Finishing last time, data parallelism, and parallel patterns

1/8/2016
Going forward

• Reading
  – Wednesday: Sections 2.3-2.4
  – Friday: Sections 3.5-3.6
  – following Monday: Sections 3.7-3.8

• Homework out today
Which of the following is an incorrect statement contrasting shared memory with message passing?

A. Shared memory makes it easier to identify where processors interact
B. Shared memory doesn’t scale as well as large systems
C. Message passing removes the need for complicated cache coherence protocols
D. Message passing requires matching send and recv calls for processors to interact
E. Message passing can be implemented in a library even on machines with shared memory
Which of the following is an incorrect statement contrasting shared memory with message passing?

A. Shared memory makes it easier to identify where processors interact
B. Shared memory doesn’t scale as well as well to large systems
C. Message passing removes the need for complicated cache coherence protocols
D. Message passing requires matching send and recv calls for processors to interact
E. Message passing can be implemented in a library even on machines with shared memory
Where we ended: Levels of abstraction

- Computational model: Model of the machine executing the program
- Programming model: Primitives provided for expressing a program
Decomposition of computation

• How the computation is split into parts
Decomposition of computation

• How the computation is split into parts

• Should this be:
  – Static (at program start) or Dynamic (on-going)?
Decomposition of computation

• How the computation is split into parts

• Should this be:
  – Static (at program start) or Dynamic (on-going)?

  – Fine-grained or Coarse-grained?
Tasks vs Threads

• Task: A piece of parallel work

• Thread: Work-performing entity associated with a computational unit
Tasks vs Threads

• Task: A piece of parallel work

• Thread: Work-performing entity associated with a computational unit
  – Unless language provides logical threads that are then assigned to computational units
  – Called a process in distributed setting
Running tasks

• Scheduling: Assignment of tasks to processes or threads
  – Goal is load balancing

• Mapping: Assignment of processes or threads to physical processes or cores
  – Load balancing again (if sharing compute units) plus minimizing communication costs
Running tasks

• Scheduling: Assignment of tasks to processes or threads *(Me: choosing *when* a program runs)*
  – Goal is load balancing

• Mapping: Assignment of processes or threads to physical processes or cores *(Me: Where tasks run)*
  – Load balancing again (if sharing compute units) plus minimizing communication costs
First HW
High-level construct: Work queues

Java’s Executor Framework

• Framework handles many details
  – creation of threads
  – task scheduling
  – assignment of tasks to threads

• Part of standard Java library
  – portable, well-documented, integrated with tools

• Unfortunately, still has some messy details
Important SimpleExecutor operations

• Creating a SimpleExecutor:
  ```java
  SimpleExecutor e = new SimpleExecutor();
  ```

• Defining tasks:
  ```java
  public class MyTask extends SimpleTask {
    public void run() { ... }
  }
  ```

• Submitting a task `t` to SimpleExecutor `e`:
  ```java
  e.submit(t);
  ```

• Waiting for a task `t` to finish:
  ```java
  t.finish()
  ```

• Closing up a SimpleExecutor `e`:
  ```java
  e.terminate();
  ```
Our wrapper classes

- **SimpleExecutor**
  - `void submit(SimpleTask)`
  - `SimpleFuture<T> submit(Callable<T>)`
  - `void terminate()`
  - `boolean isShutdown()`
  - `boolean isTerminated()`
  - `int getCores()`

- **SimpleTask**
  - `void run()` //override to use class
  - `void finish()` //block until done
  - `boolean isDone()`

- **SimpleFuture<T>**
  - `T get()`
  - `boolean isDone()`
Heat diffusion

- 2D array of cells, each with temperature
- Updates in time steps
  \[ T_{\text{new}} = \text{weighted ave. of } T_{\text{old}} \text{ and neighboring temps} \]
Screenshot
Types of parallelism

• Data parallelism: Parallelism achieved by running on different parts of the data. Amount available grows with the size of the data.

• Task parallelism: Parallelism achieved by running multiple tasks (generally doing different things)
What kind of parallelism is provided by quicksort?

A. Task parallelism
B. Data parallelism
C. It has elements of both task and data parallelism
D. Neither task nor data parallelism
E. There’s more than one kind of parallelism?!
What kind of parallelism is provided by quicksort?

A. Task parallelism
B. Data parallelism
C. It has elements of both task and data parallelism
D. Neither task nor data parallelism
E. There’s more than one kind of parallelism?!
What kind of parallelism is provided by a chess program that parallelizes a search tree?

A. Task parallelism

B. Data parallelism

C. It has elements of both task and data parallelism

D. Neither task nor data parallelism

E. I still don’t believe there is more than one kind
What kind of parallelism is provided by a chess program that parallelizes a search tree?

A. Task parallelism
B. Data parallelism
C. It has elements of both task and data parallelism
D. Neither task nor data parallelism
E. I still don’t believe there is more than one kind