HY586: Distributed Computing

Lecturer: Panagiota Fatourou

Graduate Course
Department of Computer Science
University of Crete

Section 1
Introduction - Synchronization

Motivation

- The clock speeds of computer systems are no longer raising due to physic's limitations.
- In order to improve performance, the major chip manufacturers are turning to multi-core architectures, in which multiple processors (cores) communicate directly through shared hardware caches.
- Multiprocessors make computing more effective by exploiting parallelism.
  - Exploiting parallelism is currently one of the outstanding challenges of Computer Systems!

Definition of a Distributed System

- A collection of individual computing devices that can communicate with each other.

- This definition is very broad (VLSI chip, tightly coupled shared memory multiprocessor, local-area network, Internet)

Characteristics of Concurrent Algorithms

- Inter-process Communication Mechanism
- Timing Model
- Failure Model
- Studied Problems

☺ **Distributed Systems (DSs) are highly desirable!**

☹ **Putting together a properly functioning DS is notoriously difficult.** Main difficulties are introduced by three factors:

- Asynchrony
  - Several processes are executed concurrently at different speeds, probably starting their execution at a different point in time.
  - Process execution can be halted or delayed in an unpredictable way (interrupts, preemptions, cache misses, page faults, etc.).
- Limited Local Knowledge
- Failures

Difficulties Encountered in Achieving Parallelism

Task

- On the first day in your new job, your boss ask you to find all primes between 1 and 1010.
- A machine that supports 10 concurrent threads is provided.
- The machine is rented by the minute.
- The longer the program takes the more it costs!

1st Attempt

Give each thread an equal share of the input domain.

Is there any problem in this approach?

- Equal ranges of input does not necessarily produce equal amounts of work!
- Primes do not occur uniformly!
- It takes more time to check if a large number is a prime than a small number!

UNCLEAR

- Is the work divided equal?
- Which threads are allocated the most work?

Difficulties Encountered in Achieving Parallelism

2nd Attempt

- Assign each thread one integer at a time.

Is there any problem in this approach?

- A Shared Counter is required – How do we implement it?

  1st Effort

  shared long int count = 0;
  return count++;

Is this correct for a system with two processes p_A, p_B?

Code of p_A	Code of p_B
1. tmpA = count;	1. tmpB = count;
2. tmpA = tmpA +1;	2. tmpB = tmpB +1;
3. count = tmpA;	3. count = tmpB;

What Could Go Wrong?

Process p_A	Process p_B
tmpA = count;	
tmpA = tmpA +1;	
	tmpB = count;
	tmpB = tmpB +1;
count = tmpA;	
	count = tmpB;

Time Axis

Even Worse....

Process p_A	Process p_B
tmpA = count; /* tmpA == 0 */	
	tmpB = count; tmpB = tmpB +1; count = tmpB; /* count == 1 */
	tmpB = count; tmpB = tmpB +1; count = tmpB;
	tmpB = count; tmpB = tmpB +1; count = tmpB;
	tmpB = count; tmpB = tmpB +1; count = tmpB; /* count == 4 */
tmpA = tmpA +1; count = tmpA; /* count == 1 */	
	tmpB = count; /* tmpB == 1 */
tmpA = count; tmpA = tmpA +1; count = tmpA; /* count == 2 */	
tmpA = count; tmpA = tmpA +1; count = tmpA;	
tmpA = count; tmpA = tmpA +1; count = tmpA;	
tmpA = count; tmpA = tmpA +1; count = tmpA;	
/* count == 5 */	
	tmpB = tmpB +1; count = tmpB;
	/* count == 2 */

The Harsh Realities of Parallelization

Ideal World

- Upgrading from a uniprocessor to an n-way multiprocessor should provide about an n-fold increase in computational power.

Practice

- This has never happened!!!!!
  - This is due to the cost of inter-processor communication and synchronization.

Example

- Five friends decide to paint a five room house.

☹ What happens if one room is twice as big as each of the other rooms?

Amdahl's Law

Intuition

- The extent to which we can speed up any complex job is limited by how much of the job can be executed sequentially.
  - S: maximum speedup that can be achieved by n concurrent processors (ratio between sequential time and parallel time for executing a job)
  - p: fraction of job that can be executed in parallel
- Assume that it takes (normalized) time 1 for a single processor to complete the job.
- Time needed for parallel part = p/n
- Time needed for sequential part = 1 – p
- Overall Time = 1-p+p/n

Amdahl's Law

$$S = \frac{1}{1 - p + p/n}$$

Amdahl's Law – Painting Example (I)

Assumption

- Each small room can be painted in 1 time unit. The sequential execution of the painting task requires 6 time units.
- When 5 painters are working concurrently, 5/6 of the work can be performed in parallel.
- Thus, parallel execution time = 1-5/6+1/6 = 1/6+1/6 = 2/6 = 1/3
- Thus, S = 1/(1/3) = 3 ☹

Important Notice

☹ Although we have 5 workers, the speedup we obtain is only 3-fold!!!!

☹ It could be worse:

- 10 rooms, 10 painters, one room twice as big as each other room.
- Then, parallel execution time = 1 – 10/11 + 1/11 = 2/11.
- Speedup = 1/ (2/11) = 11/2 = 5.5!!!

- Only 5.5-fold speedup, almost half of what was expected ☹
- And this is so given that > 90% of the work can be parallelized and < 10% cannot be parallelized ☹

Amdahl's Law – Painting Example (II)

Solution

- As soon as a painter's work in a room is done, s/he helps others to paint the remaining room.

- Substantial communication and synchronization is needed!!!

  - **This is a hard task ☹**

Course Objectives

- Understanding the major tools and techniques that allow programmers to effectively program the parts of the code that require substantial communication and synchronization
- Studying the core ideas behind modern coordination paradigms and concurrent data structures
- Basic principles to the best practice engineering techniques of concurrent computing
- Identifying techniques to formally prove correctness of multiprocessor programs
- Presenting techniques for formally studying the progress properties of concurrent algorithms
- Analyzing the performance of multiprocessor algorithms – Introduce a variety of methodologies and approaches for reasoning about concurrent programs
- Identifying limitations and impossibility results which express where the effort should not be put in solving a task
- Studying state-of-the-art software technologies for expressing parallelism

Model Issues

Timing Models

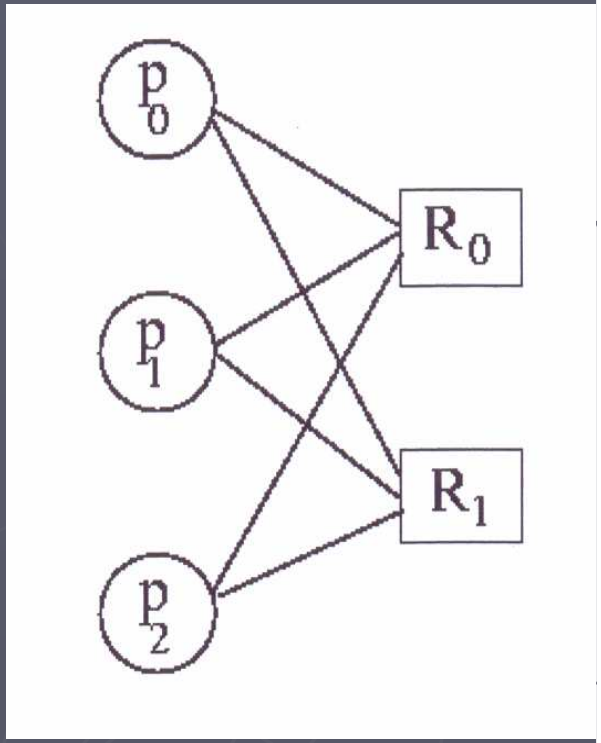
- **Asynchronous Model of Computation**
  - Processes are executed in arbitrary speeds and with arbitrary order.
- **Synchronous Model of Comoutation**
  - Some assumptions are made on the relative timing of events. For instance, there are (upper and lower) bounds on the execution time of each instruction.

Failure Models

- **Crash Failures**
  - **A process may stop its computation at any arbitrary point of its execution**
- **Byzantine Failures**
  - A failed process can behave in an arbitrary way (for instance, it may execute malicious code, i.e., code which is irrelevant to its original algorithm).

Modeling a Shared Memory System (I)

- There are n processes p_1, …, p_n, running concurrently in the system.
- Processes communicate by accessing shared registers.
- Each process is modeled as a state machine.



Registers

- Each register R has a type which specifies the following:
  - The values that can be taken on by R
  - The operations that can be executed on R
  - The value to be returned by each operation, and
  - The new value of the register resulting from each operation

Operations Supported by a Read/Write (RW) Register

- ✓ **Read(R):** returns the current value of R, leaving R unchanged
- ✓ **Write(R,v):** writes v into R and returns ack

Modeling a Shared Memory System (II)

Configurations

In a shared memory system with m registers, a configuration C is a vector $C = (q_1,\ldots, q_n, r_1, \ldots, r_m)$, where:

- q_i is the state of process p_i, $\forall$ i, $1 \leq i \leq n$, and
- r_j is the value of register R_j, $\forall$ j, $1 \leq j \leq m$
- In an initial configuration, all processors are in their initial states and all registers contain initial values.

- **A configuration describes the distributed system at some point in time.**

Events

- An event is a computational step by any process. At each computation step by some process pi, the following happen atomically:
  - p_i chooses a shared variable to access with a specific operation, based on p_i's current state;
  - the specified operation is performed on the shared variable, and
  - p_i's state changes according to pi's transition function, based on p_i's current state and the value returned by the shared memory operation performed.

Modeling a Shared Memory System (III)

- The behavior of a system over time is modeled as an execution.
- An execution fragment of an algorithm is a (finite or infinite) sequence of the following form:

$$C_l, \varphi_{l+1}, C_{l+1}, \varphi_{l+2}, C_{l+2}, \varphi_{l+3}, \cdots ,$$

where each C_k is a configuration and each φ_k is an event.

- The application of φ_k to C_{k-1} results in C_k, as follows:
  - Suppose $\varphi_k = i$ (i.e., φk is a computation step by process p_i) and p_i's state in C_k indicates that shared register R_j is to be accessed.
  - C_k is the result of changing C_{k-1} in accordance with p_i's computation step acting on p_i's state in C_{k-1} and the value of register R_j in C_{k-1}.
  - The only changes are to p_i's state and the value of R_j.

Modeling a Shared Memory System (IV)

- An execution is an execution fragment starting from an initial configuration C_0.
- The schedule of an execution is the sequence of steps φ_1, φ_2, φ_3, … taken in the execution.

Correctness & Progress Properties

- **An execution must satisfy a variety of properties**
- **Safety Properties**
  - A condition that must hold in every finite prefix of the sequence
  - It states that nothing bad has happened yet!
- **Liveness Propoerties**
  - **It must hold a certain number of times (probably an infinite number of times)**
  - **It states that eventually something good must happen!**
- An admissible execution satisfies all required safety and liveness properties.

Modeling a Shared Memory System (V)

Fairness

- In an infinite execution, each process should be given the chance to execute an infinite number of steps. Each time the process is given the chance, it may execute one more step or do nothing depending on its algorithm and its current state.

- In a finite execution, there should not be any processes that are interested in performing more steps.

Modeling a Shared Memory System – Complexity Measures

Space Complexity

- Amount of shared memory needed
  - Measured in number of shared registers used; size of this registers is also of interest.

Step (or Time) Complexity

- Usually, we measure the number of steps performed by each process to execute its algorithm. We are mainly interested on whether this number is finite, infinite, or bounded.
- Problem
  - Contention: the number of processors concurrently accessing the same variable.
- Meaningful and precise definitions of shared memory algorithms are the subject of current research!!!!

Number of Failures