

APPENDIX A

ARTIFACT DESCRIPTION APPENDIX: [PERFORMANCE EVALUATION OF THE NVIDIA TESLA V100: BLOCK LEVEL PIPELINING VS. KERNEL LEVEL PIPELINING]

A. Abstract

This artifact comprises the source code, hardware information and build instructions that can be used to reproduce the test results presented in SC’2018 Poster **Performance Evaluation of the NVIDIA Tesla V100: Block Level Pipelining vs. Kernel Level Pipelining**

B. Description

1) Check-list (artifact meta information):

- **Algorithm:** 2D Convolution, Generic Matrix-Matrix Multiplication
- **Program:** CUDA code
- **Compilation:** NVIDIA nvcc
- **Run-time environment:** CUDA 9.0.176
- **Hardware:**
 - Machine 1:
 - CPU: IBM Power9 processors
 - GPU: NVIDIA TESLA V100 NVLink2
 - Machine 2:
 - CPU: Intel Xeon Gold 6136
 - NVIDIA TESLA V100 with PCI-E
- **Output:** elapsed running time, bandwidth throughput
- **Experiment workflow:** git clone projects; run test, observe the results
- **Publicly available?:** Yes

2) *How software can be obtained (if available):* The bandwidth test and prototype artifacts can be found in the blockpipe public project on bitbucket.org.

3) *Hardware dependencies:* For adequate reproducibility, we suggest an NVIDIA V100 GPU with Intel Xeon Skylake CPU for PCI-E connection or IBM power9 CPU with NVLink2 connection. We suggest the NVIDIA GPU Driver to be newer than 396.31. We suggest to use CUDA Toolkit 9.0.176 for the test.

4) *Software dependencies:* The evaluation experiments requires the CUDA GPU driver, nvcc CUDA compiler included with the CUDA Toolkit 9.0.176.

C. Installation

First, you must install all the dependencies including GPU drivers, CUDA toolkits, etc.

Secondly, you must clone the source code to the local machine:

```
$ git clone https://bitbucket.org/superwizard2017/blockpipe.git
```

Finally, you can access the repository to start the experiments.

D. Experiment workflow

The experiment has two sections: bandwidth test and prototype evaluation.

Bandwidth test includes cudaMemcpy, cudaMemcpy2D, Direct Access Contiguous and Direct Access Non-contiguous.

You may vary the hyper parameters in the code to see the bandwidth throughputs with different parameters (NUM_CHUNK, NUM_STREAM, etc.)

A Makefile is provided in each directory. You may need to make sure the CUDA compiler is included in the system PATH.

We have implemented prototypes using our approach for 2 benchmarks: 2D Convolution and Generic Matrix-Matrix Multiplication.

For 2D Convolution, we have three versions. We denote the default synchronous version as "baseline" version, the asynchronous version using the CUDA stream kernel pipeline technique as "async" version. The "spin" version is our block level pipeline version. The hyper parameters can be tuned to in most of the versions.

For Generic Matrix-Matrix Multiplication, we denote the default synchronous version with traditional matrix-matrix multiplication algorithm as "baseline" version. We implement a block matrix-matrix multiplication algorithm with GPU shared memory optimization and denote it as "baseline-block" version. After that, we apply the traditional kernel level pipeline technique and our block level pipeline technique to this then get "block-async" and "block-spin" versions.

E. Evaluation and expected result

The expected results include GPU memory copy bandwidth throughput and prototype elapsed running time. All time data are measured by CUDA Events with cudaEventElapsedTime() function.

F. Notes

For simplicity propose, we have only tested some hyper parameters with multiples of 2 under the data set size limit.