Design Space Exploration of Memory Model for Heterogeneous Computing

Jieun Lim* and Hyesoon Kim

6/16/2012

*Seoul National University
Introduction

Heterogeneous computing has become a major architecture trend

How to design memory system
- Strongly coupled with architecture design and programming model
- Difficult to compare models

Goal
- Understand a trade-off in memory system design decisions
- Evaluate the overhead of design options
Existing Memory Address Space Design Options

Unified space: identical address space both for CPU and GPU
- No explicit data transfer, but complicated TLB/MMU designs

Disjoint space
- Scalable and easy to implement, but need explicit data transfer

Partially-shared: only part of the space is shared
- Convenience of using shared memory, but overhead of managing between spaces

ADSM: one PU can access the entire memory, but the other cannot
- Provide a shared space with discrete memories
Heterogeneous Architecture Summary

<table>
<thead>
<tr>
<th>scheme</th>
<th>address space</th>
<th>Connection</th>
<th>coherence</th>
<th>how to use shared data</th>
<th>consistency</th>
<th>synchronization</th>
<th>Locality</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU+CUDA* [28]</td>
<td>disjoint</td>
<td>PCI-E</td>
<td>-</td>
<td>NA</td>
<td>weak consistency</td>
<td>-</td>
<td>impl-pri-expl-pri</td>
</tr>
<tr>
<td>EXOCHI [32]</td>
<td>unified</td>
<td>Memory controller</td>
<td>can be coherent</td>
<td>CHI runtime API</td>
<td>weak consistency</td>
<td>unknown</td>
<td>impl-pri</td>
</tr>
<tr>
<td>CPU+LRB [30]</td>
<td>partially shared</td>
<td>PCI-E</td>
<td>coherent only in LRB/CPU</td>
<td>type qualifier, ownership</td>
<td>weak consistency</td>
<td>APIs</td>
<td>impl-pri</td>
</tr>
<tr>
<td>COMIC [21]</td>
<td>unified</td>
<td>interconnection</td>
<td>directory</td>
<td>COMIC API functions</td>
<td>centralized release consistency</td>
<td>barrier function</td>
<td>expl-pri-impl-pri-impl-shared</td>
</tr>
<tr>
<td>Rigel [18]</td>
<td>unified</td>
<td>interconnection</td>
<td>HW/SW</td>
<td>global memory operation</td>
<td>weak consistency</td>
<td>implicit barrier/Rigel LPI</td>
<td>expl</td>
</tr>
<tr>
<td>GMAC [9]</td>
<td>ADSM</td>
<td>PCI-E</td>
<td>GMAC protocol</td>
<td>global memory operation</td>
<td>weak consistency</td>
<td>sync API</td>
<td>expl-private-impl-shared</td>
</tr>
<tr>
<td>Sandy Bridge [14]</td>
<td>disjoint</td>
<td>Memory controller</td>
<td>-</td>
<td>-</td>
<td>weak consistency</td>
<td>-</td>
<td>impl-priv-exp-priv</td>
</tr>
<tr>
<td>Fusion [2]</td>
<td>disjoint</td>
<td>Memory controller</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CUBA [8]</td>
<td>disjoint</td>
<td>BUS</td>
<td>-</td>
<td>direct access to local storage</td>
<td>weak consistency</td>
<td>-</td>
<td>exp-priv</td>
</tr>
<tr>
<td>CUDA 4.0</td>
<td>unified</td>
<td>-</td>
<td>explicit copy</td>
<td>-</td>
<td>weak consistency</td>
<td>-</td>
<td>exp-priv</td>
</tr>
<tr>
<td>OpenCL</td>
<td>unified</td>
<td>-</td>
<td>explicit copy</td>
<td>-</td>
<td>weak consistency</td>
<td>-</td>
<td>exp-priv</td>
</tr>
</tbody>
</table>

None of the heterogeneous computing system has employed a unified, fully-coherent, strong-consistent memory system yet

Most proposed/existing systems have disjoint memory systems
Code Example for Partially Shared Memory

Reduction example

Show the differences of using different `malloc` and ownership changes for objects in the shared space

```c
int addGPUTwoVectors(shared int *a,
                         shared int *b,
                         shared int *c)
{
    acquireOwnership(a, b, c);
    for (i=1 to 64) {
        c[i] = a[i] + b[i];
    }
    releaseOwnership(a, b, c);
}

int kernel(...)
{
    // allocate in shared region
    int *a = sharedmalloc(...);
    int *b = sharedmalloc(...);
    int *c = sharedmalloc(...);
    int *d = malloc(...);
    int *e = malloc(...);
    int *f = malloc(...);

    for (i=1; i < 64; i++) // initialize
    {
        // initialize a, b, d, e
        releaseOwnership(a, b, c);
        addGPUTwoVectors(a, b, c); // c = a+b in GPU
        addTwoVectors(c, d, e); // f = d+e in CPU
        acquireOwnership(c);
        addTwoVectors(c, d, e); // f = c+f in CPU
        ...
    }
}
```

Ownership control

Special `malloc` function

Partially-shared memory space
Programmability vs. Memory Address Space

<table>
<thead>
<tr>
<th></th>
<th>Comp</th>
<th>UNI</th>
<th>PAS</th>
<th>DIS</th>
<th>ADSM</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix mul</td>
<td>39</td>
<td>0</td>
<td>2</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>merge sort</td>
<td>112</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>dct</td>
<td>410</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>reduction</td>
<td>142</td>
<td>0</td>
<td>2</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>convolution</td>
<td>75</td>
<td>0</td>
<td>4</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>k-mean</td>
<td>332</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

The number of source lines to handle data communication

Different programming options affect how easy/difficult it is to write programs

Use the number of source lines to indicate programmability

The number of additional source lines that are required to handle explicit data communication and data handling operations

Unified < partially-shared <= ADSM < disjoint

Unified space does not require any special APIs
Disjoint memory space requires the most additional source code lines
Conclusion

We exploited the design space of heterogeneous computing memory systems

memory space does not affect performance significantly

Partially shared memory space is the most promising option

- provides many hardware design options and moderately good programmability