



CSCI-UA.0480-003

# Parallel Computing

## Lecture 7: Performance Analysis

Mohamed Zahran (aka Z)

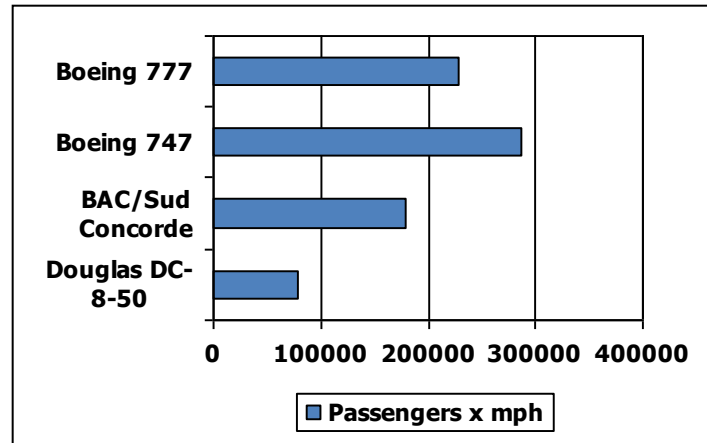
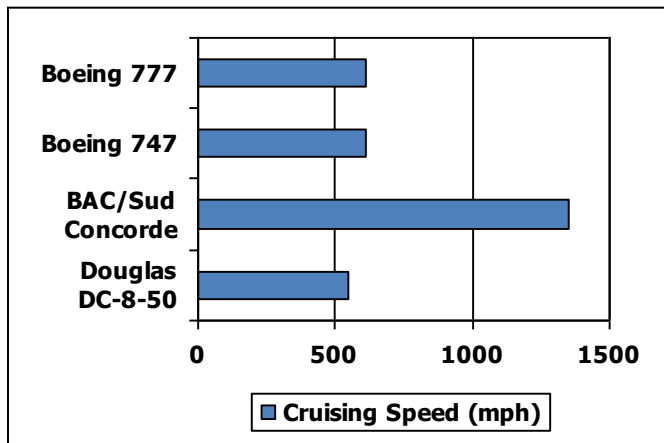
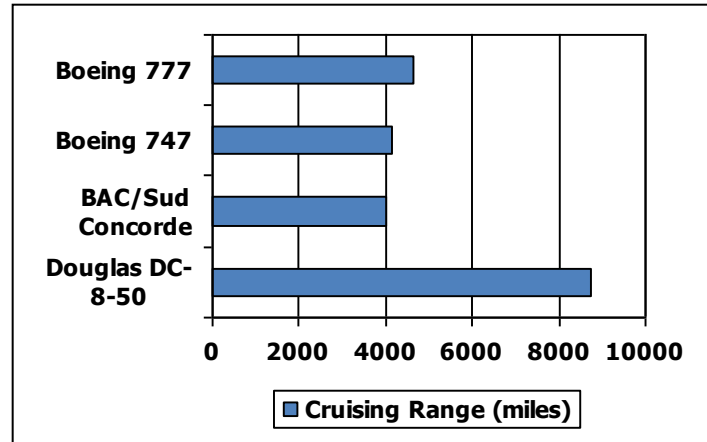
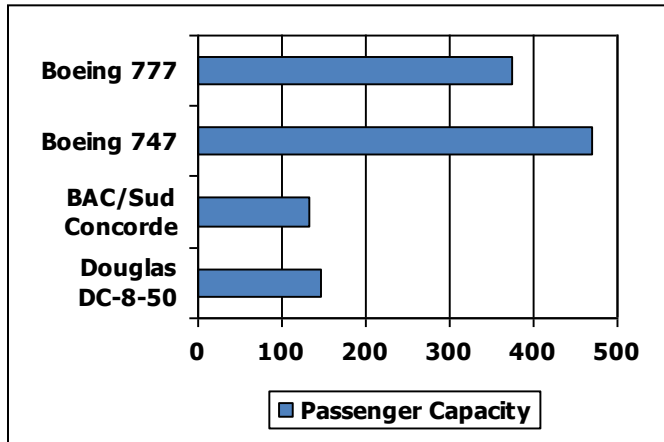
mzahran@cs.nyu.edu

<http://www.mzahran.com>



# Defining Performance

- Which airplane has the best performance?



# Standard Definition of Performance

- For some program running on machine X,

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

- "X is n times faster than Y"

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

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

# Speedup



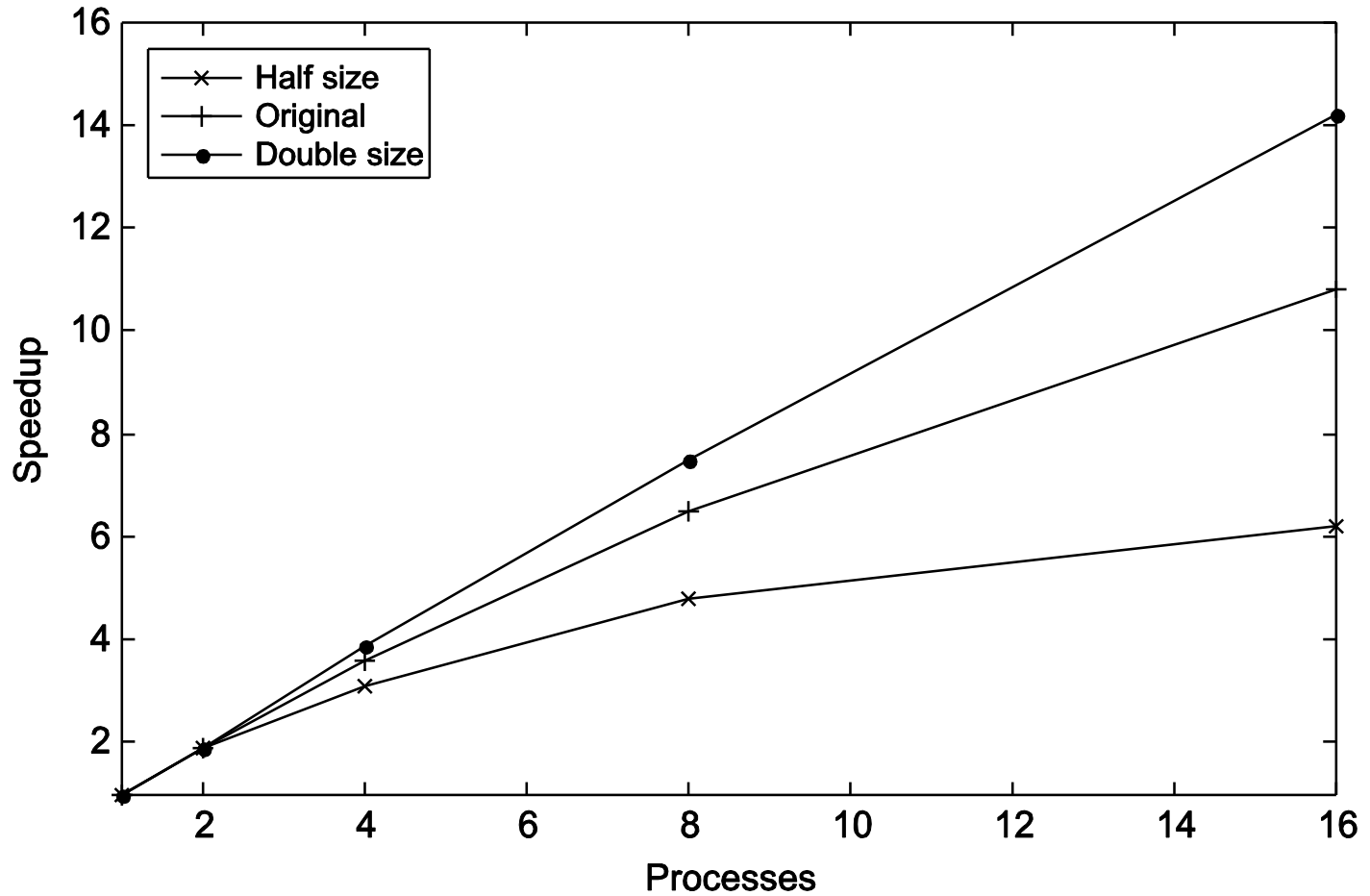
- Number of cores =  $p$
- Serial run-time =  $T_{\text{serial}}$
- Parallel run-time =  $T_{\text{parallel}}$

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

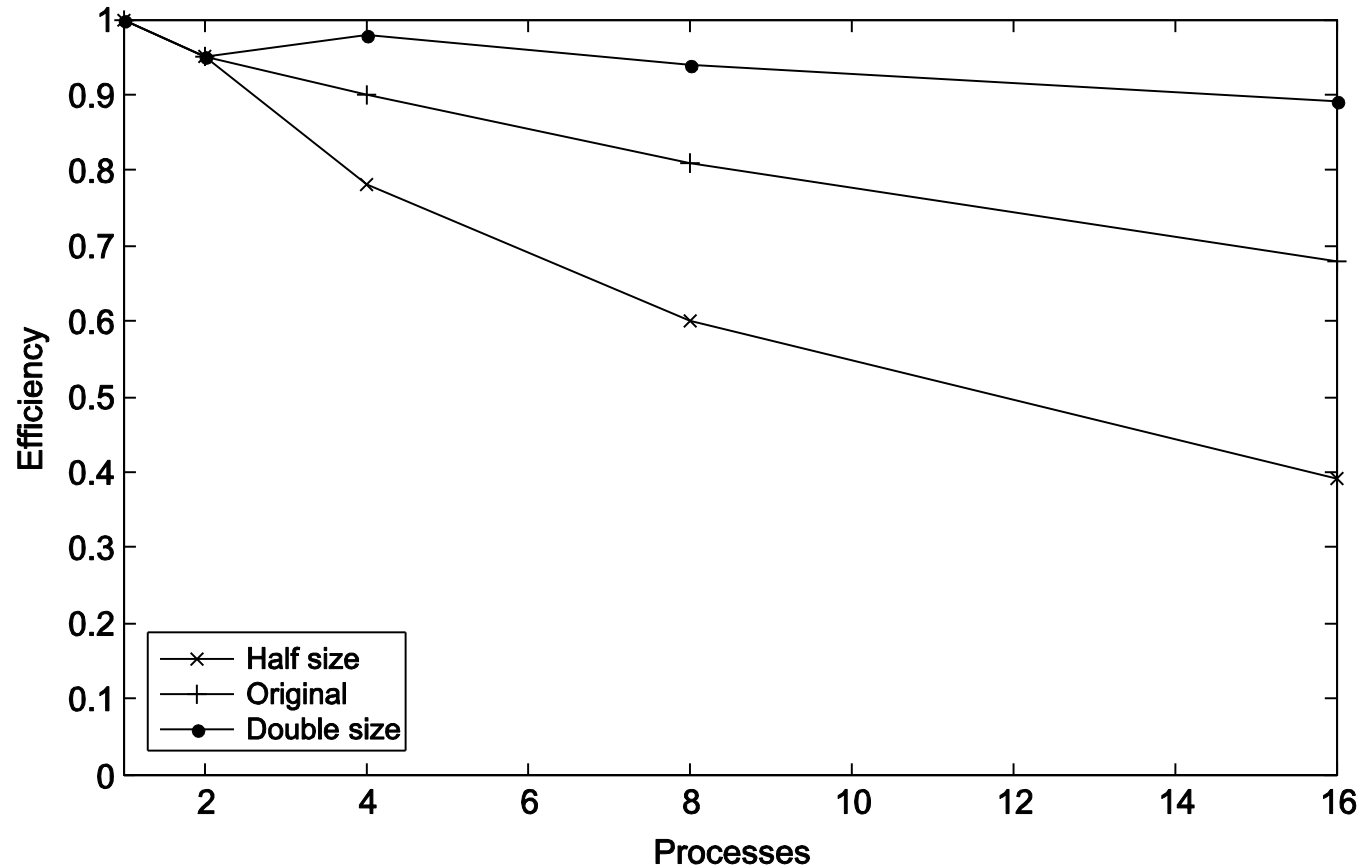
# Efficiency of a parallel program

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

# Speedup



# Efficiency



# Scalability

- In general, a problem is *scalable* if it can handle ever increasing problem sizes.
- If we increase the number of processes/threads and keep the efficiency fixed 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*.



# 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

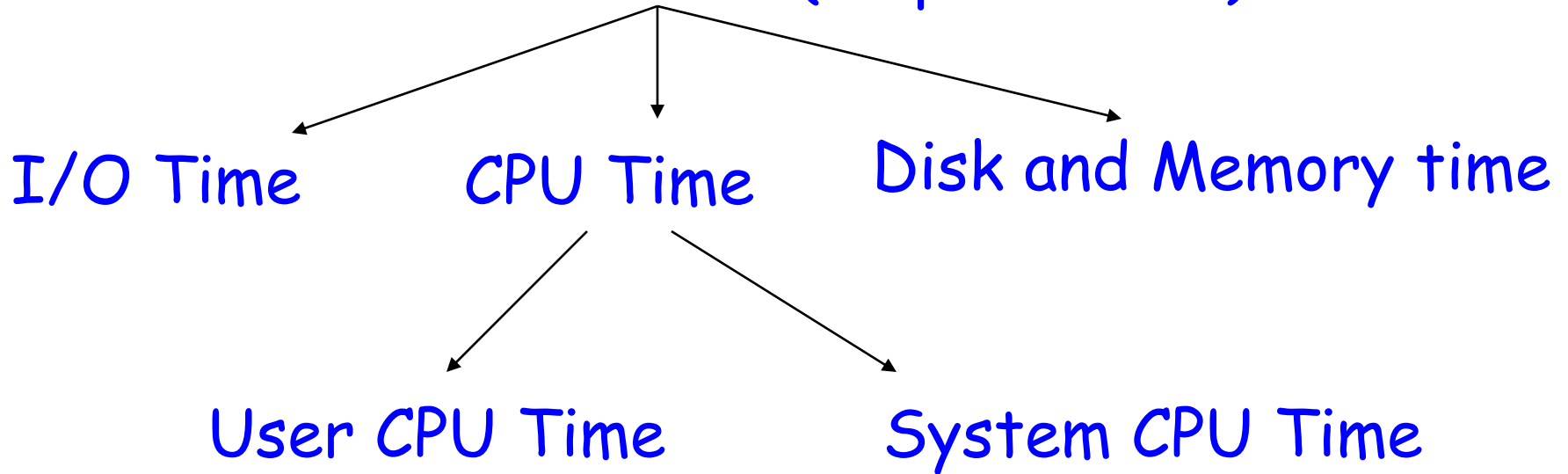
- **CPU time**

- 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

# Execution Time (Elapsed Time)



# 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?

# Never forget this:

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

ET

IC \* CPI

CT

$$ET = IC \times CPI \times CT$$

ET = Execution Time

CPI = Cycles Per Instruction

IC = Instruction Count

# An Interesting Question

- *If two machines have the same ISA which of our quantities (e.g., clock rate, CPI, execution time, # of instructions, MIPS) will always be identical?*

# Example

Our favorite 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?“



# Performance

- Performance is determined by **execution time**
- Do any of the other variables equal performance?
  - # of cycles to execute program?
  - # of instructions in program?
  - # of cycles per second?
  - average # of cycles per instruction?
  - average # of instructions per second?

# 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?

# 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

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

# 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.