Memory Technologies

Instructor: Prof. Hamed Farbeh

Amirkabir University of Technology
(Tehran polytechnic)

Optimizing Algorithm Performance with HBM-PIM: A Matrix Multiplication Case Study

Authors:

Morteza Adelkhani	Madelkhani@aut.ac.ir
Sara Zamani	sara.zamani73@aut.ac.ir

Student, project by:

Reza Adinepour	adinepour@aut.ac.ir

Spring 2024

Evaluation, Academic Integrity and Submition

Notes on the project:

1- Evaluation

The evaluation of this project is based on the following:

a) **Algorithm Implementation (10%)**: Correctness and efficiency of the algorithm implementation.

b) **Simulation Setup (30%)**: Proper configuration and usage of the PIMSimulator.

c) **Performance Analysis (30%)**: Depth of analysis and understanding of performance metrics.

d) **Report Quality (20%)**: Clarity, completeness, and professionalism of the report.

e) **Presentation (10%)**: Effectiveness and clarity of the presentation.

2- Academic Integrity

Each student is expected to adhere to the highest standards of academic integrity. Any form of copying or plagiarism will result in severe penalties. Ensure your work is original and properly referenced.

3- Submission

The project submission guideline is as follows:

a) **Progress Reports**: Submit reports via the Courses portal until the deadline. Each delay in your submission is not acceptable.

b) **Final Report and Code**: Submit your final report and source code in a zipped folder.

c) **Presentation**: Present your findings in class during the final project meeting.

Contents

1 Project Description

1.1 Proposed of this project:

Neurodegenerative diseases, including Alzheimer's In this project you are going to deal with processing in memory (PIM) structure. For surfing in real-world application of PIM, in this project you have to run an algorithm on real PIM device and report your result that you get. Based on the lack of accessibility of real PIM hardware you are going to using PIMSimulator which is based on HBM-PIM of Samsung.

1.2 Description of HBM-PIM:

High Bandwidth Memory (HBM) is a type of memory that's made to transfer data quickly and use less energy. It's built by stacking memory layers on top of each other, which lets them connect directly and move data fast. At the base of these layers, there's a special piece that controls the flow of data to and from other parts of the computer. HBM is often used with powerful computer chips like GPUs because it can handle a lot of data at once, making everything run smoother and faster.

Inside HBM, there are separate paths for data called pseudo-channels, and each one has smaller sections called banks where data is stored. When the computer needs to read data, it picks a specific path and bank, then grabs the data from there. This process is similar to how other types of memory work, but HBM's design lets it do this much quicker and with less energy.

Samsung has made a version of HBM called HBM-PIM that's even better because it can do some data processing right inside the memory itself. This means the computer doesn't have to move data around as much, which makes things faster and saves energy. This new design fits in with how memory is usually made, so it's easy to start using in products.

Overall, HBM and its improved version, HBM-PIM, are big steps forward for memory technology. They're really important for programs that need to process a lot of data quickly, like artificial intelligence and scientific computing, because they make everything more efficient and faster.

2 Project Detaile

As it described in previous section you will use the PIMSimulator to simulate the performance of different algorithms on the HBM-PIM architecture. Each student will be assigned one algorithm to analyze. The goal is to understand how PIM technology affects the performance and efficiency of these algorithms compared to traditional memory architectures. Your task to doing this project is as follow:

2.1 Choosing Algorithm:

You have to choose one of the following algorithms to analyze and submit it in your section of project table assignment in courses portal.

Note: The priority of choosing an algorithm is with the first student which choose it.

2.1.1 Sorting Algorithms:

a) QuickSort

b) MergeSort

c) HeapSort

d) Insertion Sort

e) Bubble Sort

2.1.2 Graph Algorithms:

a) Dijkstra's Algorithm

b) Breadth-First Search (BFS)

c) Depth-First Search (DFS)

d) Prim's Algorithm

e) Kruskal's Algorithm

2.1.3 Matrix Operations:

f) Matrix Multiplication

g) Sparse Matrix-Vector Multiplication (SpMV)

IV. Machine Learning and Data Processing Algorithms:

h) K-means Clustering

2.2 Literature Review:

Each student will review existing research on HBM, PIM, and the specific algorithm assigned to them.

2.3 Algorithm Implementation:

Students will implement their assigned algorithm in a compatible programming language (e.g., C++, Python) if not already available.

2.4 Simulation Setup:

Students will set up the PIMSimulator to run their algorithms, configuring necessary parameters and optimizing the code to leverage PIM features.

2.5 Performance Analysis:

Students will run simulations to collect data on execution time, power consumption, and other relevant metrics.

2.6 Comparison:

Compare the performance of the algorithm on HBM-PIM with traditional memory architectures.

2.7 Report and Presentation:

Each student will compile their findings into a detailed report and present their results to the class.

The algorithm I have chosen for this project is matrix multiplication. Therefore, I first need to build the simulator, the installation steps of which I will explain below:

3 PIMSimulator

3.1 Overview

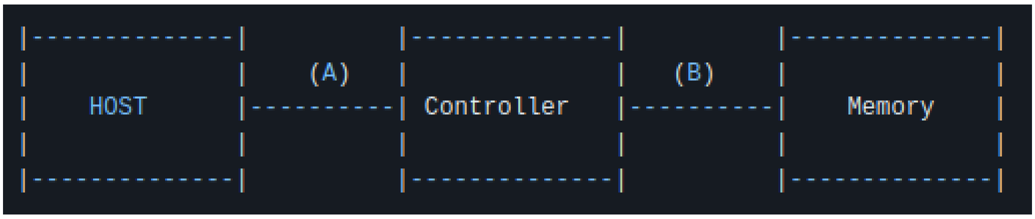
PIMSimulator is a cycle accurate model that Single Instruction, Multiple Data (SIMD) execution units that uses the bank-level parallelism in PIM Block to boost performance that would have otherwise used multiple times of bandwidth from simultaneous access of all bank. The simulator include memory and have embedded within it a PIM block, which consist of programmable command registers, general purpose register files and execution units.

Based on `github.com/umd-memsys/DRAMSim2`, the simulator includes the simulator includes

- PIM Block:
  - Register files including CRF (for command), GRF (for vector value), SRF (for scalar value)
  - ALU (ADD, MUL, MAC, MAD, MOVE, FILL, NOP, JUMP, EXIT)
- PIM Kernel:
  - Generate a set of memory transactions for enabling PIM operation
- HBM2 support (refer to `ini/HBM2_samsung_2M_16B_x64.ini`)

3.2 HW description

PIM is a HBM stack that is pin compatible with HBM2 and have embedded within it a PIM block



```
|--------------|          |--------------|          |--------------|
|              |    (A)   |              |    (B)   |              |
|     HOST     |----------| Controller   |----------|    Memory    |
|              |          |              |          |              |
|--------------|          |--------------|          |--------------|
```

Figure 1: Architecture of PIM

3.2.1 Base Architecture

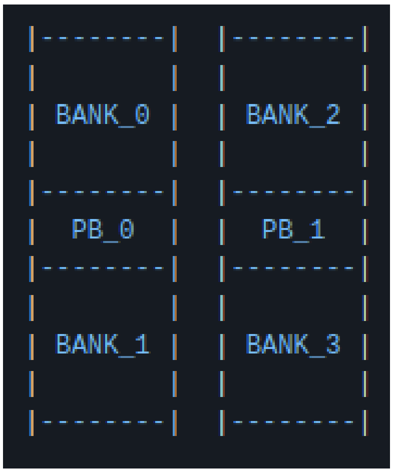
- Each channel is logically independent memory, so it has a dedicated independent controller.
- Read [Addr], Write [Addr]
- Activate, Read, Write, Precharge, Refresh, `Activate_pim`, `ALU_pim`, `Precharge_pim`, `READ_pim`

1. HBM2
   (a) System Specification: `system_hbm.ini`
   (b) HBM Specification: `ini/HBM2_samsung_2M_16B_x64.ini`
      i. 1 PIM block per 2 banks, 4 Bank per Bankgroup, 4 Bank group per pseudo channel, 4 pseudo channel per die, 4 die per stack.
      ii. Prefetch size : 256bit
      iii. burst length: 4n
      iv. Pin speed: 2Gbps
      v. The simulator supports the pseudo-channel mode only, and we assume that each pseudo-channel is totally independent.

3.2.2 Address mapping

1. The address mapping is used when the memory controller decodes the address from host.
2. Use Scheme8 addressing mode for PIM functionality.
   `|<-rank->|<-row->|<-col high->|<-bg->|<-bank->|<-chan->|<-col low->|<-offset ->|`
3. the length of `col_low` is log(BL * JEDEC_DATA_BUS_BUTS/8), which are 5b both for HBM2
4. You can also change the current addressing mode dynamically (Not recommended, though)

```
// Static Setting in system_*.ini
ADDRESS_MAPPING_SCHEME=Scheme8
```



```
|--------|  |--------|
|        |  |        |
| BANK_0 |  | BANK_2 |
|        |  |        |
|--------|  |--------|
|  PB_0  |  |  PB_1  |
|--------|  |--------|
|        |  |        |
| BANK_1 |  | BANK_3 |
|        |  |        |
|--------|  |--------|
```

Figure 2: HBM2 case

3.2.3 PIM Block Placement

`BANKS_PER_PIM_BLOCK = NUM_BANKS / NUM_PIM_BLOCKS`

1. if `2 * NUM_PIM_BLOCKS == NUM_BANK`, a PIM block is located per two banks.
   (a) `NUM_PIM_BLOCKS = 8, NUM_BANKS = 16`
2. if `NUM_PIM_BLOCKS == NUM_BANKS`, a PIM Block (PB) is located per banks.
   (a) `NUM_PIM_BLOCKS = 8, NUM_BANKS = 8`

- Supports RISC-style 32-bit instructions
- Three instructions types
  - 4 Arithmetic: ADD, MUL, MAC, MAD
  - 2 Data transfer: MOV, FILL
  - 3 control flows: NOP, JUMP, EXIT
- JUMP instruction
  - Zero-cycle static branch: supports only a pre-programmed numbers of iterations
- Operand type:
  - Vector Register (GRF_A, GRF_B)
  - Scalar Register (SRF)
  - Bank Row Buffer
- PIM instructions are stored in the Command Register File (CRF), and memory command triggers a CRF to perform a target instruction

Type	Command	Description	Result (DST)	Operand (SRC0)	Operand (SRC1)
Arithmetic	ADD	addition	GRF	GRF, BANK, SRF	GRF, BANK, SRF
Arithmetic	MUL	multiplication	GRF	GRF, BANK, SRF	GRF, BANK, SRF
Arithmetic	MAC	multiply-accumulate	GRF_B	GRF, BANK	GRF, BANK, SRF
Arithmetic	MAD	multiply-and-add	GRF	GRF, BANK	GRF, BANK, SRF
Data	MOV	load or store data from register to bank	GRF, SRF	GRF, SRF	
Data	FILL	copy data from bank to register	GRF, BANK	GRF, BANK	
Control	NOP	do nothing			
Control	JUMP	jump instruction			
Control	EXIT	exit instruction			

Table 1: Table of Commands

  - each memory command increments the CRF PC

- DRAM commands decide where to retrieve data from DRAM for PIM arithmetic operations

3.2.4 Movement of Data

Mode	Transaction	PIM Instruction	Operation
SB	Read	-	Normal Memory Read
SB	Write	-	Normal Memory Write
HAB	Write	-	PIM Write (Host to PIM Register)
PIM	-	MOV	read or write from bank to PIM Register
PIM	-	FILL	write from bank to PIM Registers

Table 2: Table of Transactions and Operations

1. SB mode: standard DRAM operation

2. HAB mode: Allowing concurrent accesses to multiple banks with a single DRAM command
3. PIM mode: Triggers the execution of PIM commands on the CRF by DRAM Command

3.3 Setup

3.4 Prerequisites

- Scons tool for compiling PIMSimulator:

```
$ sudo apt install scons
```

- gtest for running test cases:

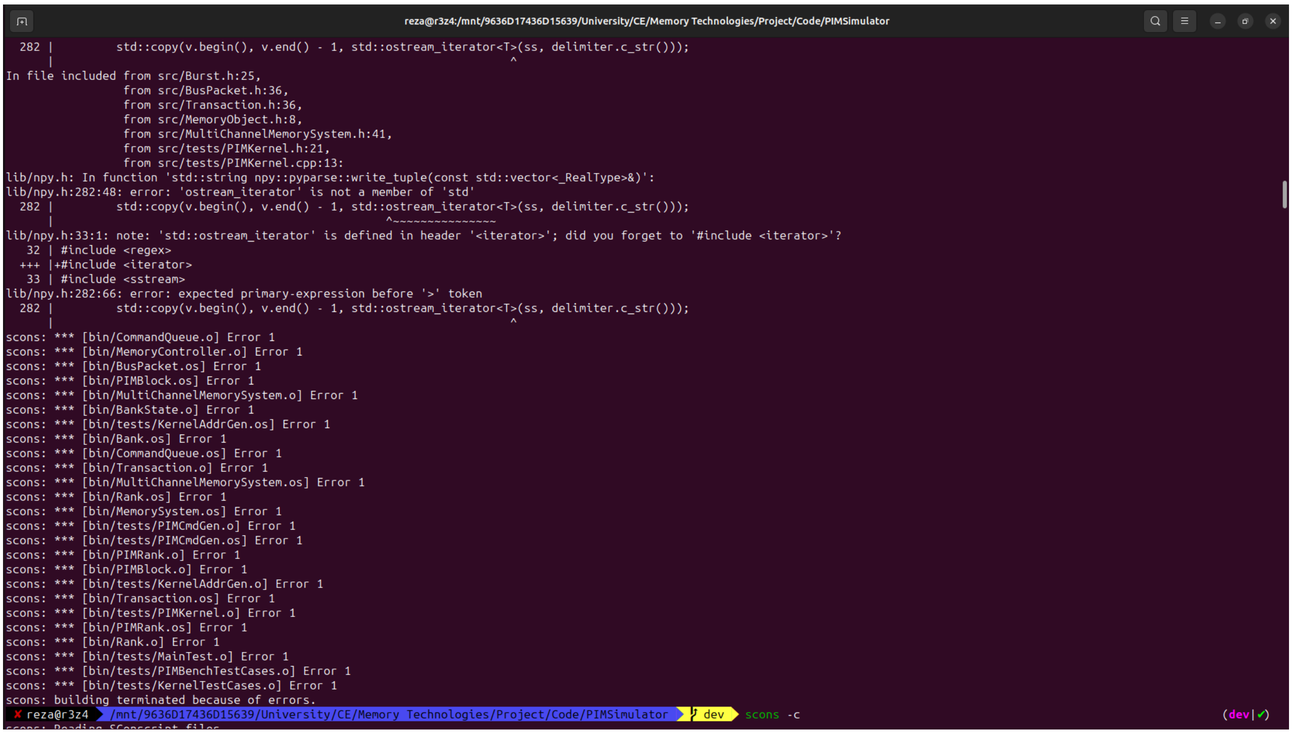
```
$ sudo apt install libgtest-dev
```

3.5 Installing

- To Install PIMSimulator:

```
# compile
scons
```

After entering the scons command, we encountered the following errors.


reza@r3z4:/mnt/9636D17436D15639/University/CE/Memory Technologies/Project/Code/PIMSimulator

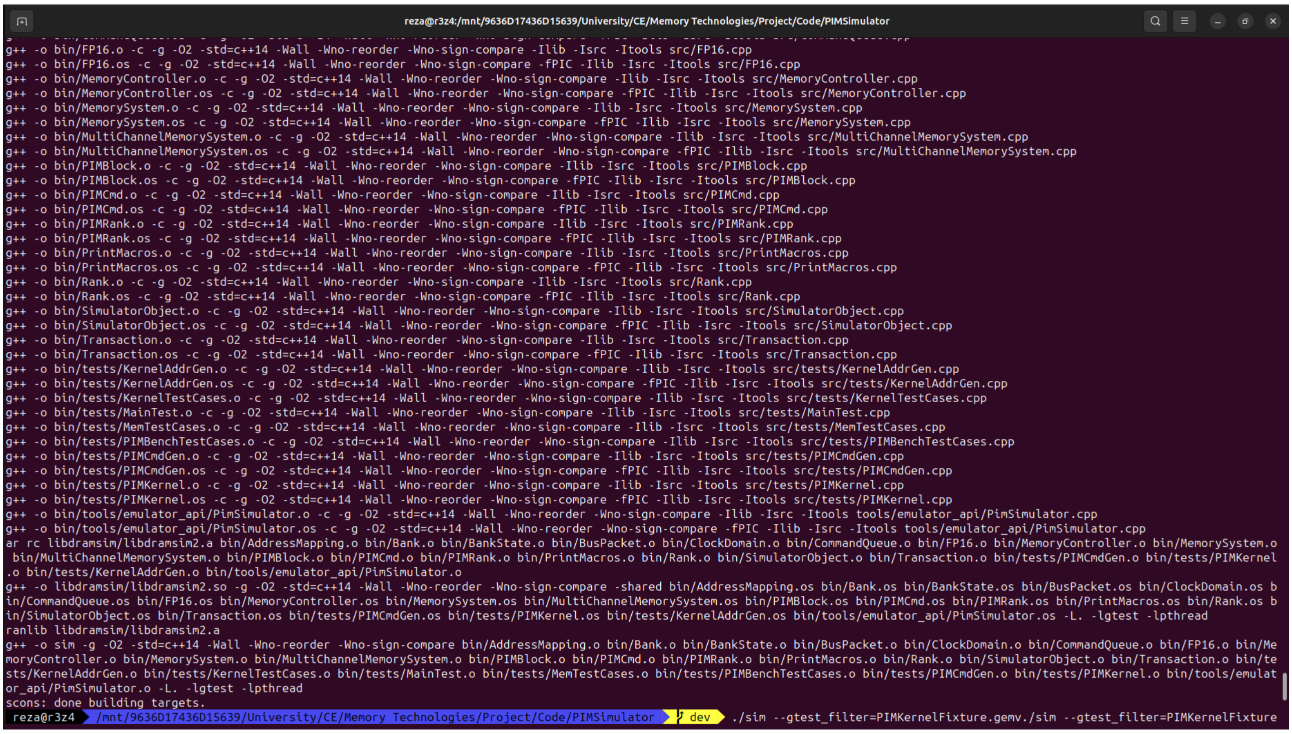
```
  282 |         std::copy(v.begin(), v.end() - 1, std::ostream_iterator<T>(ss, delimiter.c_str()));
      |                                                                  ^
In file included from src/Burst.h:25,
                 from src/BusPacket.h:36,
                 from src/Transaction.h:36,
                 from src/MemoryObject.h:8,
                 from src/MultiChannelMemorySystem.h:41,
                 from src/tests/PIMKernel.h:21,
                 from src/tests/PIMKernel.cpp:13:
lib/npy.h: In function 'std::string npy::pyparse::write_tuple(const std::vector<_RealType>&)':
lib/npy.h:282:48: error: 'ostream_iterator' is not a member of 'std'
  282 |         std::copy(v.begin(), v.end() - 1, std::ostream_iterator<T>(ss, delimiter.c_str()));
      |                                                ^~~~~~~~~~~~~~~~
lib/npy.h:33:1: note: 'std::ostream_iterator' is defined in header '<iterator>'; did you forget to '#include <iterator>'?
   32 | #include <regex>
  +++ |+#include <iterator>
   33 | #include <sstream>
lib/npy.h:282:66: error: expected primary-expression before '>' token
  282 |         std::copy(v.begin(), v.end() - 1, std::ostream_iterator<T>(ss, delimiter.c_str()));
      |                                                                  ^
scons: *** [bin/CommandQueue.o] Error 1
scons: *** [bin/MemoryController.o] Error 1
scons: *** [bin/BusPacket.os] Error 1
scons: *** [bin/PIMBlock.os] Error 1
scons: *** [bin/MultiChannelMemorySystem.o] Error 1
scons: *** [bin/BankState.o] Error 1
scons: *** [bin/tests/KernelAddrGen.os] Error 1
scons: *** [bin/Bank.os] Error 1
scons: *** [bin/CommandQueue.os] Error 1
scons: *** [bin/Transaction.o] Error 1
scons: *** [bin/MultiChannelMemorySystem.os] Error 1
scons: *** [bin/Rank.os] Error 1
scons: *** [bin/MemorySystem.os] Error 1
scons: *** [bin/tests/PIMCmdGen.o] Error 1
scons: *** [bin/tests/PIMCmdGen.os] Error 1
scons: *** [bin/PIMRank.o] Error 1
scons: *** [bin/PIMBlock.o] Error 1
scons: *** [bin/tests/KernelAddrGen.o] Error 1
scons: *** [bin/Transaction.os] Error 1
scons: *** [bin/tests/PIMKernel.o] Error 1
scons: *** [bin/PIMRank.os] Error 1
scons: *** [bin/Rank.o] Error 1
scons: *** [bin/tests/MainTest.o] Error 1
scons: *** [bin/tests/PIMBenchTestCases.o] Error 1
scons: *** [bin/tests/KernelTestCases.o] Error 1
scons: building terminated because of errors.
 ✗ reza@r3z4  /mnt/9636D17436D15639/University/CE/Memory Technologies/Project/Code/PIMSimulator  dev  scons -c                                            (dev|✓)
```

Figure 3: Error in build

After some investigation, I realized that the problem was with the compiler. I had both gcc and g++ compilers installed on my system, and by default, the gcc compiler was selected. I changed the default compiler to g++ with the following command, and the error problem was resolved.

```
$ sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-9 60
$ sudo update-alternatives --config g++
```

With the compiler change, the build is completed successfully


reza@r3z4:/mnt/9636D17436D15639/University/CE/Memory Technologies/Project/Code/PIMSimulator

```
g++ -o bin/FP16.o -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -Ilib -Isrc -Itools src/FP16.cpp
g++ -o bin/FP16.os -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -fPIC -Ilib -Isrc -Itools src/FP16.cpp
g++ -o bin/MemoryController.o -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -Ilib -Isrc -Itools src/MemoryController.cpp
g++ -o bin/MemoryController.os -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -fPIC -Ilib -Isrc -Itools src/MemoryController.cpp
g++ -o bin/MemorySystem.o -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -Ilib -Isrc -Itools src/MemorySystem.cpp
g++ -o bin/MemorySystem.os -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -fPIC -Ilib -Isrc -Itools src/MemorySystem.cpp
g++ -o bin/MultiChannelMemorySystem.o -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -Ilib -Isrc -Itools src/MultiChannelMemorySystem.cpp
g++ -o bin/MultiChannelMemorySystem.os -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -fPIC -Ilib -Isrc -Itools src/MultiChannelMemorySystem.cpp
g++ -o bin/PIMBlock.o -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -Ilib -Isrc -Itools src/PIMBlock.cpp
g++ -o bin/PIMBlock.os -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -fPIC -Ilib -Isrc -Itools src/PIMBlock.cpp
g++ -o bin/PIMCmd.o -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -Ilib -Isrc -Itools src/PIMCmd.cpp
g++ -o bin/PIMCmd.os -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -fPIC -Ilib -Isrc -Itools src/PIMCmd.cpp
g++ -o bin/PIMRank.o -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -Ilib -Isrc -Itools src/PIMRank.cpp
g++ -o bin/PIMRank.os -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -fPIC -Ilib -Isrc -Itools src/PIMRank.cpp
g++ -o bin/PrintMacros.o -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -Ilib -Isrc -Itools src/PrintMacros.cpp
g++ -o bin/PrintMacros.os -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -fPIC -Ilib -Isrc -Itools src/PrintMacros.cpp
g++ -o bin/Rank.o -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -Ilib -Isrc -Itools src/Rank.cpp
g++ -o bin/Rank.os -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -fPIC -Ilib -Isrc -Itools src/Rank.cpp
g++ -o bin/SimulatorObject.o -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -Ilib -Isrc -Itools src/SimulatorObject.cpp
g++ -o bin/SimulatorObject.os -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -fPIC -Ilib -Isrc -Itools src/SimulatorObject.cpp
g++ -o bin/Transaction.o -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -Ilib -Isrc -Itools src/Transaction.cpp
g++ -o bin/Transaction.os -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -fPIC -Ilib -Isrc -Itools src/Transaction.cpp
g++ -o bin/tests/KernelAddrGen.o -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -Ilib -Isrc -Itools src/tests/KernelAddrGen.cpp
g++ -o bin/tests/KernelAddrGen.os -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -fPIC -Ilib -Isrc -Itools src/tests/KernelAddrGen.cpp
g++ -o bin/tests/KernelTestCases.o -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -Ilib -Isrc -Itools src/tests/KernelTestCases.cpp
g++ -o bin/tests/MainTest.o -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -Ilib -Isrc -Itools src/tests/MainTest.cpp
g++ -o bin/tests/MemTestCases.o -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -Ilib -Isrc -Itools src/tests/MemTestCases.cpp
g++ -o bin/tests/PIMBenchTestCases.o -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -Ilib -Isrc -Itools src/tests/PIMBenchTestCases.cpp
g++ -o bin/tests/PIMCmdGen.o -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -Ilib -Isrc -Itools src/tests/PIMCmdGen.cpp
g++ -o bin/tests/PIMCmdGen.os -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -fPIC -Ilib -Isrc -Itools src/tests/PIMCmdGen.cpp
g++ -o bin/tests/PIMKernel.o -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -Ilib -Isrc -Itools src/tests/PIMKernel.cpp
g++ -o bin/tests/PIMKernel.os -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -fPIC -Ilib -Isrc -Itools src/tests/PIMKernel.cpp
g++ -o bin/tools/emulator_api/PimSimulator.o -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -Ilib -Isrc -Itools tools/emulator_api/PimSimulator.cpp
g++ -o bin/tools/emulator_api/PimSimulator.os -c -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -fPIC -Ilib -Isrc -Itools tools/emulator_api/PimSimulator.cpp
ar rc libdramsim/libdramsim2.a bin/AddressMapping.o bin/Bank.o bin/BankState.o bin/BusPacket.o bin/ClockDomain.o bin/CommandQueue.o bin/FP16.o bin/MemoryController.o bin/MemorySystem.o
 bin/MultiChannelMemorySystem.o bin/PIMBlock.o bin/PIMCmd.o bin/PIMRank.o bin/PrintMacros.o bin/Rank.o bin/SimulatorObject.o bin/Transaction.o bin/tests/PIMCmdGen.o bin/tests/PIMKernel
.o bin/tests/KernelAddrGen.o bin/tools/emulator_api/PimSimulator.o
g++ -o libdramsim/libdramsim2.so -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare -shared bin/AddressMapping.os bin/Bank.os bin/BankState.os bin/BusPacket.os bin/ClockDomain.os b
in/CommandQueue.os bin/FP16.os bin/MemoryController.os bin/MemorySystem.os bin/MultiChannelMemorySystem.os bin/PIMBlock.os bin/PIMCmd.os bin/PIMRank.os bin/PrintMacros.os bin/Rank.os b
in/SimulatorObject.os bin/Transaction.os bin/tests/PIMCmdGen.os bin/tests/PIMKernel.os bin/tests/KernelAddrGen.os bin/tools/emulator_api/PimSimulator.os -L. -lgtest -lpthread
ranlib libdramsim/libdramsim2.a
g++ -o sim -g -O2 -std=c++14 -Wall -Wno-reorder -Wno-sign-compare bin/AddressMapping.o bin/Bank.o bin/BankState.o bin/BusPacket.o bin/ClockDomain.o bin/CommandQueue.o bin/FP16.o bin/Me
moryController.o bin/MemorySystem.o bin/MultiChannelMemorySystem.o bin/PIMBlock.o bin/PIMCmd.o bin/PIMRank.o bin/PrintMacros.o bin/Rank.o bin/SimulatorObject.o bin/Transaction.o bin/te
sts/KernelAddrGen.o bin/tests/KernelTestCases.o bin/tests/MainTest.o bin/tests/MemTestCases.o bin/tests/PIMBenchTestCases.o bin/tests/PIMCmdGen.o bin/tests/PIMKernel.o bin/tools/emulat
or_api/PimSimulator.o -L. -lgtest -lpthread
scons: done building targets.
 reza@r3z4  /mnt/9636D17436D15639/University/CE/Memory Technologies/Project/Code/PIMSimulator  dev  ./sim --gtest_filter=PIMKernelFixture.gemv./sim --gtest_filter=PIMKernelFixture
```

Figure 4: Done building targets

In this simulator, there are pre-written examples that we can list as follows:

3.6 Launch a Test Run

- Show a list of test cases

```
./sim --gtest_list_tests

# Example
PIMKernelFixture.
gemv_tree
gemv
mul
add
relu
MemBandwidthFixture.
hbm_read_bandwidth
hbm_write_bandwidth
```

```
PIMBenchFixture.
gemv
mul
add
relu
```

For further investigation, we will thoroughly examine and explain the matrix multiplication code file located in the path `/PIMSimulator/data/gemv/gen_gemv.py`.

4 Matrix-vector Multiplication Code Breakdown

4.1 Introduction

This code explains a Python script that performs matrix-vector multiplication. The script involves generating random input data and weights, performing matrix multiplications, and saving the results to files.

```python
import numpy as np

# min dim_in = 128 -> 256bit / 16bit
# min dim_out = 8 PIM block
BATCH = 1
REAL_DIM_IN = 1024
DIM_IN = 1024
DIM_OUT = 4096

np.set_printoptions(precision=20)
np.random.seed(41)

batch_in = np.random.standard_normal(size=(DIM_IN, BATCH)).astype('float16')
for i in range(REAL_DIM_IN, DIM_IN):
  for j in range(0, BATCH):
    batch_in[i][j] = 0

data_w = np.random.standard_normal(size=(DIM_OUT, DIM_IN)).astype('float16')

np.random.shuffle(data_w)
batch_out = np.zeros((DIM_OUT, BATCH)).astype('float16')
batch_out = np.matmul(data_w, batch_in)

batch_out2 = np.zeros((DIM_OUT, BATCH)).astype('float16')

for y in range(0, DIM_OUT):
  for x in range(0, DIM_IN):
    batch_out2[y] += data_w[y][x] * batch_in[x][0]

batch_in = batch_in.T.copy()
batch_out = batch_out.T.copy()
batch_out2 = batch_out2.T.copy()

np.save("gemv_input_" + str(DIM_OUT) + "x" + str(DIM_IN), batch_in)
np.save("gemv_weight_" + str(DIM_OUT) + "x" + str(DIM_IN), data_w)
np.save("gemv_output_" + str(DIM_OUT) + "x" + str(DIM_IN), batch_out)
np.save("test_output_" + str(DIM_OUT) + "x" + str(DIM_IN), batch_out2)
print(batch_in)
print(batch_out)
print(batch_out2)
print(batch_in.shape)
print(batch_out.shape)
```

Listing 1: Python Code

4.2 Set constants

```python
BATCH = 1
REAL_DIM_IN = 1024
DIM_IN = 1024
DIM_OUT = 4096
```

- `BATCH`: The batch size of the input vectors.
- `REAL_DIM_IN`: Actual input dimension size.
- `DIM_IN`: Padded input dimension size (could be the same or larger than `REAL_DIM_IN`).
- `DIM_OUT`: Output dimension size.

4.3 Generate input data

```python
batch_in = np.random.standard_normal(size=(DIM_IN, BATCH)).astype('float16')
```

Create a `DIM_IN x BATCH` matrix filled with random values from a standard normal distribution, converted to `float16` precision.

4.4 Zero out the padded part of the input

```python
for i in range(REAL_DIM_IN, DIM_IN):
  for j in range(0, BATCH):
    batch_in[i][j] = 0
```

Set the elements of `batch_in` to zero for indices from `REAL_DIM_IN` to `DIM_IN` to handle the padded part.

4.5 Generate weight data

```python
data_w = np.random.standard_normal(size=(DIM_OUT, DIM_IN)).astype('float16')
np.random.shuffle(data_w)
```

Create a `DIM_OUT x DIM_IN` matrix filled with random values from a standard normal distribution, converted to `float16` precision. Shuffle the rows of `data_w`.

4.6 Initialize output matrices and perform matrix multiplication

```
batch_out = np.zeros((DIM_OUT, BATCH)).astype('float16')
batch_out = np.matmul(data_w, batch_in)
```

4.7 Manual matrix multiplication for verification

```
batch_out2 = np.zeros((DIM_OUT, BATCH)).astype('float16')

for y in range(0, DIM_OUT):
  for x in range(0, DIM_IN):
    batch_out2[y] += data_w[y][x] * batch_in[x][0]
```

`batch_out2` is calculated element-wise by iterating over each element of the result and summing the product of corresponding elements from `data_w` and `batch_in`.

4.8 Transpose the input and output matrices

```
batch_in = batch_in.T.copy()
batch_out = batch_out.T.copy()
batch_out2 = batch_out2.T.copy()
```

4.9 Save the input, weight, and output matrices to files

```
np.save("gemv_input_" + str(DIM_OUT) + "x" + str(DIM_IN), batch_in)
np.save("gemv_weight_" + str(DIM_OUT) + "x" + str(DIM_IN), data_w)
np.save("gemv_output_" + str(DIM_OUT) + "x" + str(DIM_IN), batch_out)
np.save("test_output_" + str(DIM_OUT) + "x" + str(DIM_IN), batch_out2)
```

4.10 Print the matrices and their shapes

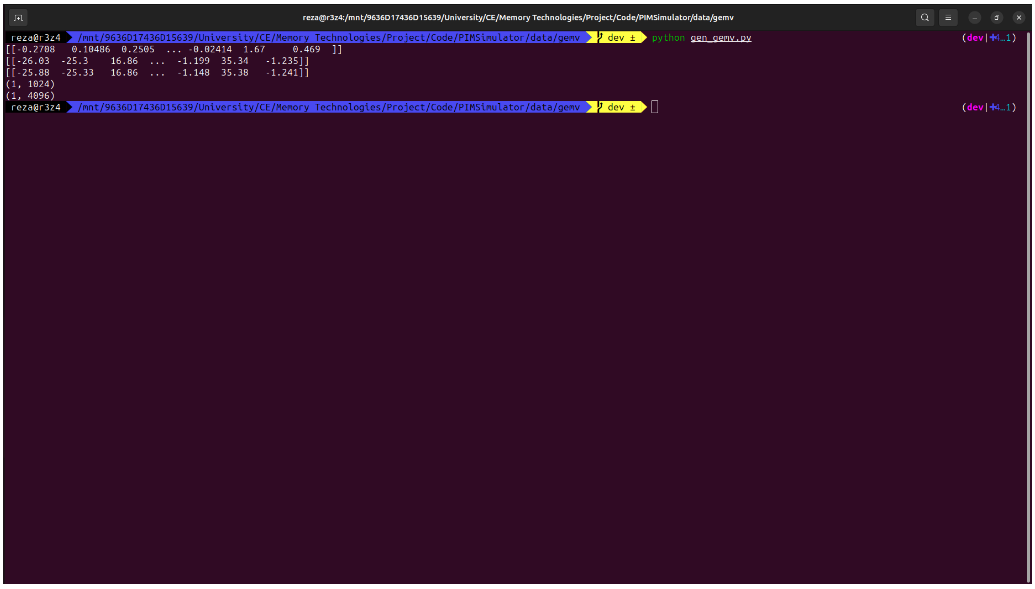
```
print(batch_in)
print(batch_out)
print(batch_out2)
print(batch_in.shape)
print(batch_out.shape)
```

After running the code, the output is as follows, and the matrices are saved as Numpy arrays in the same directory.

Now, after preparing the matrices, it is time to perform the matrix multiplication once with PIM and once without PIM.

To do this, we enter the following command in the simulator directory. The simulator runs the program once without considering PIM and once with considering PIM, and displays the outputs.

```
$ ./sim --gtest_filter=PIMBenchFixture.gemv
```



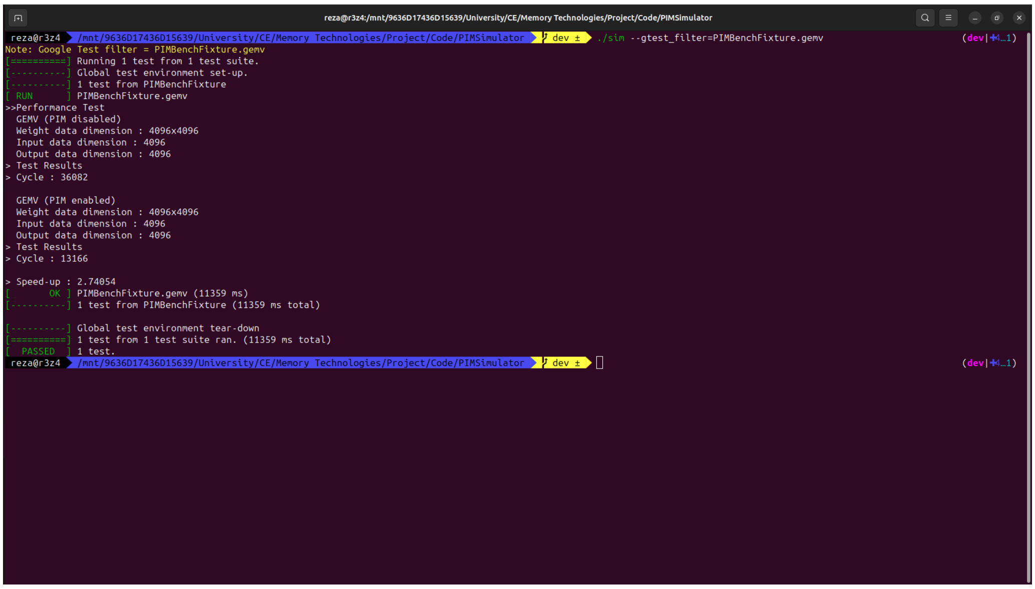
reza@r3z4:/mnt/9636D17436D15639/University/CE/Memory Technologies/Project/Code/PIMSimulator/data/gemv

```
reza@r3z4  /mnt/9636D17436D15639/University/CE/Memory Technologies/Project/Code/PIMSimulator/data/gemv  dev ±  python gen_gemv.py      (dev|4…1)
[[-0.2708   0.10486  0.2505  ... -0.02414  1.67     0.469   ]]
[[-26.03  -25.3    16.86   ...  -1.199  35.34   -1.235]]
[[-25.88  -25.33   16.86   ...  -1.148  35.38   -1.241]]
(1, 1024)
(1, 4096)
reza@r3z4  /mnt/9636D17436D15639/University/CE/Memory Technologies/Project/Code/PIMSimulator/data/gemv  dev ±       (dev|4…1)
```

Figure 5: Output of `gen_gemv.py`

5 Output simulation

The simulation output is as follows:



reza@r3z4:/mnt/9636D17436D15639/University/CE/Memory Technologies/Project/Code/PIMSimulator

```
reza@r3z4  /mnt/9636D17436D15639/University/CE/Memory Technologies/Project/Code/PIMSimulator  dev ±  ./sim --gtest_filter=PIMBenchFixture.gemv      (dev|4…1)
Note: Google Test filter = PIMBenchFixture.gemv
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from PIMBenchFixture
[ RUN      ] PIMBenchFixture.gemv
>>Performance Test
  GEMV (PIM disabled)
  Weight data dimension : 4096x4096
  Input data dimension : 4096
  Output data dimension : 4096
> Test Results
> Cycle : 36082

  GEMV (PIM enabled)
  Weight data dimension : 4096x4096
  Input data dimension : 4096
  Output data dimension : 4096
> Test Results
> Cycle : 13166

> Speed-up : 2.74054
[       OK ] PIMBenchFixture.gemv (11359 ms)
[----------] 1 test from PIMBenchFixture (11359 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (11359 ms total)
[  PASSED  ] 1 test.
reza@r3z4  /mnt/9636D17436D15639/University/CE/Memory Technologies/Project/Code/PIMSimulator  dev ±       (dev|4…1)
```

Figure 6: Output of PIM simulation

5.1 Analysis

5.1.1 Test Context

The test involves one test suite named `PIMBenchFixture` with one test case `gemv`. The simulation was executed using Google Test with the filter `PIMBenchFixture.gemv`.

5.1.2 Performance Test

The GEMV operation was performed twice:

1. **Without PIM:**
    - **Weight Data Dimension:** 4096x4096
    - **Input Data Dimension:** 4096
    - **Output Data Dimension:** 4096
    - **Cycles:** 36,082
2. **With PIM:**
    - **Weight Data Dimension:** 4096x4096
    - **Input Data Dimension:** 4096
    - **Output Data Dimension:** 4096
    - **Cycles:** 13,166

5.1.3 Results Interpretation

- **Cycle Count:**
    - **Without PIM:** The matrix multiplication took 36,082 cycles.
    - **With PIM:** The matrix multiplication took 13,166 cycles.
- **Speed-up:** The speed-up achieved by enabling PIM is calculated as 2.74054.

5.1.4 Overall Test Duration

The total time taken for the test execution was 11,359 milliseconds.

5.1.5 Significant Performance Improvement

The use of PIM technology resulted in a substantial reduction in the number of cycles required to complete the GEMV operation. Specifically, PIM enabled a 2.74x speed-up compared to the traditional approach without PIM. This indicates that PIM technology can significantly enhance the performance of matrix-vector multiplication by reducing the cycle count, which directly correlates to faster computation times.

5.1.6 Consistency in Data Dimensions

The dimensions of the weight, input, and output data were consistent across both tests (4096x4096 for weight and 4096 for both input and output). This ensures a fair comparison between the PIM-enabled and PIM-disabled scenarios.

5.1.7 Total Execution Time

The entire test suite, including setup and teardown, completed in approximately 11.359 seconds. This time includes not only the GEMV operations but also any additional overhead associated with the test framework and simulation environment.

5.1.8 Pass/Fail Status

The test passed successfully, indicating that the GEMV functionality works as expected in both PIM-enabled and PIM-disabled modes.

6 Conclusion

The simulation results clearly demonstrate the performance benefits of utilizing PIM technology for matrix-vector multiplication tasks. By significantly reducing the number of cycles required for the GEMV operation, PIM can lead to faster computations and more efficient processing, making it a valuable approach for high-performance computing applications.

The successful completion of this test and the observed speed-up highlight the potential of PIM technology in accelerating matrix operations, which are fundamental in various computational workloads, including machine learning and scientific computing.

References

[1] SAITPublic. Pimsimulator. https://github.com/SAITPublic/PIMSimulator. Accessed: 2024-07-20.

[2] Samsung Advanced Institute of Technology. Samsung advanced institute of technology. https://www.sait.samsung.co.kr/. Accessed: 2024-07-20.

[3] UMD Memsys. Dramsim2. https://github.com/umd-memsys/DRAMSim2. Accessed: 2024-07-20.