Slide 1

Administrivia

- (None?)

Slide 2

Molecular Dynamics Example — Recap

- Last time we discussed "the problem" (what we're computing and how) and sketched out how to decompose/analyze it.
- Start now by looking a little more at the next level — after we decide that *Task Parallelism* is a good overall algorithm structure. Focus just on computation of "non-bonded forces" (those caused by electrical charges). Other computations can be treated much the same way. Pseudocode in next slide.

Slide 3

Pseudocode for Non-Bonded Force Computation

```
function non_bonded_forces (N, Atoms, neighbors, Forces)
    Int const N  // number of atoms
    Array of Real :: atoms  (3,N) //3D coordinates
    Array of Real :: forces (3,N) //force in each dimension
    Array of List :: neighbors(N) //atoms in cutoff volume
    Real :: forceX, forceY, forceZ

    loop [i] over atoms
        loop [j] over neighbors(i)
             forceX = non_bond_force(atoms(1,i), atoms(1,j))
             forceY = non_bond_force(atoms(2,i), atoms(2,j))
             forceZ = non_bond_force(atoms(3,i), atoms(3,j))
             force(1,i) += forceX;    force(1,j) -= forceX;
             force(2,i) += forceY;    force(2,j) -= forceY;
             force(3,i) += forceZ;    force(3,j) -= forceZ;
        end loop [j]
    end loop [i]
end function non_bonded_forces
```

Slide 4

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 5

Molecular Dynamics and *Task Parallelism*, Continued

- How to structure program (i.e., which of *Supporting Structures* patterns to use)?

  Obvious choices here *Loop Parallelism* and *SPMD*. Decide based on target platform.

Slide 6

Design of Program for Molecular Dynamics

- Finally, we turn the design into code, probably using patterns from *Supporting Structures* design space, and possibly some information/understanding from *Implementation Mechanisms*.
- Based on previous design steps, consider *Loop Parallelism* and/or *SPMD*. Decide based mostly on target platform. Tables in section 5.3 should be helpful . . .

Slide 7

Program Structures Versus Algorithm Structures

	Geometric Decomposition	*Task Parallelism*	*Divide and Conquer*	*Pipeline*	*Event-Based Coordination*	*Recursive Data*
SPMD	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆	⋆⋆
Loop Parallelism	⋆⋆⋆	⋆⋆⋆⋆	⋆⋆			
Master/Worker	⋆	⋆⋆⋆⋆	⋆⋆	⋆	⋆	⋆
Fork/Join	⋆⋆	⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆

Slide 8

Program Structures Versus Programming Environments

	MPI	OpenMP	Java
SPMD	⋆⋆⋆⋆	⋆⋆⋆	⋆⋆
Loop Parallelism	⋆	⋆⋆⋆⋆	⋆⋆⋆
Master/Worker	⋆⋆⋆	⋆⋆	⋆⋆⋆
Fork/Join		⋆⋆⋆	⋆⋆⋆⋆

Slide 9

Molecular Dynamics and *SPMD* — Key Design Decisions

- Only parallelize computation of non-bonded forces, since that's most of the computational load.
- Keep a copy of the full force and coordinate arrays on each node.
- Have each UE redundantly update positions and velocities for the atoms (i.e., assume it's cheaper to redundantly compute these terms than to do them in parallel and communicate the results).
- Have each UE compute its contributions to the force array and then combine (or reduce) the UEs' contributions into a single global force array copied onto each UE.

Slide 10

Molecular Dynamics and *SPMD* — Code

- Slightly more detailed sequential pseudocode in figure 5.7 (p. 134).
- MPI main pseudocode in figure 5.8 (p. 135. Compare to figure 5.7.
- Pseudocode for computation of non-bonded forces in figure 5.9 (p. 136). Compare to sequential pseudocode in figure 4.4 (p. 72).
- Pseudocode for computation of neighbor list in figure 5.10 (p. 137).. Notice that we exploit the symmetry resulting from Newton's third law.
- A remaining decision — how to distribute atoms among UEs. Cyclic distribution is easy and will probably work okay. If not, could do something more complex — define "owner-computes filter" — boolean function of ID and loop iteration.
- Notice that we could do this in OpenMP too.

Slide 11

Molecular Dynamics and *Loop Parallelism* — Key Design Decisions

- Parallelize computationally intensive loop only (the one for non-bonded forces).
- Figure out what to do about shared variables:
  - Make temporary variables used inside loop private.
  - Make forces array a reduction variable.
- Decide how to map iterations onto UEs. Dynamic schedule works well if available (as it is in OpenMP).
- OpenMP-based pseudocode as shown on p. 161 (figure 5.25 and following `pragma omp` directives). Compare to pseudocode in figure 4.4 (p. 72).

Slide 12

Minute Essay

- I asked a while back about your experiences doing Homework 2, and many people hadn't done it then. I think most people have now, or they're close to being done, so let's try it again: What did you find interesting/helpful/difficult/etc.?