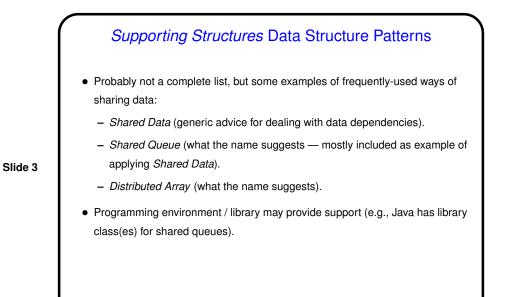


Slide 2

# Supporting Structures Program Structure patterns — Recap Basic ways parallel programs can be structured: SPMD (Single Program, Multiple Data) — "like an MPI program" (but could use same strategy in OpenMP, e.g.). Loop Parallelism — "like an OpenMP program". Master/Worker — as the name suggests. Fork/Join — if you need to be able to create / wait for UEs in any arbitrary way. (We've seen examples of most. Look also at MPI master/worker.)



Shared Queue
Many applications — especially ones using a master/worker approach — need a shared queue. Programming environment might provide one, or might not. Nice example of dealing with a shared data structure anyway.
Java code in figures 5.37 (p. 185) through 5.40 (p. 189) presents a step-by-step approach to developing implementation.

# Shared Queue, Continued

• Simplest approach to managing a shared data structure where concurrent modifications might cause trouble - one-at-a-time execution. Shown in figures 5.37 (nonblocking) and 5.38 (block-on-empty). Only tricky bits are use of dummy first node and details of take. Reasons to become clearer later. Usually a good idea to try simplest approach first, and only try more complex

Slide 5

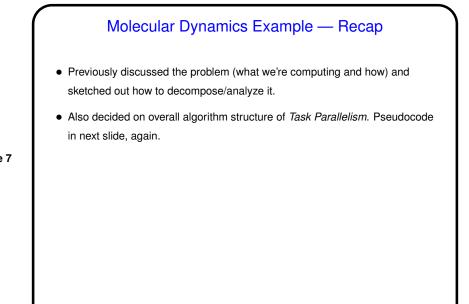
all evil." Attributed to D. E. Knuth; may actually be C. A. R. Hoare.) • Here, next thing to try is concurrent calls to put and take. Not too hard for nonblocking queue — figure 5.39. Tougher for block-on-empty queue figure 5.40. In both cases, must be very careful.

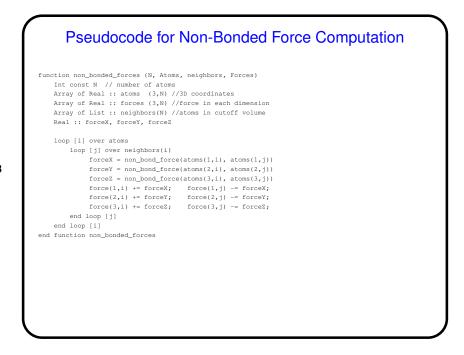
ones if better performance is needed. ("Premature optimization is the root of

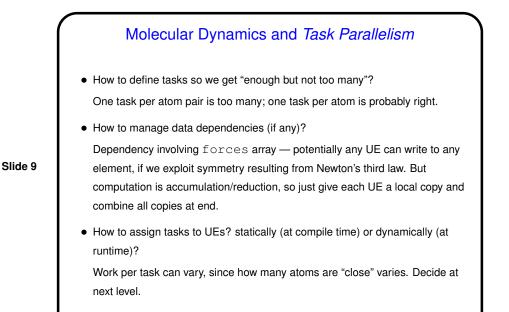
• If still too slow, or a bottleneck for large numbers of UE, explore distributed queue.

# Distributed Array, Recap

- Ideas are fairly straightforward, easy to draw. Code is apt to be messy but good use of functions and macros can help.
- Example heat-distribution problem (later slides).







# **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* (to be discussed later).
- Based on previous design steps, consider *Loop Parallelism* and/or *SPMD*. Decide based mostly on target platform.

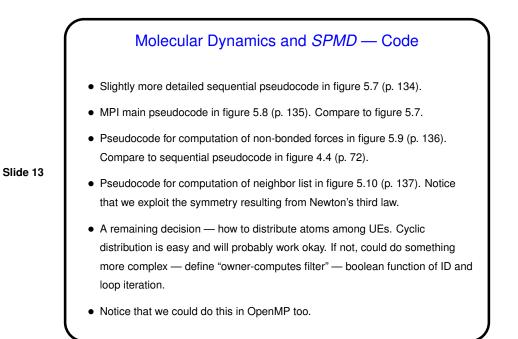
# Molecular Dynamics and *Loop Parallelism* — Key Design Decisions

- 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 in figure 5.25 (p. 161) and following pragma omp directives). Compare to pseudocode in figure 4.4 (p. 72).



- Only parallelize computation of non-bonded forces, since that's most of the computational load.
- Slide 12
- 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.



# Heat Diffusion Example — Recap

- Previously discussed the problem (what we're computing and how) and sketched out how to decompose/analyze it.
- Also decided on overall algorithm structure of *Geometric Decomposition*.

