POSTER: GPU based Near Data Processing for Image Processing with Pattern Aware Data Allocation and Prefetching

Jungwoo Choi*, Boyeal Kim*, Ji-Ye Jeon*, Hyuk-Jae Lee*, Euicheol Lim†, Chae Eun Rhee‡
*Seoul National University, {choijw8525, bykim, jyj0805, hjlee}@capp.snu.ac.kr
†SKHynix, euicheol.lim@sk.com, ‡Inha University, chae.rhee@inha.ac.kr

***Abstract*—This paper proposes a GPU-based Near-data-processing (NDP) architecture as well as a well-matched programming model considering both the characteristics of image applications and NDP constraints. First, data allocation to the processing unit is handled to keep the data locality considering the memory access pattern. Second, this predictable allocation enables to design a compact but efficient NDP architecture. By applying a prefetcher that leverages the pattern aware data allocation, the number of active warps and on-chip SRAM size of NDP are significantly reduced. This allows to satisfy the NDP constraints and increases the opportunity to integrate more processing units on a memory logic die. The evaluation results for various image processing benchmarks show that the proposed NDP GPU improves the performance compared to the baseline GPU.**

I. Introduction

Recent applications with high resolution and frame rates have very large energy and processing time overhead because of the data movement between the processor and the off-chip memory [1]. Near-data-processing (NDP) is a promising alternative for reducing the overhead caused by data movement. Programmable GPU-based NDP is attractive, because it is capable of processing more various algorithms than hardware accelerators. Recent studies have mainly been conducted to accelerate specific applications such as deep neural network (DNN) [2] or 3D rendering [3]. Therefore, previous NDP studies may not be extensively applied to various image processing algorithms. NDP for image processing has not been sufficiently studied despite the strong demand for efficient image processing in many fields.

In this paper, a GPU-based NDP architecture and a programming model are proposed. The two approaches are complementary and they are optimized together. In a general GPU, a large size of register is needed for latency hiding. However, a GPU dedicated to image processing can be made more compact than general GPUs while maintaining a high performance level. The proposed programming model allocates data to the processing unit by considering the memory access pattern and data locality of the image processing algorithms. Taking advantage of this predictable allocation, an access pattern aware prefetcher is added, and the hardware resources are greatly reduced.

II. Proposed NDP GPU

The proposed GPU-based NDP allows not only to reduce the energy consumption by decreasing the memory access overhead but also to exploit the wide memory bandwidth. Also, careful design efforts are made to consider the 3D stacked memory environment including the limited area and power constraints of the logic die.

A. Access Pattern Aware Data Allocation

To take advantage of data access pattern in image processing applications, cooperative thread array(CTA)-to-streaming multiprocessor(SM) allocation is performed based on the data area information processed by CTAs. The allocation schemes are classified into row-major, column-major, and tile-based. Software-based data allocation uses the feature that CTA is not deallocated before the job assigned to that CTA is finished. Only a single CTA is allocated to each SM, and each CTA processes their data through loop iteration. Fig. 1 shows an example of software-based realization of row-major data allocation. The left side of Fig. 1 represents that the entire image is divided into four areas, which are allocated to four SMs. The right side of Fig. 1 magnifies the image area allocated to SM1 where each of the boxes represents the area processed by one CTA, with the width and height of *blockDim.x* and *blockDim.y*, respectively. The area surrounded by a gray dotted line is called a *row block*. Firstly, the initial coordinate of the area allocated to each SM is calculated (*Step 1*). Then, the top-left coordinate for the first CTA execution denoted by a triangle is set to be the initial coordinate (*Step 2*). Following the

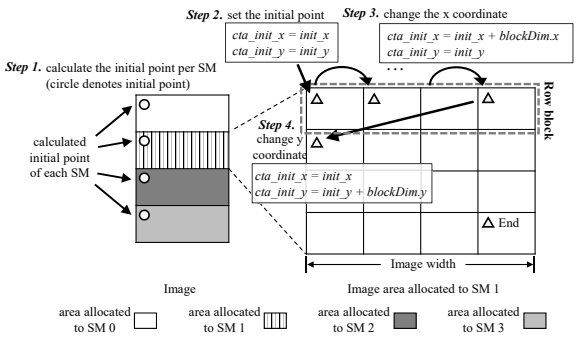

Figure 1: Software-based row major pattern aware data allocation example.

978-1-7281-3613-4/19/$31.00 ©2019 IEEE
DOI 10.1109/PACT.2019.00049

CTA execution, the x-coordinate is moved in the row-major direction (*Step 3*). *Step 3* is repeated until the execution is finished in the first *row block*, and then the process is moved to the next *row block* (*Step 4*). *Step 3* and *Step 4* are repeated until the execution is finished for the entire data area allocated to SM0.

B. NDP GPU Architecture

SM structure: The number of warps and the size of on-chip SRAM that are appropriate for the SM structure of NDP GPU are proposed in this section.

(1) Pattern-aware prefetcher: A stride-based prefetcher is mounted inside the SM of NDP GPU. Because the data access pattern of the algorithm is preserved through the proposed data allocation, accurate prefetching is provided with a simple stride-based prefetcher.

(2) The number of active warps: The access pattern-aware data allocation and prefetching make the necessary data readily available. Accordingly, as the stall decreases, the latency may be hid with a smaller number of warps in comparison with the baseline GPU.

(3) On-chip SRAM size: The size of on-chip SRAM including register, L1 Data cache, and shared memory is reduced by using a small number of warps.

Off-chip L2 cache: The proposed NDP GPU uses no L2 cache. Data allocation and the prefetcher in the proposed NDP GPU help to increase the hit rate of the L1 cache up to almost 100%. This reduces the need for L2 cache which is used to mitigate the miss penalty of L1 cache in the memory hierarchy.

The number of SMs: Area constraints of the logic die should be considered along with the performance and energy consumption to determine the number of SMs for NDP GPU.

III. Evaluation

A cycle-accurate simulator, GPGPU-Sim [4], is modified to verify the impact of the proposed NDP architecture. Four configurations are evaluated as listed below.

- Host: is a conventional GPU. It is similar to the NVIDIA Maxwell GPU [5] and has 64 SMs.
- B-NDP-L2: denotes a baseline NDP with L2 cache. It has the same SM architecture with Host, but is located on the logic die.
- B-NDP-woL2: denotes a baseline NDP where L2 cache is not used. Other configuration is same with B-NDP-L2.
- IP-NDP: is the proposed NDP specialized for image processing. It has a prefetcher and a reduced-size SRAM.

Six commonly used image processing algorithms are tested [6], [7]. For evaluation of IP-NDP, code is modified according to the proposed method.

Fig. 2 shows the increase in speed for four configurations normalized to Host. Two observations from this result are explained below. First, image processing benefits from the increased bandwidth. B-NDP-L2 shows a 2.06× performance improvement on average over Host. Second, the proposed IP-NDP outperforms other configurations. IP-NDP is 3.09× (up to 5.85×), 1.44× (up to 1.97×), and 1.31× (up to 2.01×) faster than Host, B-NDP-L2, and B-NDP-woL2, respectively. Thus, it can be concluded that the proposed pattern-aware data allocation and NDP GPU works very well. Although not shown in this paper, the proposed scheme consumes 52% less energy than Host due to less static power and fast execution, while requires only 57.3% of the area compared to Host.

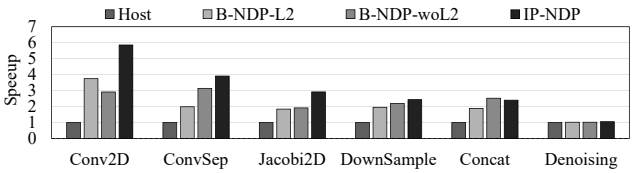

Figure 2: Speedup for B-NDP-L2, B-NDP-woL2, IP-NDP with varying the number of SMs in the NDP stack. All results are normalized to Host.

Acknowledgment

This paper was result of the research project supported by SK hynix Inc and by the R&D program of MOTIE/KEIT. [No. 10077609, Developing Processor-Memory-Storage Integrated Architecture for Low Power, High Performance Big Data Servers]

References

[1] I. K. Park, N. Singhal, M. H. Lee, S. Cho, and C. Kim, "Design and performance evaluation of image processing algorithms on gpus," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 1, pp. 91–104, 2011.

[2] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," in *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1. ACM, 2017, pp. 751–764.

[3] C. Xie, S. L. Song, J. Wang, W. Zhang, and X. Fu, "Processing-in-memory enabled graphics processors for 3d rendering," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 637–648.

[4] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE, 2009, pp. 163–174.

[5] N. G. GTX, "980: Featuring maxwell, the most advanced gpu ever made," *White paper, NVIDIA Corporation*, 2014.

[6] NVIDIA, "Nvidia cuda sdk 4.2," 2011.

[7] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a high-level language targeted to gpu codes," in *2012 Innovative Parallel Computing (InPar)*. Ieee, 2012, pp. 1–10.