LMC: Automatic Resource-Aware Program-Optimized Memory Partitioning

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

† Massachusetts Institute of Technology,
‡ Intel Corporation, § Imperial College London,

February 22nd, FPGA 2016
Motivation

Moore’s Law continues

- More transistors & memory controllers on modern FPGAs
  - Example: **Xilinx VC709**: two 4GB DDR3 memories
  - **Nallatech 510T**: eight 4GB DDR4 memories + 2GB HMC
  - **Xeon + FPGA**: three memory channels

It is difficult to fully utilize DRAM bandwidth

- Co-optimizing application cores and memory systems
- Porting an existing design to a new platform
  - Smaller FPGA -> Larger FPGA
  - Single FPGA -> Multiple FPGAs
Motivation

Moore’s Law continues

- More transistors & memory controllers on modern FPGAs
  - Example: **Xilinx VC709**: two 4GB DDR3 memories
  - **Nallatech 510T**: eight 4GB DDR4 memories + 2GB HMC
  - **Xeon + FPGA**: three memory channels

It is difficult to fully utilize DRAM bandwidth

- Co-optimizing application cores and memory systems
- Porting an existing design to a new platform
  - Smaller FPGA -> Larger FPGA
  - Single FPGA -> Multiple FPGAs

**Goal**: automatically optimizing the memory system to efficiently utilize the increased DRAM bandwidth
Utilizing Multiple DRAMs

- How to connect computational engines to DRAMs in order to maximize program performance?
  - Network topology: latency, bandwidth
  - On-chip caching
  - Area constraints
Utilizing Multiple DRAMs

• How to connect computational engines to DRAMs in order to maximize program performance?
  – Network topology: latency, bandwidth
  – On-chip caching
  – Area constraints

High design complexity!
Utilizing Multiple DRAMs

• How to connect computational engines to DRAMs in order to maximize program performance?
  – High design complexity: network, caching...
Utilizing Multiple DRAMs

• How to connect computational engines to DRAMs in order to maximize program performance?
  – High design complexity: network, caching...
• Applications have different memory behavior
Utilizing Multiple DRAMs

• How to connect computational engines to DRAMs in order to maximize program performance?
  – High design complexity: network, caching...

• Applications have different memory behavior

Need more bandwidth!
Utilizing Multiple DRAMs

• How to connect computational engines to DRAMs in order to maximize program performance?
  – High design complexity: network, caching...

• Applications have different memory behavior

Need more bandwidth!
Utilizing Multiple DRAMs

• How to connect computational engines to DRAMs in order to maximize program performance?
  – High design complexity: network, caching...

• Applications have different memory behavior

Need more bandwidth!
Automatic Construction of Program-Optimized Memories

• A clearly-defined, generic memory abstraction
  – Separate the user program from the memory system implementation

• Program introspection
  – To understand the program’s memory behavior

• A resource-aware, feedback-driven memory compiler
  – Use introspection results as feedback to automatically construct the “best” memory system for the target program and platform
Abstraction

• Abstraction hides implementation details and provides good programmability
Abstraction

• Abstraction hides implementation details and provides good programmability

Compilers & system developers

• Hardware can be optimized for the target application and platform
LEAP Memory Abstraction

Interface MEM_IIFC#(type t_ADDR, type t_DATA)
  method void readReq(t_ADDR addr);
  method void write(t_ADDR addr, t_DATA din);
  method t_DATA readResp();
endinterface

LEAP memory block
- Simple memory interface
- Arbitrary data size
- Private address space
- “Unlimited” storage
- Automatic caching

Interface

User Engine

LEAP Memory
LEAP Memory Abstraction

interface MEM_IFC#(type t_ADDR, type t_DATA)
    method void readReq(t_ADDR addr);
    method void write(t_ADDR addr, t_DATA din);
    method t_DATA readResp();
endinterface
LEAP Private Memory

LEAP Private Memory

LEAP Private Memory

LEAP Memory with Multiple DRAMs

- **Naïve solution:** unified memory with multiple DRAM banks
LEAP Memory with Multiple DRAMs

- **Naïve solution**: unified memory with multiple DRAM banks
LEAP Memory with Multiple DRAMs

- **Naïve solution:** unified memory with multiple DRAM banks

   - Simplicity
   - More capacity
   - Higher bandwidth
**LEAP Memory with Multiple DRAMs**

- **Naïve solution:** unified memory with multiple DRAM banks

- **Difficulty:** Performance is limited
  - Serialized requests
  - Long latency for large rings

- **Simplicity**
- **More capacity**
- **Higher bandwidth**
LEAP Memory with Multiple DRAMs

- **Naïve solution:** unified memory with multiple DRAM banks

**Difficulty:** Performance is limited
- Serialized requests
- Long latency for large rings

**Simplicity**
- More capacity
- Higher bandwidth

**Can we do better?**
LEAP Memory with Multiple DRAMs

• Distributed central caches and memory controllers
LEAP Memory with Multiple DRAMs

- Distributed central caches and memory controllers
Private Cache Network Partitioning

- Program introspection
  - To understand programs’ memory behavior

Statistics file
Client A: 100
Client B: 10
Client C: 50
Client D: 20

Statistics Counter
Ex: # Cache misses
   # Outstanding requests
   Queueing delays
Private Cache Network Partitioning

- Case 1: Memory clients with homogeneous behavior
Private Cache Network Partitioning

- Case 1: Memory clients with homogeneous behavior

![Diagram showing a network of private cache clients with homogeneous behavior.]
Private Cache Network Partitioning

- Case 1: Memory clients with homogeneous behavior
Case 2: Memory clients with heterogeneous behavior

Private Cache Network Partitioning

- Traffic:
  - 100
  - 10
  - 50
  - 20

- Clients and private memory clients connected through an interface.
Private Cache Network Partitioning

- Case 2: Memory clients with heterogeneous behavior

![Diagram of private memory network with heterogeneous traffic](image)
Private Cache Network Partitioning

- Case 2: Memory clients with heterogeneous behavior

Need more bandwidth!
Private Cache Network Partitioning

- Case 2: Memory clients with heterogeneous behavior
  - Load-balanced partitioning
    - Classical minimum makespan scheduling problem

\[ m \text{ controllers, } n \text{ clients, client } j \text{ with traffic } t_j \]

\[ x_{i,j} = \begin{cases} 
1 & \text{if client } j \text{ is mapped to controller } i \\
0 & \text{otherwise} 
\end{cases} \]

**ILP formulation:**

minimize \( t \)

s.t. \( \sum_{j=1}^{n} x_{i,j} t_j \leq t, \quad i = 1, \ldots, m \)

\( \sum_{i=1}^{m} x_{i,j} = 1, \quad j = 1, \ldots, n \)

\( x_{i,j} \in \{0,1\}, \quad i = 1, \ldots, m, j = 1, \ldots, n \)
Private Cache Network Partitioning

• Case 2: Memory clients with heterogeneous behavior
  – Load-balanced partitioning
    • Classical minimum makespan scheduling problem

$m$ controllers, $n$ clients, client $j$ with traffic $t_j$

$$x_{i,j} = \begin{cases} 
1 & \text{if client } j \text{ is mapped to controller } i \\
0 & \text{otherwise}
\end{cases}$$

ILP formulation:

Approximation:
Longest processing time (LPT) algorithm

minimize $t$

s.t. \( \sum_{j=1}^{n} x_{i,j} t_j \leq t, \quad i = 1, \ldots, m \)

\( \sum_{i=1}^{m} x_{i,j} = 1, \quad j = 1, \ldots, n \)

\( x_{i,j} \in \{0,1\}, \quad i = 1, \ldots, m, j = 1, \ldots, n \)
Private Cache Network Partitioning

- Case 3: Fractional load-balancing
**Private Cache Network Partitioning**

- **Case 3: Fractional load-balancing**
• **Case 3: Fractional load-balancing**

ILP->LP

\[
\begin{align*}
\text{minimize} & \quad t \\
\text{s.t.} & \quad \sum_{j=1}^{n} x_{i,j} t_j \leq t \\
& \quad \sum_{i=1}^{m} x_{i,j} = 1 \\
& \quad 0 \leq x_{i,j} \leq 1
\end{align*}
\]
LEAP Memory Compiler

• Three-phase feedback-driven compilation
  – Instrumentation (optional): to collect runtime information about the way the program uses memory
  – Analysis: to analyze the program properties and decide an optimized memory hierarchy
  – Synthesis: to implement the program-optimized memory
LEAP Memory Performance

- Baseline
LEAP Memory Performance

- Baseline

Diagram showing the performance of different cache configurations (Private cache and Central cache) with varying stride and working set size.
LEAP Memory Performance

- Memory interleaving

![3D Graph showing Memory Performance](image)
Case Study: Cryptosorter

- **Cryptosorter**: each sorter uses a LEAP private memory

![Sorter Normalized Runtime](chart.png)

- **Baseline**
- **Partition**
- **Interleaving + Partition**

Sorters: 1, 2, 3, 4
Case Study: Filtering Algorithm

- Filtering algorithm for K-means clustering (HLS kernel)
  - 8 partitions: each uses 3 LEAP private memories
Coherent Cache Network Partitioning

- Baseline coherent memory

Diagram showing the network architecture with clients, coherent memory clients, and baseline coherence manager.
Coherent Cache Network
Partitioning

• Coherent memory interleaving
Coherent Cache Network Partitioning

- Coherent memory interleaving

Private cache network optimizations can be directly composed
Case Study: Heat Transfer

- **Heat transfer**: 16 engines, 1024x1024 frame
Case Study: Heat Transfer

- **Heat transfer**: 16 engines, 1024x1024 frame

<br>

<table>
<thead>
<tr>
<th></th>
<th>Baseline Coherence Manager</th>
<th>Dual Interleaved Coherence Managers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td><img src="image1" alt="" /></td>
<td><img src="image2" alt="" /></td>
</tr>
<tr>
<td>Random Partition</td>
<td><img src="image3" alt="" /></td>
<td><img src="image4" alt="" /></td>
</tr>
<tr>
<td>Balanced Partition</td>
<td><img src="image5" alt="" /></td>
<td><img src="image6" alt="" /></td>
</tr>
<tr>
<td>Interleaving + Balanced Partition</td>
<td><img src="image7" alt="" /></td>
<td><img src="image8" alt="" /></td>
</tr>
<tr>
<td>Baseline</td>
<td><img src="image9" alt="" /></td>
<td><img src="image10" alt="" /></td>
</tr>
<tr>
<td>Random Partition</td>
<td><img src="image11" alt="" /></td>
<td><img src="image12" alt="" /></td>
</tr>
<tr>
<td>Balanced Partition</td>
<td><img src="image13" alt="" /></td>
<td><img src="image14" alt="" /></td>
</tr>
<tr>
<td>Interleaving + Balanced Partition</td>
<td><img src="image15" alt="" /></td>
<td><img src="image16" alt="" /></td>
</tr>
</tbody>
</table>

*57% (96%) performance gain*

**Private memory optimizations only**  **Private + coherent memory optimizations**
Moving to Multi-FPGA Platforms
Moving to Multi-FPGA Platforms

The diagram illustrates the architecture of Multi-FPGA platforms, showing how clients interact with different components such as private memory clients, central cache controllers, and local memory (DRAM banks). The diagram emphasizes the integration and communication between these components across multiple FPGAs.
Performance on Dual FPGAs

Normalized Runtime

<table>
<thead>
<tr>
<th></th>
<th>4 Sorters</th>
<th>8 Sorters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single VC707</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Dual VC707</td>
<td>0.44</td>
<td>0.66</td>
</tr>
<tr>
<td>Single VC709</td>
<td>0.24</td>
<td>0.54</td>
</tr>
<tr>
<td>Dual VC709</td>
<td>0.20</td>
<td>0.46</td>
</tr>
</tbody>
</table>

Merge Sort
Conclusion

• We introduce the LEAP memory compiler that can transparently optimize the memory system for a given application.
• The compiler automatically partitions both private and coherent memory networks to efficiently utilize the increased DRAM bandwidth on modern FPGAs.
• Future work:
  – More case studies on asymmetric memory clients
  – More complex memory network topologies
  – Dynamic cache partitioning
Thank You