CoolPIM: Thermal-Aware Source Throttling for Efficient PIM Instruction Offloading

Lifeng Nai† † Ramyad Hadidi† He Xiao† Hyojong Kim† Jaewoong Sim‡ Hyesoon Kim†

†Georgia Institute of Technology
‡Google, †Intel Labs

Disclaimer: This work does not relate to Google/Intel Labs
Processing-in-Memory

Processing-in-memory (PIM) is regaining attention for energy efficient computing

- Graph Workloads: Data-Intensive, Little Data Reuse

**Basic Concept:** Offload compute to memory

- Reduce *costly* energy consumption of data movement
- Enable using large internal memory bandwidth
**Thermal Challenge in PIM**

PIM could increase memory temperature beyond normal operating temperature (85°C)

- High BW (hundreds of GBs ~ TBs) from 3D-stacked memory
- Less effective heat transfer compared to DIMMs
- PIM would make these thermal problems worse!

**Too Hot Memory Stack?**

- Slower processing for memory requests
- Decreasing overall system performance

CoolPIM keeps the memory “Cool” to achieve better PIM performance

Rarely exceeds 85°C
Outline

Introduction

Hybrid Memory Cube
- Background
- Thermal Measurements & Thermal Modeling of Future HMC

CoolPIM
- Software-Based Throttling
- Hardware-Based Throttling

Evaluation

Conclusion
A Hybrid Memory Cube (HMC) from Micron

- Multiple 3D-stacked DRAM layers + one logic layer with TSVs
- Vaults: equivalent to memory channels
- Full-duplex serial links between the host and HMC

No PIM functionality for existing HMC products yet
Instruction-level PIM supported in future HMC (HMC 2.0)

- Perform **Read-Modify-Write (RMW)** operations atomically
- Similar to READ/WRITE packets; just different **CMD** in the Header
- **No HMC 2.0 product yet!**

**Q:** Can we offload all the PIM operations to HMC? What is the thermal impact of PIM in future HMC?

<table>
<thead>
<tr>
<th>Type</th>
<th>HMC 2.0 PIM Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>Signed Add</td>
</tr>
<tr>
<td>Bitwise</td>
<td>Swap, bit write</td>
</tr>
<tr>
<td>Boolean</td>
<td>AND/NAND/OR/NOR/XOR</td>
</tr>
<tr>
<td></td>
<td>Equal/greater</td>
</tr>
</tbody>
</table>

**PIM-ADD** *(addr, imm)*
Existing HMC Thermal Measurement (1)

Experiment Platform (Pico SC-6 Mini System)
- Intel Core i7 + FPGA Compute Modules (AC-510)
  - AC-510: 4GB HMC 1.1, Kintex Ultrascale

Measure the temperature on the heat sink
- Controlling memory BW via FPGA
- Applying three different cooling methods
  - High-End Active Heat Sink
  - Low-End Active Heat Sink
  - Passive Heat Sink

HMC 1.1 has no PIM functionality!
Existing HMC Thermal Measurement (2)
Thermal modeling for HMC 2.0 with commodity-server active cooling

- HMC 2.0 (w/o PIM) would reach 81°C at a full external BW (320GB/s)
  - We validated our thermal model against the measurements on HMC 1.1

We need at least commodity-server cooling to benefit from PIM!
Thermal Impact with PIM in HMC 2.0

PIM increases memory temperature due to power consumption of logic and DRAM layers.

- In our modeling, the maximum PIM offloading rate is 6.5 PIM ops/ns
- A high offloading rate could reduce memory performance for cool down
Performance Trade-off of PIM

- Higher BW benefits → Better performance
- Higher DRAM temperature → Low memory performance

PIM intensity needs to be controlled!!
CoolPIM

Controls PIM Intensity with Thermal Consideration
We propose two methods for GPU/HMC

1) A SW mechanism with no hardware changes
2) A HW mechanism with changes in GPU architectures

Dynamic source throttling based on thermal warning messages from HMC

- Thermal warning -> lowers PIM intensity -> reduces internal temperature of HMC
GPU runtime implements some components to control PIM intensity:

- **PIM Token Pool (PTP)**
  - # of maximum thread blocks that are allowed to use PIM functionality

- **Thread Block Manager**
  - Check PTP and launch PIM code if tokens are available

- **Initialization**
  - Estimate the initial PTP size based on static analysis at compile time
The GPU compiler generates PIM-enabled and non-PIM kernels at compile time

- Source-to-source translation
- IR-to-IR translation

```c
Void cuda_kernel(arg_list)
{
    for (int i=0; i<end; i++)
    {
        uint addr = addrArray[i];
        PIM_Add(addr, 1);
    }
}
```

```c
void cuda_kernel_np(arg_list)
{
    for (int i=0; i<end; i++)
    {
        uint addr = addrArray[i];
        cuda atomicAdd(addr, 1);
    }
}
```

Original PIM Code

Shadow Non-PIM Code
Hardware-Based Throttling

PIM Control Unit

- Controls # of PIM-enabled warps
- Performs dynamic binary translation
- See the paper for detail!

<table>
<thead>
<tr>
<th>Type</th>
<th>PIM Instruction</th>
<th>Non-PIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>Signed Add</td>
<td>atomicAdd</td>
</tr>
<tr>
<td>Bitwise</td>
<td>Swap, bit write</td>
<td>atomicExch</td>
</tr>
<tr>
<td>Boolean</td>
<td>AND, OR</td>
<td>atomicAND/OR</td>
</tr>
<tr>
<td>Comparison</td>
<td>CAS-equal/greater</td>
<td>atomicCAS/Max</td>
</tr>
</tbody>
</table>
Evaluation
Methodology

Thermal Evaluation
- Temp Measurement: Real HMC 1.1 Platform
- Thermal Modeling: HMC 2.0 using 3D-ICE
- Power & Area: Synopsys (28nm/50nm CMOS)

Performance Evaluation
- MacSim w/ VaultSim

Benchmark
- GraphBIG benchmark with LDBC dataset
  - BFS, SSSP, PageRank, etc...

![Diagram showing methodological flowchart]
Performance

Speedup over baseline (Non-Offloading)

- **Naïve/SW/HW**: using a commodity-server active heat sync
- **Ideal Thermal**: with unlimited cooling

On average, CoolPIM (SW/HW) improves performance by 1.21x/1.25x!
**Thermal Analysis**

**PIM Offloading Rate**

- Naïve: 3~4 op/ns → Temperature goes beyond the normal operating region.
- CoolPIM: 1.3 op/ns → No memory performance slowdown

CoolPIM maintains peak DRAM temperature within normal operating temp!
Conclusion
Conclusion

Observation: PIM integration requires careful thermal consideration
  • Naive PIM offloading may cause a thermal issue and degrades overall system performance

CoolPIM: Source throttling techniques to control PIM intensity
  • Keeps HMC ”Cool” to avoid thermal-triggered memory performance degradation

Results: CoolPIM improves performance by 1.37x over naïve offloading
  • 1.2x over non-offloading on average
Thank You
Backup
## Typical Cooling Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Thermal Resistance</th>
<th>Cooling Power*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passive heat sink</td>
<td>4.0 °C/W</td>
<td>0</td>
</tr>
<tr>
<td>Low-end active heat sink</td>
<td>2.0 °C/W</td>
<td>1x</td>
</tr>
<tr>
<td>Commodity-server active heat sink</td>
<td>0.5 °C/W</td>
<td>104x</td>
</tr>
<tr>
<td>High-end heat sink</td>
<td>0.2 °C/W</td>
<td>380x</td>
</tr>
</tbody>
</table>

* We assume the same plate-fin heat sink model for all configurations.
Thermal Model Validation

Validate our thermal evaluation environment

• Model HMC 1.1 temperature and compare with measurements

![Bar Chart]

- **Surface (measured)**
- **Die (estimated)**
- **Die (modeling)**

<table>
<thead>
<tr>
<th>Temperature (°C)</th>
<th>Low -end</th>
<th>High -end</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Evaluation Configuration

<table>
<thead>
<tr>
<th>Component</th>
<th>Configuration</th>
</tr>
</thead>
</table>
| **Host**  | GPU, 16 PTX SMs, 32 threads/warp, 1.4GHz  
16KB private L1D and 1MB 16-way L2 cache |
| **HMC**   | 8 GB cube, 1 logic die, 8 DRAM dies 32 vaults, 512 DRAM banks  
tCL=tRCD=tRP=13.75ns, tRAS=27.5ns  
4 links per package, 120 GB/s per link  
80 GB/s data bandwidth per link  
DRAM Temp. phase: 0-85 °C, 85-95 °C, 95-105 °C  
20% DRAM freq reduction (high temp. phases) |
Bandwidth Consumption

- Bandwidth consumption normalized to baseline (non-offloading)

![Bandwidth Consumption Chart]

- Non-Offloading
- Naïve-Offloading
- CoolPIM (SW)
- CoolPIM (HW)
Software-Based Throttling

Interrupt Handler → PIM Token Pool → CUDA Blk. Manager

HMC → GPU SMs

Reduce size → Issue token → Select Blk.

Offloading → Thermo warning

Launch CUDA Blk. → PIM Code

Launch CUDA Blk. → Non-PIM Code
# Hardware vs Software

<table>
<thead>
<tr>
<th>Type</th>
<th>Software-Based</th>
<th>HW-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Granularity</td>
<td>Thread Blocks</td>
<td>Warps</td>
</tr>
<tr>
<td>Control Delay</td>
<td>Long Delay</td>
<td>Short Delay</td>
</tr>
<tr>
<td>Design Complexity</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>
Hardware vs Software

Graph showing PIM Rate (op/ns) vs Time (ms) for Naïve Offloading, CoolPIM (SW), and CoolPIM (HW).