Slide 1

Administrivia

- (None?)

Slide 2

Testing/Evaluating Parallel Applications

- For most programs we write in this class, want to know two things:
- Does it give appropriate results?
- Does it seem to scale well with number of UEs (or, for OpenCL, does it perform better then sequential code for the same problem)?
- In showing examples I haven't done this carefully. So, some observations:

Slide 3

Evaluating Parallel Applications — Correctness

- Ideally, want a parallel application to give same results independent of details of execution (e.g., number of UEs), and if it's a parallelization of sequential code, the same results as that.
- Have we really checked this for the numerical integration example? no . . .

Slide 4

Numerical Integration Example — Correctness

- In fact results for varying numbers of UEs *not* the same! Close, but not exactly!
- Why? (Pause the video and record your thoughts, to include in the minute essay.)

Slide 5

Numerical Integration Example — Correctness

- Differences are because floating-point addition is not associative.
- So our definition of "correct" will allow for minor differences. (Major differences are another matter and usually indicate something wrong.)

Slide 6

Evaluating Parallel Applications — Performance

- Ideally, want a parallel application to exhibit linear speedup (or better?) as the number of UEs increases, up to a point anyway.
- Measuring performance, though . . . Complicated subject, but I say fairest comparison is wall-clock time. (Even that may be iffy for JVM-based languages, where results can be influenced by runtime compiling, garbage collection, etc.)
- To get reasonably meaningful numbers . . .

Slide 7

Evaluating Performance — Tips

- First thing to do is choose inputs (e.g., number of steps in our example). *Very* typical beginner mistake is to choose problem so small that runtimes are tiny and comparisons probably don't mean much. (Sadly, sometimes for toy problems this means picking inputs based not on what gives good answers but on what allows measuring performance meaningfully.)
- Next thing to do is choose platform, aiming for something that allows the option of many truly concurrent UEs. (So for example, a dual-core machine isn't very useful for evaluating OpenMP.)
- Finally, best to re-run experiments more than once in case they vary from execution to execution.

Slide 8

Numerical Integration Example — The Big Picture

- I showed parallelizations of this code using several programming environments without talking much about where the overall strategy came from. Sometimes it may be obvious, but . . .
- Look again explicitly at what we did to parallelize with OpenMP, as an example of how to think about designing a parallel application.

  (We'll do some of this more carefully as we work our way through the book, but a preview for now?)

Slide 9

Numerical Integration Example, Continued

- Starting point is an understanding of the problem/computation. Pretty simple here, no?
- First step in developing a parallel version: Break computation down into smallest "tasks" that could maybe execute concurrently.
- For this example, iterations of the main computation loop.

Slide 10

Numerical Integration Example, Continued

- Next step: Consider how these tasks interact.

  Are there logic/control dependencies? data dependencies? shared data?
- Here, tasks all independent except that they share some variables — so if we can manage the shared data, we can execute them in any order we want — including concurrently.

  Hooray — we just found some "exploitable concurrency"!

Slide 11

Numerical Integration Example, Continued

- Next step: Develop strategy for taking advantage of this potential for concurrent execution.
- For that, can help to try to use one of a few very common strategies (which our book captures as patterns). This example fits the simplest one (*Task Parallelism*).

Slide 12

Numerical Integration Example, Continued

- Key elements of (*Task Parallelism*) strategy, as they apply here:
  - Split "tasks" (loop iterations) among UEs as evenly as possible, since they're all the same size.
  - Make sure every UE has its own copy of work variable `x`.
  - Manage the shared variable `sum` as for "reduction operations" — give each UE its own local variable, combine at the end.
- Final step is to turn the strategy into code. (Does this match what we did?)

Slide 13

Minute Essay

- What was your explanation for why the numerical integration example gives slightly different answers for different numbers of UEs?
- Questions?