CS4379: Parallel and Concurrent Programming
CS5379: Parallel Processing

Lecture 6

Dr. Yong Chen
Associate Professor
Computer Science Department
Texas Tech University

Course Info

- **Lecture Time**: TR, 12:30-1:50
- **Lecture Location**: ECE 217
- **Sessions**: CS4379-001, CS4379-002, CS5379-001, CS5379-D01
- **Instructor:** Yong Chen, Ph.D., Associate Professor
- **Email:** yong.chen@ttu.edu
- **Phone:** 806-834-0284
- **Office**: Engineering Center 315
- **Office Hours**: 2-4 p.m. on Wed., or by appointment
- **TA:** Mr. Ghazanfar Ali, Ghazanfar.Ali@ttu.edu
- **TA Office hours:** Tue. and Fri., 2-3 p.m., or by appointment
- **TA Office:** EC 201 A
- **More info:**
  - http://www.myweb.ttu.edu/yonchen
  - http://discl.cs.ttu.edu; http://cac.ttu.edu/; http://nsfcac.org

Announcements

- Quiz#2 planned to be conducted on Feb. 6th, covering Lecture#5 (previous lecture), #6 (this lecture), #7 (next lecture on Feb. 6th)
- Two guest lectures by Mr. Misha Ahmadian next week on the subject of how to use parallel computers on campus

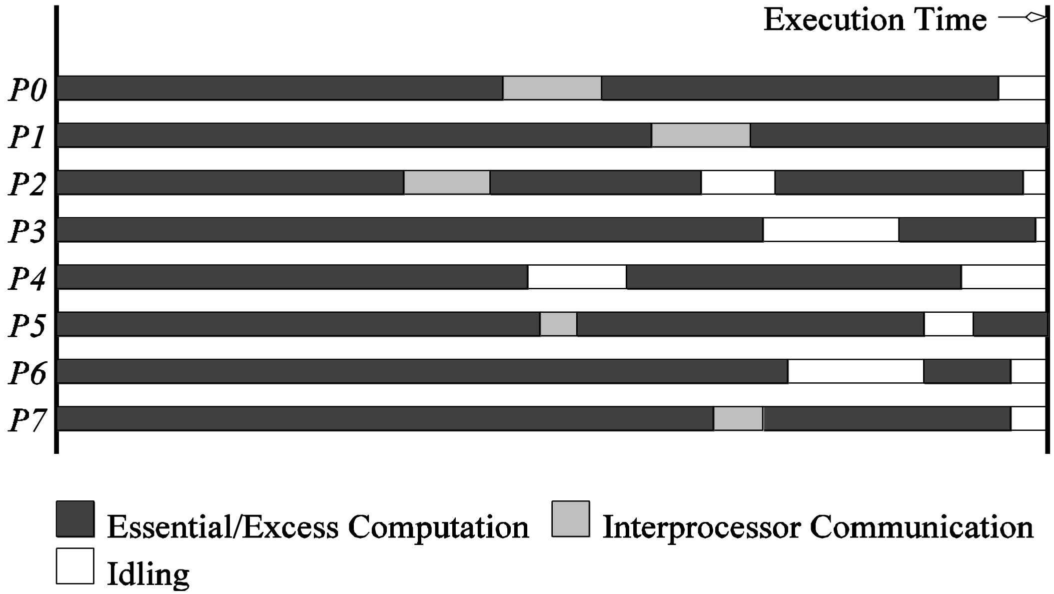
Outline

- Questions?
- Parallel performance modeling and evaluation
  - Execution time and total parallel overhead
  - Speedup and Amdahl's law
  - Efficiency
- Example of computing π
- Amdahl's law revisited and scaled speedup

Sources of Overhead in Parallel Programs

- If I use two processors, shouldn't my program run twice as fast?

- No (rarely the case) - a number of overheads associated with parallelism, such as extra computation, communication, and idling, that cause degradation in performance.



The execution profile of a hypothetical parallel program executing on eight processing elements. Profile indicates times spent performing computation (both essential and excess), communication, and idling.

Sources of Overheads in Parallel Programs

- Interprocess communication
  - Processors working on any non-trivial parallel problem will need to talk to each other.
- Idling
  - Processes may idle because of load imbalance, synchronization, or serial components.
- Extra computation
  - This is computation not performed by the serial version
  - This might be because the serial algorithm is difficult to parallelize, or that some computations are repeated across processors to minimize communication.

Performance Metrics for Parallel Systems: Execution Time

- Serial runtime of a program is the time elapsed between the beginning and the end of its execution on a sequential computer.

- The parallel runtime?
  - The time that elapses from the moment the first processor starts to the moment the last processor finishes execution.

- We denote the serial runtime by $\boldsymbol{T_S}$ and the parallel runtime by $\boldsymbol{T_P}$.

Performance Metrics for Parallel Systems: Total Parallel Overhead

- Let T_{all} be the total time collectively spent by all the processing elements.
- T_S is the serial time.
- Observe that T_{all} - T_S is then the total time spend by all processors combined in non-useful work. This is called the *total overhead*.
- The total time collectively spent by all the processing elements
  $T_{all} = p\ T_P$ (p is the number of processors).
- The overhead function (T_o) is therefore given by

$$T_o = p\ T_P - T_S$$

Performance Metrics: Speedup

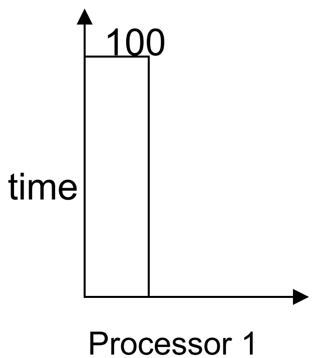
- For a given problem, there might be many serial algorithms available. These algorithms may have different runtimes and may be parallelizable to different degrees.

- For the purpose of computing speedup, we consider the best sequential program or the known fastest as the baseline

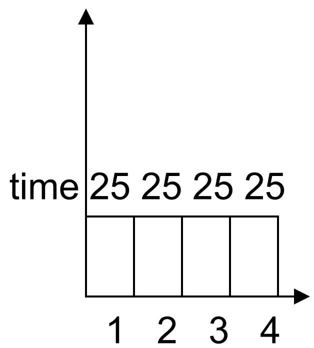
- T_s =time for the serial algorithm
- T_p=time for parallel algorithm using p processors

$$S_p = \frac{T_s}{T_p}$$

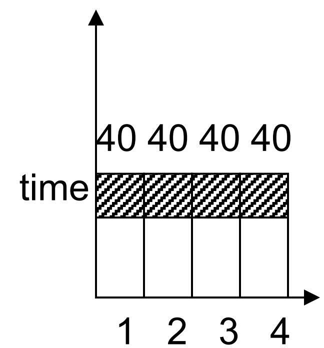
Example



(a)



(b)



(c)

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

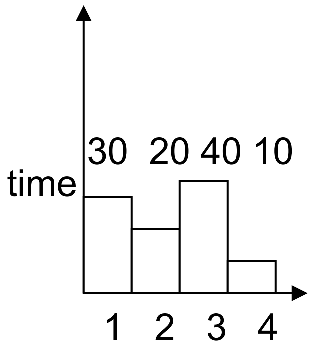
Perfect parallelization and load balancing

"embarrassingly parallel"

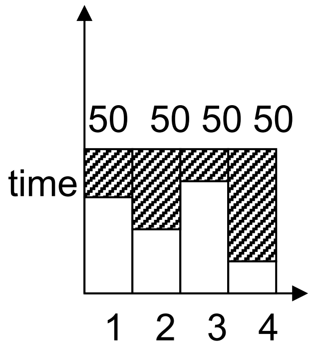
$$S_p = \frac{100}{40} = 2.5$$

Perfect parallelization and load balancing but sync cost is 15

Example (cont.)



(d)



(e)

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

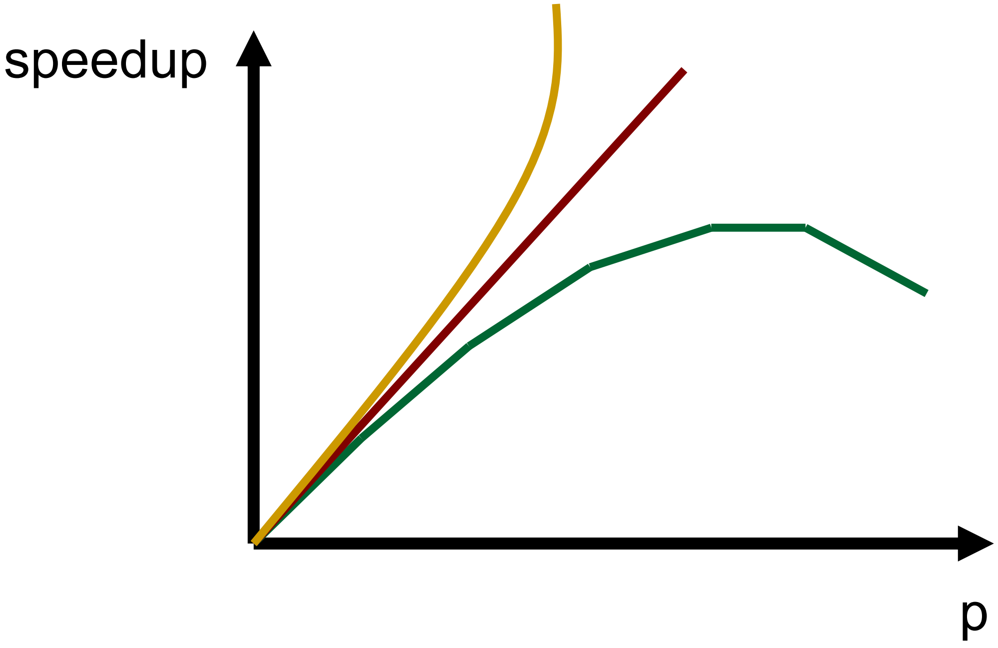
No sync cost but load imbalance

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

Load imbalance and sync cost

Cases of Speedup

- *Linear* speedup
- *Superlinear* speedup
- Sub-linear speedup



Amdahl's Law

- Gene M. Amdahl, "*Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities*", 1967
- Analyze the limit of the performance improvement that can be gained by a parallel system
- Assume a problem has a serial ratio of α (cannot be parallelized), serial runtime T_S
- Then, parallel runtime can be written as (*p* processors)?

$$\alpha\, T_s + (1-\alpha) T_s / p$$

Amdahl's Law

- The speedup that is achievable on p processors is ?

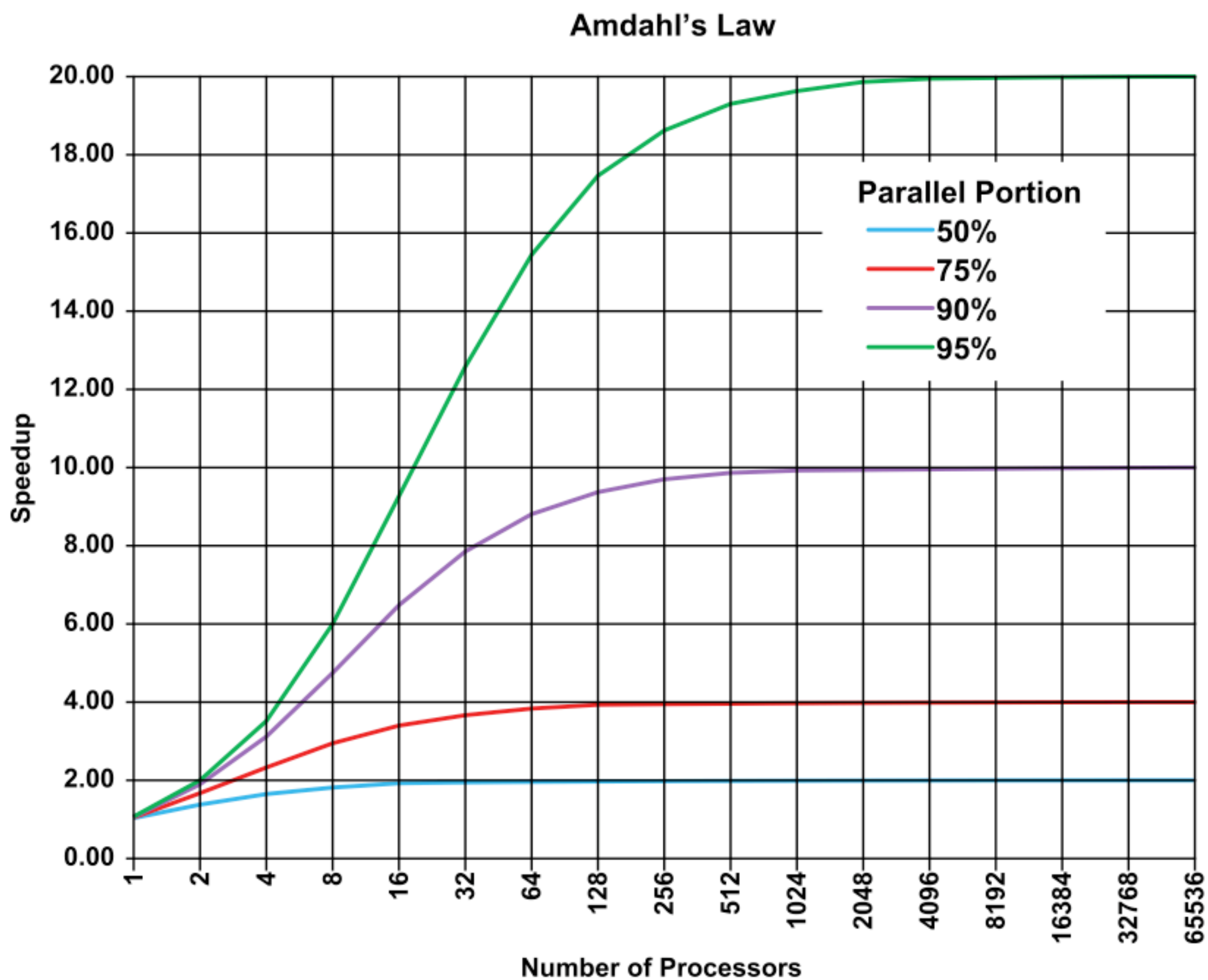
$$Sp=Ts/Tp=Ts/(\alpha\ Ts + (1-\alpha)Ts/p)=1/(\alpha + (1-\alpha)/p)$$

- If assume to have an infinite number of processors, then ?

$$limSp=1/\alpha$$

- For example, if α=10%, then ?

$$limSp=1/\alpha=10$$


Amdahl's Law
Parallel Portion
50%
75%
90%
95%
Speedup
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Number of Processors

Efficiency

- A measure of the fraction of total potential processing power that is actually used.

$$E_p = \frac{S_p}{p}$$

- A desired parallel system containing *p* processing elements can deliver a speedup equal to *p*

- Efficiency usually ranges from 0 to 1

Outline

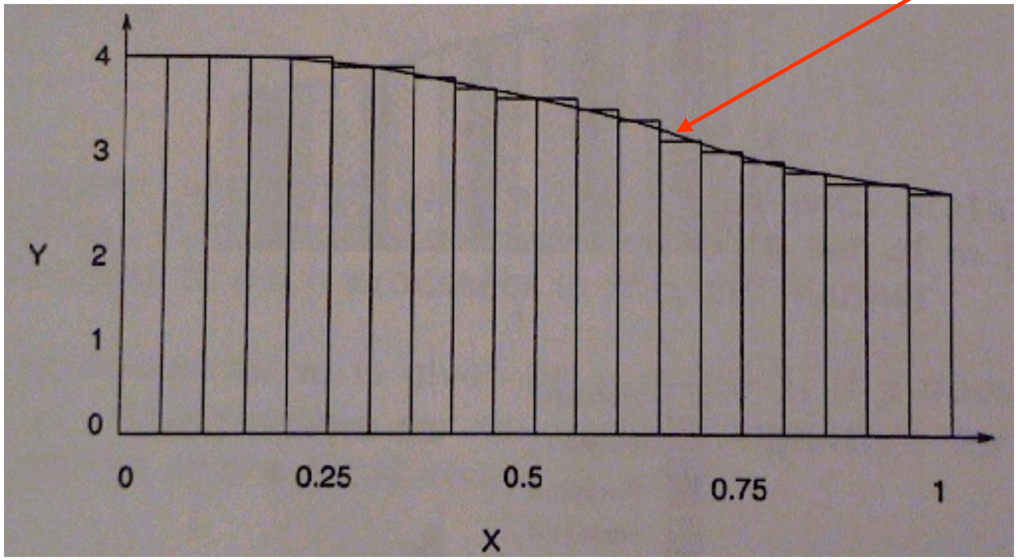
- Questions?
- Parallel performance modeling and evaluation
- Example of computing π
- Amdahl's law revisited and scaled speedup

Compute π: Problem

- Consider a parallel algorithm for computing the value of π=3.1415…through the following numerical integration

$$\pi = \int_0^1 \frac{4}{1+x^2}dx$$

$$\boxed{\frac{4}{1+x^2}}$$

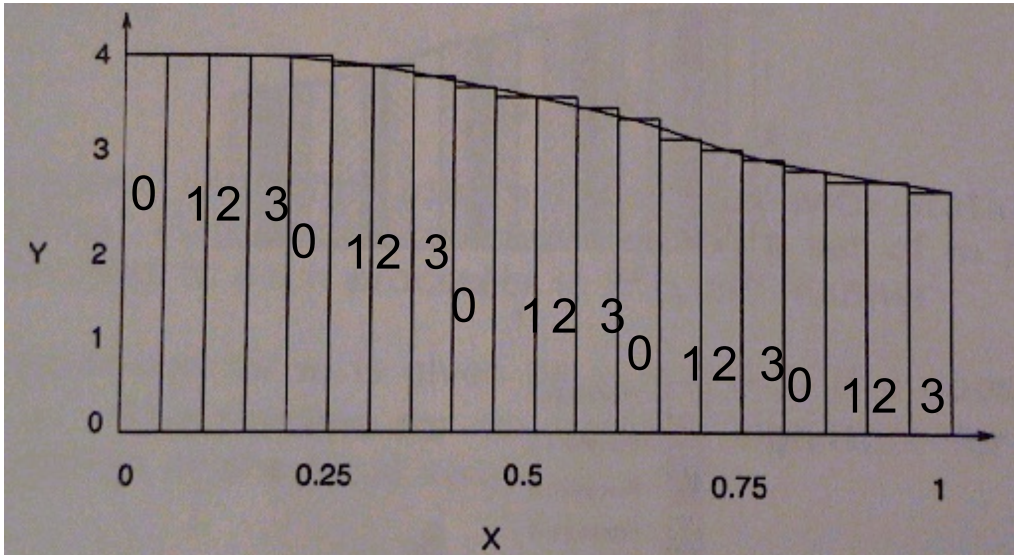


Compute π: Sequential Algorithm

```
computepi()
{
    h=1.0/n;
    sum =0.0;
    for (i=0;i<n;i++) {
        x=h*(i+0.5);
        sum=sum+4.0/(1+x*x);
    }
    pi=h*sum;
}
```

Compute π: Parallel Algorithm

- Each processor computes on a set of about *n/p* points, allocated to each processor in a cyclic manner
- Finally, we assume that the local values of π are accumulated among the p processors under synchronization



Compute π: Parallel Algorithm

```
computepi()
{
    id=my_proc_id();
    nprocs=number_of_procs():
    h=1.0/n;
    sum=0.0;
    for(i=id;i<n;i=i+nprocs) {
        x=h*(i+0.5);
        sum=sum+4.0/(1+x*x);
    }
    localpi=sum*h;
    use_tree_based_combining_for_critical_section();
        pi=pi+localpi;
    end_critical_section();
}
```

This example assumes a shared-address-space architecture

Compute π: Sequential Analysis

- Assume that the computation of π is performed over *n* points

- For *n* points, the number of operations executed in the sequential algorithm is:

Compute π: Parallel Analysis

- Assume the parallel algorithm uses *p* processors. Each processor computes on a set of *m* points which are allocated to each process in a cyclic manner
- The expression for *m* is given by $m \leq \frac{n}{p} + 1$ if *p* does not exactly divide *n*
- The runtime for the parallel algorithm for the parallel computation of the local values of π is:

Compute π: Parallel Analysis

- The accumulation of the local values of π using a tree-based combining can be performed in $\log_2(p)$ steps

- The total runtime for the parallel algorithm for the computation of π is:

- The speedup of the parallel algorithm is:

Compute π: Parallel Analysis

- The efficiency of the parallel algorithm is:

Outline

- Questions?
- Parallel performance modeling and evaluation
- Example of computing π
- Amdahl's law revisited and scaled speedup

Readings

- Reference book ITPC, Chapter 5
  - 5.1, 5.2, 5.3 (more examples)
- Foster, DBPP, Chapter 3, 3.1-3.3
  - http://www.mcs.anl.gov/~itf/dbpp/text/book.html
- Gene M. Amdahl, "*Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities*", 1967

Questions?

Questions/Suggestions/Comments are always welcome!

Write me: yong.chen@ttu.edu
Call me: 806-834-0284
See me: ENGCTR 315

If you write me an email for this class, please start the email subject with [CS4379] or [CS5379].