Slide 1

Administrivia

- Homework 2 on Web.
- Notes from last time updated with a few more Linux tips. With regard to getting output, notice that you could also just have your program write (some of its) output directly to a file or files.

Slide 2

Recap — *Algorithm Structure* Patterns

- If decomposition/analysis reveals organization in terms of tasks — *Task Parallelism* (probably most common strategy) or *Divide and Conquer*.
- If decomposition/analysis reveals organization in terms of data — *Geometric Decomposition* (second most common strategy) or *Recursive Data*.
- If organization is in terms of flow of data — (*Pipeline* and *Event-Based Coordination*).
- Last time we talked briefly about *Task Parallelism* and *Geometric Decomposition*. Other four this time.

Slide 3

Divide and Conquer

- Problem statement:

  Suppose the problem is formulated using the sequential divide and conquer strategy. How can the potential concurrency be exploited?

- Key idea in solution — create new task(s) every time we split (sub)problem, recombine when we merge.
- Examples include mergesort and some non-naive algorithms for N-body problem.
- Straightforward if you already have a sequential divide-and-conquer solution, but scalability is somewhat limited.

Slide 4

Recursive Data

- Problem statement:

  Suppose the problem involves an operation on a recursive data structure (such as a list, tree, or graph) that appears to require sequential processing. How can operations on these data structures be performed in parallel?

- Key idea in solution — "out of the box" thinking to expose concurrency.
- Probably least-used structure currently (because it doesn't map well to current architectures); included for completeness and because examples are interesting — e.g. "roots in forest" example.

Slide 5

Pipeline

- Problem statement:

  Suppose that the overall computation involves performing a calculation on many sets of data, where the calculation can be viewed in terms of data flowing through a sequence of stages. How can the potential concurrency be exploited?

- Key idea in solution — set up “assembly line” (pipeline).
- Canonical example is signal/image processing application, where you have a sequence of incoming images and want to apply same sequence of transformations to each one.

Slide 6

Event-Based Coordination

- Problem statement:

  Suppose the application can be decomposed into groups of semi-independent tasks interacting in an irregular fashion. The interaction is determined by the flow of data between them which implies ordering constraints between the tasks. How can these tasks and their interaction be implemented so they can execute concurrently?

- Key idea in solution — structure computation in terms of semi-independent entities, interacting via “events”.
- Canonical example is discrete event simulation — simulating many semi-independent entities that interact in irregular/unpredictable ways.

Slide 7

Minute Essay

- How scalable are *Pipeline* and *Event-Based Coordination*? if not very, how could you fix that?

Slide 8

Minute Essay Answer

- Neither pattern is very scalable, since they're based on a task decomposition that has one task per pipeline stage or one task per entity. Sometimes additional concurrency can be exposed by further decomposing these stages or entities.