Work is *overdecomposed* into medium-sized grains

- Fine-grain task parallelism
- Sized well for the CPU
  - Overlap of communication and computation
- GPUs rely on massive data-parallelism
  - Fine grains decrease performance
  - Each kernel instantiation has substantial overhead

To reduce overhead

- Combine fine-grain work units for the GPU
- Delay may be insignificant if the work is low priority
Terminology

- **Agglomeration**—composition of distinct work units
- **Static agglomeration**—fixed number of work units are agglomerated
- **Dynamic agglomeration**—number of work units agglomerated varies at runtime
Dynamic Scheduling for Work Agglomeration on Heterogeneous Clusters

Jonathan Lifflander (UIUC)
Implementation

- Charm++
  - Work is decomposed into objects
    - With affinity to data
    - Represents multiple tasks
    - Each task is a method of an object
    - Remote invocation occurs when a message arrives (the data is the parameters)
  - Each object lives on a processor
  - Each processor has many objects on it
  - Sets of objects that perform the same type of work are organized into arrays
Charm++ Scheduling

- Each processor has a scheduler
  - Arrival messages are put in a queue
  - They are prioritized based on priority set by the sender
  - Execution is in that order based on the current queue state
Dynamic Scheduling for Work Agglomeration on Heterogeneous Clusters

Jonathan Lifflander (UIUC)

7/26
Agglomeration API

scheduleWork()

Agglomeration

Accelerator FIFO

agglomerateWork()

Work Unit

Agglomeration

Accelerator
Programmer/Runtime Division

- **Programmer**
  - Writes GPU kernel for agglomeration
  - Creates an *offset array*
    - Each task’s input might be a different size
    - Store the offset of each task’s beginning and ending index in the contiguous data arrays

- **System**
  - Decide what work to execute and when
Dynamic Scheduling for Work Agglomeration on Heterogeneous Clusters

Jonathan Lifflander (UIUC)
Low-priority agglomeration message

Higher priority GPU messages

Low-priority agglomeration message
Dynamic Agglomeration

- Uses the following heuristic
  - If the “accelerator FIFO” reaches a size limit, work is agglomerated
    - Typically set based on memory limitations
  - Else enqueue a low priority message that causes agglomeration
    - When higher-priority work is being generated, it goes into the FIFO
    - When it lets up, work is agglomerated
    - Since low priority work is assumed, not agglomerating aggressively should not reduce performance
Experimental Setup

- NCSA AC Cluster
  - Two dual-core 2.4 GHz AMD Opterons
    - 8 GB of memory
  - NVIDIA Tesla S1070 with four GPUs
    - Each with 4 GB of memory
    - CUDA
Application: Molecular2D
Molecular2D

- Work is decomposed into:
  - **Cells**: 2D array of objects
    - Spatially decomposed
    - Each holds a set of particles
    - They interact with the neighboring cells
    - The cell holds the current particle position and updates these based on calculated forces
  - **Interactions**: 4D array of objects
    - Each interacts two particle sets
    - Bulk of the work

- Using the GPU
  - Cells on CPU
  - Interactions on GPU
  - When an interaction receives the two particles sets, it calls `scheduleWork`
Molecular 2D Interaction Kernel

```c
__global__ void interact(...) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;

    // For loop added for agglomeration
    for(int j = start[i]; j < end[i]; j++)
        // interaction work
}
```
Dynamic Scheduling for Work Agglomeration on Heterogeneous Clusters

Jonathan Lifflander (UIUC)
Dynamic Scheduling for Work Agglomeration on Heterogeneous Clusters

∥

Jonathan Lifflander (UIUC) ∥ 19/26
Application: LU Factorization without pivoting
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{1,1}$</td>
<td>$A_{1,2}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_{2,1}$</td>
<td></td>
<td>$A_{2,2}$</td>
<td></td>
</tr>
</tbody>
</table>
LU Factorization

- CPU
  - Diagonal factorization
  - Triangular solves
- GPU
  - Matrix-matrix multiples (DGEMMs)
Dynamic Scheduling for Work Agglomeration on Heterogeneous Clusters

Jonathan Lifflander (UIUC)
Conclusion

- For both benchmarks, agglomerating work increases performance
- Agglomeration does not need to be application-specific
- Statically selecting work units to agglomerate is difficult and may reduce performance
- Runtimes can agglomerate automatically
  - An agglomerating kernel still must be written
  - Obtains better performance than static