Slide 1

Administrivia

- (None? Reminder about Homework 4?)
- I hope to grade Homework 3 soon.

Slide 2

Example Applications: Numerical Integration, Heat Diffusion

- We've looked at code for both. Anything we should look at again? maybe the code for the MPI version of the heat diffusion problem?

Slide 3

Example Application: Generic Master/Worker Program (Review)

- As an illustration of the *Master/Worker* program-structure pattern, try writing a sort of mock-up of such a program, in which tasks are represented by "sleeps" of varying lengths.
- Sequential code just generates some number of fake tasks with varying times generated using `rand()`. (Look at code.)

Slide 4

Generic Master/Worker Program — OpenMP

- Parallelizing sequential code with OpenMP is fairly straightforward:
- We don't need an explicit master thread because all it would do is assign tasks to threads, and we can get that with `omp parallel for`. Here we might want to try both static and dynamic scheduling.
- (Look at code, and notice additions to also show how tasks were distributed among threads. Also notice use of `#omp critical` to avoid potential race conditions with calls to `rand()`. Not a good strategy in an application where those calls were a big contributor to overall program runtime, but here they're probably not.)

Slide 5

Generic Master/Worker Program — MPI

- Parallelizing sequential code with MPI is less straightforward:
- For static scheduling, we don't need an explicit master; we can easily have each process pick out "its" tasks.
- For dynamic scheduling, it does seem like we need an explicit master, so have one process serve in that role, with a defined protocol for master/worker interaction:
  - Each worker process repeated requests a task from the master, receives one, and executes it, continuing until it gets a task meaning "no more".
  - The master process repeatedly receives requests for a task from workers, responds to it, and records results, until all tasks are complete. It then sends each worker a "no more" task.

Slide 6

Generic Master/Worker Program — MPI, Continued

- (Look at code, and notice additions to also show how tasks were distributed among processes. Also notice that the static-distribution version just generates the whole sequence of tasks in each process and then only executes some of them. Not a good strategy in an application where generating the tasks was a big contributor to overall program runtime, but here it's probably not.)

Slide 7

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 8

Parallelization — Understanding the Problem

- Code is nested loops 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. . . (To be continued.)

Slide 9

Minute Essay

- How is Homework 4 coming?