



Charisma

A Higher Level Language for Global View of
Control

Chao Huang

[Charisma]

- Providing *global view of control*
- User interface
 - Program consists of orchestration code .or and user sequential code
- Features
 - Foreach: execute in parallel on an object array
 - Inputs and outputs of method invocation: macro dataflow
 - Variety of communication patterns
- Implementation
 - Static dependence analysis
 - Translated into target language (Charm++)

[Charisma (2)]

- Sample Code:

```
program jacobi

class JacobiMain : MainChare;
class JacobiWorker : ChareArray1D;
obj workers : JacobiWorker[10];
param lb : double[100];
param rb : double[100];
/* extern: (double)maxError, (double)epsilon */

begin
  while(maxError > epsilon)
    forall i in workers
      <lb[i], rb[i]> := workers[i].produceBorders();
    end-forall
    forall i in workers
      <+error> := workers[i].compute(lb[i+1], rb[i-1]);
    end-forall
    JacobiMain.updateError(error)
  end-while
end
```

[Expressing Flow of Control]

- Charm++: fragmented in object code

```
MainChare::MainChare{
    cells.init();
}

Cell::init(){
    /* initialization */
    startWork();
}

Cell::startWork(){
    cellPairs.recvData(...);
}

CellPair::recvData(...){
    if data is ready
        calcForces();
}
```

```
CellPair::calcForces(){
    cells.integrate(...);
}

Cell::integrate(...){
    mainProxy.reduceEnergy(...);
}

MainChare::reduceEnergy(...){
    if(timestep < NITER)
        cells.startWork();
    else
        CkExit();
}
```

[Expressing Flow of Control (2)]

- SDag: local in an object's life cycle

```
Array [3D] Cell{
  entry void runSim(...){
    atomic { init(); }
    for(timestep = 1 to maxStep){
      atomic { sendAtomPos(); }
      for(forceCnt = 1 to numForceMsg)
        when recvForces(..){
          atomic { procForce(..); }
        }
    }
    atomic { doIntegration(); }
    if(timestep % outputStep == 0)
      atomic { output(); }
  }
};
```

[Expressing Flow of Control (3)]

- Charisma: global in orchestration code

```
begin
  forall i,j,k in cells
    <coords[i,j,k]> := cells[i,j,k].init();
  end-forall
  for timestep = 1 to maxStep
    forall i,j,k,m,n,p in cellPairs
      <+forces[i,j,k], +forces[l,m,n]> :=
        cellPairs[i,j,k,l,m,n].computeForces(coords[i,j,k],
                                                coords[l,m,n]);
    end-forall
    forall i,j,k in cells
      <coords[i,j,k]> :=
        cells[i,j,k].integrate(forces[i,j,k]);
    end-forall
  end-for
end
```