CoDEN: A Hardware/Software CoDesign Emulation Platform for SSD-Accelerated Near Data Processing

Jie Zhang[1], Damian Szmulewicz[2], Erick Macias[2] and Myoungsoo Jung[1]
Computer Architecture and Memory Systems Laboratory,
[1] *The University of Texas at Dallas,*
[2] *Texas Instruments*
jie.zhang6@utdallas.edu, d-szmulewicz@ti.com, emacias@ti.com, jung@utdallas.edu

Abstract

For the past few decades, solid state disks (SSDs) significantly revamped their internal system architecture by employing more compute resources, multiple data channels, and tens or hundreds of non-volatile memory (NVM) packages. These ample internal resources in turn enable modern SSDs to accelerate near data processing. While the prior simulation-based work uncovered potential benefits of offloading the computation from a host to the SSDs, their analytical models make several assumptions that ignore not only detailed system parameters ranging from different micro-architectures through varying number of cores to deep memory hierarchy, but also a wide spectrum of device parameters such as emerging NVM technologies in SSDs. In this work, we propose a novel hardware/software co-design emulation platform, which not only offers flexible/scalable design space that can employ a broad range of SSD controller and firmware policies, but also capture the details of entire software/hardware stacks for SSD-accelerated near data processing. Our comprehensive evaluation results show that the near-data processing bandwidth of twenty different kernel functions we implemented can be as high as 2 GB/sec.

1 Introduction

Modern computing environments have more data to process than they have ever had before, and the volume of such data is expected to be immensely growing. This data explosion renders the systems difficult to process data on the existing memory hierarchy. From a storage system perspective, the root cause of this pitfall is that most data processing applications are disconnected from the underlying data warehouse systems. Specifically, a traditional CPU needs to pull a large amount of data from the storage systems (over a network or a thin-interface connection) and construct them as a recognizable data structure on their working memories [5]. After processing the data, the CPU needs to store/update results/interim-outputs back to the data storage media.

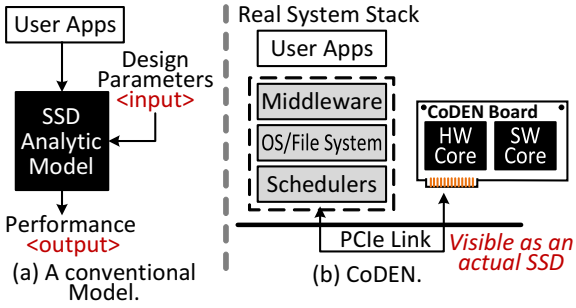


Figure 1: An illustrative comparison between conventional analytical models and our evaluation platform for near data processing.

This pull-and-push big-data processing model wastes tremendous network throughput and introduces poor system power/energy consumption behaviors. One promising solution to address these overheads is to offload the actual compute kernel functions from the host to the underlying solid state disks (SSDs), so that they internally process data without occupying external networks.

Over the last decade, SSDs significantly revamped their internal system architecture in order to improve the overall system bandwidth as well as increase their storage capacity. Specifically, modern SSDs employ tens or hundreds of non-volatile memory (NVM) packages over the multiple channels/buses, which in turn allow them to take advantage of massive device-level parallelisms. In parallel, SSDs equip multiple-core processors to efficiently parallelize data accesses across their many internal NVM resources. Thanks to these numerous architectural improvements, many prior studies are able to integrate the active storage concept [7] to modern SSDs, which can offload the host-side compute tasks to the storage devices. For example, [11] proposed a performance and energy model of active computation on the SSDs, which is used for reducing the data movement costs in scientific applications. Similarly, [2][3] introduced an intelligent SSD approach that performs the MapReduce

framework to accelerate big-data analysis.

Even though the prior simulation-based work studied the potential benefits of unloading the host-side computation to SSDs, their analytical models are unfortunately limited to explore a full design space of future near data processing. As shown in Figure 1a, their simulation-based studies rely on analytical models that make several assumptions. These ignore not only detailed system parameters ranging from different micro-architectures through varying number of cores to deep memory hierarchy, but also a wide spectrum of device parameters, such as diverse emerging NVM technologies in SSDs. In addition, they overly simplified the storage-side controller logic and firmware strategies, whereas the future SSD-accelerated near data processing applications require exploring a broad range of software stack configurations. Specifically, investigating an optimal software design for SSD-accelerated near data processing is unfortunately non-trivial due to two reasons: Firstly, SSD software/firmware modules are tightly coupled to the underlying hardware logic, which can make system designers difficult to explore the potential of near data processing by considering a variety of system-level and hardware-level design parameters. Secondly, their software stack and the corresponding components are not publicly available as commercial SSD vendors need to protect their intellectual property.

To address these limits behind the conventional simulation-based research approaches, we propose *CoDEN*, a novel hardware/software **CoD**esign **E**mulation platform for SSD-accelerated **N**ear data processing. This emulation platform is not only capable of offering flexible/scalable design space that can employ a broad range of SSD controller and firmware policies, but also examining entire software/hardware stacks from a holistic viewpoint. As shown in Figure 1b, our CoDEN can be connected to a host through *PCI Express (PCIe)* interface, a high performance memory bus, and recognized by the host as a "real SSD storage device". The host system can offload specific compute functions defined by users to our CoDEN via PCIe protocol, and those compute kernel functions are executed by one or more fully customized embedded digital signal processing (DSP) cores. Internally, the data accesses are captured by our SSD emulation engine, and the corresponding execution latency values are brought by an actual NVM device and/or NVM latency generator at runtime. In this work, to evaluate our CoDEN platform, we implement 20 kernel functions that can process data whose size varying from 32KB to 4GB. Our comprehensive evaluation results show that the data processing bandwidth of such kernel functions can be as high as 2 GB/sec.

Our **contributions** can be summarized as follows:

• *Emulating endpoint SSDs for near-data processing.* While the existing analytical models may mimic the performance behavior of real products, they cannot capture the detailed latency and energy values consumed by host-side multiple software components. This can prevent architects and system designers from exploring the full design space that the future SSD-accelerated near data processing applications require. In contrast, our hardware/software codesign platform can emulate diverse PCIe endpoint SSD device technologies, as well as data processing inside SSDs. Our CoDEN allows the storage research communities to study a broad range of cross-layer optimizations and to tailor both device-side and host-side software stacks, being aware of different sets of SSD and accelerator technologies.

• *Designing multi-core based data process and I/O acceleration.* In practice, it is non-trivial to identify the optimal number of execution units and accelerators in diverse near data processing applications. In addition, while devising internal communication methods and efficient architectures that exhibit low overheads are essential for future near data processing, the traditional analytical models are unfortunately limited to perform SSD acceleration studies as they only use overly simplified flash performance parameters. In this work, our CoDEN provides scalable computing resources (up to eight customized embedded processors), which are used for near data processing acceleration as well as SSD firmware and controller emulation. Our multi-core based emulation platform is a highly-reconfigurable and can be a fidelity research vehicle to meet a broad range of demands required by the future near data processing applications.

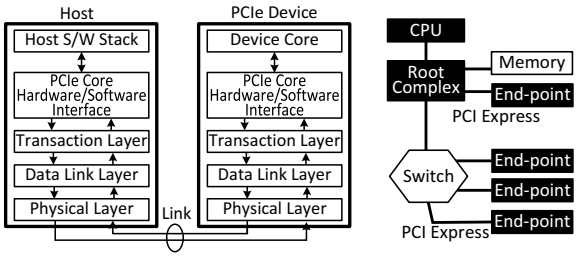
• *Providing flexible controller/firmware prototyping environments.* To hide the complexity of the underlying NVM media, many prior studies heavily performed simulations in designing advanced NVM controllers or firmware. However, as SSD vendors neither reveal nor share their intellectual properties on SSDs, there are very few publicly available resources for the research communities to exploit. This unfortunately makes research on SSD-accelerated near-data processing severely restricted. To address these, our CoDEN offers a very flexible design environment that allows the architects and system designers to prototype their firmware and controller strategies into the real storage stack by acknowledging the numerous computational resources and storage design parameters.

2 Background

In this section, we briefly describe the details of PCIe interface and discuss the SSD internal architecture and data processing therein.

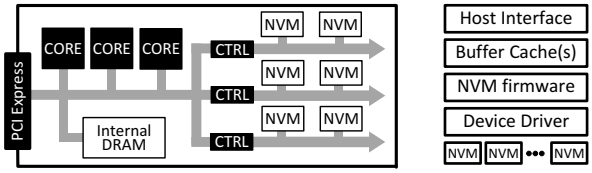
2.1 PCI Express

Figures 2a and 2b pictorially show the PCIe topology and its layered-protocol architecture, respectively. In practice, the root complex connects the CPU and the PCIe fabric/network that consists of one or more switches, each employing multiple endpoint devices. The endpoint is the bottom of such PCIe tree network; it can also be directly connected to the root complex without assistance



(a) PCIe layered architecture. (b) PCIe topology.

Figure 2: PCI architecture.



(a) SSD (b) SSD s/w stack

Figure 3: SSD's internal hardware and software architecture.

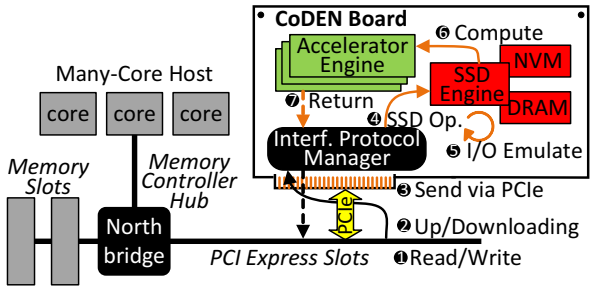


Figure 4: A high level view of our CoDEN platform.

of the switches. In addition, the PCIe interface employs a *packet-based* communication protocol, which can establish a transaction channel between the PCIe endpoints and/or the root complex using multiple packet transfers. Unlike other memory bus technologies, unexpected and unrecognizable data transfers are appropriately handled as errors; entire packets in the transaction can be protected by a 16-bit or 32-bit in-band cyclic redundancy check (CRC) code which is defined in the packet transaction protocol. Since this protocol fully utilizes a packet-based communication, PCIe employs a layered architecture that is commonly used in modern network stacks. As shown in Figure 2a, the physical layer (PHY) converts memory requests or completion commands from the target endpoint device or the root complex to a valid transaction. The data link layer (DLL) handles integrity of PCIe links by managing a sequence of packets, including acknowledgement and flow control, whereas the transaction layer routes the packets in the PCIe network.

As the bandwidth of modern SSDs exceeds the capability of thin storage interfaces (such as SATA), most high performance SSDs employ PCIe as their interface and are connected to a host (or CPU) as an endpoint device. The packetized protocol of the PCIe interface is capable of handling I/O requests as well as other types of vendor specific commands.

2.2 Data Processing in SSDs

For the past decade, SSDs have had significant architectural changes by employing more NVM devices, data buses and relevant computing resources such as multi-core processors. For example, Samsung SSD 850 PRO 1TB accommodates 3 cores for only address translation and data buffering with 1 GB internal buffer [8]. Figures 3a and 3b show an overview of the modern SSD architecture and the SSD software stack therein, respectively. The multiple cores are connected to the data paths that consist of internal DRAM buffers and multiple NVM controllers, each employing one or more NVM chips. Incoming memory accesses can be parallelized across the multiple NVM devices by those cores, and the aggregate performance brought by such parallelism can be exposed to the host via PCIe. On the other hand, as shown in Figure 3b, a storage-side device driver and NVM firmware in the SSD software stack are responsible for handling all memory transactions, whereas the host interface and buffer cache exists to hide long latency imposed by the underlying NVMs.

Leveraging these ample SSD internal resources, there are already commercially-available controllers (e.g., SandForce [9]) that perform simple data processing such as compression or deduplication within an SSD. In addition, many prior studies [7] applied the active storage concept into SSDs by utilizing the SSDs' internal resources and discussed the potential benefits of near-data processing through analytical models. However, such analytical models are limited to explore the full design space of future near-data processing in SSDs, as they only use overly simplified system-level or device-level parameters.

3 Multi-core SSD Emulation Platform for Near Data Processing

Our CoDEN provides a flexible hardware/software co-design environment, which consists of three major components: i) a PCIe based host interface manager, ii) multi-core based DSP accelerators, and iii) SSD software stack and emulation logic. In this section, we discuss the high-level view of our co-design environment and the corresponding CoDEN hardware/software modules.

3.1 Overview

Figure 4 illustrates the high-level view of our CoDEN platform and the emulation procedure in performing near

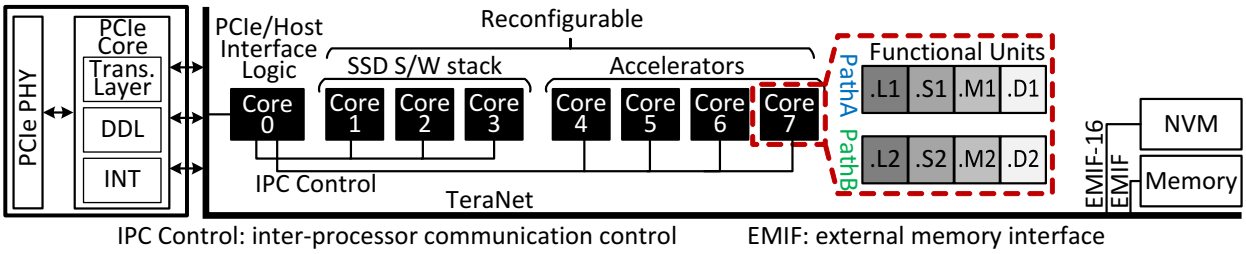


Figure 5: The important architectural components of our CoDEN platform.

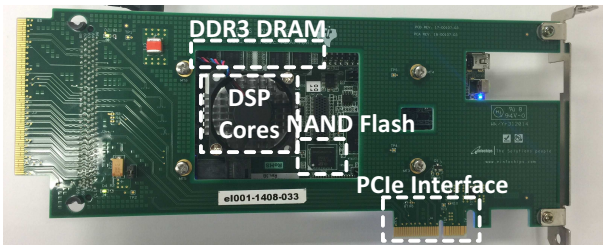
data processing based on a host request. Our platform is connected to the host's memory controller hub (e.g., Northbridge) via *interface protocol manager* that parses the incoming requests including computational kernel execution and SSD reads/writes, and returns the processed values to the host. Based on their request types, it can retrieve target data from the SSD engine, composed by multiple firmware modules and an NVM emulator. During this process, the SSD engine can capture the detailed timing values, degree of parallelism, and overheads imposed by the SSD software stack. Once the target data is loaded, our accelerator engine(s) processes them based on the host-defined compute kernel functions, and only returns the result output to the host through the interface protocol manager, which can remove unnecessary data movements, thereby significantly saving energy.

3.2 Execution Units

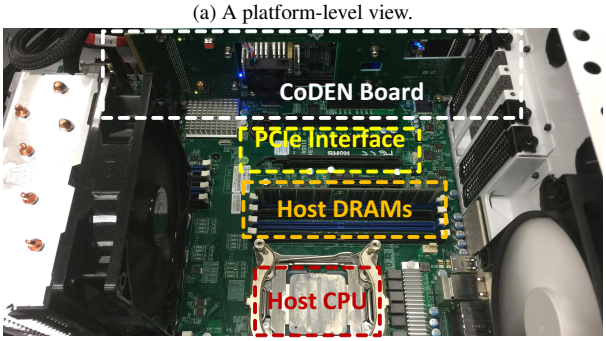
Figure 5 illustrates the design detail of our CoDEN platform. The CoDEN leverages 32-bit KeyStone based multi-core DSP TMS320C6678 from Texas Instruments [10], which is composed by eight 1GHz DSP cores. Specifically, each core of the processor is specialized to four different types of functional units (i.e., .L, .S, .D, .M). As shown in the figure, the .L/.S units can execute most of arithmetic, logical and branch functions similar to a general CPU, and the .D units exhibit high performance on loading and processing data from the internal memory system. In addition, .M units are optimized to perform the multiply-accumulate acceleration that computes the product of two operands and adds the result to an accumulator. In this architecture, the .S/.L functional units are mainly used for the SSD software stack and performing NVM-related work, whereas all other functional units are executed in accelerating near data processing. Since the performance of near data processing can vary based on which workload and access patterns employed, this DSP core assignment can be reconfigured based on the target systems employed. For example, as the default configuration of our on-going project, we allocate a single core for SSD emulation (every software modules in the stack are built in a monolithic fashion), while all remaining DSP cores are assigned for near data processing evaluations; in future, we plan to examine the best configuration to make balance between the SSD software stack and near data processing engine. Unlike other cores, the core-0 is mainly responsible for the inter-process communication (IPC) and handling the host interface logic that we will explain shortly.

3.3 Near Data Processing Emulation

The left most side of Figure 5 illustrates how the host interface (PCIe) is connected to the internal DSP resources. At the very beginning of the board initialization phase, the PCIe bootloader is loaded into the core-0 through an inter-integrated circuit bus (I2C). The bootloader then configures the PCIe *base address registers (BARs)* in order to offer memory access to the host. It then cleans up the PHY and initializes the interrupt controller (INT) that gives control signals to the core-0 in cases where the transaction layer receives a request from the host. Once this link configuration related task is completed, the host's root complex can send computation kernel functions through PCIe (similar to memory mapped I/O). The computational kernel functions are loaded into the CoDEN memory subsystem, and the core-0 distributes the work across the DSP cores, which are assigned for near data processing accelerators. The core-0 also handles communication interrupts (for both legacy interrupt and message signaled interrupt, referred to as *MSI*) on behalf of all SSD modules and accelerator engines. While executing the uploaded compute kernel functions, the accelerators use a message queue to feed the data accesses to the core-0, that forwards them to the SSD engine for each data processing operation. The SSD engine performs memory address translation between logical addresses that the accelerators employ and the physical addresses that the underlying NVM emulation module exposes. The NVM emulation module then generates appropriate latency values for each I/O request (based on address inputs), and the SSD engine holds the execution control as long as the latency values are generated. Once this I/O emulation is complete, the DSP core associated to the SSD engine sends a release message to the target accelerators via the message queue. While the SSD engine can be more complicated in practice by adding up more NVM firmware compo-

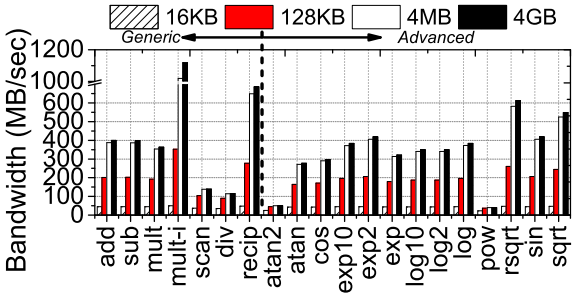


(a) A platform-level view.

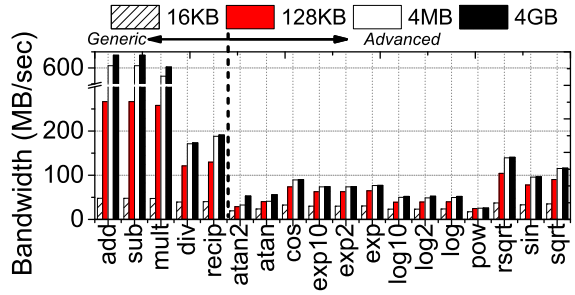


(b) A system-level view.

Figure 6: CoDEN in a real system.



(a) Single precision.



(b) Double precision.

Figure 7: Near data processing bandwidth.

nents such as a buffer cache or scheduler, in this work-in-progress, we implement a simple page-granularity address mapping logic atop of an NVM latency. After the data processing of such kernel functions are complete, the corresponding accelerator sends a signal to the core-0 in oder to return the outcomes back to the host. Finally, the PCIe core composes result packets and sends them to the host.

It should be noted that, all the target data to process reside upon the storage media of our CoDEN platform in this emulation model. Consequently, it is not required for the host to upload the target data to an accelerator. In other words, this near data processing can remove most of unnecessary data movements between the host and accelerators, which in turn can not only improve performance and device-level parallelism, but also save the energy on transferring the data.

4 Evaluation

We implemented twenty computational kernels that can process data whose sizes vary from 16KB to 4GB. Specifically, we made two different sets of kernels: i) *generic* kernels and ii) *advanced* kernels. While the generic functions are commonly used for arithmetic and scanning data, the advanced kernels are related to special math operations such as logarithm, trigonometric, and so on. The descriptions of computational kernels we evaluated are shown in Table 1. For the storage emulation, we use the single-level cell NAND flash memory [6] whose read and write latency values are 25 us and 220 us, respectively. Using our CoDEN prototype (cf. Figure 6), we first evaluate the bandwidth of data processing for kernel functions of single-precision and double-precision data. We also show the performance improvement as the number of cores assigned to accelerator increases.

4.1 Near Data Processing Bandwidth

Figure 7a shows the bandwidth of the near data processing we performed on the CoDEN platform for generic and advanced kernels. In this evaluation, all the data format we tested is single precision. One can see from this figure that the overall bandwidth of the generic functions is 286 MB/sec, on average, whereas that of advanced kernels is 223MB/sec. Among the kernels we implemented, "mult" and "recip" kernels show the best performance, which is as high as 1.2 GB/sec. This is because the .M units of our DSP core include IEEE floating-point multiplication operations that can perform one single-precision multiply each clock cycle. Both the multiply and the reciprocal operations leverage this functionality to speed up the execution of the kernels we tested.

As there are many operations that require double precision for large size matrices in linear algebra, signal processing, and scientific calculations [1], we also evaluate such kernels with the double precision format, and the results are shown in Figure 7b. While the performance of double precision operations is two times worse than the single precision operations, some functions on our CoDEN prototype (e.g,. add, sub, and mult) exhibit data processing bandwidth similar to the single precision operations. This is because of two main contributions. Firstly, the DSP cores we employed integrate double-precision multiply instructions in the DSP instruction

Op.	Param.	Description	Op.	Param.	Description
add	a,b	a + b	exp10	a	10^ a
sub	a,b	a - b	exp2	a	2^ a
mult	a,b	a * b	exp	a	e^ a
multi	a, imd	a * immediate	log10	a	log base 10
scan	a,b	if a == b	log2	a	log base 2
div	a,b	a/b	long	a	log base e
recip	a	1/a	pow	a,b	a^ b
atan2	a,b	arc tan(a/b)	rsqrt	a	1/(a)^ (1/2)
atan	a	arc tan(a)	sin	a	sine(a)
cos	a	cosine(a)	sqrt	a	a^ (1/2)

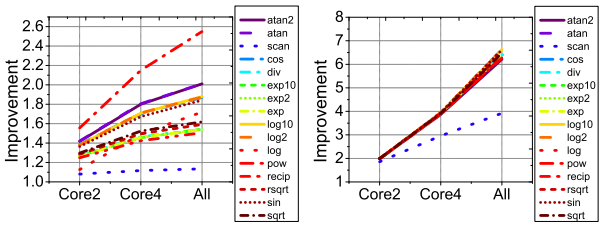
Table 1: The descriptions of the kernel functions we evaluated.

sets, which can perform one double-precision multiply per cycle and also reduce the pipeline length. Secondly, the .L and .S units support 64-bit oprands which lead to efficient parallelization of arithmetic and logical operations.

Note that the common data processing approaches need to load all the data from the underlying storage unit and put them on the target accelerators through an external interface such as PCIe we used here, which require moving data as much as they need to process. Unlike those conventional approaches, by using our CoDEN prototype, the systems do not necessarily require transferring the data back and forth, which can improve performance as well as reduce average energy requirements.

4.2 Scalablity Test

We also performed bandwidth improvement by adding more cores to the accelerator, ranging from 1 to 7 – since we need one core for SSD emulation, at most seven cores can be allocated to near data processing. In this preliminary evaluations, we use OpenMP [4] to parallelize computation kernels on the accelerator engines, and the corresponding results are shown in Figure 8. The bandwidth data are normalized to the performance observed when our CoDEN uses only a single core for near data processing. When the kernels process a small size of data (16KB), as shown in Figure 8a, the performance gain is only about 155% compared to the single core based data processing. Especially, the scan function we implemented has no performance benefit even when we fully allocate the cores to the accelerator (indicated by *all* in the figure). This is because most of the time is spent in SSD acceses and not in data processing; therefore, the performance improvement for this function is negligible. In contrast, in cases where the kernel use big data (4GB), the performance improvement reaches 6.8 times compared to the single core based data processing, and such benefits are scalable as the number of cores increases. This is because the time spent in accessing the SSD is negligible compared to the time required for the execution of the kernels resulting in improved performance due to parallelism.



(a) Small (16KB). (b) Big (4GB).

Figure 8: Performance improvement analysis for varying number of cores.

5 Acknowledgement

We would like to thank the anonymous reviewers for their constructive comments. We also thank to Shuwen Gao who helps us prepare to setup evaluation environment. This research is supported by National Science Foundation CNS-1422923.

6 Conclusion

In this work, we proposed a novel hardware/software co-design emulation platform. Our multi-core based SSD emulation platform is highly-configurable and can be a fidelity research vehicle that meets a broad range of demands required by the future near data processing applications.

References

[1] E. Anderson *et al.*, "Lapack: A portable linear algebra library for high-performance computers," in *SC 1990*.

[2] B. Y. Cho *et al.*, "Xsd: Accelerating mapreduce by harnessing the gpu inside an ssd," 2013.

[3] S. Cho *et al.*, "Active disk meets flash: A case for intelligent ssds," in *SC*, 2013.

[4] L. Dagum and R. Menon, "Openmp: an industry standard api for shared-memory programming," *Computational Science & Engineering, IEEE*, 1998.

[5] M. Jung *et al.*, "Exploring the future of out-of-core computing with compute-local non-volatile memory," in *sc*, 2013.

[6] Micron, "Nand flash memory specification for mt29f4g08aaa, mt29f8g08baa, mt29f8g08daa, mt29f16g08faa," 2006.

[7] E. Riedel *et al.*, "Active storage for large-scale data mining and multimedia applications," in *Proceedings of 24th Conference on Very Large Databases*, 1998.

[8] samsung, "Ssd 850 pro specification," 2014.

[9] Seagate, "Sandforce sf2600 and sf2500 enterprise flash controller datasheet," 2014.

[10] TI, "Multicore fixed and floating-point digital signal processor," 2014. [Online]. Available: http://www.ti.com/lit/ds/symlink/tms320c6678.pdf

[11] D. Tiwari *et al.*, "Active flash: towards energy-efficient, in-situ data analytics on extreme-scale machines." in *FAST*, 2013.