LEAP Shared Memories: Automating the Construction of FPGA Coherent Memories

Hsin-Jung Yang†, Kermin E. Fleming‡, Michael Adler‡, and Joel Emer†‡

† Massachusetts Institute of Technology
‡ Intel Corporation

May 12th, FCCM 2014
Motivation

- Goal: simplifying parallel programming on FPGAs
Motivation

• Goal: simplifying parallel programming on FPGAs
• 2D Heat Transfer Equation
Motivation

- **Goal:** simplifying parallel programming on FPGAs
- **2D Heat Transfer Equation**

```cpp
for(int t = 0; t < T; t++){
    #pragma omp parallel num_threads(4){
        int thread_id = omp_get_thread_num();
        int bid_x = thread_id%2;
        int bid_y = thread_id/2;
        for (int y = bid_y*(N/2); y < (1+bid_y)*(N/2); y++)
            for (int x = bid_x*(M/2); x < (1+bid_x)*(M/2); x++)
                U[t+1,x,y] = C0*U[t,x,y] + Cx*(U[t,x-1,y]+U[t,x+1,y])
                + Cy*(U[t,x,y-1]+U[t,x,y+1]);
    }
}
```
Motivation

- Goal: simplifying parallel programming on FPGAs
- 2D Heat Transfer Equation

```c
for(int t = 0; t < T; t++){
    #pragma omp parallel num_threads(4)
    {
        int thread_id = omp_get_thread_num();
        int bid_x = thread_id%2;
        int bid_y = thread_id/2;
        for (int y = bid_y*(N/2); y < (1+bid_y)*(N/2); y++)
            for (int x = bid_x*(M/2); x < (1+bid_x)*(M/2); x++)
                U[t+1,x,y] = C0*U[t,x,y]
                + Cx*(U[t,x-1,y]+U[t,x+1,y])
                + Cy*(U[t,x,y-1]+U[t,x,y+1]);
    }
}
```

operation on the shared array
Motivation

• Goal: simplifying parallel programming on FPGAs
• 2D Heat Transfer Equation

```c
for(int t = 0; t < T; t++){
    #pragma omp parallel num_threads(4){
        int thread_id = omp_get_thread_num();
        int bid_x = thread_id%2;
        int bid_y = thread_id/2;
        for (int y = bid_y*(N/2); y < (1+bid_y)*(N/2); y++)
            for (int x = bid_x*(M/2); x < (1+bid_x)*(M/2); x++)
                U[t+1,x,y] = C0*U[t,x,y]
                    + Cx*(U[t,x-1,y]+U[t,x+1,y])
                    + Cy*(U[t,x,y-1]+U[t,x,y+1]);
    }
}
```

operation on the shared array

implicit barrier synchronization
Motivation

- Goal: simplifying parallel programming on FPGAs
- 2D Heat Transfer Equation

```
for (int t = 0; t < T; t++) {
    #pragma omp parallel num_threads(4)
    {
        int thread_id = omp_get_thread_num();
        int bid_x = thread_id % 2;
        int bid_y = thread_id / 2;
        for (int y = bid_y * (N/2); y < ((1+bid_y) * (N/2); y++)
            for (int x = bid_x * (M/2); x < ((1+bid_x) * (M/2); x++)
                U[t+1, x, y] = C0 * U[t, x, y]
                    + Cx * (U[t,x+1, y] + U[t,x-1, y])
                    + Cy * (U[t,x, y+1] + U[t,x, y-1]);
    }
}
```

General-Purpose Processor

- User Application
- Software Libraries & Operating System (Thread Creation, Synchronization, Memory Management ... etc.)
- System layer
- Hardware

N

M
Motivation

• Goal: simplifying parallel programming on FPGAs
• 2D Heat Transfer Equation

How to implement on FPGAs?
Programming on FPGA

• 2D Heat Transfer Equation (using FPGA Block RAM)

\[
\begin{align*}
\text{for } (\text{int } t = 0; t < T; t++) & \\
& \text{for } (\text{int } y = 0; y < N; y++) & \\
& \quad \text{for } (\text{int } x = 0; x < M; x++) & \\
U[t+1,x,y] &= C0*U[t,x,y] \\
& + Cx*(U[t,x-1,y]+U[t,x+1,y]) \\
& + Cy*(U[t,x,y-1]+U[t,x,y+1]);
\end{align*}
\]
Programming on FPGA

• 2D Heat Transfer Equation (using FPGA Block RAM)

for (int t = 0; t < T; t++)
    for (int y = 0; y < N; y++)
        for (int x = 0; x < M; x++)
            U[t+1,x,y] = C0*U[t,x,y] + Cx*(U[t,x-1,y]+U[t,x+1,y]) + Cy*(U[t,x,y-1]+U[t,x,y+1]);

interface MEM_IFC#(type t_ADDR, type t_DATA);
    method void readReq (t_ADDR addr);
    method void write(t_ADDR addr, t_DATA data);
    method t_DATA readResp();
endinterface
Programming on FPGA

- **2D Heat Transfer Equation (using FPGA Block RAM)**

```java
interface MEM_IFC#(type t_ADDR, type t_DATA);
    method void readReq(t_ADDR addr);
    method void write(t_ADDR addr, t_DATA data);
    method t_DATA readResp();
endinterface

MEM_IFC#(Bit#(5), Bit#(32)) memory <- mkBRAM();
...
memory.readReq(a1);
Bit#(32) d1 <- memory.readResp();
memory.write(a2, d2);
...
```

```java
for(int t = 0; t < T; t++)
    for (int y = 0; y < N; y++)
        for (int x = 0; x < M; x++)
            U[t+1,x,y] = C0*U[t,x,y]
            + Cx*(U[t,x-1,y]+U[t,x+1,y])
            + Cy*(U[t,x,y-1]+U[t,x,y+1]);
```
Programming on FPGA

• 2D Heat Transfer Equation (using FPGA Block RAM)

Difficulty:
• Problem size cannot fit in RAM block

```java
for(int t = 0; t < T; t++)
    for (int y = 0; y < N; y++)
        for (int x = 0; x < M; x++)
            U[t+1,x,y] = CO*U[t,x,y]
            + Cx*(U[t,x-1,y]+U[t,x+1,y])
            + Cy*(U[t,x,y-1]+U[t,x,y+1]);
```

interface MEM_IFC#(type t_ADDR, type t_DATA);
    method void readReq(t_ADDR addr);
    method void write(t_ADDR addr, t_DATA data);
    method t_DATA readResp();
endinterface

MEM_IFC#(Bit#(5), Bit#(32)) memory <- mkBRAM();
....
memory.readReq(a1);
Bit#(32) d1 <- memory.readResp();
memory.write(a2, d2);
....
Programming on FPGA

- 2D Heat Transfer Equation (using LEAP Scratchpad)

interface MEM_IFC#(type t_ADDR, type t_DATA);
  method void readReq (t_ADDR addr);
  method void write(t_ADDR addr, t_DATA data);
  method t_DATA readResp();
endinterface

unlimited address space

MEM_IFC#(Bit#(28), Bit#(32)) memory <- mkScratchpad();
....
memory.readReq(a1);
Bit#(32) d1 <- memory.readResp();
memory.write(a2, d2);
....

Programming on FPGA

- 2D Heat Transfer Equation (using LEAP Scratchpad)

Programming on FPGA

- 2D Heat Transfer Equation (using LEAP Scratchpad)

Difficulty:
- Single engine is too slow


interface MEM_IFC#(type t_ADDR, type t_DATA);
   method void readReq (t_ADDR addr);
   method void write(t_ADDR addr, t_DATA data);
   method t_DATA readResp();
endinterface

unlimited address space

MEM_IFC#(Bit#(28), Bit#(32)) memory <- mkScratchpad();
....
memory.readReq(a1);
Bit#(32) d1 <- memory.readResp();
memory.write(a2, d2);
....
Parallel Programming on FPGA

- 2D Heat Transfer Equation

```
interface MEM_IFC{
  type t_ADDR, type t_DATA;
  method void readReq(t_ADDR addr);
  method void write(t_ADDR addr, t_DATA data);
  method t_DATA readResp();
}
endinterface
```
Parallel Programming on FPGA

- 2D Heat Transfer Equation

**Difficulty:** Performance is limited

interface MEM_IFC#(type t_ADDR, type t_DATA);
method void readReq(t_ADDR addr);
method void write(t_ADDR addr, t_DATA data);
method t_DATA readResp();
endinterface
Parallel Programming on FPGA

• 2D Heat Transfer Equation

Difficult: Performance is limited

.Serialized requests
Parallel Programming on FPGA

- **2D Heat Transfer Equation**

**Difficultly:** Performance is limited

- 🙁 Serialized requests
- 😞 Long latency if across FPGAs

```java
interface MEM_IFC{
    type t_ADDR, type t_DATA;
    method void readReq(t_ADDR addr);
    method void write(t_ADDR addr, t_DATA data);
    method t_DATA readResp();
} endinterface
```
Parallel Programming on FPGA

- 2D Heat Transfer Equation

```
interface MEM_IFC{
    type t_ADDR, type t_DATA;
    method void readReq(t_ADDR addr);
    method void write(t_ADDR addr, t_DATA data);
    method t_DATA readResp();
}endinterface
```

![Diagram of parallel programming on FPGA with 2D heat transfer equation](image)
Parallel Programming on FPGA

- 2D Heat Transfer Equation

Difficulty:
- Edge pixels are shared

```java
interface MEM_IFC{
    type t_ADDR, type t_DATA;
    method void readReq(t_ADDR addr);
    method void write(t_ADDR addr, t_DATA data);
    method t_DATA readResp();
} endinterface
```
Parallel Programming on FPGA

- 2D Heat Transfer Equation

Difficulty:
- Edge pixels are shared

Need cache coherence!
Shared Memory Services:
Coherent Scratchpad (CS)

(1) Ordering point

Ring-based snoopy protocol

pre-order request
ordered request
response
Shared Memory Services: Coherent Scratchpad (CS)

1. Ordering point

Ring-based snoopy protocol
Modified MOSI protocol

- Coherent Cache
- CS Interface
- Coherent Scratchpad Client

- Engine
- Interface
Shared Memory Services: Coherent Scratchpad (CS)

(1) Ordering point
(2) Store data
Shared Memory Services: Coherent Scratchpad (CS)

- Ring-based snoopy protocol
- Modified MOSI protocol

1. Ordering point
2. Store data
3. Store owner-bit information for every address
Shared Memory Services: Coherent Scratchpad (CS)

Ring-based snoopy protocol
Modified MOSI protocol

(1) Ordering point
(2) Store data
(3) Store owner-bit information
Shared Memory Services: Coherent Scratchpad
Parallel Programming on FPGA

• 2D Heat Transfer Equation

```c
for(int t = 0; t < T; t++){
    #pragma omp parallel num_threads(4){
        int thread_id = omp_get_thread_num();
        int bid_x = thread_id%2;
        int bid_y = thread_id/2;
        for (int y = bid_y*(N/2); y < (1+bid_y)*(N/2); y++)
            for (int x = bid_x*(M/2); x < (1+bid_x)*(M/2); x++)
                U[t+1,x,y] = C0*U[t,x,y]
                    + Cx*(U[t,x-1,y]+U[t,x+1,y])
                    + Cy*(U[t,x,y-1]+U[t,x,y+1]);
    }
}
implicit barrier synchronization
```
Parallel Programming on FPGA

• 2D Heat Transfer Equation

```c
for(int t = 0; t < T; t++){
    #pragma omp parallel num_threads(4){
        int thread_id = omp_get_thread_num();
        int bid_x = thread_id%2;
        int bid_y = thread_id/2;
        for (int y = bid_y*(N/2); y < (1+bid_y)*(N/2); y++)
            for (int x = bid x*(M/2); x < (1+bid x)*(M/2); x++)
                U[t+1,x,y] = C0*U[t,x,y] + Cx*(U[t,x-1,y]+U[t,x+1,y]) + Cy*(U[t,x,y-1]+U[t,x,y+1]);
    }
}
```

operation on the shared array

implicit barrier synchronization
Parallel Programming on FPGA

- 2D Heat Transfer Equation

```c
for(int t = 0; t < T; t++){
    #pragma omp parallel num_threads(4){
        int thread_id = omp_get_thread_num();
        int bid_x = thread_id%2;
        int bid_y = thread_id/2;
        for (int y = bid_y*(N/2); y < (1+bid_y)*(N/2); y++)
            for (int x = bid_x*(M/2); x < (1+bid_x)*(M/2); x++)
                U[t+1,x,y] = C0*U[t,x,y]
                + Cx*(U[t,x-1,y]+U[t,x+1,y])
                + Cy*(U[t,x,y-1]+U[t,x,y+1]);
    }
}
```

- Operation on the shared array
- Implicit barrier synchronization

- Finish the inner loop operations
  - Computations complete
  - Memory operations complete
- Wait until all threads are finished
**Shared Memory Services: Memory Consistency**

• **Block RAM/Private Scratchpad Interface**

```markdown
interface MEM_IFC#(type t_ADDR, type t_DATA);
    method void readReq (t_ADDR addr);
    method void write(t_ADDR addr, t_DATA data);
    method t_DATA readResp();
endinterface
```

• **Coherent Scratchpad Interface**

```markdown
interface MEM_IFC#(type t_ADDR, type t_DATA);
    method void readReq (t_ADDR addr);
    method void write(t_ADDR addr, t_DATA data);
    method t_DATA readResp();
    // t_REQ r := {READ, WRITE, FULL}
    method Bool requestPending(t_REQ r);
endinterface
```
Shared Memory Services: Memory Consistency

- Block RAM/Private Scratchpad Interface

```plaintext
interface MEM_IFC#: (type t_ADDR, type t_DATA);
    method void readReq (t_ADDR addr);
    method void write(t_ADDR addr, t_DATA data);
    method t_DATA readResp();
endinterface
```

- Coherent Scratchpad Interface

```plaintext
interface MEM_IFC#: (type t_ADDR, type t_DATA);
    method void readReq (t_ADDR addr);
    method void write(t_ADDR addr, t_DATA data);
    method t_DATA readResp();
    // t_REQ r := {READ, WRITE, FULL}
    method Bool requestPending(t_REQ r);
endinterface
```

Fence support (memory consistency)
Parallel Programming on FPGA

• 2D Heat Transfer Equation

```c
for(int t = 0; t < T; t++){
    #pragma omp parallel num_threads(4){
        int thread_id = omp_get_thread_num();
        int bid_x = thread_id%2;
        int bid_y = thread_id/2;
        for (int y = bid_y*(N/2); y < (1+bid_y)*(N/2); y++)
            for (int x = bid_x*(M/2); x < (1+bid_x)*(M/2); x++)
                U[t+1,x,y] = C0*U[t,x,y]
                    + Cx*(U[t,x-1,y]+U[t,x+1,y])
                    + Cy*(U[t,x,y-1]+U[t,x,y+1]);
    }
}
```

operation on the shared array

implicit barrier synchronization

• Finish the inner loop operations
  ✓ Computations complete
  ✓ Memory operations complete
• Wait until all threads are finished
Synchronization Services: Memory Barrier

- In Processor: software through-memory barriers
  - via shared memory & locks

```c
void barrier(num_threads_const, lock_addr, eflag_addr, lflag_addr, ecounter_addr, lcounter_addr)
{
    while (*eflag_addr);
    lock(lock_addr);
    (*ecounter_addr)++;
    if ((*ecounter_addr) == num_thread_const){
        (*eflag_addr) = 0;
        (*lflag_addr) = 1;
    }
    unlock(lock_addr);
    while (*lflag_addr);
    lock(lock_addr);
    (*lcounter_addr)++;
    if ((*lcounter_addr) == num_thread_const){
        (*lcounter_addr) = 0;
        (*ecounter_addr) = 0;
        (*eflag_addr) = 1;
        (*lflag_addr) = 0;
    }
    unlock(lock_addr);
}
```
Synchronization Services: Memory Barrier

• In Processor: software through-memory barriers
  – via shared memory & locks

• In FPGA:
Synchronization Services: Memory Barrier

- **In Processor: software through-memory barriers**
  - via shared memory & locks
- **In FPGA:**

Suppose clients A & B need to synchronize
Synchronization Services: Memory Barrier

- **In Processor:** software through-memory barriers
  - via shared memory & locks
- **In FPGA:**

Suppose clients A & B need to synchronize
Synchronization Services: Memory Barrier

- **In Processor: software through-memory barriers**
  - via shared memory & locks
- **In FPGA:**

Suppose clients A & B need to synchronize
Synchronization Services: Memory Barrier

• **In Processor:** software through-memory barriers
  – via shared memory & locks

• **In FPGA:**

Suppose clients A & B need to synchronize
Synchronization Services: Memory Barrier

- In Processor: software through-memory barriers
  - via shared memory & locks
- In FPGA:

Suppose clients A & B need to synchronize
Synchronization Services: Memory Barrier

• **In Processor: software through-memory barriers**
  – via shared memory & locks

• **In FPGA:**

Suppose clients A & B need to synchronize

```c
void barrier()
{
    send(isDone);
    while (receive(allDone));
}
```

![Diagram of synchronization services with clients A, B, C, and synchronization client with allDone and mask]
Synchronization Services: Memory Barrier

- **In Processor:** software through-memory barriers
  - via shared memory & locks
- **In FPGA:**
  - outside of shared memory
- **Performance Comparison:**

<table>
<thead>
<tr>
<th>System</th>
<th>Barriers per Second</th>
<th>Normalized Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEAP Barrier Service</td>
<td>7352076</td>
<td>342</td>
</tr>
<tr>
<td>Hardware Lock Barrier via Coherent Scratchpad</td>
<td>85088</td>
<td>4</td>
</tr>
<tr>
<td>Spin-Lock Mutex-Enabled Cache*</td>
<td>21510</td>
<td>1</td>
</tr>
</tbody>
</table>

Performance on 2D Heat Transfer

FPGA: Xilinx VC707
Frame Size: 512x512
Coherent Cache Size: 8KB
Pixel Size: 8bit
Performance of Dual FPGA

- 2D Heat Transfer Equation

FPGA: Xilinx VC707
Frame Size: 512x512
Coherent Cache Size: 64KB
Pixel Size: 8bit
Conclusion

• Programming on FPGA is difficult due to the lack of useful abstractions

• We provide a set of FPGA-based shared memory primitives:
  – Coherent scratchpads: manage multiple coherent caches
  – Synchronization primitives

• We improve programming efficiency
  – Common interface:
    Block RAM -> multi-FPGA coherent memory
  – It took only a few hours to write the 2D heat transfer equation
Thank You