MiniMD in Chapel

Brad Chamberlain (reporting for Ben Harshbarger)
Chapel Intern, Cray Inc./University of Washington

MiniMD in Chapel

(or: Molecular Dynamics for ~~dummies~~ enthusiasts of emerging parallel languages who have great taste by a dummy)

Brad Chamberlain (reporting for Ben Harshbarger)

Chapel Intern, Cray Inc./University of Washington

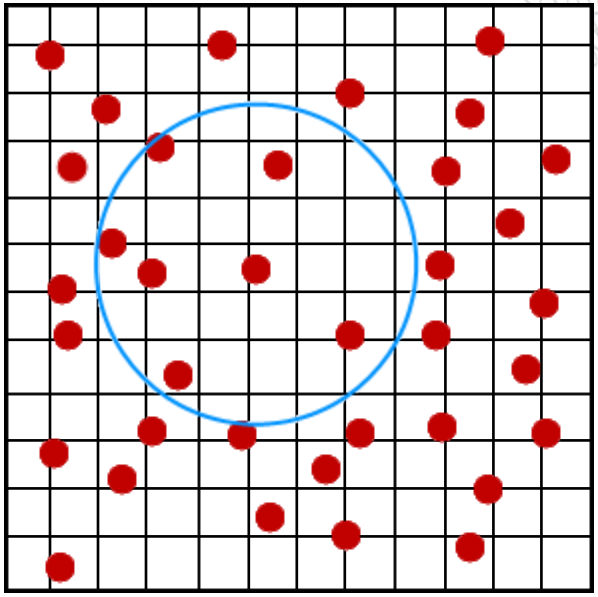
What is MiniMD?

- **"Mini Molecular Dynamics"**
  - A proxy application from Sandia's Mantevo group
  - Representative of key idioms from their real applications
  - ~5000 lines of C++/MPI
    - ~2000 lines in Chapel
- **Molecular Dynamics?**
  - Computing physical properties like energy, pressure, and temperature for a simulated space containing moving atoms
- **An important strategic benchmark for Chapel**

Store atoms in spatial bins

- Given a bunch of atoms…

```
record atom {
  var vel, force, position : 3*real;
}
```



- Place atoms in bins based on spatial position

```
const binSpace = {1..12, 1..12};
var perBinSpace = {1..8};
var bins : [binSpace] [perBinSpace] atom;
```

- Reduce number of atoms to compute against
  - Use cutoff to build list of neighbors
  - Complexity goes from $O(n^2)$ to ~$O(n)$

Compute forces between atoms

```
forall bin in bins {
  for atom in bin {
    for neighbor in atom.neighbors {
      if distance(atom, neighbor) < cutoff {
        updateForces(atom, neighbor);
      }
    }
  }
}
```

Now let’s go to distributed memory…

Distributing Bins in C++/MPI

```
while(ipx <= nprocs) {
    if(nprocs % ipx == 0) {
      nremain = nprocs / ipx;
      ipy = 1;

      while(ipy <= nremain) {
        if(nremain % ipy == 0) {
          ipz = nremain / ipy;
          surf = area[0] / ipx / ipy +
                 area[1] / ipx / ipz +
                 area[2] / ipy / ipz;

          if(surf < bestsurf) {
            bestsurf = surf;
            procgrid[0] = ipx;
            procgrid[1] = ipy;
            procgrid[2] = ipz;
          }
        }

        ipy++;
      }
    }

    ipx++;
  }
```

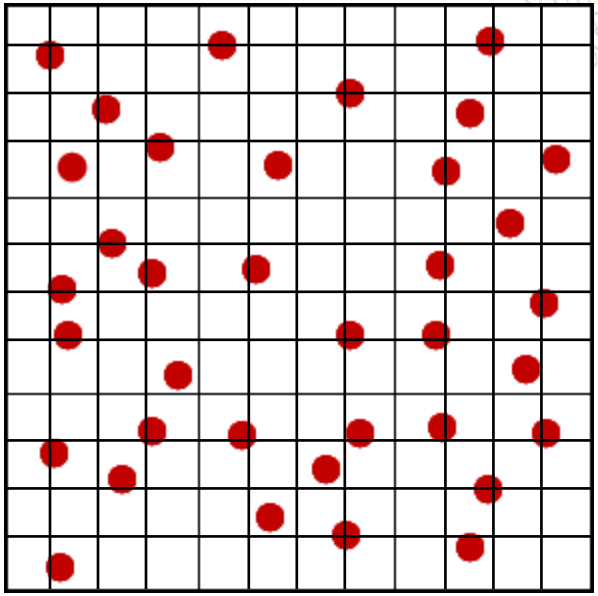
```
 int reorder = 0;
  periods[0] = periods[1] = periods[2] = 1;

  MPI_Cart_create(MPI_COMM_WORLD, 3, procgrid,
                  periods, reorder, &cartesian);
  MPI_Cart_get(cartesian, 3, procgrid, periods,
               myloc);
  MPI_Cart_shift(cartesian, 0, 1, &procneigh[0][0],
                 &procneigh[0][1]);
  MPI_Cart_shift(cartesian, 1, 1, &procneigh[1][0],
                 &procneigh[1][1]);
  MPI_Cart_shift(cartesian, 2, 1, &procneigh[2][0],
                 &procneigh[2][1]);

for(int idim = 0; idim < 3; idim++)
    for(int i = 1; i <= need[idim]; i++, iswap += 2) {
      MPI_Cart_shift(cartesian, idim, i,
                     &sendproc_exc[iswap],
                     &sendproc_exc[iswap + 1]);
      MPI_Cart_shift(cartesian, idim, i,
                     &recvproc_exc[iswap + 1],
                     &recvproc_exc[iswap]);
    }
```

+ Hundreds of lines of additional MPI setup

Distributing Bins in Chapel

- Given a bunch of atoms…

```
record atom {
  var vel, force, position : 3*real;
}
```



- Place atoms in bins based on spatial position

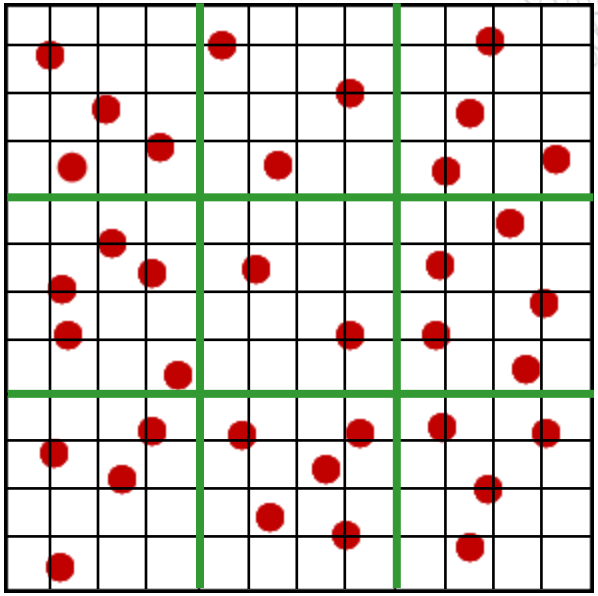
```
const binSpace = {1..12, 1..12};
var perBinSpace = {1..8};
var bins : [binSpace] [perBinSpace] atom;
```

- Reduce number of atoms to compute against
  - Use cutoff to build list of neighbors
  - Complexity goes from $O(n^2)$ to ~$O(n)$

Distributing Bins in Chapel

- Given a bunch of atoms…

```
record atom {
  var vel, force, position : 3*real;
}
```



- Place atoms in bins based on spatial position

```
const binSpace = {1..12, 1..12} dmapped Block(…);
var perBinSpace = {1..8};
var bins : [binSpace] [perBinSpace] atom;
```

- Reduce number of atoms to compute against
  - Use cutoff to build list of neighbors
  - Complexity goes from $O(n^2)$ to ~$O(n)$

Compute forces between atoms (dist. mem.)

Runtime distributes work across locales and handles communication of data

```
forall bin in bins {
  for atom in bin {
    for neighbor in atom.neighbors {
      if distance(atom, neighbor) < cutoff {
        updateForces(atom, neighbor);
      }
    }
  }
}
```

There must be a catch…?

Yes, performance! (today, at least)

So what's an impatient HPC programmer to do?

Using Chapel's Multiresolution Features…

1) Ben wrote an explicit version of MiniMD

- SPMD + manually fragmented data structures as in an MPI code
  - but using PGAS array slicing rather than message passing

2) Then he refactored that logic into a *Stencil* domain map:

- an extension of *Block* supporting ghost cells/overlap regions/fluff
  …with user-callable routines to update these values

```
const binSpace = {1..12, 1..12} dmapped Stencil(…);
var perBinSpa
var bins : [b
```

```
bins.updateFluff();
forall bin in bins {
  for atom in bin {
    for neighbor in atom.neighbors {
      if distance(atom, neighbor) < cutoff {
        updateForces(atom, neighbor);
      }
    }
  }
}
```

Next Steps

- **Longer-term:**
  - Have Chapel compiler automatically insert calls to update fluff
    - (reproduce ZPL work within Chapel)
- **Shorter-term:**
  - Detailed review of code for performance/elegance improvements
  - Performance studies, comparisons, and optimizations

Chapel Productivity

Ben…

- an undergraduate
- with no significant parallel programming experience
- no Chapel experience
- no MiniMD experience

…wrote 4 elegant versions of MiniMD in ~13 weeks

- **2 weeks:** learned Chapel, miniMD, **wrote single-locale transliteration**
- **2 weeks:** edited for Chapel style based on feedback from team
- **2 weeks:** performance improvements and **Block multi-locale version**
- **3 weeks:** **explicitly distributed version**
- **2.5 weeks:** wrote the **Stencil distribution version** (and the dist. itself)
- **1.5 weeks:** merged single-locale, Block, and Stencil versions into one
  - select between them with a compiler flag

For more information

- Download Chapel release
- See examples/benchmarks/miniMD/

CRAY
THE SUPERCOMPUTER COMPANY

http://chapel.cray.com chapel_info@cray.com http://sourceforge.net/projects/chapel/