Dynamic Multi-Resolution Data Storage *

Yu-Ching Hu
Department of Computer Science and Engineering
University of California, Riverside

Murtuza Lokhandwala
Department of Electrical and Computer Engineering
North Carolina State University

Te I
Google

Hung-Wei Tseng
Department of Electrical and Computer Engineering
University of California, Riverside

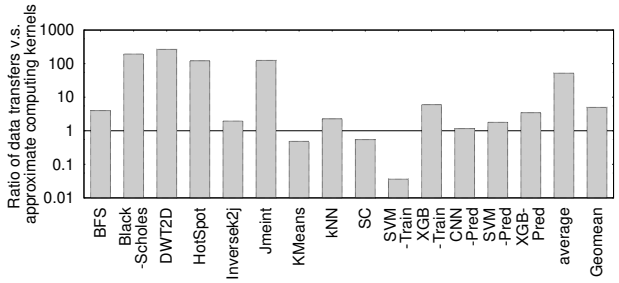

Figure 1: The data-exchange overhead compared against the execution time of performing compute kernels on the same amount of data.

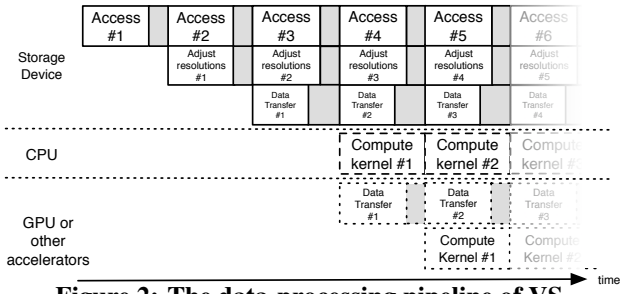

Figure 2: The data-processing pipeline of VS.

1. INTRODUCTION

Approximate computing is gaining traction in commercialized systems because many applications can now tolerate small errors in input data. The concept of approximate computing is to receive fewer details from the raw data, processors and hardware accelerators that use approximate computing can make trade-offs in accuracy to improve performance, energy, power and cost by applying simplified circuits or reducing computation. Therefore, approximate computing creates a new demand of presenting datasets in different resolutions, meaning details about the raw data (e.g., data precision levels, summarized results, intermediate results, and sampled contexts).

As existing approximate-computing research mainly focuses on the kernel computation while leaving other architectural components remaining the same as exact computing, the storage device must faithfully present raw data in the system main memory before computation starts. Due to the limited number of system interconnect links, the part of transferring raw data through SSD's external PCIe links bottlenecks the whole application performance. With recent approximate-computing-inspired hardware accelerators (e.g., NVIDIA's latest Tensor Cores [1]) significantly shrinking the execution time in compute kernels, the data-preparation overhead has overtaken compute kernels as a new bottleneck in many applications as in Figure 1. However, as the application only needs lower-resolution data, sending raw data to the main memory is more than what the application really needs. To fundamentally address the aforementioned bottleneck in approximate computing, the storage device needs to work with the running application to deliver datasets in the required resolution.

This paper introduces a holistic, hardware-software co-design system architecture, *Varifocal Storage* (VS), that dynamically adjusts the data resolution for the demand of applications inside the storage device. By extending approximate computing to all layers within a system, we can achieve better performance but still maintain the flexibility and quality without additional costs. We revisit the task allocation of approximate computing on general-purpose computers to place tasks such as raw-data retrieval, data-resolution adjustment and quality control in the most appropriate place within a system in full-stack system design perspective. The storage device can work directly with the running application to deliver datasets in the desired resolution before going through the interconnect to save the bandwidth demand from the data source as well as to decrease the most latency-critical data-transfer overhead.

We evaluate VS by running a wide range of applications on our prototype SSD. VS achieves 1.52X speedup on average over conventional approximate computing.

2. VARIFOCAL STORAGE SYSTEM ARCHITECTURE

Figure 2 demonstrates the data processing pipeline in VS. By using operators and quality-control mechanisms inside the storage device, VS exploits the richer internal bandwidth and idle processing power to efficiently adjust/prepare datasets in lower resolutions for approximate computing applications. Instead of always sending raw data, VS allows the storage device to send adjusted datasets to the system main memory. Then, the approximate computing components can directly work on lower resolution data, without additional conversion. In this way, VS reduces the total latency of transferring data over the system interconnect since the size of lower-resolution data is smaller than raw data. VS also mitigates the idle time in compute units and frees up CPU resources to tackle more useful workloads, leading to performance gains for approximate applications on the host side.

This section will briefly describe the programming model, operators, quality control mechanisms, and architectural support of VS.

Programming model To prepare an application to take advantage of the VS model, the programmer uses the VS library to specify data resolutions and retrieve adjusted data for the application. These library functions help the application to set up (1) the operators required to read data and whether any quality control mechanism is enabled, and (2) the parameters that allow the underlying storage device to adjust data as well as control variables that quality control mechanisms use to control the adjusted data.

*Dynamic Multi-Resolution Data Storage. In the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019. Best paper honorable mention.

VS operators VS provides a set of operators to adjust data resolutions and expose these operators through the NVMe interface as well as the system API. VS-operators are selected under the following criteria: (1) The computation overhead must match the processing power inside the storage device. Therefore, VS can minimize the impact on access latency and power consumption and avoid extra hardware costs. (2) A wide range of applications must be able to apply the operator, thereby allowing for more efficient use of valuable device resources (VS identifies the most useful operators from previous efforts [2, 3]). (3) The operator must allow VS to take advantage of mismatches between external and internal bandwidths and downsize the outgoing data. These operators can flexibly support various resolutions and accommodate exact computing.

First-level quality control in the storage devices If the input quality is way too far from the origin, the approximate computing can hardly generate meaningful results [4, 5, 6, 7]. Therefore, we designed two quality control mechanisms, Autofocus and iFilter, both control the quality of adjusted data, avoiding the cases of sending data that fail to pass the quality control knobs to the host before the computation occurs. In contrast, all previous research projects censor computation results and always require at least part of raw data to present in the host main memory as well as being computed.

Autofocus allows the programmer to simply specify the desired VS-operator, letting VS decide the most appropriate resolution that every checked piece of the adjusted data successfully passes through the pre-defined, operator-dependent threshold values. If low-resolution data failed on the quality control knobs, VS will reject low-resolution data and gradually use higher resolutions until the quality fits the demand and apply this resolution for the same dataset later. Utilizing another important observation from previous research that a small subset of input data is representative of the rest of the input data in approximate-computing applications that tolerate inaccuracies [5], Autofocus select the resolution using only a small portion of the raw input data from a requested dataset and then monitor the quality of the adjusted input data.

iFilter can work without programmer input and is more effective than Autofocus for applications having compute kernels that are compatible with multiple VS-operators. The iFilter algorithm is similar to the Autofocus algorithm in that it selects the most appropriate resolution for each compatible operator, except that iFilter will keep track of the resolution and the resulting data size for each operator. After selecting an operator that passes all quality control variables and generates the smallest data size among all passing operators, iFilter will enter the monitoring phase as in Autofocus.

Building a VS-compliant storage device Building a VS-compliant storage device means tackling challenges associated with (1) providing a hardware/software interface that allows applications to describe the resolutions and quality of the target data, and (2) minimizing the computational overhead/cost of adjusting data resolutions. VS overcomes the former challenge by extending the NVMe interface; this requires the fewest modifications to the system stack and applications. VS addresses the latter challenge by exploiting the idle cycles available in modern SSD controllers.

3. RESULTS

Figure 3 shows the relative end-to-end latency of running a complete workload using workloads with the conventional approximate computing approach with GPU-accelerated kernels as the baseline. Since VS efficiently prepares input datasets in storage devices for approximate computing kernels running on the GPU, the manual programmer-directed VS leads to a speedup of 1.52X for these applications. VS also achieved an average energy savings of 32% for applications compared to the conventional approximate computing.

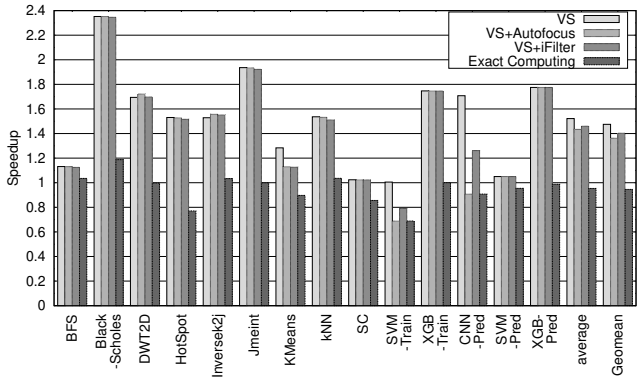

Figure 3: The speedup of the end-to-end latency.

Using Autofocus to dynamically select the desired data resolutions, these applications achieve an average speedup of 1.43X. Without any programmer intervention, iFilter can improve performance by 1.46X. In contrast, the exact computing is 7% slower than the conventional approximate computing.

Several groups of our workloads shared the same datasets, but applied different operators and resolutions to accommodate each individual demand. Without an architecture like VS, the storage system must store multiple versions of a shared dataset or provide raw data to the host for preprocessing, hurting either space-efficiency or performance.

We also compare our mechanisms with other alternatives. Autofocus outperforms a state-of-the-art quality control mechanism by up to 2.86X. VS also outperforms the best compression algorithm for each dataset by 4.40X since VS incurs zero overhead in decoding data on the host side.

4. REFERENCES

[1] NVIDIA Corporation, "NVIDIA T4 TENSOR CORE GPU." https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-datasheet-951643.pdf, 2019.

[2] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke, "Sage: Self-tuning approximation for graphics engines," in *Microarchitecture (MICRO), 2013 46th Annual IEEE/ACM International Symposium on*, pp. 13–24, IEEE, 2013.

[3] M. Samadi, D. A. Jamshidi, J. Lee, and S. Mahlke, "Paraprox: Pattern-based Approximation for Data Parallel Applications," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, (New York, NY, USA), pp. 35–50, ACM, 2014.

[4] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke, "Rumba: An online quality management system for approximate computing," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pp. 554–566, June 2015.

[5] M. A. Laurenzano, P. Hill, M. Samadi, S. Mahlke, J. Mars, and L. Tang, "Input responsiveness: Using canary inputs to dynamically steer approximation," in *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '16, (New York, NY, USA), pp. 161–176, ACM, 2016.

[6] I. Lazaridis and S. Mehrotra, "Approximate selection queries over imprecise data," in *Proceedings. 20th International Conference on Data Engineering*, pp. 140–151, April 2004.

[7] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, "Dynamic knobs for responsive power-aware computing," in *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pp. 199–212, ACM, 2011.