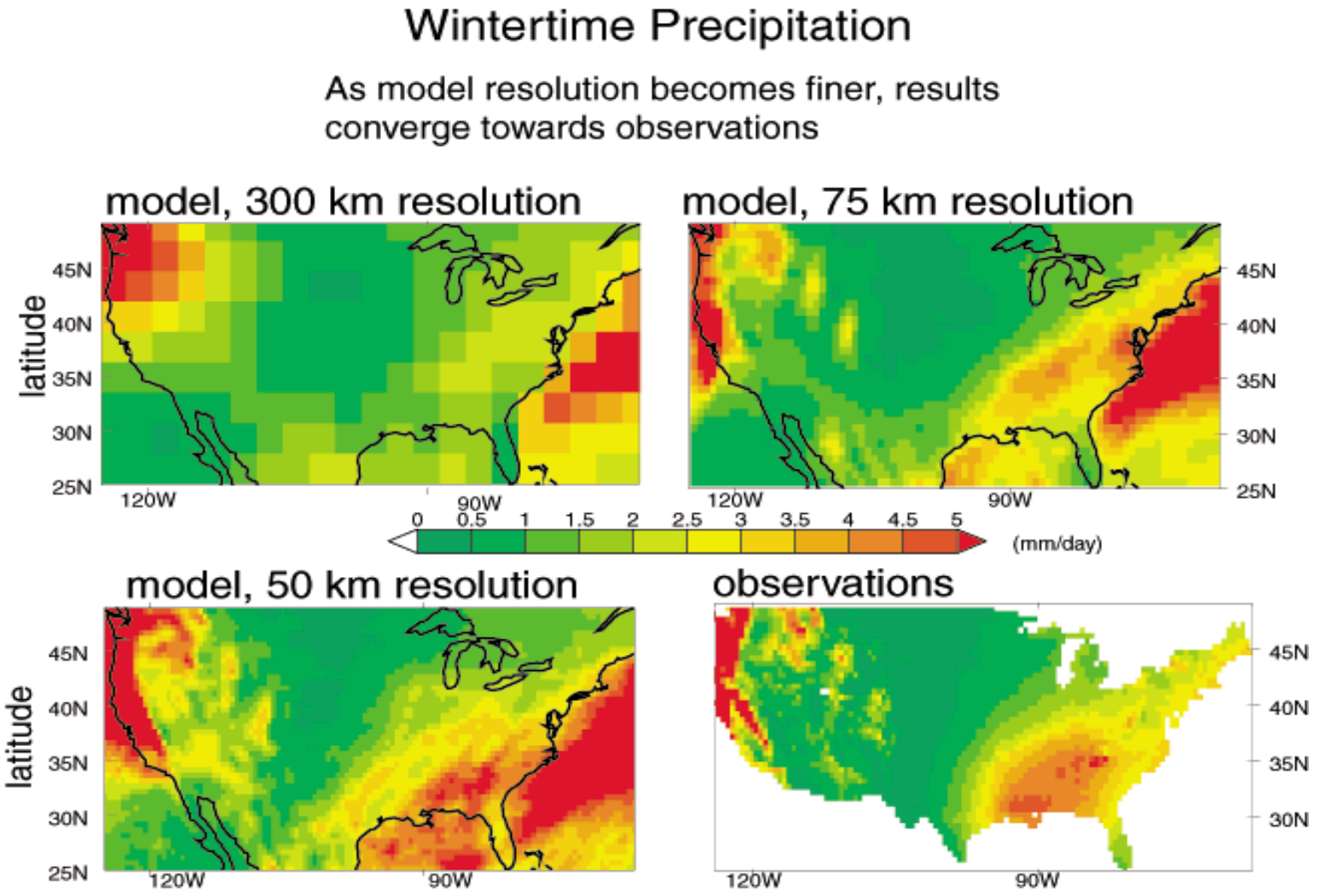
  f(latitude, longitude, elevation, time) → *temperature, pressure, humidity, wind velocity*

- **Approach**
  - *Discretize the domain, e.g., a measurement point every 10 km*
  - *Devise an algorithm to predict weather at time t+δt given t*



Source: http://www.epm.ornl.gov/chammp/chammp.html

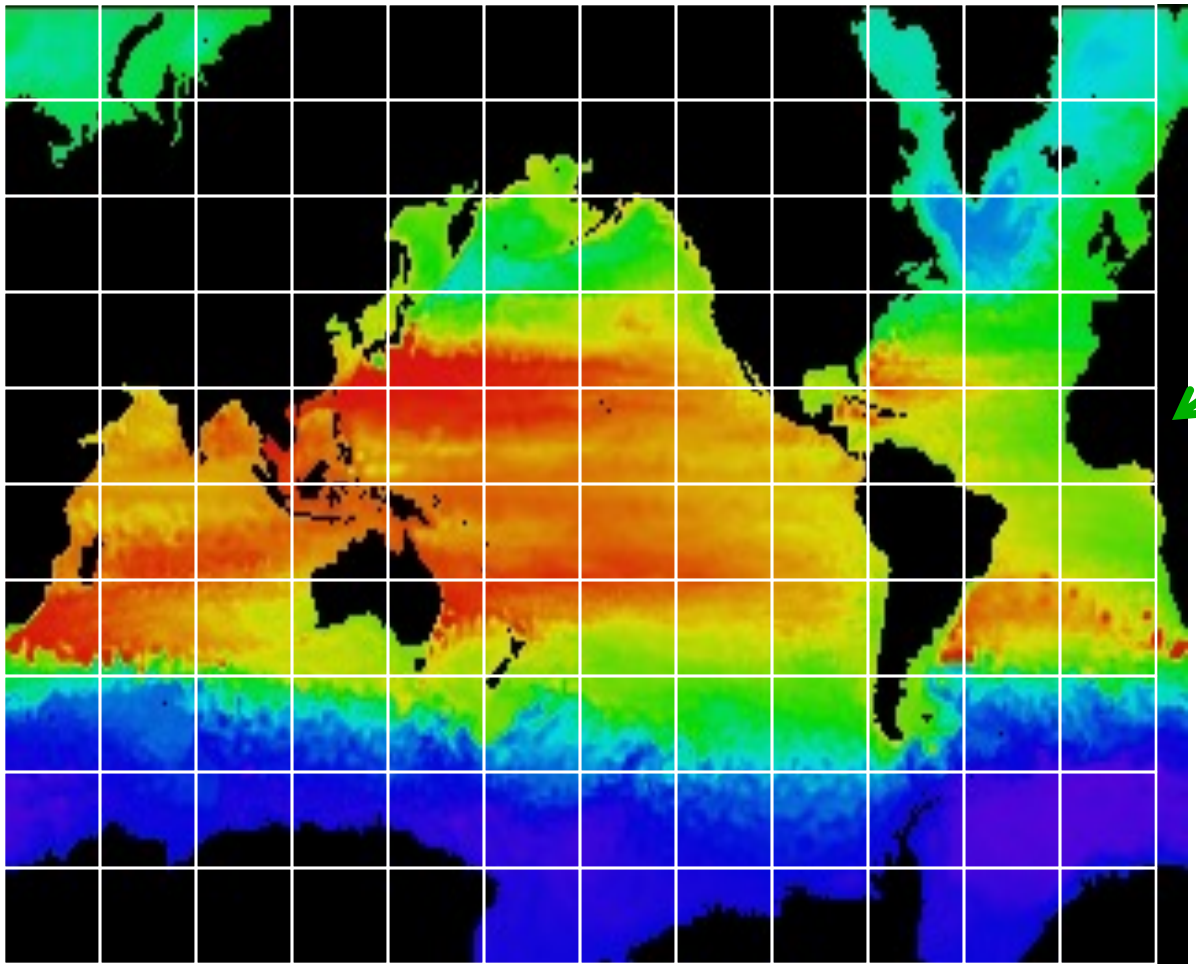
Example


Wintertime Precipitation
As model resolution becomes finer, results converge towards observations
model, 300 km resolution
model, 75 km resolution
model, 50 km resolution
observations
latitude
45N
40N
35N
30N
25N
120W
90W
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
(mm/day)

Steps in Climate Modeling

- **Discretize physical or conceptual space into a grid**
  - *Simpler if regular, may be more representative if adaptive*
- **Perform local computations on grid**
  - *Given yesterday's temperature and weather pattern, what is today's expected temperature?*
- **Communicate partial results between grids**
  - *Contribute local weather result to understand global weather pattern.*
- **Repeat for a set of time steps**
  - *Possibly perform other calculations with results*
  - *Given weather model, what area should evacuate for a hurricane?*

Steps in Climate Modeling



Another processor computes this part in parallel

One processor computes this part

Processors in adjacent blocks in the grid communicate their result.

The Need for Scientific Simulation

- **Scientific simulation will continue to push on system requirements**
  - *To increase the precision of the result*
  - *To get to an answer sooner (e.g., climate modeling, disaster modeling)*

- **Major countries will continue to acquire systems of increasing scale**
  - *For the above reasons*
  - *And to maintain competitiveness*

Commodity Devices

- More capabilities in software
- Integration across software
- Faster response
- More realistic graphics
- Computer vision
- AI

Approaches to Write Parallel Program

- **Rewrite serial programs so that they're parallel.**
  - *Sometimes the best parallel solution is to step back and devise an entirely new algorithm*

- **Write translation programs that automatically convert serial programs into parallel programs.**
  - *This is very difficult to do.*
  - *Success has been limited.*
  - *It is likely that the result will be a very inefficient program.*

Parallel Program Example

- Compute "n" values and add them together

- Serial solution

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

Parallel Program Example

- 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 it's own private variables and executes this block of code independently of the other cores.

Parallel Program Example

- After each core completes execution of the code, is a private variable my_sum contains the sum of the values computed by its calls to Compute_next_value.

- Ex., 8 cores, n = 24, then the calls to Compute_next_value return:

1,4,3, 9,2,8, 5,1,1, 5,2,7, 2,5,0, 4,1,8, 6,5,1, 2,3,9

Parallel Program Example

- 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.

```
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;
}
```

Parallel Program Example

Core	0	1	2	3	4	5	6	7
my_sum	8	19	7	15	7	13	12	14

Global sum

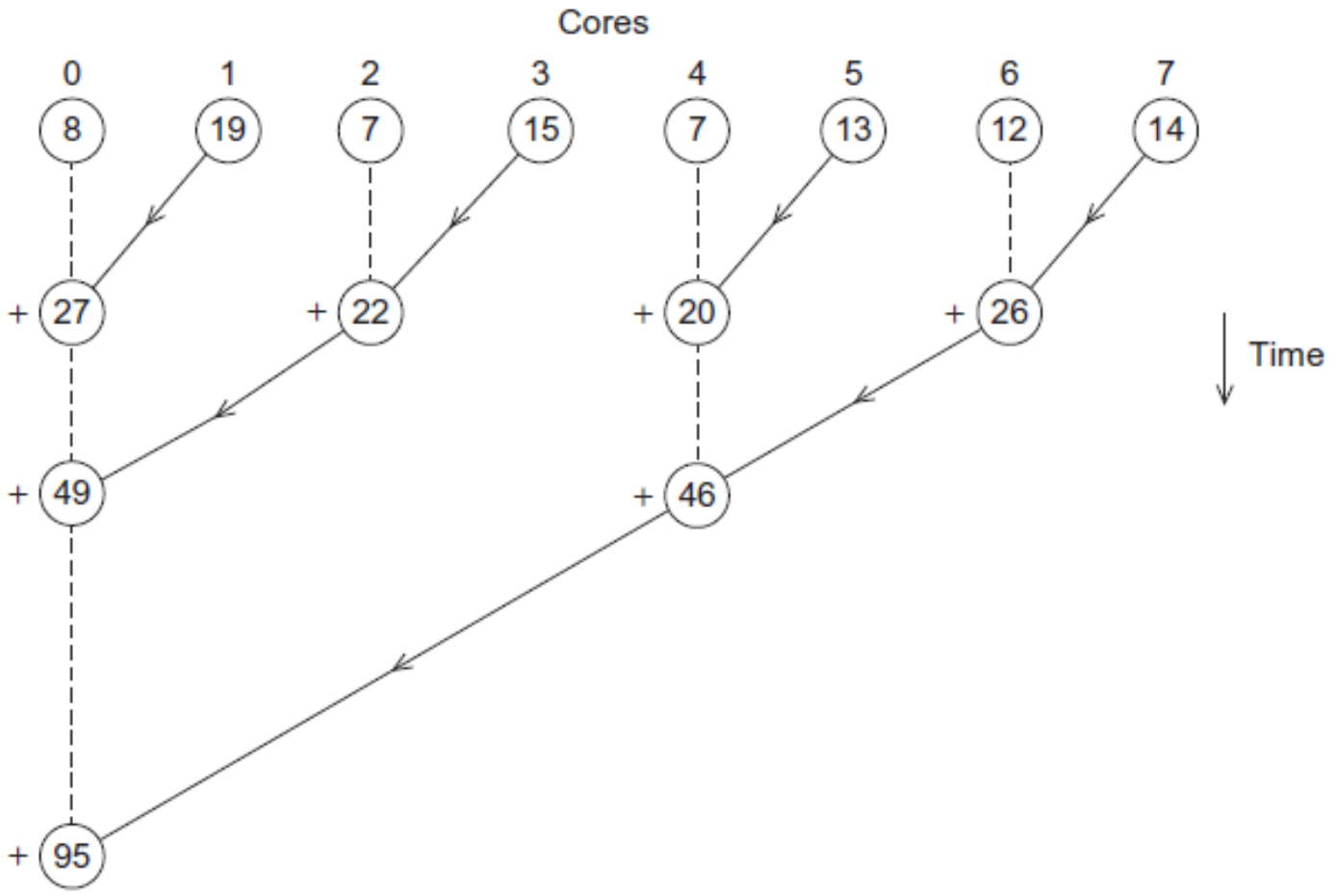
8 + 19 + 7 + 15 + 7 + 13 + 12 + 14 = 95

Core	0	1	2	3	4	5	6	7
my_sum	95	19	7	15	7	13	12	14

Better Parallel Program Example

- 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.
- 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.

Better Parallel Program Example



Better Parallel Program Example

- **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!**

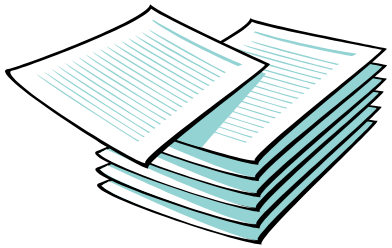
Types of Parallelism

- Task parallelism
  - *Partition various tasks carried out solving the problem among the cores.*
- Data parallelism
  - *Partition the data used in solving the problem among the cores.*
  - *Each core carries out similar operations on it's part of the data.*

Types of Parallelism

15 questions

300 exams

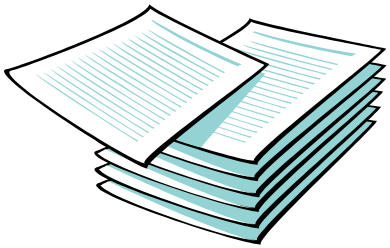


Types of Parallelism

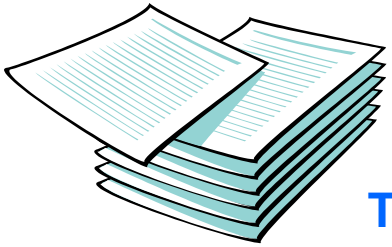
Types of Parallelism

Data Parallelism

TA#1

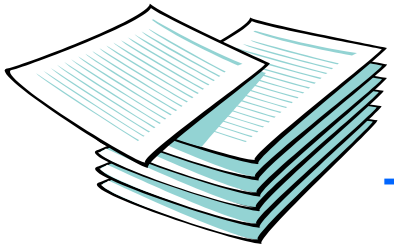


100 exams



TA#3

100 exams



TA#2

100 exams

Types of Parallelism

Task Parallelism

TA#1

Questions 1 - 5

TA#3

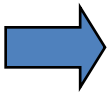
Questions 11 - 15

TA#2

Questions 6 - 10

Principles of Parallelism

- Finding enough parallelism (Amdahl’s Law)
- Granularity
- Locality
- Load balance
- Coordination and synchronization
- Performance modeling



All of these things makes parallel programming even harder than sequential programming.

Finding Enough Parallelism

- **Suppose only part of an application seems parallel**
- **Amdahl's law**
  - *let s be the fraction of work done sequentially, so (1-s) is fraction parallelizable*
  - *P = number of processors*

$$\text{Speedup}(P) = \text{Time}(1)/\text{Time}(P)$$

$$<= 1/(s + (1-s)/P)$$

$$<= 1/s$$

- **Even if the parallel part speeds up perfectly performance is limited by the sequential part**

Overhead of Parallelism

- **Given enough parallel work, this is the biggest barrier to getting desired speedup**

Parallelism overheads include

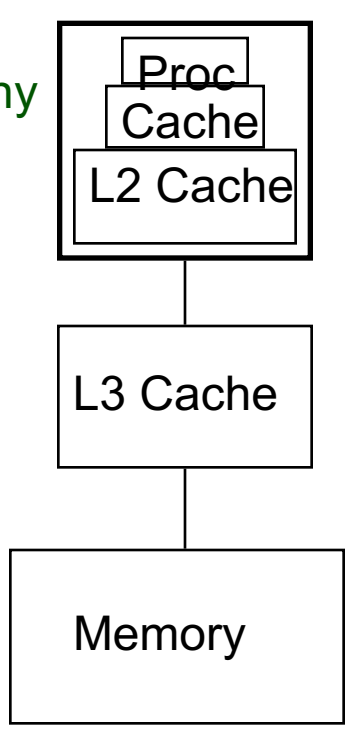
- *cost of starting a thread or process*
- *cost of communicating shared data*
- *cost of synchronizing*
- *extra (redundant) computation*

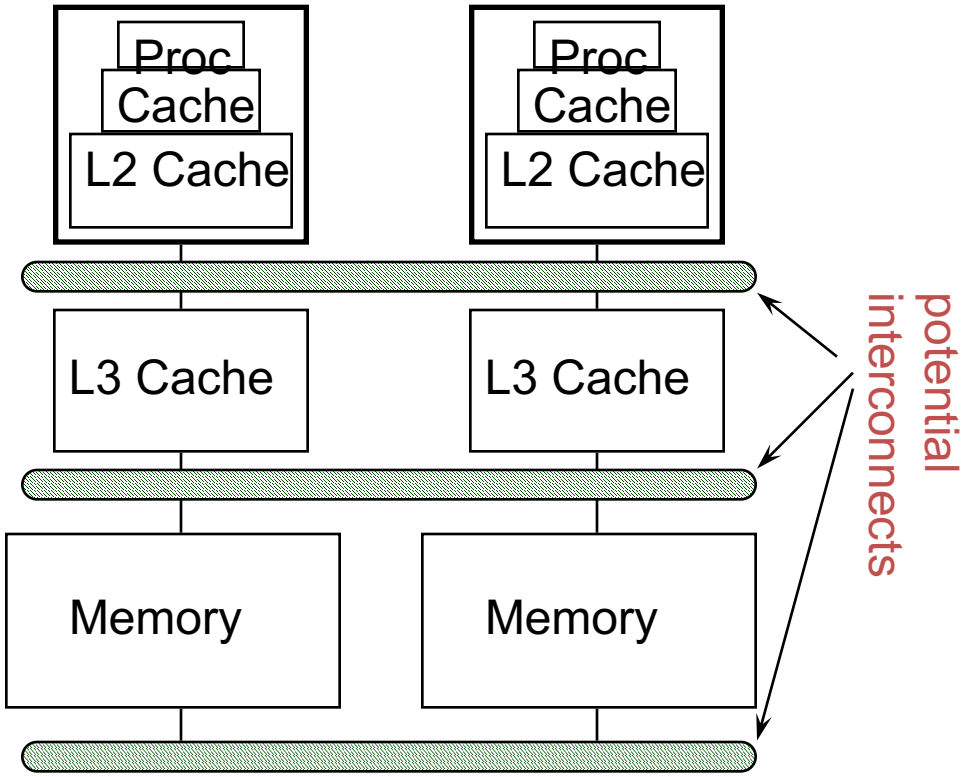
- **Each of these can be in the range of milliseconds (=millions of flops) on some systems**

- **Tradeoff: Algorithm needs sufficiently large units of work to run fast in parallel (I.e. large granularity), but not so large that there is not enough parallel work**

Locality

Conventional Storage Hierarchy



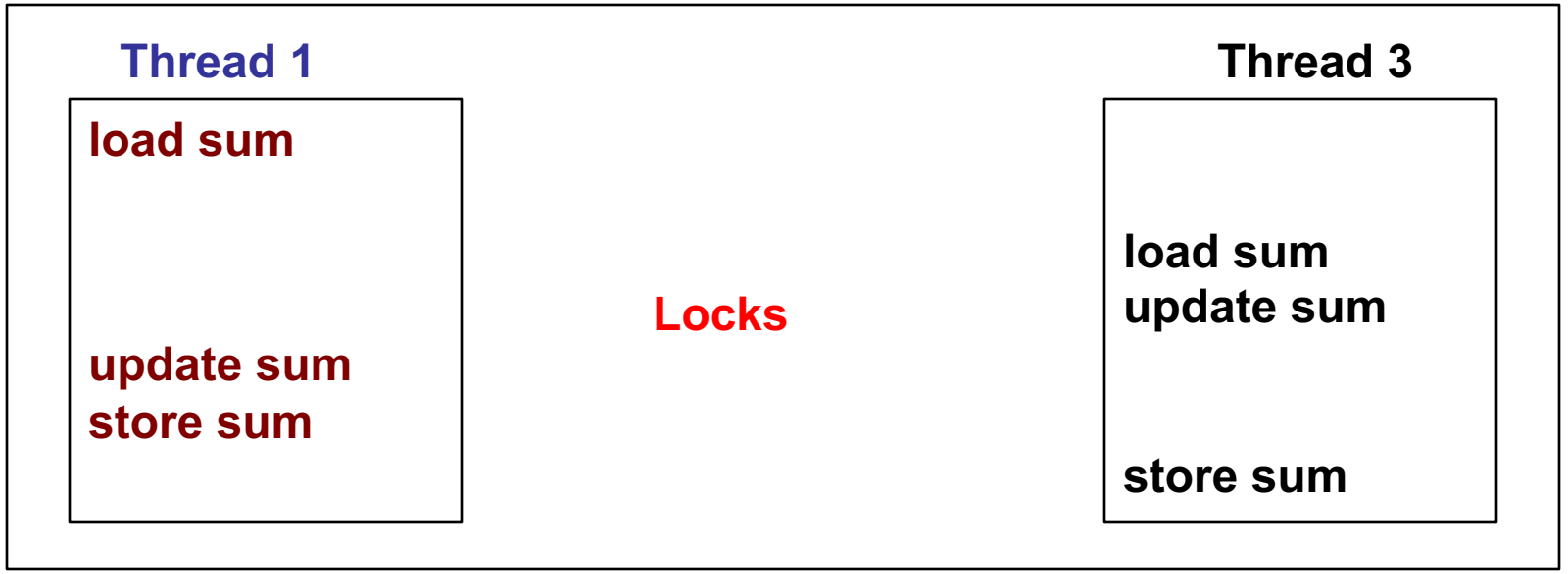


- Large memories are slow, fast memories are small
- Storage hierarchies are large and fast on average
- Parallel processors, collectively, have large, fast cache
  - *the slow accesses to "remote" data we call "communication"*
- Algorithm should do most work on local data

Load Balancing

- **Load imbalance is the time that some processors in the system are idle due to**
  - *insufficient parallelism (during that phase)*
  - *unequal size tasks*
- **Examples of the latter**
  - *adapting to "interesting parts of a domain"*
  - *tree-structured computations*
  - *fundamentally unstructured problems*
- **Algorithm needs to balance load**

Locks and Barriers



A **barrier** is used to block threads from proceeding beyond a program point until all of the participating threads has reached the barrier.