AnalyzeThat:
A Programmable Shared-Memory System
for Processing-In-Memory Devices

Presentation by Youngjae Kim
May 16, 2017

Sangkuen Lee, Hyogi Sim, Sudharshan S. Vazhkudai
Oak Ridge National Laboratory

Youngjae Kim
Sogang University
Motivation

• What is PIM?
  – concept of integrating processing units (cores) with memory device
  – to reduce memory latency and increase memory bandwidth

• Why PIM again?
  – Enabling technologies now available (e.g., 3D memory)
  – Potential advantages for processing big data in terms of both energy efficiency and processing time (higher throughput, low latency, ..)

Micron’s Automata Processor

AMD’s 3D stacking technology
Motivation (cont’d)

• Exascale Characteristics and PIM
  – U.S. Department of Energy’s deployment plan of O(100) petaflops systems (e.g., SUMMIT at ORNL)
  – *Data movement cost* between the deep memory tiers \( \approx \) *Computation cost*
  – *PIM and processing near memory* are expected to help alleviate this concern

• Main Challenge
  – To integrate PIM architectures into extant user application software
  – How to help users to easily and properly distribute *data and tasks* to fully take advantage of *data locality* and *parallelism* of the PIM devices
AnalyzeThat 3 Goals

• Easy Programming Interface for PIM-augmented systems
  – Easy and effective programming interface
  – Hide system-specific details (e.g., memory management, thread operation, non-uniform memory access latency)

• Reduce Data Movement
  – Data movement cost minimization between the host and the PIM devices using *PIM-specific information*

• Programming Flexibility and Generality
  – More controls (e.g., manually place a piece of data object or offload a task to a specific PIM device) for advanced programmers
AnalyzeThat Overview

PADS (PIM-Aware Data Structure) and Runtime
- Key-value container for PIM + parallel operations
- Dynamically decision makings on data and task load distribution

Low-level PIM Library
- Direct control of PIM devices for advanced programmers

PIM Device Driver
- Communication path between PIM HW and SW components
- PIM-specific internal information (e.g., wearout, data load) can be queried

Array of PIM Devices + DRAM/CPU
Single shared memory address space

Data-Intensive Applications

High-Level PADS Library

Runtime

Low-Level PIM Library

PIM Device Driver

Shared Memory Abstraction

Low-Level Framework

Userspace

OS

Hardware Architecture

CPU

CPU

CPU

Main Memory (DRAM)

PCle Switch

Coherent Memory Access

PIM Array

PC

PIM Memory

PC

PIM Memory

PC

PIM Memory

PC

PIM Memory
AnalyzeThat Low-level Framework

• Hardware Architecture
  – PIM core
    • A fully programmable low-power processor PIM core similar to ARM
    • contains own hardware cache, MMU (Memory Management Unit), and firmware to control the internal hardware
  – Connection between PIM devices and the Host
    • via a fast switch interconnect that supports cache coherent accesses across heterogeneous memory devices (e.g., Cache Coherent Interconnect for Accelerators (CCIX))

• Shared memory abstraction
  – PIM core can access not only its own local PIM memory but also other remote PIM memories and DRAM on the host
  – No system software or OS modification required
AnalyzeThat Low-level Framework

• PIM Device Driver
  – assists user space programs to allocate memory space and to access the programmable registers via memory-mapped I/O interface
    • memory allocation on a specific PIM device
    • initiate a task execution on a PIM device
    • access PIM-specific information (e.g., PIM core utilization, memory usage)
  – /proc

• Low-level PIM Library
  – layered atop the PIM device driver
  – Resembles POSIX dynamic memory and pthread functions

<table>
<thead>
<tr>
<th>Low-level</th>
<th>void *pimmalloc(size_t size, int pim)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>void pimfree(void *addr)</td>
</tr>
<tr>
<td></td>
<td>int pimexec_exec(pimexec_data_t *pe)</td>
</tr>
<tr>
<td></td>
<td>int pimexec_wait(pimexec_data_t *pe)</td>
</tr>
</tbody>
</table>

| Allocate size bytes of memory in PIM device pim |
| Frees memory of address addr |
| Initiate offloading of a user-defined function |
| Block the current thread until the execution completes |
AnalyzeThat PADS and Runtime

• PADS (PIM-Aware Data Structure)
  – Key-value container data structure i.e., data is stored as key-value pairs
  – Internal n sub-containers distributed across PIM devices (each SC associated with a PIM device)

• PADS Operations
  – Run on the associated PADS object
  – Resemble Map-Reduce Programming Interface
  – Data placement & task execution delegated to runtime

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void PADS.import(char* file, parser_t* pf)</td>
<td>Import data from a file using a parser function pf</td>
</tr>
<tr>
<td>void PADS.map(PADS&amp; out, mapper_t* mf, void* arg)</td>
<td>Performs a user-defined map function mf and stores results in out</td>
</tr>
<tr>
<td>void PADS.reduce(PADS&amp; out, reducer_t* rf, void* arg)</td>
<td>Performs a user-defined reduce function rf and stores results in out</td>
</tr>
<tr>
<td>void PADS.export(char* file)</td>
<td>Export data into file</td>
</tr>
</tbody>
</table>
AnalyzeThat PADS and Runtime (Cont’d)

• Example Application Code

```c
// (a) Group-by-Aggregation and Aggregation
void aggMap(char *k, char *v, PADS& t) {
    strcpy(new_val, v);
    strcat(new_val, "1");
    t.insert(k, new_val); // "A" instead of k for AG
}
char *aggReduce(char *k, char *v, char *reduced) {
    tokenizer(v, head, tail, ",", " ");
    sum = stoi(sum) + stoi(head)
    cnt = stoi(cnt) + stoi(int)
    avg = (double) sum / cnt;
    sprintf(reduced, "%d,%d,%f", sum, int, avg);
}
int main(void) {
    PADS data, mapped, result;
    data.import("input.txt");
    data.map(mapped, aggMap);
    mapped.reduce(result, aggReduce);
}

// (b) Grep
void grepMap(char *k, char *v, char *arg, PADS& t) {
    if (strstr(v, arg))
        t.insert(k, v);
}
int main(void) {
    PADS data, result;
    data.import("input.txt");
    data.map(result, grepMap, "bob");
}
```

// (c) Word-Count
```c
void wordCountMap(char *k, char *v, PADS& t) {
    while ((token = strsep(&v, ",")) != NULL)
        t.insert(k, 1);
}
int main(void) {
    PADS data, mapped, result;
    data.import("input.txt");
    data.map(mapped, wordCountMap);
    mapped.reduce(result, sumByKey);
}
```

![Word Count Example Diagram]

---

OAK RIDGE National Laboratory
AnalyzeThat Runtime: Task Manager

- Task Manager
  - Responsible for the execution of the PADS application program by utilizing the PIM-architecture spawning a PIM thread on every PIM device
  - Local Reduce (LR)
    - each PIM thread performs the reduce() operation locally by creating an intermediate PADS
    - To optimize remote PIM memory accesses

PIM 1
A, 1
A, 1
B, 1
B, 1
C, 1

PIM 2
C, 1
C, 1
C, 1
D, 1
E, 1

PIM 3
E, 1
D, 1
D, 1
A, 1
C, 1

PIM 1
A, 3

PIM 2
B, 2
C, 3

PIM 3
D, 3
E, 2

Global Reduce

PIM 1
A, 1
A, 1
B, 1
B, 1
C, 1

PIM 2
C, 1
C, 1
D, 1
D, 1
E, 1

PIM 3
E, 1
D, 1
D, 1
A, 1
C, 1

PIM 1
A, 2
A, 1
B, 2
B, 1
C, 1

PIM 2
C, 3
D, 1
E, 1

PIM 3
E, 1
D, 2
A, 1
C, 1

Local Reduce + Global Reduce

Oak Ridge National Laboratory
AnalyzeThat Runtime: Data Placement Manager

• Data Placement Strategies
  – Balancing the load evenly across the PIM array
  – Grouping key-value pairs in a single PIM device based on their keys
  – Storing output key-value pairs in a local PIM memory.

• Algorithms
  – Round-Robin (RR): a PIM device for storing data in a circular, round-robin order
  – Local-Assignment (LA): always places a key-value pair to the same PIM device where it is requested
  – Hashing (HS): uses a hash function to select a PIM device to store a key-value pair
  – Dynamic (DY): Using HS and LA while avoiding the mostly used Device to store data, wear-out awareness can be enabled
Evaluation

• **PIM device Emulation**
  – pre-allocating the system memory and binding threads to cores
  – introduced delays (using TSC) according to the memory access types
    • i.e., remote memory is 32.2% slower than local memory.
    • DRAM access is 4 slower than PIM’s local memory access

• **Testbed**
  – Our test machine comprised of two 1.8 GHz Intel Xeon E5-2603 processors, each with four cores, 64 GB RAM and ran the RedHat Enterprise Linux 6.5 with the 3.1.22 kernel.
• AnalyzeThat with a single PIM (PIM(1)) is 15-30% slower than the host-based approach with a single core (Host(1)).

• AnalyzeThat with two PIM devices (PIM(2)) outperforms the host-based approach by 20-30%.

• Performance shows different trends for workloads, Dynamic + LR shows the most stable performance
Evaluation (cont’d)

- Dynamic Placement Algorithm, Wearout awareness test
  - Initial wearout setup
    - #PIM=7, Local Reduce Enabled
      - When wearout-awareness (WA) enabled DY avoids the data placement to the PIM device with the high wearout-level.
      - Two applications (Group-by Aggregation, Word Count) tested

<table>
<thead>
<tr>
<th></th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total (MB)</td>
<td>1000.00</td>
<td>1000.00</td>
<td>1000.00</td>
</tr>
<tr>
<td>Maximum (MB)</td>
<td>142.86</td>
<td>250.00</td>
<td>666.67</td>
</tr>
<tr>
<td>Minimum (MB)</td>
<td>142.86</td>
<td>35.71</td>
<td>0.00</td>
</tr>
<tr>
<td>Standard Deviation (MB)</td>
<td>0.00</td>
<td>77.15</td>
<td>243.98</td>
</tr>
</tbody>
</table>

- Enabling WA did not largely affect the Runtime, but it can significantly reduce the CV
Conclusions

• PIM architecture
  – Several advantages to data analysis applications
  – High entry barrier to programmers due to complexity and new aspects

• AnalyzeThat provides a **high-level** data (Key-value container) and programming **abstraction** (Built-in Map-Reduce like operations) and makes intelligent decisions on
  • Dynamic data placements
  • Parallel task executions

• We implemented representative data analysis applications and validated the developed capabilities
Contact: Youngjae Kim (youkim@sogang.ac.kr)
SOGANG University, Seoul South Korea

Matt Sangkeun Lee (lee4@ornl.gov)
Oak Ridge National Laboratory