Using Shared Arrays in Message-Driven Parallel Programs

Phil Miller
Object-Based, Message Driven

- All data owned by objects
  - Locality
  - Virtualization & Migratability
- Communication is explicit & directed
Why Shared Arrays

- Many units accessing common data
- No 'owner'
- Irregular or unpredictable communication
Shared Arrays in Charm++

Client Threads

MSA

Message-driven Objects
Shared Arrays in Charm++
Many applications are 'nice'
- All write, all read, repeat
- Utility?
Mode-Based Safety

- Read-only
- Exclusive Write
- Accumulate
Typed Handles

do {
    double minDistance = distance(r, curSeed);
    for (int i = 0; i < numClusters; ++i) {
        double d = distance(r, i);
        if (d < minDistance) {
            minDistance = d;
            newSeed = i;
        }
    }
    // Start accumulating new positions
    a = r.syncToExcAccum();
    // Each PE adds itself to its new seed
    for (int i = 0; i < numMetrics; ++i)
        a(newSeed, i) += metrics[i];

    // Update membership and change count
    a(newSeed, numMetrics) += 1;
    if (curSeed != newSeed)
        a(0, numMetrics+1) += 1;
    curSeed = newSeed;
}
// Put the array in Read mode
r = a.syncToRead();
} while(r(0, numMetrics+1) > 0);
- Synchronization couples very tightly
- Hurts modularity
- Constrained control flow can hurt performance
- Messaging to the rescue!
void Driver::step() {
    // Phase 1: Atoms to Charge grid
    computes.contributeCharges(Acharges);
direct.dummySync(Acharges);
energy.dummySync(Acharges);
inner[1].syncRestriction(Acharges);
gread Rcharges = Acharges.syncToRead();
    // Phase 2: Charge grid to Potential grid
direct.calculate(Rcharges, Apotentials);
inner[1].calculate(Rcharges, Apotentials);
computes.dummySync(Apotentials);
energy.dummySync(Apotentials);
gread Rpotentials = Apotentials.syncToRead();
    // Phase 2.5: Potential energy
    if (calc_potential_energy)
        energy.calculate(Rcharges, Rpotentials);
computes.dummySync(Rcharges);
Acharges = Rcharges.syncToEAccum();
    // Phase 3: Potential grid to Atoms
computes.readPotentials(Rpotentials);
direct.dummySync(Rpotentials);
inner[1].syncProlongation(Rpotentials);
Apotentials = Rpotentials.syncToEAccum();
}
entry void MSMCompute::step() {
   when interact(ParticleDataMsg *msg),
       contributeCharges(Accum charges) {
       particles = msg;
       computeChargeGrid(charges);
       charges.syncDone();
   }

   when readPotentials(Read potentials) {
       forceMsg = new ParticleForceMsg;
       computeForces(potentials, forceMsg);
       patch.receiveForces(forceMsg);
       potentials.syncDone();
       delete particles;
   }
}

entry void Energy::calculate(Read charges, 
                              Read potentials) {
   double u = 0.0;

   for(int i = 0; i < grid_x[0]; ++i)
       for(int j = 0; j < grid_y[0]; ++j)
           for(int k = 0; k < grid_z[0]; ++k)
               u += charges(i,j,k) * potentials(i,j,k);

   CkPrintf("MSM Potential: %f\n", u);

   charges.syncDone();
   potentials.syncDone();
}
Conclusion

- In progress:
  - QM/MM
  - Migratability

- Questions?