Slide 1

Administrivia

- Reminder: Homework 3 code due today. (Accepted through Tuesday night without late penalty.) Homework 4 to be on Web soon.

Slide 2

Homework 3

- Shared-memory parallelization with OpenMP should be straightforward. (Agreed?)
- Distributed-memory parallelization with MPI is less so, especially if you split up the board among processes (as described in *Distributed Array*). I say you will learn more by doing it that way.

  Questions include how you initialize the board (from a file, using random sequence), how you print it, and what kinds of experiments will show whether the parallelization speeds up the calculations.

Slide 3

Example Application: Mandelbrot Set

- (Review code, minute essay, etc. Questions?)

Slide 4

Example Application: Mergesort

- Recall(?) from earlier classes — mergesort as a (recursive) divide-and-conquer algorithm.
- Sequential code is relatively straightforward(?).

Slide 5

Parallelization — *Finding Concurrency*

- Task-based decomposition seems more logical, with each recursive call to sort function as a task.
- How do the tasks depend on each other? every time we split the problem, we create two tasks that are independent of each other — but there is a dependency between these tasks and the task that creates them.

Slide 6

Parallelization — *Algorithm Structure*

- Tasks here form a recursive tree-like hierarchy, so *Divide and Conquer* should seems like a good choice.
- Key design decisions are how to assign tasks to UEs, whether to really consider each call to sort function as a separate task (probably not!) or merge them.

Slide 7

Parallelization — *Supporting Structures*

- *Fork/Join* structure probably makes sense.

Slide 8

Parallelization — Code

- (Look at code.)

Slide 9

Minute Essay

- Why does the parallel mergesort not “scale” well (performance doesn’t continue to improve as the number of threads increases), and indeed in general doesn’t seem to performn very well?

Slide 10

Minute Essay Answer

- A nontrivial part of the computation — the merge operation — isn’t being parallelized, so the full number of threads isn’t being used during all of the program’s execution.