Slide 1

Administrivia

- (None?)

Slide 2

Finding Concurrency Patterns — Recap

- Decomposition patterns (*Task Decomposition*, *Data Decomposition*).
- Dependency analysis patterns (*Group Tasks*, *Order Tasks*, *Data Sharing*).
- *Design Evaluation* pattern.

Slide 3

Molecular Dynamics Example — Identify Tasks

- Tasks that find the vibrational forces on an atom.
- Tasks that find the rotational forces on an atom.

  (Together, these are tasks to compute "bonded forces" — those due to chemical bonds.)
- Tasks that find the non-bonded forces on an atom.
- Tasks that update the position and velocity of an atom.
- A task to update the neighbor list for all the atoms (leave this sequential since it's not a big part of the computational load).

Slide 4

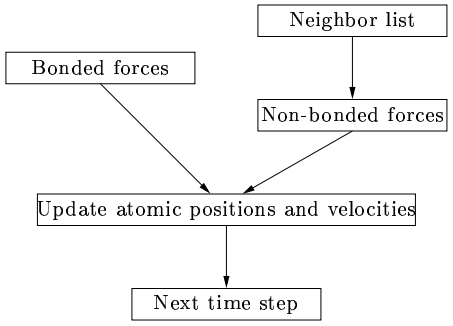
Molecular Dynamics Example — Consider Key Data

- An array of atom coordinates, one element per atom.
- An array of atom velocities, one element per atom.
- An array of lists, one per atom, each defining the neighborhood of atoms considered to be "close".
- An array of forces on atoms, one element per atom.

Slide 5

Molecular Dynamics Example — Group Tasks, Order Groups

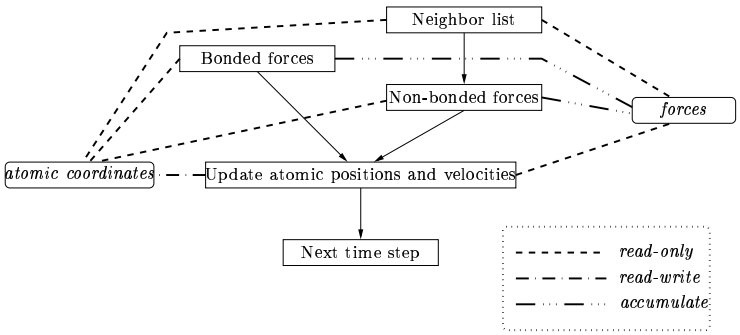
Pictorially (figure 3.4):



Slide 6

Molecular Dynamics Example — Analyze Task/Data Dependencies

Pictorially (figure 3.5):



Slide 7

Heat Diffusion Example — Consider Key Data

- Array for "old values" (time step k).
- Array for "new values" (time step $k + 1$).
- Most computation involves updating (or otherwise operating on these two arrays). Think of partitioning into "chunks".

Slide 8

Heat Diffusion Example — Identify Tasks

- Tasks to compute new values from old values, one per chunk.
- Tasks to compute maximum difference from new and old values, one per chunk.

Slide 9

Heat Diffusion Example — Group Tasks, Order Groups

- Two groups of tasks, computation alternates between them.

Slide 10

Heat Diffusion Example — Analyze Task/Data Dependencies

- Each “chunk” requires data from neighbors — so, some elements of array used by more than one task, but not by all tasks.
- Computation of maximum difference involves “accumulate” data.

Slide 11

Design Evaluation — Suitability for Target Platform

- How many processing elements (PEs) are available? Need at least one task per PE, often want many more — unless we can easily get exactly one task per PE at runtime, with good load balance.
- How are data structures shared among PEs? If there's a lot of shared data, or sharing is very "fine-grained", implementing for distributed memory will likely not be easy or fast.
- How many UEs are available and how do they share data? Similar to previous questions, but in terms of UEs — with some architectures, can have multiple UEs per PE, e.g., to hide latency. For this to work, "context switching" must be fast, and problem must be able to take advantage of it.
- How does time spent doing computation compare to overhead of synchronization/communication, on target platform? May be a function of problem size relative to number of PEs/UEs.

Slide 12

Design Evaluation — Design Quality

- Is it flexible? Will it adapt well to a range of platforms (if appropriate), differing numbers of UEs/PEs, different problem sizes? Does it deal gracefully with "boundary cases"?
- Is it efficient? Can you get good load balance? Is overhead minimal? consider UE creation and scheduling, communication, and synchronization.
- Is it (paraphrasing Einstein) "as simple as possible, but not simpler"? Is it reasonable to think mortals can produce working code relatively quickly? which can later be ported and/or enhanced?

Slide 13

Design Evaluation — Preparation for Next Phase

- How regular are tasks and their data dependencies?
- Are interactions between tasks (or groups of tasks) synchronous or asynchronous?
- Are tasks grouped in the best way?

Slide 14

Minute Essay

- Tell me something you've learned from reading (and not from class).