Slide 1

Administrivia

- Reminder: Homework 2 due today.

Slide 2

Supporting Structures Program Structure Patterns — Recap

- Basic ways parallel programs can be structured:
  - *SPMD* (Single Program, Multiple Data) — “like an MPI program” (but could use same strategy in OpenMP, e.g.).
  - *Loop Parallelism* — “like an OpenMP program”.
  - *Master/Worker* — as the name suggests. Look briefly at MPI example.
  - *Fork/Join* — if you need to be able to create / wait for UEs in any arbitrary way.
- How to choose one? usually based on combination of programming environment (MPI, OpenMP, etc.) and overall strategy (*Algorithm Structure* pattern).

Supporting Structures Data Structure Patterns

Slide 3

- Probably not a complete list, but some examples of frequently-used ways of sharing data:
  - *Shared Data* (generic advice for dealing with data dependencies).
  - *Shared Queue* (what the name suggests — mostly included as example of applying *Shared Data*).
  - *Distributed Array* (what the name suggests).
- Programming environment / library may provide support (e.g., Java has library class(es) for shared queues).

Shared Queue

Slide 4

- Many applications — especially ones using a master/worker approach — need a shared queue. Programming environment might provide one, or might not. Nice example of dealing with a shared data structure anyway.
- Java code in figures 5.37 (p. 185) through 5.40 (p. 189) presents a step-by-step approach to developing implementation.

Shared Queue, Continued

- Simplest approach to managing a shared data structure where concurrent modifications might cause trouble — one-at-a-time execution. Shown in figures 5.37 (nonblocking) and 5.38 (block-on-empty). Only tricky bits are use of dummy first node and details of `take`. Reasons to become clearer later.

  Usually a good idea to try simplest approach first, and only try more complex ones if better performance is needed. ("Premature optimization is the root of all evil." Attributed to D. E. Knuth; may actually be C. A. R. Hoare.)
- Here, next thing to try is concurrent calls to `put` and `take`. Not too hard for nonblocking queue — figure 5.39. Tougher for block-on-empty queue — figure 5.40. In both cases, must be very careful.
- If still too slow, or a bottleneck for large numbers of UE, explore distributed queue.

Distributed Array

- Key data structures for many scientific-computing applications are large arrays, often 2D or 3D.
- If we have lots and lots of memory shared among UEs, and time to access an element doesn't depend on UE, all is well. Usually not the case. though — obviously true for distributed-memory systems, somewhat true for NUMA systems also.
- So — typical approach is to partition array into blocks and distribute them among UEs. Idea is to do this to get:
  - Good load balance.
  - Minimum communication.
  - "Clarity of abstraction". Key idea — global indices versus local indices.

  Pictures are easy to draw and understand; code can get messy.

Slide 7

Distributed Array, Continued

- Commonly used approaches (“distributions”):
  - 1D block.
  - 2D block.
  - Block-cyclic.
- For some problems (such as heat distribution problem), makes sense to extend each “local section” with “ghost boundary” containing values needed for update.
- Look at some versions of code for the heat-distribution problem. (MPI code in book as Figures 4.14 and 4.15 (pp. 90–91).)

Slide 8

Minute Essay

- What did you find interesting about Homework 2? What did you find difficult?