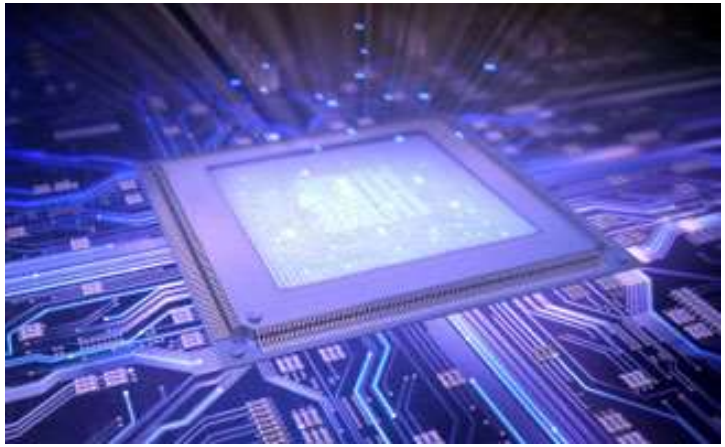
NYU
New York University

Parallel Computing

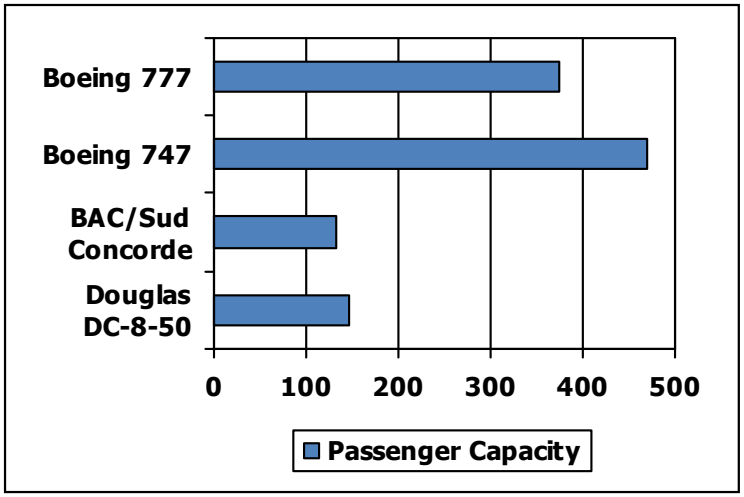
Performance Analysis

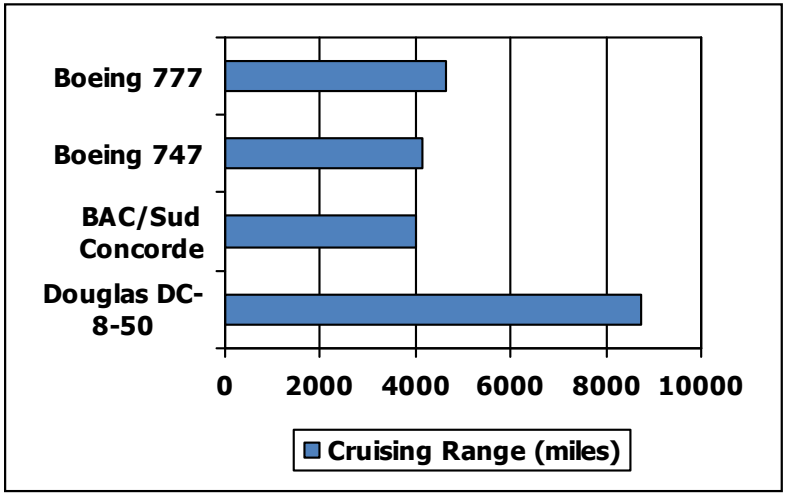
Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com

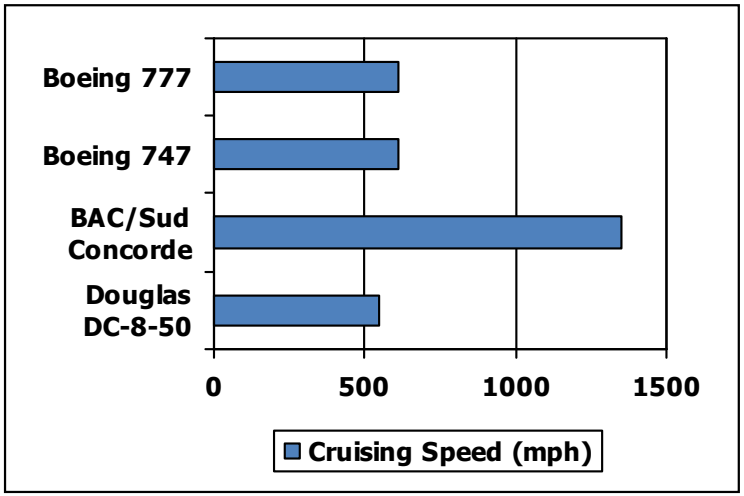


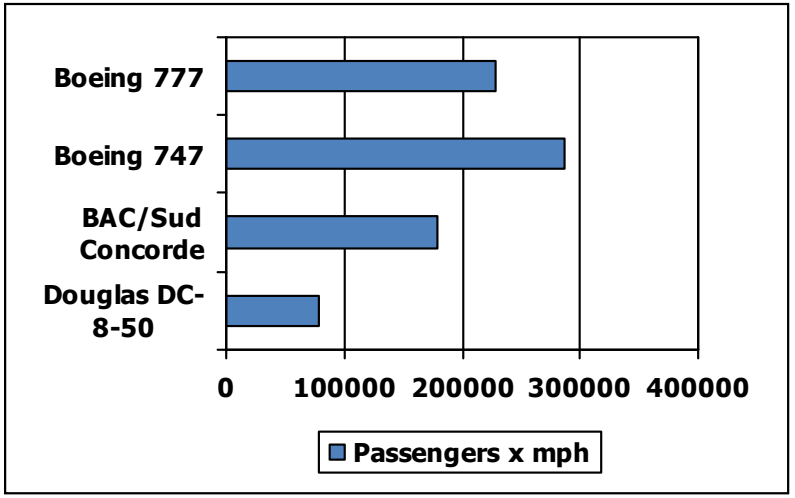
Defining Performance

- 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 on machine A and machine B
    - 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

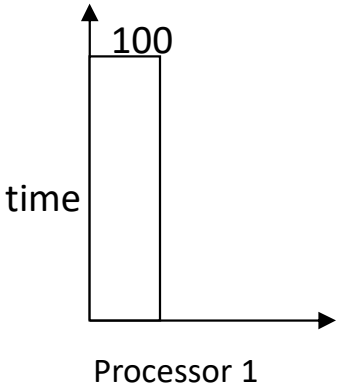
Speedup

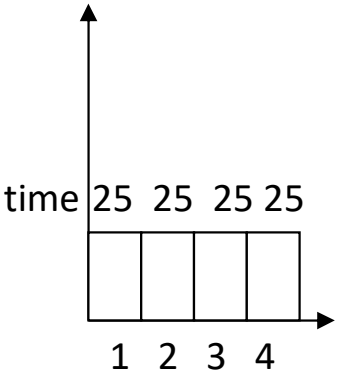
- Number of cores = p
- Serial run-time = T_{serial}
- Parallel run-time = $T_{parallel}$

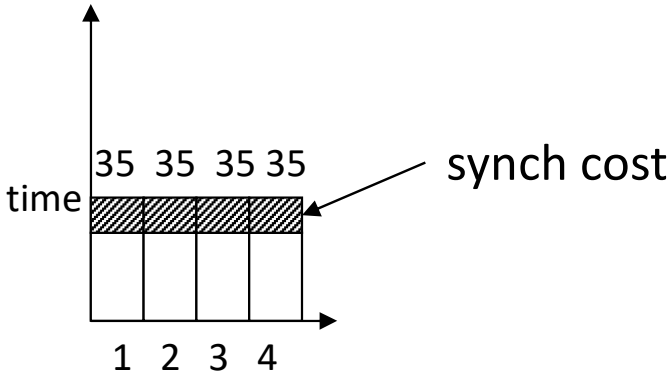
$$S = \frac{T_{serial}}{T_{parallel}}$$

Copyright © 2010, Elsevier Inc.
All rights Reserved

Example







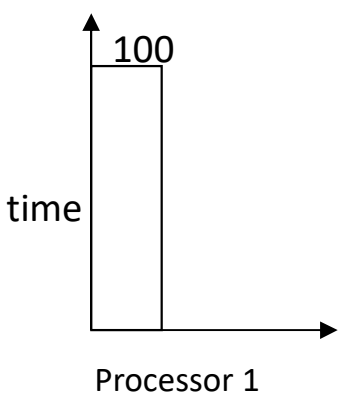
$$S_p = \frac{100}{25} = 4.0,$$

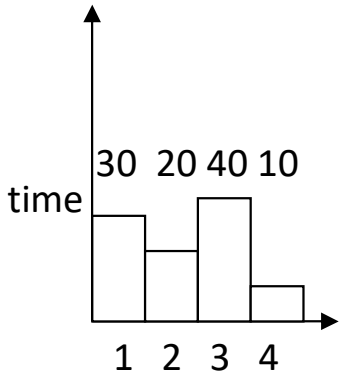
Perfect parallelization!
Does it ever occur?

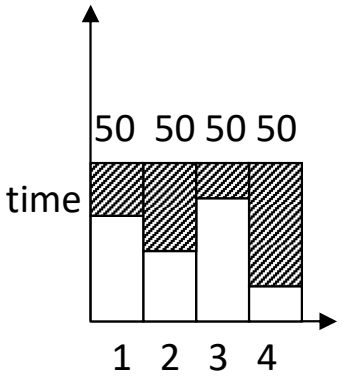
$$S_p = \frac{100}{35} = 2.85,$$

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 (Nov 2021 List)

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
5	**Perlmutter** - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	761,856	70,870.0	93,750.0	2,589

Rmax: maximum achieved
Rpeak: theoretical maximum

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

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

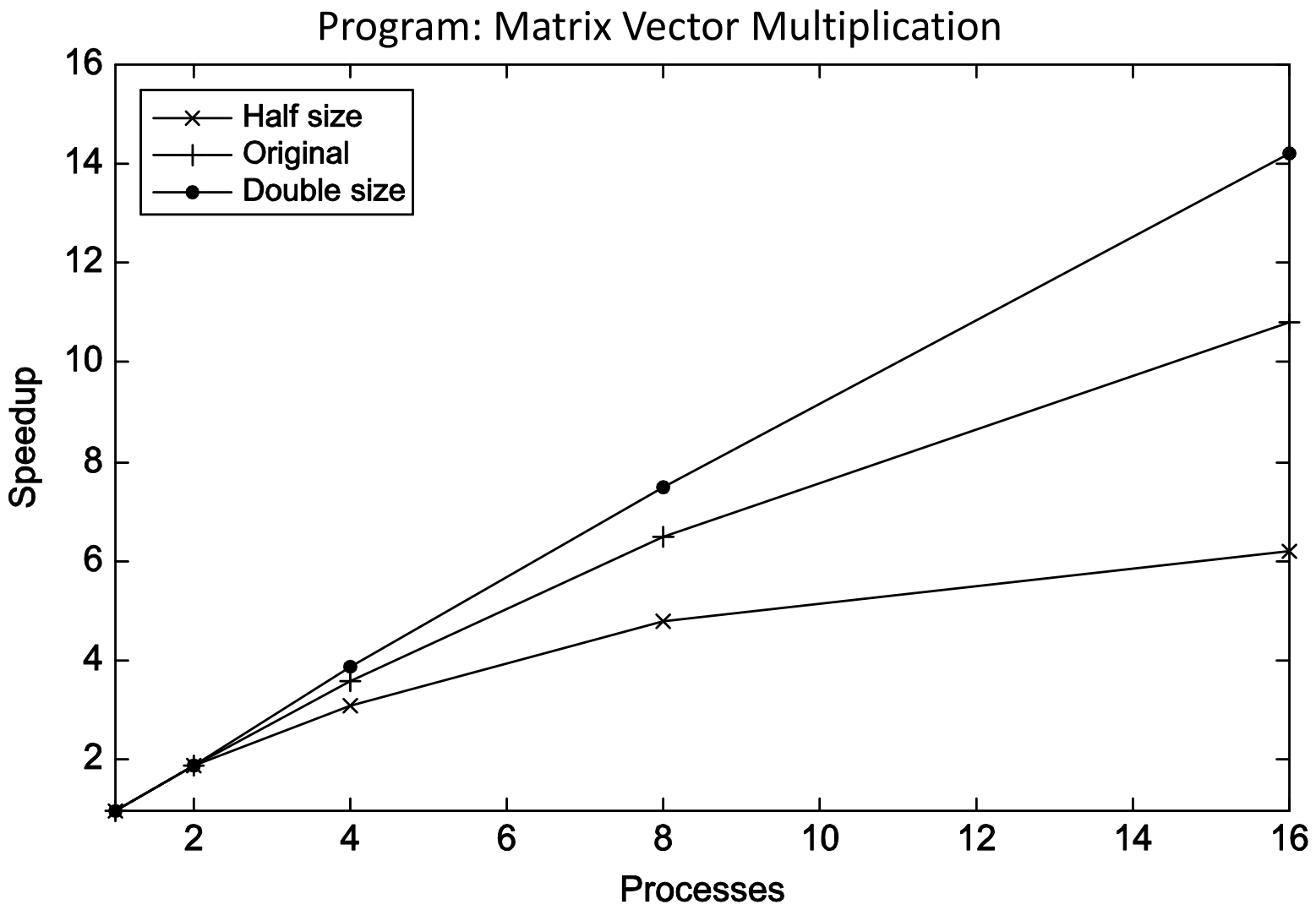
P = can be number of threads, processes, or cores

Copyright © 2010, Elsevier Inc.
All rights Reserved

Be Careful about T

- Both T_{serial} 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.)

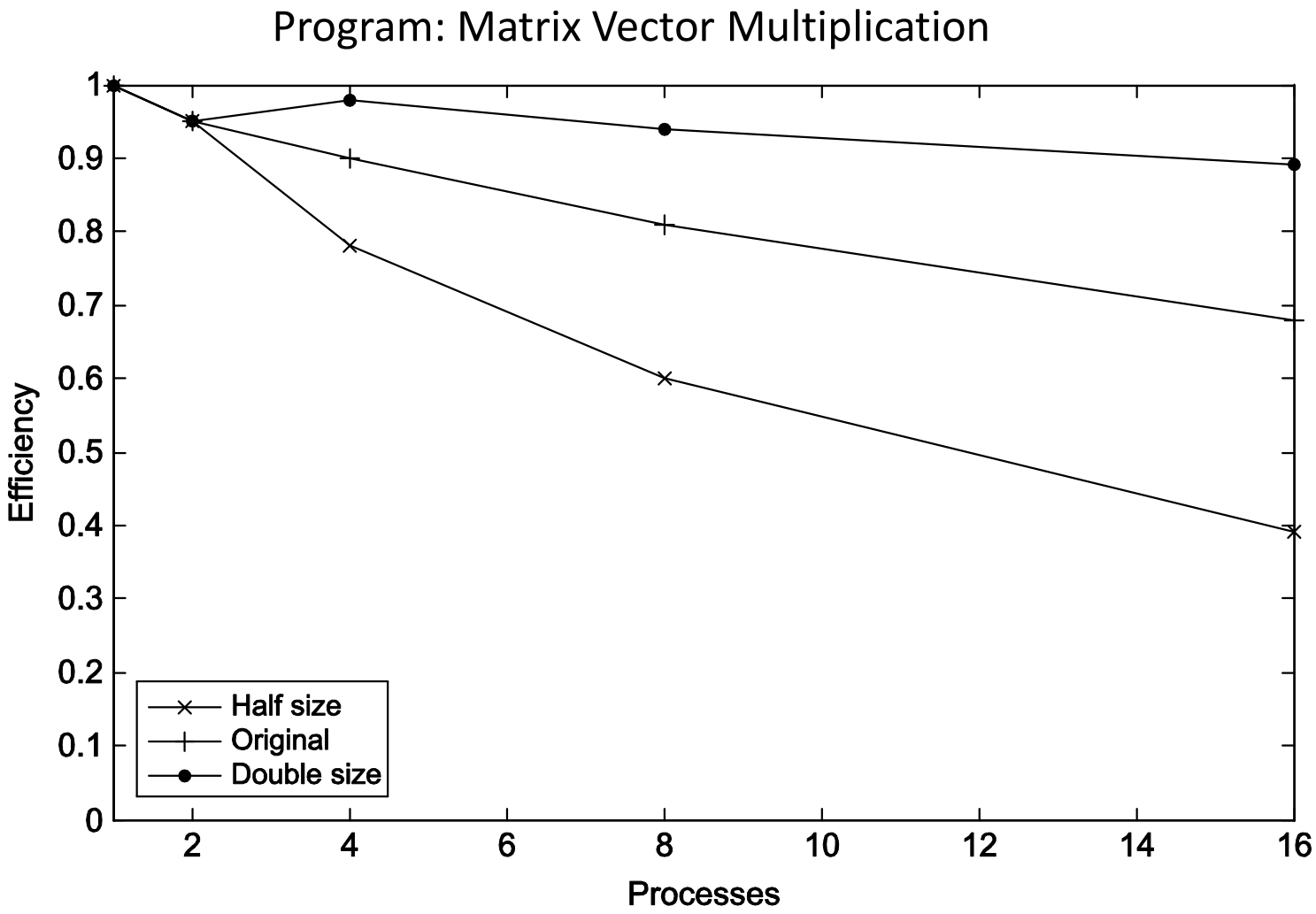
Speedup



Graph shows how speedup is affected by the number of processes and problem size.

Copyright © 2010, Elsevier Inc.
All rights Reserved

Efficiency



Copyright © 2010, Elsevier Inc.
All rights Reserved

Scalability

$$E = \frac{\text{Speedup}}{P} = \frac{\frac{T_{serial}}{T_{parallel}}}{p}$$

- 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*.

Copyright © 2010, Elsevier Inc.
All rights Reserved

Let's take a closer look at timing.

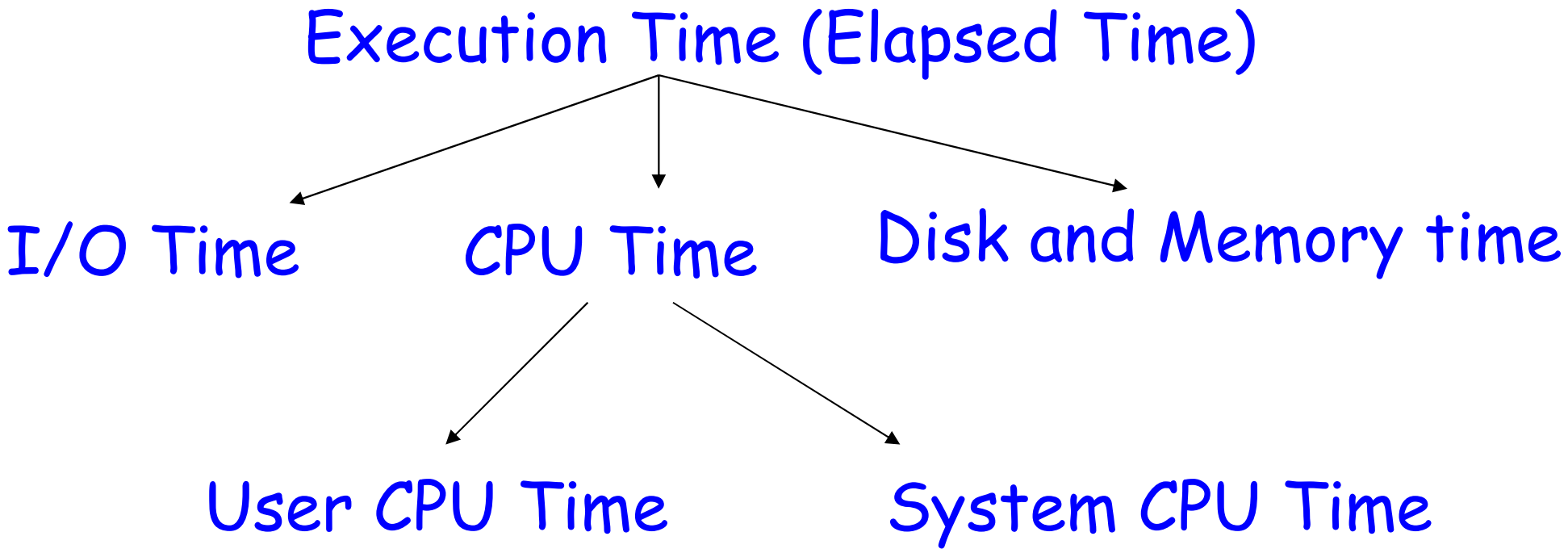
Taking Timings

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

Copyright © 2010, Elsevier Inc.
All rights Reserved

Execution Time

- **Elapsed Time (aka wall-clock 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)
I/O Time
CPU Time
Disk and Memory time
User CPU Time
System CPU 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

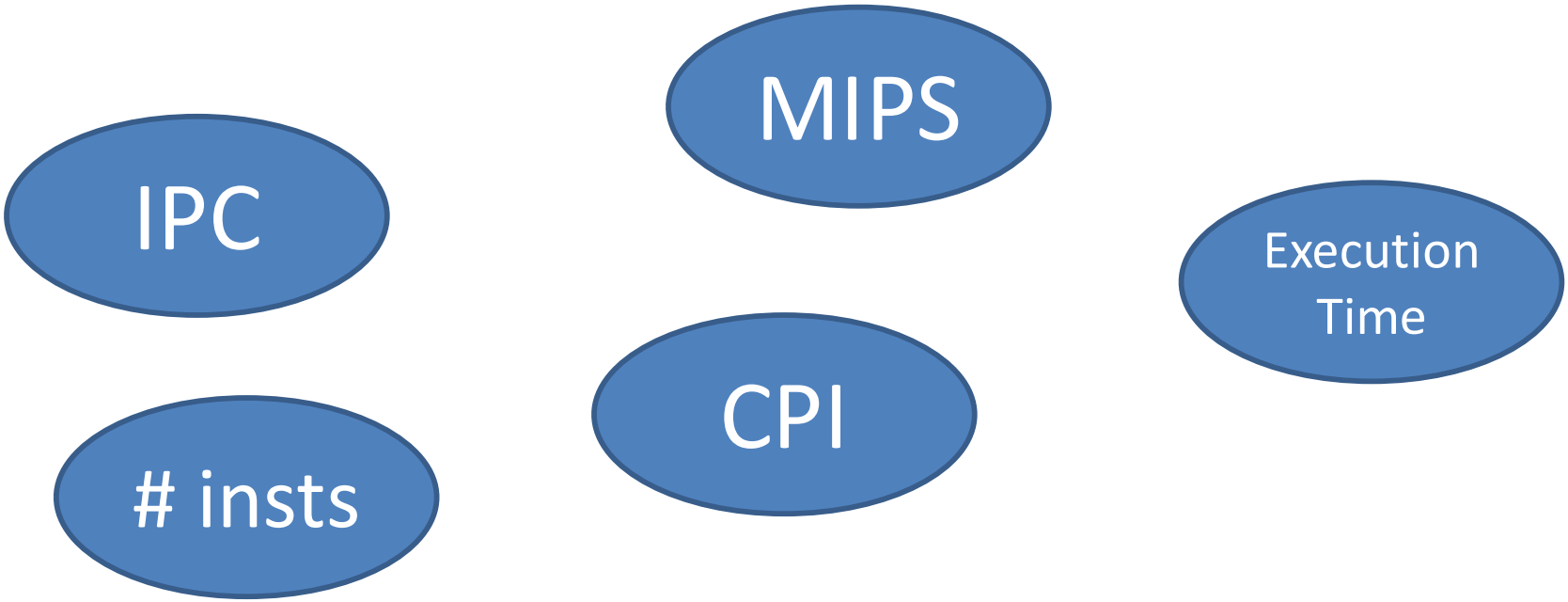
```
#include <time.h>
#include <stdio.h>

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?

Timing for sequential programs



Execution time for sequential program:

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

ET

IC * CPI

CT

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?“

ET = IC * CPI * CT = total_cycles * CT

A: 10 = total_cycles * (1/4GHz)
B: 6 = 1.2 * total_cycles * (1/clock_rate)

10/6 = clock_rate/(1.2 * 4GHz)
Clock_rate = 8GHz

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]

ET = IC*CPI*CT

ET_A = IC * 2*250 = 500IC

ET_B = IC * 1.2 * 500 = 600IC

#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?

Same machine = CT is the same for both sequences.

#Instructions Example (cont'd)

ET = IC * CPI * CT = total number of cycles * CT

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

total cycles = (2*1) + (1*2) + (2*3) = 10

ET = 10*CT

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

total cycles = (4*1) + (1*2) + (1*3) = 9

ET = 9*CT

Second sequence is faster.

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

CPI = Cycles per Instructions = #cycles / # instructions

For 1st sequence = 10/(2+1+2) = 2

For 2nd sequence = 9/6 = 1.5

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?

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?

MIPS = Million Instructions per Second = # instructions in millions / total time in seconds

Total time in seconds = total number of cycles * cycle time = total number of cycles/Frequency

1st compiler:

total #cycles = $((5*1)+(1*2)+(1*3) * 10^6) = 10^7$

Frequency = $4* 10^9$

MIPS = $7/(10^7 / 4* 10^9) = 2800$

2nd compiler:

total #cycles = $((10*1)+(1*2)+(1*3) * 10^6)$

$= 15* 10^6$

MIPS = $12/(15* 10^6 / 4* 10^9) = 3200$

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?

ET = IC * CPI * CT = total number of cycles * CT = total number of cycles/frequency

1st compiler:
total #cycles = $((5*1)+(1*2)+(1*3) * 10^6) = 10^7$
Frequency = $4* 10^9$
ET = $10^7 /(4* 10^9) = 1/400$

2nd compiler:
total #cycles = $((10*1)+(1*2)+(1*3) * 10^6)$
$= 15* 10^6$
ET = $15* 10^6 /(4* 10^9) = 1.5/400$

Pitfalls in timing in Parallel Machines

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

You need to decide: Shall we use **execution time? throughput**? or both?

The decision is application specific.

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
  - Companies have agreed on a set of real program and inputs
  - e.g., compilers/editors, scientific applications, graphics, etc.
- Example of benchmarks:
  - Parallel Benchmarks: **PARSEC**, **Rodinia**, **SPLASH-2**
  - Sequential benchmarks: **SPEC** (System Performance Evaluation Cooperative)

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
  - To know whether your execution time is good, you need to compare it with sequential code, another parallel code, etc.
- Scalability and efficiency measure the quality of your code.