Using Charm++ with Arrays

Laxmikant (Sanjay) Kale
Parallel Programming Lab
Department of Computer Science, UIUC
charm.cs.uiuc.edu

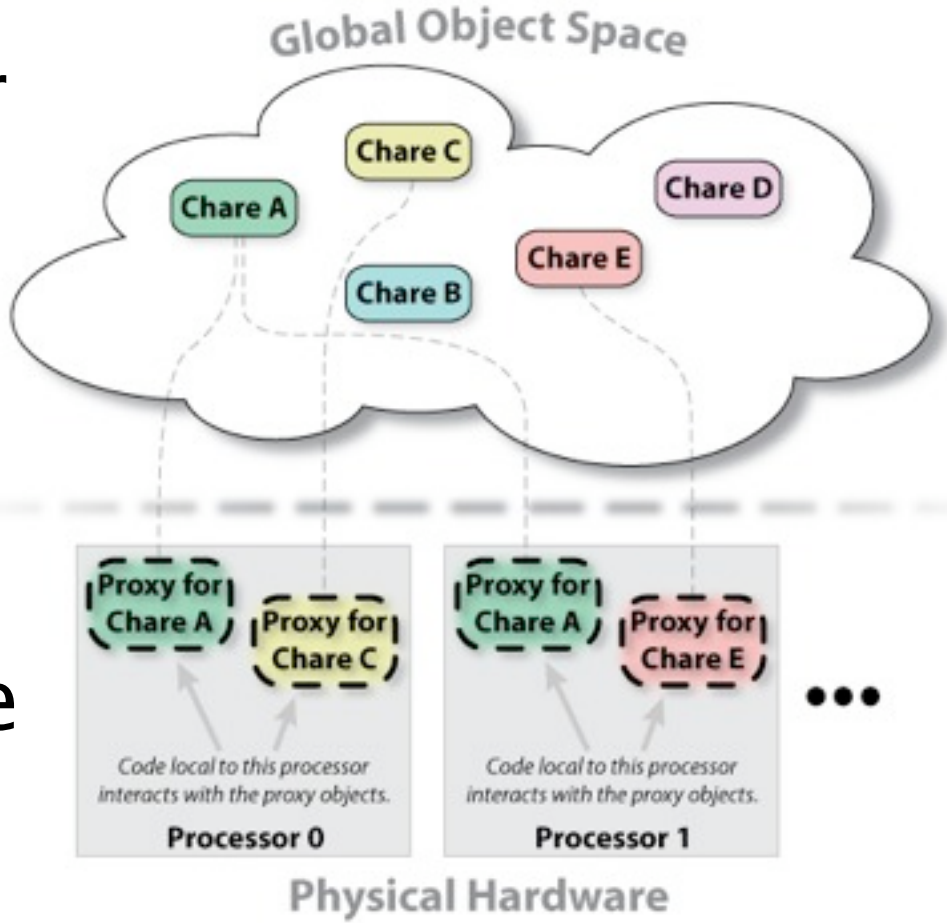
ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

PARALLEL PROGRAMMING LAB
DEPT OF COMPUTER SCIENCE, UNIVERSITY OF ILLINOIS

PPL UIUC

Proxy Objects

Entry methods for each chare are invoked using proxy objects--lightweight handles to potentially remote chares.

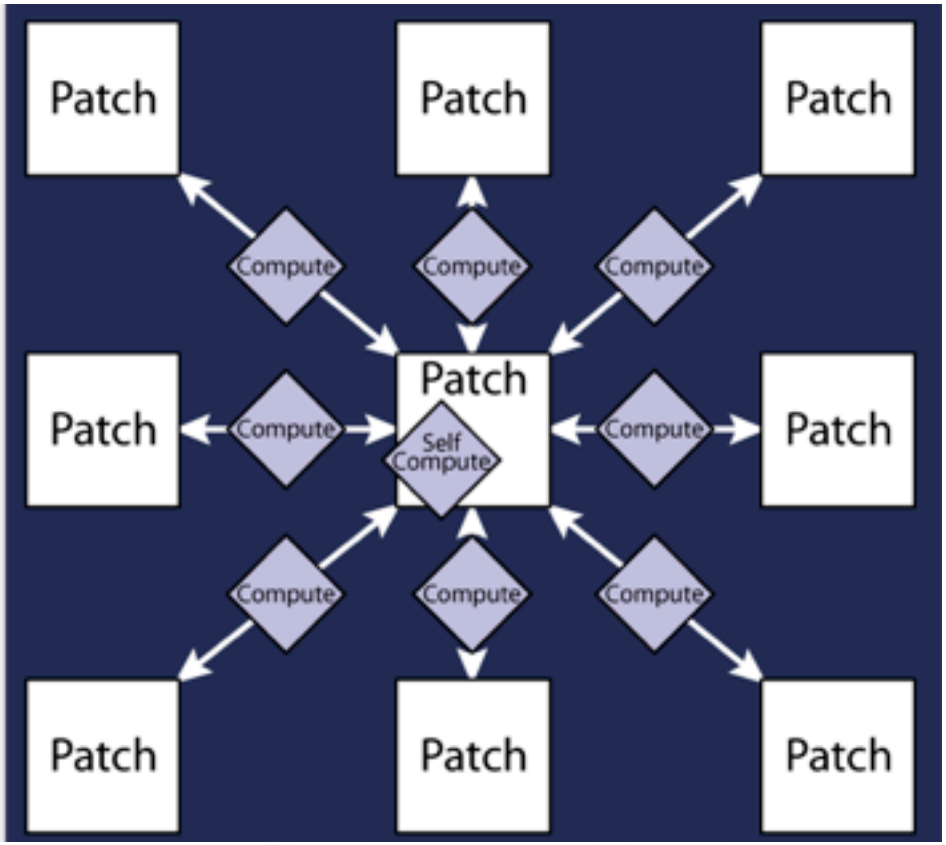


Limitations of Plain Proxies

- In a large program, keeping track of all the proxies is difficult
- A simple proxy doesn't tell you anything about the chare other than its type.
- Managing collective operations like broadcast and reduce is complicated.

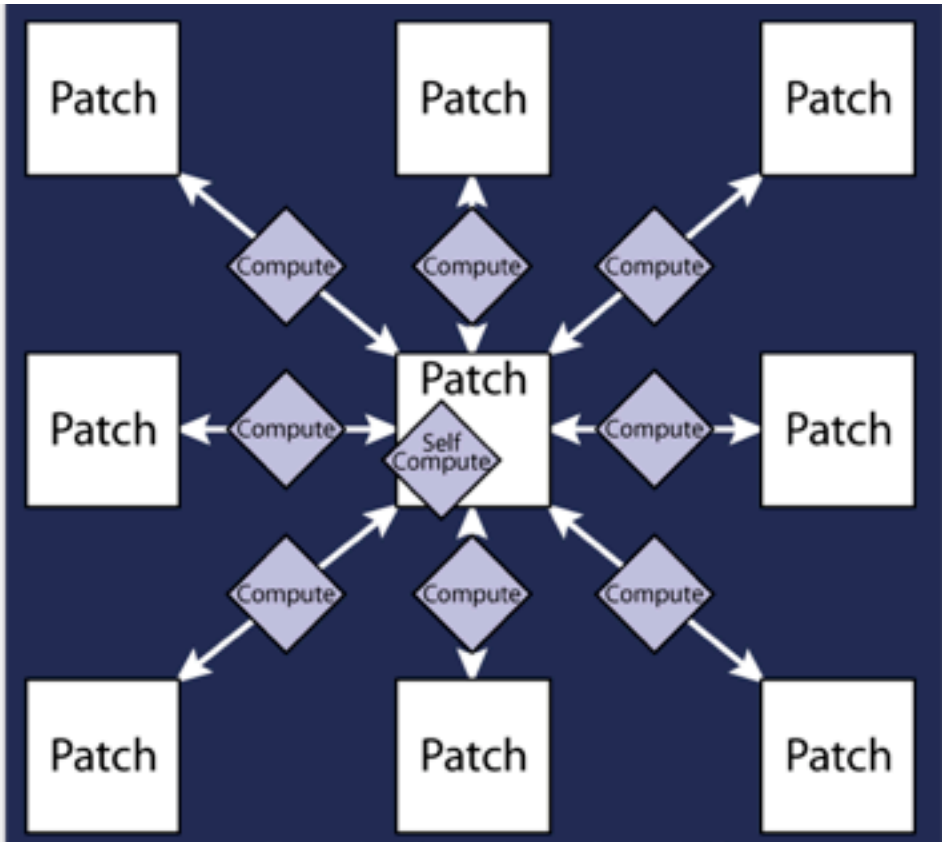
Example: Molecular Dynamics

- Patches: blocks of space containing particles.
- Computes: chares which compute the interactions between particles in two patches.



Example: Molecular Dynamics

- We could write this program using just chares and plain proxies
- But, it'll be much simpler if we use Chare Arrays.



note: you can find complete MD code in examples/charm++/Molecular2D in the Charm distribution.

Chare Arrays

- Arrays organize chares into indexed collections.
- Each chare in the array has a proxy for the other array elements, accessible using simple syntax

```
sampleArray[i] // i'th proxy
```

Array Dimensions

- Anything can be used as array indices
  - integers
  - tuples
  - bit vectors
  - user-defined types

Arrays in MD

array [2D] Patch { ... }

x and y coordinates of patch

array [4D] Compute { ... }

x and y coordinates of both patches involved

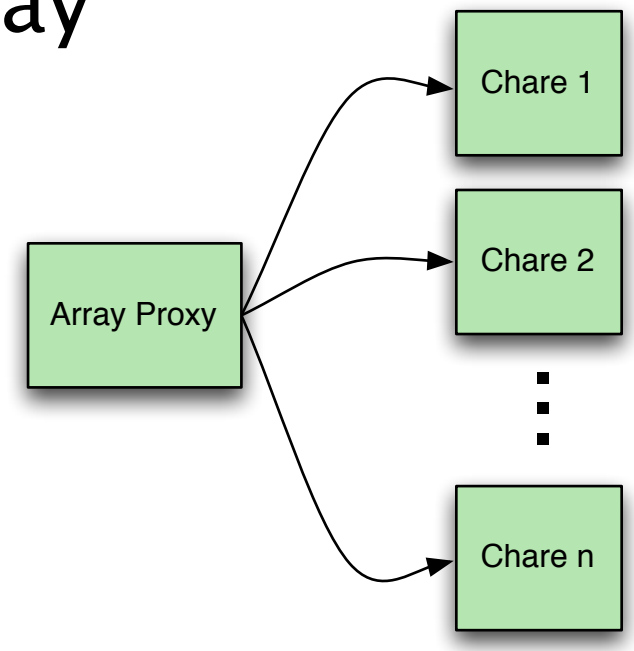
note: arrays can be sparse

Managing Mapping

- Arrays let the programmer control the mapping of array elements to PEs.
  - Round-robin, block-cyclic, etc
  - User defined mapping

Broadcasts

- Simple way to invoke the same entry method on each array element



- Syntax:

```
// call on one element
sampleArray[i].method()
// broadcast to whole array
sampleArray.method()
```

Broadcasts in MD

Used to fire off the first timestep after initialization is complete

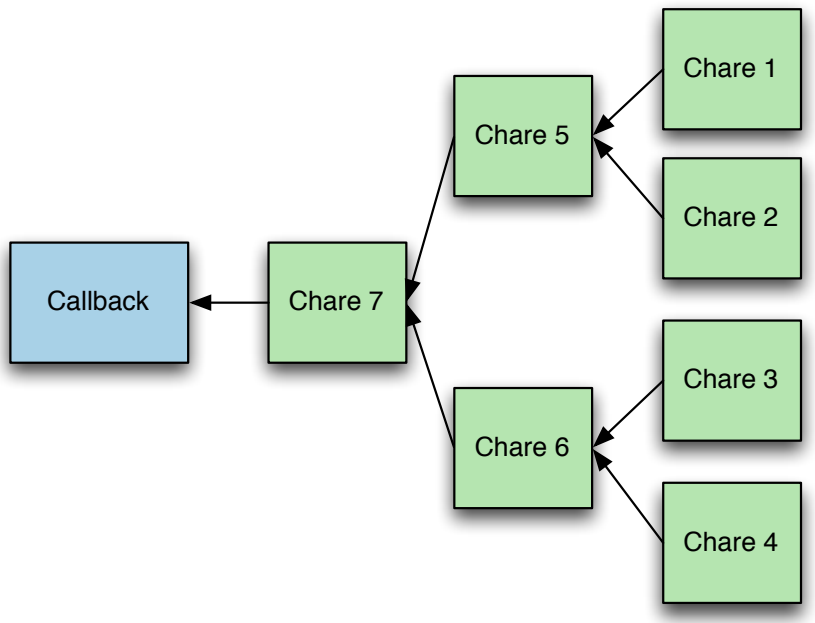
```
// All chares are created, time to
// start the simulation
patchArray.start();
```

Reductions

- No global flow of control, so each chare must contribute data independently using `contribute()`.
- A user callback (created using `CkCallback`) is invoked when the reduction is complete.

Reductions

- Runtime system assembles reduction tree
- User specifies reduction operation
- At root of tree, a callback is performed on a specified chare



Reductions in MD

Used to determine when all computes have been created.

```
// all local computes have been created
contribute(0, 0, CkReduction::concat, // do nothing
   CkCallback(
       CkIndex_Main::computeCreationDone(),
       mainProxy));
```

Function to call

Chare to handle callback function