Slide 1

## Administrivia

- Homework 1 to be on the Web tomorrow, due next Wednesday. I will send mail.

Slide 2

## Numerical Integration, Revisited

- Recall numerical integration example, sequential version.
- Before talking about how to parallelize using MPI, let's try to be explicit about what we did to parallelize with OpenMP, as an example of how to think about designing a parallel application . . .

Slide 3

## Numerical Integration, Continued

- Starting point is an understanding of the problem/computation. Pretty simple here, no?
- First step in developing a parallel version is to break the computation down into the smallest “tasks” that can execute concurrently. Here, that’s the iterations of the main computation loop.
- Next step is to consider how these tasks interact — are there logic/control dependencies? data dependencies? shared data? Here, the tasks are 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. We just found some “exploitable concurrency”.

Slide 4

## Numerical Integration, Continued

- Next step is to develop a strategy for taking advantage of this potential for concurrent execution.
- For that, it 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*).

## Numerical Integration, 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 — which we already did in OpenMP.

**Slide 5**

## Numerical Integration in MPI

- Now figure out how to apply the overall strategy using MPI. Key difference is lack of shared memory — means we don’t have problems with threads stepping on shared work variables, but we have to work harder to combine partial results.
- Sample program `num-int-par.c`.

**Slide 6**

Slide 7

## Minute Essay

- If you add the following lines to sample program `send-recv.c`, right after the call to `printf()` for process 0

  ```
  buff[0] = 30;
  buff[1] = 40;
  ```

  what does process 1 print?

Slide 8

## Minute Essay Answer

- The same thing as before — the old data has already been sent to process 1 (or at least copied to a system buffer somewhere), so the change doesn't affect what happens in process 1.