

Hands-on Lab

Programming and Understanding a Real Processing-in-Memory Architecture

Dr. Juan Gómez Luna

Professor Onur Mutlu

Real PIM Tutorial: Hands-on Lab

ASPLOS 2023 TUTORIAL: REAL-WORLD PROCESSING-IN-MEMORY SYSTEMS FOR MODERN WORKLOADS.
MARCH 26, 2023

1/7

Programming and Understanding a Real Processing-in-Memory Architecture

INSTRUCTORS: DR. JUAN GÓMEZ LUNA, PROF. ONUR MUTLU

1. Introduction

In this lab, you will work hands-on with a real processing-in-memory (PIM) architecture. You will program the UPMEM PIM architecture [1, 2, 3, 4] for several workloads and will experiment with them. Your main goals are (1) to become familiar with the UPMEM PIM system organization (as an example of real-world memory-centric computing system), (2) to understand the UPMEM programming model and write your own code, and (3) to understand the microarchitecture and instruction set architecture (ISA) of UPMEM's PIM core (called *DRAM Processing Unit, DPU*).

As we introduced in this tutorial, the UPMEM PIM architecture is composed of multiple DPUs (up to 2,560), each of which has access to its own DRAM bank (called *Main RAM, MRAM*) and its own scratchpad memory (called *Working RAM, WRAM*). You can find a full description of the UPMEM PIM system in [3, 4].

2. Your Task 0/4: Accessing the UPMEM PIM Server

UPMEM has granted us with remote access to servers with UPMEM DIMMs in a datacenter.

Our username is: ethasplos23 and we are part of the group upmem0063 (ETH ASPLOS 2023 team). You can download the SSH private key used to connect the machines from here: <https://events.safari.ethz.ch/asplos-pim-tutorial/lib/exe/fetch.php?media=upmemcloud.ethasplos23.zip> (download and unzip!)

Put the following base configuration in your `.ssh/config` file:

```
Host upmemcloud*
  User ethasplos23
  Hostname %h.cloud.upmem.com
  IdentityFile ~/.ssh/upmemcloud.ethasplos23
  StrictHostKeyChecking no
  UserKnownHostsFile=/dev/null
```

You can connect to the booked machine anytime until 6am (Vancouver time) on Monday, March 27, 2023.

The booked machine for this period is upmemcloud5 with '20 UPMEM-P21'. You can connect to it by doing: `ssh upmemcloud5`, if you have the private SSH key and the `.ssh/config` file provided above.

The machine is installed with the latest and greatest UPMEM SDK version (also available on <https://sdk.upmem.com>). As an introduction, the public demonstration program doing a trivial checksum in parallel on one DPU can be run by doing:

```
git clone https://github.com/upmem/dpu.demo.git
cd /dpu.demo/checksum
NR_DPUS=1 make test
```

Please read the entire Section 2 before you access the server.

In summary, the steps are:

1. Paste the configuration into `.ssh/config`.
2. Copy the private key `upmemcloud.ethasplos23` to your `.ssh` folder. You may need to change permissions, as indicated in Section 2.1.
3. `ssh upmemcloud5` from the terminal. Note that the server is already reserved for us. No booking is needed.

How to Access the UPMEM PIM Server?

1. Paste the configuration into `.ssh/config`

```
Host upmemcloud*
```

```
User ethasplos23
```

```
Hostname %h.cloud.upmem.com
```

```
IdentityFile ~/.ssh/upmemcloud_ethasplos23
```

```
StrictHostKeyChecking no
```

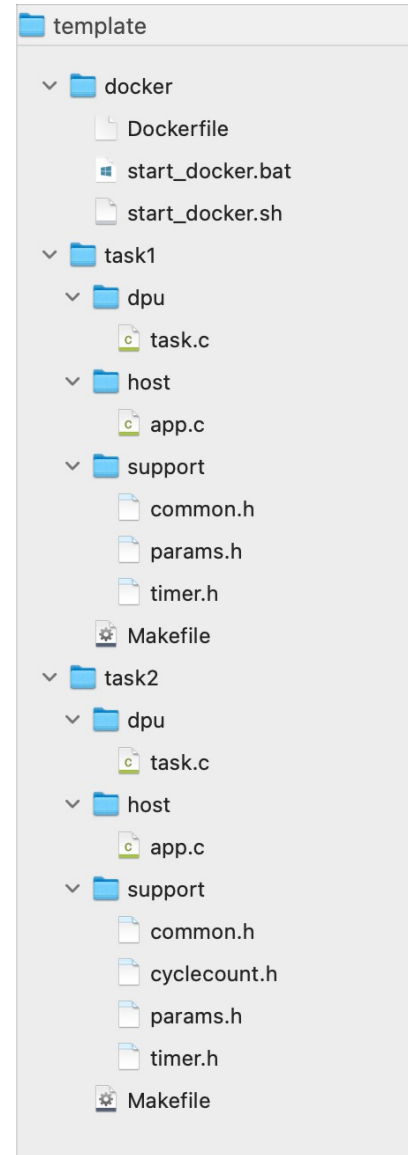
```
UserKnownHostsFile=/dev/null
```

2. Copy the private key `upmemcloud_ethasplos23` to your `.ssh` folder. You may need to change permissions

3. `ssh upmemcloud5` from the terminal

Template Files

- Contain templates for task 1 and task 2
- Task 2's template can be used for the remaining tasks



Task 1: CPU-DPU and DPU-CPU Transfers

- Use serial, parallel, and broadcast transfers

Your tasks are as follows:

1. Write a host program that exercises all types of data transfers between the host main memory and one or multiple MRAM banks. Concretely, there are three types of data transfers [2]: (1) serial, (2) parallel, and (3) broadcast. Serial and parallel transfers move data from main memory to the MRAM banks or vice versa. Broadcast transfers can only happen from the main memory to the MRAM banks.
2. Evaluate all different types of data transfers for data transfers of size (1) 1MB, (2) 24MB, (3) 48MB per DPU. Use different numbers of DPUs between 1 and 64.

Serial Transfers

- `dpu_copy_to()`;
- `dpu_copy_from()`;
- We transfer (part of) a buffer to/from each DPU in the `dpu_set`
- `DPU_MRAM_HEAP_POINTER_NAME`: Start of the MRAM range that can be freely accessed by applications
 - We do not allocate MRAM explicitly

```
DPU_FOREACH(dpu_set, dpu) {
    DPU_ASSERT(dpu_copy_to(dpu, DPU_MRAM_HEAP_POINTER_NAME,
        input_size_dpu_Bytes * sizeof(T),
        bufferA + input_size_dpu_Bytes * i,
        input_size_dpu_Bytes * sizeof(T)));
    DPU_ASSERT(dpu_copy_from(dpu, DPU_MRAM_HEAP_POINTER_NAME,
        input_size_dpu_Bytes * sizeof(T),
        bufferB + input_size_dpu_Bytes * i,
        input_size_dpu_Bytes * sizeof(T)));
}
```

Offset within MRAM Pointer to main memory Transfer size

SAFARI

73

Parallel Transfers

- We push different buffers to/from a DPU set in one transfer
 - All buffers need to be of the same size
- First, prepare (`dpu_prepare_xfer`); then, push (`dpu_push_xfer`)
- Direction:
 - `DPU_XFER_TO_DPU`
 - `DPU_XFER_FROM_DPU`

```
DPU_FOREACH(dpu_set, dpu) {
    DPU_ASSERT(dpu_prepare_xfer(dpu,
        bufferA + input_size_dpu_Bytes * i,
        input_size_dpu_Bytes * sizeof(T)));
    DPU_ASSERT(dpu_push_xfer(dpu_set,
        DPU_XFER_TO_DPU, DPU_MRAM_HEAP_POINTER_NAME,
        input_size_dpu_Bytes * sizeof(T), DPU_XFER_DEFAULT));
    DPU_FOREACH(dpu_set, dpu) {
        DPU_ASSERT(dpu_prepare_xfer(dpu,
            bufferB + input_size_dpu_Bytes * i,
            input_size_dpu_Bytes * sizeof(T)));
        DPU_ASSERT(dpu_push_xfer(dpu_set,
            DPU_XFER_FROM_DPU, DPU_MRAM_HEAP_POINTER_NAME,
            input_size_dpu_Bytes * sizeof(T), DPU_XFER_DEFAULT));
    }
}
```

Pointer to main memory Offset within MRAM Transfer size Direction

SAFARI

74

Broadcast Transfers

- `dpu_broadcast_to()`;
 - Only CPU to DPU
- We transfer the same buffer to all DPUs in the `dpu_set`

```
DPU_ASSERT(dpu_broadcast_to(dpu_set, DPU_MRAM_HEAP_POINTER_NAME,
    bufferA, input_size_dpu * sizeof(T), DPU_XFER_DEFAULT));
```

Pointer to main memory Transfer size

SAFARI

75

Task 2: AXPY

Your tasks are as follows:

1. Write a DPU kernel that executes the AXPY operation ($y = y + \alpha \times x$) [5] on every element of a vector. You have to (1) transfer two input vectors, Y and X , to the MRAM bank/s, (2) perform the AXPY operation with a variable number of tasklets, (3) write the results to the output vector, Y , and (4) transfer the output vector back to the host main memory.

- VA is a good reference code for this task

Programming a DPU Kernel (I)

- Vector addition

```
1 // Vector addition kernel
2 int main_kernel1() {
3     unsigned int tasklet_id = me()
4     uint32_t input_size_dpu_bytes = DPU_INPUT_ARGUMENTS.size; // Input size per DPU in bytes
5     uint32_t input_size_dpu_bytes_transfer = DPU_INPUT_ARGUMENTS.transfer_size; // Transfer input size per DPU in bytes
6
7     // Address of the current processing block in MRAM
8     uint32_t base_tasklet = tasklet_id << BLOCK_SIZE_LOG2;
9     uint32_t mram_base_addr_A = (uint32_t)DPU_MRAM_HEAP_POINTER;
10    uint32_t mram_base_addr_B = (uint32_t)(DPU_MRAM_HEAP_POINTER + input_size_dpu_bytes_transfer);
11
12    // Initialize a local cache to store the MRAM block
13    T *cache_A = (T *) mem_alloc(BLOCK_SIZE);
14    T *cache_B = (T *) mem_alloc(BLOCK_SIZE);
15
16    for(unsigned int byte_index = base_tasklet; byte_index < input_size_dpu_bytes; byte_index += BLOCK_SIZE * NR_TASKLETS){
17        // Bound checking
18        uint32_t l_size_bytes = (byte_index + BLOCK_SIZE >= input_size_dpu_bytes) ? (input_size_dpu_bytes - byte_index) : BLOCK_SIZE;
19
20        // Load cache with current MRAM block
21        mram_read((__mram_ptr void const*)(mram_base_addr_A + byte_index), cache_A, l_size_bytes);
22        mram_read((__mram_ptr void const*)(mram_base_addr_B + byte_index), cache_B, l_size_bytes);
23
24        // Computer vector addition
25        vector_addition(cache_B, cache_A, l_size_bytes >> DIV);
26
27        // Write cache to current MRAM block
28        mram_write(cache_B, (__mram_ptr void*)(mram_base_addr_B + byte_index), l_size_bytes);
29    }
30    return 0;
31 }
```

SAFARI

87

Task 3: Operations and Datatypes

Your tasks are as follows:

1. Modify your AXPY DPU kernel to make it a vector addition ($y = y + x$) and to support other operations besides addition (i.e., subtraction, multiplication, division).
 2. Evaluate the performance of your new kernel for different operations (addition, subtraction, multiplication, division) and data types (char, short, int, long long int, float, double).
- You will observe significant variations in arithmetic throughput for different operations and datatypes

Task 4: Vector Reduction

Your tasks are as follows:

1. Your vector reduction DPU kernel should have four different versions: (1) final reduction with a single tasklet, (2) final tree-based reduction with barriers, (3) final tree-based reduction with handshakes, (4) final reduction with mutexes.

- Performance differences due to the final reduction step

Final Reduction

- A single tasklet can perform the final reduction

```
1 v for(unsigned int byte_index = base_tasklet; byte_index < input_size_dpu_bytes; byte_index += BLOCK_SIZE * NR_TASKLETS){
2
3     // Bound checking
4     uint32_t l_size_bytes = (byte_index + BLOCK_SIZE >= input_size_dpu_bytes) ? (input_size_dpu_bytes - byte_index) : BLOCK_SIZE;
5
6     // Load cache with current WRAM block
7     mram_read((__mram_ptr void const*)(mram_base_addr_A + byte_index), cache_A, l_size_bytes);
8
9     // Reduction in each tasklet
10    l_count += reduction(cache_A, l_size_bytes >> DIV); Accumulate in a local sum
11
12 }
13
14 // Copy local count to shared array in WRAM
15 message[tasklet_id] = l_count; Copy local sum into WRAM
16
17
18 // Single-thread reduction
19 // Barrier
20 barrier_wait(&my_barrier); Barrier synchronization
21
22 if(tasklet_id == 0){
23     #pragma unroll
24     for (unsigned int each_tasklet = 1; each_tasklet < NR_TASKLETS; each_tasklet++){
25         message[0] += message[each_tasklet]; Sequential accumulation
26     }
27
28 // Total count in this DPU
29 result->t_count = message[0];
30 }
```

SAFARI

94

Hands-on Lab

Programming and Understanding a Real Processing-in-Memory Architecture

Dr. Juan Gómez Luna

Professor Onur Mutlu