

APPENDIX A

ARTIFACT DESCRIPTION: Compiling SIMT Programs on Multi- and Many-core Processors with Wide Vector Units: A Case Study with CUDA

Hancheng Wu

North Carolina State University
hwu16@ncsu.edu

John Ravi

North Carolina State University
jjravi@ncsu.edu

Michela Becchi

North Carolina State University
mbecchi@ncsu.edu

A. Abstract

This description contains instructions on how to reproduce the results reported in the SC'18 Poster "Compiling SIMT Programs on Multi- and Many-core Processors with Wide Vector Units: A Case Study with CUDA". We show the hardware and software configurations used in all 4 platforms and explain how to run microbenchmarks and Real-world applications

B. Description

1) Check-list (artifact meta information):

Algorithm: Various applications written in CUDA

Program: C++ Binary and CUDA Code

Compilation: nvcc and icpc

Data set: the original dataset from the referenced papers

Execution: run the bash script in each program folder

Output: running time

Experiment workflow: Set up the platform; Compile the code; Run the script provided

Experiment customization: [-Affinity|-A] affects how many Hardware Threads to invoke for each X86 core.

Publicly available?: Yes. By QR code or URL link.

2) How software can be obtained:

The softwares needed to run our experiments are Nvidia CUDA tool kit and Intel Parallel Studio XE. They can be downloaded from Nvidia and Intel websites, respectively. Both are free for non-profit use by students.

Our code repository can be acquired at github: at: https://github.com/HanchengWu/CUDA_on_Manycore.git.

3) Hardware dependencies:

Experiments are performed on devices specified in Table I.

4) Software dependencies:

Our CUDA programs are compiled using CUDA 9 tool kit. The transformed programs that run on Intel devices are compiled using Intel Parallel Studio 17 (for Intel Xeon Phi coprocessor) and

Intel Parallel Studio 18 (for Intel Xeon Phi processor and Intel Xeon Skylake processor).

We compile all codes with optimization level -O3. All the devices except for the Skylake processor run on a system with REHL 7.5, kernel v3.10. The Skylake processor is on Google Cloud (select 24 logical cores) with Ubuntu 18.04 LTS, kernel v4.15

5) Datasets:

We use public benchmarks and we use input datasets from original papers that are referenced. They are listed below.

S. Che, M. Boyer, and J. Meng, "Rodinia: A benchmark suite for heterogeneous computing," presented at the IEEE International Symposium on Workload Characterization, Austin, TX, USA, 2009.

H. Wu and M. Becchi, "An Analytical Study of Recursive Tree Traversal Patterns on Multi- and Many-Core Platforms," presented at the 2017 IEEE 23rd International Conference on Parallel and Distributed Systems, Shenzhen, China, 2017.

For convenience, whenever possible, we also include the tools that generate the input dataset for each application in the corresponding application directory. Refer to the README file in each corresponding directory to see how to generate or modify the application-specific input dataset.

C. Installation

1) Install necessary tool kits

For experiments on Nvidia GPUs, install CUDA tool kit from Nvidia website:

<https://docs.nvidia.com/cuda/#installation-guides>

TABLE I. PLATFORM SUMMARY

Category	Architecture	Device	Abbr
Nvidia GPU	Fermi	Tesla C2070	<i>Gfer</i>
	Maxwell	GTX TITAN X	<i>Gmax</i>
	Pascal	TITAN XP	<i>Gpas</i>
Intel MIMD/SIMD Hybrid	Knights Corner	Xeon Phi 5110P	<i>Hco</i>
	Knights Landing	Xeon Phi 7210	<i>Hpro</i>
	Skylake	Xeon Skylake (Google Cloud)	<i>Hsky</i>

For experiments on Intel devices, install Intel Parallel Studio from: <https://software.intel.com/en-us/parallel-studio-xe/choose-download>

2) Install our Git repository

Type the following in the command line:

```
$ git clone https://github.com/HanchengWu/CUDA\_on\_Manycore.git
$ cd CUDA_on_Manycore
```

Microbenchmarks are organized in folder *microbenchmarks*. Real-world applications, based on which platform is used, are organized in four folders. They are *gpu*, *xeonCo*, *xeonPro*, *skylake* which containing codes that run on Nvidia GPUs, Intel Xeon Phi coprocessor, Intel Xeon Phi processor and Intel Xeon Skylake processor, respectively.

Suppose we want to run BFS on Skylake, go to folder *skylake/vector/bfs*, type *make*, then execute *benchmark.sh*.

```
$ cd repository_folder/skylake/vector/bfs;
$ make;
$ ./benchmark.sh;
```

D. Experiment workflow

After the installation and compilation, we can run the .sh bash file in each application folder to run the program.

E. Evaluation and expected result

We average our results over 10 runs. The expected result of each run is the kernel execution time printed on the command line.

F. Experiment customization

We can specify [-Affinity|-A *affinity_num*] on the command line when we run and tune an application. The parameter *affinity_num* controls the number of hardware threads that will be invoked on each x86 core. The legal range is 1-4 for Intel Xeon Phi (co)processors and 1-2 for Intel Xeon Skylake processor. Using a low value for *affinity_num* will lead to less hardware threads to be spawned, therefore result in low device resource utilization.