Slide 1

Administrivia

- Reminder: Please do watch the short recording for 10/01, and then send me an e-mail so you get that one attendance point.

Slide 2

Example Applications

- Before starting on *Finding Concurrency* patterns — two example applications to be used as running examples.
- Meant to be representative as the kinds of problems our book was intended to address.

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.

  At each step use classical mechanics to figure out new positions for atoms based on current positions and forces among them.

  In more detail . . .

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 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 in each, so 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

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 7

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 8

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 9

Minute Essay

- Questions?