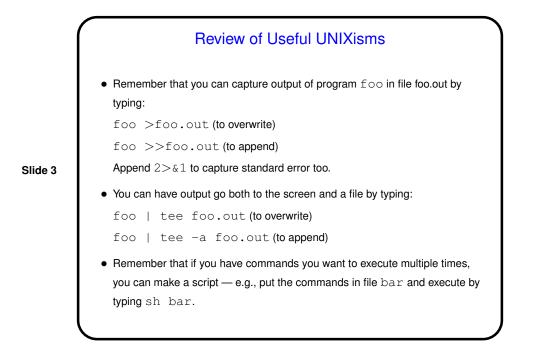
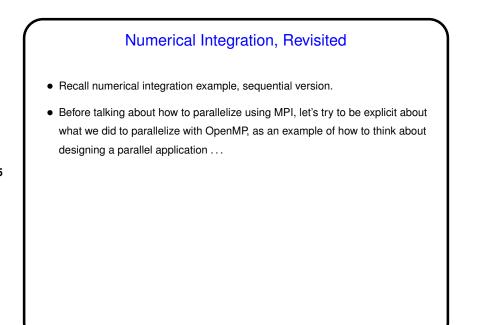


Sidebar: Timing Parallel Programs
"How long did it take?" often of interest. Can use system tools (e.g., time command) to check total elapsed time. Or can time "interesting" parts of program:
MPI_Wtime returns elapsed time; call twice and subtract to find out how long something takes (time_msg.c on "sample programs" page).
(In OpenMP, use omp_get_wtime.)
I like to print time plus enough identifying info (number of processes/threads, problem size) that you can "collect performance data" by capturing program output.
How meaningful output is depends — e.g., on whether the system is otherwise idle. Probably best to repeat observations a few times, and do some sort of averaging.

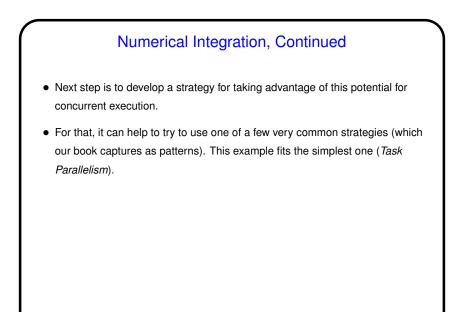


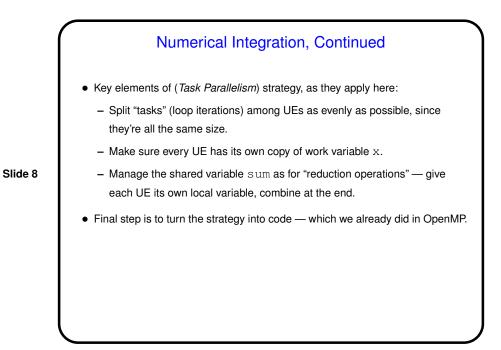
Review of Useful UNIXisms, Continued — Text Editors
If you're using vim to edit code and finding it painful — try spending half an hour with the tutorial (command vimtutor), and/or try the graphical version gvim.
If you haven't imprinted on a UNIX editor, also spend half an hour with the tutorial for emacs (emacs, then follow instructions on screen for accessing tutorial).
If neither one appeals to you, there are others (gedit and pico come to mind).



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".





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.

Slide 9

• Sample program num-int-par.c.

Parallel Programming in Java

- Java supports multithreaded (shared-memory parallel) programming as part of the language — synchronized keyword, wait and notify methods of Object class, Thread class. Programs that use the GUI classes (AWT or Swing) multithreaded under the hood. Justification probably has more to do with hiding latency than HPC, but still useful, and recent versions (5.0 and beyond) includes much useful library stuff.
- Java also provides support for forms of distributed-memory programming, through library classes for networking, I/O (java.nio), and Remote Method Invocation (RMI).

