Slide 1

Administrivia

- Reminder: Homework 2 due today. Remember to turn in both revised code and graphs/plots.

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).

Slide 3

Supporting Structures Data Structure Patterns

- 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).

Slide 4

Shared Queue

- 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.

Slide 5

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.

Slide 6

Minute Essay

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