CSCI-UA.0480-003

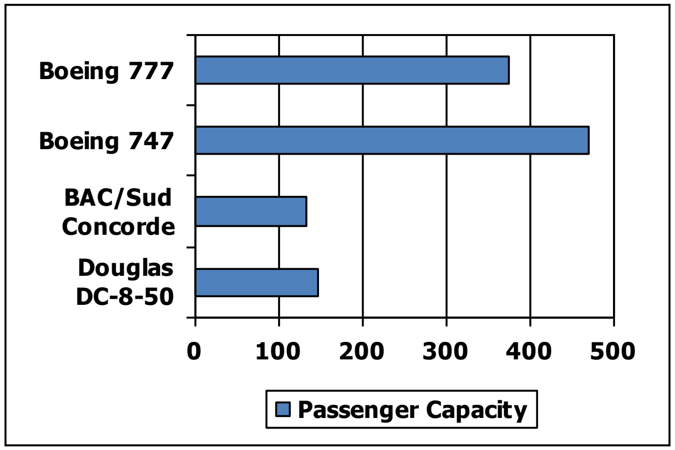
Parallel Computing

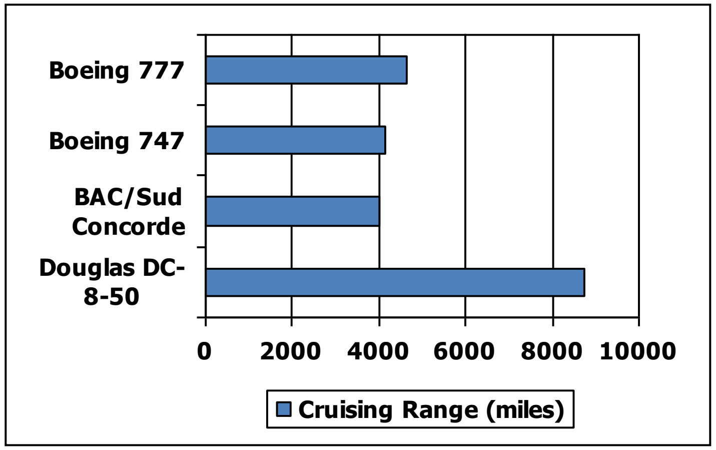
Lecture 6: Performance Analysis

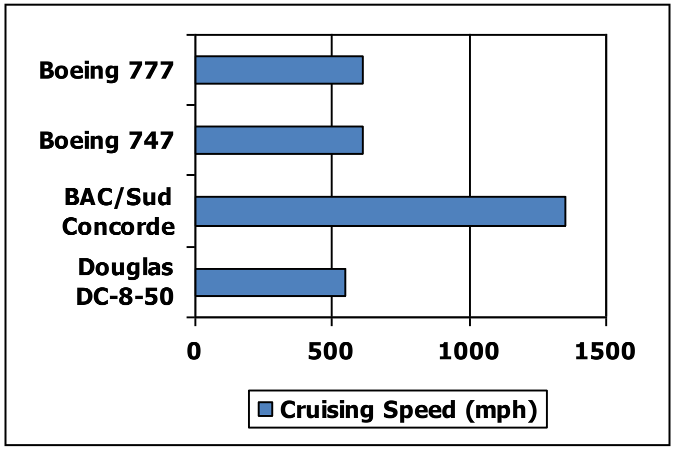
Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com

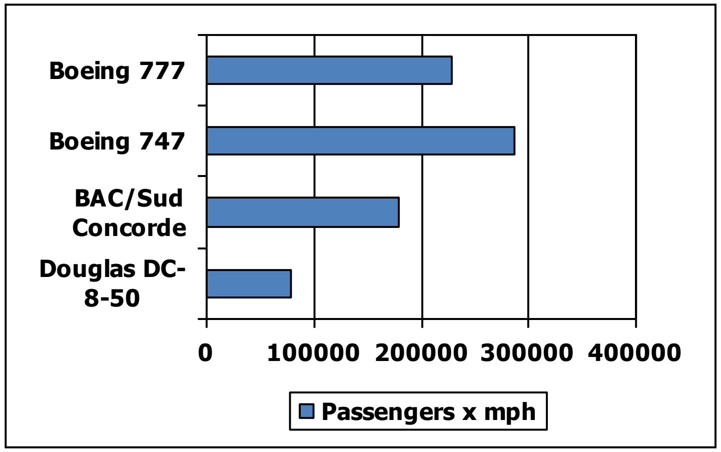
Defining Performance

- Which airplane has the best performance?









Standard Definition of Performance

- For some program running on machine X,

$$\boxed{\text{Performance}_X = 1 \,/\, \text{Execution time}_X}$$

- "X is n times faster than Y"

$$\boxed{\text{Performance}_X \,/\, \text{Performance}_Y = n}$$

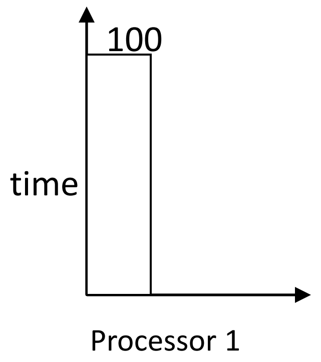
- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time$_B$ / Execution Time$_A$ = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

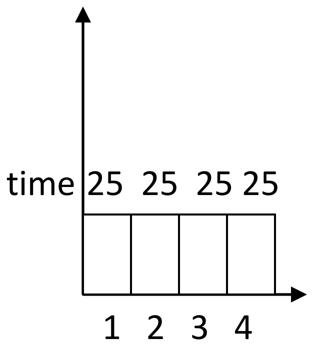
Speedup

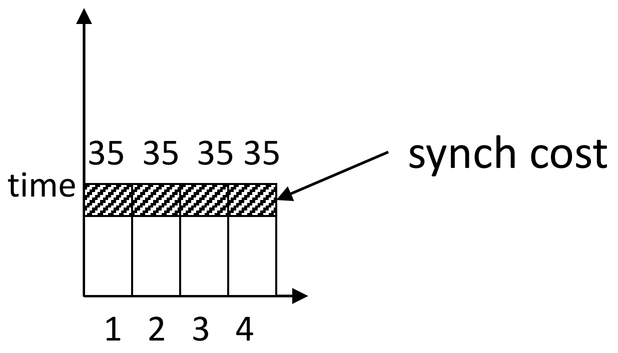
- Number of cores = p
- Serial run-time = T_{serial}
- Parallel run-time = $T_{parallel}$

$$S = \frac{T_{serial}}{T_{parallel}}$$

Example







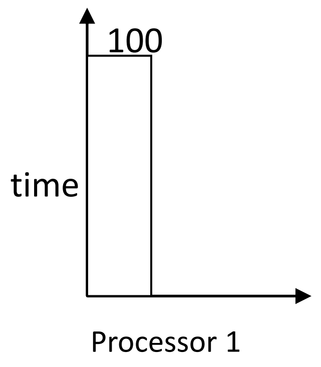
$$S_p = \frac{100}{25} = 4.0,$$

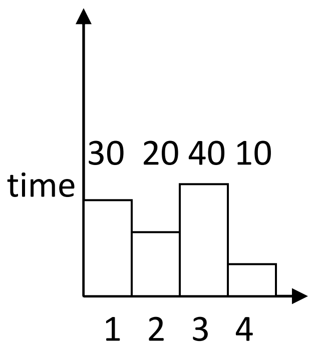
Perfect parallelization!
Does it ever occur?

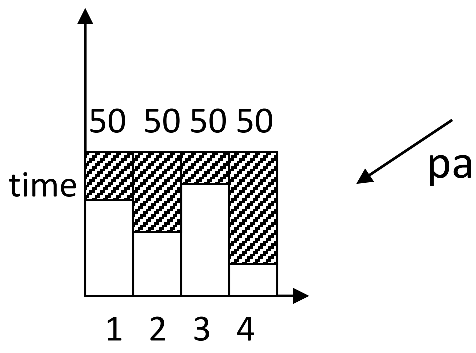
$$S_p = \frac{100}{35} = 2.85,$$

perfect load balancing

Example (cont.)







closest to
real life
parallel programs

$$S_p = \frac{100}{40} = 2.5,$$

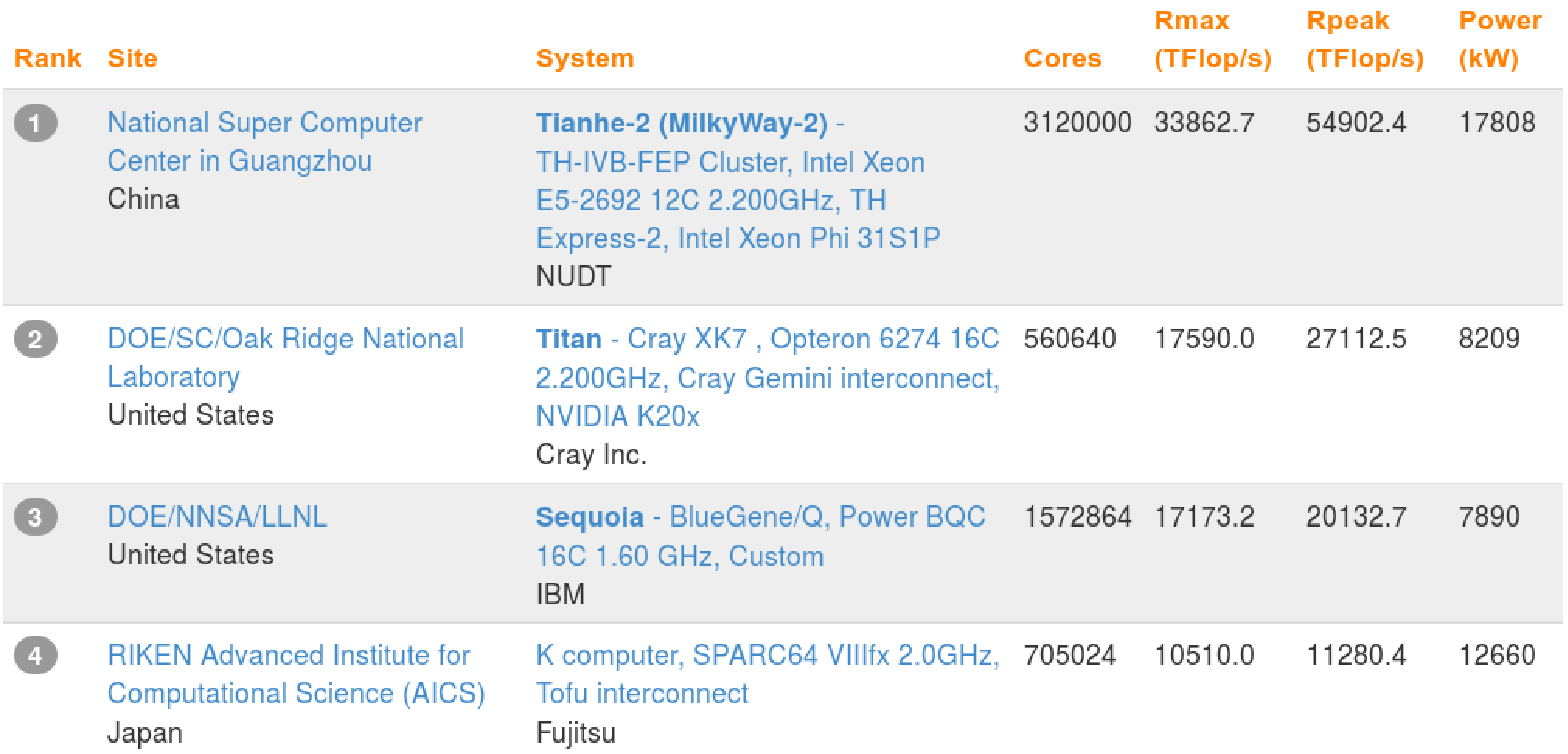
load imbalance

$$S_p = \frac{100}{50} = 2.0,$$

load imbalance

and sync cost

A Glimpse at the Top 500



Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3120000	33862.7	54902.4	17808
2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560640	17590.0	27112.5	8209
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1572864	17173.2	20132.7	7890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705024	10510.0	11280.4	12660

Rmax: Maximal achieved
Rpeak: Theoretical peak

Sources of Parallel Overheads

- Overhead of creating threads/processes
- Synchronization
- Load imbalance
- Communication
- Extra computation
- Memory access (for both sequential and parallel!)

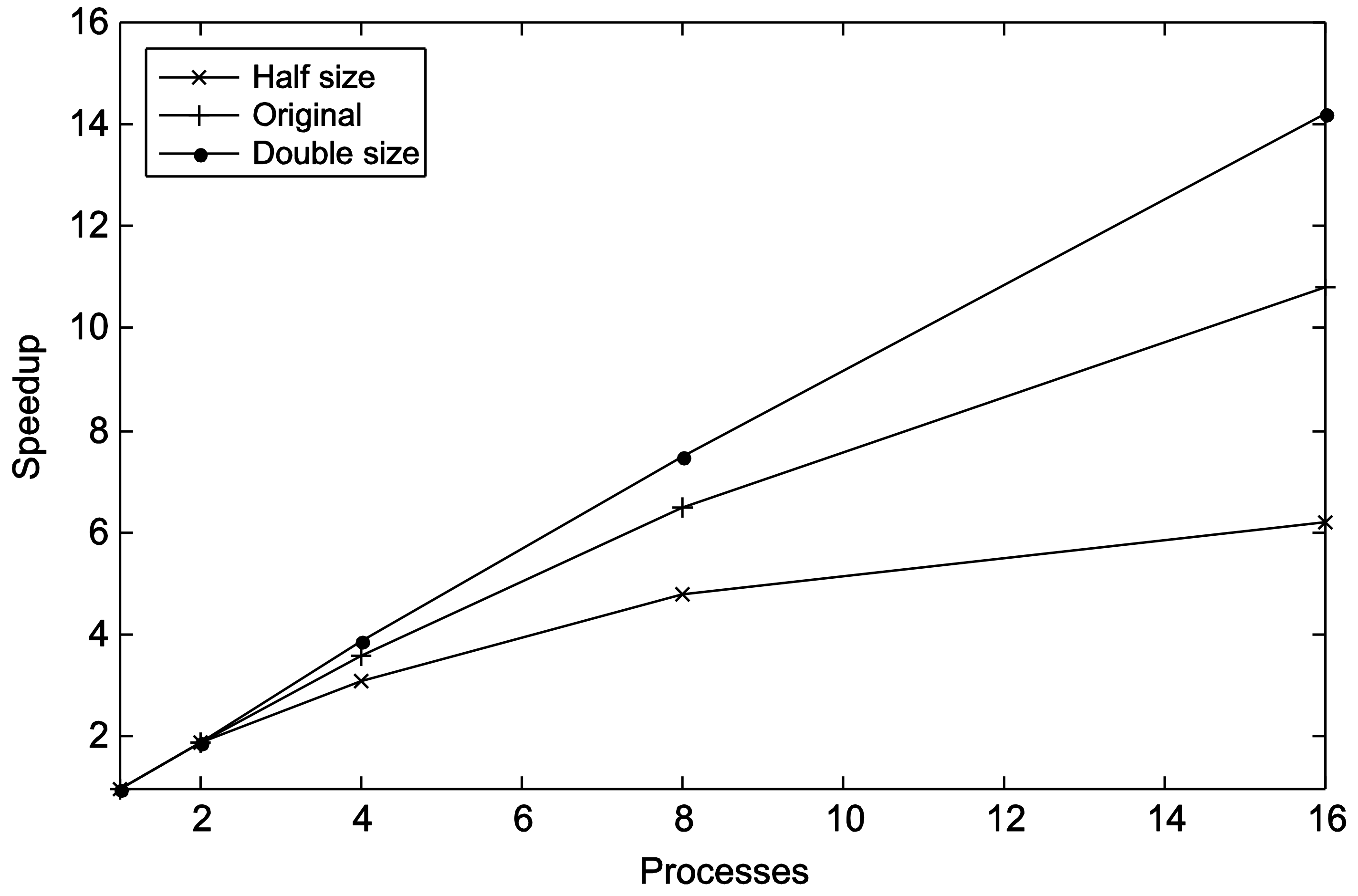
Efficiency of a parallel program

$$E = \frac{S}{p} = \frac{\left(\frac{T_{serial}}{T_{parallel}}\right)}{p}$$

Be Careful about T

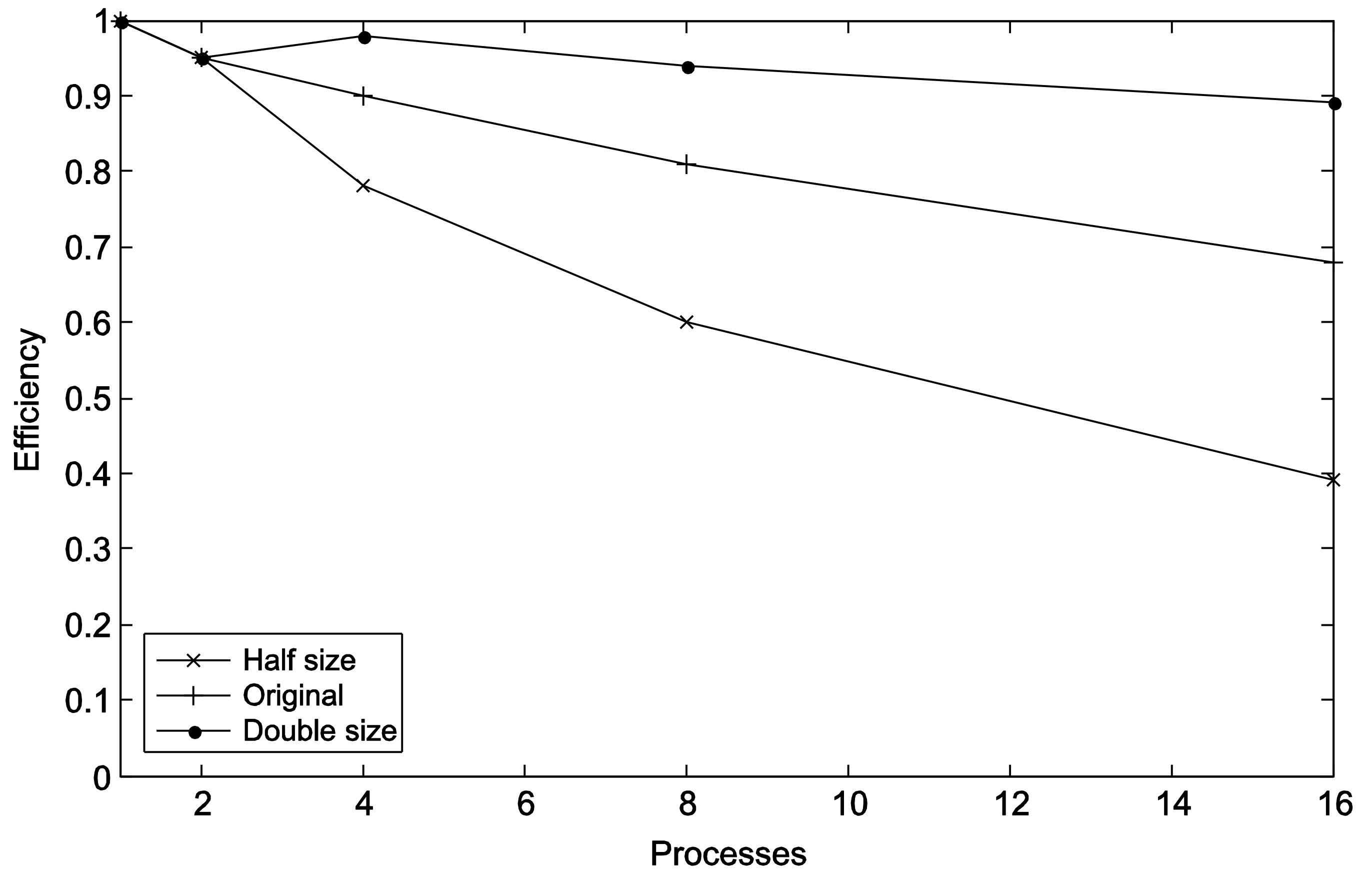
- Both T_{seq} and T_{par} are wall-clock times, and as such they are not objective. They can be influenced by :
  - The skill of the programmer who wrote the implementations
  - The choice of compiler (e.g. GNU C++ versus Intel C++)
  - The compiler switches (e.g. turning optimization on/off)
  - The operating system
  - The type of filesystem holding the input data (e.g. EXT4, NTFS, etc.)
  - The time of day... (different workloads, network traffic, etc.)

Speedup



Half size
Original
Double size
Speedup
Processes
16
14
12
10
8
6
4
2
2
4
6
8
10
12
14
16

Efficiency


Efficiency
Processes
Half size
Original
Double size
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
2
4
6
8
10
12
14
16

Scalability

- Scalability is the ability of a (software or hardware) system to handle a growing amount of work efficiently.
- If we keep the efficiency fixed by increasing the number of processes/threads and without increasing problem size, the problem is *strongly scalable*.
- If we keep the efficiency fixed by increasing the problem size at the same rate as we increase the number of processes/threads, the problem is *weakly scalable*.

Scalability

With scalability we want two things:

- Make use of more computing resources

$$strongScalingEfficiency(N) = \frac{t_{seq}}{N \cdot t_{par}}$$

- Solving bigger problems

$$weakScalingEfficiency(N) = \frac{t_{seq}}{t'_{par}}$$

t'_{par} is the time to solve a N-times bigger problem than t_{seq}

Let's take a closer look at timing.

Taking Timings

- What is time?
- Start to finish?
- A program segment of interest?
- CPU time?
- Wall clock time?

Execution Time

- **Elapsed Time**
  - counts everything *(disk and memory accesses, I/O , etc.)*
  - a useful number, but often not good for comparison purposes
- **CPU time**
  - doesn't count I/O or time spent running other programs
  - can be broken up into system time, and user time
- Our focus: **user CPU time**
  - time spent executing the lines of code that are "in" our program

$$\begin{tikzcd}
 & \text{Execution Time (Elapsed Time)} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{I/O Time} & \text{CPU Time} \arrow[dl] \arrow[d] & \text{Disk and Memory time} \\
\text{User CPU Time} & \text{System CPU Time} &
\end{tikzcd}$$

Taking Timings

In Linux:
time prog

Returns
real Xs
user Ys
sys Zs

Inside your C program:
clock_t clock(void)returns the number of clock ticks elapsed since the program started

```
#include <time.h>
#include <stdio.h>

int main() {
        clock_t  start, end, total;
        int i;

        start = clock();

        for(i=0; i< 10000000; i++) { }

        end = clock();
        total= (double)(end – start) / CLOCKS_PER_SEC;

      printf("Total time taken by CPU: %f\n", total);
}
```

Let's Look at Two Simple Metrics

- Response time (aka Execution Time)
  - The time between the start and completion of a task
- Throughput
  - Total amount of work done in a given time

What is the relationship between execution time and throughput?

Timing for sequential programs

MIPS

IPC

Execution Time

CPI

# insts

Execution time for sequential program:

$$\begin{tikzcd} \frac{\text{seconds}}{\text{program}} \arrow[d] & = & \frac{\text{cycles}}{\text{program}} \arrow[d] & \times & \frac{\text{seconds}}{\text{cycle}} \arrow[d] \\ ET & & IC * CPI & & CT \end{tikzcd}$$

$$\boxed{\text{ET = IC X CPI X CT}}$$

ET = Execution Time
CPI = Cycles Per Instruction
IC = Instruction Count

Example

A program runs in 10 seconds on computer A, which has a 4 GHz. clock. We are trying to help a computer designer build a new machine B, that will run this program in 6 seconds. The designer can use new (or perhaps more expensive) technology to substantially increase the clock rate, but has informed us that this increase will affect the rest of the CPU design, causing machine B to require 1.2 times as many clock cycles as machine A for the same program. What clock rate should we tell the designer to target?“

CPI Example

- Suppose we have two implementations of the same instruction set architecture (ISA).

  For some program,
  Machine A has a clock cycle time of 250 ps and a CPI of 2.0
  Machine B has a clock cycle time of 500 ps and a CPI of 1.2
  What machine is faster for this program, and by how much?

[10^{-3} = milli, 10^{-6} = micro, 10^{-9} = nano, 10^{-12} = pico, 10^{-15} = femto]

#Instructions Example

- A compiler designer is trying to decide between two code sequences for a particular machine. Based on the hardware implementation, there are three different classes of instructions: Class A, Class B, and Class C, and they require one, two, and three cycles (respectively).

  The first code sequence has 5 instructions:
  2 of A, 1 of B, and 2 of C

  The second sequence has 6 instructions:
  4 of A, 1 of B, and 1 of C.

  Which sequence will be faster? How much?
  What is the CPI for each sequence?

MIPS Example

- Two different compilers are being tested for a 4 GHz. machine with three different classes of instructions: Class A, Class B, and Class C, which require one, two, and three cycles (respectively). Both compilers are used to produce code for a large piece of software.

  The first compiler's code uses 5 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

  The second compiler's code uses 10 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

- Which sequence will be faster according to MIPS?
- Which sequence will be faster according to execution time?

Pitfalls in timing in Parallel Machines

For Multithreaded Programs

- Shall we use **execution time** or **throughput**? or both?
- IPC is not accurate here
  - small timing variations may lead to different execution
  - Order at which threads enter critical section may vary
  - Different interrupt timing may lead to different scheduling decisions

The total number of instructions executed may be different across different runs!

For Multithreaded Programs

The total number of instructions executed may be different across different runs!

This effect increases with the number of cores

System-level code account for a significant fraction of the total execution time

Your Program Does Not Run in A Vacuum

- OS at least is there.
- Multi-programming and/or mulithreading setting is very common in multicore settings
- Independent programs affect each other performance (why?)

How to check the performance of a parallel machine?

Benchmarks

- Performance best determined by running a real application
  - Use programs typical of expected workload
  - Or, typical of expected class of applications
  - e.g., compilers/editors, scientific applications, graphics, etc.
- Small benchmarks
  - nice for architects and designers
  - easy to standardize
- Parallel Benchmarks: PARSEC, Rodinia, SPLASH-2
- SPEC (System Performance Evaluation Cooperative)
  - companies have agreed on a set of real program and inputs
  - valuable indicator of performance (and compiler technology)

Role of Benchmarks

- help designer explore architectural designs
- identify bottlenecks
- compare different systems
- conduct performance prediction

Example: PARSEC

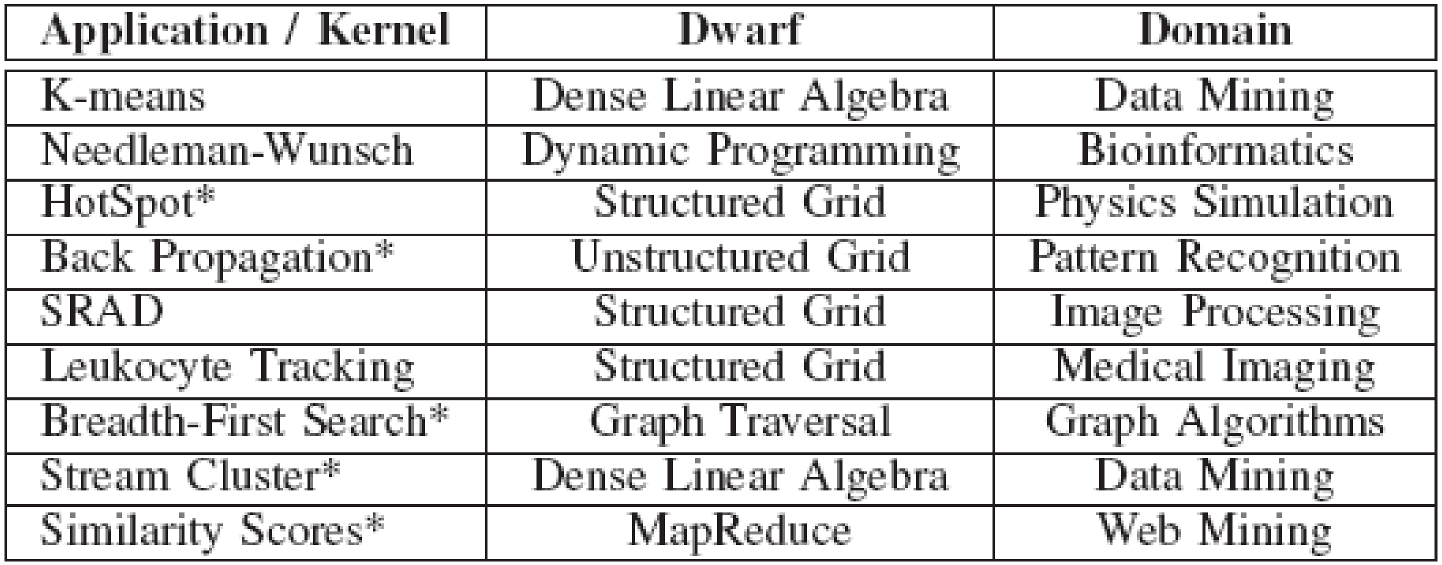
- Princeton Application Repository for Shared-Memory Computers
- Benchmark Suite for Chip-Multiprocessors
- Freely available at: http://parsec.cs.princeton.edu/
- Objectives:
  - Multithreaded Applications: Future programs must run on multiprocessors
  - Emerging Workloads: Increasing CPU performance enables new applications
  - Diverse: Multiprocessors are being used for more and more tasks
  - State-of-Art Techniques: Algorithms and programming techniques evolve rapidly

Example: PARSEC

Program	**Application Domain**	**Parallelization**
Blackscholes	Financial Analysis	Data-parallel
Bodytrack	Computer Vision	Data-parallel
Canneal	Engineering	Unstructured
Dedup	Enterprise Storage	Pipeline
Facesim	Animation	Data-parallel
Ferret	Similarity Search	Pipeline
Fluidanimate	Animation	Data-parallel
Freqmine	Data Mining	Data-parallel
Streamcluster	Data Mining	Data-parallel
Swaptions	Financial Analysis	Data-parallel
Vips	Media Processing	Data-parallel
X264	Media Processing	Pipeline

Example: Rodinia

- A Benchmark Suite for Heterogeneous Computing: multicore CPU and GPU
- University of Virginia



Application / Kernel	Dwarf	Domain
K-means	Dense Linear Algebra	Data Mining
Needleman-Wunsch	Dynamic Programming	Bioinformatics
HotSpot*	Structured Grid	Physics Simulation
Back Propagation*	Unstructured Grid	Pattern Recognition
SRAD	Structured Grid	Image Processing
Leukocyte Tracking	Structured Grid	Medical Imaging
Breadth-First Search*	Graph Traversal	Graph Algorithms
Stream Cluster*	Dense Linear Algebra	Data Mining
Similarity Scores*	MapReduce	Web Mining

Conclusions

- Performance evaluation is very important to assess programming quality as well as the underlying architecture and how they interact.
- The following capture some aspects of the system but do not represent overall performance: MIPS, #instructions, #cycles, frequency
- Execution time is what matters: system time, CPU time, I/O and memory time
- Scalability and efficiency measure the quality of your code.