Fast, Energy Efficient Scan inside Flash Memory SSDs

Sungchan Kim
Chonbuk National University, Korea
sungchan.kim@chonbuk.ac.kr

Hyunok Oh
Hanyang University, Korea
hyunok.oh@hanyang.ac.kr

Chanik Park
Samsung Electronics Co., Ltd., Korea
ci.park@samsung.com

Sangyeun Cho
University of Pittsburgh, U.S.A
cho@cs.pitt.edu

Sang-Won Lee
Sungkyunkwan University, Korea
wonlee@ece.skku.ac.kr

ABSTRACT

Today, an SSD (Solid State Drive) is essentially a block device attached to a legacy host interface. As a result, the system I/O bus remains a bottleneck, and the abundant flash memory bandwidth as well as the computing capabilities inside SSD is largely untapped. In this paper, we motivate an efficient in-storage computing approach where (part of) data-intensive processing is moved from the host CPU to inside flash SSDs, close to the data source itself ("in-storage processing"). Especially, we focus on accelerating a key database operation, scan. To realize the idea in a cost-effective manner, we deploy special-purpose computing modules using the System-on-Chip technology. While data from flash memory are transferred, a target database operation is applied to the data stream on the fly without any delay. This reduces the amount of data to transfer to the host drastically, and in turn, ensures all components along the data path in an SSD are utilized in a balanced way. Our experimental results show that in-storage processing outperforms conventional CPU based processing by over 13 times for scan operation. It also turns out that in-storage processing can offer sizable energy savings of up to 7×. This drastic performance improvement is mainly attributable to the parallelism inside single flash SSD, while the similar existing hard disk based approaches have not been such successful because of the limited I/O bandwidth provided by conventional hard disk.

1. INTRODUCTION

Recently, there is a substantial influx of NAND flash based Solid State Drives (SSDs) in the enterprise storage market [1]. For example, large-scale storage appliances like Teradata's Extreme Performance Appliance [2] and Oracle's Exadata [3] harness SSDs to realize very high I/O operations per second (IOPS). Moreover, there are a host of positive forecasts for SSDs in the enterprise market [4].

Most prior developments have focused on the potential of flash SSD as fast and cost-effective hard disk (in terms of IOPS/$) with the legacy block interface. Taking into account the impressive performance and other advantages of flash SSD, this approach is very practical and will remain mainstream for a while. However, we argue that this conventional usage model of SSD fails to fully exploit the performance improvement opportunities offered by continued semiconductor technology advances. Notably, the aggregate raw bandwidth of flash memory devices in SSDs has already exceeded the peak bandwidth of most existing legacy interfaces. Furthermore, future SSD controllers are expected to have high computing power by necessity, as they integrate parallel flash interfaces, large high-speed memory, and more compute cores and logic. Sticking to the legacy block interface implies, unfortunately, that the abundant flash memory bandwidth and the computing capabilities inside SSD will be largely untapped.

Meanwhile, new digital data are generated at astounding rates. Digital information stored in corporations, on the public Internet and on home computers is doubling every month [5]. The size of the web pages indexed by Google was roughly 20 Billion in 2009 and is climbing to 30 to 40 Billion as of March 2011 [6]. The enterprise systems domain is no exception and petabyte databases are realistic (e.g., eBay's 2.4 PB relational data [7]). Large-scale data analysis has become common and will be increasingly important for enterprises. In order to efficiently warehouse and analyze data at such scales, a shared-nothing, massive parallel processing storage tier of many storage servers is common [8].

Unfortunately, the performance of key data-intensive applications is and will be severely limited by the available system bandwidth (through I/O, network, main memory and CPU memory hierarchy) and low data locality [9][10][11]. In large data-intensive applications like TPC-H and Map-Reduce, the dominant computations on data are relatively simple scan and filtering, aggregation, sorting and join; data sets flow from the storage (to network) to all memory hierarchy levels in the host, only to be touched by the CPU briefly. Bandwidth mismatch along the data access path results in performance loss and moving around the massive data consumes sizable energy. This unwarranted inefficiency, of both performance and energy, remains a serious roadblock to database systems to scale out.

In order to significantly improve the efficiency of data-intensive computing, this paper proposes and explores the idea of accelerating database operations for data warehouse workload by moving all or portions of data processing to inside flash SSDs, as close as to the data source itself (i.e., flash memory chips). We refer to this approach with a term "in-storage processing" (ISP in short). To motivate this approach, this paper will focus

specifically on *table scan*, one of the key access methods in any database system.

The contributions of this paper can be summarized as follows. First, using the state-of-the-art CPU, DRAM and flash memory specification, we investigate which hardware component becomes a performance bottleneck when we execute a TPC-H query processing on modern flash SSDs. Second, we explore the design space to obtain appropriate SSD architectures for the scan operation. Third, we formulate and validate models that explain the performance differences between a conventional host CPU based processing, an ISP using CPU embedded inside flash SSD, and an ISP using dedicated SoC hardware logics on flash memory controller. Our results show that both embedded CPU- and hardware-based ISPs outperform host processing by as much as 13× for scan. The ISP approach is promising also in terms of energy efficiency; our results show that measured energy saving is large (as much as 7×) with hardware-based ISP.

In the remainder of this paper, we will first discuss prior related work in Section 2. Section 3 will describe important details of SSD's hardware architecture. Section 4 gives the proposed ISP framework on an intelligent SSD architecture and formulates its performance models. We map a key operation, scan, to the ISP framework. Detailed performance evaluation of the ISP framework is given in Section 5, and finally, Section 6 concludes.

2. RELATED WORK

The idea of database machines in 1970s and 1980s was shown to have the potential of achieving great performance improvements for simple operations (such as "scan") with processor per head, track, or disk [1][12][13]. As summarized in [14], however, database machines failed to be commercially successful for reasons: First, most database machines employed special-purpose hardware (such as associative disks and magnetic bubble memory), but the performance gains were not enough to justify the additional cost and design time. Second, improvements in the host-side processor architecture were much faster than disk bandwidth, leading to the underutilization of the special-purpose hardware. Finally, there was little performance gain for complex operations like join.

In late 1990s, when the disk array became popular and the disk controller was equipped with low-cost general purpose CPU and on-board DRAM, some researchers explored the concept of active or intelligent disks that execute database operators inside the storage controller [15][16]. They aimed at offloading the host CPU with the excess computing power of the embedded CPU in disk drives and reducing the data transfer between the host and the storage. They suggested in-storage processing of primitive operators like scan, selection, aggregation, project, sorting and join [17][18]. However, these approaches didn't take off due to: limited storage interface, overheads of changing database software [16], the lack of large data-intensive killer applications and the absence of vendors who drive changes actively. In fact, while hard disks dominate the storage market, there was no compelling technical and commercial reason to drive changes to the storage interface and the DBMS software stack.

However, we do not believe that the concept of active disk has failed. In fact, it has morphed into different forms. Recently, for example, Teradata's Extreme Performance Appliance [2] and Oracle's Exadata [3] have started to put complex processing into their storage servers. In particular, Exadata can execute SQL projection, restriction and simple join filtering. They seem to have developed new proprietary storage interface between the host machine and the storage servers and changed its DBMS software stack to leverage the processing power inside storage. Consequently we anticipate that the data filtering effect of ISP (as suggested by active disks) will be of increasing significance as the aggregate storage bandwidth grows, clearly evidenced with the introduction of SSDs.

Our work shares the same spirit with these efforts and strives to develop more intelligent storage devices. We focus specifically on flash SSDs. As we will discuss further, we identify that an embedded CPU-based ISP may not be the most desirable in flash SSDs, and in turn, suggest a hardware-assisted ISP.

There is a fundamental difference among our approach, the database machine approach, and the active disk work approach. The concept of the active disk was mainly motivated by the superfluous computing power inside individual disks. In a similar vein, as Boral and DeWitt concluded in their retrospect paper for database machines [14], the limiting factor was the I/O bandwidth of storage media (i.e., disks), not the embedded CPU's processing power or the DRAM bandwidth. Therefore, the existing "CPU-based ISP" with hard disks was I/O bound. In contrast, the proposed ISP with SSD is no longer I/O bound. Instead, the CPU processing power and the DRAM bandwidth inside flash SSD become new bottlenecks. There are two reasons as following.

First, the internal I/O bandwidth of flash SSDs can easily scale by adopting multi-way interleaving over multiple NAND flash chips on a shared I/O bus using multiple chip enable signals [19]. Furthermore, multi-channel interleaving is used with multiple independent flash controllers. The multi- way and multi-channel interleaving of NAND flash memory chips have been deployed as main techniques to significantly improve not only sequential read and write performance but also random read and write performance.

Second, new flash memory chips themselves come with much improved data bandwidth; NAND interface has evolved from 40 Mbps single data rate to 400 Mbps double data rate [20]. Therefore, assuming an SSD with 16 channels where each channel connects to 400 Mbps 8-bit NAND flash memory, the aggregated raw bandwidth amounts to as high as 6.4 GB/s! This raw bandwidth simply surpasses the computing power of contemporary embedded CPUs like ARM9 processor, as well as the bandwidth of DRAM and system bus bandwidth. As a consequence, embedded CPU- based ISP inside a flash SSD would be CPU-bound and data processing cannot proceed at the maximum speed of the storage media bandwidth.

Beside ISP, there have been a plenty of works that are focused on acceleration of database operations with the aid of external special-purpose or commodity hardware. In [21] and [22], the authors presented an FPGA-based approach, where an FPGA is attached to the host interface of a conventional hard disk. In their approach, data from the disk are fed into the FPGA for preprocessing as necessary, offloading computation burdens of the host processor. Commercial realization of this idea is found in [23]. In another approach, acceleration of primitive database operations with the IBM Cell processor has been studied [24][25]. Lastly, the use of general-purpose GPUs to speed up database operations has been examined [9][26][27]. The main ideas of these studies are similar to ours in that the performance of primitive database operations can be improved with the aids of external, non-host computing resources. However, the major

difference is that we carry out the acceleration inside a flash SSD directly. It is the most scalable solution in a cost-effective way to satisfy the huge computing performance requirements of large-scale data processing applications with tens or hundreds of disk drives.

3. BACKGROUND: FLASH SOLID STATE DRIVES

Figure 1 illustrates the general architecture of an SSD with its major components: a host interface controller, a NAND flash memory array, flash memory controllers, an embedded CPU, and a DRAM.

Host Interface Controller: The function of the host interface controller is to support a specific bus interface protocol such as SATA, SAS, and PCI-e. The host interface bandwidth has steadily increased with the introduction of new standards, from P-ATA to SATA in desktop systems and from SCSI to SAS in enterprise applications. The bandwidth of the SATA interface ranges from 3 Gbps to 6 Gbps today. Recently, PCI-e has emerged as a new interface for storage due to its scalable bandwidth and relatively short latency compared to other existing storage interfaces. The PCI-e interface has additional benefits to SSD because host resources can be utilized for memory and computing-intensive parts of flash management such as address mapping and wear leveling.

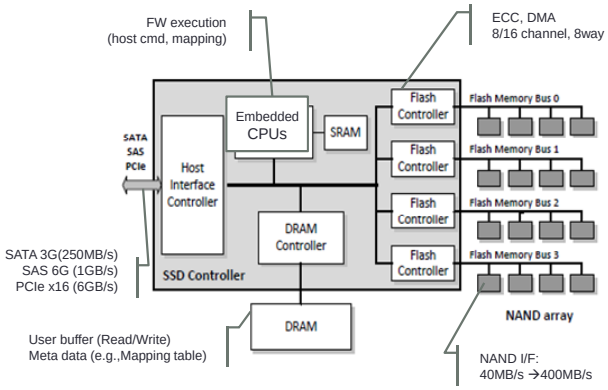

Figure 1: The general architecture of an SSD.

NAND Flash memory: An array of NAND flash memory is used as the permanent storage medium for user data. A flash memory consists of multiple blocks, each of which has multiple pages. A block is a unit of erase operation while a page is associated with read/write operations. NAND flash interface has evolved from 40 Mbps single data rate to 400 Mbps double data rate [28]. Unlike a hard drive, an SSD can achieve scalable performance in proportion to the number of NAND flash memory chips to be integrated.

Flash Memory Controllers (FMC): FMCs are responsible for data transfer between DRAM and NAND flash memory using Direct Memory Access (DMA). They also guarantee data integrity based on Error Correction Code (ECC) like Reed-Solomon and BCH. As NAND flash memory is continuously shrunk, a stronger ECC capability is required. As a result, ECC logic has become a dominant part of an SSD controller chip in terms of cost and power consumption. An FMC also utilizes multi-way interleaving over multiple NAND flash chips on a shared I/O bus using multiple chip enable signals [29]. Moreover, multi-channel interleaving is used with independent channels and FMCs [30]. The multi-channel and multi-way interleaving of NAND flash chips have been deployed as main techniques to improve the performance of both sequential and random accesses. The FMC can be implemented as dedicated hardware for high performance and power efficiency or an application-specific instruction-set processor to support diverse NAND flash memory commands.

Embedded CPU: The embedded CPU(s) together with SRAM provide the execution environment for running flash management firmware called Flash Translation Layer (FTL). FTL parses incoming host commands and translates associated logical block address (LBA) to physical address on NAND flash memory based on a mapping table [31]. Typically, a 32-bit RISC processor is used, which runs at 200 to 400 MHz. Depending on the performance requirement of a given application, multiple CPUs can be incorporated to handle multiple host requests (i.e., NCQ) and NAND flash management simultaneously.

DRAM: DRAM is used to temporarily save user data and FTL metadata. Its size can range from tens to hundreds of MB depending on the target application. Because DRAM is the target of data transfers from both the host interface and FMCs, it is operated at a high clock frequency of 667 MHz or higher, which corresponds to the bandwidth of 2.6 GB/s or more (32-bit bus).

4. IN-STORAGE PROCESSING OF SCAN OPERATION

In this section, we present the details of in-storage processing to perform primitive database operations inside SSDs. We first explain how to carry out ISP in a baseline SSD controller, and derive an analytic model to capture the performance characteristics of ISP according to the variation of architecture parameters. We then propose a cost-efficient solution with special computing modules that are dedicated for computing-intensive functions in place of an embedded CPU in an SSD controller. Finally, we provide the performance model of a conventional in-host processing for the purpose of comparison.

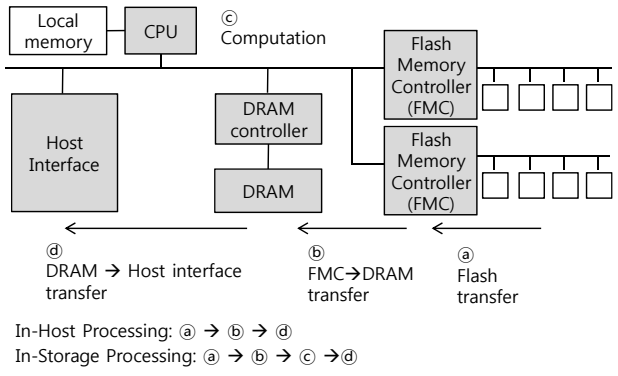

Figure 2: Data path in the baseline architecture.

4.1 Preliminaries

Figure 2 illustrates our baseline architecture, where an embedded CPU[1] is the only computation resource. Therefore any data to be processed should be sent to DRAM, paying the cost of data transfer time. There are two types of DMA transfers in the

[1] The term '*CPU*' indicates the embedded CPU inside an SSD controller. We use the term '*host CPU*' to refer to the CPU of a host machine.

baseline architecture. First one occurs between FMCs and the DRAM (*FMC-DRAM DMA*) while the other is the transfer between the DRAM and the host interface (*DRAM-Host DMA*).

Both DMA operations are always invoked on any data request to an SSD. Thus, we first develop the performance model of the DMA operations that are basic building blocks of the analytic models for both ISP and in-host processing (*IHP*).

FMC-DRAM DMA: We begin by defining some notations to describe an FMC and flash chips. N_{ch} is the number of FMCs in an SSD controller, *i.e.*, the number of channels and N_{way} is the number of ways in a channel. We also define L_{page} to be the size of a single NAND flash page in bytes. Then, let t_{flash_read} be time to read *P* bytes from flash chips to the FMC ignoring cycles to issue an operation command. Similar to the derivation in [32],

$$t_{flash\_read} = \left\lceil \frac{P}{L_{page}} \right\rceil t_{page} + \left\lceil \frac{P}{L_{page} \cdot N_{way}} \right\rceil t_R \quad (1)$$

where t_{page} is the elapsed time to load a page from a flash memory bus after the busy phase for a page read, t_R. These parameters are easily obtained from vendor datasheet books.

We also define t_{DRAM_read} and t_{DRAM_write} as time to read or write *P* bytes from/to the DRAM. When performing FMC-DRAM DMA, data transfers at both flash chips and DRAM are activated in a pipelined fashion to maximize transfer efficiency. As such, if we define $t_{FMC \rightarrow DRAM}$ as time for FMC-to-DRAM transfer of *P* bytes, it is bound to a longer path between them. Therefore,

$$t_{FMC \rightarrow DRAM} = \max\left(t_{flash\_read}, t_{DRAM\_write}\right) \quad (2)$$

For brevity, the details of t_{DRAM_read} that require DRAM architecture specific parameters are omitted, but those parameters are statically determined once the amount of data transfer is given.

DRAM-Host DMA: Next, we focus on time to transfer *P* bytes from DRAM to the host interface and denote it by $t_{DRAM \rightarrow Host}$. Similarly we use $t_{Host \rightarrow DRAM}$ to indicate the transfer in the reverse direction. We also define additional parameters t_{Host_read} and t_{Host_write} as time to read and write *P* bytes through the host interface, which corresponds to the host interface speed. Similar to FMC-DRAM DMA, DRAM-Host DMA is also assumed to work in a pipelined fashion. Therefore,

$$t_{DRAM \rightarrow Host} = \max\left(t_{Host\_write}, t_{DRAM\_read}\right) \quad (3)$$

$$t_{Host \rightarrow DRAM} = \max\left(t_{Host\_read}, t_{DRAM\_write}\right) \quad (4)$$

4.2 Baseline Architecture

In the scan operation, the '*where*' clause of a SQL query is performed on each record, returning only the matching records to the host. ISP of the scan operation on top of the baseline architecture is straightforward as depicted in Figure 2. In the first step, a table to be processed is partially read from FMCs into DRAM using FMC-DRAM DMA (ⓐ and ⓑ in the figure). We assume that the table is evenly distributed over all NAND channels such that the table may be loaded at the maximum bandwidth the flash chip array offers. In the next step (ⓒ), the embedded CPU scans the partial table in the DRAM. At the same time, the matching records of the table are output to another location of the DRAM. If "aggregation" is required, the embedded CPU just updates the aggregated result instead. Finally, the result of the ISP is delivered to the host using DRAM-Host DMA (ⓓ) in response to the query. The procedure above is repeated until the entire table is examined.

In order to derive the performance model of the ISP for scan, suppose that a table has N_{rec} records and the length of a record is fixed to L_{rec} bytes long. We use t_{scan_cpu} to denote the computation time by the CPU to scan the entire table. Then, it is

$$t_{scan\_cpu} = N_{rec} \cdot \left(t_{scan\_comp} + \alpha \cdot t_{scan\_result}\right) \quad (5)$$

where α is scan selectivity, t_{scan_comp} is time for the CPU to execute the scan operation on a single record, and t_{scan_result} is the time to write a matched result, *i.e.*, copy of the record or update of an aggregate value, to DRAM. The lower α is, the smaller the amount of data transfer to the host becomes. For example, α would be almost negligible if we perform Scan-Aggregation since a single value will be sent to the host as a result. Note that t_{scan_comp} and t_{scan_result} depend on the implementation of the ISP. Since the exact modeling of those parameters is non-trivial and beyond the scope of this paper, we resort to measurement results, as will be explained in Section 5.1.

Then, let t_{scan} be the execution time of the scan operation for the table. Since the operation is composed of the three sequential steps, FMC-DRAM DMA, computation by the CPU, and DRAM-Host DMA, it is formulated as follows:

$$t_{scan} = \frac{N_{rec} \cdot L_{rec}}{P \cdot N_{ch}} t_{FMC \rightarrow DRAM} + t_{scan\_cpu} + \frac{\alpha \cdot N_{rec} \cdot L_{rec}}{P} t_{DRAM \rightarrow Host} \quad (6)$$

Note that data delivered to the host depends on the scan selectivity. If we want to reduce $t_{FMC \rightarrow DRAM}$ for better performance, we may widen the peak bandwidth of the flash chips by using more channels or faster flash chips as long as the DRAM bandwidth surpasses that of the flash chip array. Otherwise a faster DRAM is a proper solution, which also reduces $t_{DRAM \rightarrow Host}$. Even though $t_{DRAM \rightarrow Host}$ may also be improved by a faster host interface, we assume the host interface and DRAM speed are fixed so that the variation of $t_{FMC \rightarrow DRAM}$ depends on the configuration of the flash chip array only, and $t_{DRAM \rightarrow Host}$ cannot be altered.

Experimental results show that more than half the latency of the baseline ISP is occupied by the embedded CPU. However, to reduce t_{scan_cpu}, the benefit of using a faster CPU is not a scalable solution as we already discussed in Section 1.

4.3 Hardware Acceleration

To resolve the aforementioned performance bottleneck, we consider dedicated hardware logic for computing as alternative to the CPU as shown in Figure 3. The hardware logic is placed inside an FMC and is composed of a main controller, registers, compare logics and aggregation logics. The main controller is responsible for examining the inbound data stream from the flash memory bus to extract attributes of records to scan and sending them to the compare logics. The registers contain the required information such as matching conditions and values. Whenever any attribute for scan is found from the incoming data stream, a proper filtering condition is applied by the compare logic. At the same time, the predicate of the filtering result is evaluated.

On detecting the end of each record, the accumulated predicate evaluation is used to determine whether the current record is forwarded to the DRAM or not. Also, it triggers the update of the aggregate value if necessary. In this way, there is no need for the

CPU to directly intervene data streams, *i.e.*, zero CPU time. Consequently, the hardware ISP is capable of performing the scan operation on-the-fly without degrading the bandwidth of the inbound data stream. Note that each FMC has its own hardware logic for scan, enabling FMC-wide parallel computing for high processing rates.

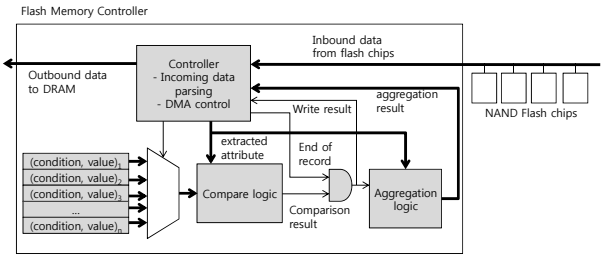

Figure 3: FMC architecture for the hardware ISP.

As a consequence, two terms $t_{FMC \to DRAM}$ and $t_{DRAM \to Host}$ are the main contributors to the execution time of the scan operation with the hardware ISP. The amount of data that are written to the DRAM can be reduced if FMCs discard unmatched records. Hence, $t_{FMC \to DRAM}$ in Equation (2) is replaced with $t^S_{FMC \to DRAM}$ for the hardware ISP, which is:

$$t^S_{FMC \to DRAM} = \max\left(t_{flash\_read}, \alpha \cdot t_{DRAM\_write}\right) \quad (7)$$

Also, t_{scan} with the hardware ISP is modified considering the zero CPU time, then

$$t_{scan} = \frac{N_{rec} \cdot L_{rec}}{P \cdot N_{ch}} t^S_{FMC \to DRAM} + \frac{\alpha \cdot N_{rec} \cdot L_{rec}}{P} t_{DRAM \to Host} \quad (8)$$

For workloads that produce a small value for α, which is the case in our experiment, the performance of the hardware ISP is bound to that of the flash chip array. This means that the proposed ISP is able to maximally exploit the internal bandwidth of an SSD.

4.4 In-Host Processing

The derivation of the performance model for IHP (in-host processing) can be simply formulated. Let us denote by t_{IHP_scan} time to perform the scan operation. Then

$$t_{IHP\_scan} = \frac{N_{rec} \cdot L_{rec}}{P \cdot N_{ch}} t_{FMC \to DRAM} + \frac{N_{rec} \cdot L_{rec}}{P} t_{DRAM \to Host} + N_{rec} \cdot t_{IHP\_scan\_rec\_cpu} \quad (9)$$

where $t_{IHP\_scan\_rec\_cpu}$ is the time for a host CPU to scan a record on average. Note that t_{IHP_scan} is independent of the scan selectivity, meaning all records are transferred to the host.

5. Evaluation

5.1 Setup

In this section, we evaluate the performance of ISP in comparison with IHP for the scan operation. We consider a variety of architectures configured by the parameters in Table 1. For other parameters listed in Table 2, we obtain realistic values from profiling experiments that employ two platforms: a Linux workstation with an Intel Xeon processor (2.26 GHz) and 4 GB main memory (for host CPU time and selectivity) and a commercial simulator of a 200 MHz ARM9 processor [33] (for embedded CPU timings). The purpose of the profiling was to measure the execution times devoted purely to computing of the target database operation itself using different processors.

To measure the CPU execution time, we used Q6 in the TPC-H benchmark [34] as shown in Figure 4. The input table size was chosen carefully to ensure that the table stays in the main memory of the workstation throughout the execution and no unintended disk I/Os occur. To do so, the table was generated with a scale factor of 1.0. We used the same set of conditions in the ARM simulation environment. Using this method, we were able to extract pure CPU times devoted to scanning a record without disk access overhead.

Table 1: Configurable parameters for the performance model

Category	Description or parameter	Value(s)
NAND Flash	N_{ch}	8, 16
	N_{way}	8
	t_R (us)	50
	NAND interface speed (Mbps)	100, 200, 400
	P (bytes)	8192
DRAM	DDR2 clock frequency (MHz)	666, 1333
Host Interface	Bandwidth of host interface (Gbps)	3, 6, 64
Embedded CPU	Clock frequencies of processor/bus (MHz)	200/100

Table 2: Measured numbers for the performance model

Category	Description or parameter	Value
Embedded CPU	t_{comp} (cycles)	24
	t_{result} (cycles)	403
Host CPU	$t_{IHP\_scan\_cpu}$ (us)	0.0142
Selectivity	scan selectivity, α	0.013

```
SELECT
  sum (l_extendedprice * l_discount)
FROM
  lineitem
WHERE
  l_shipdate >= '1994-01-01'
  and l_shipdate < 1995-01-01
  and l_discount < 0.07
  and l_discount > 0.05
  and l_quantity < 24;
```

Figure 4: Query used for profiling.

5.2 Results of Performance Evaluation

5.2.1 Accuracy of the performance model

In the first set of experiments, we validate the accuracy of the proposed performance model of the ISP. To produce reference data, we built a separate, realistic simulation model of the ISP-enabled SSD controller. Then simulation was carried out on a commercial tool, Carbon SoC Designer [35], which is widely used in industry for cycle-accurate simulation of SoC architectures. Note that since the simulation speed is quite slow, it took about 5.6 hours to simulate just one second of the hardware ISP execution. This implies that it is not appropriate to apply this time-consuming simulation to all architecture candidates.

We configured a target architecture with a 200 MHz ARM processor and a 8-channel and 8-way of 100 Mbps flash array. The results of the comparison are shown in Table 3. The estimation error is 6.5% at most, implying that the analytic model accurately predicts the performance of ISP. The performance

prediction of the hardware-ISP is relatively more accurate than that of the baseline-ISP because dedicated hardware blocks behave more deterministically than an embedded CPU.

Table 3: Accuracy of the analytic ISP model of scan operation compared with cycle-accurate simulation

Architecture	Model (cycles)	Simulation (cycles)	Error (%)
Baseline-ISP	297282	317446	6.4 %
Hardware-ISP	16827	16984	0.9 %

5.2.2 Throughput comparison

Next, we investigate the effects of the configuration of NAND flash arrays on the scan throughput to see how ISP and IHP scale according to the variation in the number of flash channels and the speed of NAND flash chips while the number of ways, i.e., number of FMCs, is fixed to 8. For comparison, we consider IHP, baseline ISP, and hardware ISP. As shown in Figure 5, each of them is prefixed by *IHP-*, *cpu-*, and *hw-* and followed by the number of channels. For example, hardware ISP with 8 NAND channels is '*hw-ch8*'. We assume that the DRAM operates at 667 MHz and the host interface is SATA 3 Gbps. The values of the parameters used in the performance model follow Table 2 unless otherwise stated.

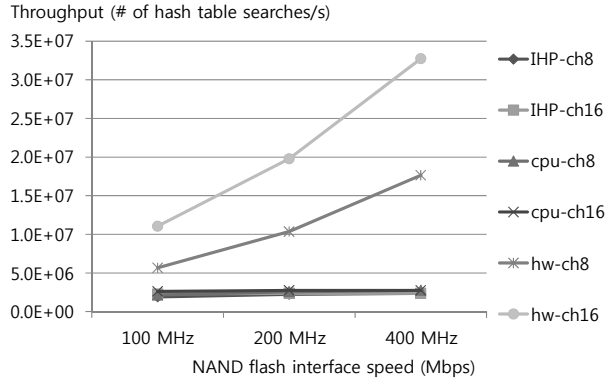

Figure 5: Throughput comparison of ISP and IHP varying the speed of NAND flash interface and the number of channels for scan operation.

The results are depicted in Figure 5. We observe that the throughput of hardware ISP scales linearly with the NAND flash speed while the baseline ISP and IHP remain virtually unchanged. In case of IHP, the host interface appears to be the performance bottleneck. Although IHP needs to transmit the entire table to the host, the bandwidth of the host interface (3 Gbps or 375 MB/s) is lower than that of DRAM (2.66 GB/s). In addition, the sustainable bandwidth for reading the flash array is larger than that of the host interface (3 Gbps) even with the slowest configuration of the flash array, *i.e.*, 100 MHz NAND interface and 8 channels. Even the bandwidth of the next generation interface, SATA with 6 Gbps, is also easily saturated with a 16-channel architecture. This implies that IHP cannot cope with the growth of flash memory bandwidth efficiently while the hardware ISP fully exploits the rich internal bandwidth thanks to the dedicated per-FMC hardware logic. Consequently, the performance gap between the hardware ISP and others grows as the internal bandwidth of the storage becomes higher. In our case, the throughput of the hardware IHP is up to 13.9 times higher over both the baseline ISP and IHP.

On the other hand, the poor performance of the baseline ISP is mainly due to the low computing power of a single embedded CPU. According to the execution time profile of the embedded CPU, it requires about 29 bus cycles to process a 128-byte record on average, yielding a data processing rate of 441 MB/s. However, since inbound data stream from flash memory is at least 743 MB/s, the embedded CPU cannot satisfy the computation requirement. This observation shows that the performance bottleneck of ISP on an SSD is not storage media bandwidth but other components, such as an embedded CPU or host interface, along with the data path unlike hard disk-based database machines [14] where the performance bottleneck is the storage media itself.

5.2.3 Structural breakdown of latency

It is worth decomposing the target operation's execution into distinct parts to provide means of reasoning the performance variations according to the processing methods. For this purpose, latency of the scan operation is divided into four components: (1) transfers between DRAM and host interface, (2) transfers between DRAM and FMC, (3) computation with the embedded CPU, and (4) computation with the host CPU as shown in Figure 6(a).

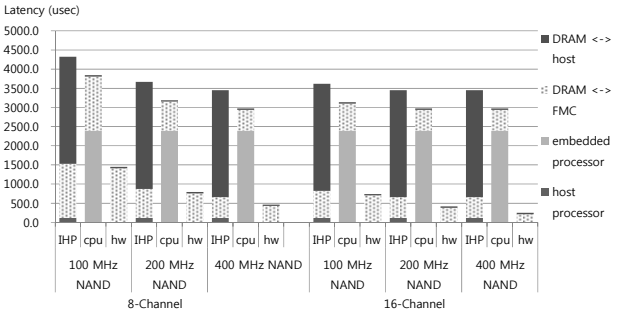

Figure 6: Breakdown of execution times for the proposed ISP and the IHP varying the speed of NAND flash interface and the number of channels for scan.

For IHP, we observe that a major portion in the overall execution time is devoted to data transfer via the host interface. As stated previously, even the slowest flash configuration (8 channels with 100 Mbps flash) offers greater bandwidth than the host interface. Thus, increasing the bandwidth of the flash array does not benefit IHP. In the baseline ISP, a similar observation is found except that the embedded CPU is the new bottleneck (not the host interface).

In the baseline ISP, the entire table should be loaded to the DRAM since the embedded CPU performs the scan operation. Therefore according to the increase in the FMC bandwidth, the DRAM may not be able to accept data from the FMC as the full speed. For example, the sustainable bandwidth of 16 channels with 200 MHz flash chips amounts to 2,776 MB/s while the peak bandwidth of DRAM at 667 MHz is only 2,664 MB/s. Moreover, the actual DRAM bandwidth available to FMC-DRAM DMA may be significantly less than the peak bandwidth because other data transfers, DRAM-HostIF DMA for example, can use DRAM simultaneously. This explains why the time for DRAM-FMC transfer remains the same in the underlying 16 channels with both 200 Mbps and 400 Mbps flash chips. The hardware ISP, however, does not suffer from the aforementioned DRAM bandwidth limitation since FMCs discard filtered data without sending them to DRAM. The more records are filtered out in the FMCs, the more the bandwidth of the flash array is exploited. Also since the

scan selectivity is quite small (0.013), the data transfer through the host interface is small with the ISP.

To summarize, ISP provides the means to exploit the internal bandwidth of flash array maximally. The dedicated hardware logic at each FMC performs on-the-fly computation without hindering the flow of inbound data from the flash array, efficiently offloading the burden of computation and entailed communication from the host. As a result, the performance of the scan operation follows the raw performance of the flash array itself with the hardware ISP.

5.2.4 Effect of selectivity and the host interface

The previous experiments are based on the fixed low selectivity. Therefore, data transfer via the host interface has limited impact on the performance of ISP. One may guess that if the selectivity approaches 1, then the host interface can lower the performance of ISP. To investigate the impact of the selectivity and the host interface speed, we carried out another set of experiments and show the results in Figure 7. We consider three kinds of host interface speeds (3, 6, and 64 Gbps) to represent the existing host interfaces, SATA rev. 2.0 (3 Gbps), SATA rev. 3.0 (6 Gbps) and PCI-e x16 (64 Gbps, 8 GB/s), respectively. To ensure that the bandwidth of the host interface is fully utilized, we set the configuration of DRAM and flash array to the highest performance; we use DRAM of 1,333 MHz and 16 channels with 400 Mbps flash chips.

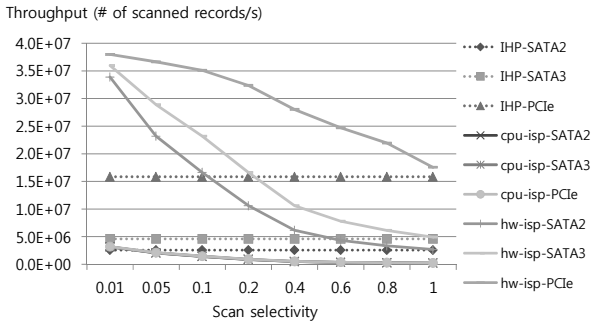

Figure 7: Performance of the ISP according to the variation of the selectivity and the host interface.

Table 4: Ideal performance of architecture components in terms of data processing rate (unit: number of processed records per second)

Component		Data processing rate
Host CPU		7.06×10^7
Embedded CPU		2.34×10^5
Hardware-ISP		3.83×10^7
Host interface	SATA 2.0	2.93×10^6
Host interface	SATA 3.0	5.86×10^6
Host interface	PCI-e	6.25×10^7

The result in Figure 7 shows that the performance of the hardware ISP converges to IHP as the selectivity approaches 1 over all host interface types. The performance gap of IHP and hardware ISP decreases as the DRAM-to-host interface transfer becomes dominant as the selectivity on the hardware ISP grows. To explain this, we provide the ideal data processing rate of each architecture component in Table 4. Note that the data processing rate may differ from the data bandwidth. For scan, computing capability of the host processor amounts to 70.6 M records per second while the hardware ISP operates at 38.3 M records per second. In this situation, the DRAM-to-host interface transfer is a performance bottleneck when the host interface is SATA both in IHP and the hardware ISP. On the other hand, with the PCI-e interface, flash memory bandwidth turns out to be the performance bottleneck. However, the storage media bandwidth of an SSD will scale easily to the host interface speed by growing NAND channels or NAND interface speed. Further since the proposed hardware-ISP can fully exploit the storage media bandwidth in a scalable way, the growth of the host interface speed would not confine the applicability of the ISP.

5.3 Energy Consumption Evaluation

Energy reduction is another key benefit of ISP. In a conventional system, searching for a specific value that is associated with a given key requires that all data be transferred to the host CPU, via various system components including the host interface, main memory (DRAM), and L1/L2 cache memories. The data is finally loaded into a CPU register before being compared with the search key. With a match, the search operation is done. Otherwise, the data is discarded after uselessly spending energy and time. Note that this inefficiency can be amplified in the network-prevalent data center environment because the data should travel through even more system components and cables. On the contrary, the proposed ISP performs the compare operations inside the SSD through simple hardware logics in FMCs. As a result, most data that do not match the key are filtered early at the minimum distance from the storage medium.

Table 5: Comparison of normalized energy consumption.

Processing method	Energy consumption
ISP (modified firmware)	0.142
IHP (conventional)	1.000

To estimate the energy reduction benefit of the ISP approach, we measured the energy consumption of a string search workload on a real platform. The benchmark is executed on a laptop PC with a 2 GHz Intel Centrino Core 2 processor, 1 GB of memory and a Samsung 64 GB SSD. Since an actual implementation of the hardware ISP is not yet available, we modified the SSD firmware to emulate the hardware ISP behavior. Assuming that there are hardware comparison logic modules, we randomly selected data within the SSD and returned them in response to a specific command of the host. As the hardware comparison logic is simple, and incurs negligible latency and power overhead, this emulation method gives us a reasonable estimate of the potential energy gain of ISP. The results are normalized to that of IHP and shown in Table 5. Energy consumption of ISP is just 14% of the IHP scheme.

6. CONCLUSIONS

We presented the idea of "in-storage processing" (ISP) for data-intensive applications on flash-based SSDs. ISP addresses the low utilization of available data bandwidth and computation power in SSDs and opens up new exciting opportunities to increase the performance and energy efficiency of data-intensive workloads. The main idea of ISP is to move data-intensive processing to inside flash SSDs, close to the data source (flash memory chips) and to send the (reduced) results of the processing to the host. This allows us to fully exploit the anticipated high raw flash memory bandwidth and to reduce the amount of upward data transfer through the host interface. The special-purpose computing module deployed in the SSD controller is a key enabler of

practical ISP. We showed in this paper that the hardware-based ISP approach realizes significant performance improvement for the key database operation, scan, compared with the conventional host processing approach. Moreover, the hardware-based ISP consumes much less energy at negligible cost overheads.

As large data-intensive applications are popular and their demands for data processing grow exponentially, the current computing paradigm of *bringing data to host CPU* for computation will encounter the unprecedented "bandwidth crisis" along the path from storage, network, DRAM to CPU. Unfortunately, the contemporary solution with the simple map-reduce programing paradigm on massively large numbers of commodity PC clusters would be also sub-optimal because it also brings data to the host CPU. A more fundamental solution is to *bring the computation close to data itself*, and thus to remove the potential bandwidth bottleneck. Fortunately, with the advent of the bandwidth breakthrough in flash memory and the intrinsic parallelism inside an SSD, it is the right time to revisit the concept of database machines and active disks with the cost-effective SoC technology. As we demonstrated in this paper, ISP can be a very promising scale-out solution for the next generation data-intensive computing paradigm in terms of performance, cost and power.

7. ACKNOWLEDGMENTS

This investigation was financially supported by Semiconductor Industry Collaborative Project between Hanyang University and Samsung Electronics Co. Ltd. This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2010-0005982). This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. 2010-0025649 and No. 2010-0026511).

8. REFERENCES

[1] Intel Corporation, Solid-State Drives in the Enterprise: A Proof of Concept, Mar. 2009.

[2] Teradata Corporation, Teradata Extreme Performance Alliance. Product Site. http://www.teradata.com/t/extreme-performance-appliance.

[3] Oracle Corporation. Oracle Exadata White Paper. Jul. 2010.

[4] Objective Analysis, Solid State Drives in the Enterprise, 2008.

[5] B. Head. Data doubles daily a decade hence, ITWire, Mar. 2011. http://www.itwire.com/storage/45827-data-doubles-daily-a-decade-hence.

[6] http://www.worldwidewebsize.com/.

[7] A. Pavlo, E. Paulson, A. Rasin, D. Abadi, D. DeWitt, S. Madden, and M. Stonebraker. A Comparison of Approaches to Large-Scale Data Analysis, SIGMOD, 2009.

[8] Hung-chih Yang, Ali Dasdan, Ruey-Lung Hsiao, and D. Stott Parker, Map-reduce-merge: simplified relational data processing on large clusters, SIGMOD, 2007.

[9] C. Kim, J. Chhugani, N. Satish, E. Sedlar, A. D. Nguyen, T. Kaldewey, V. W. Lee, S. A. Brandt, and P. Dubey. FAST: fast architecture sensitive tree search on modern CPUs and GPUs, SIGMOD, 2010.

[10] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters, OSDI, 2004.

[11] Maya Gokhale. Hardware Technologies for High-Performance Data-Intensive Computing, IEEE Computer vol. 41(4), 2008.

[12] S. Y. W. Su and G. J. Lipovski. CASSM: a cellular system for very large data bases, VLDB, 1975.

[13] E. A. Ozkarahan, S. A Schuster, and K. C. Smith. RAP - associative processor for database management, AFIPS, 1975.

[14] H. Boral and D. J. Dewitt. Database Machines: An Idea whose time has passed?, IWDM, 1983.

[15] E. Riedel, C. Faloutsos, and G. Gibson, Active storage for large-scale data mining and multimedia, VLDB, 1998.

[16] K. Keeton, D. Patterson, and Jm Hellestein. A case for intelligent disks, SiGMOD record, 1998.

[17] E. Riedel, C. Faloutsos, and D. Nagle. Active Disk Architecture for Databases. Technical Report CMU-CS-00-145, Carnegie Mellon University, Apr. 2000.

[18] E. Riedel, C. Faloutsos, G. A. Gibson, D. Nagle. Active Disks for Large-Scale Data Processing. IEEE Computer, vol. 34(6), 2001.

[19] Samsung Electronics Inc., Samsung solid state drive basics. http://www.samsungssd.com/meetssd/overview, 2010.

[20] M. Sivathanu, L. Bairavasundaram, A. Arpaci-Dusseau, and R. Arpaci-Desseau. Database-Aware Semantically-Smart Storage, FAST05.

[21] R. Mueller and J. Teubner. FPGA: what's in it for a database?. SIGMOD, 2009.

[22] R. Mueller and J. Teubner, and Gustavo Alonso, Data Processing on FPGAs, PVLDB, 2009.

[23] Netezza Corporation. The Netezza Data Appliance Architecture: A Platform for High Performance Data Warehousing and Analytics White Paper. 2010.

[24] B. Gedik, R. R. Bordawekar, and P. S. Yu. CellSort: High Performance Sorting on the Cell Processor, VLDB, 2007.

[25] B. Gedik, P. S. Yu, and R. Bordawekar. Executing Stream Joins on the Cell Processor, VLDB, 2007.

[26] N. Govindaraju, J. Gray, R. Kumar, and D. Manocha. GPUTeraSort: High Performance Graphics Co-processor Sorting for Large Database Management, SIGMOD, 2006.

[27] B. He, K. Yang, R. Fang, M. Lu, N. K. Govindaraju, Q. Luo, and P. V. Sander. Relational Joins on Graphics Processors, SIGMOD, 2008.

[28] http://www.samsung.com/global/business/semiconductor/products/flash/Products_Toggle_DDR_NANDFlash.html.

[29] Samsung Electronics Co., K9XXG08UXM flash memory data sheet, 2007.

[30] Y. J. Seong et. al. Hydra: A block-mapped parallel flash memory solid-state disk architecture. IEEE Trans. Computers vol.59(7), 2010.

[31] J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee. A Superblock-Based Flash Translation Layer for NAND Flash Memory. EMSOFT, 2006.

[32] S.-K. Won, S.-H. Ha, and E.-Y. Chung. Fast Performance Analysis of NAND Flash-Based Storage Device. IET Electronics Letters, vol 45(24), 2009.

[33] ARM Ltd., RealView ARMulator. http://www.arm.com/products/DevTools/RealViewDevSuite.html.

[34] Transaction Processing Performance Council. http://www.tpc.org/.

[35] Carbon Design Systems, Inc. SoC Designer Plus, http://carbondesignsystems.com/SocDesignerPlus.aspx.