Hardware Abstractions for targeting EDDO Architectures with the Polyhedral Model

Angshuman Parashar  
NVIDIA

Prasanth Chatarasi  
IBM Research/Georgia Tech

Po-An Tsai  
NVIDIA

IMPACT workshop  
January 20, 2021
DATA MOVEMENT IS A PROBLEM

Raw Energy Costs

<table>
<thead>
<tr>
<th>Energy costs</th>
<th>Energy Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit Integer Multiply</td>
<td>0.2 pJ</td>
</tr>
<tr>
<td>Fetch two 8-bit operands from large SRAM</td>
<td>2 pJ</td>
</tr>
<tr>
<td>Fetch two 8-bit operands from DRAM</td>
<td>128 pJ</td>
</tr>
</tbody>
</table>

Energy stack for mapped GEMM layer

Explicit Decoupled Data Orchestration (EDDO) architectures attempt to minimize data movement costs
**EDDO ARCHITECTURES**

*Implicit Coupled Data Orchestration (ICDO)*  
e.g., CPUs, GPUs

*Explicit Decoupled Data Orchestration (EDDO)*  
e.g., Simba, NVDLA, Eyeriss, etc.
Benefits

- Dedicated (and often statically programmed) state machines more efficient than general cores
- Perfect “prefetching”
- Buffet storage idiom provides fine-grain synchronization and efficient storage
- Hardware mechanisms for reuse

Explicit Decoupled Data Orchestration (EDDO)

e.g., Simba, NVDLA, Eyeriss, MAERI, etc.

Pellauer et. al., “Buffets: An Efficient and Composable Storage Idiom for Explicit Decoupled Data Orchestration”, ASPLOS 2019
EDDO ARCHITECTURES

Challenges

1. **No single binary: Collection of distinct binaries** that program distributed state machines working together to execute algorithm
   - E.g., CNN layer on EDDO arch → ~250 distinct state machines.

2. **Reuse optimization is critical for efficiency**
   - E.g., CNN layer on EDDO arch → 480,000 mappings, 11x spread in energy efficiency, 1 optimal mapping
   - Need an optimizer or mapper

3. **Variety of EDDO architectures, constantly evolving**
   - Need an abstraction that Mapper and Code Generator will target

Explicit Decoupled Data Orchestration (**EDDO**) e.g., Simba, NVDLA, Eyeriss, MAERI, etc.
HARDWARE SPACE-TIME (HST)

Mapper * (optimizer) → PolyEDDO Code Generator → Arch-specific configuration generator → Configuration Binaries

Workload → Mapping → Arch-independent decoupled programs → Data-movement activity counts

EDDO Architecture described using Hardware Space-Time (HST)

Perf, Energy, Area

Analytical microarchitecture and energy model †

HARDWARE SPACE-TIME (HST)

This talk focuses on the HST abstraction, with a high-level overview of PolyEDDO

---

**Mapper** *(optimizer)*

Workload → Mapping → PolyEDDO Code Generator → Arch-specific configuration generator → Configuration Binaries

Arch-independent decoupled programs

----

EDDO Architecture described using Hardware Space-Time (HST)

Perf, Energy, Area

Analytical microarchitecture and energy model †

Data-movement activity counts

---

EXAMPLE 1
Symbolic Hardware Space-Time (SHST)

\[ \text{SpaceTime}_2 [s_2, t_2] \rightarrow \text{SpaceTime}_1 [s_1, t_1] : \]

\[ s_2 = 0 \quad t_2 = 0 \quad 0 \leq s_1 < 4 \quad 0 \leq t_1 < 3 \]

Single L2, 4 L1s, 3 time-steps
- In each step, the L2 delivers a tile of data to each L1
- Across all these L1 time steps, the resident tile in L2 does not change. In effect, time is stagnant for L2
EXAMPLE 2

\[ \text{SpaceTime}_3[s_3, t_3] \rightarrow [\text{SpaceTime}_2[s_2, t_2] \rightarrow \text{SpaceTime}_1[s_1, t_1]] : \]

\[
\begin{align*}
    s_3 &= 0 & t_3 &= 0 \\
    0 \leq s_2 &< 2 & 0 \leq t_2 &< 2 \\
    0 \leq s_1 &< 4 & 0 \leq t_1 &< 3
\end{align*}
\]

\[ \text{SpaceTime}_3[0,0] \rightarrow \text{SpaceTime}_2[1,0] \rightarrow \text{SpaceTime}_1[2,1] \]
EXAMPLE 3
Partitioned Buffers

\[
\text{PHST} = \text{DRAM} \quad \rightarrow \quad \text{SHST} \quad \rightarrow \\
\text{OperandA} \quad \rightarrow \quad \text{Result} \quad \rightarrow \\
\text{OperandB} \quad \rightarrow \\
\text{Result} \quad \rightarrow \\
\]
EXAMPLE 4

SHST: $SpaceTime_4 [s_4, t_4] \rightarrow SpaceTime_3 [s_3, t_3] \rightarrow [SpaceTime_2 [s_2, t_2] \rightarrow SpaceTime_1 [s_1, t_1]]$

See paper for full HST

Observe how different the architecture is from CPUs and GPUs

Workload mappings target SHST
POLYEDDO

- Architecture HST
- Workload Mapping
- T-relation generation
  - Tiling (T)-relations
- Decoupling
  - Data Transfer (X)-relations
- Reuse Analysis
  - Delta (Δ)-relations
  - Δ schedules
- Schedule creation
- AST generation
  - Decoupled ASTs
MAPPING WORKLOADS
Perfectly-nested affine loops

Workload Iteration Space

L1

L2

L3

SHST

Project

Project

Project

Tile

Tile
MAPPING WORKLOADS

The Tiling-relation (or T-relation)

**T-relation:** Projection from SHST coordinate to a set of tensor coordinates
- Tells you what tiles of data *must* be present at that point in space-time to honor the mapping.
- Does not tell you how the data got there.


DECOUPLING

Breaking the hierarchy

PHST

Data transfer relations (X-relations)

Tensor coords

L3

DRAM

L2

BufA
BufB
BufZ

BufA
BufB
BufZ

L2


L1

BufA
BufB
BufZ

BufA
BufB
BufZ

L1


L1

 MACCs

MACCs

[MACC[s1, t1]] -> MulAcc[k, p, r] : …
REUSE ANALYSIS

Local Temporal Reuse
REUSE ANALYSIS

Fill from Peer
REUSE ANALYSIS

L2

L1

Fill from parent
OPTIMIZATION PROBLEM (FOR A SINGLE MAPPING!)

Options:

1. Enumerate all possibilities and find optimum solution
2. Use a heuristic
3. Expose choices to mapping (and thereby the mapspace)
POLYEDDO

T-relation generation

Decoupling

Reuse Analysis

Schedule creation

AST generation

Described in paper

Architecture HST
Workload Mapping

Tiling (T)-relations
Data Transfer (X)-relations
Delta (Δ)-relations
Δ schedules
Decoupled ASTs
EXAMPLE OUTPUT

Program to read weights from DRAW into RowBuffer.
if (P >= 1) {
    for (int c3 = 0; c3 < min(15, K - 1); c3++)
        for (int c1 = 0; c1 < min(2, R - 1); c1++)
            ACTION_READ("DRAW", "DRAW", "RowBuffer", "Weights");
}

Program to read inputs from DRAW into DiagBuffer.
if (K >= 1666 P >= 1666 R >= 1) {
    for (int c3 = 0; c3 < min(min(15, P + 1), P + R - 2); R >= 12); c3 <= 1)
        ACTION_READ("DRAW", "DRAW", "DiagBuffer", "Inputs");
}

Program to read outputs from DRAW into ColBuffer.
if (R >= 1) {
    for (int c3 = 0; c3 < min(15, K - 1); c3++)
        for (int c1 = 0; c1 < min(13, P - 1); c1++)
            ACTION_READ("DRAW", "DRAW", "ColBuffer", "Outputs");
}

Program to read weights from RowBuffer into RowBroadcaster.

Program to read inputs from DiagBuffer into DiagBroadcaster.

Program to read outputs from ColBuffer into ColSpatialReducer.

Program to compute Multiply at Multiplier.

Program to read Inputs from DiagBroadcaster into OperandB.
if (K >= 1) {
    for (int c3 = 0; c3 < min(min(6, P + 1), P + R - 2); R >= 3); c3++)
        for (int c8 = max(min(5 * c3 - 16, c3), 4 * P + 5 * c3 + 4); c8 <= min(min(4 * R + c3 - 4, 5 * c3), c3 + 8); c8++)
            ACTION_READ("DiagBroadcaster", "DiagBroadcaster", "OperandB", "Inputs");
}

Program to read Outputs from ColSpatialReducer into Result.
if (K >= 1) {
    for (int c0 = 0; c0 < min(15, K - 1); c0++)
        for (int c4 = 0; c4 <= min(4, P - 1); c4++)
            for (int c8 = min(min(5 * R + c4 - 5, c4 + 10); c8 <= 5)
                for (int c6 = max(min(5 * c3 - 18, c3), 4 * P + 5 * c3 + 4); c6 <= min(min(4 * R + c3 - 4, 5 * c3), c3 + 8)
                    ACTION_SHRINK("DiagBroadcaster", "DiagBroadcaster", "Inputs");
}

Program to compute Multiply at Multiplier.

Present capability: build generated code against an EDDO emulator (automatically configured from the PHST)
FINAL REMARKS

Contributions

• HST (Hardware Space-Time) - an abstraction for EDDO architectures represented using the Polyhedral Model
• PolyEDDO (WIP) - an analysis and code-generation flow based on HST

Future Work

• Complete implementation and description of PolyEDDO
• Optimizer/Mapper
• Integration with existing toolchains (Timeloop, MARVEL, MAESTRO)
• Imperfectly nested loops
• Support for sparsity (longer term)