CS390C: Principles of Concurrency and Parallelism

Suresh Jagannathan
suresh@cs.purdue.edu
http://www.cs.purdue.edu/homes/suresh

and

Ananth Grama
ayg@cs.purdue.edu
http://www.cs.purdue.edu/homes/ayg

http://www.cs.purdue.edu/homes/suresh/CS390C

CS390C: Principles of Concurrency and Parallelism
Course Layout

• Introduction to Concurrency and Parallelism
  – Motivation
  – Platforms
• Basic Concepts
  – Interaction Models for Concurrent Tasks
  – Elements of Concurrency
    • Threads, Co-routines, Events
  – Concurrency Control
    • Serializability, Atomicity
  – Performance Measures
Course Layout

- System Support for Concurrency and Parallelism
  - Scheduling Techniques
  - Synchronization Mechanisms (locks, transactions, wait-notify)
- Data Structures
  - Queues, Heaps, Trees
- Algorithms
  - Sorting, Graph Algorithms
- Abstractions
  - Dataflow, Selective Communication, Continuations
- Formal Methods and Analyses
Grading and Evaluation

- Three to four small programming projects
- One larger project, designed and implemented in consultation with the instructor(s)
- One final exam
Introduction to Concurrency and Parallelism

• What is Concurrency?

Traditionally, the expression of a task in the form of multiple, possibly interacting subtasks, that may potentially be executed at the same time constitutes concurrency.
Introduction to Concurrency and Parallelism

• What is Concurrency?

  - Concurrency is a programming concept.
  - It says nothing about how the subtasks are actually executed.
  - Concurrent tasks may be executed serially or in parallel.
Why Concurrency?

- Concurrency plays a critical role in serial as well as parallel/distributed computing environments.
Why Concurrency?

- In a serial environment, consider the following simple example of a server, serving requests from clients (e.g., a web server and web clients)

```
request 1
request 2
```

Non-concurrent serial server

```
t = 0
```
Let us process requests serially

Total completion time = 8 units, Average service time = \((6 + 8)/2 = 7\) units
Try a concurrent server now!

- t = 0
  - request 1
  - request 2

- t = 1
  - request 1
  - request 2

- t = 2
  - request 1
  - request 2
We reduced mean service time!

Total completion time = 8 units, Average service time = (4 + 8)/2 = 6 units
Why Concurrency?

- The lesson from the example is quite simple:
  - Not knowing anything about execution times, we can reduce average service time for requests by processing them concurrently!

- But what if I knew the service time for each request?
  - Would “shortest job first” not minimize average service time anyway?
  - Aha! But what about the poor guy standing at the back never getting any service (starvation/fairness)?
Why Concurrency?

- Notions of service time, starvation, and fairness motivate the use of concurrency in virtually all aspects of computing:
  - Operating systems are multitasking
  - Web/database services handle multiple concurrent requests
  - Browsers are concurrent
  - Virtually all user interfaces are concurrent
Why Concurrency?

• In a parallel context, the motivations for concurrency are more obvious:
  - Concurrency + parallel execution = performance
What is Parallelism?

- Traditionally, the **execution of concurrent tasks on platforms capable of executing more than one task at a time** is referred to as “parallelism”
- Parallelism integrates elements of execution -- and associated overheads
- For this reason, we typically examine the **correctness of concurrent programs and performance of parallel programs**.
Why Parallelism?

• We can broadly view the resources of a computer to include the processor, the data-path, the memory subsystem, the disk, and the network.

• Contrary to popular belief, each of these resources represents a major bottleneck.

• Parallelism alleviates all of these bottlenecks.
Why Parallelism?

- Starting from the least obvious:
  - I/O (disks) represent major bottlenecks in terms of their bandwidth and latency
  - Parallelism enables us to extract data from multiple disks at the same time, effectively scaling the throughput of the I/O subsystem
  - An excellent example is the large server farms (several thousand computers) that ISPs maintain for serving content (html, movies, mail).
  - Try logging into mail.yahoo.com and see where your mail is hosted at (mine is at us.f301.mail.yahoo.com)
Why Parallelism?

- Most programs are memory bound – i.e., they operate at a small fraction of peak CPU performance (10 – 20%)
- They are, for the most part, waiting for data to come from the memory.
- Parallelism provides multiple datapath's to memory – effectively scaling memory throughput as well!
Why Parallelism?

- The process itself is the most obvious bottleneck.
- Moore's law states that the component count on a die doubles every 18 months.
- Contrary to popular belief, Moore's law says nothing about processor speed.
- What does one do with all of the available “components” on the die?
Parallelism in Processors

- Processors increasingly pack multiple cores into a single die.

Why?
Parallelism in Processors

- The primary motivation for multicore processors, contrary to belief is not speed, it is power.
- Power consumption scales quadratically in supply voltage.
- Reduce voltage, simplify cores, and have more of them – this is the philosophy of multicore processors.
Why Parallel?

- Sometimes, we just do not have a choice – the data associated with the computations is distributed, and it is not feasible to collect it all.
  - What are common buying patterns at Walmart across the country?
- In such scenarios, we must perform computations in a distributed environment.