Heterogeneous Task Execution Frameworks in Charm++

Michael Robson
Parallel Programming Lab
Charm Workshop 2016
Charm++ GPU Frameworks

(Map high-level languages and abstractions to asynchronous message-driven execution model)
Accelerator Overview

• NVIDIA GPUs
  – Programmed with CUDA
  – 1,000s of threads
  – 100s GB/s bandwidth
  – ~16 GB of memory
  – ~300 GFLOPS Double Precision
Charm++ GPU Frameworks

(Map high-level languages and abstractions to asynchronous message-driven execution model)
GPU Manager

- Task Offload and Management Library
- Advantages:
  1. Automatic task management and synch.
  2. Overlap data transfer and kernel invocation
  3. Simplified workflow via callbacks
  4. Reduce overhead via centralized management
GPU Manager

- One queue of GPU requests per process
- Utilize pinned memory pools
- Integrated in mainline
- Visualization in projections

http://charm.cs.illinois.edu/manuals/html/libraries/7.html
GPU Manager

Host
Data In
Execute
Data Out

Key
- L: Kernel Launch
- CB: Call Back
- WORK: Useful Work
- Colors rep. diff. kernels
GPU Manager

Key
- L: Kernel Launch
- CB: Call Back
- WORK: Useful Work
- Colors rep. diff. kernels
Using GPU Manager

• Build charm with cuda target
• Create and enqueue a work request
  – Mark/pass buffers
  – Give a callback to resume work
• Write kernel launch functions
nodeGPU Manager

• “Node-level” version of GPU Manager
• One centralized queue per GPU
• Enable GPU applications to run (well) in SMP mode

https://charm.cs.illinois.edu/gerrit/#/c/802/ or branch: mprobson/nodeGPU_ff
nodeGPU Manager Improved API

• Replace globals with functions
• Register kernel launching functions
• Convenience functions for marking buffers
• Build with or without CUDA code
Improved API Example

- enqueue(wrQueue, wr); -> enqueue (wr);
- kernel<<..., kernel_stream>> ->
- kernel<<..., getKernelStream()>>
- dataInfo *info = new dataInfo;
  - info->hostBuffer = hapi_poolMalloc(size);
  - info->size = size;
  - memcpy(info->hostBuffer, data, size);
  - info->bufferID = -1;
  - info->transferToDevice = YES;
  - info->transferFromDevice = NO;
  - info->freeBuffer = YES;
- initBuffer(info, siez, data, true, false, true);
Charm++ GPU Frameworks

(Map high-level languages and abstractions to asynchronous message-driven execution model)
[accel] Framework

• Allow the runtime systems (RTS) to choose to execute on the host or device
• RTS can proactively move needed data
• RTS can map to various platforms
• Originally targeted at cell processor
[accel] Framework

• Builds on top of GPU manager
• Annotate charm entry methods
• Mark data as read, write, persistent, etc
• Automatically generate accelerated code
• Batch fine grained kernel launches

https://charm.cs.illinois.edu/gerrit/#/c/824/ and branch: mprobson/accel-doc
module myModule {

  array [1D] myChareArray {

    // Constructor
    entry myChareArray();

    // Standard Entry Method
    entry void myEntryMethod(
      type passedParameter1, // scalar
      type passedParameter2[passedParameter1], // array w/ length specified
    ...);

    // Accelerated Entry Method
    entry [accel] void myAccelEntryMethod(
      type passedParameter1,
      type passedParameter2[passedParameter1],
    ...)
    |
    modifier : type localParameter1 <impl_obj->memberVariable1>,
    modifier : type localParameter2[localParameter1] <impl_obj->memberVariable2>,
    ...)
  } {
    // ... Function body code goes here ... //

  } callback_function_name;

  // ... other entry method declarations for this array here ... //

};

// ... other chare, chare array, group, etc. declarations here ... //
[accel] Framework Example

```c
entry [triggered splittable(matrixSize) threadsperblock(192) accel] void beginWork(int matrixSize)
{
    readonly : float A[matrixSize*matrixSize]<impl_obj->A>,
    readonly : float B[matrixSize*matrixSize]<impl_obj->B>,
    writeonly : float C[matrixSize*matrixSize]<impl_obj->C>
}
    for (int i=splitIndex; i<matrixSize; i+=numSplits) {
        for (int j=0; j<matrixSize; j++) {
            C[i*matrixSize + j] = 0;
            for (int k=0; k<matrixSize; k++) {
                C[i*matrixSize + j]+= A[i*matrixSize +k] * B[k * matrixSize + j];
            }
        }
    }
}complete;
```
[accel] Framework Usage

- **modifiers:**
  - read-only, write-only, read-write
  - shared – one copy per batch
  - persist – resident in device memory

- **parameters:**
  - triggered – one invocation per chare in array
  - splittable (int) – AEM does part of work
  - threadsPerBlock (int) – specify block size
$version

• Allow users to write platform specific accelerator code
• Either as two separate, equivalent kernels
• Or machine specific sections/tweaks
• Automatically generate multiple kernels

https://charm.cs.illinois.edu/gerrit/#/c/1104/
$version Target Specific

```c
entry [accel] void myEntry() [
    writeonly : vec_t myVec[
        VEC_SIZE]<impl_obj->myVec>]
    {
        $version cpu
            cpu_t myVec;
        $version cuda
            cuda_t myVec;
        $version end
            for(int i = 0; i < myVec.size();
                i++)
                myVec[i] *= 3;
    }
```
$version Two Implementations

```c
entry [accel] void myEntry2(vec_t myVec) [readwrite : vec_t myVec [VEC_SIZE]<impl_obj->myVec>] {
  $version cpu
  for (int i = 0; i < myVec.size(); i++)
    myVec[i] *= 3;
  $version cuda
    myVec[threadIdx.x] *= 3;
  $version end
}
```
Charm++ GPU Frameworks

(Map high-level languages and abstractions to asynchronous message-driven execution model)
NAMD GPU Acceleration

- NAMD GPU code is about 5x faster than the CPU code
  - CPU version is becoming somewhat obsolete
- General requirements
  - Keep data on device as much as possible
  - Use pinned host memory
  - Hide CUDA kernel launch latency
    - Merge all computation into few kernels
    - Avoid unnecessary cudaStreamSynchronize()
NAMD GPU Performance

- Explicit solvent: 30% - 57% faster simulations

Speedup vs. NAMD 2.11

- DHFR (24K atoms)
- ApoA1 (92K atoms)
NAMD GPU Performance

- GB implicit solvent: Up to 3.5x faster simulations
NAMD PME computation – case for direct GPU-GPU communication

• Particle Mesh Ewald (PME) reciprocal computation requires a 3D FFT, which in turn requires repeated communications between GPUs
• Communication is the bottleneck
• In the current implementation, we must handle intra- and inter-node cases separately
Intra-node

• Sending PE

transposeDataOnGPU(d_data, stream); // Transpose data locally
copyDataToPeerDevice(destGPU, d_data, stream); // Copy data to GPU on same node
cudaStreamSynchronize(stream); // Wait for CUDA stream to finish

PmeMsg* msg = new (0) PmeMsg(); // Allocate empty message
pmePencil.recvData(msg); // Send message to PE that has “destGPU”

• Receiving PE

void recvData(PmeMsg* msg) {
  // Receiving empty message lets PE
  // know its GPU now has the data in “d_data”
  fftWork(d_data, stream);
  // Perform work on data
  ...
}

• Requires lots of tedious work from the user
• Error prone
Inter-node

• Sending PE

transposeDataOnGPU(d_data, stream); // Transpose data locally
PmeMsg* msg = new (dataSize) PmeMsg(); // Create message
copyDataToHost(d_data, msg->data, stream); // Copy data to host
cudaStreamSynchronize(stream); // Wait for CUDA stream to finish
pmePencil.recvData(msg); // Send data to PE on different node

• Receiving PE

void recvData(PmeMsg* msg) {
    copyDataToDevice(msg->data, d_data, stream); // Copy data to device buffer d_data
cudaStreamSynchronize(stream); // Wait for CUDA stream to finish
fftWork(d_data, stream); // Perform work on data
....
}

• Stalls PE at cudaStreamSynchronize()
• Host buffer is non-pinned, slow memcopy
How it could be

• **Sending PE**

```cpp
PmeMsg* msg = new (dataSize) PmeMsg();
transposeDataOnGPU(msg->data, stream);
pmePencil.recvData(msg, stream);
```

// Create message, **data on device**
// Transpose data locally
// Send data using CUDA stream

• **Receiving PE**

```cpp
void recvData(PmeMsg* msg) {
  fftWork(msg->data, stream);
  // Perform work on data
  ... ...
}
```

• Details hidden from user

• Works seamlessly on any node configuration
High message latency

- On idle nodes high message latency observed
ChaNGa

- Cosmological N-body simulations
- Leverages nodeGPU and GPU Manager
- Offloads gravity kernels
- Active work in optimization
ChaNGa Performance

![Graph showing performance comparison]

- CPU-SMP
- GPU-SMP
- GPU-SMP-Wang

Relative time per step (s)

<table>
<thead>
<tr>
<th>θ (opening angle)</th>
<th>CPU-SMP</th>
<th>GPU-SMP</th>
<th>GPU-SMP-Wang</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>946.26</td>
<td>526.60</td>
<td>96.48</td>
</tr>
<tr>
<td>0.2</td>
<td>30.59</td>
<td>16.48</td>
<td>6.82</td>
</tr>
<tr>
<td>0.3</td>
<td>6.82</td>
<td>4.10</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>2x</td>
<td>4x</td>
<td></td>
</tr>
</tbody>
</table>

The graph shows the relative time per step for different θ values and hardware configurations.
ChaNGa Performance

![Bar chart showing ChaNGa dwf1 performance on 4 BlueWaters XK Nodes. The chart compares execution time per step without CUDA and with CUDA.]
Charm++ GPU Frameworks
Heterogeneous Load Balancing

- Automatically overlap useful work between CPU and GPU
- Based on various parameters:
  - Idle time, latency, load
- Exists in accel branch currently
GPU Thread

• Much like today’s comm-thread
• Spawn threads per-node equal to GPUs
• Part of a larger threads project
  – Comm threads
  – GPU threads
  – Drone threads
  – Worker threads
QUESTIONS?

Michael Robson
mprobson@illinois.edu
Accelerator Overview

- Intel Xeon Phi
  - Programmed using icc -mmic
  - ~60 modified Pentiums
  - 4 hardware threads
  - 512-bit vectors
  - ~300 GB/S bandwidth
  - ~1 TFLOPS (Double Precision)
Steps to Get Xeon Phi Working

• Build two (almost) identical versions of charm
  – Regular and passing -mmic option
• Modify makefile to build two binaries, mic ending in .mic
• Properly configure nodelist
  – ++cpus aka nodesize
  – repeated for each node
  – ++ext .mic
• On Stampede:
  – ++usehostname
  – -bro
  – -mico
• Run! branch: mprobson/mic-fix