COMP 322: Fundamentals of Parallel Programming

Vivek Sarkar
Department of Computer Science, Rice University
vsarkar@rice.edu

https://wiki.rice.edu/confluence/display/PARPROG/COMP322

COMP 322 Course Information: Spring 2012

- **"Fundamentals of Parallel Programming"**
- **Lectures:** MWF, 1pm – 1:50pm, DH 1070 (all sections)
- **Labs (mandatory):**
  - Tuesdays, 4:00pm - 5:20pm (section A03)
  - Wednesdays, 3:30pm - 4:50pm (section A02)
  - Thursdays, 4:00pm - 5:20pm (section A01)
- **Instructor:** Vivek Sarkar (vsarkar@rice.edu)
- **Prerequisite:** COMP 215 or equivalent
- **Cross-listing:** ELEC 323

Scope of Course

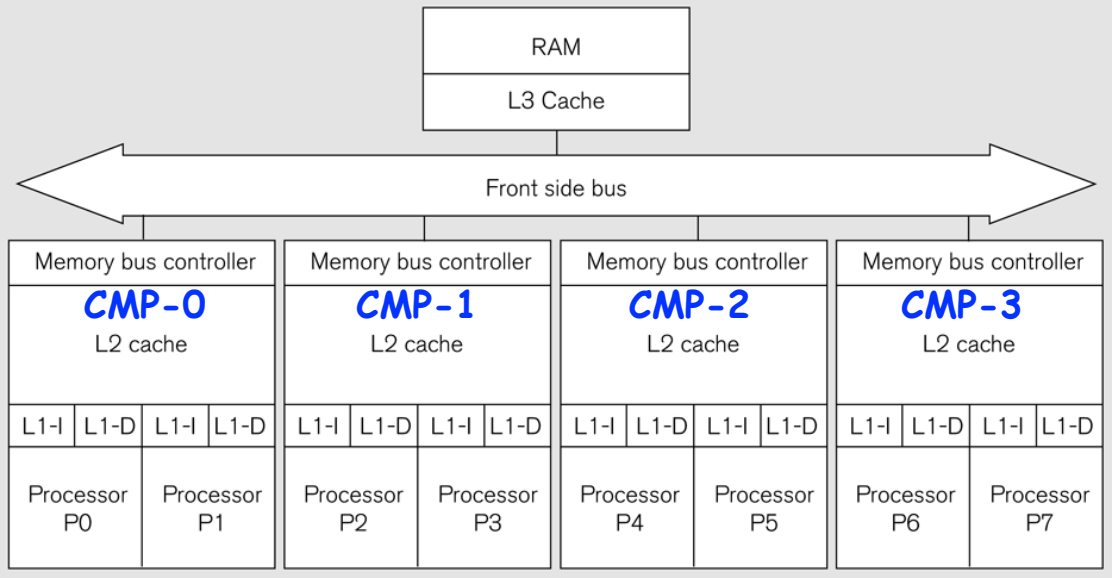
- Fundamentals of parallel programming
  - Primitive constructs for task creation & termination, collective & point-to-point synchronization, task and data distribution, and data parallelism
  - Abstract models of parallel computations and computation graphs
  - Parallel algorithms & data structures including lists, trees, graphs, matrices
  - Common parallel programming patterns
- Habanero-Java (HJ) language, developed in the Habanero Multicore Software Research project at Rice
- Java Concurrency
- Beyond HJ and Java: Map-Reduce, CUDA, MPI
- Written assignments
- Programming assignments
  - Abstract metrics
  - Real parallel systems (8-core Intel, Rice SUG@R system)

Lecture 1: The What and Why of Parallel Programming

- Acknowledgments
  - CS 194 course on "Parallel Programming for Multicore" taught by Prof. Kathy Yelick, UC Berkeley, Fall 2007
    - http://www.cs.berkeley.edu/~yelick/cs194f07/
  - "Principles of Parallel Programming", Calvin Lin & Lawrence Snyder, Addison-Wesley 2009
  - COMP 322 Lecture 1 handout

What is Parallel Computing?

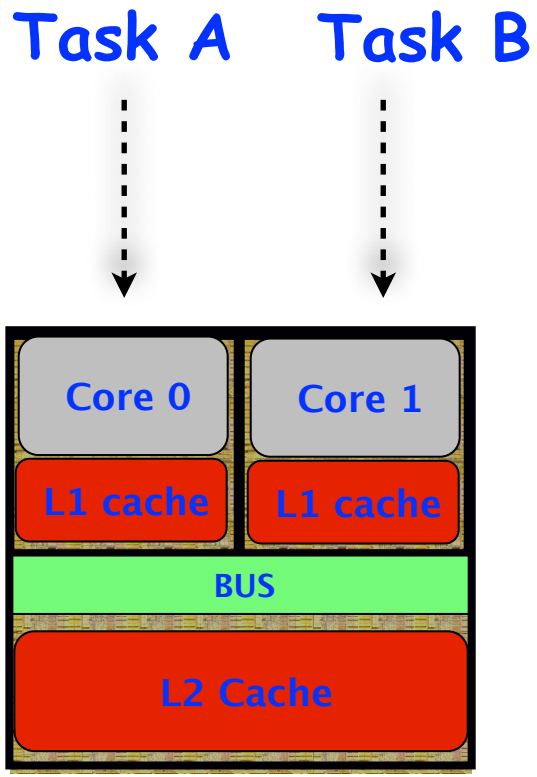
- **Parallel computing:** using multiple processors in parallel to solve problems more quickly than with a single processor and/or with less energy
- Examples of a parallel computer
  - An 8-core Symmetric Multi-Processor (SMP) consisting of four dual-core Chip Multi-Processors (CMPs)



Source: Figure 1.5 of Lin & Snyder book, Addison-Wesley, 2009

What is Parallel Programming?

- Specification of operations that can be executed in parallel
- A parallel program is decomposed into sequential subcomputations called tasks
- Parallel programming constructs define task creation, termination, and interaction

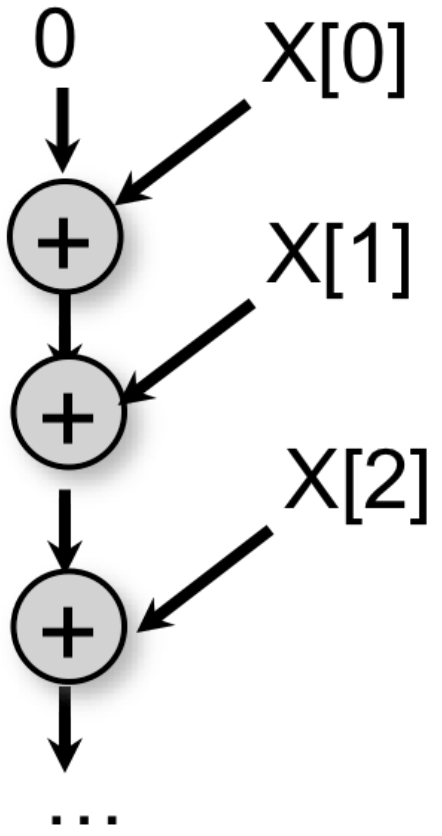


Schematic of a dual-core Processor

Example of a Sequential Program: Computing the sum of array elements

```
int sum = 0;
for (int i=0 ; i < X.length ; i++)
    sum += X[i];
```

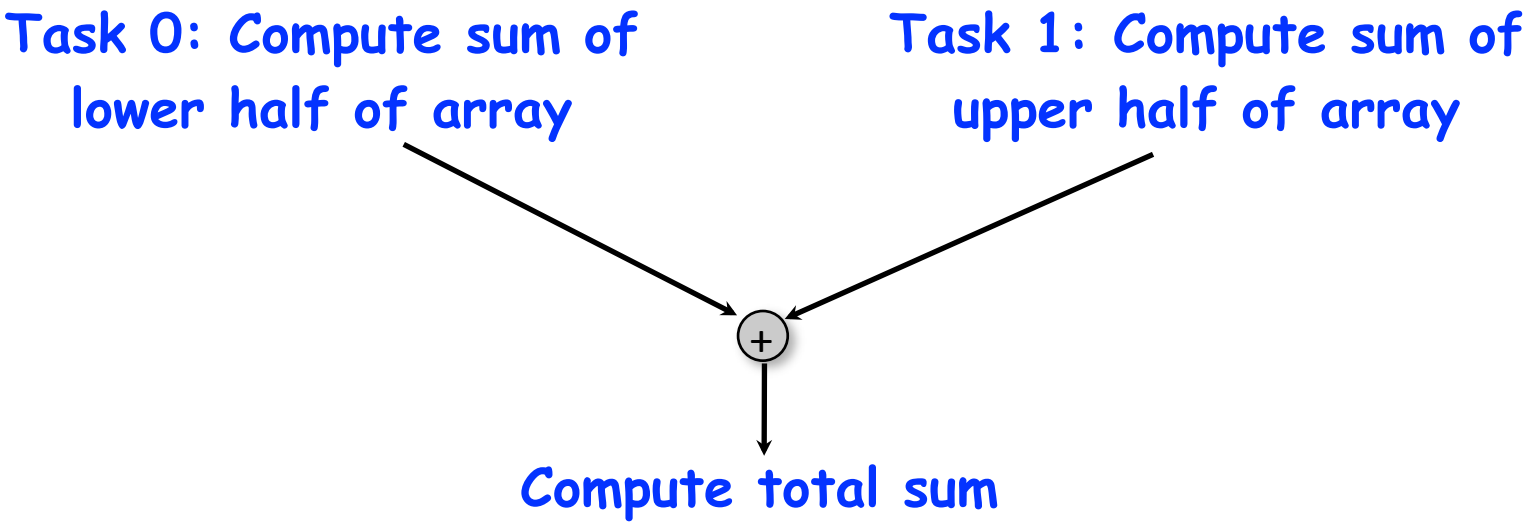
Computation Graph



Observations:

- The decision to sum up the elements from left to right was arbitrary
- The computation graph shows that all operations must be executed sequentially

Parallelization Strategy for two cores



Basic idea:

- Decompose problem into two tasks for partial sums
- Combine results to obtain final answer
- Parallel divide-and-conquer pattern

Example of a Parallel Program: Array Sum using async & finish constructs

```
1.  // Start of Task T0 (main program)
2.  sum1 = 0; sum2 = 0; // sum1 & sum2 are static fields
3.  finish {
4.    async { // Task T1 computes sum of upper half of array
5.      for(int i=X.length/2; i < X.length; i++) sum2 += X[i];
6.    }
7.    // Continue in T0 and compute sum of lower half of array
8.    for(int i=0; i < X.length/2; i++) sum1 += X[i];
9.  } // finish
10. // Task T0 waits for Task T1 (join)
11. return sum1 + sum2;
```


Async and Finish Statements for Task Creation and Termination

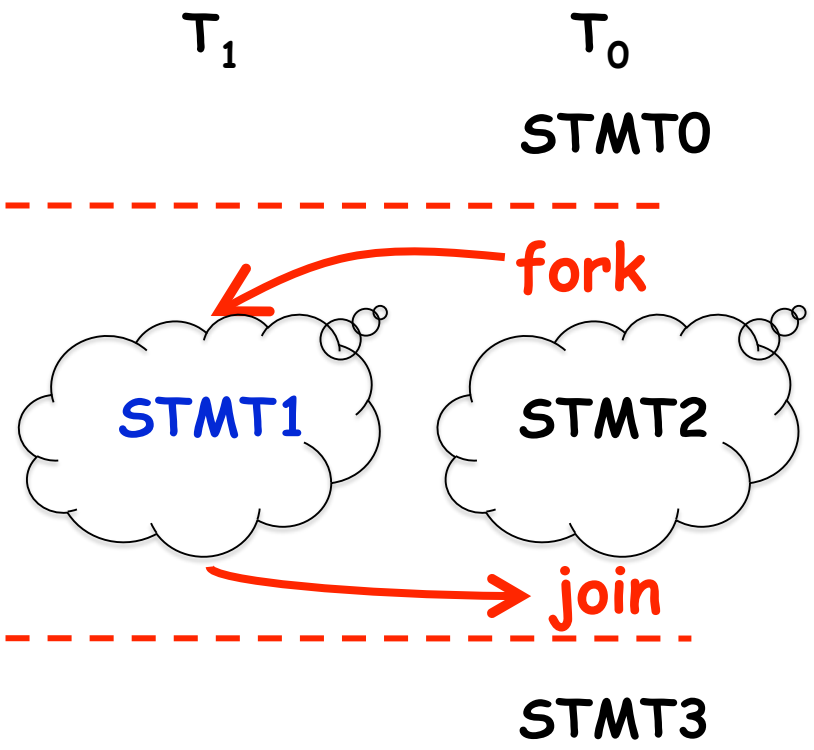
async S

- Creates a new child task that executes statement S

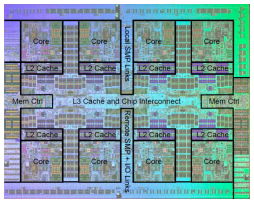
finish S

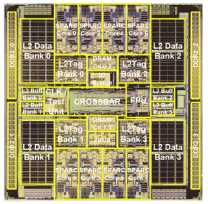
- Execute S, but wait until *all* asyncs in S's scope have terminated.

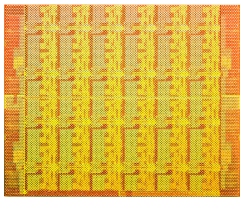
```
// T0 (Parent task)
STMT0;
finish {    //Begin finish
  async {
    STMT1; //T1 (Child task)
  }
  STMT2;    //Continue in T0
            //Wait for T1
}           //End finish
STMT3;      //Continue in T0
```

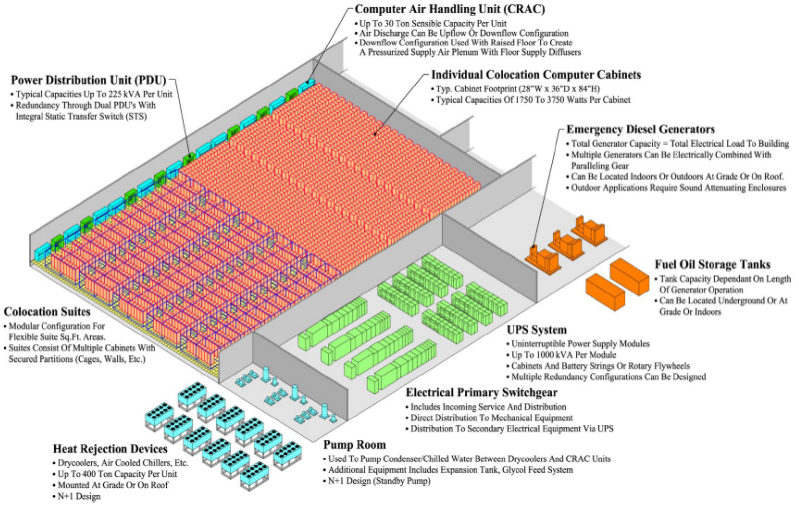


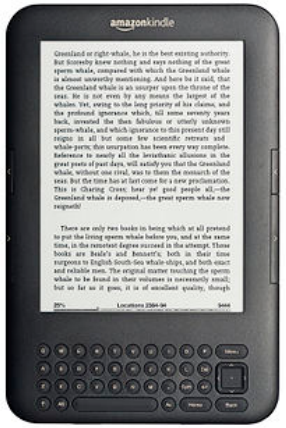
All Computers are Parallel Computers



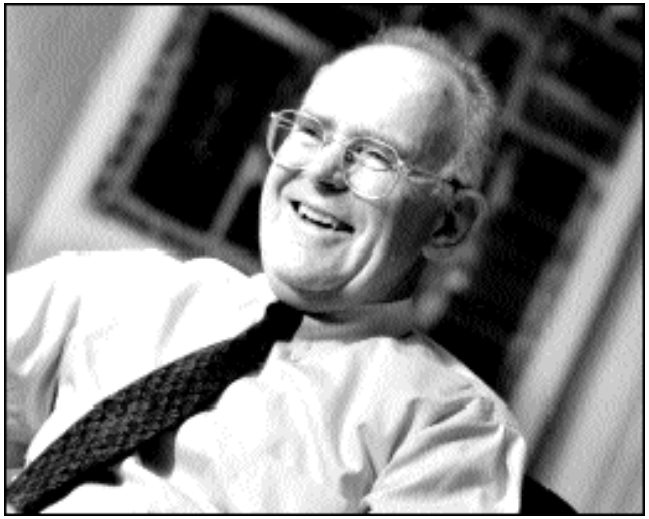


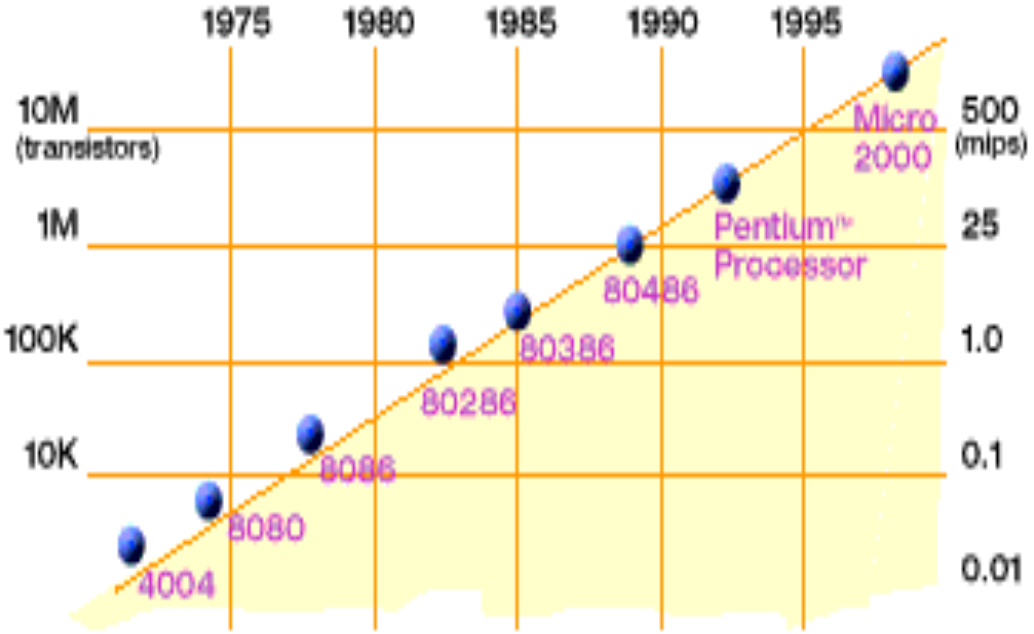






Moore's Law





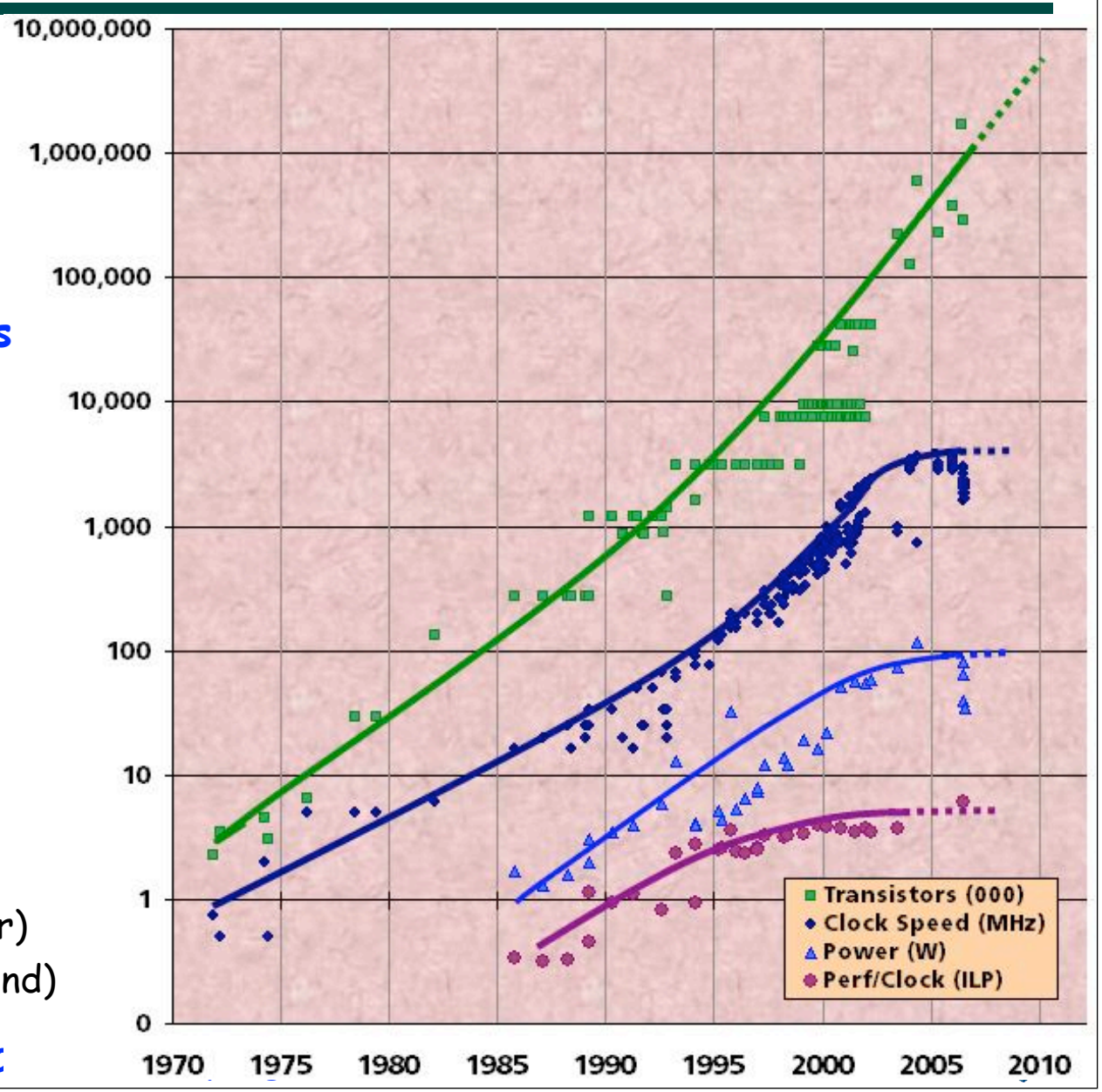
Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 1-2 years

Resulted in CPU clock speed doubling roughly every 18 months, but not any longer

Slide source: Jack Dongarra

Current Technology Trends

- Chip density is continuing to increase ~2x every 2 years
  - Clock speed is not
  - Number of processors is doubling instead
- Parallelism must be managed by software



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

Parallelism Saves Power

Power = (Capacitance) * (Voltage)^2 * (Frequency)

➔ Power α (Frequency)^3

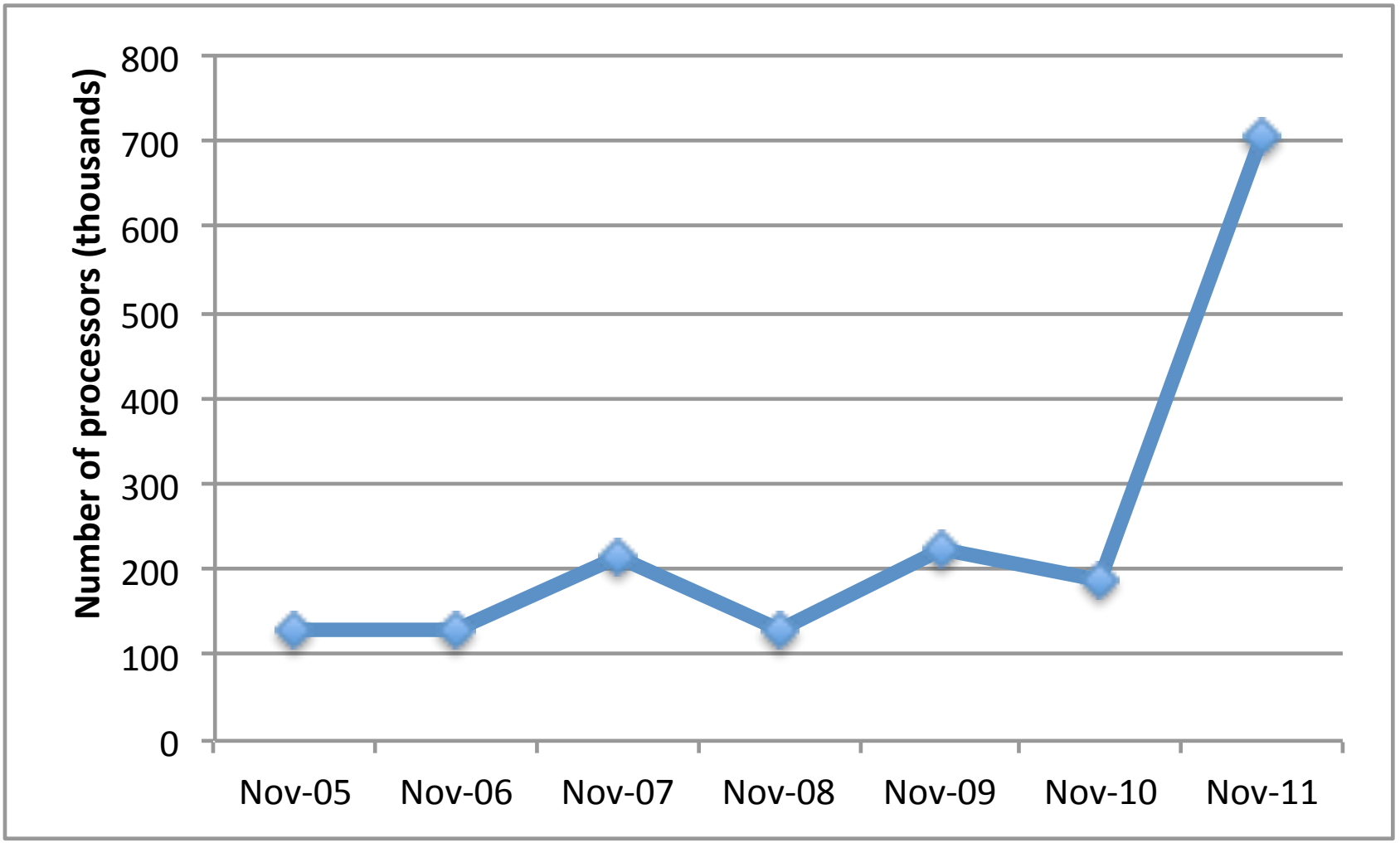
Baseline example: single 1GHz core with power P

Option A: Increase clock frequency to 2GHz ➔ Power = 8P

Option B: Use 2 cores at 1 GHz each ➔ Power = 2P

- Option B delivers same performance as Option A with 4x less power … provided software can be decomposed to run in parallel!

Number of processors in the world's fastest computers during 2005-2011



Source: http://www.top500.org

Parallel Programming Challenges

- Correctness
  - New classes of bugs can arise in parallel programming, relative to sequential programming
    - Data races, deadlock, nondeterminism
- Performance
  - Performance of parallel program depends on underlying parallel system
    - Language compiler and runtime system
    - Processor structure and memory hierarchy
    - Degree of parallelism in program vs. hardware
- Portability
  - A buggy program that runs correctly on one system may not run correctly on another (or even when re-executed on the same system)
  - A parallel program that performs well on one system may perform poorly on another

Food for thought

- Consider adding async and finish keywords to any sequential Java program that you've written
  - —Will the parallel version generate the same answer as the sequential version?
  - —Will the output of the parallel version depend on the order in which tasks execute their statements?
- Suppose you were given a parallel computer with an unbounded number of processors
  - —How many async tasks can you create that can execute at the same time?

COMP 322 Course Information: Spring 2012

- "Fundamentals of Parallel Programming"
- **Lectures:** MWF, 1pm – 1:50pm, DH 1070 (all sections)
- **Labs (mandatory):**
  - Tuesdays, 4:00pm - 5:20pm (section A03)
  - Wednesdays, 3:30pm - 4:50pm (section A02)
  - Thursdays, 4:00pm - 5:20pm (section A01)
- **Course Requirements:**
  - Homeworks (7) 50%
  - Exams (2) 40%
  - Lab attendance 10%
- **HW1 is assigned today and is due on Friday, Jan 13th**