Slide 1

Administrivia

- None? (Everyone almost done with Homework 2?)

Slide 2

Review — *Supporting Structures* Design Space

- Frequently-used data structures — e.g., shared queue — to be discussed next week.
- Program structures — as discussed briefly last time:
  - *SPMD* (Single Program, Multiple Data) — "like an MPI program".
  - *Master/Worker* — like the name suggests.
  - *Loop Parallelism* — "like an OpenMP program".
  - *Fork/Join* — when none of the others fits.

Slide 3

Example — Molecular Dynamics

- Goal is to simulate what happens to large molecule. Of interest, e.g., in modeling how a drug interacts with a protein.
- Approach is to treat molecule as a collection of balls (atoms) connected by springs (chemical bonds). Then do "standard time-stepping" — divide time into discrete steps, and at each step use classical mechanics to figure out new positions for atoms based on current positions and forces among them.

  In more details . . .

Slide 4

Molecular Dynamics — Computation

- At each time step:
  - Compute forces (vibrational and rotational) on atoms caused by chemical bonds between them. Short-range interaction, so not too much computation here.
  - Compute forces on atoms caused by their electrical charges. Potentially must consider all pairs of atoms, so lots of computation here.
  - Use forces to update atoms' positions and velocities.
  - Compute other physical properties of the system — e.g., energies.
- To reduce the computational load, can limit computation of electrical-charge-induced forces to atoms that are "close". To do this, calculate for each atom a list of "neighbors". If time steps are short, atoms don't move much, and we don't have to do this every step.

Slide 5

Molecular Dynamics Pseudocode

```
Int const N   // number of atoms
Array of Real :: atoms  (3,N) //3D coordinates
Array of Real :: velocities (3,N) //velocity vector
Array of Real :: forces (3,N) //force in each dimension
Array of List :: neighbors(N) //atoms in cutoff volume

loop over time steps
    vibrational_forces (N, atoms, forces)
    rotational_forces (N, atoms, forces)
    neighbor_list (N, atoms, neighbors)
    non_bonded_forces (N, atoms, neighbors, forces)
    update_atom_positions_and_velocities
        (N, atoms, velocities, forces)
    physical_properties ( ... Lots of stuff  ... )
end loop
```

Slide 6

Analysis of Molecular Dynamics Pseudocode

- First step in designing a parallel algorithm is to use patterns in our *Finding Concurrency* space to analyze problem — break it apart into tasks, analyze how they fit together, analyze how they interact with data, etc.
- We'll sketch this . . .

Slide 7

First Step: Identify Tasks With Potential Concurrency

- 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 8

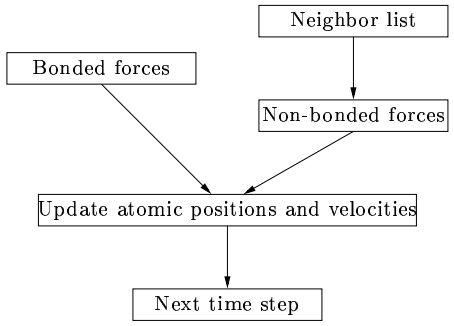
Next Step: Consider Key Data Structures

- 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 9

Next Steps: Group Tasks, Order Groups

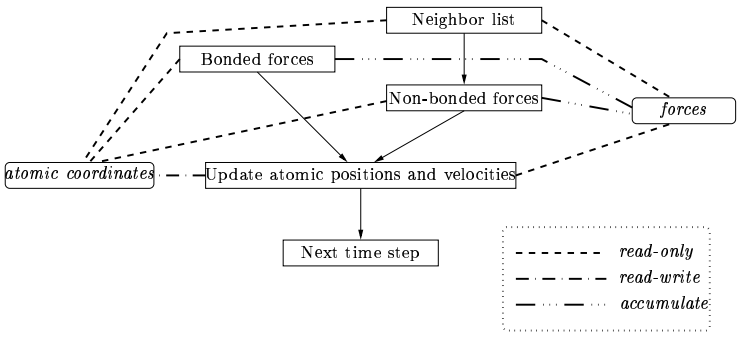
Pictorially (figure 3.4):



Slide 10

Last Step: Analyze Task/Data Dependencies

Pictorially (figure 3.5):



Slide 11

Design of Algorithm for Molecular Dynamics

- Next step is to figure out overall structure for algorithm. We hope for most problems, this will be based on one or more patterns in *Algorithm Structure* design space.
- For this problem, *Task Parallelism* is a good fit. Capsule description:

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

  Notice that we might need "more than one instance" of the pattern for this problem.

  To be continued next time . . .

Slide 12

Minute Essay

- None — sign in.