Higher Level Languages on Adaptive Run-Time System

Chao Huang
Parallel Programming Lab
University of Illinois at Urbana-Champaign

Motivation

- Productivity and Performance
  - Different algorithms → different tools
  - Charm++/AMPI: powerful adaptive run-time system

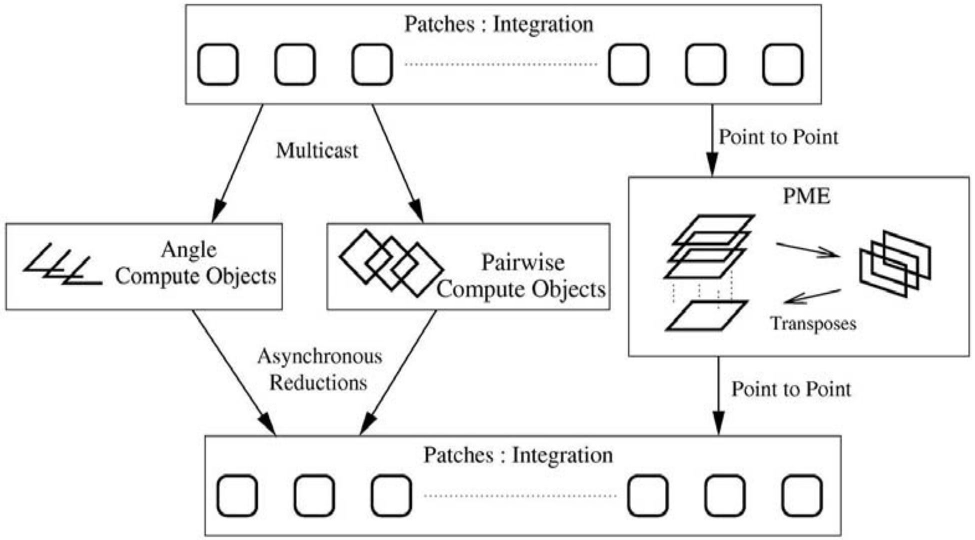
	Local Data	Global Data
Local Control	Charm++ MPI (AMPI)	MSA
Global Control		OpenMP Charisma

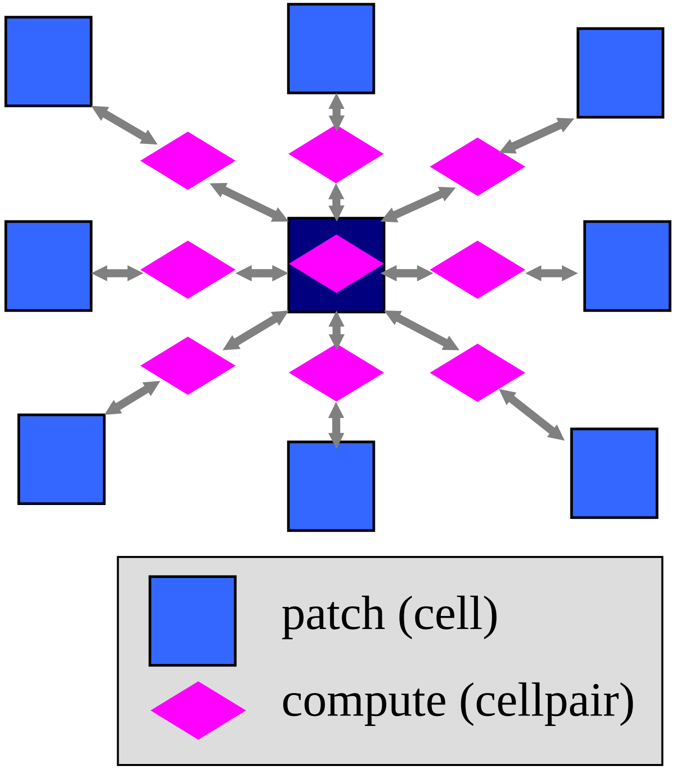
Outline

- Motivation
- Expressing Flow of Control
- Charisma
- Multiphase Shared Array (MSA)
- Conclusions

Example: MD

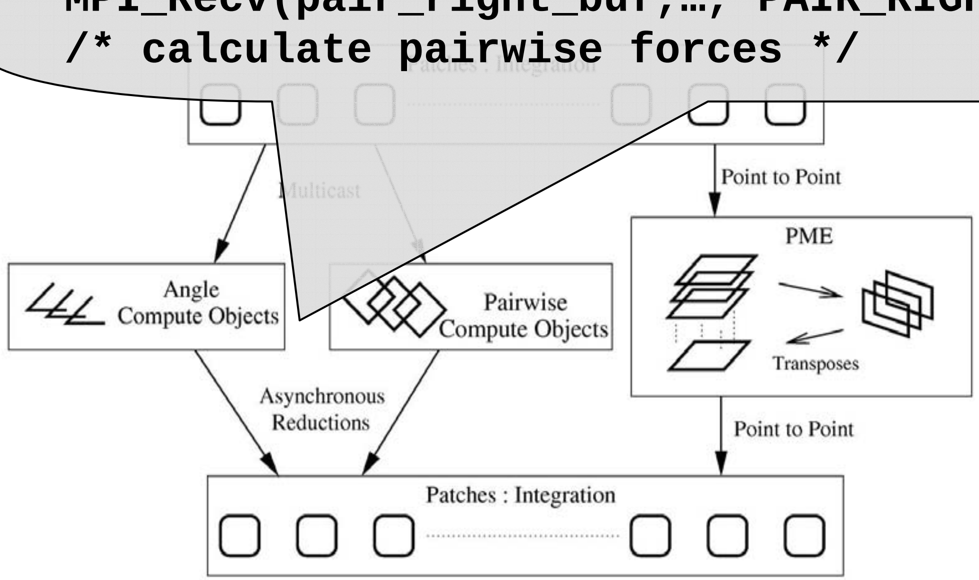
- Structure of a simple MD simulation

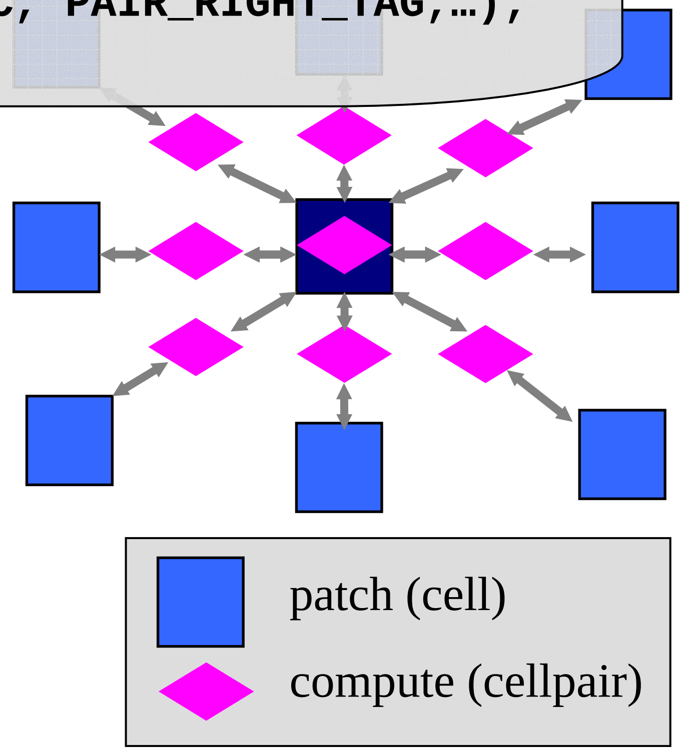




Example: MD

```
MPI_Recv(angle_buf,…, ANGLE_SRC, ANGLE_TAG,…);
/* calculate angle forces */
MPI_Recv(pair_left_buf,…, PAIR_LEFT_SRC, PAIR_LEFT_TAG,…);
MPI_Recv(pair_right_buf,…, PAIR_RIGHT_SRC, PAIR_RIGHT_TAG,…);
/* calculate pairwise forces */
```

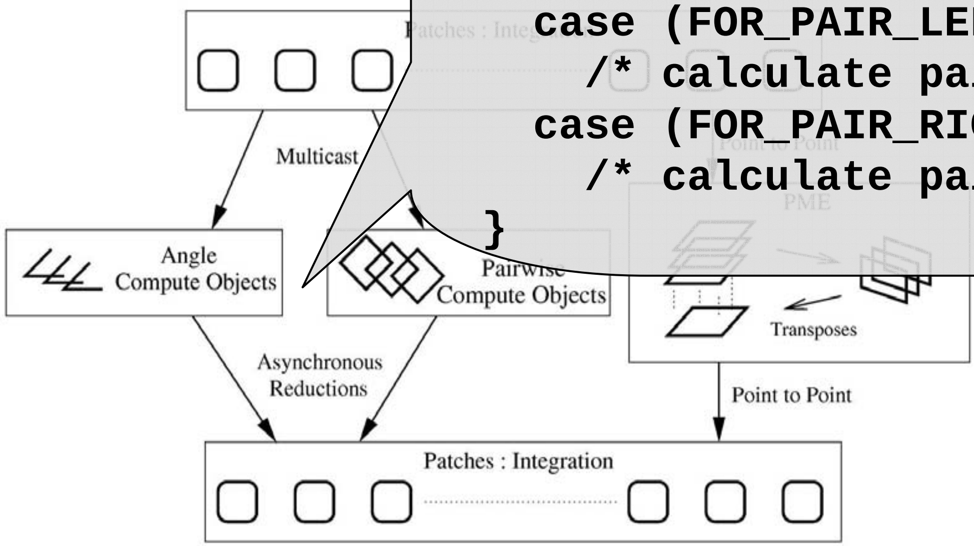


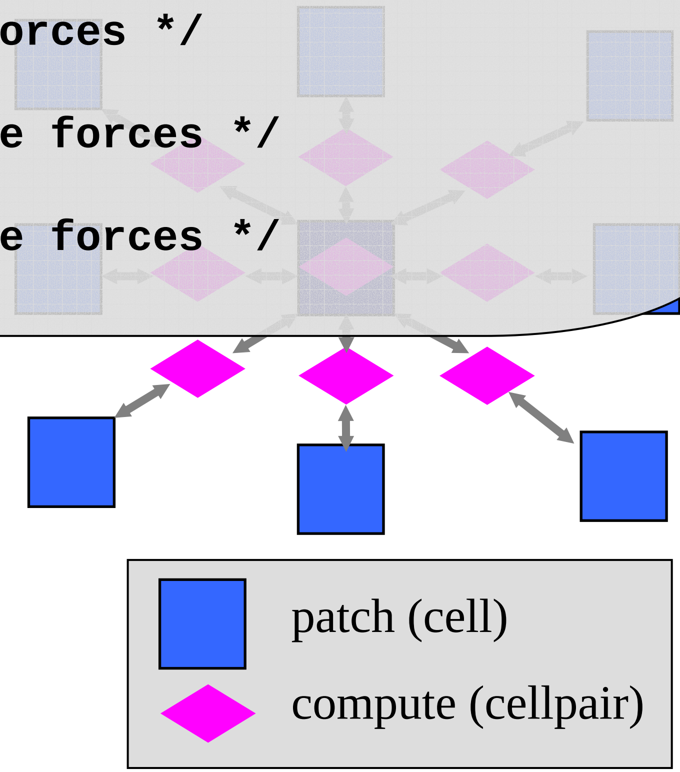


Example: MD

- Structure of a simple MD simulation

```
MPI_Recv(buf,…, MPI_ANY_SOURCE, MPI_ANY_TAG,…);
switch(GET_TYPE(buf)){
  case (FOR_ANGLE):
    /* calculate angle forces */
  case (FOR_PAIR_LEFT):
    /* calculate pairwise forces */
  case (FOR_PAIR_RIGHT):
    /* calculate pairwise forces */
}
```





Expressing Flow of Control

- Charm++: fragmented in object code (Add fig with arrows)

```
MainChare::MainChare{
  cell.sendCoords();
}

MainChare::reduceEnergy(energy){
  totalEnerty+= energy;
  if iter++ < MAX_ITER
    cells.sendCoords();
  else
    CkExit();
}
```

```
Cellpair::recvCoords(coords){
  if not coords from both cells received
    buffer(coords);
    return;
  else      // all coords ready
    force = calcForces();
    for index in 2 cells
      cells(index).recvForces(forces);
}
```

```
Cell::sendCoords(){
  for index in 26 neighbor cellpairs
    cellpairs(index).recvCoords(coords);
}

Cell::recvForces(forces){
  totalforces += forces;
  if not all forces from all cellpairs received
    return;
  else     // neighborhood reduction completed
    integrate();
    mainProxy.reduceEnergy(energy);
}
```

Expressing Flow of Control (2)

- SDag: restricted in an object's life cycle

```
Array [3D] Cell{
    . . .
    entry void runSim(..){
        atomic { init(); }
        for(timeStep = 1 to MAX_ITER){
            atomic { sendCoords(); }
            for(forceCnt = 1 to numForceMsg){
                when recvForces(..){
                    atomic { addForces(..);
                }
            }
            atomic { integrate(); }
        }
    }
};
```

Expressing Flow of Control (2)

- SDag: restricted in an object's life cycle

```
Array [3D] Cell{
    . . .
    entry void runSim(..){
        atomic { init(); }
        for(timeStep = 1 to MAX_ITER){
            atomic { sendCoords(); }
            for(forceCnt = 1 to numForceMsg){
                when recvForces(..){
                    atomic { addForces(..);
                }
            }
            atomic { integrate(); }
        }
    }
};
```

Expressing Flow of Control (2)

- SDag: restricted in an object's life cycle

```
Array [3D] Cell{
    . . .
    entry void runSim(..){
        atomic { init(); }
        for(timeStep = 1 to MAX_ITER){
            atomic { sendCoords(); }
            for(forceCnt = 1 to numForceMsg){
                when recvForces(..){
                    atomic { addForces(..);
                }
            }
            atomic { integrate(); }
        }
    }
};
```

Expressing Flow of Control (3)

- Charisma: global view of control

```
foreach i,j,k in cells
  <coords[i,j,k]> := cells[i,j,k].produceCoords();
end-foreach
for iter := 1 to MAX_ITER
  foreach i1,j1,k1,i2,j2,k2 in cellpairs
    <+cforces[i1,j1,k1],+cforces[i2,j2,k2]> :=
           cellpairs[i1,j1,k1,i2,j2,k2].
           calcForces(coords[i1,j1,k1],coords[i2,j2,k2]);
  end-foreach
  foreach i,j,k in cells
    <coords[i,j,k],+energy> :=
           cells[i,j,k].integrate(cforces[i,j,k]);
  end-foreach
  MDMain.updateEnergy(energy);
end-for
```

Expressing Flow of Control (3)

- Charisma: global view of control

```
foreach i,j,k in cells
  <coords[i,j,k]> := cells[i,j,k].produceCoords();
end-foreach
for iter := 1 to MAX_ITER
  foreach i1,j1,k1,i2,j2,k2 in cellpairs
    <+cforces[i1,j1,k1],+cforces[i2,j2,k2]> :=
           cellpairs[i1,j1,k1,i2,j2,k2].
           calcForces(coords[i1,j1,k1],coords[i2,j2,k2]);
  end-foreach
  foreach i,j,k in cells
    <coords[i,j,k],+energy> :=
           cells[i,j,k].integrate(cforces[i,j,k]);
  end-foreach
  MDMain.updateEnergy(energy);
end-for
```

Expressing Flow of Control (3)

- Charisma: global view of control

```
foreach i,j,k in cells
  <coords[i,j,k]> := cells[i,j,k].produceCoords();
end-foreach
for iter := 1 to MAX_ITER
  foreach i1,j1,k1,i2,j2,k2 in cellpairs
    <+cforces[i1,j1,k1],+cforces[i2,j2,k2]> :=
           cellpairs[i1,j1,k1,i2,j2,k2].
           calcForces(coords[i1,j1,k1],coords[i2,j2,k2]);
  end-foreach
  foreach i,j,k in cells
    <coords[i,j,k],+energy> :=
           cells[i,j,k].integrate(cforces[i,j,k]);
  end-foreach
  MDMain.updateEnergy(energy);
end-for
```

Charisma

- Expressing *global view of control*
  - Parallel constructs in orchestration code
  - Sequential code separately in user C++ code
- Features
  - Object-level parallelism
  - Producer-consumer model
  - Communication patterns
- Implementation
  - Static dependence analysis
  - Generating parallel code, integrating sequential code
  - Output Charm++ program

Charisma (2)

- **`foreach`** Statement

```
foreach i in workers
  workers[i].doWork();
end-foreach
```

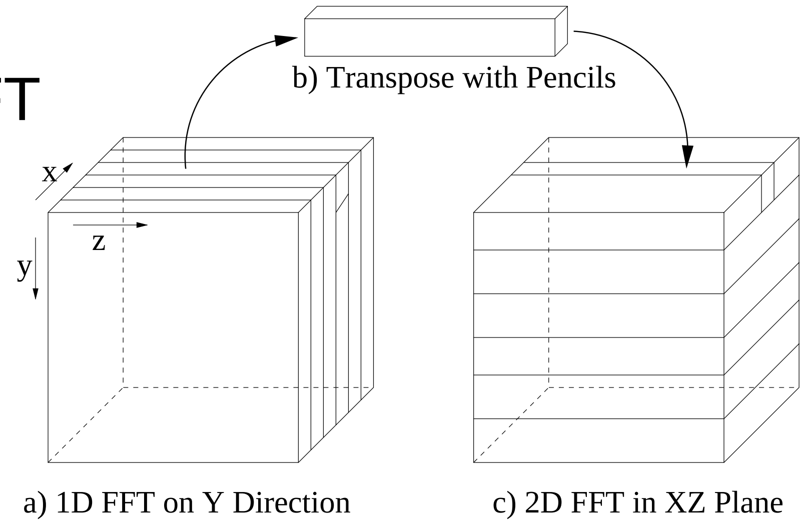
  - Invokes method on all elements: object-level parallelism
- Producer Consumer Mode

```
foreach i in workers
  <p[i]>:=workers[i].foo();
  workers[i].bar(p[i+1]);
end-foreach
```

  - Sequential code unaware of source of input values and destination of output values
  - Data is sent out as soon as it becomes available
- Capable of expressing various communication patterns
  - Point-to-point, broadcast, reduction, multicast, scatter, gather and permutation operations

Charisma (3)

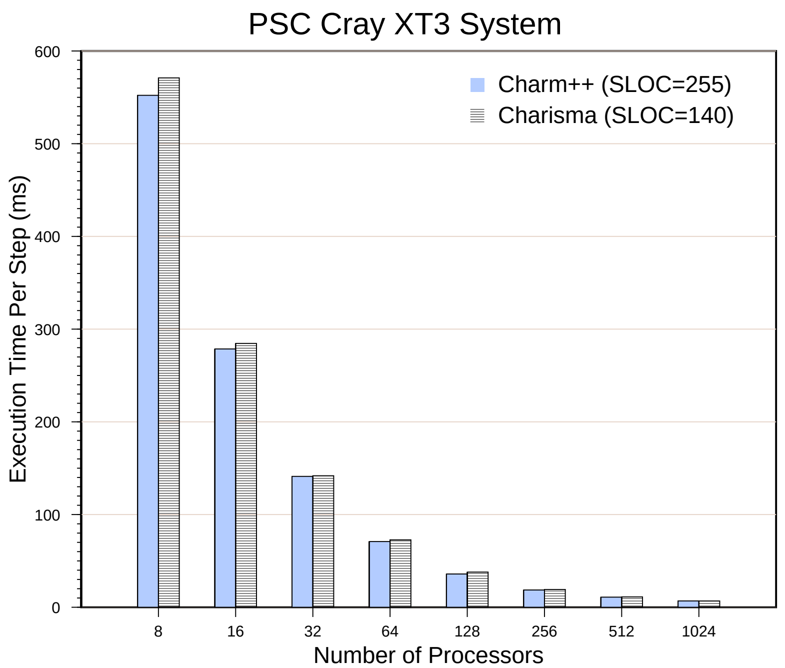
- Example:
  Parallel 3D FFT



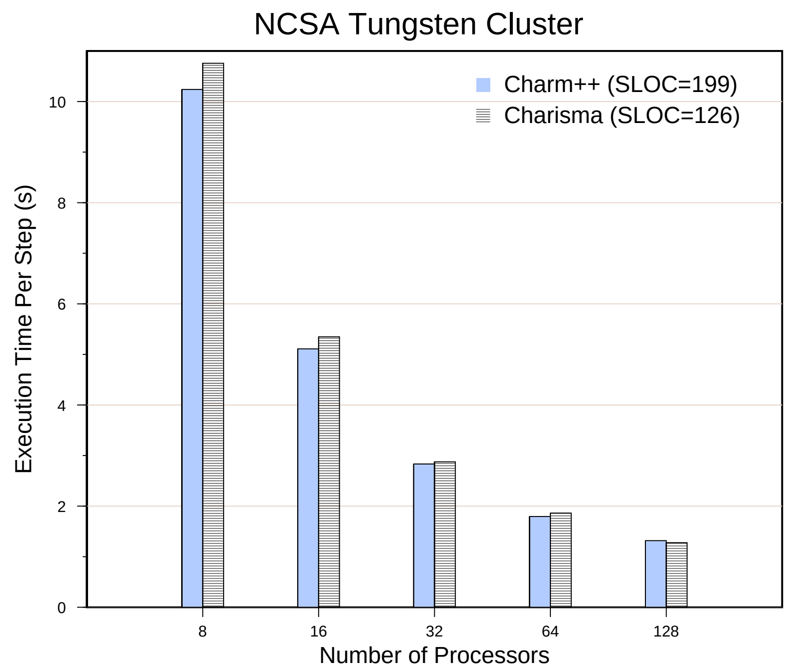

```
foreach x in planes1
 <pencils[x,*]>:=planes1[x].fft1d();
end-foreach
foreach y in planes2
 planes2[y].fft2d(pencils[*,y]);
end-foreach
```

Charisma (4)

- Experiment Results



2D Jacobi
(Size: 16384^2 on 4096 objects)



3D FFT
(Size: 512^3 on 256 objects)

Multiphase Shared Array (MSA)

- Providing *global view of data*
- Features
  - Phases: ReadOnly, WriteByOne, AccumulateOnly
  - Explicit synchronization between phases
- Implementation
  - An object array of pages (virtualized global storage)
  - A per-processor object array managing the local buffer
    - Interface between worker arrays and page array

MSA (2)

- Sample Code:
  - Template instantiation (in .ci file)

```
array [1D] MSA_PageArray<double,DefaultEntry<double>,ENTRIES_PER_PAGE>;
group MSA_CacheGroup<double,DefaultEntry<double>,ENTRIES_PER_PAGE>;
```

  - Declaration and creation

```
MSA1D <...> msa1d = new
        MSA1D <...> (NUM_ENTRIES,NUM_ENROLLERS,LOCAL_CACHE_SIZE);
```

  - Enrolling, Accessing, Synchronizing

```
msa1d.enroll(NUM_ENROLLERS);     // init
double d = msa1d.get(2);         // read
double e = msa2d(i,j);           // read
msa1d.sync();                    // sync (phase change)
msa1d.set(3) = d;                // write
msa1d.sync();                    // sync (phase change)
msa1d.accumulate(i,newValueI);   // accumulate
```

MSA (3)

- Example: ParFUM
  - A global hashtable to store elements on shared edges
  - Partitions contribute elements on a particular edge: accumulate mode
  - Partitions read elements on a shared edge: multiple read mode

Conclusions

- Multi-paradigm helps improving productivity
- Higher-level languages on ARTS are interoperable
- Support multi-paradigm on a common run-time retains performance benefits