Slide 1

Administrivia

- Homework 2 on Web soon. To be due a week from Wednesday.

Slide 2

Minute Essay From Last Lecture

- Several sensible suggestions. Discuss.

Slide 3

Homework 2 — "Homework 1 Revisited"

- Write your own "thread-safe" RNG — one that can be called from multiple threads concurrently without ill effects. (Think about how to do that in Java versus C.)

  (References to some reasonable choices to be included in writeup on Web.)
- Modify your programs to use your RNG. Do something you think is reasonable so that each UE starts with a different seed.
- Measure your programs' accuracy and performance and plot results.
- Make a second version of your code in which all UEs start from the same seed but each works on a different part of the problem space (domain decomposition). Measure its accuracy and performance.

Slide 4

Shell Scripts — Basics

- "Shell script" is just a (text) file containing commands you could type to a shell. It should be fairly easy to "mass produce" multiple executions of the same command with different parameters using cut-and-paste in a text editor. (Or you could use the shell's constructs for looping, parameters, etc.)
- Execute commands in `scriptname` two ways:
  - `sh scriptname`
  - Make the file executable (`chmod u+x scriptname`) and execute directly (`scriptname` or `./scriptname`).

Slide 5

Capturing Output — Basics

- To capture output of a command in a file:
  - `cmd >outfile` to overwrite `outfile`.
  - `cmd >>outfile` to append.
- To capture output in a file but also view it as it's generated:
  - `cmd | tee outfile` to overwrite `outfile`. (That `|` is the "pipe symbol", a vertical bar.)
  - `cmd | tee -a outfile` to append.
- Once you have the output, you can edit as needed . . .

Slide 6

Text Editors — Some Tips

- (Maybe should be subtitled "What To Do If You Hate `vi`"?)
- Learning a little more about your text editor of choice can save you time.
- To learn more about `vi` (really `vim`), try tutorial (`vimtutor`). Or try graphical version `gvim`.
- Many other choices if you don't like `vi`. The other major player is `emacs` (control-H brings up tutorial, `xemacs` is the graphical version). Also `gedit`, `pico`, many others.

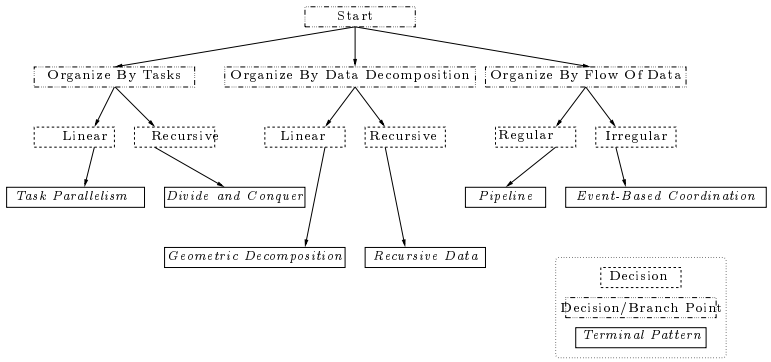
Slide 7

Algorithm Structure Design Space

- Historical note: These are the patterns with the longest history. We started out trying to identify commonly-used overall structures for parallel programs (these patterns), and then at some point added the other "design spaces".
- After much thought, writing, revision, and arguing, we came up with . . .

Slide 8

Algorithm Structure Decision Tree (Fig. 4.2)



Slide 9

Task Parallelism

- Problem statement:

  When the problem is best decomposed into a collection of tasks that can execute concurrently, how can this concurrency be exploited efficiently?

- Key ideas in solution — managing tasks (getting them all scheduled), detecting termination, managing any data dependencies.

- Many, many examples, including:
  - Molecular dynamics example (next slide).
  - Mandelbrot set computation.
  - Branch-and-bound computations: Maintain list of "solution spaces". At each step, pick item from list, examine it, and either declare it a solution, discard it, or divide it into smaller spaces and put them back on list. Tasks consist of processing items from list.

Slide 10

Molecular Dynamics and *Task Parallelism*

- How to define tasks so we get "enough but not too many"?

  One task per atom pair is too many; one task per atom is probably right.

- How to manage data dependencies (if any)?

  Dependency involving `forces` array — potentially any UE can write to any element, if we exploit symmetry resulting from Newton's third law. But computation is accumulation/reduction, so just give each UE a local copy and combine all copies at end.

- How to assign tasks to UEs? statically (at compile time) or dynamically (at runtime)?

  Work per task can vary, since how many atoms are "close" varies. Decide at next level.

Slide 11

Geometric Decomposition

- Problem statement:

  How can an algorithm be organized around a data structure that has been decomposed into concurrently updatable "chunks"?

- Key ideas in solution — distributing data, arranging for needed communication.

- Probably second most common pattern. Examples include:
  - Heat-diffusion problem previously discussed (next slide).
  - Matrix multiplication using blocks.

Slide 12

Heat Diffusion and *Geometric Decomposition*

- How to distribute data?

  One chunk per UE will probably work well. (Note that for other problems it might not.) Might be nice to include in data structure a place to store values from neighboring chunks. More in *Distributed Array*, next chapter.

- How to synchronize/communicate?

  With shared memory, just need barrier synchronization.

  With distributed memory, need to exchange values with neighbor UEs, also perform reduction.

Slide 13

Minute Essay

- Does the overall strategy for the two examples make sense? Do you think you could (almost?) turn them into code?