Dynamic Load Balancing

• Object-based decomposition (i.e. virtualized decomposition) helps
  – Allows RTS to remap them to balance load
  – But how does the RTS decide where to map objects?
  – Just move objects away from overloaded processors to underloaded processors
  – How is load determined?
Measurement Based Load Balancing

• **Principle of Persistence**
  – Object communication patterns and computational loads tend to persist over time
  – In spite of dynamic behavior
    • Abrupt but infrequent changes
    • Slow and small changes
  – Recent past is a good predictor of near future

• **Runtime instrumentation**
  – Measures communication volume and computation time

• **Measurement-based load balancers**
  – Measure load information for shares
  – Periodically use the instrumented database to make new decisions and migrate objects
  – Many alternative strategies can use the database
Using the Load Balancer

• Link a LB module
  – -module <strategy>
  – RefineLB, NeighborLB, GreedyCommLB, others
  – EveryLB will include all load balancing strategies

• Compile time option (specify default balancer)
  – -balancer RefineLB

• Runtime option (override default)
  – +balancer RefineLB
Instrumentation

• By default, instrumentation is enabled
  – Automatically collects load information

• Sometimes, you want LB decisions to be based only on a portion of your program
  – To disable by default, provide runtime argument +LB0ff
  – To toggle instrumentation in code, use LBTurnInstrumentOn() and LBTurnInstrumentOff()
Code to Use Load Balancing

• Write PUP method to serialize the state of a chare
• Set `usesAtSync = true;` in chare constructor
• Insert `AtSync()` call at a natural barrier
  – Call from every chare in all collections
  – Does not block
• Implement `ResumeFromSync()` to resume execution
  – A typical `ResumeFromSync()` contributes to a reduction
// Synchronize at every iteration: Main starts next iteration
void Main::endIter() { stencilProxy.sendBoundaries(); }

// Assume a 1D Stencil chare array with near neighbor communication
void Stencil::sendBoundaries() {
    thisProxy(wrap(x-1)).updateGhost(RIGHT, left_ghost);
    thisProxy(wrap(x+1)).updateGhost(LEFT, right_ghost);
}

void Stencil::updateGhost(int dir, double ghost) {
    updateBoundary(dir, ghost);
    if (++remoteCount == 2) {
        remoteCount = 0;
        doWork();
    }
}
Example: Stencil cont.

```cpp
void Stencil::doWork() {
    underThreshold = (computeKernel() < DELTA);

    i = 0;
    pns
    else { contribute(CkCallback(CkReductionTarget(Main, endIter), mainProxy)); } }

    contribute(CkCallback(CkReductionTarget(Main, endIter), mainProxy));
}
```