The Breakthrough Memory Solutions for Improved Performance on LLM Inference

Project Implementation

Angel Chuy Cho – M123040116

Outline

Introduction

In this implementation, a PIM Simulator was used to conduct some tests referenced in the paper. Simulations were performed based on the content of the document; however, various physical implementations could not be demonstrated due to the lack of essential tools, such as Samsung's AquaBolt-XL, a CXL-Memory controller, and several other tools mentioned in the paper. Therefore, efforts were made to carry out the work using the resources available and the online PIM simulator. An attempt was made to simulate the functionalities and behaviors discussed in the paper to the best extent possible. This approach allowed for a partial validation, even if the full physical implementation could not be realized.

PIM Simulator

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.

This Simulator was developed by Samsung Advanced Institute of Technology (SAIT) and includes PIM Block which supports various instruction sets as ALU operations (ADD, MUL, MAC, etc..). A PIM Kernel which it is use to generate a set of memory transactions for enabling PIM operations and HBM2 support.

For the installation of the PIM Simulator, it is required to run it in a Linux environment, in this test, Ubuntu Linux was used.

Installation

1. Download or clone the following repository into your pc through linux terminal:

   https://github.com/SAITPublic/PIMSimulator/tree/dev?tab=readme-ov-file#4-programming-guide

2. Once on Linux Environment, run this following command in the terminal for compiling the Simulator:

   ```
   sudo apt install scons
   ```

3. Proceed to run this command for the tests:

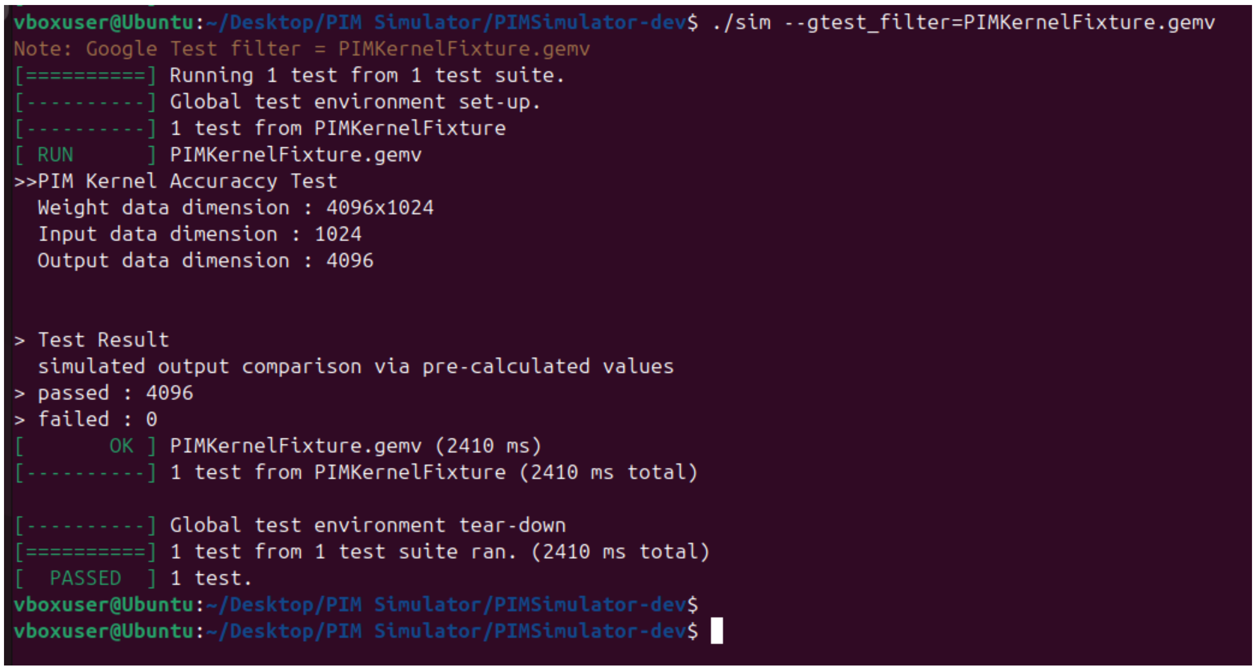
   ```
   sudo apt install libgtest-dev
   ```

4. Once done, install the PIM simulator (Keep in mind, these processes are done inside the PIMSimulator folder):

   ```
   Scons
   ```

5. Now, were done, proceed to run the following command to test if it works:

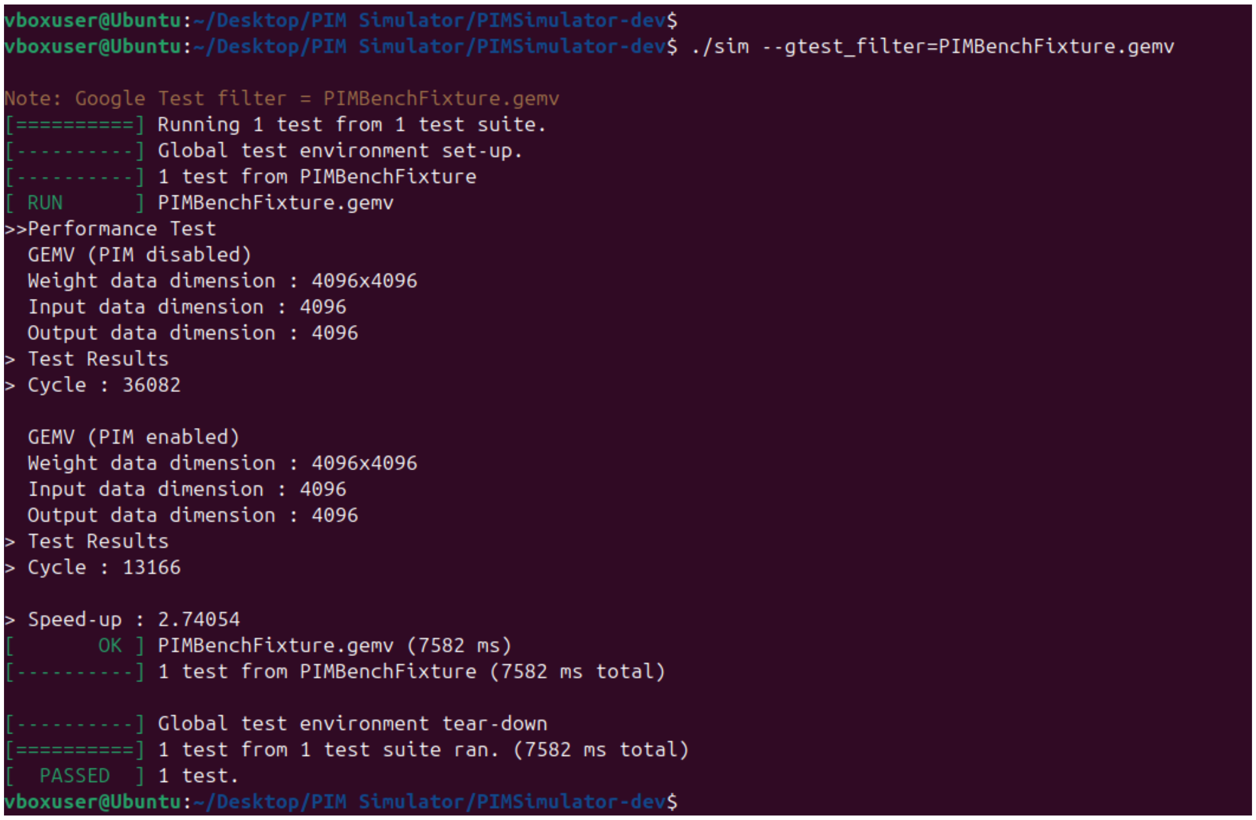
   ```
   # Running: functionality test (GEMV)
   ./sim --gtest_filter=PIMKernelFixture.gemv
   ```



```
vboxuser@Ubuntu:~/Desktop/PIM Simulator/PIMSimulator-dev$ ./sim --gtest_filter=PIMKernelFixture.gemv
Note: Google Test filter = PIMKernelFixture.gemv
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from PIMKernelFixture
[ RUN      ] PIMKernelFixture.gemv
>>PIM Kernel Accuraccy Test
  Weight data dimension : 4096x1024
  Input data dimension : 1024
  Output data dimension : 4096


> Test Result
  simulated output comparison via pre-calculated values
> passed : 4096
> failed : 0
[       OK ] PIMKernelFixture.gemv (2410 ms)
[----------] 1 test from PIMKernelFixture (2410 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (2410 ms total)
[  PASSED  ] 1 test.
vboxuser@Ubuntu:~/Desktop/PIM Simulator/PIMSimulator-dev$
vboxuser@Ubuntu:~/Desktop/PIM Simulator/PIMSimulator-dev$
```

6. Now, to test the performance of the GEMV, we run this command:

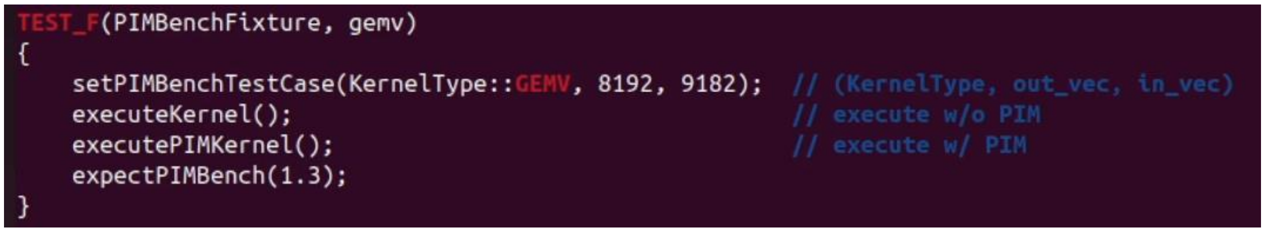
   ./sim --gtest_filter=PIMBenchFixture.gemv



```
vboxuser@Ubuntu:~/Desktop/PIM Simulator/PIMSimulator-dev$
vboxuser@Ubuntu:~/Desktop/PIM Simulator/PIMSimulator-dev$ ./sim --gtest_filter=PIMBenchFixture.gemv

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 (7582 ms)
[----------] 1 test from PIMBenchFixture (7582 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (7582 ms total)
[  PASSED  ] 1 test.
vboxuser@Ubuntu:~/Desktop/PIM Simulator/PIMSimulator-dev$
```

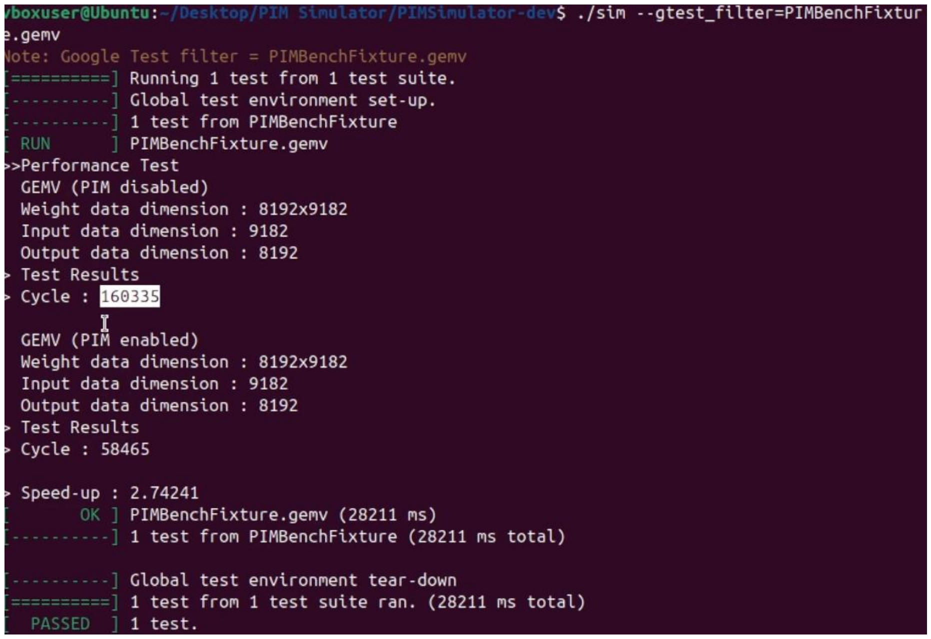
Solutions/ Implementations mentioned in the paper

As one solution mentioned in the paper “To address the memory wall problem, various architectural approaches have been proposed. The solutions involve increasing the number of pins (input–output [I/O]) or raising the I/O frequency in the system”. The data was modify to these value



```
TEST_F(PIMBenchFixture, gemv)
{
    setPIMBenchTestCase(KernelType::GEMV, 8192, 9182);  // (KernelType, out_vec, in_vec)
    executeKernel();                                    // execute w/o PIM
    executePIMKernel();                                 // execute w/ PIM
    expectPIMBench(1.3);
}
```

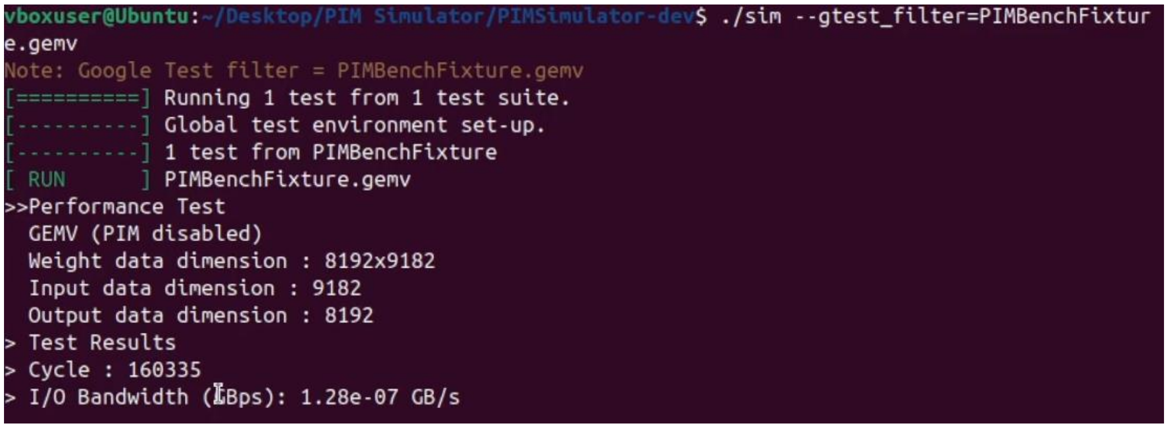
Which simulating it, would give us this result:



```
vboxuser@Ubuntu:~/Desktop/PIM Simulator/PIMSimulator-dev$ ./sim --gtest_filter=PIMBenchFixtur
e.gemv
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 : 8192x9182
  Input data dimension : 9182
  Output data dimension : 8192
> Test Results
> Cycle : 160335

  GEMV (PIM enabled)
  Weight data dimension : 8192x9182
  Input data dimension : 9182
  Output data dimension : 8192
> Test Results
> Cycle : 58465

> Speed-up : 2.74241
[       OK ] PIMBenchFixture.gemv (28211 ms)
[----------] 1 test from PIMBenchFixture (28211 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (28211 ms total)
[  PASSED  ] 1 test.
```

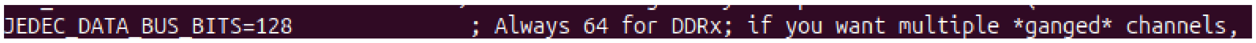
Cycle of 160335 with a speed-up of 2.74.

Printing the Bandwidth (GB/s). we would get this as a result:



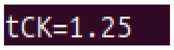
```
vboxuser@Ubuntu:~/Desktop/PIM Simulator/PIMSimulator-dev$ ./sim --gtest_filter=PIMBenchFixtur
e.gemv
Note: Google Test filter = PIMBenchFixture.gemv
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from PIMBenchFixture
[ RUN      ] PIMBenchFixture.gemv
>>Performance Test
  GEMV (PIM disabled)
  Weight data dimension : 8192x9182
  Input data dimension : 9182
  Output data dimension : 8192
> Test Results
> Cycle : 160335
> I/O Bandwidth (GBps): 1.28e-07 GB/s
```

Now, to simulate or demonstrate this part of the paper “The solutions involve increasing the number of pins (input–output [I/O]) or raising the I/O frequency in the system”. To perform this simulation, it was through increasing the **PIN JEDEC_DATA_BUS_BITS** value from 64(Default) to 128. As Increasing this value simulates increasing the number of I/O pins.



```
JEDEC_DATA_BUS_BITS=128             ; Always 64 for DDRx; if you want multiple *ganged* channels,
```

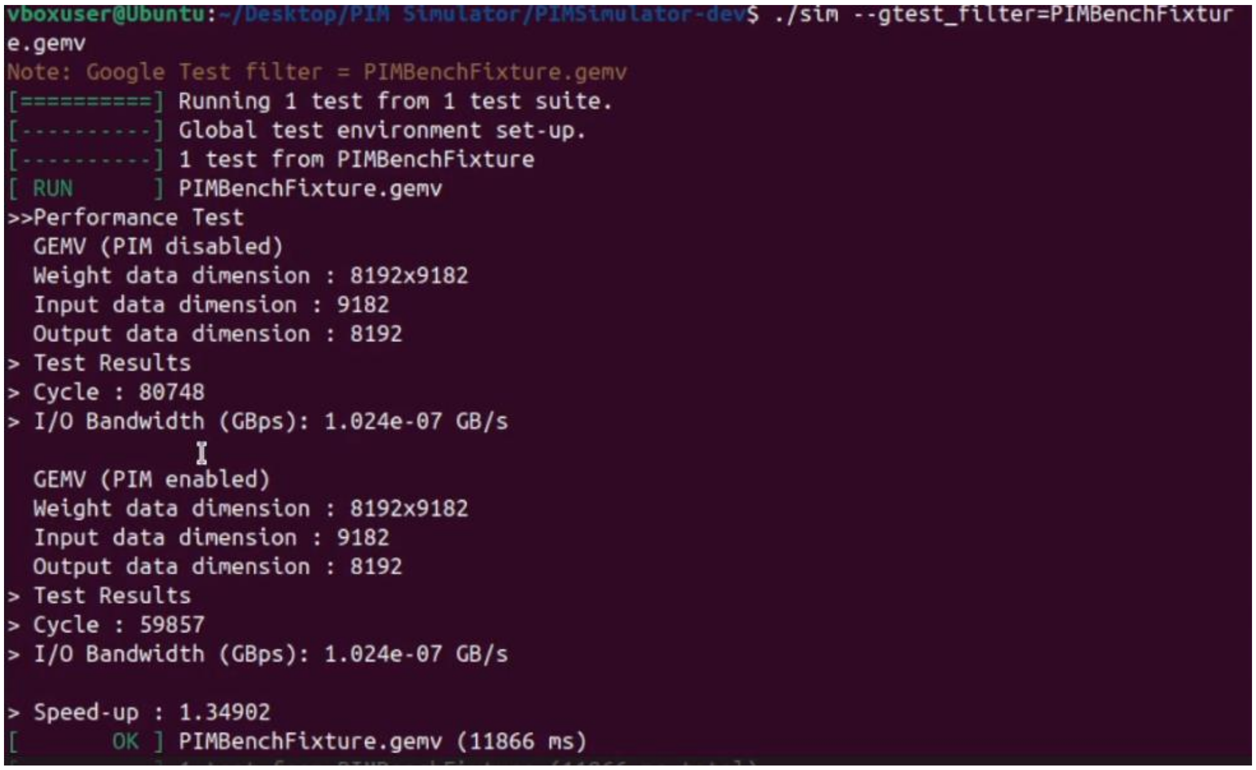
And the number of frequencies of **tCK** to 1.25 as reducing tCK increases the frequency, thereby boosting I/O throughput.



```
tCK=1.25
```

So, the changes made in the system was **PIN JEDEC_DATA_BUS_BITS** (I/O Pins (Bandwidth) and **tCK** (I/O) Frequency

Simulating the test, would give us the output:



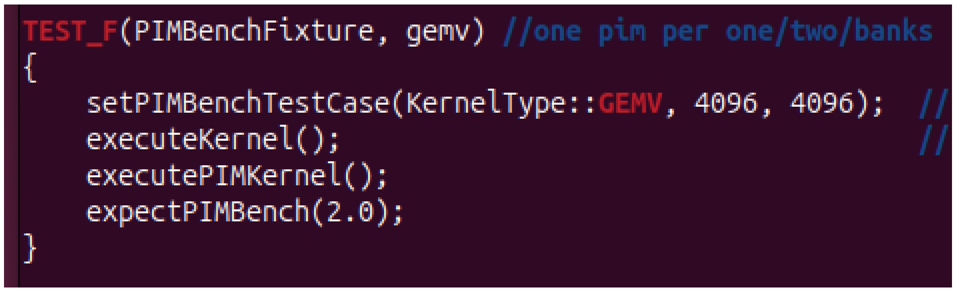
```
vboxuser@Ubuntu:~/Desktop/PIM Simulator/PIMSimulator-dev$ ./sim --gtest_filter=PIMBenchFixtur
e.gemv
Note: Google Test filter = PIMBenchFixture.gemv
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from PIMBenchFixture
[ RUN      ] PIMBenchFixture.gemv
>>Performance Test
  GEMV (PIM disabled)
  Weight data dimension : 8192x9182
  Input data dimension : 9182
  Output data dimension : 8192
> Test Results
> Cycle : 80748
> I/O Bandwidth (GBps): 1.024e-07 GB/s

  GEMV (PIM enabled)
  Weight data dimension : 8192x9182
  Input data dimension : 9182
  Output data dimension : 8192
> Test Results
> Cycle : 59857
> I/O Bandwidth (GBps): 1.024e-07 GB/s

> Speed-up : 1.34902
[       OK ] PIMBenchFixture.gemv (11866 ms)
```

We can see that it performs better than the default values.

	Default Values (Default Bandwidth, Frequency)	**Increased I/O (for bandwidth and frequency**)
Cycle	160335	80748
I/O Bandwidth (GB/s)	1.28e-07	1.024e-07
Speed-up	2.7421	1.34902

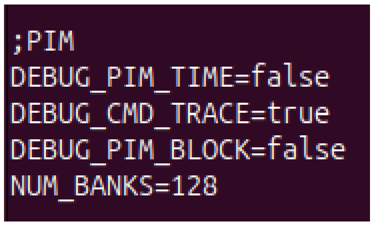
So, we can conclude that by increasing the number of pins or raising the I/O frequency in the system can be a solution for memory wall problem as mentioned in the paper. Now following with another solution mentioned, according to what is it mentioned in the pdf “While we have demonstrated the performance and energy characteristics using one PIM block for every two banks, it is also possible to deploy one PIM block for each bank. In such a scenario, there is the potential to double the internal memory bandwidth, which means that the PIM can achieve twice the processing power than before” this refers that Deploying one PIM block per memory bank, instead of one for every two banks, could double the internal memory bandwidth, allowing the PIM to achieve twice the processing power.

For this part, we modify the file PIMBenchTestCases.cpp by adding two tests, one with one_pim_per_bank and one_pim_per_two_banks to prove that we can achieve twice the processing power with one pim per bank



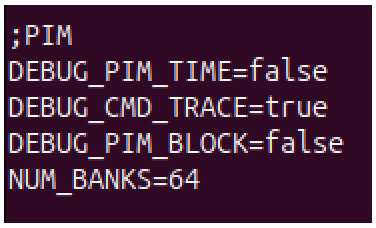
```
TEST_F(PIMBenchFixture, gemv) //one pim per one/two/banks
{
    setPIMBenchTestCase(KernelType::GEMV, 4096, 4096);  //
    executeKernel();                                     //
    executePIMKernel();
    expectPIMBench(2.0);
}
```

In the one_pim_per_bank and one_pim_per_two_banks running at a speed of 2.0.



```
;PIM
DEBUG_PIM_TIME=false
DEBUG_CMD_TRACE=true
DEBUG_PIM_BLOCK=false
NUM_BANKS=128
```

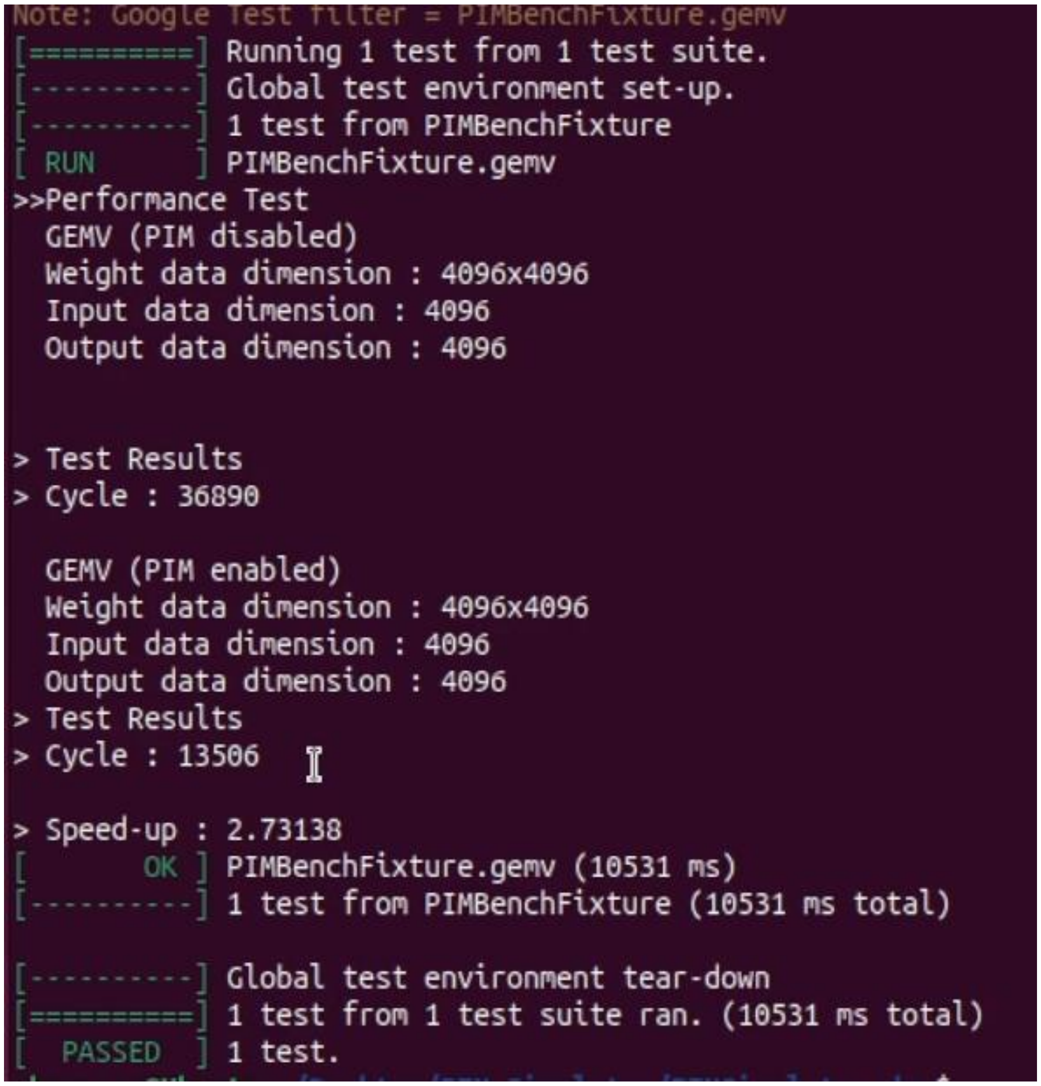
Number of banks for one_pim_per_bank will be 128



```
;PIM
DEBUG_PIM_TIME=false
DEBUG_CMD_TRACE=true
DEBUG_PIM_BLOCK=false
NUM_BANKS=64
```

Number of banks for one_pim_per_two_banks will be 64 since its one pim per 2 banks=128

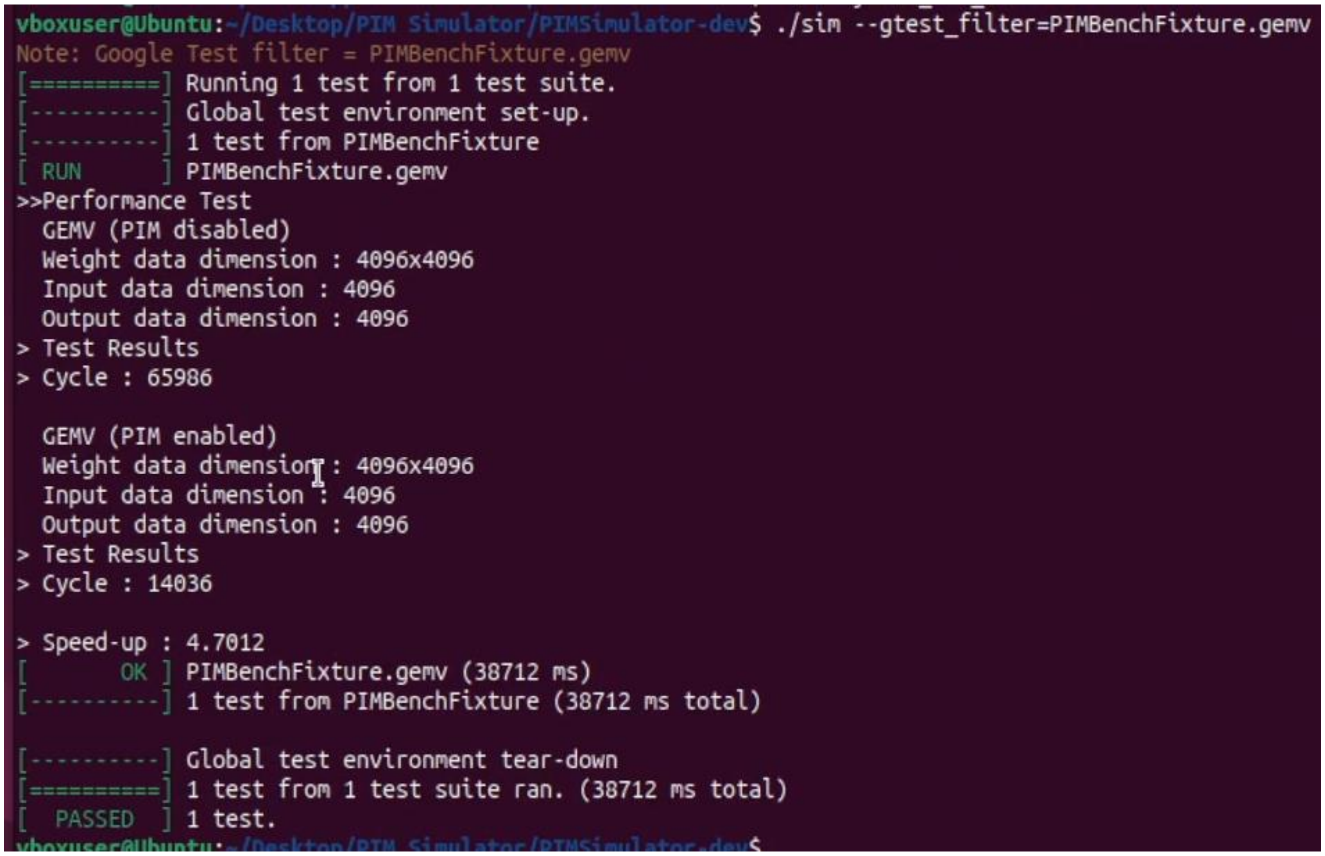
Result for ONE_PIM_PER TWO_BANKS



```
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 : 36890

  GEMV (PIM enabled)
  Weight data dimension : 4096x4096
  Input data dimension : 4096
  Output data dimension : 4096
> Test Results
> Cycle : 13506

> Speed-up : 2.73138
[       OK ] PIMBenchFixture.gemv (10531 ms)
[----------] 1 test from PIMBenchFixture (10531 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (10531 ms total)
[  PASSED  ] 1 test.
```

Using NUM_BANKS=64. Which is 128/2= 64 (Per two banks), we get a speed-up of 2.73138

Result for ONE_PIM_PER_BANK



```
vboxuser@Ubuntu:~/Desktop/PIM Simulator/PIMSimulator-dev$ ./sim --gtest_filter=PIMBenchFixture.gemv
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 : 65986

  GEMV (PIM enabled)
  Weight data dimension : 4096x4096
  Input data dimension : 4096
  Output data dimension : 4096
> Test Results
> Cycle : 14036

> Speed-up : 4.7012
[       OK ] PIMBenchFixture.gemv (38712 ms)
[----------] 1 test from PIMBenchFixture (38712 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (38712 ms total)
[  PASSED  ] 1 test.
vboxuser@Ubuntu:~/Desktop/PIM Simulator/PIMSimulator-dev$
```

Using NUM_BANKS=128 (Default) gives us a speed up of 4.7012

In summary, we can see the one_pim_per_bank is faster over the one_pim_per_two_banks as mention in the paper. Meaning the enhanced system (PIM-enabled for ONE_PIM_PER_BANK) runs 4.7012 times faster than the baseline while ONE_PIM_PER_TWO_BANKS only runs 2.70 faster than the baseline meaning that one_pim_per_bank is faster over the one_pim_per_two_banks as mentioned above.

Conclusion

In conclusion, the use of this tool PIM Simulator provided a opportunity to and demonstrate some of the solutions mentioned in the paper. However, due to the tools that were used in the real-world implementation such as Samsung's AquaBolt-XL, a CXL-Memory controller, and other key components, it was not possible to implement all solutions mentioned in the paper due to materials and tools used as mentioned above.