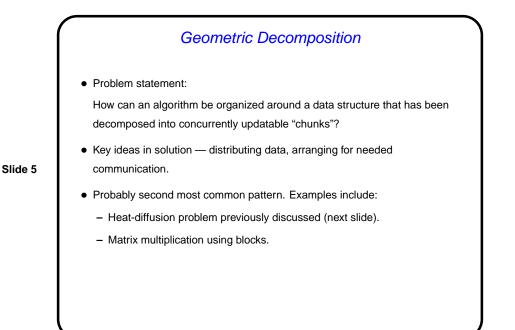


Molecular Dynamics and Task Parallelism
How to define tasks so we get "enough but not too many"? One task per atom pair is too many; one task per atom is probably right.
How to manage data dependencies (if any)? Dependency involving forces array — potentially any UE can write to any element, if we exploit 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.

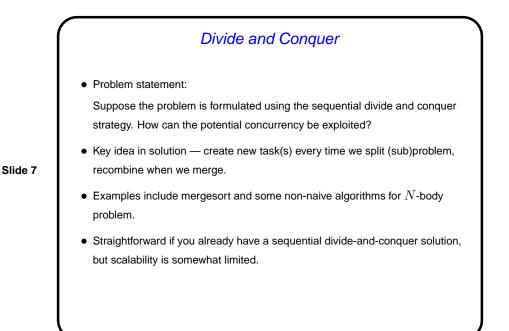


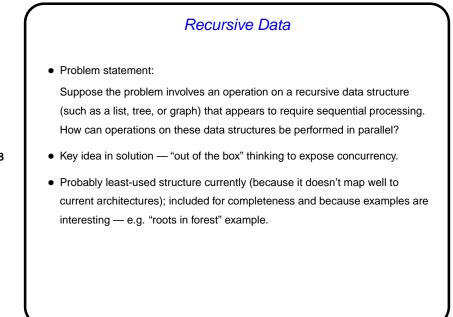
Heat Diffusion and Geometric Decomposition
How to distribute data?

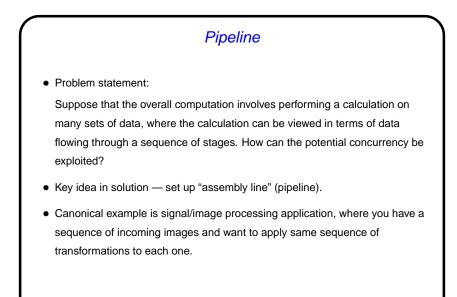
One chunk per UE will probably work well. (Note that for other problems it might not.) Might be nice to include in data structure a place to store values from neighboring chunks. More in *Distributed Array*, next chapter.

How to synchronize/communicate?

With shared memory, just need barrier synchronization.
With distributed memory, need to exchange values with neighbor UEs, also perform reduction.







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.

