ISCA 2023 Tutorial

Real-world Processing-in-Memory Systems for Modern Workloads

Hands-on Lab

Programming and Understanding a Real Processing-in-Memory Architecture

Dr. Juan Gómez Luna

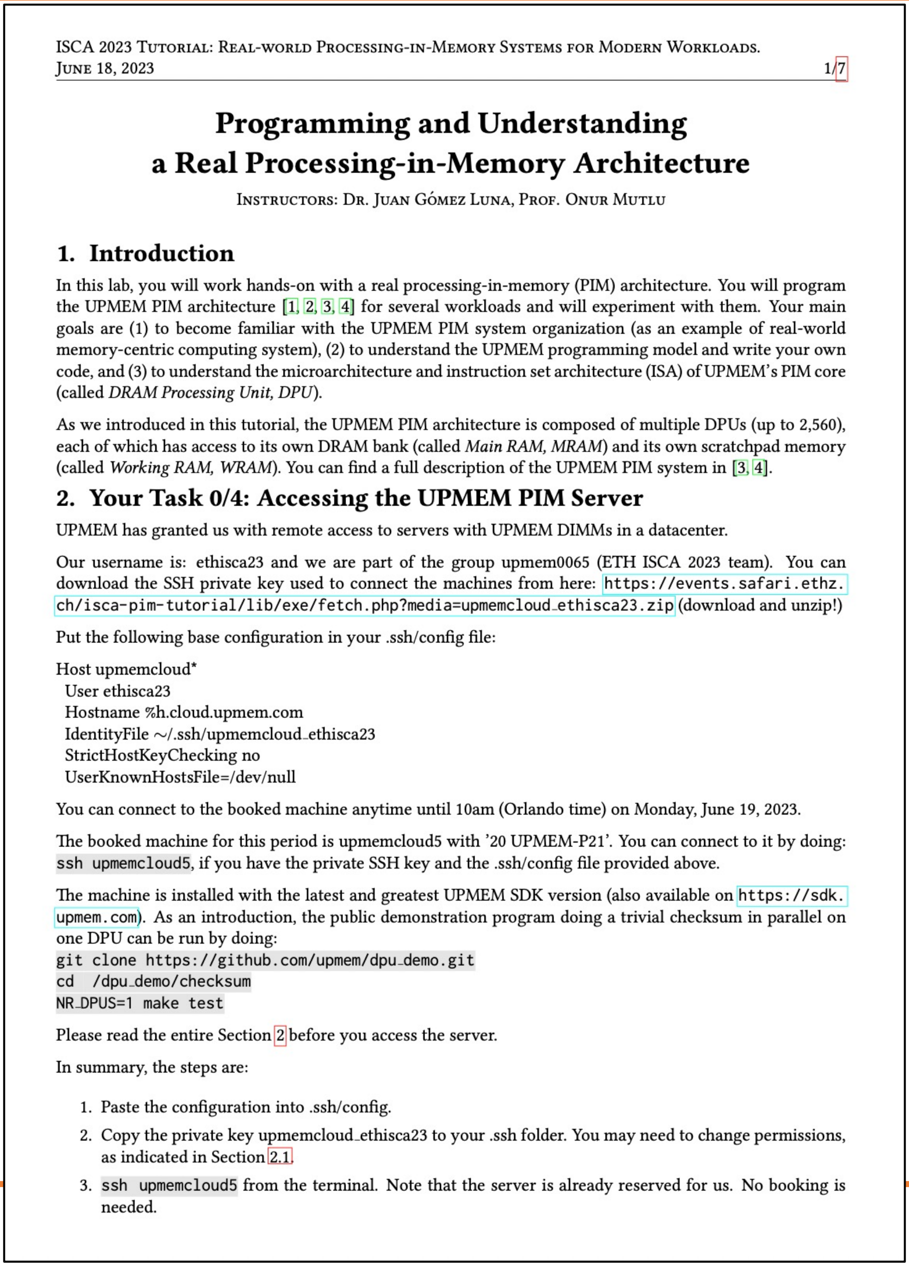
Professor Onur Mutlu

ETH Zürich

SAFARI

Sunday, June 18, 2023

Real PIM Tutorial: Hands-on Lab



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: ethisca23 and we are part of the group upmem0065 (ETH ISCA 2023 team). You can download the SSH private key used to connect the machines from here: `https://events.safari.ethz.ch/isca-pim-tutorial/lib/exe/fetch.php?media=upmemcloud_ethisca23.zip` (download and unzip!)

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

```
Host upmemcloud*
 User ethisca23
 Hostname %h.cloud.upmem.com
 IdentityFile ~/.ssh/upmemcloud_ethisca23
 StrictHostKeyChecking no
 UserKnownHostsFile=/dev/null
```

You can connect to the booked machine anytime until 10am (Orlando time) on Monday, June 19, 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_ethisca23 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

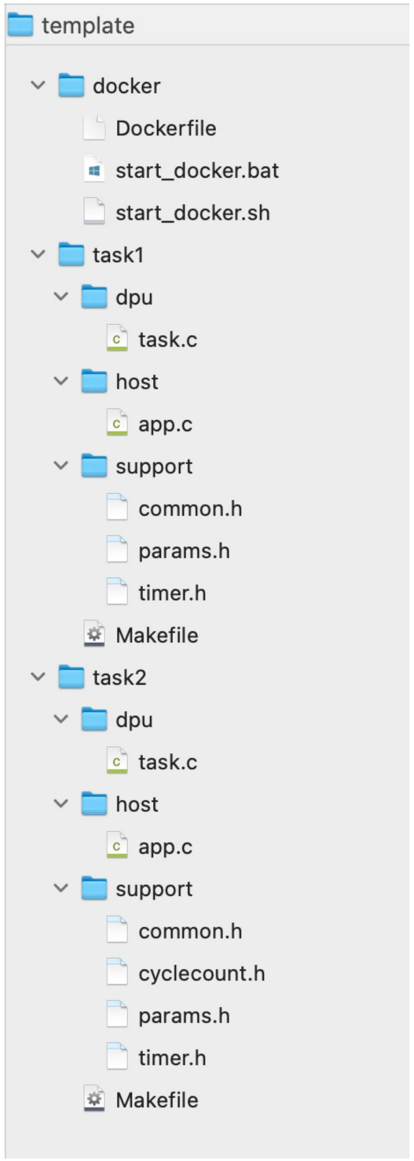
```
Host upmemcloud*
    User ethisca23
    Hostname %h.cloud.upmem.com
    IdentityFile ~/.ssh/upmemcloud_ethisca23
    StrictHostKeyChecking no
    UserKnownHostsFile=/dev/null
```

2. Copy the private key `upmemcloud_ethisca23` to your .ssh folder. You may need to change permissions

3. `ssh upmemcloud5` from the terminal

Template Files

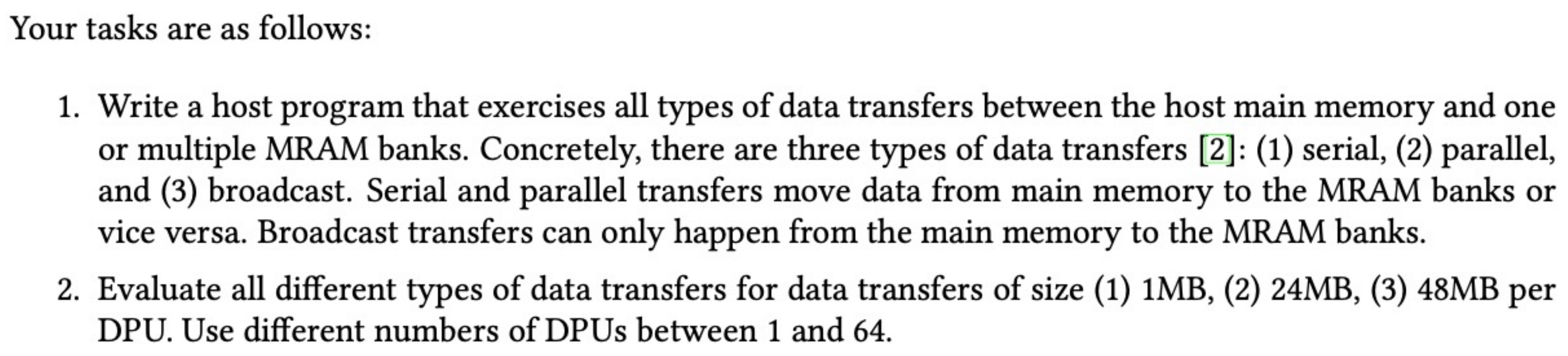
- Contain templates for task 1 and task 2
- Task 2's template can be used for the remaining tasks



```
template
  docker
    Dockerfile
    start_docker.bat
    start_docker.sh
  task1
    dpu
      task.c
    host
      app.c
    support
      common.h
      params.h
      timer.h
    Makefile
  task2
    dpu
      task.c
    host
      app.c
    support
      common.h
      cyclecount.h
      params.h
      timer.h
    Makefile
```

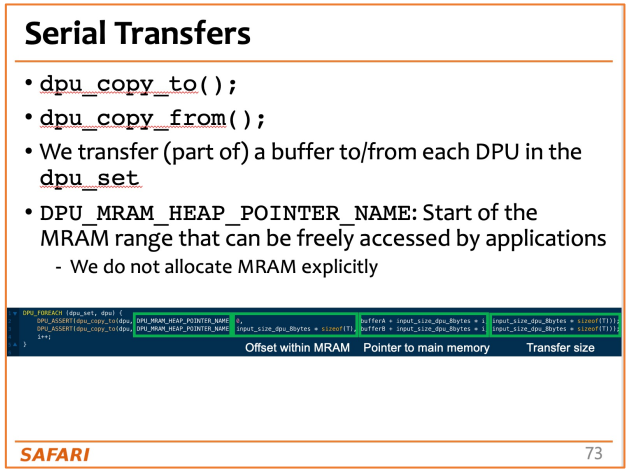
Task 1: CPU-DPU and DPU-CPU Transfers

- Use serial, parallel, and broadcast transfers



Your tasks are as follows:

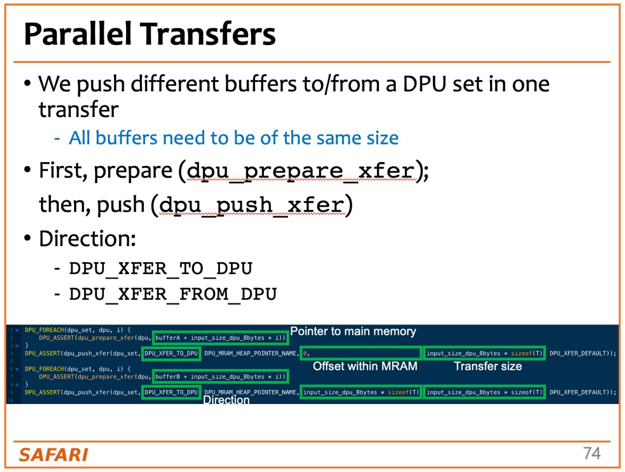
1. Write a host program that exercises all types of data transfers between the host main memory and one or multiple MRAM banks. Concretely, there are three types of data transfers [2]: (1) serial, (2) parallel, and (3) broadcast. Serial and parallel transfers move data from main memory to the MRAM banks 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

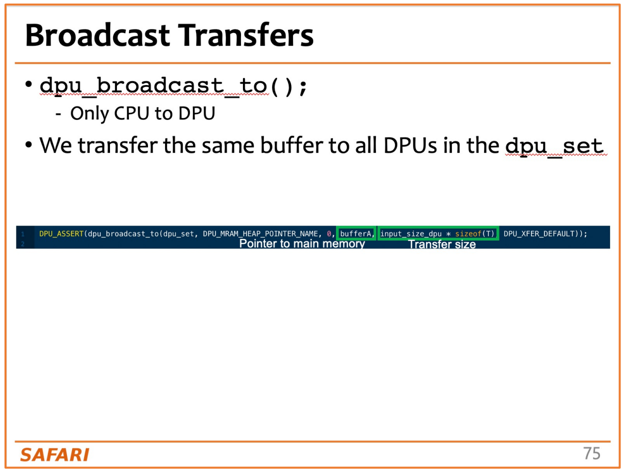
```
DPU_FOREACH (dpu_set, dpu) {
    DPU_ASSERT(dpu_copy_to(dpu, DPU_MRAM_HEAP_POINTER_NAME 0,                                bufferA + input_size_dpu_8bytes * i input_size_dpu_8bytes * sizeof(T)));
    DPU_ASSERT(dpu_copy_to(dpu, DPU_MRAM_HEAP_POINTER_NAME input_size_dpu_8bytes * sizeof(T), bufferB + input_size_dpu_8bytes * i input_size_dpu_8bytes * sizeof(T)));
    i++;
}
```



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, i) {
    DPU_ASSERT(dpu_prepare_xfer(dpu, bufferA + input_size_dpu_8bytes * i))
}
DPU_ASSERT(dpu_push_xfer(dpu_set, DPU_XFER_TO_DPU DPU_MRAM_HEAP_POINTER_NAME, 0,                                 input_size_dpu_8bytes * sizeof(T) DPU_XFER_DEFAULT));

DPU_FOREACH(dpu_set, dpu, i) {
    DPU_ASSERT(dpu_prepare_xfer(dpu, bufferB + input_size_dpu_8bytes * i))
}
DPU_ASSERT(dpu_push_xfer(dpu_set, DPU_XFER_TO_DPU DPU_MRAM_HEAP_POINTER_NAME, input_size_dpu_8bytes * sizeof(T) input_size_dpu_8bytes * sizeof(T) DPU_XFER_DEFAULT));
```



Broadcast Transfers

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

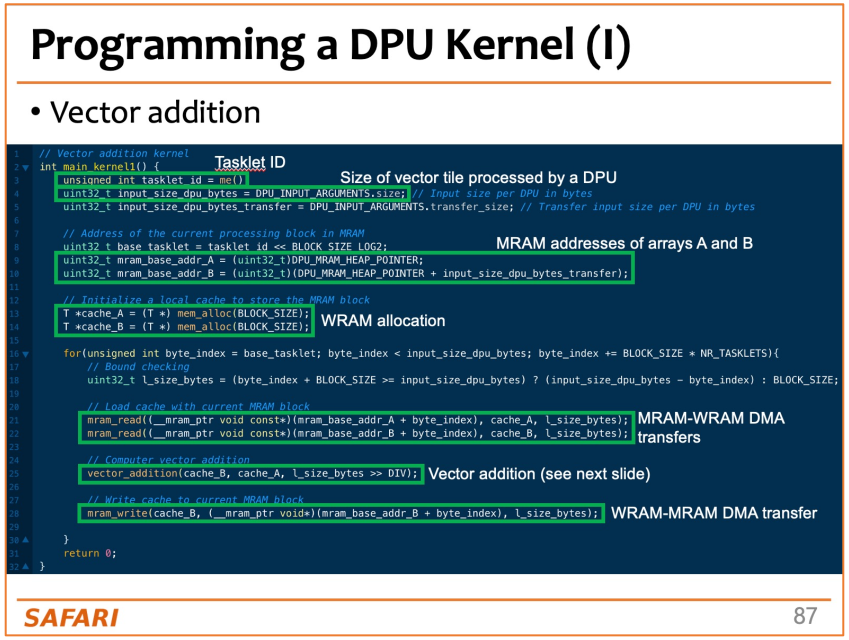
```
DPU_ASSERT(dpu_broadcast_to(dpu_set, DPU_MRAM_HEAP_POINTER_NAME, 0, bufferA, input_size_dpu * sizeof(T) DPU_XFER_DEFAULT));
```

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)

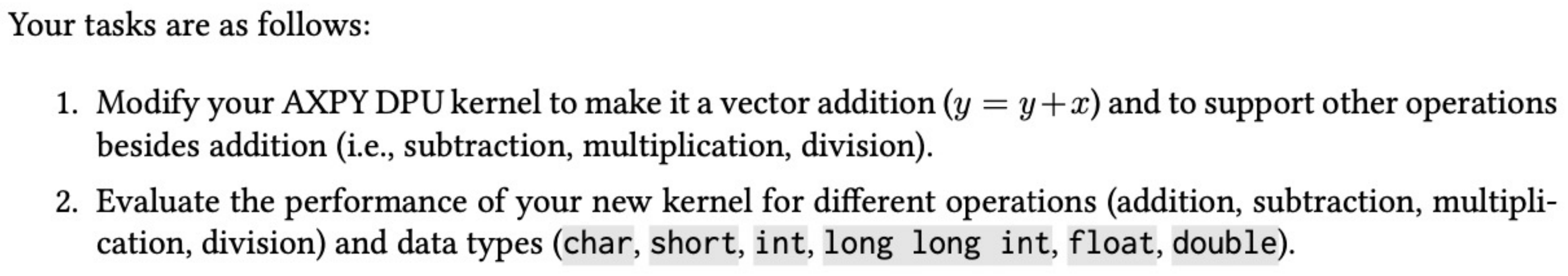
- Vector addition

```
// Vector addition kernel
int main_kernel1() {
    unsigned int tasklet_id = me()
    uint32_t input_size_dpu_bytes = DPU_INPUT_ARGUMENTS.size; // Input size per DPU in bytes
    uint32_t input_size_dpu_bytes_transfer = DPU_INPUT_ARGUMENTS.transfer_size; // Transfer input size per DPU in bytes

    // Address of the current processing block in MRAM
    uint32_t base_tasklet = tasklet_id << BLOCK_SIZE_LOG2;
    uint32_t mram_base_addr_A = (uint32_t)DPU_MRAM_HEAP_POINTER;
    uint32_t mram_base_addr_B = (uint32_t)(DPU_MRAM_HEAP_POINTER + input_size_dpu_bytes_transfer);

    // Initialize a local cache to store the MRAM block
    T *cache_A = (T *) mem_alloc(BLOCK_SIZE);
    T *cache_B = (T *) mem_alloc(BLOCK_SIZE);

    for(unsigned int byte_index = base_tasklet; byte_index < input_size_dpu_bytes; byte_index += BLOCK_SIZE * NR_TASKLETS){
        // Bound checking
        uint32_t l_size_bytes = (byte_index + BLOCK_SIZE >= input_size_dpu_bytes) ? (input_size_dpu_bytes - byte_index) : BLOCK_SIZE;

        // Load cache with current MRAM block
        mram_read((__mram_ptr void const*)(mram_base_addr_A + byte_index), cache_A, l_size_bytes);
        mram_read((__mram_ptr void const*)(mram_base_addr_B + byte_index), cache_B, l_size_bytes);

        // Computer vector addition
        vector_addition(cache_B, cache_A, l_size_bytes >> DIV);

        // Write cache to current MRAM block
        mram_write(cache_B, (__mram_ptr void*)(mram_base_addr_B + byte_index), l_size_bytes);

    }
    return 0;
}
```

Tasklet ID
Size of vector tile processed by a DPU
MRAM addresses of arrays A and B
WRAM allocation
MRAM-WRAM DMA transfers
Vector addition (see next slide)
WRAM-MRAM DMA transfer

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`).

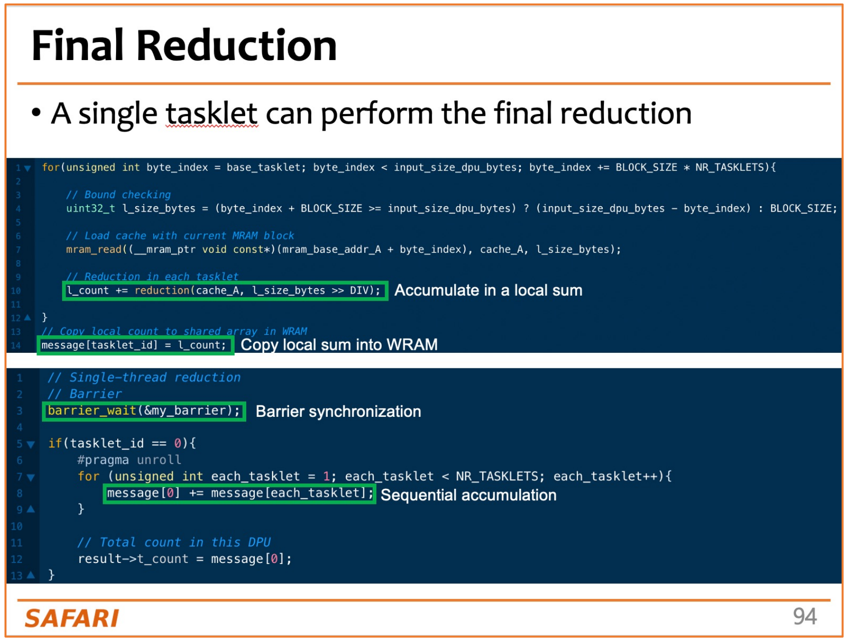
- You will observe significant variations in arithmetic throughput for different operations and datatypes

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

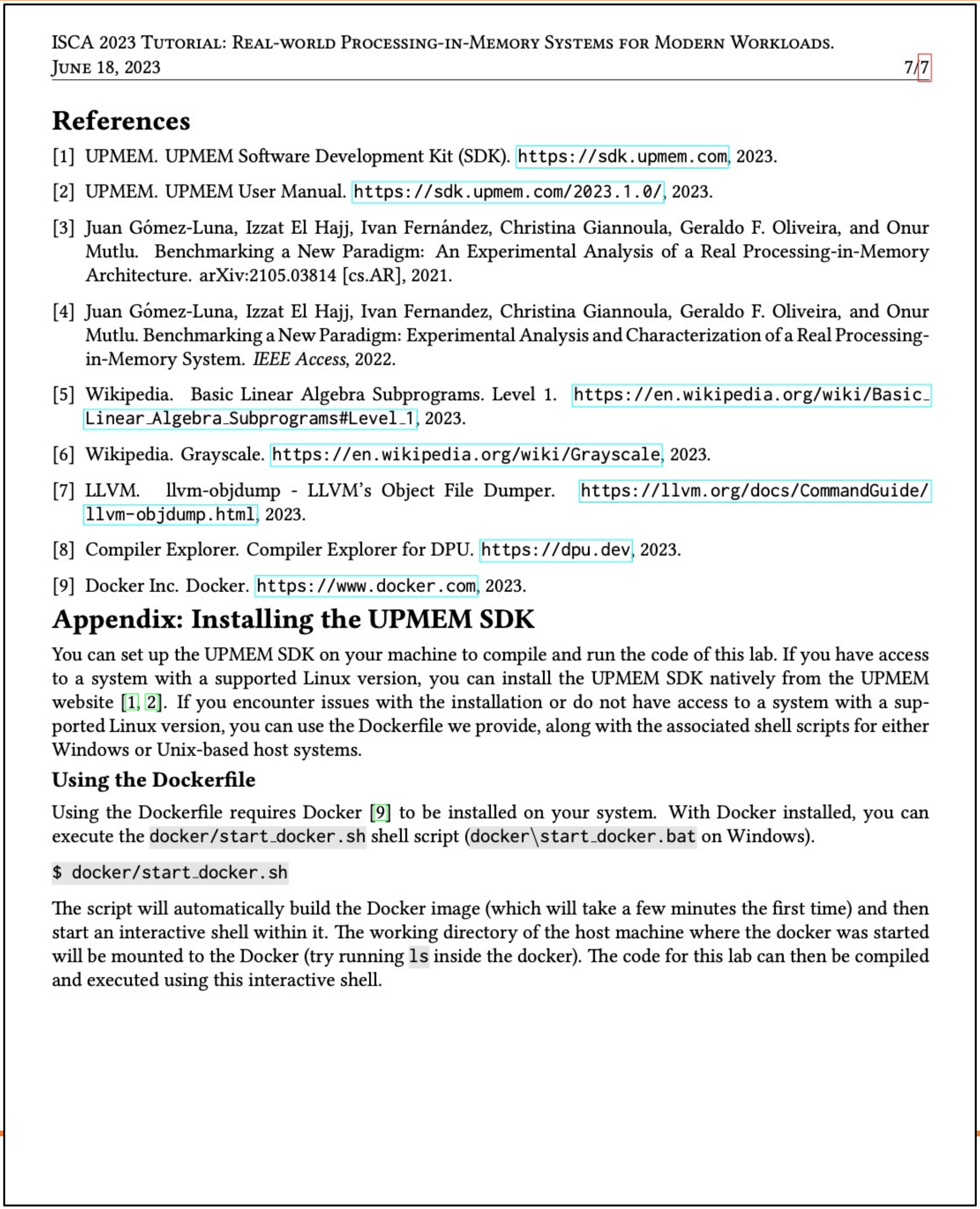
- A single tasklet can perform the final reduction

```
for(unsigned int byte_index = base_tasklet; byte_index < input_size_dpu_bytes; byte_index += BLOCK_SIZE * NR_TASKLETS){

    // Bound checking
    uint32_t l_size_bytes = (byte_index + BLOCK_SIZE >= input_size_dpu_bytes) ? (input_size_dpu_bytes - byte_index) : BLOCK_SIZE;

    // Load cache with current MRAM block
    mram_read((__mram_ptr void const*)(mram_base_addr_A + byte_index), cache_A, l_size_bytes);

    // Reduction in each tasklet
    l_count += reduction(cache_A, l_size_bytes >> DIV);   Accumulate in a local sum

}
// Copy local count to shared array in WRAM
message[tasklet_id] = l_count;   Copy local sum into WRAM
```

```
// Single-thread reduction
// Barrier
barrier_wait(&my_barrier);   Barrier synchronization

if(tasklet_id == 0){
    #pragma unroll
    for (unsigned int each_tasklet = 1; each_tasklet < NR_TASKLETS; each_tasklet++){
        message[0] += message[each_tasklet];   Sequential accumulation
    }

    // Total count in this DPU
    result->t_count = message[0];
}
```

SAFARI 94

Real PIM Tutorial: Hands-on Lab



References

[1] UPMEM. UPMEM Software Development Kit (SDK). `https://sdk.upmem.com`, 2023.

[2] UPMEM. UPMEM User Manual. `https://sdk.upmem.com/2023.1.0/`, 2023.

[3] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernández, Christina Giannoula, Geraldo F. Oliveira, and Onur Mutlu. Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture. arXiv:2105.03814 [cs.AR], 2021.

[4] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F. Oliveira, and Onur Mutlu. Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System. *IEEE Access*, 2022.

[5] Wikipedia. Basic Linear Algebra Subprograms. Level 1. `https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms#Level_1`, 2023.

[6] Wikipedia. Grayscale. `https://en.wikipedia.org/wiki/Grayscale`, 2023.

[7] LLVM. llvm-objdump - LLVM's Object File Dumper. `https://llvm.org/docs/CommandGuide/llvm-objdump.html`, 2023.

[8] Compiler Explorer. Compiler Explorer for DPU. `https://dpu.dev`, 2023.

[9] Docker Inc. Docker. `https://www.docker.com`, 2023.

Appendix: Installing the UPMEM SDK

You can set up the UPMEM SDK on your machine to compile and run the code of this lab. If you have access to a system with a supported Linux version, you can install the UPMEM SDK natively from the UPMEM website [1, 2]. If you encounter issues with the installation or do not have access to a system with a supported Linux version, you can use the Dockerfile we provide, along with the associated shell scripts for either Windows or Unix-based host systems.

Using the Dockerfile

Using the Dockerfile requires Docker [9] to be installed on your system. With Docker installed, you can execute the `docker/start_docker.sh` shell script (`docker\start_docker.bat` on Windows).

```
$ docker/start_docker.sh
```

The script will automatically build the Docker image (which will take a few minutes the first time) and then start an interactive shell within it. The working directory of the host machine where the docker was started will be mounted to the Docker (try running `ls` inside the docker). The code for this lab can then be compiled and executed using this interactive shell.

ISCA 2023 Tutorial

Real-world Processing-in-Memory Systems for Modern Workloads

Hands-on Lab

Programming and Understanding a Real Processing-in-Memory Architecture

Dr. Juan Gómez Luna

Professor Onur Mutlu

ETH Zürich

Sunday, June 18, 2023

SAFARI