Slide 1

Administrivia

- Reminder: Homework 3 due Thursday. Sample solutions for previous homeworks coming soon.
- (Review minute essay from last time.)

Slide 2

Example Application: Mandelbrot Set

- For each point $c = a + bi$ in the complex plane, look at the sequence $z_0, z_1, z_2, \ldots$, where

$$\begin{aligned} z_0 &= 0 \\ z_{k+1} &= {z_k}^2 + c \end{aligned}$$

- For some points, this sequence is "quasi-stable" ($|z_k|$ bounded); for others, it's not.
- We can get interesting pictures by discretizing and then computing, for each point, how long it takes this sequence to "diverge".

Slide 3

Parallelization — Understanding the Problem

- Code is a loop over points in a 2D space, where at each point we calculate until divergence / maximum iterations and then plot the result (to something implicitly or explicitly shared).
- Consider parallelizing for a distributed-memory environment. (Along the way, also consider what would be different with shared memory.)

Slide 4

Parallelization — *Finding Concurrency*

- Task-based decomposition seems more logical. Consider calculations for one point as a task.
- How do the tasks depend on each other? they don't really, unless "plotting" a result means doing something with a shared resource.

Slide 5

Parallelization — *Algorithm Structure*

- Many mostly-independent tasks, forming a flat set rather than a hierarchy, so *Task Parallelism* seems like a good choice.
- Key design decisions are how to assign tasks to UEs, how to manage "plotting".
- Probably makes sense to group tasks by rows rather than individual points. We could try a simple static distribution, but might have to do something more complex if that doesn't give good load balance.
- Managing plotting? in a distributed-memory environment, might make sense to just assign that job to a process that does nothing else.

Slide 6

Parallelization — *Supporting Structures*

- *SPMD* structure probably makes sense, but with elements of *Master/Worker* (a master process to manage the computation and the displays, and workers to do the calculations).
- (For shared memory, *Loop Parallelism* will probably make sense, also possibly with elements of *Master/Worker*.)

Slide 7

Parallelization — Code

- (Look at code, multiple versions.)

Slide 8

Minute Essay

- I have two versions of the OpenMP version of the Mandelbrot program, both of which use an OpenMP `critical` directive to be sure only one thread at a time changes the display. In one version, the critical section displays one point; in the other, it displays a whole row. Which one do you think will be faster, and why?
- Using `schedule(static)` in the OpenMP program seems to give poor load balance? What could you use instead, to improve load balance?

Minute Essay Answer

- Probably the one that has fewer and larger critical sections — less overhead.
- `schedule(static,N)` where N is some "chunk size", to distribute statically, or `schedule(dynamic)` (possibly also with a "chunk size"). See the OpenMP documentation for details.