A Multiparadigm Approach to Parallel Programming

Presented by

Aaron Becker, Pritish Jetley, and Phil Miller

11/30/2009

ILLINOIS

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

PARALLEL PROGRAMMING LAB

DEPT OF COMPUTER SCIENCE, UNIVERSITY OF ILLINOIS

PPL UIUC

Multiparadigm Computing

- A parallel computing paradigm is a way of expressing concurrency in an application
- Multiparadigm applications use multiple paradigms together
- For example, mixed-mode MPI/ OpenMP applications

Some Parallel Paradigms

- MPI (Message passing)
- OpenMP (Shared memory)
- OpenCL (Accelerator)
- Charm++ (Message driven)
- MapReduce

Not Parallel Paradigms

- C++
- Object-oriented programming
- Clusters

Why Multiparadigm?

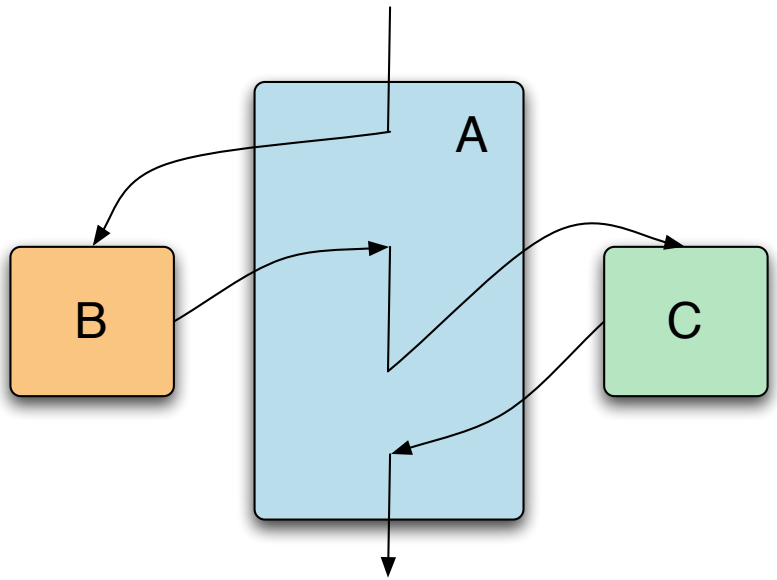
- Interoperability
- Flexibility
- Efficiency
- Simplicity

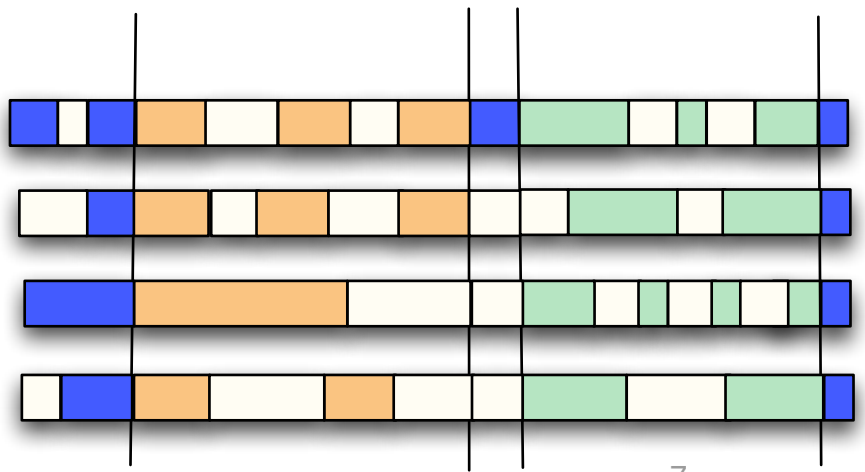
Composing Parallel Modules

- High-performance, scalable libraries are very valuable.
- What if your library
  - requires MPI?
  - assumes it will run by itself?
  - has to coexist with other parallel libraries?

Composing Parallel Modules

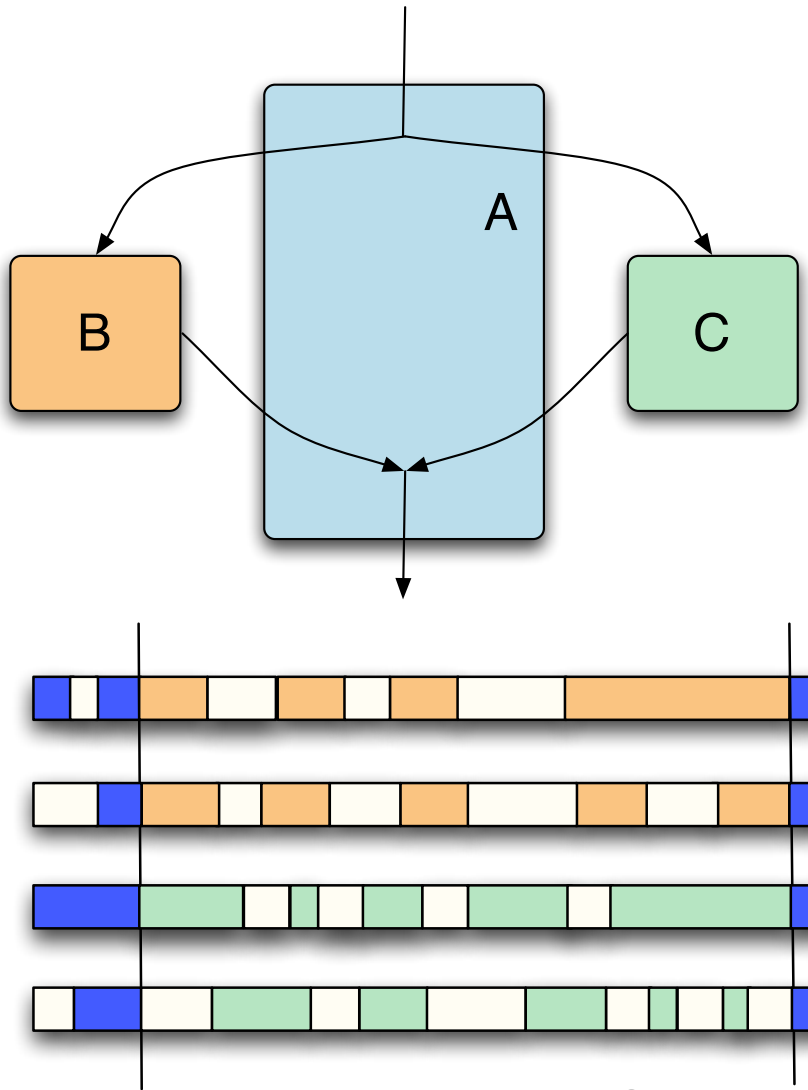
Sequentialization



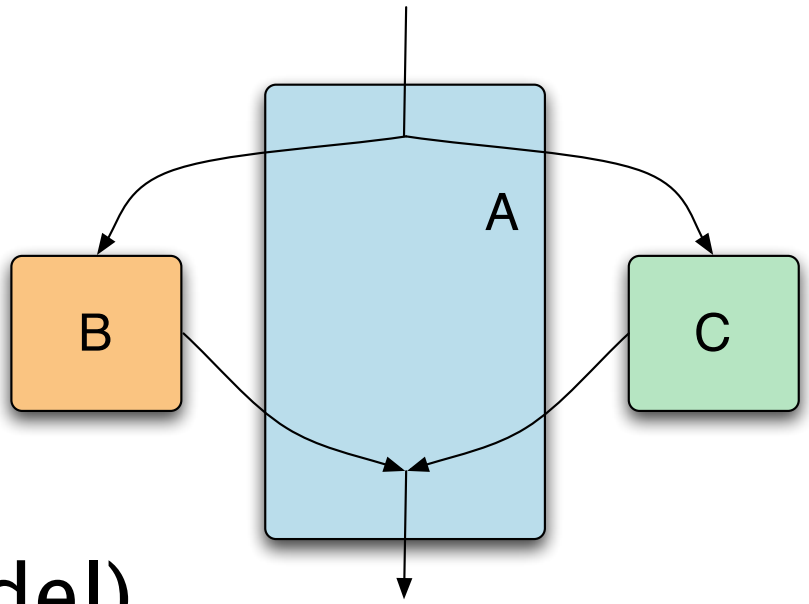


Composing Parallel Modules

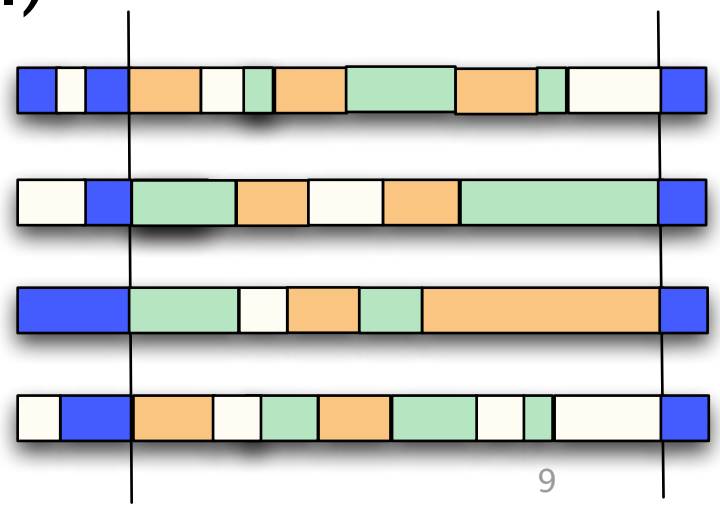
Space Division



Composing Parallel Modules



Composition
(the Charm++ model)



Benefits of Parallel Composition

- Flexibility
  - No need to statically allocate hardware resources.
  - No artificial synchronization points.
- Utilization
  - Overlap of idle time from one library with work from another.
  - Each library need not scale to the whole machine.

PPL UIUC

Simplicity

- Sometimes you don't need a fully general programming model
- Complete freedom implies the freedom to express all possible incorrect programs

Incomplete Models

Downside:

some things won't be expressible

PPL UIUC

Incomplete Models

Upside:

some things won't be expressible

PPL UIUC

Incomplete Models

Upside:

- stronger safety guarantees
- easier to optimize
- possibility for greater expressiveness

PPL UIUC

Incomplete Models

- Idea: impose restrictions on the programming model in exchange for improved safety or performance.
- Use the restricted model where it fits best, and fall back to general-purpose models elsewhere.

Incomplete Models in Charm

Multiphase Shared Arrays

disciplined shared memory

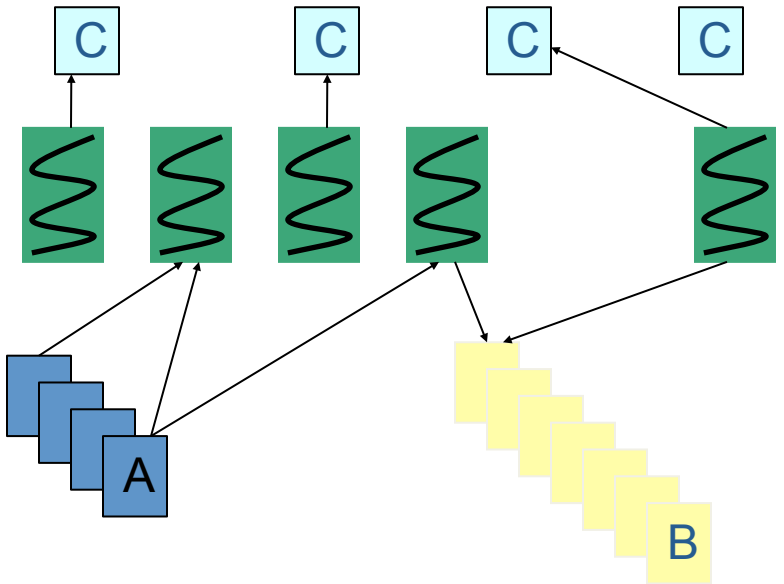
Charisma

static data-flow

Tied together with the Charm RTS

Multiphase Shared Arrays (MSA)

- An MSA application consists of:
  - Multiple computing threads
  - Data arrays (MSAs), whose elements are accessible by all threads
  - A local cache of MSA data for each thread



MSA Phases

- Observation: many shared memory applications can be split into phases.
- These phases are part of the algorithm, but aren't expressed directly in code.

MSA Phases

- In MSA, each shared array has explicit phases, e.g.
  - Read only
  - Exclusive Write
  - Accumulate
- Synchronization at phase boundaries

Safety Properties

- Provably race free
- All MSA operations obey phase semantics
- All threads accessing an MSA agree on the same sequence of phases

Asynchrony

- On remote access, fetch a remote chunk of the array into local cache
- While waiting, other work can be scheduled
  - With overdecomposition, other MSA threads can work.
  - Unrelated chares using other programming models can run, in a multiparadigm application.

Example: Plimpton MD

```
for  timestep = 0 to Tmax {
  // Phase I : Force Computation: for a section of the interaction matrix
  for i = i_start to i_end
    for j = j_start to j_end
      if (nbrlist[i][j]) { // nbrlist enters ReadOnly mode
        force = calculateForce(coords[i], atominfo[i], coords[j], atominfo[j]);
        forces[i] += force; // Accumulate mode
        forces[j] += -force;
      }
  nbrlist.sync(); forces.sync();  coords.sync();


  for k = myAtomsbegin to myAtomsEnd // Phase II : Integration
     coords[k] = integrate(atominfo[k], forces[k]); // WriteOnly mode
  coords.sync(); atominfo.sync(); forces.sync();


  if  (timestep %8 == 0) { // Phase III: update neighbor list every 8 steps
     for i = i_start to i_end
        for j = j_start to j_end
```