Slide 1

Administrivia

- Deadlines for project proposal, plan extended one class period. (So proposal is due next Tuesday.)

Slide 2

Review — Organization of Our Pattern Language

- Four “design spaces” corresponding to phases in design:
  - *Finding Concurrency* patterns — how to decompose problems, analyze decomposition.
  - *Algorithm Structure* patterns — high-level program structures.
  - *Supporting Structure* patterns — program structures (e.g., SPMD, fork/join), data structures (e.g., shared queue).
  - *Implementation Mechanisms* — no patterns, but generic discussion of “building blocks” provided by programming environments.
- We’ve looked at the bottom two, and at selected parts of the others. Now look at “algorithm structure” level in more detail.

Slide 3

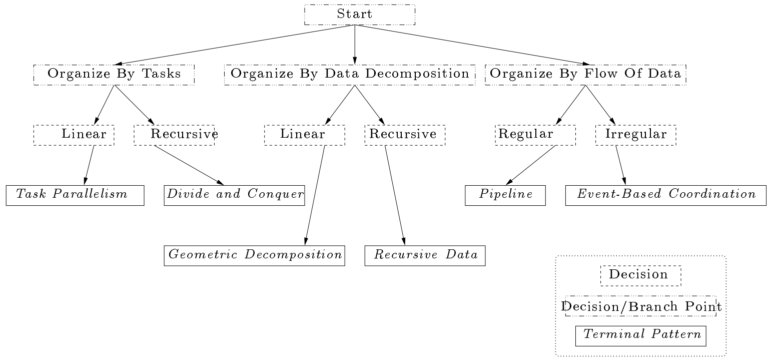
Algorithm Structure Design Space

- 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 4

Algorithm Structure Decision Tree

(Figure 4.2):



Task Parallelism

Slide 5

- 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 previously discussed.
  - 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.

Divide and Conquer

Slide 6

- 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.

Slide 7

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.
  - Matrix multiplication using blocks.

Slide 8

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 9

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 10

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.

Minute Essay

- None — sign in.

Slide 11