Slide 1

Administrivia

- (None.)

Slide 2

Recap — *Supporting Structures* Patterns

- Program structure patterns (earlier this week).
- Patterns for commonly-used data structures — *Shared Queue* last time. One more, a fairly important one . . .

Slide 3

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 4

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 5

Minute Essay

- The simple strategy for parallelizing the heat diffusion program with OpenMP involves a lot of thread creation (twice per time step). Is there a way to do better? (Does the strategy you'd use for MPI provide hints?)

Slide 6

Minute Essay Answer

- There's certainly a way that might do better: You could essentially duplicate the MPI strategy in OpenMP – make the whole program an OpenMP "parallel section", with each thread doing the time step loop, with barriers at the end of each phase of the calculation. We did something like this with the numerical integration example — "SPMD" versions in OpenMP.