P&S Processing-in-Memory

Real-World Processing-in-Memory Architectures: Samsung AxDIMM

Dr. Juan Gómez Luna
Prof. Onur Mutlu

ETH Zürich
Spring 2023
20 April 2023
UPMEM PIM System Organization

• A UPMEM DIMM contains 8 or 16 chips
  - Thus, 1 or 2 ranks of 8 chips each
• Inside each PIM chip there are:
  - 8 64MB banks per chip: Main RAM (MRAM) banks
  - 8 DRAM Processing Units (DPUs) in each chip, 64 DPUs per rank
FIMDRAM: System Organization (III)

- PIM units respond to standard DRAM column commands (RD or WR)
  - Compliant with unmodified JEDEC controllers
- They execute one wide-SIMD operation commanded by a PIM instruction with deterministic latency in a lock-step manner
- A PIM unit can get 16 16-bit operands from IOSAs, a register, and/or the result bus
AiM: System Organization

- **GDDR6-based AiM architecture**

![Diagram of GDDR6-based AiM architecture](image-url)
Samsung AxDIMM
Samsung Brings In-Memory Processing Power to Wider Range of Applications

Integration of HBM-PIM with the Xilinx Alveo AI accelerator system will boost overall system performance by 2.5X while reducing energy consumption by more than 60%

PIM architecture will be broadly deployed beyond HBM, to include mainstream DRAM modules and mobile memory

Samsung Electronics, the world leader in advanced memory technology, today showcased its latest advancements with processing-in-memory (PIM) technology at Hot Chips 33—a leading semiconductor conference where the most notable microprocessor and IC innovations are unveiled each year. Samsung’s revelations include the first successful integration of its PIM-enabled High Bandwidth Memory (HBM-PIM) into a commercialized accelerator system, and broadened PIM applications to embrace DRAM modules and mobile memory, in accelerating the move toward the convergence of memory and logic.

DRAM Modules Powered by PIM

The Acceleration DIMM (AXDIMM) brings processing to the DRAM module itself, minimizing large data movement between the CPU and DRAM to boost the energy efficiency of AI accelerator systems. With an AI engine built inside the buffer chip, the AXDIMM can perform parallel processing of multiple memory ranks (sets of DRAM chips) instead of accessing just one rank at a time, greatly enhancing system performance and efficiency. Since the module can retain its traditional DIMM form factor, the AXDIMM facilitates drop-in replacement without requiring system modifications. Currently being tested on customer servers, the AXDIMM can offer approximately twice the performance in AI-based recommendation applications and a 40% decrease in system-wide energy usage.

Samsung AxDIMM (2021)

- DIMM-based PIM
  - DLRM recommendation system
Near-Memory Processing in Action: Accelerating Personalized Recommendation with AxDIMM

Liu Ke*,†, Xuan Zhang†, Jinin So‡, Jong-Geon Lee‡, Shin-Haeng Kang‡, Sukhan Lee‡, Songyi Han‡, YeonGon Cho‡, JIN Hyun Kim‡, Yongsuk Kwon‡, KyungSoo Kim‡, Jin Jung‡, Ilkwon Yun‡, Sung Joo Park‡, Hyunsun Park‡, Joonho Song‡, Jeonghyeon Cho‡, Kyomin Sohn‡, Nam Sung Kim‡, Hsien-Hsin S. Lee*

*Facebook, †Washington University in St. Louis, ‡Samsung

https://doi.org/10.1109/MM.2021.3097700
An Architecture of Sparse Length Sum Accelerator in AxDIMM

Shin-haeng Kang
DRAM Design Team 1
Samsung Electronics
Hwasung, South Korea
s-h.kang@samsung.com

Byeongho Kim
DRAM Design Team 1
Samsung Electronics
Hwasung, South Korea
bh1122.kim@samsung.com

Sukhan Lee
DRAM Design Team 1
Samsung Electronics
Hwasung, South Korea
sh1026.lee@samsung.com

Kyomin Sohn
DRAM Design Team 1
Samsung Electronics
Hwasung, South Korea
kyomin.sohn@samsung.com
Recommendation Systems
Overview of Recommendation Models

- **Personalized recommendation**: recommend content to users, e.g., Facebook’s DLRM recommendation system

---

**Dense features**: continuous inputs in vectors and matrices are processed by typical DNN layers (e.g., fully connected layers)

---

Overview of Recommendation Models

- **Personalized recommendation**: recommend content to users, e.g., Facebook’s DLRM recommendation system

Overview of Recommendation Models

- **Personalized recommendation**: recommend content to users, e.g., Facebook’s DLRM recommendation system.

Embedding tables are organized as a set of potentially millions of vectors: lookup and pooling operations represent sparse features learned during training and generally exhibit **Gather-Reduce pattern**, via Caffe2’s **SparseLengths (SLS) operators**.

AxDIMM Design: Overview

- Accelerator DIMM (AxDIMM)
  - DDR4-compatible FPGA-based platform with standard memory interfaces

- AxDIMM can potentially
  - support both in-order general-purpose processor and specialized accelerator modules
  - be an interesting prototyping platform for near-memory processing

- Personalized recommendation case study, including:
  - hardware implementation
  - software-stack support
AxDIMM Hardware & Architecture
AxDIMM Design: Hardware Architecture

FPGA board with standard DIMM interface:
It serves as a real-system near-memory processing implementation
AxDIMM Design: Hardware Architecture

**Rank-level parallelism:**
Two DRAM ranks are activated in parallel to load embedding entries from memory.

**Element-wise summation** is performed inside the FPGA module.

As illustrated in Figure 4, the execution of Rank-NMP modules is supported in two modes: non-acceleration mode and acceleration mode.

In the non-acceleration mode, the host can directly access the commodity DRAM devices. In the acceleration mode, NMP-instructions are issued by the host to perform NMP operations and load the embedding data. The adder array contains 16 floating-point and Psum vectors. One DRAM read cycle bursts 64-byte commands to DRAM and Psum buffer to load the embedding data. The command generator generates the data loading loads and decodes NMP-instructions from the instruction module. The instruction buffer stores NMP-instructions from the host-side; and the partial sum (Psum) buffer holds the intermediate Psum values for embedding pooling.

As shown in Figure 5, the final pooled results are produced by the Rank-NMP module. (c) Host-NMP offloading model.
AxDIMM Design: Hardware Architecture

Two execution modes:
(1) non-acceleration mode
(2) acceleration mode (blocking)
AxDIMM Design: Hardware Architecture

The AxDIMM platform is capable of theoretically providing 2\(^{64}\) results as a normal DDR4 DRAM DIMM. All accesses from the AxDIMM platform exploit rank-level parallelism of its DDR4 memory modules and accelerate embedding lookup and pooling operations.

**Figure 4:**
(a) DDR4-compatible FPGA-based real-system near-memory processing implementation to execute Rank-NMP modules.
(b) Present the detailed hardware architecture of the Rank-NMP module.
(c) Host-NMP offloading model.

**Figure 5:**
(a) AxDIMM address map.
(b) Control flow of DDR.C/A DDR.DQ.
(c) FPGA MIG (PHY).

**NMP-Inst (64 bits):**

- **OpCode:** 2 bit
- **Locality:** 1 bit
- **PSUM Tag:** 12 bit
- **Trace End:** 1 bit
- **Reserved:** 17 bit
- **Row Addr:** 17 bit
- **BG:** 2 bit
- **BA:** 2 bit
- **Col Addr:** 10 bit

The execution of Rank-NMP modules is supported in two modes—non-acceleration mode and acceleration mode. In the non-acceleration mode, the host can directly access the DDR read/write commands to offload the NMP-instructions. The communication between the Rank-NMP modules and all the regular DRAM accesses from the host are blocked internally.

The instruction buffer (INST BUF), a 256KB SRAM, holds the intermediate Psum values for embedding pooling and stores NMP-instructions from the host-side. The partial sum (Psum) buffer (PSUM BUF), also a 256KB SRAM, stores the element-wise summation performed inside the embedding lookup and pooling operations.

The Rank-NMP module operates following the mechanism of the loaded embedding entry and Psum vectors. One DRAM read cycle bursts 64-byte commands to DRAM and Psum buffer to load the embedding and Psum vectors. The instruction buffer (INST BUF) is a 256KB SRAM for the NMP-instructions.
AxDIMM Design: Hardware Architecture

256-KB partial sum buffer:
It stores intermediate values for embedding pooling operations

<table>
<thead>
<tr>
<th>NMP-Inst (64 bits)</th>
<th>Opcode</th>
<th>Locality</th>
<th>PSUM Tag</th>
<th>Trace End</th>
<th>Reserved</th>
<th>Row Addr</th>
<th>BG</th>
<th>BA</th>
<th>Col Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 bit</td>
<td>1 bit</td>
<td>12 bit</td>
<td>1 bit</td>
<td>17 bit</td>
<td>17 bit</td>
<td>2 bit</td>
<td>2 bit</td>
<td>10 bit</td>
</tr>
</tbody>
</table>
AxDIMM Design: Hardware Architecture

### 3.1 Hardware Architecture

To support the interaction with the host, one DDR4 slave PHY is instantiated to receive the DRAM commands and deliver the final results. As shown in Figure 4, each Rank-NMP module has a DDR4 interface to receive the DRAM commands and Psum buffer to load the embedding entries and the element-wise summation is performed inside the non-acceleration mode. In the acceleration mode, NMP-instructions are issued by the host to bypass the logic of the Rank-NMP modules. With two Rank-NMP modules implemented to interface with the two internal ranks, NMP instructions from the host side are divided and delivered to DRAM devices.

The execution of Rank-NMP modules is supported in two modes: the non-acceleration mode and the acceleration mode. In the non-acceleration mode, the host can directly access the end-point DRAM ranks, and AxDIMM functions in the same commodity DRAM devices. In the acceleration mode, NMP-instructions are issued by the host to perform vector operations. To load the embedding and Psum vectors from the DRAM devices and Psum buffer, the instruction decoder loads and decodes NMP-instructions from the instruction buffer. The command generator generates the data loading commands to DRAM and Psum buffer to load the embedding vectors. One DRAM read cycle bursts 64-byte data.

The instruction decoder, located in the FPGA board with a standard DIMM interface, serves as an end-point interface. This design allows the FPGA to communicate with the host system and execute NMP-instructions. The instruction decoder decodes the NMP-instructions and passes them to the appropriate block for further processing. The FPGA board is the central processing unit that receives and executes the NMP-instructions.

#### Instruction decoder
- **Purpose**: Loads and decodes NMP instructions from the instruction buffer.
- **Function**: Receives and decodes NMP-instructions issued by the host to perform vector operations.

#### NMP-Inst Table

<table>
<thead>
<tr>
<th>OpCode</th>
<th>Locality</th>
<th>PSUM Tag</th>
<th>Trace End</th>
<th>Reserved</th>
<th>Row Addr</th>
<th>BG</th>
<th>BA</th>
<th>Col Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 bit</td>
<td>1 bit</td>
<td>12 bit</td>
<td>1 bit</td>
<td>17 bit</td>
<td>17 bit</td>
<td>2 bit</td>
<td>2 bit</td>
<td>10 bit</td>
</tr>
</tbody>
</table>

#### FPGA Board Components
- **Input I/F**: Receives NMP-instructions and data from the host.
- **Output I/F**: Sends results back to the host.
- **INST BUF**: Buffer for instruction decoding.
- **CMD GEN**: Generates commands based on the decoded instructions.
- **ADDER ARRAY**: Performs arithmetic operations.
- **MIG (PHY)**: Manages memory interface and communication with DRAM devices.
- **DDR4 Slave PHY**: Connects to DRAM devices for data transfer.

The figure depicts the architecture of the AxDIMM, showing the interaction between different components such as the Host, DDR4 Slave PHY, FPGA, and DRAM devices. The instruction decoder is highlighted to indicate its role in processing NMP-instructions. The FPGA board serves as the processing unit, executing the instructions and communicating with the host and DRAM devices.
AxDIMM Design: Hardware Architecture

Command generator issues read commands to DRAM ranks and Psum buffer (64 bytes from each in 1 cycle)

NMP-Inst (64 bits)

<table>
<thead>
<tr>
<th>OpCode</th>
<th>Locality</th>
<th>PSUM Tag</th>
<th>Trace End</th>
<th>Reserved</th>
<th>Row Addr</th>
<th>BG</th>
<th>BA</th>
<th>Col Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 bit</td>
<td>1 bit</td>
<td>12 bit</td>
<td>1 bit</td>
<td>17 bit</td>
<td>17 bit</td>
<td>2 bit</td>
<td>2 bit</td>
<td>10 bit</td>
</tr>
</tbody>
</table>
AxDIMM Design: Command Generator (I)

Command generator issues read commands to DRAM ranks and Psum buffer (64 bytes from each in 1 cycle)

AxDIMM Design: Command Generator (II)

1. Address into bank reservation table
2. **Next target bank checker** does round-robin across banks and bank groups
3. **FSM per bank** for next command
4. **Refresh** management according to tREFI and tRFC
5. **Command serializer** with timing parameters (tRCD, tCL, tRP)
6. Mem(Addr) and pid to rate controller
AxDIMM Design: Hardware Architecture

The execution of Rank-NMP modules is supported in two ways: (a) by offloading NMP-instructions from the host side and (b) by using the standard memory interface. The communication between the Rank-NMP module and the host is performed by normal DRAM accesses between the Rank-NMP module and the commodity DRAM devices.

Figure 4: (a) DDR4-compatible FPGA-based AxDIMM platform, (b) hardware architecture of the AxDIMM.

Figure 5: (a) AxDIMM address map, (b) control flow of NMP-instruction processing.

<table>
<thead>
<tr>
<th>NMP-Inst (64 bits)</th>
<th>OpCode</th>
<th>Locality</th>
<th>PSUM Tag</th>
<th>Trace End</th>
<th>Reserved</th>
<th>Row Addr</th>
<th>BG</th>
<th>BA</th>
<th>Col Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 bit</td>
<td>1 bit</td>
<td>12 bit</td>
<td>1 bit</td>
<td>17 bit</td>
<td>17 bit</td>
<td>2 bit</td>
<td>2 bit</td>
<td>10 bit</td>
</tr>
</tbody>
</table>
AxDIMM Execution Flow
AxDIMM Design: Execution Flow

Diagram showing the execution flow in a DDR4 memory system with various modules such as Emb Table, Mode Change, WR Inst, SLS Execute, RD Status Reg, and RD Psum. The diagram also includes a decoder and data fetch blocks for each rank of the NMP.
AxDIMM Design: Execution Flow
AxDIMM Design: Execution Flow
AxDIMM Design: Execution Flow
Near-Memory Processing in Action: Accelerating Personalized Recommendation with AxDIMM

Liu Ke*, Xuan Zhang†, Jinin So‡, Jong-Geon Lee‡, Shin-Haeng Kang‡, Sukhan Lee‡, Songyi Han‡, YeonGon Cho‡, JIN Hyun Kim‡, Yongsuk Kwon‡, KyungSoo Kim‡, Jin Jung‡, Ilkwon Yun‡, Sung Joo Park‡, Hyunsun Park‡, Joonho Song‡, Jeonghyeon Cho‡, Kyomin Sohn‡, Nam Sung Kim‡, Hsien-Hsin S. Lee*

*Facebook, †Washington University in St. Louis, ‡Samsung

https://doi.org/10.1109/MM.2021.3097700
An Architecture of Sparse Length Sum Accelerator in AxDIMM

Shin-haeng Kang
DRAM Design Team 1
Samsung Electronics
Hwasung, South Korea
s-h.kang@samsung.com

Byeongho Kim
DRAM Design Team 1
Samsung Electronics
Hwasung, South Korea
bh1122.kim@samsung.com

Sukhan Lee
DRAM Design Team 1
Samsung Electronics
Hwasung, South Korea
sh1026.lee@samsung.com

Kyomin Sohn
DRAM Design Team 1
Samsung Electronics
Hwasung, South Korea
kyomin.sohn@samsung.com

https://doi.org/10.1109/AICAS54282.2022.9869896
Improving In-Memory Database Operations with Acceleration DIMM (AxDIMM)

Donghun Lee
Minseon Ahn
Jungmin Kim
dong.hun.lee@sap.com
minseon.ahn@sap.com
jungmin.kim@sap.com
SAP Labs Korea

Jinin So
Jong-Geon Lee
Jeonghyeon Cho
Vishnu Charan Thummala
jinin.so@samsung.com
jg1021.lee@samsung.com
caleb1@samsung.com
vishnu.c.t@samsung.com
Samsung Electronics

Oliver Rebholz
oliver.rebholz@sap.com
SAP SE

Ravi Shankar JV
Sachin Suresh Upadhya
Mohammed Ibrahim Khan
Jin Hyun Kim
venkata.ravi@samsung.com
sachin1.s@samsung.com
ibrahim.khan@samsung.com
kjh5555@samsung.com
Samsung Electronics

https://doi.org/10.1145/3533737.3535093
Scan operation filters out database entries that satisfy a predicate
AxDIMM for Database Scan Operation

The lookup-compare module reads data and compares it with the predicate
AxDIMM for Database Scan Operation

The prefetcher loads data to an SRAM buffer in advance.

Lee et al. "Improving in-memory database operations with acceleration DIMM (AxDIMM)," DaMoN 2022
The **write buffer** stores the scan result temporarily before writing to memory.
AxDIMM for Database Scan Operation

The memory crossbar arbitrates accesses from the three DBA

Lee et al. "Improving in-memory database operations with acceleration DIMM (AxDIMM)," DaMoN 2022
Improving In-Memory Database Operations with Acceleration DIMM (AxDIMM)

Donghun Lee
Minseon Ahn
Jungmin Kim
dong.hun.lee@sap.com
minseon.ahn@sap.com
jungmin.kim@sap.com
SAP Labs Korea

Jinin So
Jong-Geon Lee
Jeonghyeon Cho
Vishnu Charan Thummala
jinin.so@samsung.com
jg1021.lee@samsung.com
caleb1@samsung.com
vishnu.c.t@samsung.com
Samsung Electronics

Oliver Rebholz
oliver.rebholz@sap.com
SAP SE

Ravi Shankar JV
Sachin Suresh Upadhya
Mohammed Ibrahim Khan
Jin Hyun Kim
venkata.ravi@samsung.com
sachin1.s@samsung.com
ibrahim.khan@samsung.com
kjh5555@samsung.com
Samsung Electronics
Longer Lecture on AxDIMM

AxDIMM Design: Address Map

- Memory map of AxDIMM

AxDIMM Rank-0
  - Reserved
  - PSUM BUF
  - CONF REG
  - Reserved
  - INST BUF

AxDIMM Rank-1

Embedding Table

PIM Course: Lecture 7: Real-world PIM: Samsung AxDIMM - Fall 2022

https://youtu.be/SXdzQZAKG-Y
Another Longer Lecture on AxDIMM

DLRM Performance Characterization

- Identifying key performance bottlenecks for the DLRM system

SparseLengths (SLS) operators:
- Low FP intensity
- Larger batch size:
  - Higher memory footprint
  - Higher memory intensity

The memory bandwidth can easily be saturated by embedding operations especially as both the batch size and the number of threads increase

Processing in Memory Course: Meeting 5: Real-world PIM architectures IV - Fall'21

https://youtu.be/2FMQg786GKs
Upcoming Lectures

- More real-world PIM architectures
- PUM architectures and prototypes
- Enabling the adoption of PIM
P&S Processing-in-Memory

Real-World Processing-in-Memory Architectures: Samsung AxDIMM

Dr. Juan Gómez Luna
Prof. Onur Mutlu

ETH Zürich
Spring 2023
20 April 2023