Dynamic Load Balancing

Dynamic Load Balancing using Objects

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 chares
  - 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
  - `+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 `+LBOff`
  - To toggle instrumention 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 `if (myLBStep) AtSync();` call at natural barrier
  - Does not block
- Implement `ResumeFromSync()` to resume execution
  - A typical `ResumeFromSync()` contributes to a reduction
- Tip: can pass `+LBDebug {verbose level}` at runtime to get debugging output for load balancing

Example: Stencil

```
void sendBoundaries() {
  copyToBoundaries();
  thisProxy(wrapX(x−1),y,z).updateGhosts(i, RIGHT, dimY, dimZ, right); // Assume x, y, z, etc. defined above
  /* ...similar calls to send the 6 boundaries... */
  checkIfBufferComplete(); // See if we've already received neighbors' data
}

void updateGhosts(int i, int d, int w, int h, double b[w*h]) {
  if (i > this->i) { bufferBoundary(d, w, h, b); } // Data for next iteration, so buffer
  else {
    updateBoundary(d, w, h, b);
    if (++remoteCount == 6) { remoteCount = 0; doWork(); }
  } }

void doWork() {
  underThreshold = computeKernel() < DELTA;
  if (++i % 10 == 0) { AtSync(); } // Allow load balancing every 10 iterations
  else { thisProxy(x, y, z).sendBoundaries(); }
}

void ResumeFromSync() {
  if (i % 20 == 0) {
    CkCallback cb(CkReductionTarget(Jacobi, checkConverged), thisProxy);
    contribute(sizeof(int), &underThreshold, CkReduction::logical_and, cb);
  }
  else { thisProxy(x,y,z).sendBoundaries(); }
}

void checkConverged(bool result) {
  if (result) { mainProxy.done(); }
  else { thisProxy(x,y,z).sendBoundaries(); } }
```

How to Diagnose Load Imbalance

- Often hidden in statements such as:
  - Very high synchronization overhead
    - ★ Most processors are waiting at a reduction
- Count total amount of computation (ops/flops) per processor
  - In each phase!
  - Because the balance may change from phase to phase

Golden Rule of Load Balancing

Fallacy: objective of load balancing is to minimize variance in load across processors

Example:

- 50,000 tasks of equal size, 500 processors:
  - ★ A: All processors get 99, except last 5 gets $100 + 99 = 199$
  - ★ OR, B: All processors have 101, except last 5 get 1

Identical variance, but situation A is much worse!

Golden Rule: It is ok if a few processors idle, but avoid having processors that are overloaded with work

Finish time $= max_i$(Time on processor i)

excepting data dependence and communication overhead issues

The speed of any group is the speed of slowest member of that group.

Serialization

Serialization in Charm++

To do load balancing, we move chares to different PEs

- How do we do this for arbitrary objects?
- Charm++ has a framework for serializing data called PUP

PUP

What is PUP?

- **P**ack and **U**n**p**ack
- With PUP, chares become serializable and can be transported to memory, disk, or another processor
- Used in dynamic load balancing framework for object movement

Writing a PUP Routine

```
class MyChare : public CBase_MyChare
{
  int a;
  float b;
  char c;
  float localArray[LOCAL_SIZE];
}
```

```
void pup(PUP::er &p)
{
  CBase_MyChare::pup(p);
  p | a;
  p | b;
  p | c;
  p(localArray, LOCAL_SIZE);
}
```

Writing an Advanced PUP Routine

```
class MyChare : public CBase_MyChare {
  int heapArraySize;
  float* heapArray;
  MyClass* pointer;
}
```

```
void pup(PUP::er &p) {
  CBase_MyChare::pup(p);
  p | heapArraySize;
  if (p.isUnpacking()) { heapArray = new float[heapArraySize]; }
  p(heapArray, heapArraySize);
  bool isNull = !pointer;
  p | isNull;
  if (!isNull) {
    if (p.isUnpacking()) { pointer = new MyClass(); }
    p | *pointer;
  }
}
```

PUP: Applicability

PUP works on:

- A simple type, e.g. char, short, int, long, float, or double
- Any object with a PUP method defined
- STL containers (include `pup_stl.h`)
- Some others, see Section 6 of Charm++ manual for details

PUP Uses

- Moving objects for load balancing
- Marshalling user defined data types
  - When using a type you define as a parameter for an entry method
  - Type has to be serialized to go over network, uses PUP for this
  - Can add PUP to any class, doesn't have to be a chare
- Serializing for storage

Checkpointing

Split Execution

- Can use to stop execution and resume later
  - The job runs for 5 hours, then will continue in new allocation another day!
- We can use PUP for this!
- Instead of migrating to another PE, just "migrate" to disk

How to Enable Split Execution

- Call to checkpoint the application is made in the main chare at a synchronization point
- `log_path` is file system path for checkpoint
- Callback `cb` called when checkpoint (or restart) is done
  - For restart, user needs to provide argument `+restart` and path of checkpoint file at runtime

```
CkCallback cb(CkIndex_Hello::SayHi(), helloProxy);
CkStartCheckpoint(“log_file”, cb);

Shell> ./charmrun hello +p4 +restart log_file
```

Example: Stencil with Checkpointing

In Jacobi **chare**:

```
void startStep() {
  if (iterations % checkpointFreq == 0) { // Do checkpoint
    contribute(CkCallback(CkReductionTarget(Main, checkpoint),
          mainProxy);
  }
  else { // If we're not on a checkpoint iteration, continue
    thisProxy[index].checkpointPhaseDone();
  } }

void checkpointPhaseDone() {
  /* ... do normal stencil boundary exchange and calculation ... */
}
```

In Main **chare**:

```
void checkpoint() {
  CkCallback cb(CkIndex_Jacobi::checkpointPhaseDone(), arrayProxy);
  CkStartCheckpoint(“log_file”, cb);
}
```

Fault Tolerance

- Checkpointing can also be used for fault tolerance
- Makes programs robust against software or hardware faults
  - Becoming more common as process size becomes smaller and chips become more dense
- Can use disk checkpoints for this, but they're slow
- Charm++ can also PUP to memory

Double In-Memory Checkpointing with Automatic Restart

- Can checkpoint data in a buddy processor's memory, in addition to local checkpoint
- System auto detects when node crashes using heartbeat mechanism
- Failed process restarted on a working core, retrieves checkpoint from buddy
- Every other processor uses local checkpoint

Using Double In-Memory Checkpointing with Automatic Restart

- Build Charm++ with `syncft` option on a net based machine layer
- At synchronization point, call from main chare:

```
CkStartMemCheckpoint(CkCallback& cb);
```

- Callback `cb` called when checkpoint or restart is complete
- To test, invoke `CkDieNow()` to mimic failure

Other techniques also exist, more details to come later.