ACCELERATION VIA EXPLICIT DECOUPLED DATA ORCHESTRATION

Michael Pellauer*
1/26/2019

[Extended version to appear in: ASPLOS 2019]

In collaboration with:
Yakun Sophia Shao*, Jason Clemons*, Neal Crago*, Kartik Hegde**, Rangharajan Venkatesan*, Stephen W. Keckler*, Christopher W. Fletcher, ** Joel Emer**

*Nvidia    **UIUC      ^MIT
ACCELERATORS ARE GREAT.... BUT!

Custom Datapath

Off-Chip Memory
WHAT IS DATA ORCHESTRATION?
Feeding data to a functional unit exactly when it wants it

When data is moved over a transfer substrate

Where data is placed in available staging buffers

Who the “actors” are that touch data and their synchronization with each other

How data is accessed (strides, patterns, etc.), including when it is no longer needed

ML ASICs use workload knowledge to optimize orchestration at design-time without caches
GUIDING PRINCIPLES FOR EFFICIENT DATA ORCHESTRATION

Local reuse - staged physically close to consuming units

Cross-unit use - amortize data access and communication

Bandwidth efficiency - Maximize delivery rate by controlling outstanding requests

Delivery/use overlap - Next tile should be available when current is done (e.g., double-buffering)

Precise synchronization - Only wait for exactly data you need, respond quickly (e.g., no barriers or remote polling)

Simple structures - Minimize hardware area/power
CLASSIFYING APPROACHES: IMPLICIT VERSUS EXPLICIT

**Implicit:**

- **Way 0:**
  - Local Request
  - Local Response
  - Global Request
  - Global Response

- **Way 1:**
  - Local Request
  - Local Response
  - Global Request
  - Global Response

- **Cache:**
  - Address: 0x1000

**Explicit:**

- **Scratchpad (e.g., GPU shared memory):**
  - Address: 0x1

- **DRAM:**
  - Address: 0x1000

- **Global Response**
- **Global Request**
CLASSIFYING APPROACHES: COUPLED VERSUS DECOUPLED

Implicit + Coupled

Implicit + Decoupled
EXPLICIT DECOUPLED DATA ORCHESTRATION

Implicit + Decoupled

Explicit + Decoupled
## PROPERTIES OF APPROACHES

<table>
<thead>
<tr>
<th></th>
<th>CPU + Cache Implicit, Coupled</th>
<th>SM + ShMem Spad Explicit, Coupled</th>
<th>DAE CPU + Cache Implicit, Decoupled</th>
<th>DMA Eng. + FIFO Explicit, Decoupled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buf. Area/Energy</td>
<td>High</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Placement policy</td>
<td>Heuristic</td>
<td>Programmatic</td>
<td>Heuristic</td>
<td>Programmatic</td>
</tr>
<tr>
<td>Hier. Composable</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Access Multicast</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MLP of Fills</td>
<td>Complex</td>
<td>Complex</td>
<td>Cheap</td>
<td>Cheap</td>
</tr>
<tr>
<td>Landing Zone Holding Time</td>
<td>Round-trip</td>
<td>Round-trip</td>
<td>Hop-to-Hop</td>
<td>Hop-to-Hop</td>
</tr>
<tr>
<td>Data Availability Synchronization</td>
<td>Encapsulated (load-to-use)</td>
<td>Encapsulated (load-to-use)</td>
<td>Out-of-band</td>
<td>Encapsulated (peek stalling)</td>
</tr>
<tr>
<td>Access order</td>
<td>Arbitrary</td>
<td>Arbitrary</td>
<td>Arbitrary</td>
<td>Fixed FIFO</td>
</tr>
<tr>
<td>In-place updates</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Removal</td>
<td>Heuristic</td>
<td>Programmatic</td>
<td>Heuristic</td>
<td>Dequeue/clear</td>
</tr>
</tbody>
</table>
## PROPERTIES OF APPROACHES

<table>
<thead>
<tr>
<th></th>
<th>CPU + Cache Implicit, Coupled</th>
<th>SM + ShMem Spad Explicit, Coupled</th>
<th>DAE CPU + Cache Implicit, Decoupled</th>
<th>DMA Eng. + FIFO Explicit, Decoupled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buf. Area/Energy</td>
<td>High</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Placement policy</td>
<td>Heuristic</td>
<td>Programmatic</td>
<td>Heuristic</td>
<td>Programmatic</td>
</tr>
<tr>
<td>Hier. Composable</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Access Multicast</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MLP of Fills</td>
<td>Complex</td>
<td>Complex</td>
<td>Cheap</td>
<td>Cheap</td>
</tr>
<tr>
<td>Landing Zone Holding Time</td>
<td>Round-trip</td>
<td>Round-trip</td>
<td>Hop-to-Hop</td>
<td>Hop-to-Hop</td>
</tr>
<tr>
<td>Data Availability Synchronization</td>
<td>Encapsulated (load-to-use)</td>
<td>Encapsulated (load-to-use)</td>
<td>Out-of-band</td>
<td>Encapsulated (peek stalling)</td>
</tr>
<tr>
<td>Access order</td>
<td>Arbitrary</td>
<td>Arbitrary</td>
<td>Arbitrary</td>
<td>Fixed FIFO</td>
</tr>
<tr>
<td>In-place updates</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Removal</td>
<td>Heuristic</td>
<td>Programmatic</td>
<td>Heuristic</td>
<td>Dequeue/clear</td>
</tr>
</tbody>
</table>

- These are not limitations of EDDO, but of the FIFO idiom
- Buffets change these points to \{Arbitrary, Yes, Programmatic (Contiguous)\}
BUFFETS: COMPOSABLE IDIOM FOR E.D.D.O.

Details to appear in ASPLOS 2019 [April, Providence]
ARCHITECTURAL VISION FOR E.D.D.O.

Traditional JIT

- Portable Code

- uArch Specific Code

- JIT

- uArch Description

Data-Size Dependent JIT Mapper

- Portable Code

- Input Data Description

- uArch Description

- Blocked, mapped uArch-Specific Code

+ Mapper
IDEAS FOR POTENTIAL AUTOMATIC MAPPERS

Have the program pre-select a “menu” and provide a heuristic?

Train a neural net?

Use tensor decomposition + tensor prediction?

Key idea: run the mapper on the accelerator itself…

Open question: how to make this work with sparsity? What can be conveyed to the mapper in O(1) time?