1. About PIMSim

PIMSim is a trace-based simulator of PIM architecture, which provides both host-side and memory-side simulation. Besides, PIMSim provides high degree of freedom of PIM unit configuration which can be defined by users. PIMSim provides three input types to adapt different simulation demanding. PIMSim aims to support new memory types such as HMC, HBM et.

1. What can PIMSim do?

PIMSim provides simulation of Process-In-Memory(PIM) and applies to the following groups:

* People who want to get insight of PIM architecture.
* Researchers to build new PIM designs.
* Programmers who want to discover the potential of PIM architecture.

PIMSim can provide following experimental statistics information:

* PIM
  + Speed up while applying new PIM design
  + Off-Chip Bandwidth
  + Message statistics
  + Internal bandwidth
  + Energy (under development)
  + PIMProcessor simulation details
  + Pipeline simulation details
  + Data Coherence of PIM architecture
  + Program partitions details
* Processor
  + Cache simulation details
  + MSHR simulation details
  + ALU simulation details
  + Hybrid memory simulation details

To take a deep understanding of PIMSim, we’ll show some cases to take advantage of PIMSim:

Case1: Develop new PIM design.

PIMSim is a simulator that simulates host-side CPU, memory, and PIM units (processors or computational logic) at the same time. You can implement your design logic in PIMSim and it will bring you detailed simulation statistics.

Case2: Test programs and explore the potential of PIM architecture.

If you want to know the performance of your code on PIM architecture, or test your program partitions for best performance, you can use PIMSim to partition your programs and easily get your performance results.

Case3: Get insight of PIM architecture

If you want to know the internal bandwidth of PIM or the detailed off-chip traffic, you can use PIMSim to get this information.

1. Getting started with PIMSim
   1. Prepare input traces.

PIMSim is a trace-based simulator. All the trace filename should be “ CPUx.trace ” (x indicate the id of CPU, or you can customize your own inputs). Detailed trace input is aim to get detailed simulation of instructions and cycles. General trace input is designed for best simulation speed and the situations that only cares about memory behaviors. The traces can be fetched by either running programs on full-system simulator or dynamic program analysis tools on physical machines. For each trace line, it should provide cycle, instruction, [read/write], [Data], [Address]. The square bracket mean this information can be null.

0|ld t1, SS:[rsp]|R|D=0x0000000000000001 A=0x7de145ffee20

1|addi rsp, rsp, 0x8

2|ld t1, SS:[rsp]|R|D=0x0000000000000001 A=0x024ee20

4|mov rdx, rdx, rsp

5|limm t1, 0xfffffffffffffff0

6|and rsp, rsp, t1

7|st rax, SS:[rsp + 0xffdfff8]|W|D=0x0000000000000000 A=0x04fee18

There are four input labels are applied to indicate target operation is executed at memory-side:

1. You can use “PIM\_Operation” to indicate that this instruction should be executed in memory-side units, like this:

26|PIM\_st r14, SS:[rsp + 0xffffff8]|W|D=0x0000000000000000 A=0x7fffffffedf8

1. If you want to nominate a snippet of code to be executed in memory-side, you can label traces like this:

PIM\_BLOCK\_START

18|rdip t7, %ctrl153,

19|st t7, SS:[rsp + 0xffffffffff8]|W|D=0x00000000004001ba A=0x7fffffffee08

20|subi rsp, rsp, 0x8

21|wrip , t7, t1

22|st r15, SS:[rsp + 0xfffffffff8]|W|D=0x0000000000000000 A=0x7fffffffee00

PIM\_BLOCK\_END

1. If you don’t know the actual instructions executed but know the memory access address, we provide **function** to help. You can add this to use function:

3|Function\_Add\_Start ; cycle/function name

input = 0x7fffffffee20 ;input address

input = 0x7fffffffee18 ; input address

output = 0x5807b0 ;output address

latency = 2 ;operation latency without data fetching

Function\_Add\_End

This input indicates a function named 'Add', which has two inputs and one output. The operation duration lasts 2 cycles.

1. You can customize your own policy by modifying PIMConfigs.cs. For example, you can specify a certain kind of instruction to be executed at memory-side by modifying PIMConfig\ PIM\_Ins\_List
   1. Prepare the configuration files

PIMSim has two kinds of configuration inputs: PIM settings and RAM settings. For PIM settings, you can attach them by PIMConfig Class in "/Configs/PIMConfigs.cs". For RAM settings, we modified and integrate HMCSim (https://github.com/tactcomplabs/gc64-hmcsim) and DRAMSim2 (https://github.com/dramninjasUMD/DRAMSim2) to adapt PIM Simulation. If you need simulate HMC or DRAM (or both of them), you should provide configure files of them. Detailed document of HMCSIM and DRAMSim2 can be found on their GitHub. In the future, we'll add more new memory simulations such as NVM, Memristor and so on.

* 1. Building PIMSim

To build PIMSim, you can locate the root folder of PIMSim and type:

$make

To clean the previous buildings, you can type:

$make clean

3.4 Running PIMSim

You can run PIMSim by providing such parameters:

$PIMSim -t tracefilepath -config configfilepath –o outputfile –n processorcount –c cycle

-t, -trace FILEPATH specify the path folder of input trace.

-config FILEPATH specify the path folder of input configs.

-o, -output FILENAME specify the file name of output file."

-n, -N PROCCOUNT specify the count of host proc."

-c, -cycle CYCLES specify the execution cycles.

4. Data Flow Diagram

4.1 Overview



4.2 Processor and PIMProcessor



4.3 TraceFetcher



4.4 Instruction Partitioner

