Slide 1

Administrivia

- (None.)

Slide 2

Example Applications — Review/Continued

- Before starting on *Finding Concurrency* patterns — two example applications to be used as running examples.

Slide 3

Example — Molecular Dynamics

- Goal is to simulate what happens to large molecule.
- 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.

Slide 4

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 5

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 6

Example — Heat Diffusion

- A simple example, representative of a big class of scientific-computing applications — “heat distribution problem”.
- Goal is to simulate what happens when two ends of a pipe are put in contact with things at different (constant) temperatures — pipe conducts heat, its temperature changes over time, eventually converging on a smooth gradient.
- Can model mathematically how temperature in pipe changes over time using partial differential equations.
- Can approximate solution by “discretizing” — spatially and with regard to time.

Slide 7

Heat Diffusion Code

```
double *uk = malloc(sizeof(double) * NX);
double *ukp1 = malloc(sizeof(double) * NX);
double *temp;
double dx = 1.0/NX; double dt = 0.5*dx*dx;
double maxdiff, diff;

initialize(uk, ukp1);

for (int k = 0; (k < NSTEPS) && (maxdiff >= threshold); ++k) {

  /* compute new values */
  for (int i = 1; i < NX-1; ++i) {
    ukp1[i]=uk[i]+ (dt/(dx*dx))*(uk[i+1]-2*uk[i]+uk[i-1]);
  }

  /* check for convergence */
  maxdiff = 0.0;
  for (int i = 1; i < NX-1; ++i) {
    diff = fabs(uk[i] - ukp1[i]);
    if (diff > maxdiff) maxdiff = diff;
  }

  /* "copy" ukp1 to uk by swapping pointers */
  temp = ukp1; ukp1 = uk; uk = temp;

  printValues(uk, k);
}
```

Slide 8

Finding Concurrency Design Space

- Starting point in our grand strategy for developing parallel applications. Overall idea — capture how experienced parallel programmers think about initial design of parallel applications. Might not be necessary if clear match between application and an *Algorithm Structure* pattern.
- Idea is to work through three groups of patterns in sequence (possibly with backtracking):
  - Decomposition patterns (*Task Decomposition*, *Data Decomposition*): Break problem into tasks that maybe can execute concurrently.
  - Dependency analysis patterns (*Group Tasks*, *Order Tasks*, *Data Sharing*): Organize tasks into groups, analyze dependencies among them.
  - *Design Evaluation*: Review what you have so far, possibly backtrack.
- Keep in mind — best to focus attention on computationally intensive parts of problem.

Slide 9

Task-Based Versus Data-Based Decomposition

- Two basic approaches to decomposing a problem — task-based and data-based. Usually one will seem more logical than the other, but may need to think through both.
- Either way, you'll look at both tasks and data; difference is in which you look at first, and then the other follows.

Slide 10

Task Decomposition

- Goal here is to break up (some of) computation into "tasks" — logical elements of overall computation that might be independent enough to do concurrently.
- At this stage, try to stay abstract and portable; also try to identify lots of tasks (can always recombine them later if too many), as independent of each other as possible.
- Places to look for tasks include groups of function calls (e.g., in divide-and-conquer strategy), loop iterations (e.g., many examples we've discussed).
- Simple example — matrix multiplication.
- Once you have this, consider data related to each task (*Data Decomposition*).

Slide 11

Data Decomposition

- Goal here is to break up (some of) problem data into parts ("chunks") that can be operated on concurrently. Good choice if most computation consists of updates to big data structure(s).
- Again, try to stay abstract and portable; also try to "parameterize" decomposition so you can easily try various choices at runtime.
- Data structures to look at include arrays, recursive structures such as trees.
- Simple example — matrix multiplication.
- Once you have this, consider computation related to each chunk of data (*Task Decomposition*).

Slide 12

Molecular Dynamics Example — Task Decomposition

- 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 (the ones due to electrical charges).
- Tasks that update the position and velocity of an atom.
- Tasks that update the neighbor list for an atom. (Or we could consider updating all the neighbor lists as one task, as in the book, if we think it won't be done very often and therefore is not worthwhile to parallelize.)

Slide 13

Molecular Dynamics Example — Data Decomposition

- 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.
- Decompose each of these to match task decomposition — into elements corresponding to individual atoms.

Slide 14

Heat Diffusion Example — Data Decomposition

- Key data structures:
  - 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 15

Heat Diffusion Example — Task Decomposition

- Tasks to compute new values from old values, one per chunk.
- Tasks to compute maximum difference between new and old values, one per chunk.
- Task to swap pointers (to fake copying new values to old values).

Slide 16

Group Tasks

- Once you've broken down problem into tasks / data chunks, need to put it back together as design for parallel algorithm.
- First step — look for "groups of tasks" — logically related, or interdependent, or all with same constraints, etc. Often just one group.

Slide 17

Order Tasks

- Next step — identify constraints on groups of tasks. Possibilities:
  - "First this, then that."
  - "All of these together."

Slide 18

Molecular Dynamics Example — Group Tasks, Order Groups

- Task groups based on list of a few slides back — each type of task (e.g., compute rotational forces) defines a task group.
- Ordering constraints, for each timestep:
  - Task group to compute neighbor list must run before task group to compute non-bonded forces.
  - Task groups to compute bonded and non-bonded forces must run before task group to update positions and velocities.
  - Task group to update positions and velocities must run before next timestep.
- (Also see Figure 3.4 in the book.)

Slide 19

Heat Diffusion Example — Group Tasks, Order Groups

- Task groups based on list of a few slides back — each type of task (e.g., compute new values from old values) defines a task group.
- Ordering constraints, for each timestep:
  - Three tasks groups must run in sequence.
  - All task groups must run before next timestep.

Slide 20

Minute Essay

- What did you find most interesting about Homework 1? most difficult?