Unleash The Performance of Emerging Storage via Reconfigurable Drive Controller

We have entered a golden age of storage devices for ten years. As shown in Figure 1, the performance of high-end storage drives are increasing at an exponential speed, and the ten-year overall improvement rate is up to 130X. However, meanwhile the performance of host/drive interconnection barely increases. This mismatch sets up the "data movement wall" which prevents the end-user from utilizing the performance of emerging storage devices, therefore making the advanced storage technology to be futile.

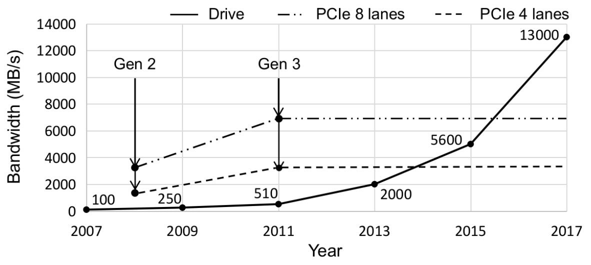

Figure 1: The bandwidth evolution of the storage drive and the host/drive interconnection (in 4-lane PCIe and 8-lane PCIe).

To tackle this challenge, one line of research offloads the data processing task partially to the ARM-based embedded drive controller [2]. Another line of research brings ASIC (Application-Specific Integrated Circuit) into storage drive to enable higher-performing in-storage processing (ISP [1]. By pushing the first-stage processing like reduction or filtering directly into drive, the volume of data for following processing stages (which happen at host CPU) is reduced, therefore alleviating the bottleneck of host/drive interconnection. While current works address the issue of "data movement wall" in some extent, they all more or less fall into following shortcomings, which greatly limits their application scenario.

A) Limited computing capability or limited generality. Drive-embedded CPUs are usually wimpy ARM cores which can be up to 100X slower compared to the host-side CPU. This greatly limits the speedup of ISP system — only 1%∼15% speedup is observed in Summarizer [2]. On the other hand, work like Biscuit [1] integrates ASIC designed for specific workloads, therefore providing satisfactory computing performance. However, due to the architectural limitation, this solution cannot be applied to the general processing scenario.

B) Collision with the firmware execution. Drive-embedded CPUs are originally used for executing the firmware code. In order to avoid the negative performance effects, developers should carefully schedule the CPU resource between the firmware code and user-defined tasks. This puts forward requirements of the application characteristic, therefore limits the application scenario.

C) No/limited isolation and protection. Naturally, the drive is shared among multiple processes, which implies the scenario of the concurrent execution of multiple ISP applications. However, for the sake of performance and energy efficiency, the embedded cores are only equipped with the proprietary SSD firmware without an OS; thus the burden of conducting isolation and protection is left to users.

D) Ad-hoc programming model and interface. Existing works introduce non-standard programming models and interface for ISPs which leans the burden of integration to users.

To address these issues, we build INSIDER, a reconfigurable drive system for ISP. We highlight our design principles as followings:

1) FPGA-based reconfigurable drive controller. We introduce Field-Programmable Gate Array into INSIDER drive as the ISP unit, which is able to provide performance close to a customized hardware while retaining a software-like programmability.

2) Separating the data plane with the control plane. INSIDER removes in-storage data processing from the path of the firmware execution. Control plane consists of drive firmware and host-side INSIDER library. With the coordination of them, we enforce the file permission check, and issue accessing requests to storage chips based on the file location. The ISP unit is unable to access storage chips directly, which prevents it from manipulating the unauthorized data. Instead, ISP unit serves as a middleware on the data path between storage chips and the drive PCIe module, performing preliminary processing on data before it is sent back to the host.

3) Integrating with POSIX Interface. INSIDER abstracts the ISP as the *virtual file* operation using the standard POSIX interface. INSIDER provides API for users to pre-register the virtual file by providing the path of the *real file* and the *accelerating kernel*. For any POSIX operation performed upon the virtual file, the R/W data will be captured and processed by the accelerating kernel, and finally be performed upon the real file. The integration with the POSIX interface provides a seamless way to embed the ISP into the traditional code, alleviating the burden of user-side code refactoring.

4) Architectural Support for Simultaneous Multiprocessing. By leveraging the FPGA partial reconfiguration technique, INSIDER partitions the reconfigurable fabrics into one static region and multiple dynamic regions. The dynamic regions, which are programmable, are used to accommodate the accelerating kernels from users. Whereas the static region, which cannot be programmed by users, is designed to contain the INSIDER hardware library. It is responsible to enforce the data isolation and the credit-based bandwidth scheduling policy among multiple accelerating kernels.

In the performance experiment of six data analytics workloads, INSIDER achieves 8X-11X throughput compared with the traditional host system, 3X-600X throughput and 1.6X-320X cost efficiency compared with the fully-offloaded single ARM-core based ISP system.

References

[1] GU, B., ET AL. Biscuit: A Framework for Near-data Processing of Big Data Workloads. In *Proceedings of the 43rd International Symposium on Computer Architecture*.

[2] KOO, G., ET AL. Summarizer: Trading Communication with Computing Near Storage. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*.