MEDEA: Improved Memory-Level Parallelism in a decoupled execute/access vector accelerator
(work in progress)

Umair Riaz
Luis A. Plana
Peter Wilson
John D. Davis

Barcelona Supercomputing Center
Overview

• Motivation
• Introduction to MEDEA
• Microarchitecture
  • Interfaces
  • Supported types of requests
  • Building blocks of MEDEA
• Discussion
Motivation

- Efficient use of memory bandwidth for sparse access patterns
- Reducing data movements between compute node and memory
- Reducing NoC traffic
- Efficient vector data processing
- Improve memory-level parallelism (MLP)
Introduction — A classical system

• A classical system’s representation

```
L2 cache

computation node

vector loads/stores turn into sequences of cache line transfers

memory
```
Introduction – MEDEA in a system

• A classical system with MEDEA
Introduction - MEDEA in ACME

- ACME increases MLP by shifting memory-accessing responsibilities from compute tile to specialized Memory Engine for Decoupled Execute/Access (MEDEA)

- VPU is commonly known to exploit data-level parallelism (DLP), but with the addition of MEDEA, it adds up capabilities for MLP
Microarchitecture - MEDEA
Microarchitecture - Interfaces

MEDEA

- vNoC
- cNoC
- memory crossbar
- memory controller i/f
Microarchitecture – Interfaces (2)

- cNoC (compute NoC) $\leftrightarrow$ compute node
- vNoC (vector NoC) $\leftrightarrow$ LVRF
- Memory crossbar $\leftrightarrow$ interconnecting all the MEDEA tiles and memories
- Memory controller i/f $\leftrightarrow$ HBM and NVRAM
## Microarchitecture — Types of Requests

<table>
<thead>
<tr>
<th>Request</th>
<th>Request Parameters</th>
<th>Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache miss — read</td>
<td>physical address, length</td>
<td>memory data</td>
</tr>
<tr>
<td>cache miss - write</td>
<td>physical address, length, data</td>
<td>completion acknowledge</td>
</tr>
<tr>
<td>virtual-to-physical address translation</td>
<td>virtual address</td>
<td>physical address</td>
</tr>
<tr>
<td>vector parameter set</td>
<td>application-requested vector length</td>
<td>granted vector length</td>
</tr>
<tr>
<td>vector load</td>
<td>virtual address, addressing mode, vector register, renamed vector register, (mode-dependent) parameters: stride, index vector</td>
<td>(densified) memory data</td>
</tr>
<tr>
<td>vector store</td>
<td>virtual address, addressing mode, vector register, renamed vector register, (mode-dependent) parameters: stride, index vector</td>
<td>completion acknowledge</td>
</tr>
<tr>
<td>atomic memory operations</td>
<td>TBD</td>
<td>completion acknowledge</td>
</tr>
</tbody>
</table>
Microarchitecture – Building Blocks

Vector Fragment Sequencer

The data of the different transactions is put together in the LVRF.
Vector Fragment Sequencer

- RISC-V vector operations support following addressing modes
  - unit-stride: managed as dense memory accesses
  - strided, indexed: managed as sparse memory accesses

- In the case of strided or indexed mode, a fragment might end up having a single vector element

- All the elements from different fragments are collected and packed locally and transferred to LVRF as a dense vector
  - Less parasitic data movements
  - Consequently, saving energy and NoC traffic
Prefetcher

- Application
  - application vector length (AVL)

- LVRF
  - granted vector length (GVL)
  - GVL is constrained by the physical limits of the LVRF

- Prefetcher
  - AVL - GVL

- non-speculative prefetch - knowing that request will eventually arrive

- Memory
Memory CPU (MCPU)

- A scalar processor
- Tightly-coupled memory and a low-latency interface to the memory controller
- Provides a collection of memory-intensive functions that can be accessed by the compute tiles
- Executing the functions locally and close to memory improves:
  - Performance
  - Energy
  - NoC traffic
Discussion

- Sparse Matrix Vector (SpMV) benchmark simulation time comparison
Thank you!

umair.riaz@bsc.es