Administrivia

- Sample solutions for Homeworks 1 and 2 on Web.

Slide 1

Supporting Structures Data Structure Patterns — Review

- 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 2

Shared Queue

Slide 3

- (Review slides from last time briefly.)

Distributed Array

Slide 4

- 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.

Distributed Array, Continued

- Commonly used approaches ("distributions"):

    - 1D block.

    - 2D block.

    - Block-cyclic.

Slide 5

- For some problems (such as heat-diffusion problem), makes sense to extend each "local section" with "ghost boundary" containing values needed for update.

Example Application — Heat-Diffusion Problem

- We've talked about this problem in general terms. Now look at code . . .

- OpenMP version is fairly straightforward — parallelize two inner loops, only somewhat-tricky part is the reduction operation to compute `maxdiff`.

Slide 6

- MPI version is less straightforward — applying *Distributed Array* pattern to the two big arrays is straightforward in principle but in practice full of messy details.

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 7

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.

Slide 8