CSE 613: Parallel Programming

Department of Computer Science
SUNY Stony Brook
Spring 2012

"For over a decade prophets have voiced the contention that the organization of a single computer has reached its limits and that truly significant advances can be made only by interconnection of a multiplicity of computers."

— *Gene Amdahl, 1967*

Course Information

- **Lecture Time:** TuTh 2:20 pm - 3:40 pm
- **Location:** Computer Science 2129, West Campus

- **Instructor:** Rezaul A. Chowdhury
- **Office Hours:** TuTh 12:00 pm - 1:30 pm, 1421 Computer Science
- **Email:** rezaul@cs.stonybrook.edu

- **TA:** No idea!
- **TA Office Hours:** Same as above
- **TA Email:** Same as above

- **Class Webpage:**

  http://www.cs.sunysb.edu/~rezaul/CSE613-S12.html

Prerequisites

- **Required:** Background in algorithms analysis
  (e.g., CSE 373 or CSE 548)
- **Required:** Background in programming languages (C / C++)
- **Helpful but Not Required:** Background in computer architecture
- **Please Note:** This is not a course on
  - Programming languages
  - Computer architecture
- **Main Emphasis:** Parallel algorithms

Topics to be Covered

The following topics will be covered

- Analytical modeling of parallel programs
- Scheduling
- Programming using the message-passing paradigm and for shared address-space platforms
- Parallel algorithms for dense matrix operations, sorting, searching, graphs, computational geometry, and dynamic programming
- Concurrent data structures
- Transactional memory, etc.

Grading Policy

- Homeworks (three: lowest score 5%, others 10% each): 25%
- Exams (two: higher one 20%, lower one 10%): 30%
  - Midterm (in-class): March 27
  - Final (in-class): May 15
- Group project (one): 30%
  - Proposal (in-class): Feb 28
  - Progress report (in-class): April 10
  - Final presentation (in-class): May 8 - 10
- Scribe note (one lecture): 10%
- Class participation & attendance: 5%

Programming Environment

This course is supported by educational grants from

- Extreme Science and Engineering Discovery Environment (XSEDE): https://www.xsede.org
- Amazon Web Services (AWS): http://aws.amazon.com

We will use XSEDE for homeworks/projects involving

- Shared-memory parallelism
- Distributed-memory parallelism

AWS will be used for those involving (mainly for CSE590)

- GPGPUs
- MapReduce

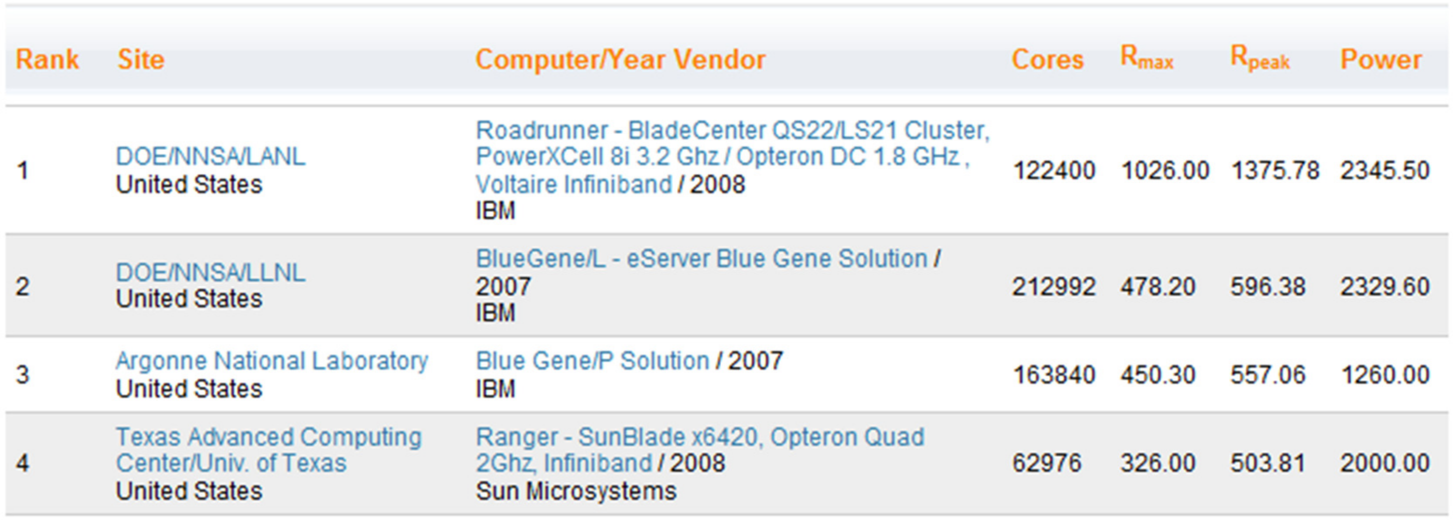
Programming Environment

On XSEDE we have access to

– Ranger: ≈ 4,000 compute nodes with 16 cores/node

– Lonestar 4: ≈ 2,000 compute nodes with 12 cores/node

World's Most Powerful Supercomputers in June, 2008
(www.top500.org)



Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
1	DOE/NNSA/LANL United States	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz , Voltaire Infiniband / 2008 IBM	122400	1026.00	1375.78	2345.50
2	DOE/NNSA/LLNL United States	BlueGene/L - eServer Blue Gene Solution / 2007 IBM	212992	478.20	596.38	2329.60
3	Argonne National Laboratory United States	Blue Gene/P Solution / 2007 IBM	163840	450.30	557.06	1260.00
4	Texas Advanced Computing Center/Univ. of Texas United States	Ranger - SunBlade x6420, Opteron Quad 2Ghz, Infiniband / 2008 Sun Microsystems	62976	326.00	503.81	2000.00

Textbooks

Required

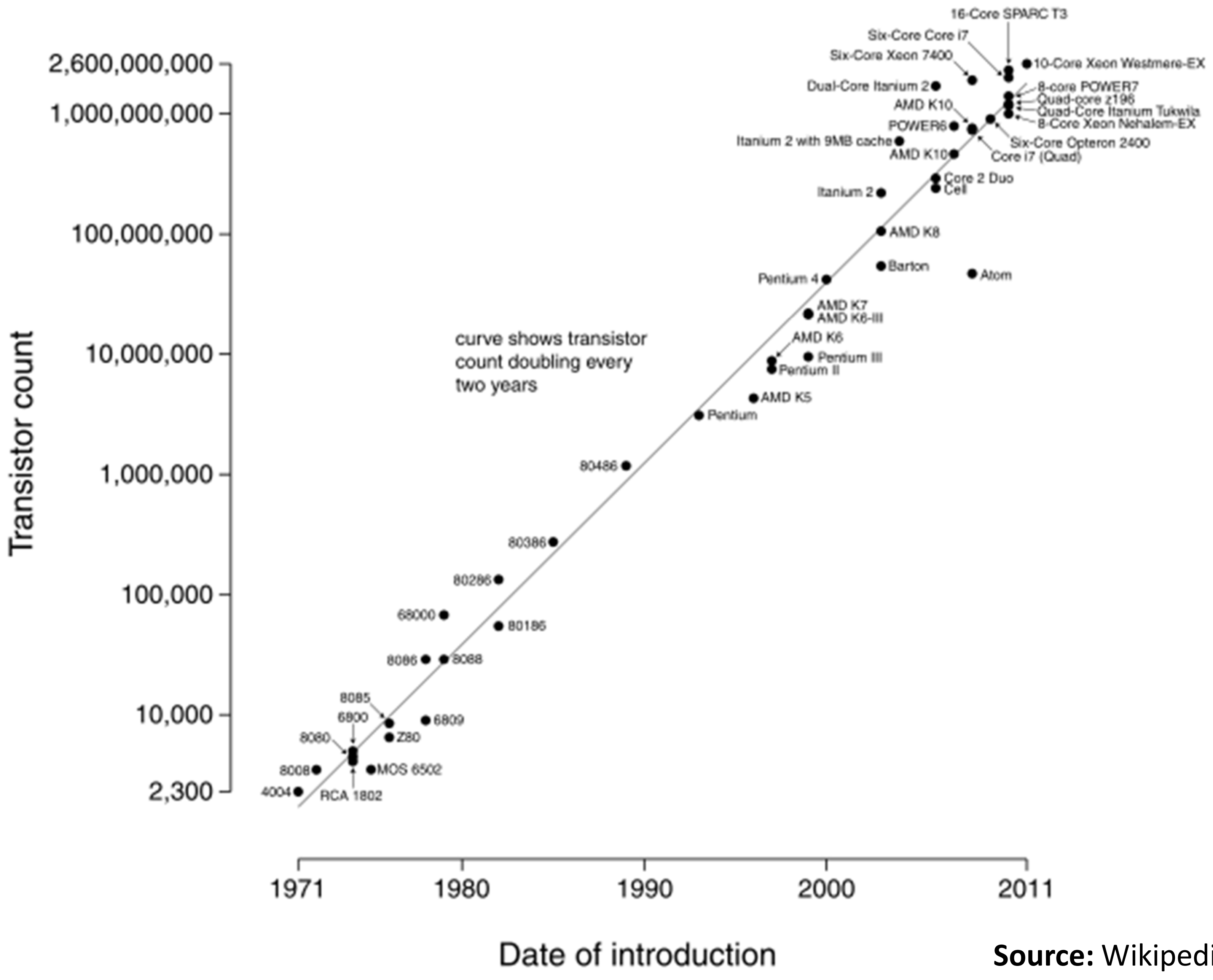
- A. Grama, G. Karypis, V. Kumar, and A. Gupta. ***Introduction to Parallel Computing*** (2nd Edition), Addison Wesley, 2003.

Recommended

- M. Herlihy and N. Shavit. ***The Art of Multiprocessor Programming*** (1st Edition), Morgan Kaufmann, 2008.
- F. Gebali. ***Algorithms and Parallel Computing*** (1st Edition), Wiley, 2011.
- T. Cormen, C. Leiserson, R. Rivest, and C. Stein. ***Introduction to Algorithms*** (3rd Edition), MIT Press, 2009. (chapter 27 on *Multithreaded Algorithms*)
- P. Pacheco. ***Parallel Programming with MPI*** (1st Edition), Morgan Kaufmann, 1996.

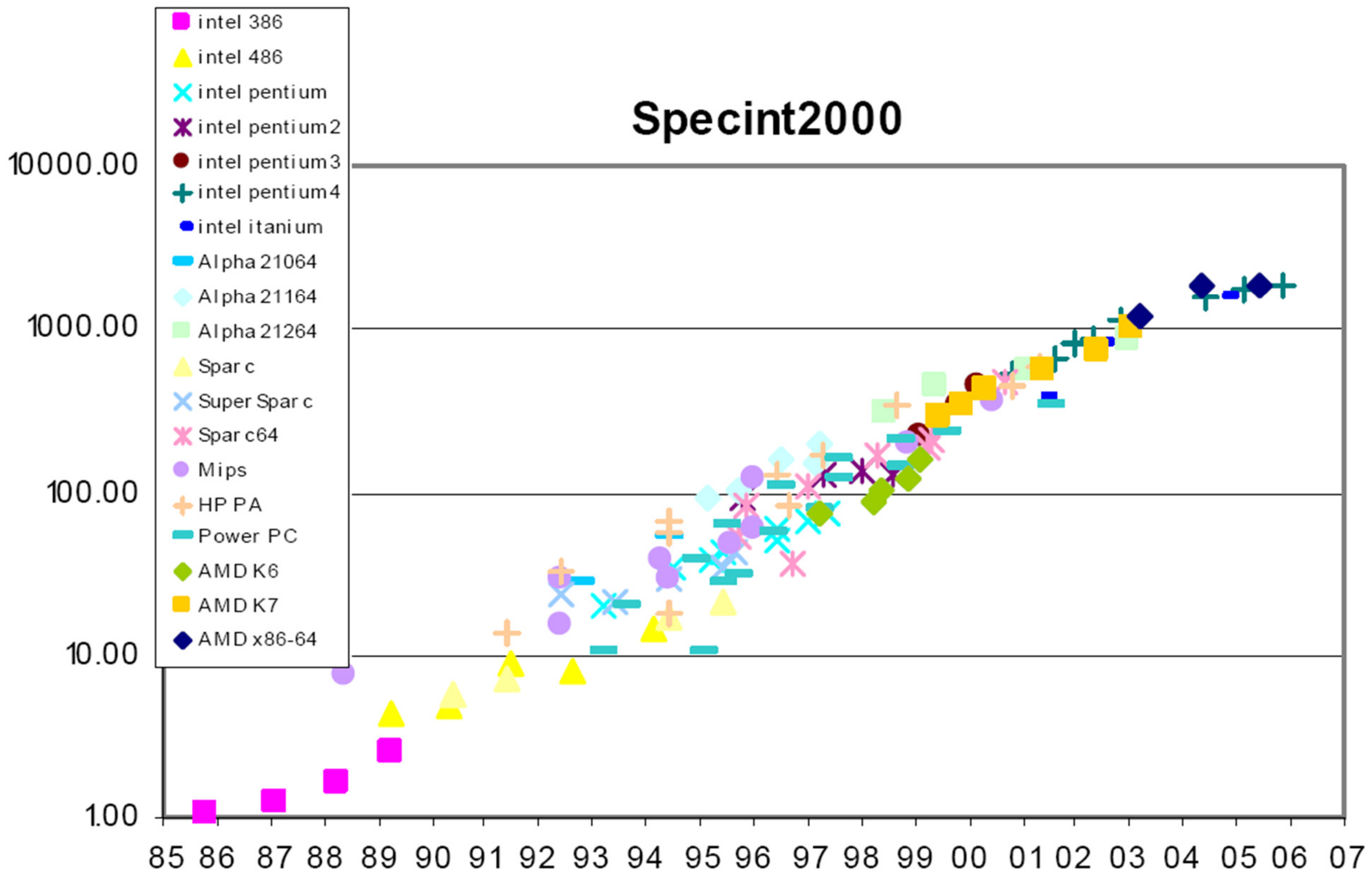
Why Parallelism?

Moore's Law



Source: Wikipedia

Unicore Performance



Source: Chung-Ta King, Department of Computer Science, National Tsing Hua University

Unicore Performance Has Hit a Wall!

Some Reasons

- Lack of additional ILP
  (Instruction Level Hidden Parallelism)
- High power density
- Manufacturing issues
- Physical limits
- Memory speed

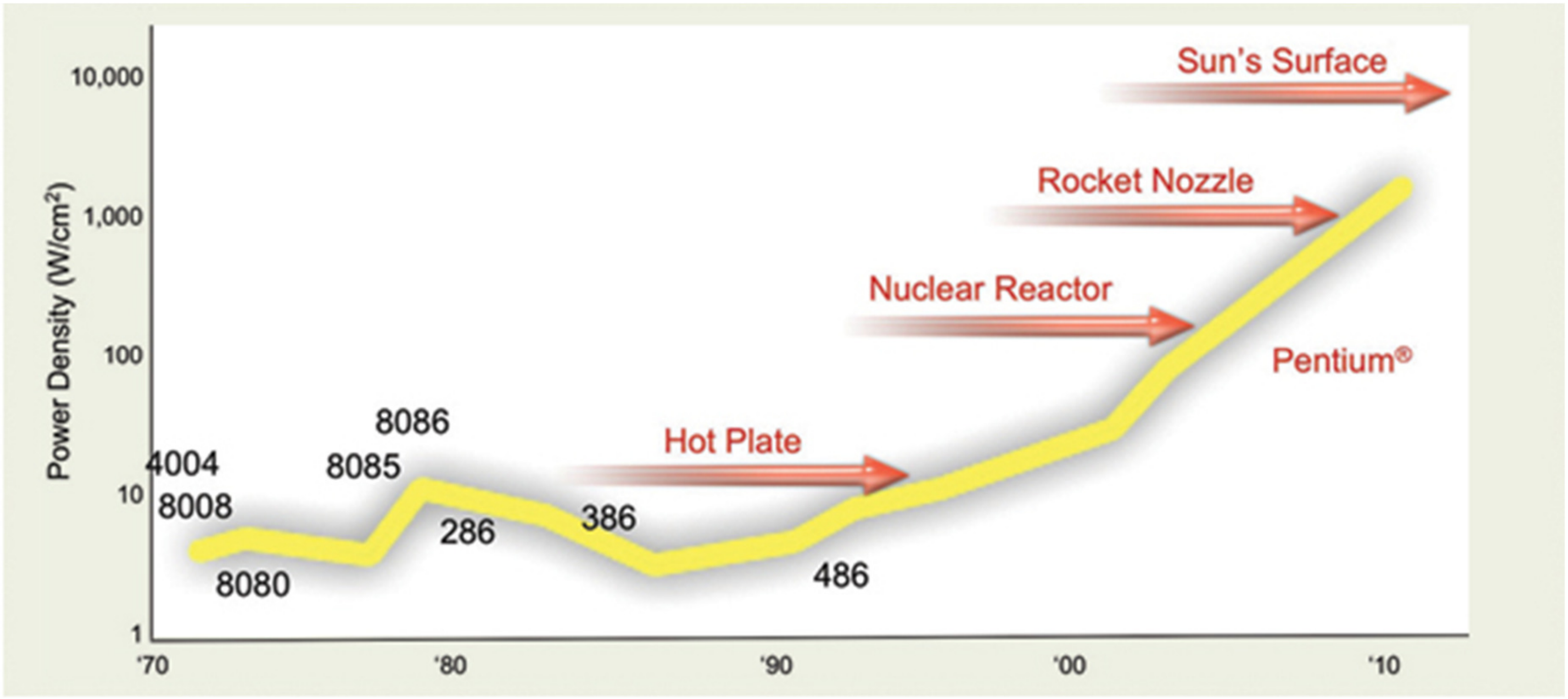
Unicore Performance: No Additional ILP

Exhausted all ideas to exploit hidden parallelism?

- Multiple simultaneous instructions
- Dynamic instruction scheduling
- Branch prediction
- Out-of-order instructions
- Speculative execution
- Pipelining
- Non-blocking caches, etc.

Unicore Performance: High Power Density

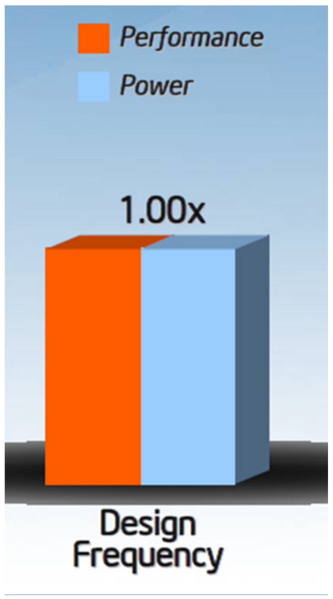
- Dynamic power, $P_d \propto V^2 f C$
  - V = *supply voltage*
  - f = *clock frequency*
  - C = *capacitance*
- But $V \propto f$
- Thus $P_d \propto f^3$



Source: Patrick Gelsinger, Intel Developer Forum, Spring 2004 (Simon Floyd)

Unicore Performance: High Power Density

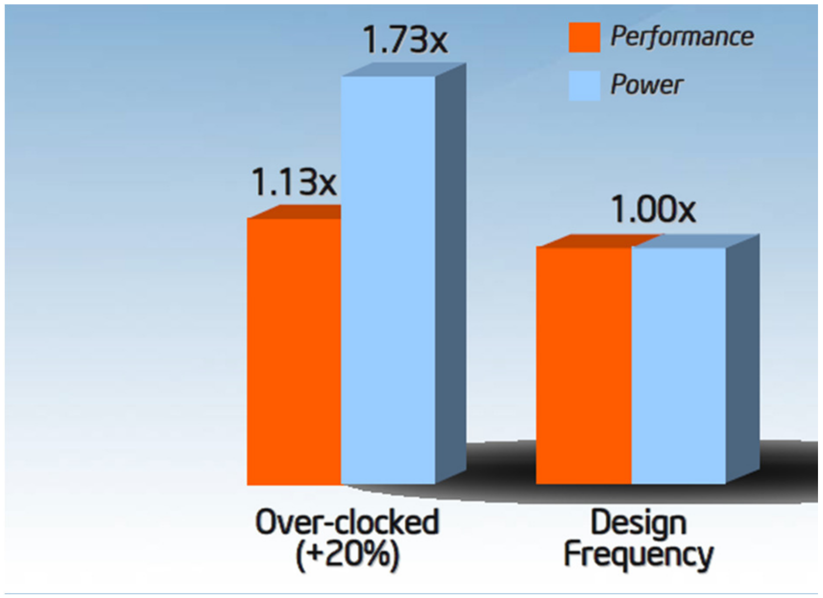
- Changing *f* by 20% changes performance by 13%
- So what happens if we overclock by 20%?
- And underclock by 20%?



Source: Andrew A. Chien, Vice President of Research, Intel Corporation

Unicore Performance: High Power Density

- Changing *f* by 20% changes performance by 13%
- So what happens if we overclock by 20%?
- And underclock by 20%?



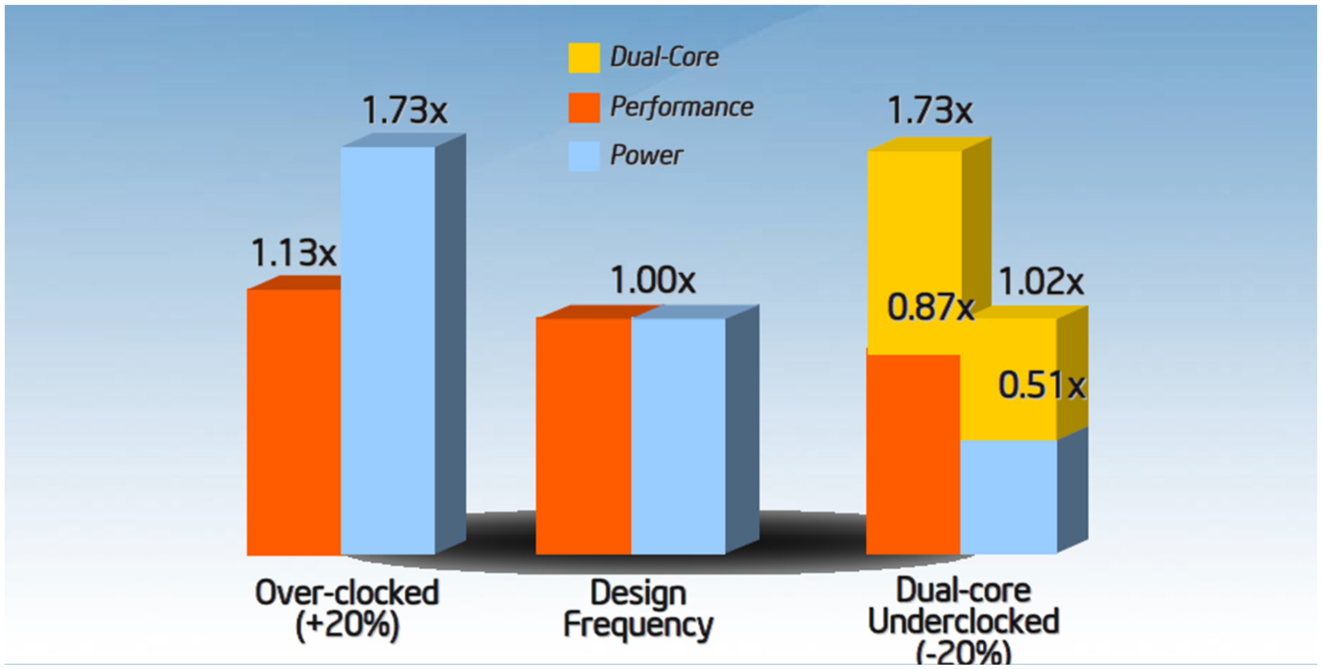
Source: Andrew A. Chien, Vice President of Research, Intel Corporation

Unicore Performance: High Power Density

– Changing *f* by 20% changes performance by 13%

– So what happens if we overclock by 20%?

– And underclock by 20%?



Source: Andrew A. Chien, Vice President of Research, Intel Corporation

Unicore Performance: Manufacturing Issues

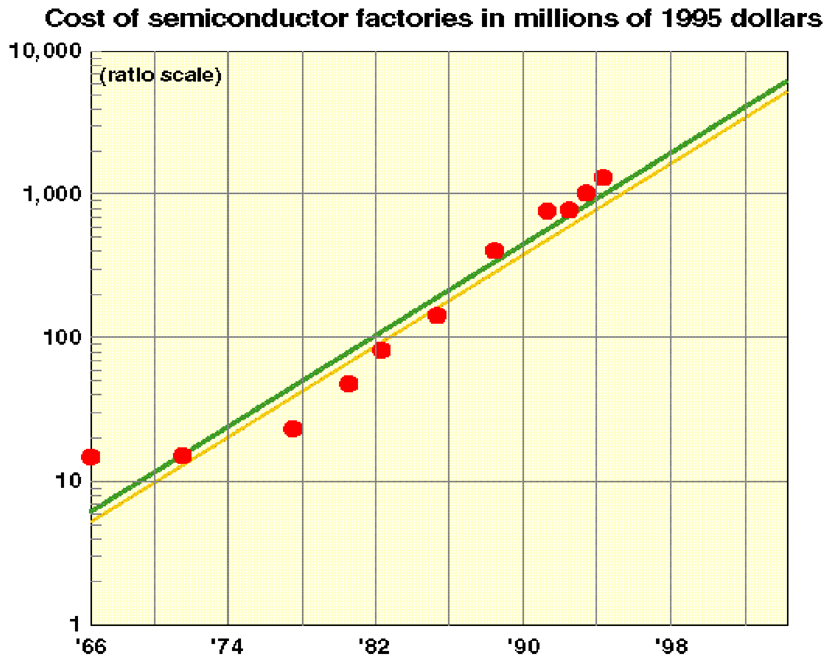
- Frequency, $f \propto 1 / s$
    - *s = feature size (transistor dimension)*
- Transistors / unit area $\propto 1 / s^2$
- Typically, die size $\propto 1 / s$
- So, what happens if feature size goes down by a factor of x?
    - Raw computing power goes up by a factor of x^4 !
    - Typically most programs run faster by a factor of x^3 without any change!

Source: Kathy Yelick and Jim Demmel, UC Berkeley

Unicore Performance: Manufacturing Issues

As feature size decreases

- Manufacturing cost goes up
  - Cost of a semiconductor fabrication plant doubles every 4 years (Rock's Law)
- Yield (% of usable chips produced) drops



Source: Kathy Yelick and Jim Demmel, UC Berkeley

Unicore Performance: Physical Limits

Execute the following loop on a serial machine in 1 second:

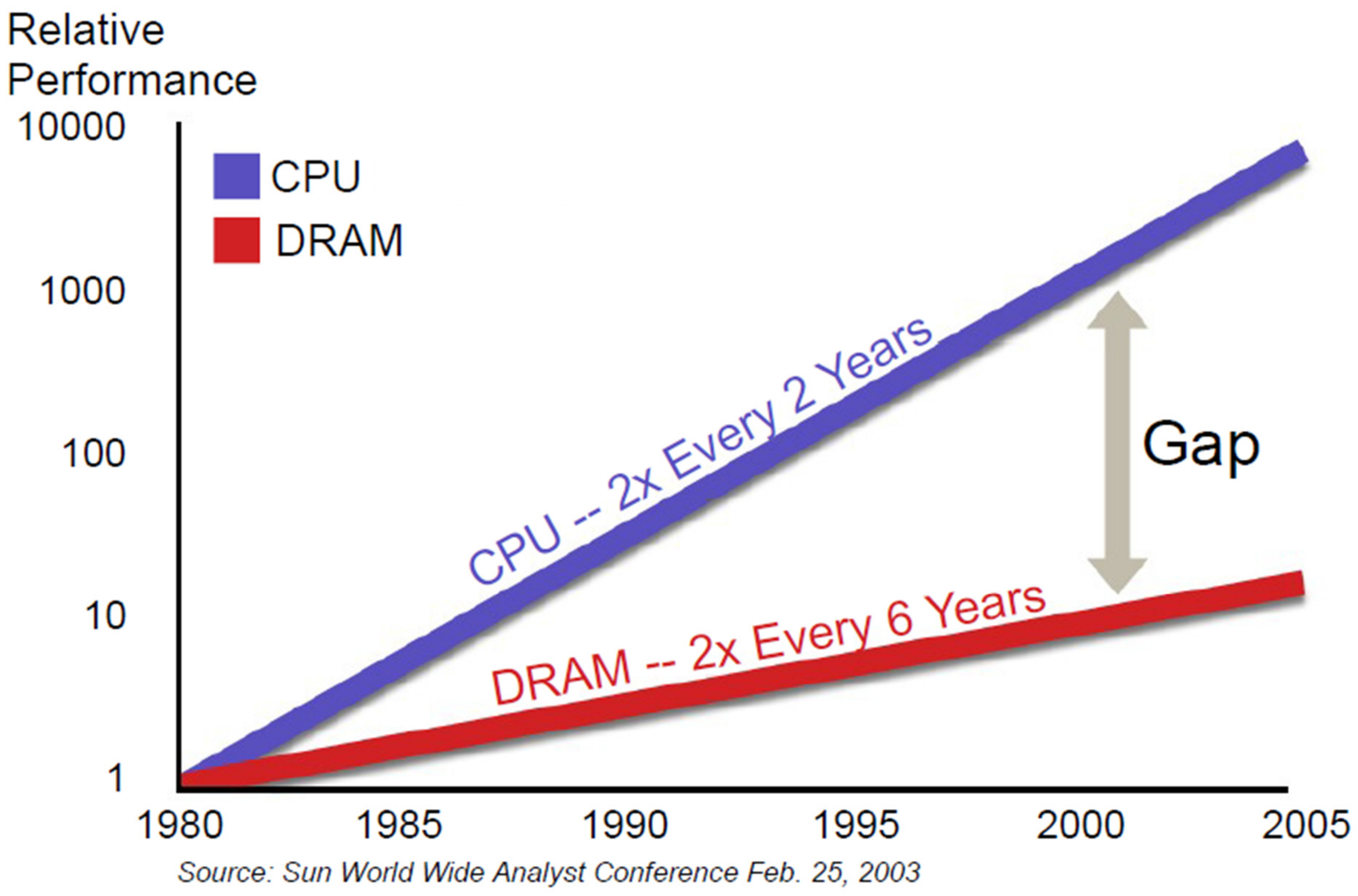
```
for ( i = 0; i < 10^12; ++i )
    z[ i ] = x[ i ] + y[ i ];
```

- We will have to access 3×10^{12} data items in one second
- Speed of light is, $c \approx 3 \times 10^8$ m/s
- So each data item must be within $c / 3 \times 10^{12} \approx 0.1$ mm from the CPU on the average
- All data must be put inside a 0.2 mm × 0.2 mm square
- Each data item (≥ 8 bytes) can occupy only 1 $Å^2$ space!

(size of a small atom!)

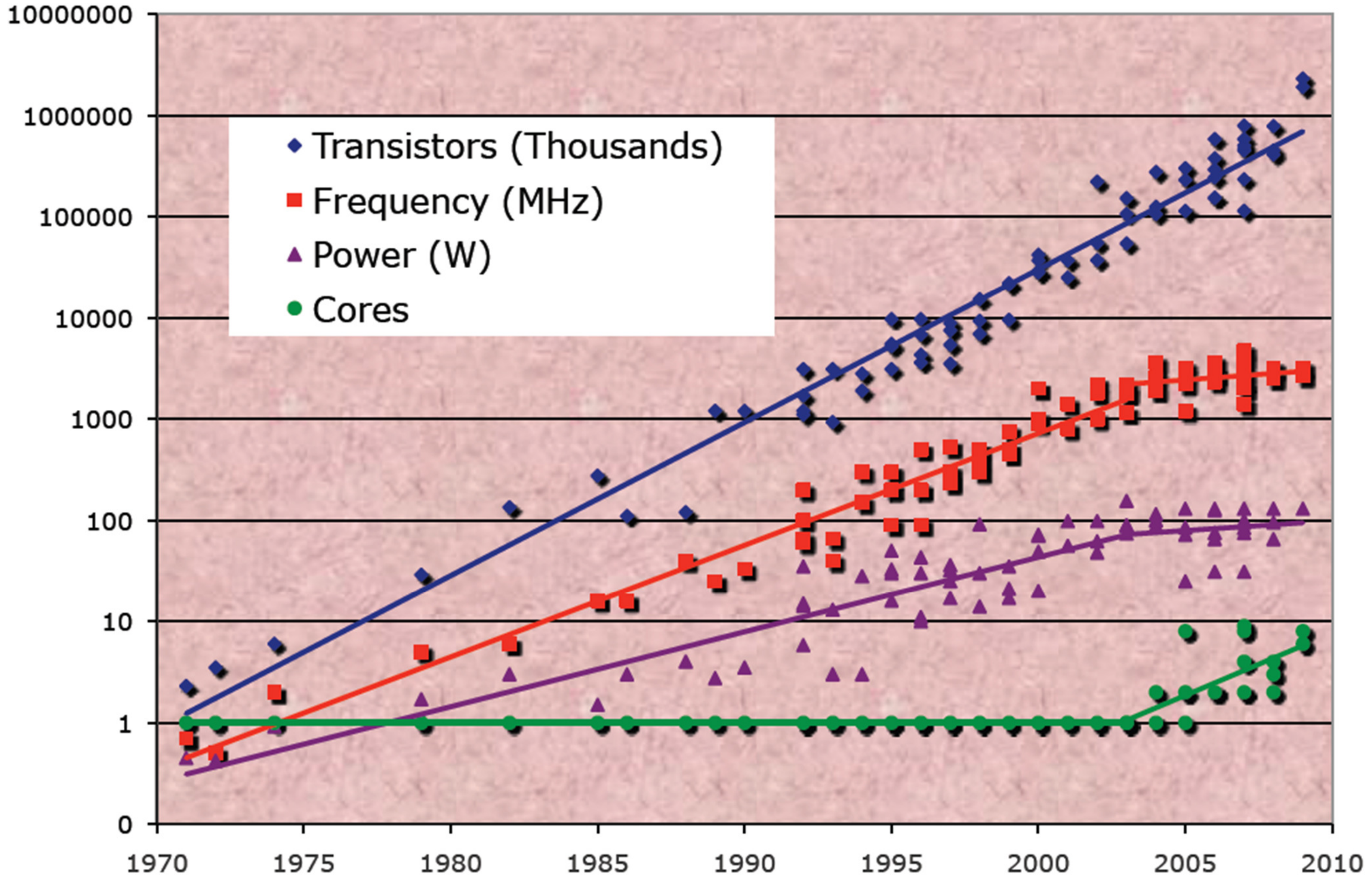
Source: Kathy Yelick and Jim Demmel, UC Berkeley

Unicore Performance: Memory Wall



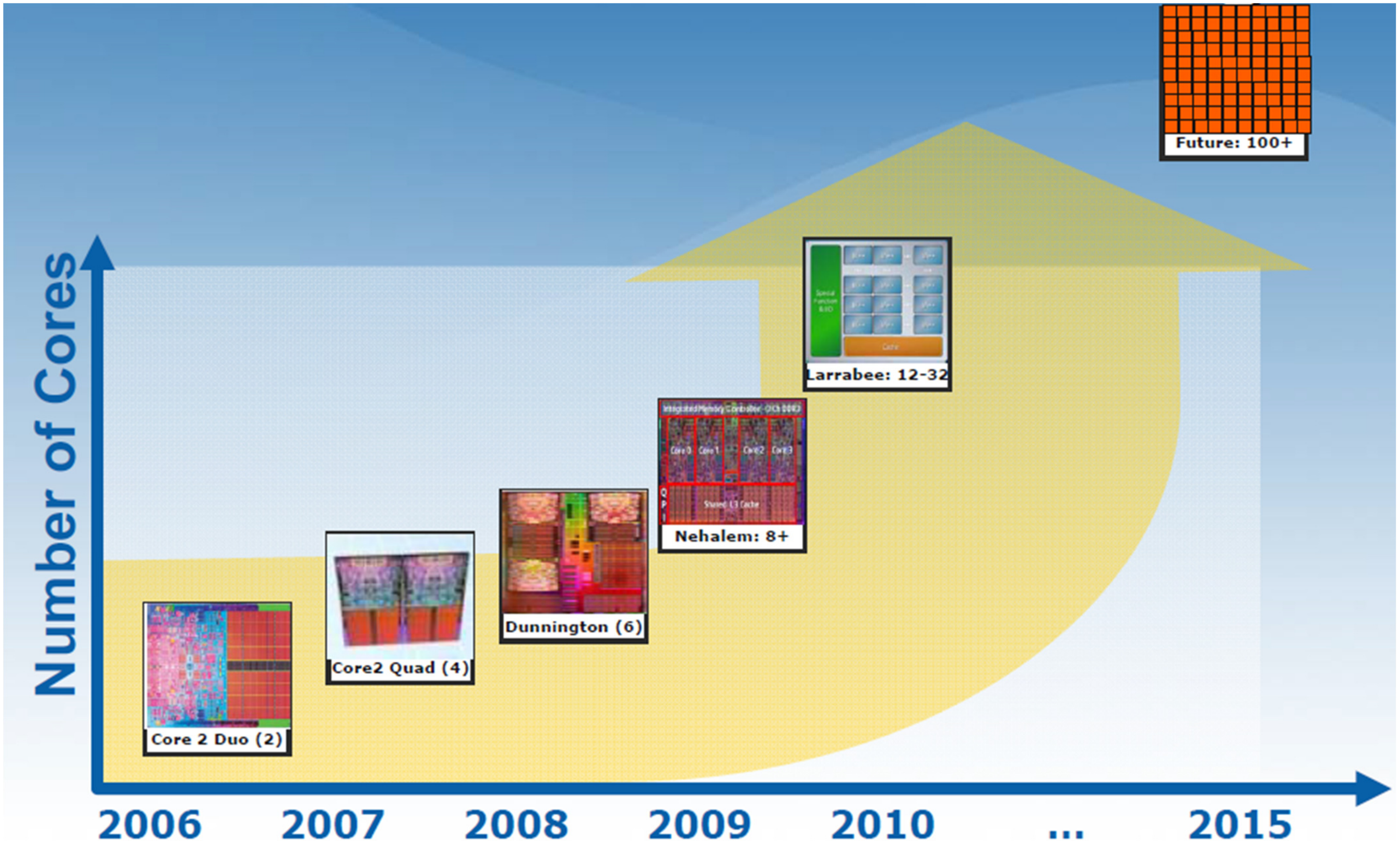
Source: Rick Hetherington, Chief Technology Officer, Microelectronics, Sun Microsystems

Moore's Law Reinterpreted



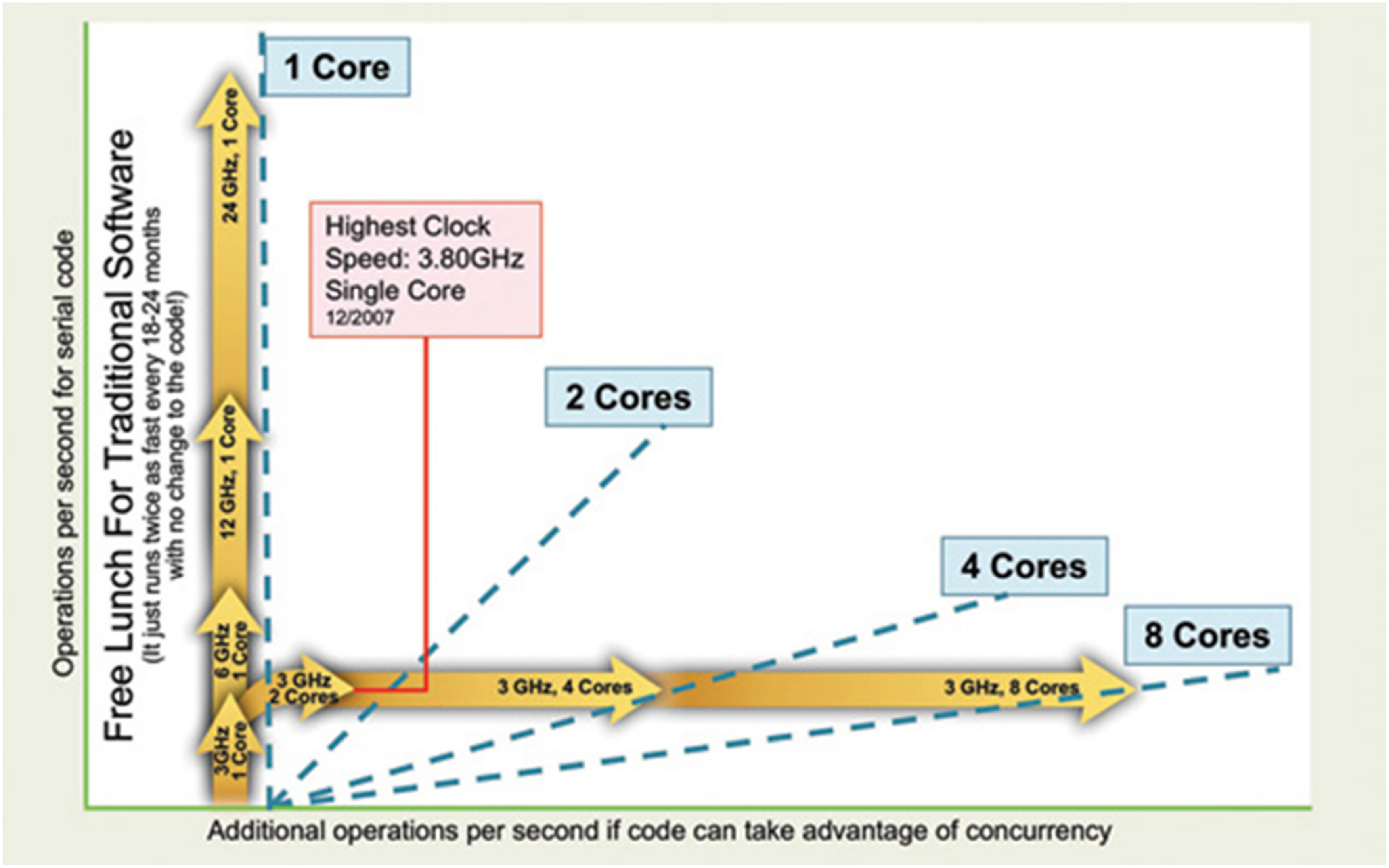
Source: Report of the 2011 Workshop on Exascale Programming Challenges

Cores / Processor (General Purpose)



Source: Andrew A. Chien, Vice President of Research, Intel Corporation

No Free Lunch for Traditional Software



Source: Simon Floyd, Workstation Performance: Tomorrow's Possibilities (Viewpoint Column)

Insatiable Demand for Performance

Source: Patrick Gelsinger, Intel Developer Forum, 2008

Numerical Weather Prediction

Problem: (*temperature*, *pressure*, …, *humidity*, *wind velocity*)
$\leftarrow$ *f*(*longitude*, *latitude*, *height*, *time*)

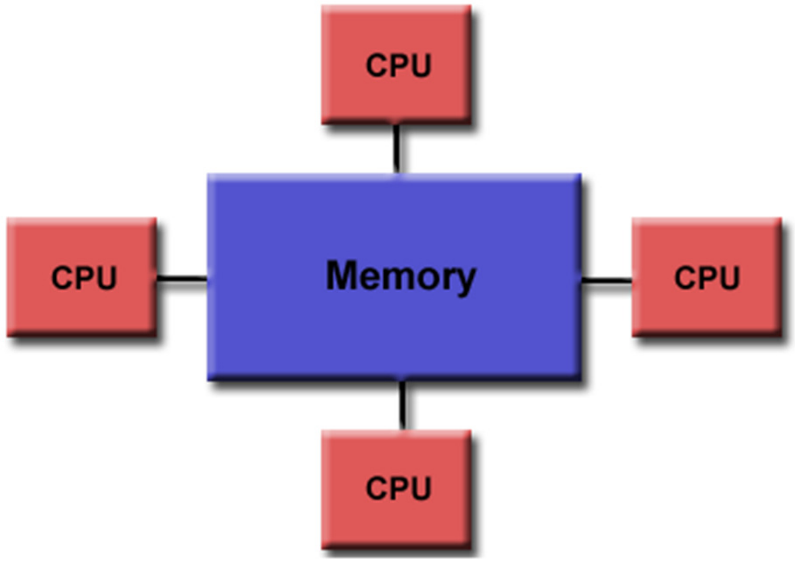
Approach (very coarse resolution):

- Consider only modeling fluid flow in the atmosphere
- Divide the entire global atmosphere into cubic cells of size 1 mile × 1 mile × 1 mile each to a height of 10 miles
  $\approx 2 \times 10^9$ cells
- Simulate 7 days in 1 minute intervals
  $\approx 10^4$ time-steps to simulate
- 200 floating point operations (flop) / cell / time-step
  $\approx 4 \times 10^{15}$ floating point operations in total
- To predict in 1 hour $\approx$ 1 Tflop/s (Tera flop / sec)

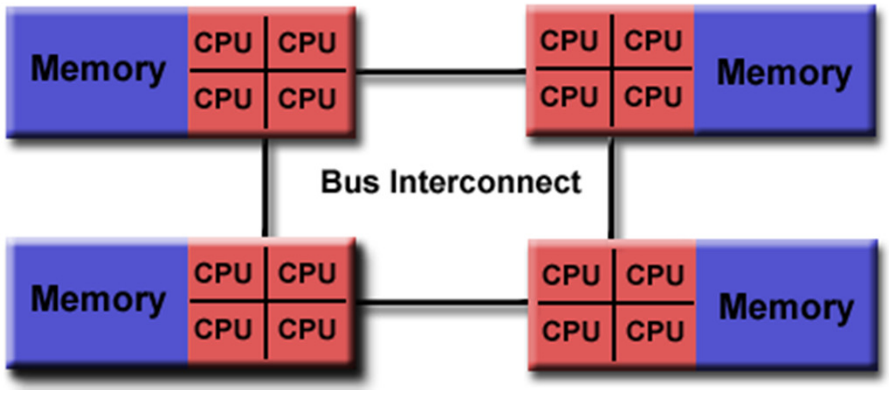
Some Useful Classifications of Parallel Computers

Parallel Computer Memory Architecture (Shared Memory)

- All processors access all memory as global address space
- Changes in memory by one processor are visible to all others
- Tow types:
  - Uniform Memory Access (UMA)
  - Non-Uniform Memory Access (NUMA)



UMA



NUMA

Source: Blaise Barney, LLNL

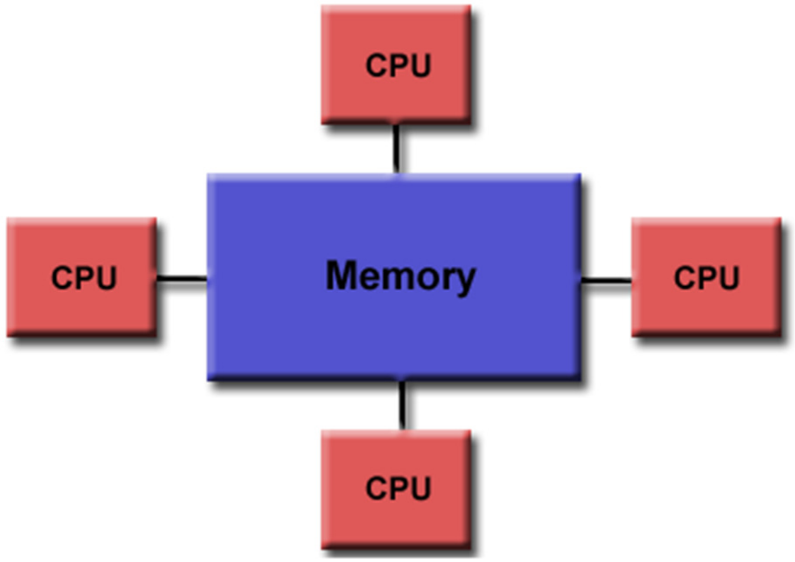
Parallel Computer Memory Architecture (Shared Memory)

Advantages

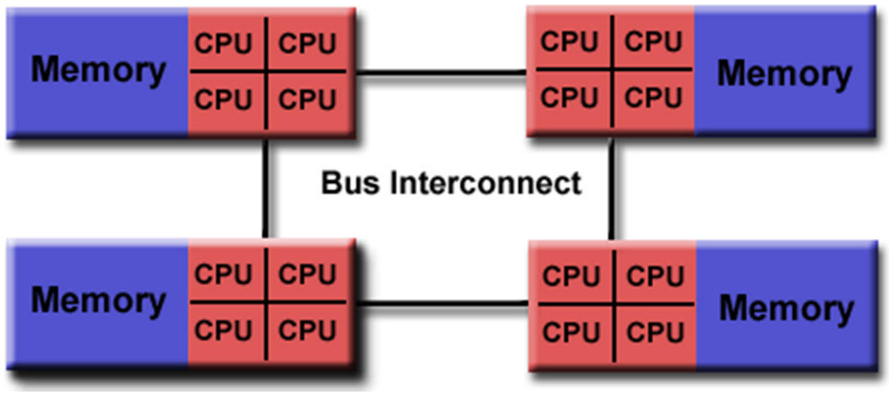
- User-friendly programming perspective to memory
- Fast data sharing

Disadvantages

- Difficult and expensive to scale
- Correct data access is user responsibility



UMA



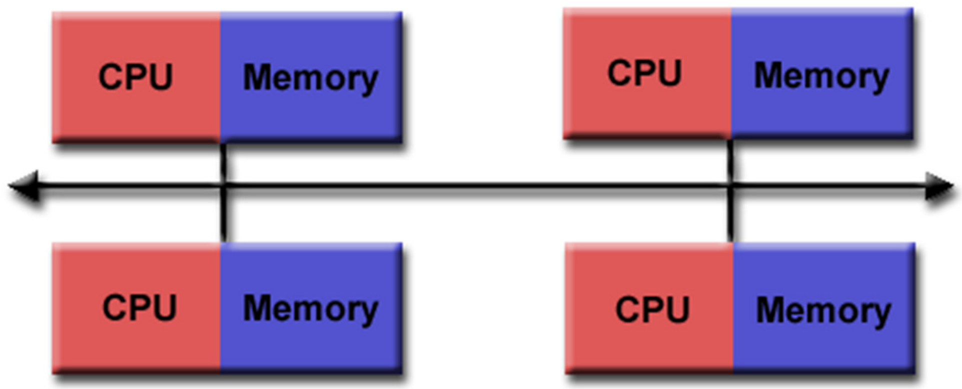
NUMA

Source: Blaise Barney, LLNL

Parallel Computer Memory Architecture (Distributed Memory)

– Each processor has its own local memory — no global address space

– Changes in local memory by one processor have no effect on memory of other processors



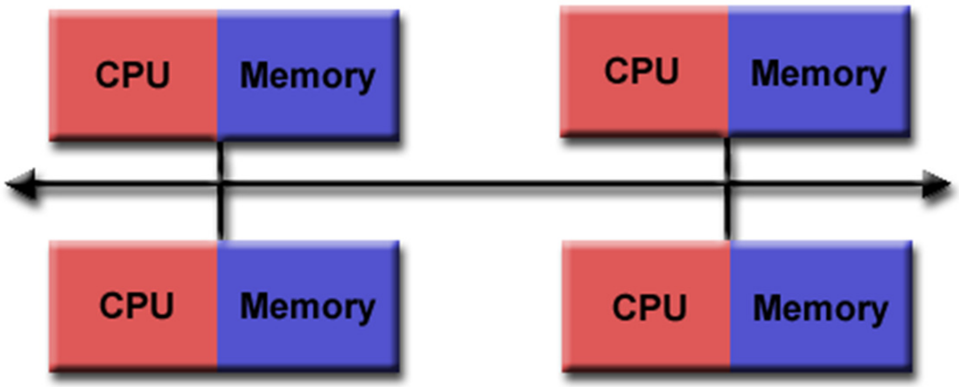
Source: Blaise Barney, LLNL

– Communication network to connect inter-processor memory

Parallel Computer Memory Architecture (Distributed Memory)

Advantages

- Easily scalable
- No cache-coherency needed among processors
- Cost-effective



Source: Blaise Barney, LLNL

Disadvantages

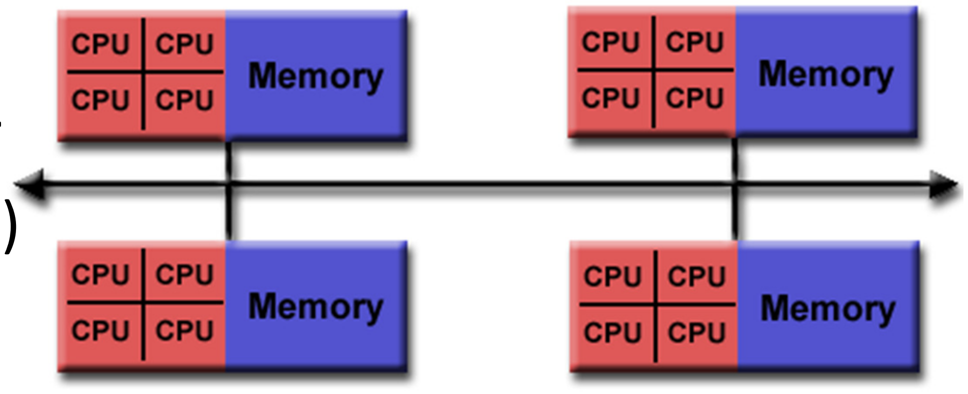
- Communication is user responsibility
- Non-uniform memory access
- May be difficult to map shared-memory data structures to this type of memory organization

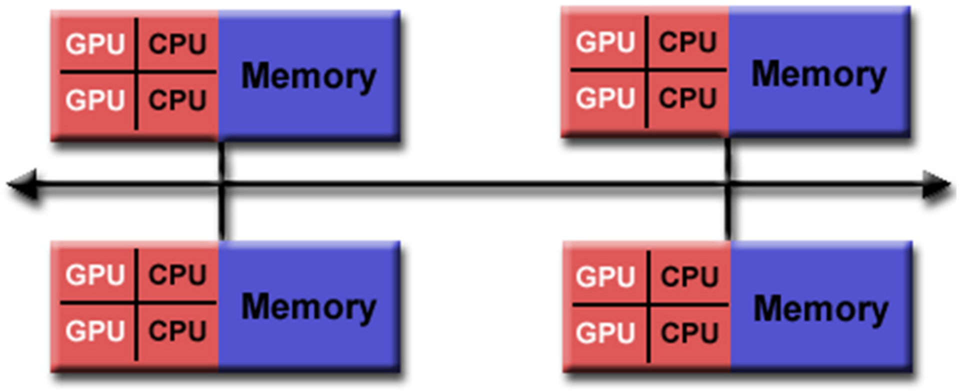
Parallel Computer Memory Architecture (Hybrid Distributed-Shared Memory)

– The share-memory component can be a cache-coherent SMP or a Graphics Processing Unit (GPU)

– The distributed-memory component is the networking of multiple SMP/GPU machines

– Most common architecture for the largest and fastest computers in the world today





Source: Blaise Barney, LLNL

Flynn’s Taxonomy of Parallel Computers

Flynn’s classical taxonomy (1966):

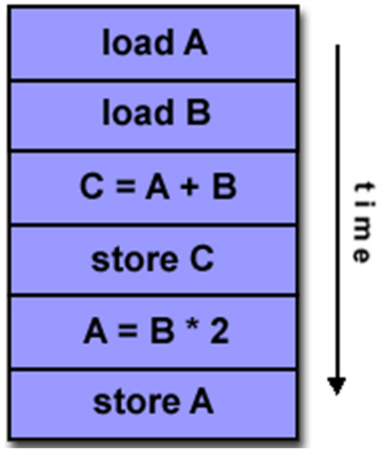
Classification of multi-processor computer architectures along two independent dimensions of *instruction* and *data*.

	Single Data (SD)	Multiple Data (MD)
Single Instruction (SI)	**SISD**	**SIMD**
Multiple Instruction (MI)	**MISD**	**MIMD**

Flynn's Taxonomy of Parallel Computers

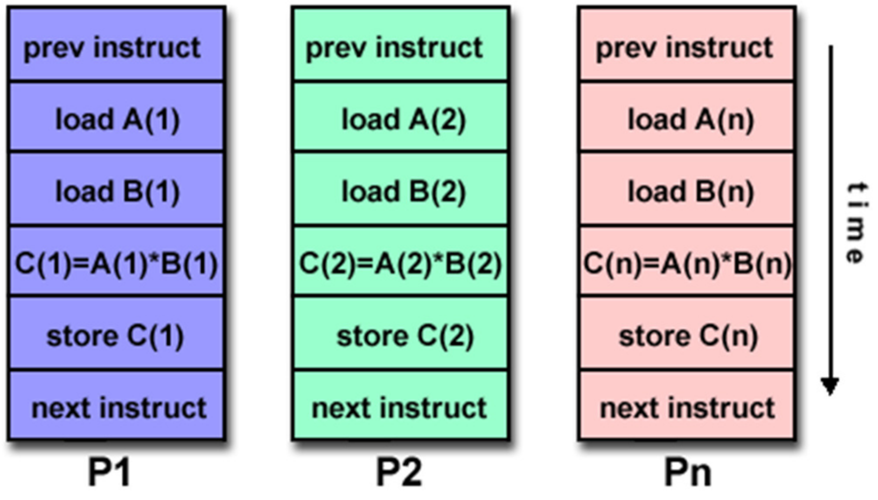
SISD

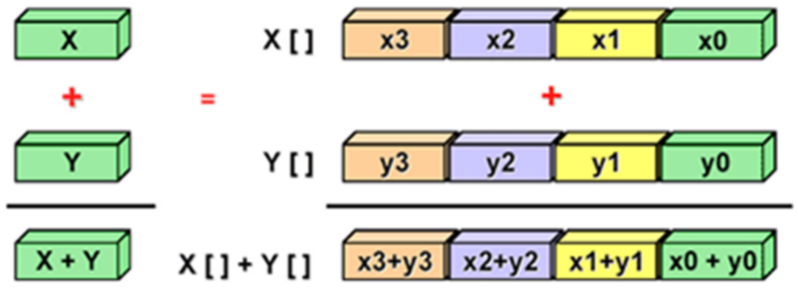
- A serial (non-parallel) computer
- The oldest and the most common type of computers
- Example: Uniprocessor unicore machines



Source: Blaise Barney, LLNL

Flynn's Taxonomy of Parallel Computers





Source: Blaise Barney, LLNL

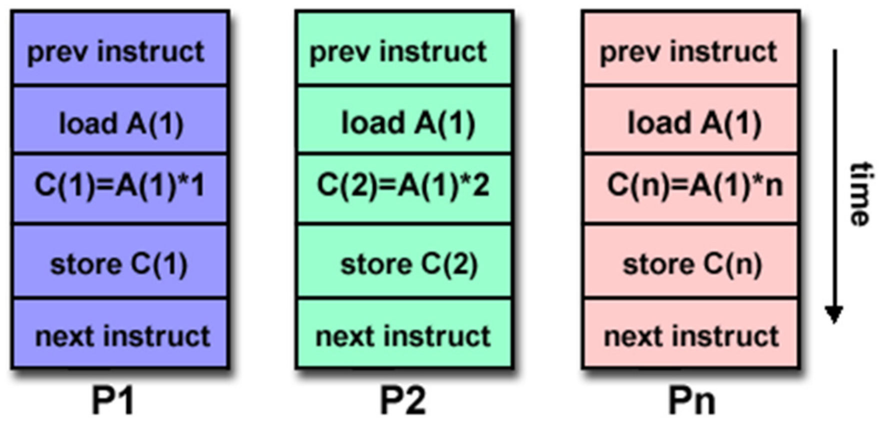
SIMD

- A type of parallel computer
- All PU's run the same instruction at any given clock cycle
- Each PU can act on a different data item
- Synchronous (lockstep) execution
- Two types: processor arrays and vector pipelines
- Example: GPUs (Graphics Processing Units)

Flynn's Taxonomy of Parallel Computers

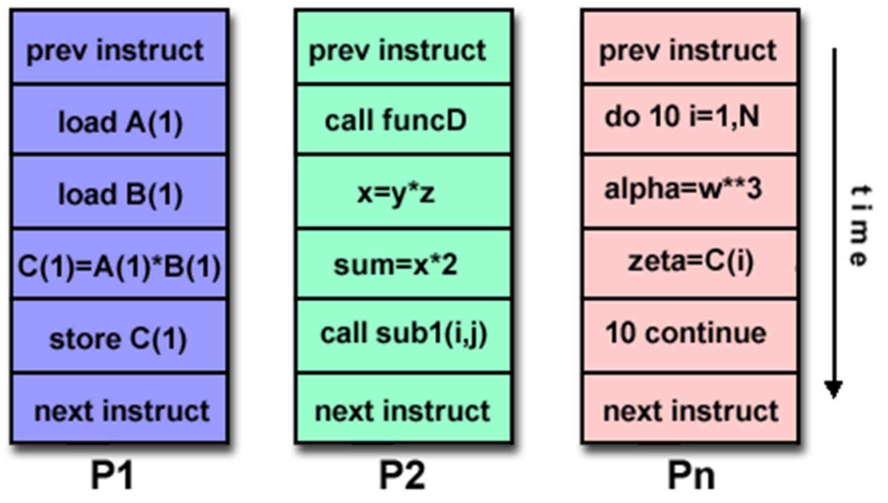
MISD

- A type of parallel computer
- Very few ever existed



MIMD

- A type of parallel computer
- Synchronous /asynchronous execution
- Examples: most modern supercomputers, parallel computing clusters, multicore PCs



Source: Blaise Barney, LLNL

Parallel Algorithms
Warm-up

"The way the processor industry is going, is to add more and more cores, but nobody knows how to program those things. I mean, two, yeah; four, not really; eight, forget it."

— *Steve Jobs, NY Times interview, June 10 2008*

Parallel Algorithms Warm-up (1)

Consider the following loop:

```
for $i$ = 1 to $n$ do
    $C[\, i \,] \leftarrow A[\, i \,] \times B[\, i \,]$
```

– Suppose you have an infinite number of processors/cores

– Ignore all overheads due to scheduling, memory accesses, communication, etc.

– Suppose each operation takes a constant amount of time

– How long will this loop take to complete execution?

  – $O(\, 1 \,)$ time

Parallel Algorithms Warm-up (2)

Now consider the following loop:

```
$c \leftarrow 0$
$for\ i = 1\ to\ n\ do$
    $c \leftarrow c + A[\,i\,] \times B[\,i\,]$
```

– How long will this loop take to complete execution?

  – $O(\log n)$ time

Parallel Algorithms Warm-up (3)

Now consider quicksort:

```
$QSort( A )$
$if\ |A| \leq 1\ return\ A$
$else\quad p \leftarrow A[\ rand(\ |A|\ )\ ]$
          return $QSort( \{ x \in A : x < p \} )$
                    $\# \{ p \} \#$
               $QSort( \{ x \in A : x > p \} )$
```

– Assuming that A is split in the middle everytime, and the two recursive calls can be made in parallel, how long will this algorithm take?

  – $O(\ \log^2 n\ )$ time!