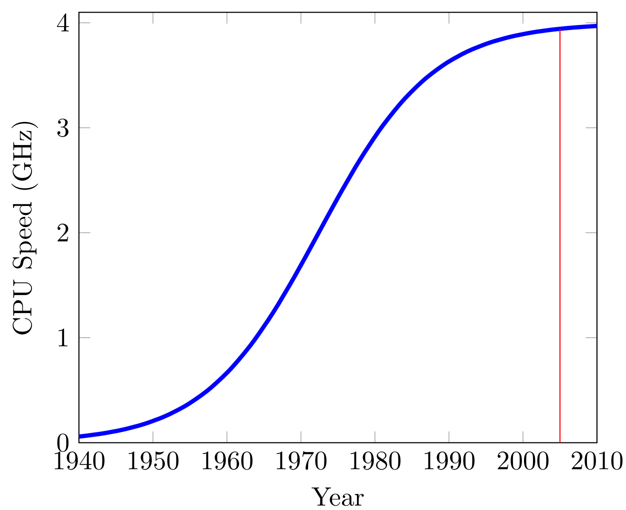
**CS452: Parallel Algorithms** **Spring 2019**

# Lecture 1: January 14

*Lecturer: Dr. Ankur Gupta* *Scribe: Rachel Burke*

## 1.1 Introduction

### 1.1.1 History of Computing




- **Moore's Law**: Every 18 months computing speed doubles
- Why did computing get faster?
  1. Smaller computers
  2. Wrote better algorithms
     (a) Algorithmic ideas were refined (amortization, randomization, approximation)
     (b) Leveraged hardware better
  3. Wider busses (16 $\rightarrow$ 32 $\rightarrow$ 64 $\rightarrow$ 128 bits)
  4. Fewer bridges
  5. Manufacturing got better
- Why is it flatlining?
  1. Small computers taking in large amounts of electricity generates a lot of heat
  2. Algorithms are starting to slow in the ability to improve, NP=P Problem
- What are we doing?
  1. More cores
  2. Parallel computing

**CS452: Parallel Algorithms** **Spring 2019**

# Lecture 2: January 16

*Lecturer: Dr. Ankur Gupta* *Scribe: Rachel Burke*

## 2.2 MPI Programming Basics

```
*******************************************************************************
These are things found in the template file!
*******************************************************************************
#include "mpi.h"                         ~ message passing interface header file
mpicxx -o blah file.cpp                  ~ compile a program using MPI
mpirun -q -np 32 blah                    ~ run a program using MPI with 32 processors

int my_rank;                             ~ my CPU number for this process
int p;                                   ~ number of CPUs that we have
int source;                              ~ rank of the sender
int dest;                                ~ rank of destination
int tag = 0;                             ~ message number
char message[100];                       ~ message itself
MPI_Status status;                       ~ return status for receive

MPI_Init(&argc, &argv);                  ~ start MPI
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank); ~ find ranks
MPI_Comm_size(MPI_COMM_WORLD, &p);       ~ find number of processes
MPI_Finalize();                          ~ shutdown MPI

*******************************************************************************
These functions wait until they are executed and can cause deadlocks!
*******************************************************************************
MPI_Send(...);                           ~ send messages to process(es)
MPI_Recv(...);                           ~ receive messages from other process(es)

*******************************************************************************
These are variables you can send in that are helpful!
*******************************************************************************
MPI_ANY_SOURCE                           ~ take things in any order to process
```

**CS452: Parallel Algorithms** **Spring 2019**

Lecture 6: February 6

*Lecturer: Dr. Ankur Gupta* *Scribe: Rachel Burke*

## 6.3 Ways to Handle Concurrent Write Scenarios

1. Arbitrary CW
   - random process wins the ability to write
2. Priority CW
   - programmer assigns an apriori hierarchy for processors and you follow those guidelines
   - then the highest priority writes
3. Common CW
   - only allow writes when all processors that want to write agree on what to write

## 6.4 Distributed Memory Model

1. $p$ synchronous processors
2. each processor has its own private memory
3. communication among processors is expensive

## 6.5 Work and Time

1. 1 "round" or "pulse" of time $\rightarrow$
2. `for each p where p is a processor from 1 to n pardo`
3. `B[i] = A[i]`
4. The **time** is one unit
5. The **work** or amount of stuff to do is still $n$

4 16 3 7 1 9 2 6

## 6.6 Summation Problem

### 6.6.1 Array A with size 8

- sequential sum $\rightarrow O(n)$ time for arrays of size $n$
- How much time in parallel?
  - $O(\frac{n}{p})$ for an embarassingly parallel solution
  - In Practice: $O(\frac{n}{constant} \implies O(n)$
- Notice that there are 3 rounds to move from the base to the root node
- Thus optimally we can solve this problem in $O(\log_2 n$ rounds