Information Sciences 327 (2016) 183–200

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

In-storage processing of database scans and joins

Sungchan Kim[a,*], Hyunok Oh[b], Chanik Park[c], Sangyeun Cho[c], Sang-Won Lee[d], Bongki Moon[e]

[a] *Chonbuk National University, Republic of Korea*
[b] *Hanyang University, Republic of Korea*
[c] *Samsung Electronics Co., Ltd., Republic of Korea*
[d] *Sungkyunkwan University, Republic of Korea*
[e] *Seoul National University, Republic of Korea*

ARTICLE INFO

Article history:
Received 15 September 2014
Revised 3 March 2015
Accepted 29 July 2015
Available online 22 August 2015

Keywords:
SSD
Database
Performance
Energy
Scan
Join

ABSTRACT

Flash memory-based SSD is becoming popular because of its outstanding performance compared to conventional magnetic disk drives. Today, SSDs are 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 and the computing capabilities of SSD are largely untapped. In this paper, we propose to accelerate key database operations, scan and join, for large-scale data analysis by moving data-intensive processing from the host CPU to inside flash SSDs ("*in-storage processing*"), close to the data source itself. To realize the idea of in-storage processing in a cost-effective manner, we deploy special-purpose compute 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 processing with a host CPU by over up to 7 ×, 5 × and 47 × for scan, join, and their combination, respectively. It also turns out that in-storage processing can be realized at only 1% of the total SSD cost, while offering sizable energy savings of up to 45 × compared to host processing.

1. Introduction

Recently, there is a substantial influx of NAND flash-based Solid State Drives (SSDs) in the enterprise storage market [18]. For example, large-scale storage appliances like Teradata's Extreme Performance Appliance [42] and Oracle's Exadata [33] 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 [31].

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 observe that this conventional usage model of SSD would

[*] Corresponding author. Tel.: +82-63-270-2411.
E-mail addresses: sungchan.kim@chonbuk.ac.kr (S. Kim), hoh@hanyang.ac.kr (H. Oh), ci.park@samsung.com (C. Park), sangyeun.cho@samsung.com (S. Cho), wonlee@ece.skku.ac.kr (S.-W. Lee), bkmoon@snu.ac.kr (B. Moon).

http://dx.doi.org/10.1016/j.ins.2015.07.056

fail to fully exploit the performance improvement opportunities offered by continued technology advances. Notably, the aggregate raw bandwidth of flash memory devices in SSDs has already exceeded the peak bandwidth of most existing legacy host interfaces. Furthermore, future SSD controllers are expected to have high computing horsepower by necessity, as they integrate parallel flash interfaces, large high-speed memory, and more computing cores and logic. Maintaining the legacy block devices implies, unfortunately, that the abundant flash memory bandwidth and the computing capabilities inside an SSD will be largely untapped.

Meanwhile, new digital data are generated at astounding rates. Digital information in corporations, on the public Internet and on home computers is doubling every month [16].

The enterprise systems domain is no exception (e.g., eBay's 2.4 PB relational data [34]). Large-scale data analysis has become a necessity 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 [48].

Unfortunately, the performance of data-intensive applications would be severely limited by the available system bandwidth (through I/O, network, and CPU memory hierarchy) and low data locality [10,12,24]. In large data-intensive applications like TPC-H and Map-Reduce, the dominant computations on data are scan and filtering, aggregation, sorting and join; data sets flow from the storage (through network) to all memory hierarchy levels in the host, only to be touched by a host 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 processing large data sets, this paper proposes and explores the idea of accelerating database operations for data warehouse workload by moving all or portions of data-intensive processing to inside flash SSDs, close to the data source itself (i.e., flash memory chips). We refer to this approach as "*in-storage processing*" (*ISP* in short) because data processing is performed inside a storage device.

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. The contemporary solution with the simple Map-Reduce programming 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, owing to 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 System-on-Chip (SoC) technology. As we demonstrated in this paper, ISP can be a very promising solution for the next generation data-intensive computing paradigm in terms of performance, cost, and energy.

The idea of intelligent processing and data storage together was examined previously. In the late 1970s and early 1980s, this idea was actively studied in the context of database machines [5]. These systems became out of favor from the late 1980s mainly because their modest performance gain did not justify the high cost of special-purpose hardware [5]. The idea was then revisited and extended in the late 1990s to using an array of commodity active disks that integrate embedded general-purpose processors and memory [23,36].

There is a fundamental difference among our approach, the database machine and the active disk approach. The concept of 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 [5], the limiting factor was the I/O bandwidth of storage media (i.e., disks), not the CPU processing power or the DRAM bandwidth. Therefore, the previous "Disk-based ISP" was I/O bound.

In contrast, the proposed ISP with SSD is no longer I/O bound. Instead, the processing power of embedded CPU and the DRAM bandwidth inside flash SSD become new bottlenecks. Let us explain why this is the case. First, the internal I/O bandwidth of flash SSDs can be easily scaled by adopting multi-way and multi-channel interleaving of NAND flash memory chips [38]. Second, new flash memory devices provide much improved data bandwidth; recent DDR 2.0 NAND interface provides 400 Mbps of pin bandwidth [41]. Therefore, assuming an SSD with 16 channels each of which 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 and DRAM. As a consequence, SSD-based ISP would be CPU-bound rather than I/O bound. Furthermore, the CPU performance is not the only bottleneck in the SSD-based ISP. Along the data path from multiple flash chips to an embedded CPU, other bottlenecks exist such as DRAM bandwidth and DRAM–CPU cache bandwidth.

Instead of exhaustively exploring the design space of the CPU-based ISP, in this paper, we turn our attention to leveraging the intrinsic data parallelism of flash SSD and deploying cost-effective data computing modules brought close to each flash chip using the System-on-Chip (SoC) technology. This "*hardware-accelerated ISP*" approach is analogos to the process-per-track and process-per-head architectures of database machines [5]. We show that the two common operators, *scan* (or *selection*) and *join* in SQL queries can be easily and efficiently implemented. In particular, each flash channel can be augmented with hardware logic that carries out a simple selection operation. While data from flash memory are transferred, the selection operation can be applied to the data stream on the fly without any delay. As a result, only the selected records will be placed in the SSD controller's DRAM. This selection operation is performed in parallel (in all flash channels) and the amount of data to be transferred to the DRAM (and ultimately to the host) can be drastically reduced, depending on the selectivity of the given filtering predicate. Then, every component along the data path within SSD can work in a balanced way as possible.

Another benefit of our hardware-accelerated ISP[1] using the SoC technology is to accelerate the hash join. In fact, it is well known that one hashing of a value requires several tens CPU cycles and the random access patterns inherent in the build and probe phase of hash join have little temporal or spatial locality, thus suffering excessive data stalls [24] and high CPU cycles per record. However, the hardware-based implementation with moderately sized content addressable memory (CAM) can achieve one cycle per hash function in both build and probe phase in hash join and can also achieve one cycle per record matching in probe phase. Consequently, we can drastically reduce the execution time of hash join.

It is noteworthy that Boral and DeWitt criticized that the concept of highly parallelizable database machine was predicated on the availability of mass storage technologies, not commercially viable at that time and later [5]. This criticism has remained valid until very recently, when flash SSDs emerged as commercial product. We believe it is the right time to revisit the database machine concept, considering the fast evolving flash memory technology.

Another compelling reason to put more intelligence into the flash SSD is that we can explore many SSD-aware query processing optimizations. For example, by controlling the location of temporary data during a join operation, we can mitigate the limitation of erase-before-overwrite and read/write speed asymmetry. Also, we can control the location of each partition from a hash join so that the probing phase of different partitions can be performed in parallel. If the query processing is carried out in the host side, such fine optimization is impossible with the limited interface between the host and the storage [6]. These optimizations are neither necessary nor possible with previous ISP approaches.

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 where time goes and which hardware component becomes a performance bottleneck when we execute TPC-H selection and join queries. Second, we explore the design space for putting the selection and join operators in the form of SoC along the data path from flash memory chips to embedded CPU. Third, we formulate and validate models that explain the performance differences between IHP, CPU-ISP, and HW-ISP. Our experimental results show that in-storage processing outperforms conventional processing with a host CPU by up to 7 ×, 5 ×, and 47 × for scan, join, and their combination, respectively. It also turns out that in-storage processing can be realized at only 1% of the total SSD cost, while offering sizable energy savings of up to 45 × compared to host processing.

In the remainder of this paper, we will first discuss prior related work in Section 2. Section 3 will describe important details of SSDs hardware architecture. Section 4 gives the proposed ISP framework on an intelligent SSD architecture and formulates its performance models. We map two key operations, scan and join, to the ISP framework. Section 5 describes experimental setups. The quantitative results of the ISP framework are given in Section 6, and finally, Section 7 concludes.

2. Related work

In the late 1970s and early 1980s, the idea of executing database operations in storage devices was extensively explored in the context of database machines [5]. The idea was shown to have the potential of achieving great performance improvements for simple operations (such as “scan”) with processor per head, track, or disk. As summarized in [5], however, it has failed because of several difficulties. First, most database machines used 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 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.

With the prevalence of SSDs having the aforementioned nice characteristics, the idea of in-storage processing has been studied very actively for recent several years. To our best knowledge, our previous work [25] is the first attempt to evaluate the performance and energy benefits of in-storage processing. In order to exploit the high internal bandwidth of SSDs for scan operations in database applications, we proposed to put dedicated hardware logic for each of flash channels so that filtering of unmatched records can be performed at the rate of the internal bandwidth of an SSD for given scan conditions as “*WHERE*” clauses in SQL queries. Our work demonstrated significant performance gain and energy saving of in-storage processing. The idea of putting hardware accelerator has also been exploited for data mining applications [4].

Because a realization of HW-ISP is not available so far, the evaluation of actual ISP implementation has been performed on existing SSD platforms, i.e., CPU-ISP, that are designed as normal storage devices. In [43,44], the benefit of ISP is examined in the context of scientific simulation. The authors in [26] proposed to exploit ISP for sorting. They implemented CPU-ISP on so-called the OpenSSD platform [32] that has a single embedded processor running at 75 MHz. As a result, performance benefits of the ISP are restricted owing to the relatively low computing capability of the embedded processor. On the other hand, several researches have been conducted on a recent SSD platform that has triple processors running at 200 MHz and higher internal bandwidth than the OpenSSD platform [11,20,21]. Among them, the work done in [11] evaluated the prototype of CPU-ISP-assisted scan operation that is integrated into an actual DBMS, which turned out to be 2 × faster than the conventional host CPU-based processing. Another work focused on the possibility of applying CPU-ISP to the Hadoop Map-Reduce framework, showing the energy saving benefits of the ISP [20]. Performing ISP requires the extension of the communication interface between a host and a storage device, which has started to be studied recently. In the approach proposed in [21], a host can reconfigure the parameters of Flash Translation Layer (FTL) in an SSD through the host interface, which provides adaptivity to dynamic disk access patterns, thus providing application-specific storage customization.

[1] In the remaining of the paper, we denote the CPU-based ISP by *CPU-ISP*, hardware-accelerated ISP by *HW-ISP*, and the conventional host CPU-based processing by *IHP*.

More aggressive innovation of SSD controller architecture includes the deployment of reconfigurable stream processors [9] and GPGPUs (General Purpose Graphics Processing Units) [8] inside the controller aiming at leveraging data-level parallelism. Both approaches proposed to use Map-Reduce as a programming model inside the controller. The stream processor provides programmability by configuring its datapath connecting an array of ALUs in response to required computational kernel. On the contrary, the GPU executes a CUDA program. In these approaches, however, the impacts on database applications were not evaluated.

Besides ISP, there have been a plenty of work that focuses on accelerating database operation with the aid of external special-purpose or commodity hardware. Recently, for example, Teradatas Extreme Performance Appliance [42] and Oracles Exadata [33] have started to put complex processing into their storage servers. In particular, Exadata can execute SQL projection, restriction, and simple join filtering. In [28,29,46,47], 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 [30]. This work differs from ours in that data processing occurs outside the storage device boundary. This approach is non-intrusive and makes sense when the storage and the host interface are well balanced in terms of bandwidth. However, the data processing rate of this approach is bound to the host interface bandwidth. Also, if a target database operation produces temporary data to be written back to the storage, it causes additional disk I/Os. Therefore, its application is restricted to stream-type data processing such as scan, sorting, aggregation, and so on.

GPUs have gained a lot of attention recently as an alternative to a host processor for computation- and memory-intensive applications, thanks to its massive hardware parallelism with up to thousands of light-weight processor cores. GPU acceleration of two primitive operations, scan and sorting, was proposed in [13,15].

To summarize, the main ideas of these researches are similar to ours in that the performance of primitive database operations can be improved with the aids of external, non-host computing resource. However, we carry out the acceleration inside a flash SSD directly, providing the most scalable solution in a cost-effective way to satisfy the huge computing performance requirements of large-scale data processing applications.

3. Background: flash Solid State Drives

Fig. 1 illustrates the general architecture of an SSD with its major components: host interface controller, an array of NAND flash memory, flash memory controllers, embedded CPU, and DRAM. We will first describe the main components in some detail and then, discuss the anticipated future trends to put our proposed approach in perspective.

3.1. SSD components

Host interface controller: The function of the host 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 300 MB/s to 600 MB/s today. Recently, PCI-e has emerged as a new interface for storage devices due to its scalable bandwidth and relatively short latency compared to other existing storage interfaces. In addition, the PCI-e protocol allows the attached devices to communicate with each other through packet exchange. It implies that an embedded CPU inside an SSD with the PCI-e interface may request another PCI-e device of higher compute capability, e.g. GPU, to offload heavy computations such as address mapping and wear-leveling in flash memory management, or may use memory of the PCI-e devices if necessary.

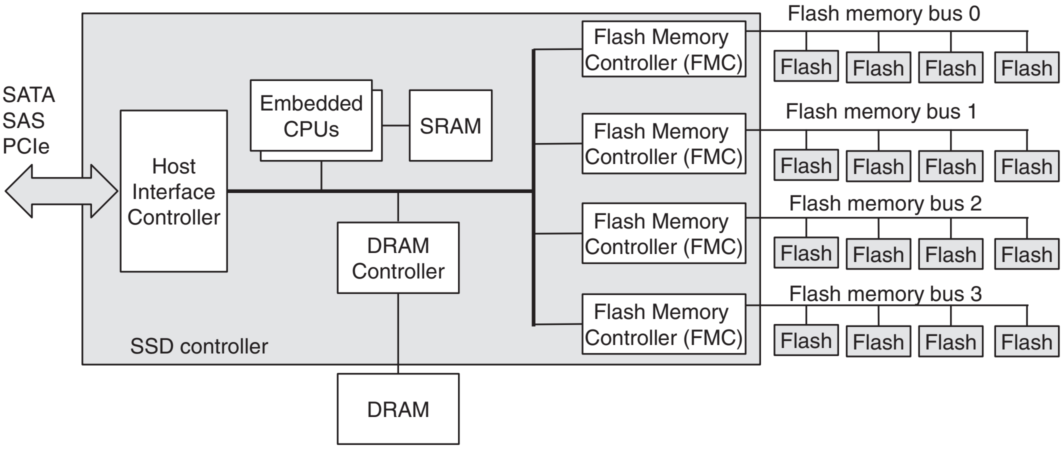


Fig. 1. An example of SSD architecture with 4 channels and 4-way interleaving.

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 [39]. Unlike hard disk, SSD can achieve scalable performance in proportion to the number of NAND flash memory chips to be integrated.

Flash memory controllers (FMCs): 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, 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, which is usually called a *channel*, using multiple chip enable signals [40]. Moreover, multi-channel interleaving is also possible [40]. 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 Application-Specific Instruction-set Processor (ASIP) to support diverse NAND flash memory commands.

Embedded CPU: The embedded CPU(s) together with SRAM provide the execution environment for running flash management software 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 [19]. Typically, a 32-bit RISC processor is used, which runs at 200–400 MHz. Depending on the performance requirement of a given application, multiple CPUs can be incorporated to handle multiple host requests 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 666 MHz or higher, which corresponds to the bandwidth of 2.6 GB/s or more assuming 32-bit bus.

3.2. Future trends

We anticipate innovations in the coming years that will fuel continued bandwidth improvement of SSDs. Especially, all major components in SSDs will (have to) offer wider and higher bandwidth. First of all, significantly larger raw bandwidth will be provided by NAND flash memory through higher per-pin bit rates and wider interleaving in the next five years [37]. The per-pin bit rate of NAND flash chips can scale from 400 Mbps to 800 Mbps to 1.6 Gbps, closely following the proven trend seen in DRAMs. The number of flash channels may increase from 8 to 16 to 32, reaching a bandwidth in excess of 50 GB/s. Clearly, this unprecedented level of raw bandwidth in a single storage device will place a huge pressure on the internal data path of SSDs. Accordingly, we predict that the SSD performance will be CPU and DRAM bandwidth bound, not raw I/O bandwidth bound. By comparison, in a typical SSD today, the total bandwidth along the data path from flash memory all the way to the main controller processor is estimated to be only about 400 MB/s.[2]

To cope with the increased flash memory bandwidth, processor speed and DRAM bandwidth must scale. To improve processor performance, major design parameters like processor pipeline (e.g., issue width and ordering), the number of processors, and processor clock frequency will all have to be carefully determined to obtain adequate performance at reasonable cost. In the case of DRAM, no conventional interface will cost-effectively match the expected flash bandwidth increase. More advanced process like 3D integration and very wide bus width (e.g., 512 bits) DRAM chips may be the only feasible solution [35].

Similarly, SSD host interface will have to evolve. PCI-e has been already adopted in both server and consumer devices. 4- to 16-lane configurations are widely used, providing bandwidth of up to 6.4 GB/s. We are unsure of what host interface standards will emerge to match the flash bandwidth in the future. However, what is inevitable when this very high bandwidth becomes the norm in SSD architectures is that even more daunting pressure will be on the compute hosts and the host side network, which have to process the aggregate storage bandwidth from a large number of (about 100s if not 1000s) SSDs.

In summary, the future technical trends clearly point to an unprecedented situation when there is abundant raw bandwidth at the lowest, flash memory chip and channel level, that renders the embedded processor data path within an SSD and the system-wide network and host-side compute resources a serious bottleneck in large data-intensive applications. Our HW-ISP approach aims at performing data processing at the data source (in or near flash memory chips) and achieving high performance and energy efficiency by leveraging custom hardware logic that augments each flash memory controller. For example, assuming an SSD with 16 channels, the aggregate bandwidth amounts to 6.4 GB/s at 400 Mbps per pin. If the compare operation is performed on the data stream at data speed, the compare performance of an SSD can be 6.4 Giga operations per second (using a byte granularity).

4. Designing in-storage processing on SSDs

In this section, we present the details of ISP to perform primitive database operations, scan and join, inside SSDs. We first describe the characteristics of the database operations that are eligible for ISP. Then we explain how to carry out ISP in a baseline SSD controller for scan operation, and derive an analytic model to capture the performance characteristics of ISP according to the variation of architecture parameters. Note that our model aims at providing average performance of ISP. Based on the developed

[2] The value was obtained from our in-house measurement.

performance model, we identify the potential performance bottleneck and introduce the hardware acceleration of scan. Similarly, we also propose a cost-efficient solution for hash join with special dedicated computing logics and memory resource in place of an embedded CPU in an SSD controller.

4.1. Background

We regard an architecture illustrated in Fig. 1 as a baseline architecture, where an embedded CPU is the only computation resource. Therefore any data to be processed should be sent to DRAM, paying the cost of data transfer time. Two types of DMA transfers exist in the baseline architecture. First one occurs between Flash Memory Controllers (FMCs) and the central DRAM (FMC–DRAM DMA) while the other corresponds to the transfer between the DRAM and the host interface (DRAM–Host DMA).

Two DMA operations above are always involved in any type of data requests to an SSD. Thus, we first develop the performance models of the DMA operations that are basic building blocks of both ISP and IHP.

FMC–DRAM DMA: We begin by defining some notations to describe an FMC and flash chips. We assume that data is stored onto the NAND flash chips evenly throughout all flash channels. The number of FMCs in an SSD controller, i.e., the number of channels, is defined as N_{ch}, and the number of ways in a channel, i.e., the number of flash chips attached to the channel, is given by N_{way}. We define t_{flash_read} as time to read D bytes from flash chips to the FMC ignoring cycles to issue an operation command that consumes negligible bus cycles compared to actual data transfer. Similar to the derivation in [45],

$$t_{flash\_read} = \left\lceil \frac{D}{P} \right\rceil \cdot t_{page} + \left\lceil \frac{D}{P \times N_{way}} \right\rceil \cdot t_R \tag{1}$$

where P is the size of a page in a flash memory in byte, and 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. Similarly, t_{flash_write} corresponds to the write to flash chips.

We also define t_{DRAM_read} and t_{DRAM_write} as time to read or write P bytes from/to the DRAM, respectively. When performing FMC–DRAM DMA, data transfers at both flash chips and DRAM are activated in a pipelined fashion to maximize throughput.

Hence, if we define $t_{FMC2DRAM}$ ($t_{DRAM2FMC}$) as time for FMC-to-DRAM (DRAM-to-FMC) transfer of P bytes, then it is bound to a longer path between them as follows:

$$t_{FMC2DRAM} = \max\left(t_{flash\_read}, t_{DRAM\_write}\right) \tag{2}$$

$$t_{DRAM2FMC} = \max\left(t_{flash\_write}, t_{DRAM\_read}\right). \tag{3}$$

These parameters are statically determined according to the DRAM specification for given amount of data transfer.

DRAM–Host DMA: Next, let us focus on time to transfer P bytes from DRAM to the host interface, $t_{DRAM2Host}$. Similarly, we use $t_{Host2DRAM}$ to denote transfers in reverse direction. We also define additional parameters t_{Host_read} and t_{Host_write} as time to read or write D bytes through the host interface, which corresponds to the host interface speed. Similar to FMC–DRAM DMA, DRAM–Host DMA works in a pipelined fashion. Therefore,

$$t_{DRAM2Host} = \max\left(t_{Host\_write}, t_{DRAM\_read}\right) \tag{4}$$

$$t_{Host2DRAM} = \max\left(t_{Host\_read}, t_{DRAM\_write}\right). \tag{5}$$

Several key parameters that are introduced throughout this section are associated with the components depicted in Fig. 2.

Example. Suppose we read a table that contains two million records. The length of each record is fixed to 128 bytes. Thus, the table size is about 1464.6 MB. The page size of a NAND chip is 8192 bytes and time for a page read, t_R, is given as 50 μs. We assume that 8 NAND chips are attached to an FMC, $N_{way} = 8$, and there are 16 FMCs in the SSD, $N_{ch} = 16$. Then, t_{flash_read} in Eq. (1) will be 206.6 ms. Consider 666 MHz DRAM with 32-bit data width. Then, t_{DRAM_write} for reading the table will be 792.6 μs if we have a single FMC only. If we assume that DRAM bandwidth is shared by all FMCs identically, t_{DRAM_write} corresponding to a single FMC in the system with 16 FMCs would grow as 12.7 ms, meaning that $t_{FMC2DRAM}$ equals t_{flash_read}. If a host interface is SATA 2.0, which offers 300 MB/s of bandwidth, t_{Host_read} for transmitting the table is 3954 ms and $t_{DRAM2Host}$ is the same. The parameters for writing to a disk can be calculated similarly.

4.2. In-storage processing of scan operation

4.2.1. Baseline architecture of scan operation

Scan is the first candidate operation of ISP. We briefly describe how ISP is applied to scan operation. The "*WHERE*" clause of an SQL query is performed on each record, returning only the matching records to the host. ISP of a scan operation on top of the baseline architecture is straightforward as depicted in Fig. 2. In the first step, a table to be processed is partially read from FMCs into the DRAM using FMC–DRAM DMA (ⓐ and ⓑ in the figure). Note that the table is assumed to be 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.

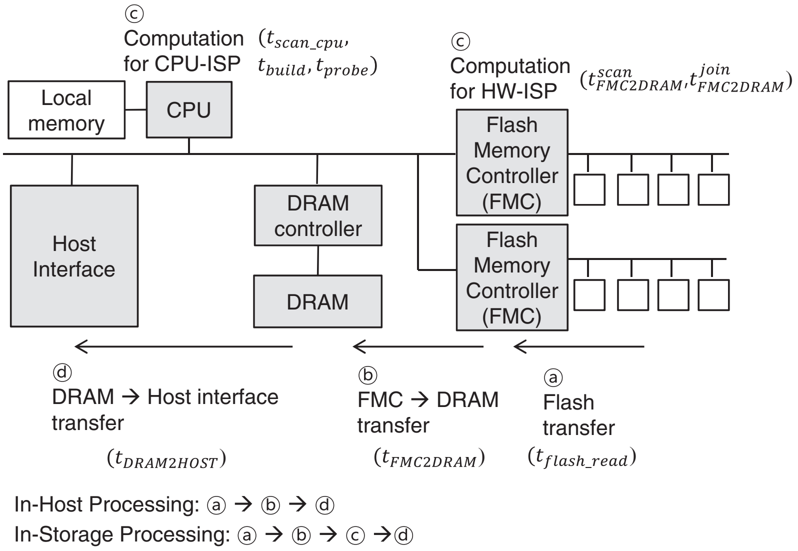

ⓒ
Computation for CPU-ISP
$(t_{scan\_cpu}, t_{build}, t_{probe})$
ⓒ
Computation for HW-ISP
$(t^{scan}_{FMC2DRAM}, t^{join}_{FMC2DRAM})$
Local memory
CPU
Flash Memory Controller (FMC)
Flash Memory Controller (FMC)
Host Interface
DRAM controller
DRAM
ⓓ
DRAM → Host interface transfer
$(t_{DRAM2HOST})$
ⓑ
FMC → DRAM transfer
$(t_{FMC2DRAM})$
ⓐ
Flash transfer
(t_{flash_read})
In-Host Processing: ⓐ → ⓑ → ⓓ
In-Storage Processing: ⓐ → ⓑ → ⓒ →ⓓ

Fig. 2. Data path in the baseline architecture.

Finally, the result of the ISP is delivered to the host using DRAM-Host DMA (ⓓ) in response to the query. The procedure above is continuously 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. We use t_{scan_cpu} to denote the computation time by the embedded CPU to scan the entire table. Then, it is

$$t_{scan\_cpu} = N_{rec} \cdot \left(t_{scan\_comp} + \alpha \cdot t_{scan\_update}\right) \tag{6}$$

where α is scan selectivity, t_{scan_comp} is time for the embedded CPU to execute the scan operation on a single record, and t_{scan_update} is the time to write a matched result, i.e., copy of the record or update of an aggregate value, to the DRAM. The lower the value of 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_update} depend on the implementation of the CPU-ISP. Since the exact modeling of those parameters is non-trivial and beyond the scope of this paper, we resort to empirically measured results, as will be explained in Section 5.

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

$$t_{scan} = \frac{N_{rec} \cdot L_{rec}}{P \cdot N_{ch}} \cdot t_{FMC2DRAM} + t_{scan\_cpu} + \frac{\alpha \cdot N_{rec} \cdot L_{rec}}{P} \cdot t_{DRAM2Host}. \tag{7}$$

Note that data delivered to the host depends on the scan selectivity. If we want to reduce $t_{FMC2DRAM}$ 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 $t_{DRAM2Host}$. Even though $t_{DRAM2Host}$ may also be improved by a faster host interface, we assume the speeds of the host interface and DRAM are fixed so that the variation of $t_{FMC2DRAM}$ depends on the configuration of the flash chip array only, and $t_{DRAM2Host}$ cannot be altered.

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

4.2.2. Hardware acceleration of scan operation

To resolve the aforementioned performance bottleneck, we consider dedicated hardware logic for computing as an alternative to the CPU as shown in Fig. 3. The hardware logic is placed inside an FMC and is composed of a main controller, register file, compare logics and aggregation logics. Scan controller in the figure 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 register file contains 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 HW-ISP is capable of performing the scan operation on-the-fly without degrading the bandwidth of the inbound data stream.

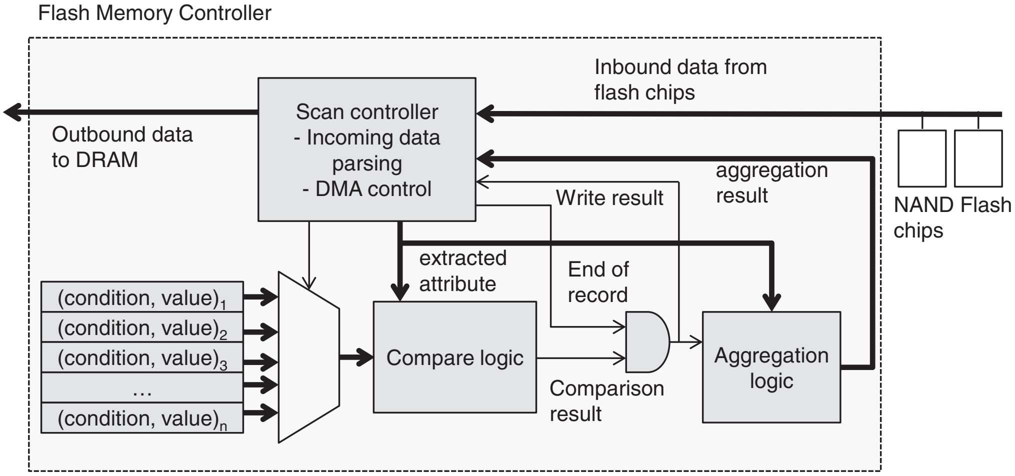


Fig. 3. FMC architecture for the HW-ISP of scan operation.

As a consequence, two terms $t_{FMC2DRAM}$ and $t_{DRAM2Host}$ are the main contributors to the execution time of the scan operation with the HW-ISP. The amount of data that are written to the DRAM can be reduced if FMCs discard unmatched records. Hence, $t_{FMC2DRAM}$ in Eq. (2) is replaced with $t^{scan}_{FMC2DRAM}$ for the HW-ISP, which is:

$$t^{scan}_{FMC2DRAM} = max(t_{flash\_read}, \alpha \cdot t_{DRAM\_write}). \tag{8}$$

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

$$t_{scan} = \frac{N_{rec} \cdot L_{rec}}{P \cdot N_{ch}} t^{scan}_{FMC2DRAM} + \frac{\alpha \cdot N_{rec} \cdot L_{rec}}{P} t_{DRAM2Host}. \tag{9}$$

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

4.2.3. In-host processing of scan operation

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

$$t_{IHP\_scan} = \frac{N_{rec} \cdot L_{rec}}{P \cdot N_{ch}} \cdot t_{FMC2DRAM} + \frac{N_{rec} \cdot L_{rec}}{P} \cdot t_{DRAM2Host} + N_{rec} \cdot t_{IHP\_scan\_rec\_cpu} \tag{10}$$

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 because all records need to be transferred to the host.

4.3. In-storage processing of join operation

4.3.1. Baseline architecture of join operation

Next we move to join operation. The goal of the join is to merge two tables R and S on common join attribute(s). If the value of a join attribute for a record in R matches a record in S, two records are merged into a single one, and the resultant record is output. Even though many join algorithms exist, this paper chooses Hash-Join algorithm since it is known to perform the best for unsorted records in the tables. Hash-Join consists of two phases: *build* and *probe*. In the build phase, Hash-Join first partitions records of each table into buckets. In the probe phase, for each bucket, we first create a hash table for records of the smaller table, say R, on main memory of a host. Then records of the larger table, S, are sequentially matched by probing the hash table with the join attribute, and combined records are produced if a match is found.

CPU-ISP for the build phase of Hash-Join works as follows: for the table R, a part of the table is loaded from an FMC to the DRAM using FMC–DRAM DMA, and the embedded CPU runs hash function $\mathcal{H}_1$ to determine a bucket to which each of records is partitioned, writing the record into the associated buffers on DRAM. Whenever a buffer for a particular bucket is full, the buffer is flushed into flash memories using FMC–DRAM DMA. This procedure continues until the entire records of R and S are examined.

To construct the performance model of CPU-ISP for join, let t_{build_R} be time to partition table R, the size of which is $|R| = N_{rec\_R} \cdot L_{rec\_R}$ where N_{rec} and L_{rec} are the number of records and the length of a record in byte for R, respectively. Similarly, $|S| = N_{rec\_S} \cdot L_{rec\_S}$. Then,

$$t_{build\_R} = \frac{|R|}{P \cdot N_{ch}}(t_{FMC2DRAM} + t_{DRAM2FMC}) + N_{rec\_R} \cdot t_{build\_rec\_R\_cpu} \tag{11}$$

where N_{rec_R} is the number of records in R, and $t_{bulid\_rec\_R\_cpu}$ is time for the embedded CPU to perform hash function $\mathcal{H}_1$ and to write a record of R to a buffer for the proper bucket. Time associated with a table S, t_{build_S}, is formulated in the same way using parameters N_{rec_S}, $t_{bulid\_rec\_S\_cpu}$, and $|S|$. Then the execution time for the build phase t_{build} simply becomes

$$t_{build} = t_{build\_R} + t_{build\_S}. \tag{12}$$

Next, we model the behavior of the probe phase. If we define time to build hash tables of a table R as t_{probe_R}, it is written as

$$t_{probe\_R} = \frac{|R|}{P \cdot N_{ch}} \cdot t_{FMC2DRAM} + N_{rec\_R} \cdot t_{probe\_rec\_R\_cpu} \tag{13}$$

where $t_{probe\_rec\_R\_cpu}$ is the time required to insert a record of R into a hash table. The first product term on the right side of Eq. (13) indicates time to load the partitions of table R into the DRAM, and the second product indicates time to build the associated hash tables of the partitions of R on DRAM using another hash function $\mathcal{H}_2$. Also, let t_{probe_S} be the time to probe records in S and then to create combined records to output to the host. It is

$$t_{probe\_S} = \frac{|S|}{P \cdot N_{ch}} \cdot t_{FMC2DRAM} + N_{rec\_S} \cdot t_{probe\_rec\_S\_cpu} + t_{res\_out} \tag{14}$$

where $t_{probe\_rec\_S\_cpu}$ is the time to probe a record using the hash table and writing a combined record to DRAM if a match is found, and t_{res_out} is the time to output a joined table through the host interface.

The amount of data to be sent to a host depends on how much join occurred. We define the selectivity of join operation, β, as the ratio of the number of records in the final joined table and that of S. Then,

$$t_{res\_out} = \frac{\beta \cdot N_{res\_S} \cdot (L_{rec\_R} + L_{rec\_S})}{P} \cdot t_{DRAM2Host}. \tag{15}$$

We use t_{probe} to denote time for the probe phase, which simply becomes $t_{probe} = t_{prob\_R} + t_{prob\_S}$. Finally, time for join operation, t_{join}, is the sum of latency experienced in the build and probe phases.

$$t_{join} = t_{build} + t_{probe}. \tag{16}$$

In case ISP is performed by the embedded CPU, i.e. CPU-ISP, the portion of the CPU time on join operation is much bigger than that of scan, and, in turn, makes CPU-ISP slower rather than IHP. Experimental results show that, in such a case, more than 90% of the execution time is spent for computation by the embedded CPU on average.

4.3.2. Hardware acceleration of join operation

HW-ISP to accelerate join operation can be achieved similarly to scan. As a result, the dedicated hardware logics, instead of the embedded and host CPUs, carry out the computation for input table partitioning and hash table creation/look up during hash joins. The proposed hardware accelerator for the build phase is depicted in Fig. 4(a). The build phase controller in an FMC parses data stream from flash chips and triggers hardware hashing whenever a join attribute of a record being examined is found. Then, the hash result is delivered to an FCM–DRAM DMA controller such that the DMA controller determines relevant target address of the DMA to store the current record into a buffer for the associated bucket on detecting the end of the record. At the same time, the embedded CPU keeps track of the status of the buffers on the DRAM and initiates DRAM–FMC DMA to flush the buffers whenever they are full. We ignore the overhead caused by such a buffer management by the embedded CPU. In this way, the input tables are instantaneously partitioned at the rate of reading flash chips. In other words, zero CPU time for hashing and partitioning is achieved with this architecture. Therefore, Eq. (11) is rewritten as

$$t_{build\_R} = \frac{|R|}{P \cdot N_{ch}} (t_{FMC2DRAM} + t_{DRAM2FMC}). \tag{17}$$

t_{build_S} is revised similarly.

Next, the working principle of the probe phase is as follows. In order to create a hash table for a partition of table R, records that are located in flash chips of a particular channel are loaded to the associated probe phase controller as shown in Fig. 4(b). Then, it is stored into small record memory. At the same time, the stored address of the record in the record memory is delivered to Content Addressable Memory (CAM) where a hash table is managed similarly to [1]. In general, it is known that CAM is costly in terms of both hardware area and energy consumption with the conventional CMOS technology. For example, CAM cell is 3.8 × and 90 × as large as SRAM cell and DRAM cell, respectively [14]. A recent circuit implementation technique in [14], however, showed that CAM can be implemented with hardware area and energy consumption close to that of DRAM, for example, 1.4 × as large as DRAM in terms of cell size, justifying the use of CAM for the proposed HW-ISP of join operation. In order to minimize the use of CAM, only the pairs of join attribute and address of the associated record in the record memory are kept in the CAM. Note that in Hash-Join, the size of buckets can be flexibly determined according to memory capacity available for buffers. Therefore CAM, which has relatively small capacity compared to the DRAM by an order of magnitude, can be used in practice if we manage small hash table to fit into CAM.

After the hash table creation is finished, the probe phase controller reads records in S and searches a match in the CAM using the join attribute of the records. If a match is found, the corresponding record of R is retrieved from the record memory. Then the probe phase controller creates a joined record, initiating a DMA transfer to write it to output buffer on the DRAM. The buffer is

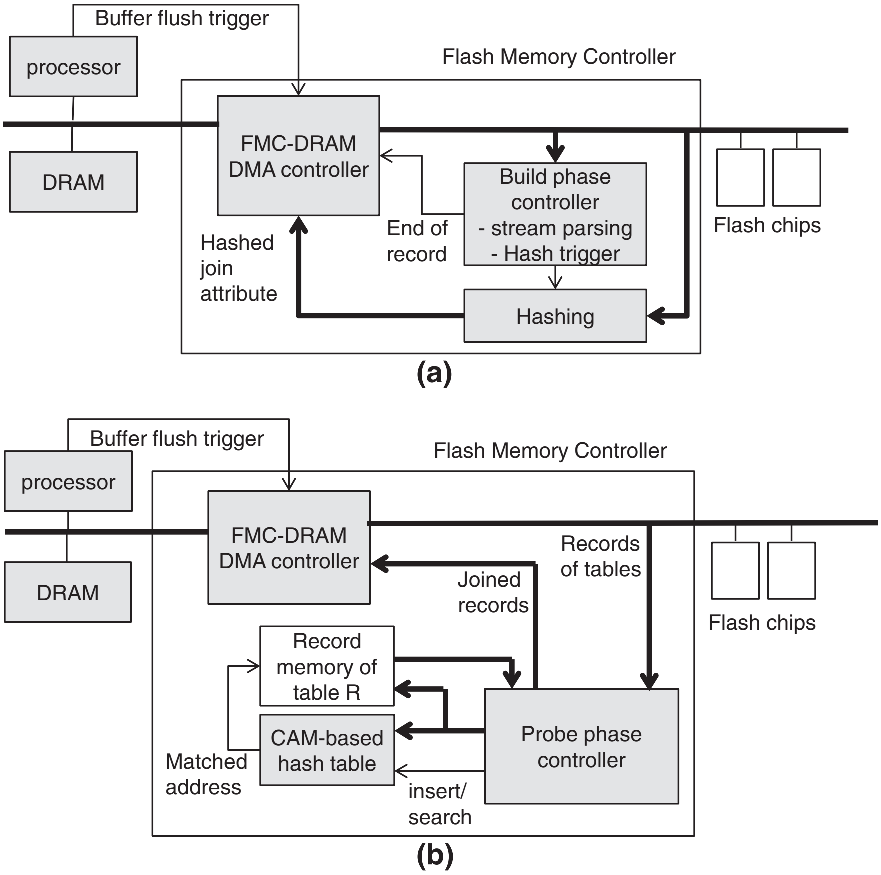


Fig. 4. FMC architecture of HW-ISP for (a) the build phase and (b) the probe phase of join operation.

managed by the embedded CPU as we do in the build phase. Whenever the buffer is full, the CPU initiates DRAM–Host DMA to send the joined records to a host. The execution of the probe phase controller and access to the CAM may be designed to operate with FMC–DRAM DMA in a pipelined way or in parallel. As a result, the latency experienced by the embedded CPU is eliminated so that Eq. (13) is modified as

$$t_{probe\_R} = \frac{|R|}{P \cdot N_{ch}} \cdot t_{FMC2DRAM}. \tag{18}$$

When probing table S, unmatched records are discarded by the probe phase controller, reducing data transfer to the DRAM. Therefore, in this case, $t_{FMC2DRAM}$ in Eq. (17) is redefined as $t^{join}_{FMC2DRAM}$ considering the selectivity, β, which is

$$t^{join}_{FMC2DRAM} = \max\left(t_{flash\_read},\ \beta \cdot t_{DRAM\_write}\right). \tag{19}$$

Then, with the removal of the embedded CPU-based hashing as in Eq. (18), Eq. (14) is modified to

$$t_{probe\_S} = \frac{|S|}{P \cdot N_{ch}} \cdot t^{join}_{FMC2DRAM} + t_{res\_out}. \tag{20}$$

To summarize, the performance improvement by the HW-ISP is mainly due to the elimination of the CPU time and the reduction of DRAM transfer while reading flash memories. Since the gain of the reduced DRAM transfer may vary according to join selectivity, we examine its effects on the ISP performance in the experiments. Note that the proposed HW-ISP accelerates the Hash-Join algorithm, thus supporting an equi join only. A nonequi join will be considered in the future work.

4.3.3. In-host processing of join operation

The derivation of the performance model of join operation by IHP is straightforward and simple. We use t_{IHP_join} to denote the execution time of join operation with two tables R and S. t_{IHP_join} consists of t_{IHP_build} and t_{IHP_probe} with respect to the build and probe phases

$$t_{IHP\_join} = t_{IHP\_build} + t_{IHP\_probe}. \tag{21}$$

If we define average time for a host CPU to process a record in the build phase as $t_{IHP\_build\_rec\_cpu}$, t_{IHP_build} becomes

$$t_{IHP\_build} = \tfrac{|R|+|S|}{P \cdot N_{ch}}(t_{FMC2DRAM} + t_{DRAM2FMC}) + \frac{|R|+|S|}{P} \cdot (t_{DRAM2Host} + t_{Host2DRAM}) \ + (N_{rec\_R} + N_{rec\_S}) \cdot t_{IHP\_build\_rec\_cpu}. \qquad (22)$$

The above equation includes the write back of partitioned tables to an SSD. Similarly, t_{IHP_probe} is formulated as

$$t_{IHP\_probe} = \frac{|R|+|S|}{P \cdot N_{ch}} \cdot t_{FMC2DRAM} + \frac{|R|+|S|}{P} \cdot t_{DRAM2Host} + (N_{rec\_R} + N_{rec\_S}) \cdot t_{IHP\_probe\_rec\_cpu} \qquad (23)$$

where $t_{IHP\_probe\_rec\_cpu}$ denotes the average elapsed time to probe a record in table S in the probe phase including the creation and the lookup of a hash table.

Although the ISP approach is suggested in this paper, it is not always better than IHP. Hence, there are many interesting issues to be further explored in the ISP-based hash join. To name a few, 1) is it still beneficial to take the ISP approach when the join selectivity is higher than one?, 2), more generally, for a given query task, which part of the query plans could be processed more efficiently either in host or in storage, and 3) can we exploit the semi-join technique to reduce the data traffic between host and storage?. But, these issues are beyond the scope of this paper, and we remain them as future work.

5. Experimental setup

In this section, we describe the methodology for evaluating the performance of ISP in comparison with IHP for scan and join operations based on the models developed throughout the previous section. We consider a variety of architectures configured by the parameters in Table 1. We measured other parameters listed in Table 2 with 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, Real View Design Suite 2.2 [2], (for embedded CPU timings), whereby the former corresponds to IHP while the latter to the CPU-ISP. The purpose of the profiling was to measure the execution times devoted purely to computing of the target database operation itself using different processors.

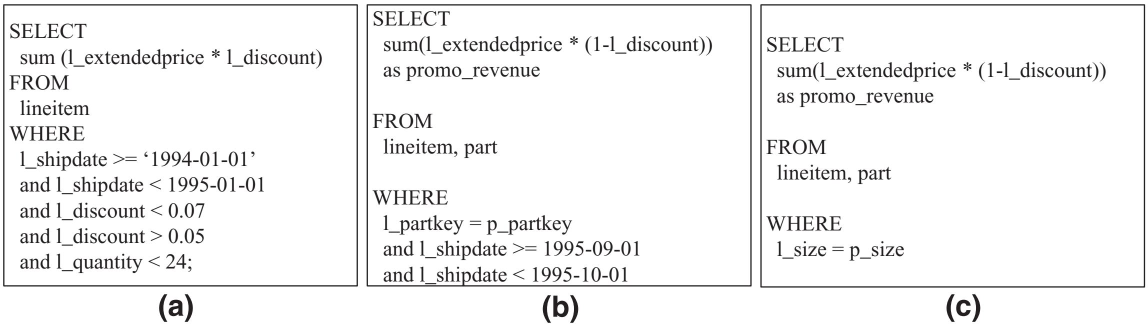
We used a simplified Q6 in the TPC-H benchmarks for scan operation to measure the CPU execution time. We also used two modified versions of Q14 in the benchmarks to perform join and scan–join combination, respectively. We wrote the code for scan and join operations for the aforementioned queries in C. Those workloads are compiled using GCC and armcc, which is a C compiler offered by Real View Design Suite 2.2, with -O2 optimization. Because join operations involved in the original queries of the TPC-H benchmarks are composed of the foreign keys of input tables, executing the original queries will always result in

Table 1
Configurable parameters for the performance model.

Category	Description or parameter	Value(s)
Tables	N_{rec_R}, N_{rec_S} (records)	200,000, 6,001,215
R (*lineitem*), S (*part*)	L_{rec_R}, L_{rec_S} (bytes/record)	168, 128
	N_{ch}	8,16
	N_{way}	8
NAND flash	t_R (μs)	50
	t_{prog} (μs)	1200
	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.

Operation	Component	Description or parameter	Value(s)
Scan	Embedded CPU	t_{comp} (cycles)	24
		t_{result} (cycles)	403
	Host CPU	$t_{IHP\_scan\_cpu}$ (μs)	0.0142
	Selectivity	Scan selectivity, α	0.013
Join	Embedded CPU	$t_{build\_rec\_R\_cpu}$ (cycles)	281
		$t_{build\_rec\_S\_cpu}$ (cycles)	238
		$t_{probe\_rec\_R\_cpu}$ (cycles)	156
		$t_{probe\_rec\_S\_cpu}$ (cycles)	174
	Host CPU	$t_{IHP\_build\_rec\_cpu}/t_{IHP\_probe\_rec\_cpu}$ (μs)	0.206/0.013
	Selectivity	Join selectivity, β	0.000251



```
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;
```

(a)

```
SELECT
  sum(l_extendedprice * (1-l_discount))
  as promo_revenue

FROM
  lineitem, part

WHERE
  l_partkey = p_partkey
  and l_shipdate >= 1995-09-01
  and l_shipdate < 1995-10-01
```

(b)

```
SELECT
  sum(l_extendedprice * (1-l_discount))
  as promo_revenue

FROM
  lineitem, part

WHERE
  l_size = p_size
```

(c)

Fig. 5. Queries used in the experiments, which are modified TPC-H queries: (a) simplified Q6, and modifications of Q14: (b) join only and (c) combination of scan and join.

the join selectivity of 1, which does not serve the purpose of the experiments. In order to have variable (and low) join selectivity, we added a field *size*, which is unique to the *lineitem* table in the benchmark, to the *part* table when generating the table. The queries are depicted in Fig. 5. Evaluations using more complicated quires comprised of other operations such as sort, group-by, nested SQL, and so on, will be considered in future work. The input table sizes were chosen carefully to ensure that the tables stay in the main memory of the workstation throughout the execution and no unintended disk I/Os occur. To do so, the tables were generated with a scale factor of 1.0. We used the same set of conditions in the ARM simulation environment. This method allows us to extract pure CPU times devoted to scanning and joining records without disk access overhead.

In order to validate the accuracy of the proposed performance model of ISP, we built a separate, realistic simulation model of the ISP-enabled SSD controller to produce reference data using a commercial tool, Carbon SoC Designer [7], which is widely used in industry for cycle-accurate simulation of System-on-Chip architectures.

In the simulation, we configured a target architecture with a 200 MHz ARM processor and an 8-channel and 8-way of 100 Mbps flash array. The same configuration was also used to instantiate the performance model developed in Section 4 with the parameters in Table 2, and to compare the estimated performance number with that of the simulation.

6. Evaluation results

In this section, we provide the quantitative results to show the benefit of our ISP approach through five sets of experiments. We first show that our performance estimation model proposed in Section 4 is reasonably accurate compared to detailed simulation for CPU-/HW-ISP of scan and join. Then, we show the throughput gain of ISP against IHP according to the variations of the number of NAND channels and the speed of NAND flash interface. Next, we provide the breakdown of the execution time for ISP and IHP to obtain further insight on the performance improvement of ISP in the previous set of experiments. The fourth experiment examines the effects of query selectivity and host interface speed on the performance of ISP. In the final experiment, we show that ISP also benefits in energy consumption over various NAND interface speeds, and the ISP can be realized into the existing SSD hardware in an economic way through a simple cost estimation.

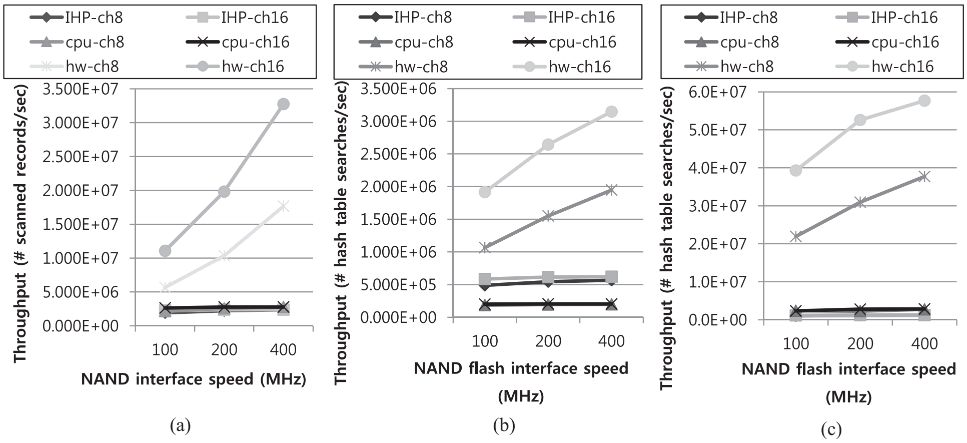
6.1. Accuracy of the performance model

In the first set of experiments, we validate the accuracy of the proposed performance model of ISP by comparing the predicted performance by our model with the cycle-accurate simulation as explained in the previous section. Note that since the simulation speed is quite slow, it took about 5.6 and 10 h to simulate just 1 s of the HW-ISP execution for scan and join, respectively. This implies that the simulation is not appropriate to apply this time-consuming simulation to all architecture candidates.

Table 3 shows the results of the comparison. The estimation error is 9% at most, implying that the analytic model accurately predicts the performance of ISP. The performance prediction of the HW-ISP is relatively more accurate than that of the CPU-ISP because dedicated hardware blocks behave more deterministically than an embedded CPU.

Table 3
Accuracy of the analytic ISP model of scan and join operations compared with cycle-accurate simulations.

Operation	Architecture	Model (cycles)	Simulation (cycles)	Error (%)
Scan	CPU-ISP	297,282	317,446	6.4
	HW-ISP	16,827	16,984	0.9
Join	CPU-ISP	64,505,654	58,700,145	9
	HW-ISP	11,256,688	11,819,522	5



(a) (b) (c)

Fig. 6. Throughput comparisons of ISP and IHP varying the speed of NAND flash interface and the number of channels for (a) scan only, (b) join only, and (c) combined join and scan.

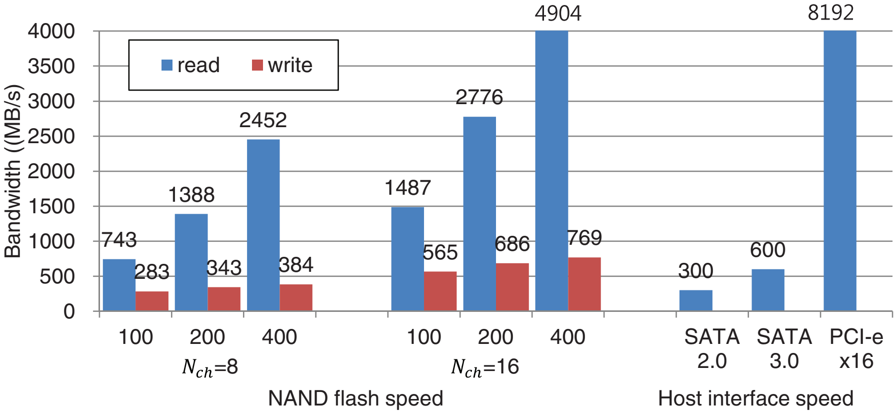


Fig. 7. Sustainable bandwidths of various flash array configurations.

6.2. Throughput comparison

Next, we investigate the effects of the configuration of NAND flash arrays on the throughput of join and scan operations depicted in Fig. 5 to see how ISP and IHP scale according to the variation of architectures. We consider the number of flash chip channels and the speed of NAND flash interface as design axes with the number of ways fixed to 8. For comparison, we apply IHP, CPU-ISP, and HW-ISP to execute the queries. As shown in Fig. 6, each processing method is prefixed by *IHP-*, *cpu-*, and *hw-* and is followed by the number of channels. For example, HW-ISP with 8 NAND channels is "hw-ch8". We assume that the DRAM operates at 666 MHz and the host interface is SATA 2.0, providing the bandwidths of 2.66 GB/s and 300 MB/s, respectively. The values of the parameters used in the performance model follow Table 2 unless otherwise stated.

We observe that the throughput of the HW-ISP scales linearly in NAND flash speed while the CPU-ISP and IHP remain almost the same for both scan and join from Fig. 6(a) and (b). We define the throughputs of scan and join as the numbers of scanned records and hash table lookups per second, respectively. In the case of IHP, the host interface appears as performance bottleneck because the entire table needs to be transmitted to a host, but the bandwidth of the host interface is less than that of the DRAM. In addition, the sustainable bandwidth for reading flash array is also larger than the host interface bandwidth even with the slowest configuration of the flash array, i.e., 100 MHz NAND interface and 8 channels as illustrated in Fig. 7. Moreover, the bandwidth of host interface with SATA 3.0 is also easily saturated in 16-channel architectures. This implies that IHP cannot cope with the growth of flash memory bandwidth efficiently while the HW-ISP fully exploits the rich internal bandwidth by per-FMC dedicated hardware logics, and the performance gap between the HW-ISP and others grows as the internal bandwidth of the storage becomes bigger.

The throughput of the HW-ISP is up to 13.9 × higher over both CPU-ISP and IHP for scan while it is 15.4 × and 5.27 × for join, respectively. The poor performance of the CPU-ISP is mainly due to the low compute capability of the single embedded CPU. According to the execution time profile of the embedded CPU, scan operation requires about 29 bus cycles to process a 128-byte record on average, yielding a data processing rate of 441 MB/s, and join takes about 5.33 μs to probe a 128-byte record on

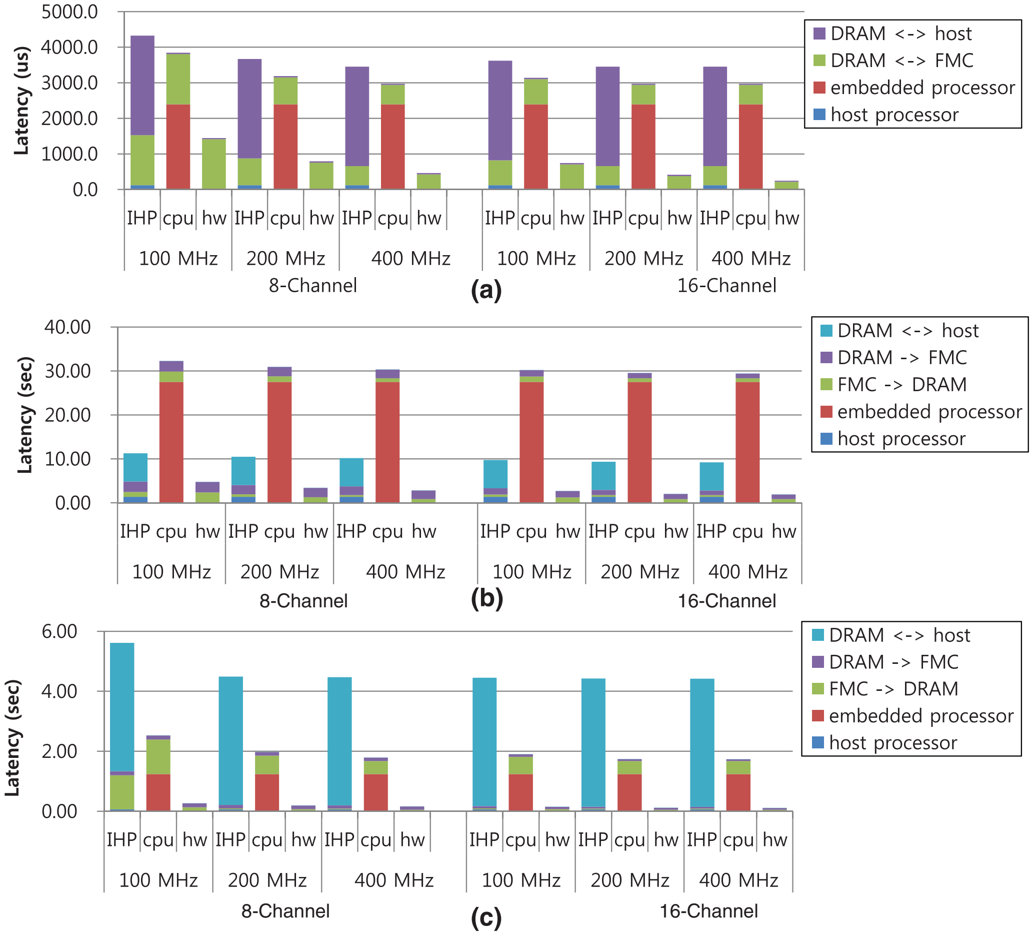


Fig. 8. Breakdown of execution times for the proposed ISP and the IHP varying the speed of NAND flash interface and the number of channels for (a) scan, (b) join, and (c) combined join and scan.

average, yielding 22.9 MB/s of data processing rate. Since, however, inbound data stream from flash memory is at least 743 MB/s, the embedded CPU cannot satisfy the computation requirement.

On the other hand, flash writes in the build phase of join operation mainly limit the performance of the HW-ISP to store table partitions back to flash memory. Write to flash memory requires longer busy period by more than 20 × compared to read operation (see Table 1), resulting in less speed up compared with the read. We assume very conservative read and write speeds in Table 1. Thus the performance advantages of the HW-ISP for hash join would be much bigger if the faster write speed is assumed.

In addition, the I/O time takes large portion of the total execution time in the HW-ISP for the join query (see Fig. 8(b) for details.) Therefore, the relative performance benefit of the HW-ISP would be much larger as the intermediate I/O time during hash joins reduces. In fact, it is typical in a database that join operation takes the result of scan operation as input. In order to confirm this conjecture, we tested a selection-then-join query in Fig. 5(c). Its performance result is presented in Fig. 6(c). The realization of such combined operations using the HW-ISP is straightforward. The resultant outbound stream from the HW-ISP for scan operation in Fig. 3 is forwarded to the input port of the build phase controller for join. In such a way, the portion of table *S* to the build phase controller can be significantly reduced according to scan selectivity. We assume that the baseline architecture of the combined operation has the hardware module for scan only, which is “cpu-ch-∗” in Fig. 6(c). The scan selectivity of the table *lineitem* in Q14 was about 0.013, i.e., 1.3% of table *S* is partitioned and written back to flash chips, leading to the drastic reduction of the HW-ISP execution time. We achieve the speedup of the HW-ISP by 20 × and 47 × compared to the CPU-ISP and IHP, respectively. Note that even the baseline architecture outperforms IHP unlike the simple join case. This is because the reduction of computation demand for partitioning table *S* in the CPU-ISP outweighs that of the data transfer through the host interface in IHP.

6.3. Structural breakdown of execution time

It is worthy de-composing operation execution into several parts for reasoning the performance variations according to the processing methods. The database operations are composed of four steps: 1) transfers between the DRAM and the host interface,

2) transfers between the DRAM and FMCs, 3) computations on the embedded CPU and 4) a host CPU as shown in Fig. 8. In particular, we distinguish DRAM-to-FMC transfers from FMC-to-DRAM transfers since writing partitioned tables back to flash array is involved in the build phase of join operation.

We observe that a major portion of execution time for IHP is devoted to data transfer via the host interface from Fig. 8. The slowest flash configuration of 8 channels with 100-Mbps flash offers greater bandwidth than the host interface as stated previously. Thus, increasing the bandwidth of flash array does not benefit in such a situation. In the CPU-ISP, the similar observation is found except that the embedded CPU is a bottleneck instead of the host interface. The amount of transfer via the host interface depends on the join selectivity. In our experiment, the selectivity was quite low, 0.000251, where 1507 matches were found out of the table with 6 million records. Hence, the latency for transfer to the host disappears. However, poor computing capability of the embedded CPU prolongs the latency significantly, necessitating the hardware acceleration. Furthermore, the entire table should be loaded to the DRAM for the embedded CPU to perform the join operation. Therefore, performance bottleneck moves from flash chips to the DRAM according to the increase of the flash channel bandwidth. For example, the sustainable bandwidth of 16 channels with 200 MHz flash chips amounts to 2776 MB/s while the peak bandwidth of DRAM at 666 MHz is 2664 MB/s. This explains why time for DRAM-FMC transfer remains the same in the underlying 16 channels with both 200 Mbps and 400 Mbps flash chips.

The HW-ISP, however, does not suffer from the aforementioned DRAM bandwidth limitation since an FMC discards unnecessary data without sending to the DRAM. The more records are filtered out in an FMC, the better the bandwidth of flash array is exploited. For instance, since the scan selectivity is quite small, which is 0.013, the data transfer through the host interface is greatly reduced with the ISP. In other words, another benefit of the HW-ISP is flexibility to match computation demand of a target database operation using special purpose processing elements in parallel compared to IHP and the CPU-ISP.

Fig. 8(c) shows that the input size reduction of join operation by the preceding scan affects the computation time of the embedded CPU greatly than other components, which is reduced by about 22 × compared to the join-only case. Hence, the host interface requires the longest time and, in turn, makes IHP the worst solution as a result. On the other hand, the performance of the HW-ISP is mostly dominated by flash read and write. This implies that the reduction of input to join is the most advantageous to the HW-ISP. The HW-ISP outperforms at most by up to 20 × and 47 × compared to the CPU-ISP and IHP respectively in our experiment.

To summarize, ISP provides the means to exploit the internal bandwidth of flash array maximally. The dedicated hardware logics at each FMC perform on-the-fly computation without hindering the flow of inbound data from flash array, efficiently offloading computation burden and entailed communication with a host. As a result, the performance of the database operations is bound to the performance of flash array itself with the HW-ISP.

6.4. Effects of selectivity and host interface speed

The previous experiments were conducted under the condition of the low selectivity. Therefore, data transfers via the host interface do not have significant impacts on the performance of ISP. One may guess that if the selectivity approaches 1, then the host interface may degrade the performance of ISP. To investigate the impacts of the selectivity and the host interface speed, we carried out another set of experiments. We consider three host interface bandwidth (300 MB/s, 600 MB/s, and 8 GB/s) to represent the existing host interfaces, SATA 2.0, SATA 3.0, and PCI-Express 2.0 x16 lane, respectively. To ensure that the bandwidth of the host interface is fully utilized, we set the configuration of the DRAM and flash array to the highest performance; we use the DRAM of 1333 MHz and 16 channels with 400 Mbps flash chips. Such a configuration is maximally available in practice for today's high performance SSDs, offering the sustainable bandwidths of 5328 MB/s and 4904 MB/s, respectively.

Fig. 9 shows that the performance degradation of ISP appears to degrade as the selectivity becomes larger. Such a tendency is more obvious with the slower host interface while the performance of IHP is invariant regardless of the selectivity. But the reasons behind such behaviors are different according to target operation and host interface bandwidth used.

Scan computing capability of the host CPU excluding disk access overhead can be made simply by referring to $t_{IHP\_scan\_cpu}$ and the input table size in Table 2, which is about 8.82 GB/s and is greater than the bandwidth offered by the fastest host interface, PCI-e. Thus, the performance of IHP for scan depends on the SSD. The DRAM inside the SSD is the fastest component in the configuration, and then, the host interface or flash memory dominates the performance of data transfer to the host. In the case of PCI-e, the host interface provides bigger bandwidth than flash memory in our configuration, making the flash memory performance bottleneck. With other host interfaces providing lower bandwidths than the flash memory, however, the interfaces are bottlenecks for IHP. The HW-ISP tends to perform worse, being close to the case of IHP according to the increasing scan selectivity as shown in Fig. 9(a). This is due to latency for transferring matched records from the DRAM to the host interface in proportion to the scan selectivity.

On the other hand, when performing join operation, it turns out that the host CPU occupies up to 66% of the execution time as we increase the join selectivity. This is due to the high computation demand of the join operation. As a result, the deficient computing capability of the host CPU makes performance gap between IHP and the HW-ISP over all the host interfaces as shown in Fig. 9(b). Even in the worst case of the selectivity of 1.0, the HW-ISP with PCI-e outperforms the IHP by 1.7 times. Similar performance gains are also found in other host interfaces. Those results confirm that the benefit of the HW-ISP is more prominent on a target database operation where more computation demand exists.

In order to see the applicability of the ISP in more practical case, we repeat the similar experiment to execute the combined join and scan. We set the scan selectivity of *S* table to 0.013 referring to the previous experiments, and the maximum join

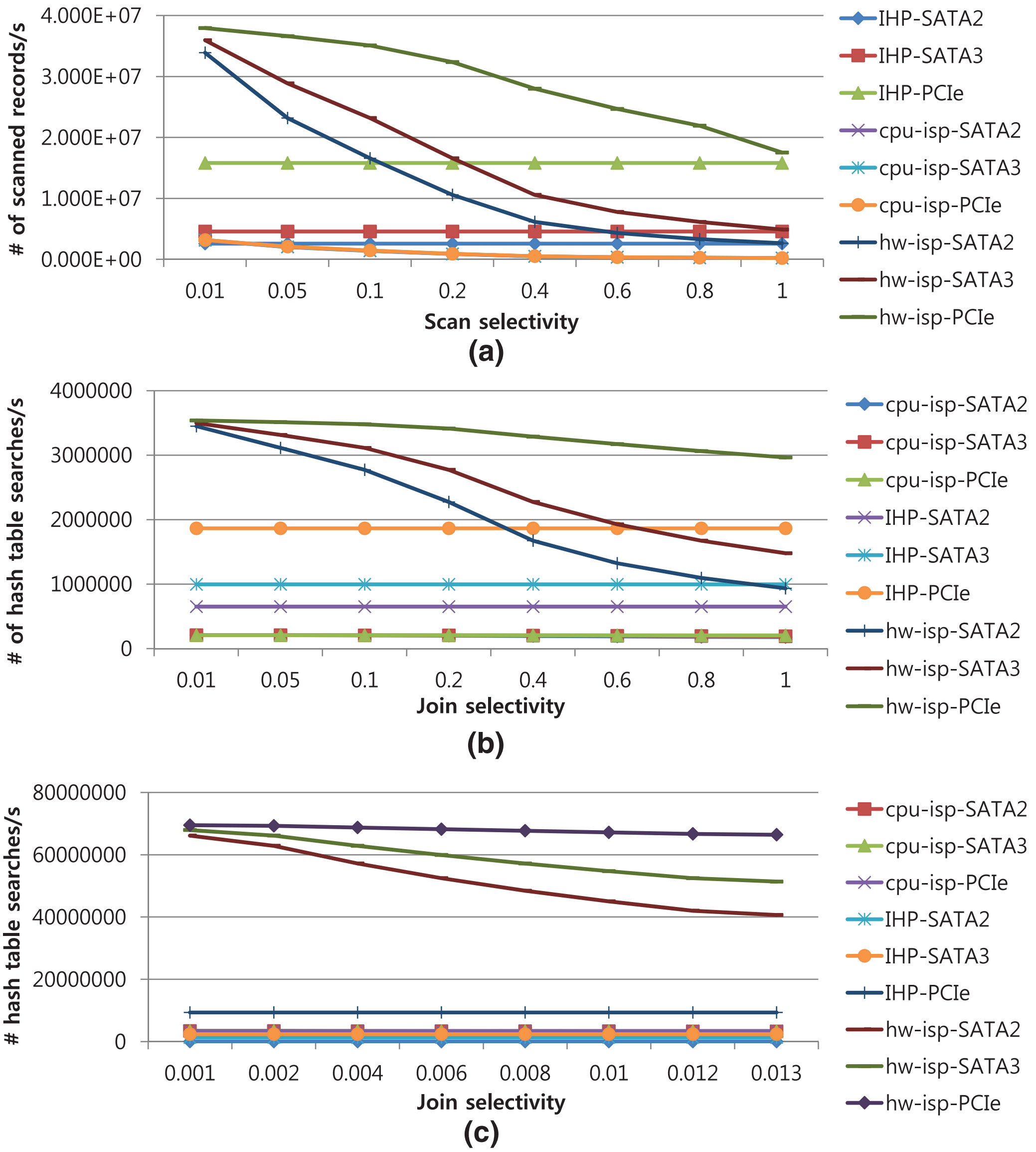


Fig. 9. Performance of the ISP according to the variation of the selectivity and the host interface speed for (a) scan, (b) join, and (c) combined join and scan.

selectivity is upper-bound to the scan selectivity. The graphs in Fig. 9(c) demonstrate that the benefit of the ISP remains evident throughout the range. The HW-ISP and the CPU-ISP are at least 32 × and 2.6 × faster than IHP. It is obvious that the graph in Fig. 9(c) will converge to Fig. 9(b) according to the increase of the scan selectivity.

6.5. More on energy and cost

Besides the performance improvement, energy saving is another key benefit of ISP. For example, all data should be transferred to the host CPU for comparison in a conventional system, going through host interfaces, main memory (DRAM), and L1/L2 cache to search for a specific value associated with a given key. The data finally loaded into host CPU registers is then compared with a given key, and if matched, 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 multiple system components and cables. On the contrary, the proposed ISP performs the compare operations inside an SSD through the simple hardware logics in FMCs. As a result, most useless data are filtered immediately at the minimum distance from the storage medium.

In order to estimate the benefit of ISP on energy saving, we compare the energy consumption of IHP and ISP to execute the simplified join query in Fig. 5(b) as described in Section 5. The energy consumption of IHP was obtained by measuring actual

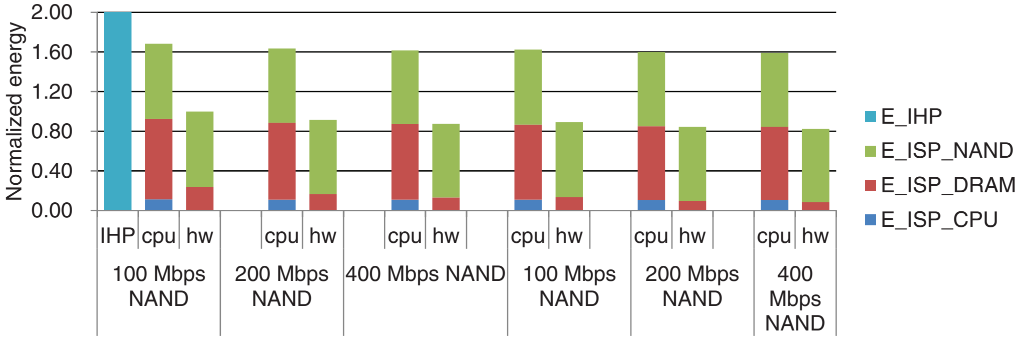


Fig. 10. Normalized energy consumption of IHP and ISP for join operation.

power dissipation of a workstation with Xeon 3.0 GHz processor and 2 GB main memory running Linux. We used Oracle 11g with 8 MB hash memory to execute the query. The energy estimation of ISP considers major architecture components of SSDs to execute the proposed ISP, which includes the embedded CPU and its local memory, the DRAM, NAND flash chips, and the hardware accelerator. Then the total energy is calculated as a sum of component-wise energy consumption. The embedded CPU's energy is obtained from [3]. The embedded CPU's local memory, CAM in the hardware accelerator, and the DRAM are estimated using [17,27]. Flash memory energy according to the interface speed is obtained from in-house power measurement. Lastly, the energy of the hardware accelerator is calculated based on [22]. We omit the explanation of further details.

We provide the energy consumption of IHP and the ISPs that are normalized to that of the HW-ISP in Fig. 10. Because full system energy consumption is measured directly for IHP, we are not able to provide its breakdown. Both the baseline and the HW-ISP achieve energy savings by an order of magnitude compared to IHP, which is much more significant than the performance improvement. The graph shows that most portion of energy is taken by components along with data path, i.e., flash memory and the DRAM. The flash memory interface speed does not have significant effects on energy saving because the protocol overhead of flash read/write is constant and dominant in flash memory operations. On the other hand, the DRAM energy consumption decreases in proportion to the execution time of the ISP. The portion of computation is at most 11% in the CPU-ISP and is negligible in the HW-ISP. This implies that the HW-ISP is an energy-efficient solution with negligible overhead for the hardware computation modules, which is also the case in the performance aspect.

Lastly, our in-house bill of material (BOM) analysis indicates that the SSD controller occupies less than 10% of the total SSD cost, which is dominated by the flash memory array. Therefore, even if we increase the controller's cost by 10% using additional embedded CPUs and special hardware logic, the hardware overhead of HW-ISP is only 1% of the total system cost. See [9] for more details.

7. Conclusion

We presented the idea of "*in-storage processing*" (ISP) for database scan and join operations 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 HW-ISP approach realizes significant performance improvement for database scan and join, compared to the conventional host processing approach. Moreover, the HW-ISP consumes much less energy at negligible cost overhead. In future work, we will extend ISP to other database operations such as *group by* and *sorting*. The validation of the proposed performance model by comparing with real hardware also will be considered.

Acknowledgments

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 (NRF-2013R1A1A1012715 and NRF-2013R1A1A1013384), IT R&D program MKE/KEIT (No. 10041608, Embedded system Software for New-memory based Smart Device), Korea Institute of Science and Technology Information (RFP-2015-19) in part, and Institute for Information & communications Technology Promotion (IITP) (R0126-15-1088).

References

[1] D. Agrawal, A.E. Abbadi, Hardware acceleration for database systems using content addressable memories, in: Proceedings of International Workshop on Data management on New Hardware, 2005. Article No. 1.

[2] ARM, ARM RealView Development Suite. http://www.arm.com/products/tools/software-tools/rvds/index.php.
[3] ARM, Cortex A9 processor. http://www.arm.com/products/processors/cortex-a/cortex-a9.php.
[4] D.-H. Bae, J.-H. Kim, S.-W. Kim, H. Oh, C. Park, Intelligent SSD: a turbo for big data mining, in: Proceedings of ACM International Conference on Information & Knowledge Management, 2013, pp. 1573–1576.
[5] H. Boral, D.J. Dewitt, Database machines: an idea whose time passed? A critique of the future of database machines, in: Proceedings of International Workshop on Database Machines, 1983, pp. 166–187.
[6] L. Bouganim, P. Bonnet, Flash device support for database management, in: Proceedings of Biennial Conference on Innovative Data Systems Research, 2011, pp. 1–8.
[7] Carbon Design Systems, SOC designer plus. http://carbondesignsystems.com/SocDesignerPlus.aspx.
[8] B.Y. Cho, W.S. Jeong, D. Oh, W.W. Ro, XSD: accelerating mapreduce by harnessing the GPU inside an SSD, in: Proceedings of the 1st Workshop on Near-Data Processing, 2013a.
[9] S. Cho, C. Park, H. Oh, S. Kim, Y. Yi, G.R. Ganger, Active disk meets flash: a case for intelligent SSDs, in: Proceedings of ACM conference on InternationProceedings of 1st Workshopal Conference on Supercomputing, 2013b, pp. 91–102.
[10] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, in: Proceedings of Symposium on Opearting Systems Design & Implementation, 2004, pp. 137–150.
[11] J. Do, Y.-S. Kee, J.M. Patel, C. Park, K. Park, D.J. DeWitt, Query processing on smart SSDs: opportunities and challenges, in: Proceedings of ACM SIGMOD International Conference on Management of Data, 2013, pp. 1221–1230.
[12] M. Gokhale, J. Cohen, A. Yoo, W.M. Miller, A. Jacob, C. Ulmer, R. Pearce, Hardware technologies for high-performance data-intensive computing, IEEE Comput. 41 (4) (2008) 60–68.
[13] N. Govindaraju, J. Gray, R. Kumar, D. Manocha, GPUTeraSort: high performance graphics co-processor sorting for large database management, in: Proceedings of ACM SIGMOD International Conference on Management of Data, 2006, pp. 325–336.
[14] Q. Guo, X. Guo, Y. Bai, E. ?pek, A resistive TCAM accelerator for data-intensive computing, in: Proceedings of the IEEE/ACM International Symposium on Microarchitecture, 2011, pp. 339–350.
[15] B. He, K. Yang, R. Fang, M. Lu, N. Govindaraju, Q. Luo, P. Sander, Relational joins on graphics processors, in: Proceedings of ACM SIGMOD International Conference on Management of Data, 2008, pp. 511–524.
[16] B. Head, Data doubles daily a decade hence 2011. http://www.itwire.com/cloud-computing/45827-data-doubles-daily-a-decade-hence.
[17] HP Labs., Cacti 5.3, 2008.
[18] Intel, Solid-state drives in the enterprise: a proof of concept, White Paper, 2009.
[19] J.-U. Kang, H. Jo, J.-S. Kim, J. Lee, A superblock-based flash translation layer for NAND flash memory, in: Proceedings of International conference on Embedded software, 2006, pp. 161–170.
[20] Y. Kang, Y.-s. Kee, E.L. Miller, C. Park, Enabling cost-effective data processing with smart SSD, in: Proceedings of IEEE Symposium on Mass Storage Systems and Technologies, 2013, pp. 1–12.
[21] J.-U. Kang, J. Hyun, H. Maeng, S. Cho, The multi-streamed solid-state drive, in: Proceedings of USENIX Workshop on Hot Topics in Storage and File Systems, 2014.
[22] U.J. Kapasi, S. Rixner, W.J. Dally, B. Khailany, J.H. Ahn, P. Mattson, J.D. Owen, Programmable stream processors, IEEE Comput. 36 (8) (2003) 54–62.
[23] K. Keeton, D.A. Patterson, J.M. Hellerstein, A case for intelligent disks (idisks), ACM SIGMOD Rec. 27 (3) (1998) 42–52.
[24] C. Kim, T. Kaldewey, V.W. Lee, E. Sedlar, A.D. Nguyen, N. Satish, J. Chhugani, A.D. Blas, P. Dubey, Sort vs. hash revisited: fast join implementation on modern multi-core cpus, Proc. VLDB Endow. 2 (2) (2009) 1378–1389.
[25] S. Kim, H. Oh, C. Park, S. Cho, S.-W. Lee, Fast, energy efficient scan inside flash memory SSDs, in: Proceedings of International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures, 2011.
[26] Y.-S. Lee, L.C. Quero, Y. Lee, J.-S. Kim, S. Maeng, Accelerating external sorting via on-the-fly data merge in active SSDs, in: Proceedings of USENIX Workshop on Hot Topics in Storage and File Systems, 2014.
[27] Micron, System power calculators. http://www.micron.com/support/dram/power_calc.html.
[28] R. Mueller, J. Teubner, FPGA what's in it for a database? in: Proceedings of ACM SIGMOD International Conference on Management of Data, 2009, pp. 999–1004.
[29] R. Mueller, J. Teubner, G. Alonso, Data processing on FPGAS, Proc. VLDB Endow. 2 (1) (2009) 910–921.
[30] Netezza, The Netezza data appliance architecture: a platform for high performance data warehousing and analytics, White Paper, 2010.
[31] Objective Analysis, Solid State Drives in the Enterprise, Technical Report, 2008.
[32] OpenSSD Project. http://www.openssd-project.org/wiki/The_OpenSSD_Project.
[33] Oracle, Oracle exadata, White Paper, 2010.
[34] A. Pavlo, E. Paulson, A. Rasin, D.J. Abadi, D.J. DeWitt, S. Madden, M. Stonebraker, A comparison of approaches to large-scale data analysis, in: Proceedings of ACM SIGMOD International Conference on Management of Data, 2008, pp. 165–178.
[35] S.H. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, F. Li, NDC: analyzing the impact of 3d-stacked memory+logic devices on MapReduce workloads, in: IEEE International Symposium on Performance Analysis of Systems and Software, 2014, pp. 190–200.
[36] E. Riedel, G.A. Gibson, C. Faloutsos, Active storage for large-scale data mining and multimedia, in: Proceedings of International Conference on Very Large Data, 1998, pp. 62–73.
[37] S.J. Roland Schuetz, Looking ahead to higher performance SSDs with HLNAND, in: Proceedings of Flash Memory Summit, 2010.
[38] Samsung Electronics, Samsung solid state drive basics 2010. http://www.samsungssd.com/meetssd/overview.
[39] Samsung Electronics, Samsung offers industry's first 64-gigabit MLC NAND flash, using toggle DDR 2.0 interface 2011. http://www.samsung.com/global/business/semiconductor/news-events/press-releases/detail?newsId=4023.
[40] Y.J. Seong, E.H. Nam, J.H. Yoon, H. Kim, J.-y. Choi, S. Lee, Y.H. Bae, J. Lee, Y. Cho, S.L. Min, Hydra: a block-mapped parallel flash memory solid-state disk architecture, IEEE Trans. Comput. 59 (7) (2010) 905–921.
[41] M. Sivathanu, L.N. Bairavasundaram, A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, Database-aware semantically-smart storage, in: Proceedings of USENIX Conference on File and Storage Technologies, 2005, pp. 239–252.
[42] Teradata, Teradata active enterprise data warehouse. http://www.teradata.com/Active-Enterprise-Data-Warehouse/.
[43] S. testest, Y. Kim, S.S. Vazhkudai, P. Desnoyers, G.M. Shipman, Active flash: out-of-core data analytics on flash storage, in: Proceedings of IEEE Symposium on Mass Storage Systems and Technologies, 2012, pp. 1–12.
[44] D. Tiwari, S. Boboila, S.S. Vazhkudai, Y. Kim, X. Ma, P.J. Desnoyers, Y. Solihin, Active flash: towards energy-efficient, in-situ data analytics on extreme-scale machines, in: Proceedings of USENIX Conference on File and Storage Technologies, 2013, pp. 119–132.
[45] S.-K. Won, S.-H. Ha, E.-Y. Chung, Fast performance analysis of NAND flash-based storage device, IET Electron. Lett. 45 (24) (2009) 1219–1221.
[46] L. Woods, Z. Istvan, G. Alonso, Ibex – an intelligent storage engine with support for advanced SQL offloading, Proc. VLDB Endow. 7 (11) (2014) 963–974.
[47] L. Woods, J. Teubner, G. Alonso, Less watts, more performance: an intelligent storage engine for data appliances, in: Proceedings of ACM SIGMOD International Conference on Management of Data, 2013, pp. 1073–1076.
[48] H.-c. Yang, A. Dasdan, R.-L. Hsiao, D.S. Parker, Map-Reduce-Merge: simplified relational data processing on large clusters, in: Proceedings of ACM SIGMOD international conference on Management of Data, ACM, 2007, pp. 1029–1040.