# RISP: A Reconfigurable In-Storage Processing Framework with Energy-Awareness

Xiaojia Song
*San Diego State University*
*5500 Companile Drive*
*San Diego, USA*
*Email: xsong2@sdsu.edu*

Tao Xie
*San Diego State University*
*5500 Companile Drive*
*San Diego, USA*
*Email: txie@mail.sdsu.edu*

Wen Pan
*San Diego State University*
*5500 Companile Drive*
*San Diego, USA*
*Email: wpan@sdsu.edu*

***Abstract*—Existing in-storage processing (ISP) techniques mainly focus on maximizing data processing rate by always utilizing total storage data processing resources for all applications. We find that this ”always running in full gear” strategy wastes energy for some applications with a low data processing complexity. In this paper we propose RISP (Reconfigurable ISP), an energy-aware reconfigurable ISP framework that employs FPGA as data processing cells and NVM controllers. It can reconfigure storage data processing resources to achieve a high energy-efficiency without any performance degradation for big data analysis applications. RISP is modeled and then validated on an FPGA board. Experimental results show that compared with traditional host-CPU based computing RISP (with 16 channels or more) improves performance by** $1.6$**-**$25.4\times$ **while saving energy by a factor of** $2.2$**-**$161$**. Further, its reconfigurability can provide up to** $77.2\%$ **additional energy saving by judiciously enabling data processing resources that are sufficient for an application.**

***Keywords*-In-storage processing; FPGA; SSD; energy-aware.**

## I. Introduction

Big data analysis is the process of examining massive data sets (e.g., scientific data, photographs and videos) to discover useful information such as hidden patterns and market trends. Typically, its first step is to filter/aggregate a huge amount of raw data. The outcome of this pre-processing step is usually a much smaller data set, which will then be carefully examined by a particular algorithm. Delivering these large raw data sets all the way from a storage system to host CPU(s) puts a tremendous pressure on a host-CPU based computing architecture as it incurs a substantial data transfer latency and energy consumption [4].

One promising solution to alleviate the pressure is in-storage processing (ISP, also called in-storage computing or ISC) [1], which offloads part of conventional host-CPU based computations to a storage device. A modern flash SSD (solid state drive) now has a very high internal bandwidth that often exceeds its external bandwidth by factors of 2-4$\times$ [4]. Therefore, processing data in-storage could achieve a higher performance and consume less energy than delivering them all the way to the host CPU(s). This is mainly because a modern flash SSD is normally equipped with a multi-core general-purpose embedded CPU (e.g., two ARM Cortex R7 cores [9]) and multiple channels. It has a considerable data processing capability, which recently sparks a resurgent interest in developing NVM (non-volatile memory)-based ISP techniques from both industry and academia [4][7][9][12][13][14][15][21][22]. These ISP techniques all have shown that they can significantly improve performance, energy-efficiency, or both for some applications compared with traditional host-CPU based computing. However, a common issue among them is that they always utilize the entire storage data processing resources (e.g., all channels in an SSD) to serve a data-intensive application without considering its data processing complexity, which in this paper is defined as the number of CPU cycles per byte or CPB [4]. An application with a higher CPB is more compute-intensive as it demands more CPU cycles to process its input data. We find that this ”always running in full gear” strategy could cause two problems. First, it incurs unnecessary energy consumption for some data-intensive applications with a low CPB. For these applications, since their in-storage data processing bandwidth could be higher than the external data transfer bandwidth, data transfer from a host interface to the host might become a bottleneck. Thus, in-storage devices (e.g., channels and NVM controllers) have to stay in a stand-by status to wait for the completion of data transfer, which wastes energy. For instance, we discover that Sobel [18], an image processing application with a low CPB, only needs to use 10 of 64 channels in an SSD so that its in-storage data processing bandwidth can match the external data transfer bandwidth. Using more than 10 channels only wastes energy without any further performance improvement (see Section V-C). Second, more storage data processing resources (e.g., channels) working simultaneously requires more design efforts to solve issues like high peak current. Excessive peak current could cause some problems (e.g., supply voltage drop, ground bounce, signal noise, etc.), which lead to unreliable SSD operations [10].

To solve the common issue, in this paper we propose a new ISP framework called RISP (reconfigurable in-storage processing), which employs FPGA (field-programmable gate array) as data processing cells and NVM controllers. It is more energy-efficient than ISP techniques that use embedded processors as data processing units because FPGA consumes less energy than a general-purpose embedded processor [3]. Besides, FPGA can easily realize massive pipelining and parallelism [3]. It also avoids the two limitations of embedded processors [4][21] (see Section II). RISP has two desirable features. First, it can better serve an application by implementing the application into its FPGA-based data processing units (e.g., processing cell on each channel). It only needs to be reconfigured when it starts to serve a new application. We assume that within a given period of time it only serves one application. In other words, simultaneously serving multiple applications is not supported by RISP in order to avoid frequent FPGA reconfigurations. Second, it is the first ISP technique that can reconfigure storage data processing resources according to the data processing complexity of an application. In particular, for an application with a low CPB it can provide further energy-saving by judiciously enabling data processing resources that are sufficient for the application. Next, we build performance and energy consumption models for RISP and a conventional host-CPU based computing architecture (hereafter, Baseline). Further, we utilize three widely used data-intensive applications (i.e., K-Means [17], Binarization [18], Sobel [18]) with distinct CPBs to validate the models on a Linux machine and a Xilinx FPGA board [24]. Finally, we leverage them to evaluate RISP by comparing it with Baseline. We also evaluate energy-saving brought by the reconfigurability of RISP, which is defined as its capability of judiciously selecting an appropriate number of channels to serve a particular application. Experimental results show that RISP improves performance by 1.6-25.4$\times$ while reducing energy consumption by 2.2-161$\times$. Its reconfigurability offers up to 77.2% additional energy-saving.

The rest of paper is organized as follows. Section II summarizes related work. Section III presents RISP framework. Section IV provides performance and energy consumption models for RISP and Baseline. Section V evaluates RISP followed by Section VI, which concludes the paper.

## II. Related work

ISP techniques leverage analytical modeling [4][21], hardware prototyping [7][9][12][13][14][15], or both [22] to quantify their performance or energy gains compared with traditional host-CPU based computing. For modeling-based ISP techniques [4][21], it is very hard for a third-party to reproduce their experimental results although their models and major parameters are provided [9]. This is mainly because the detailed information about how to integrate these models to generate the reported results is unclear. Besides, the parameters (e.g., data transfer rate per NVM channel and clock frequency of an embedded CPU) used in these prior studies [4][21] are already outdated. Prototyping-based ISP techniques [7][9][12][13][14][15] employ a customized hardware prototype. Unfortunately, these prototypes are normally not publicly available. In short, none of existing ISP techniques is open source in the sense that its results can be reproduced by a third-party.

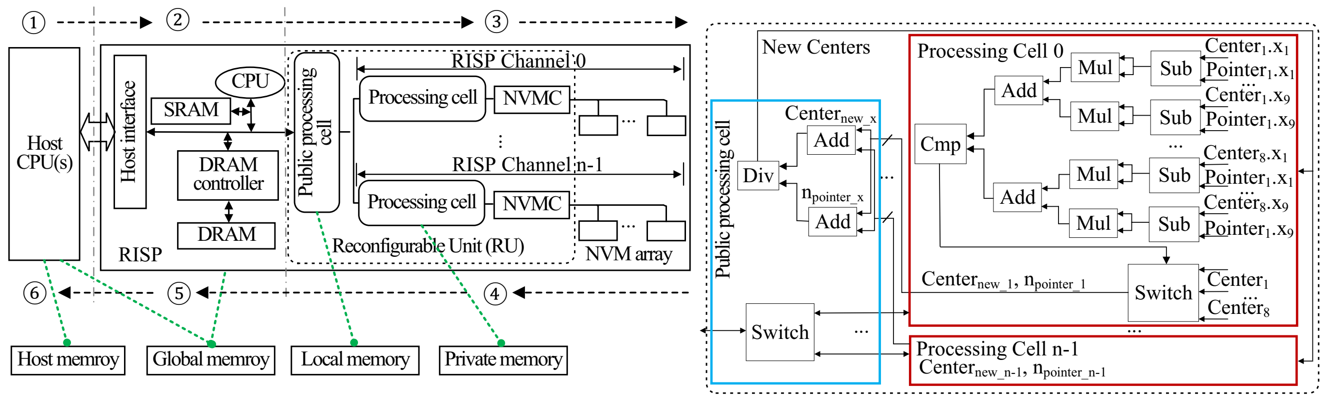
In addition, ISP techniques normally target different applications running on distinct architectures. For example, while iSSD [4] focuses on some common data-intensive application kernels (e.g., K-means [17], a mean-based data partitioning method) running on an enterprise server, Active Flash [21] concentrates on high performance computing (HPC) simulations (e.g., astrophysics and climate) performed on a supercomputer like Gordon. Biscuit [9], Caribou [12], BlueDBM [14], and YourSQL [13], however, focus on how to integrate ISP in a database system. Summarizer [15] designs a set of application programming interfaces (APIs) that can be used by a host application to offload a task to the SSD processor. The huge differences among these applications and architectures also make a comparison between two ISP techniques challenging. These huge differences plus non-open-source explained why all existing ISP techniques used the traditional host-CPU based computing as a competitor and none of them compared its performance with one of its peers. In this paper, we also compare RISP with the conventional host-CPU based computing.

Part of existing ISP techniques [4][9][21] utilize embedded processors as in-storage CPUs to process data in-situ. For example, in order to translate raw flash memory bandwidth into high data processing rates, an iSSD adds an embedded processor and a stream processor into an FMC (flash memory controller) processing unit on each flash memory channel [4]. Similarly, Biscuit incorporates a hardware pattern matcher on each flash memory channel of an SSD to rapidly scan large datasets on-the-fly [9]. On the other hand, Active Flash employs many compute nodes to achieve a high performance with each node being an SSD with one ARM core as its data processing unit [21].

However, they share two limitations. First, a typical embedded CPU is a 32-bit RISC processor clocked at 200 to 800 MHz [4]. It is not optimized to carry out computations, and thus, running compute-intensive algorithms on it incurs extra instructions, which creates a new performance bottleneck [21]. Second, there is a data transfer bottleneck between an on-device DRAM and an embedded processor [4]. To avoid these limitations, Minerva uses FPGA-based storage controllers as data processing units for a PCM (phase change memory)-based SSD [7]. Still, it adopts the "always running in full gear" strategy, which wastes energy for some applications with a low data processing complexity. .

## III. The RISP Framework

The architecture of RISP is illustrated in the Fig. 1 a. The difference between a conventional SSD and a RISP-based SSD is that the latter has a reconfigurable unit (RU), which



(a) The architecture and memory model of RISP

(b) The implementation of K-Means in RU

Figure 1: The architecture of RISP.

is located between an array of NVM chips and embedded CPU(s). The RU is composed of three components: a processing cell and an NVM controller (hereafter, NVMC) on each channel as well as a public processing cell shared by all channels. A processing cell processes data on a channel by running the algorithm of a particular application. Similar to a conventional SSD, an NVMC here executes basic operations like read/write on the NVM array. It takes commands issued by a processing cell on the same channel to either fetch data from or write data to the NVM array. A processing cell, an NVMC, and NVM chips on the same channel are combined together as a basic data processing unit called a RISP channel (see Fig. 1a). Although RISP channels can work independently from each other, many applications require a further processing of results generated from each RISP channel or data exchange between different channels. For example, in K-Means (See Fig. 1b) intermediate results from all RISP channels after an iteration need to be aggregated and processed to get the new cluster centers before next iteration can be launched [17]. Therefore, RISP incorporates a public processing cell, which acts as a ”coordinator” to conduct a further processing of intermediate results from RISP channels (e.g., an aggregation of these results) or serve as a shared memory providing communications among RISP channels. Eventually, the public processing cell delivers the final results of data processing to the on-device DRAM, which serves as a buffer. The final results are then transferred to the host through DMA (direct memory access). Since RISP only needs to output the results whose size is usually much smaller than that of the raw data, the time and energy used to transfer raw data to the host can be largely saved. Note that RU can work in two modes: regular mode and ISP mode. In the regular mode, it works as a data path between on-device DRAM and NVM and there is no data processing logic in RU. In this mode, RISP is just like a regular SSD without any ISP capability. In the ISP mode, RU is in charge of in-storage data processing as described above.

The three components of RU are all implemented on FPGA fabric. Fig. 1b manifests a reconfigurable public processing cell instance (blue box) and a reconfigurable processing cell instance (red box) for K-Means [17]. While an FTL (flash translation layer) module runs on the embedded CPU, an application (e.g., K-Means) is executed on the RU. All RISP channels are reconfigurable hardware resources. Fig. 1a shows how data are processed in six steps. When a request of an application arrives, the host CPU collects information including identification code, the address of data, and other parameters. The identification code enables the embedded CPU in RISP to know which application image to select to configure the RU. Next, the host CPU packages all the information into a request, which is then sent to RISP through the host interface (Step 1). After the embedded CPU receives and decodes the request, it launches the configuration of the RU according to the identification code. Once the configuration is done, the embedded CPU initializes the RU by writing the application's parameters and data address to the RU. When the initialization is completed, the embedded CPU triggers data processing in RU (Step 2). Now, all RISP channels work concurrently to process raw data fetched from NVM chips. The public processing cell works as a coordinator of all RISP channels as we explained before (Step 3). When the RU is ready to output the final results, these data will be aggregated to on-device DRAM through the public processing cell (Step 4). After the results are ready in the on-device DRAM, the embedded CPU informs the host CPU through an interrupt event. Next, it waits for the response of the host-CPU (Step 5). After the host CPU receives a data-ready interrupt from RISP, it reads the results from the on-device DRAM in the RISP (Step 6).

To maintain memory consistency and smooth data transfer between host and RISP, RISP employs a memory model (see Fig. 1a), which divides the entire memory system into four parts: host memory, global memory, local memory, and private memory. The host memory is the DRAM-based main memory of the host and it can only be accessed by the host CPUs. Global memory is the on-device DRAM of an SSD, which can be accessed by both the host CPUs and RISP. However, at a given time only one of them can visit it. Local memory is the block RAM in the FPGA fabric and it can only be accessed by RISP. It is used to share data between different RISP channels. The private memory is provided by the registers in the FPGA fabric and it is accessible to a processing cell only.

An application can use user-space I/O operations to access data on an external storage device so that the OS and file system can be bypassed [2]. This function makes it possible for an application to send its requests directly to the embedded CPU in RISP. After the configuration of an algorithm in RU is done, RISP can deliver a stable acceleration for it. The algorithm accelerated in RU normally generates a small size configuration file (i.e., bitstream). For example, none of the three applications (i.e., K-Means, Binarization, Sobel) has a configuration file larger than 4 MB. It will take around 10 ms for the configuration of a 4 MB bitstream using a 32-bit-wide slave mode at 100 Mhz CCLK (Configuration Clock) [25]. Thus, the CPU time overhead caused by a reconfiguration is trivial compared to the data processing time. Besides, for an application with a less than 4 MB bitstream, the power of an FPGA dynamic partial reconfiguration does not exceed 400 mW [16]. The energy overhead of a reconfiguration can be estimated by 10 ms $\times$ 400 mW = 4 mJ, which is pretty tiny compared to the energy consumed by data processing. Therefore, the overhead of CPU time and energy consumption caused by a reconfiguration of RISP can be ignored.

## IV. Models and Validations

### A. Performance and energy models

In this section, we will develop performance and energy consumption models for each component of Baseline and RISP. Next, the models will be validated. In Section V, we will leverage these models to evaluate the benefits of RISP. Since performance is mainly determined by data transfer time and data analysis time, we calculate these two metrics for each stage of Baseline and RISP so that we can obtain their overall execution times.

For Baseline, data transfer time from NVM to on-device DRAM is

$$t_{ch2DRAM} = \frac{D}{n_{ch} * b_{ch}}, \tag{1}$$

where $D$ is the size of raw data that have been evenly split among all channels. $n_{ch}$ and $b_{ch}$ are number of NVM channels and read bandwidth of each channel, respectively. Data transfer time from on-device DRAM to Host can be calculated as

$$t_{DRAM2host} = \frac{D}{b_{hostIO}}, \tag{2}$$

where $b_{hostIO}$ is the bandwidth of the host interface. Since the two data transfer stages are organized in a pipelined manner, the overall data transfer time is determined by

$$t_{totalXsfer} = max\left\{t_{ch2DRAM}, t_{DRAM2host}\right\}. \tag{3}$$

Assume that a host has $n_{cpu}$ CPUs and each needs to spend $CPB$ (cycles per byte) cycles to process one byte of data. Hence, data analysis time on the host can be obtained by

$$t_{hostComp} = \frac{CPB * D}{f_{hostcpu} * n_{cpu}}, \tag{4}$$

where $f_{hostcpu}$ is the frequency of each CPU. Thus, the total data processing time for Baseline is

$$T_{Baseline} = t_{totalXsfer} + t_{hostComp}. \tag{5}$$

For RISP, data transfer time from NVM to RU is denoted by $t_{ch2RU}$, which is equal to $t_{ch2DRAM}$ in Baseline. Unlike a host CPU, FPGA provides massive pipelining and parallelism. Hence, data analysis time in RU is

$$t_{RUComp} = \frac{D/(\alpha * n_{ch}) + n_{delay}}{f_{RU}}, \tag{6}$$

where $\alpha$ is the number of bytes that can be fed into a RISP channel per clock cycle, $f_{RU}$ is FPGA running frequency, and $n_{delay}$ is the time delay between the input data arriving RU and the output data (i.e., results) starting to leave RU. Note that $n_{delay}$ is calculated in clock cycles. After data analysis is completed in RU, the public processing cell aggregates all results and then writes them to the on-device DRAM. The time taken by this step is decided by

$$t_{RU2DRAM} = \frac{D}{\beta * b_{DRAM}}, \tag{7}$$

where $\beta$ is a reduction factor, which is the ratio of raw data size to the size of results. Its value depends on the feature of an application. $b_{DRAM}$ is the bandwidth of on-device DRAM. Since data processing through these steps in FPGA could be easily organized in a pipelined manner, the total execution time can be obtained by

$$T_{RISP} = max\{t_{ch2RU}, t_{RUComp}, t_{RU2DRAM}\}. \tag{8}$$

Now, we estimate energy consumption for each component of Baseline and RISP. Since at any given time a component could be in one of the three statuses (i.e., active, idle, and power-off), total energy consumption can be computed by

$$E_{sys} = \sum_{i \in sys} (P_{i\_active} * t_{i\_active} + P_{i\_idle} * t_{i\_idle}), \tag{9}$$

where $P_{i\_active}$, $P_{i\_idle}$, $t_{i\_active}$, $t_{i\_idle}$ are power and duration of each component in either an active or an idle status. For Baseline, $sys$ ={NVM, on-device DRAM, SSD controller, host interface, host DRAM, and host chipset}. For RISP, $sys$ ={NVM, RU, on-device DRAM, SSD chipset}.

In Baseline, the SSD controller includes the embedded CPU(s) and SSD chipset. Assume that energy consumption for each CPU cycle is $EPC$ (energy per cycle) and it needs total $n_{cycle}$ cycles from all $n_{cpu}$ CPUs to finish data analysis for one application. So, energy consumption of host CPUs is $E_{hostCPU}$, which is the product of $EPC$ and $n_{cycle}$. The energy consumption of Baseline can be obtained by

$$E_{Baseline} = E_{sys} + E_{hostCPU} \tag{10}$$

For NVM, on-device DRAM, and SSD chipset in RISP, we use the same methods as Baseline to estimate their energy consumption. As for RU, its energy consumption is mainly decided by the utilization of FPGA and we use an FPGA design tool Vivado [23] to estimate the energy consumption of RU. Table III summarizes all energy-related parameters that are used in our experiments.

## B. Model validations

To examine the impact of data processing complexity of an application on performance and energy gains of RISP, three applications with distinct CPB values (see Table I) are selected to validate the models. We run the three applications on a 3.1 GHz Intel Core i5 iMac computer to validate the performance model of Baseline. The parameters used in the Baseline performance model can be found in the Table II. We demonstrate how to obtain the $CPB$ for Eq. (4) for each application. We introduce three parameters ($CPI$, $IPB$ and $CPUfrequencyratio$) to calculate $CPB$ for each application. These parameters are provided by Vtune [19], a performance analysis tool from Intel. $CPI$ (cycles per instructions) is closely related to machine setting and a higher $CPI$ value implies a higher latency. $IPB$ (instruction per byte) indicates the computational complexity of an application and a higher $IPB$ value means more compute-intensive. $CPUfrequencyratio$ is the ratio between the actual and the nominal CPU frequencies. A value above 1.0 indicates that the CPU is operating in a turbo boost mode. $CPB$ is calculated by: $CPB$=$CPI$*$IPB$*$CPUfrequencyratio$. Therefore, the CPB value is 72.3757, 16.4822, and 106.5680 for K-Means, Binarization, and Sobel, respectively.

Table I: Applications

| Applications | CPI | IPB | CPU frequancy ratio |
|---|---|---|---|
| K-Means [17] | 0.502 | 131.5462 | 1.096 |
| Binarization [18] | 0.601 | 24.8411 | 1.104 |
| Sobel filter [18] | 0.438 | 224.2452 | 1.085 |

Fig. 2 shows the real execution time measured from the iMac computer and the execution time calculated by Eq. 5. A performance gap is the value of an execution time calculated by an equation (i.e., execution time from a model) subtracted by a real execution time. A performance error is defined as a ratio between a performance gap and a real execution time. We found that there is a high correlation between a performance error and the size of the data set. Fig. 2 shows that K-means yields a performance error of 9.39% as it analyzes a small data set. On the other hand, Binarization has a very small performance error of 0.66% because it processes a large data set (see Table II). The average performance error of the three applications is 5.68%, which is a reasonable value. The implication is that the Baseline performance model is valid.

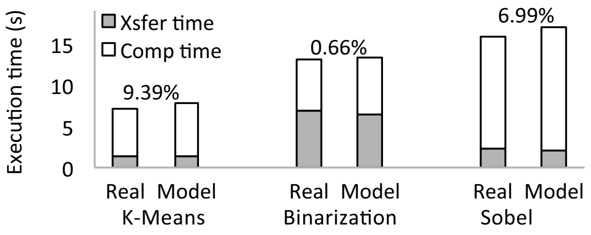


Figure 2: Baseline performance model validation.

We leverage a Xilinx Virtex-7 FPGA platform [24] to validate the performance model of RISP. The three applications are implemented in Verilog to obtain their data processing parameters (e.g., $\alpha$, $\beta$, and $n_{delay}$ in Table II) needed for the RISP performance model. We use the on-chip RAM to emulate the NVM and on-device DRAM. We found that the implementation precision of all three applications on the FPGA platform is in one clock cycle. Therefore, the parameters used in the RISP performance model (see Table II), which are extracted from the implementations of the three applications on the FPGA platform, are accurate. The energy consumption models of Baseline and RISP adopt the same energy modeling paradigm proposed by [5], which has proved to have a 8% error from real SSDs. In this paper, we directly use these validated energy consumption models.

## V. Evaluation

In this section, we first evaluate RISP by comparing it with Baseline in terms of performance and energy consumption. Next, we present the reconfigurability of RISP. Table II provides the parameters of the hardware platform and the three applications. K-Means is a method of vector quantization that is popular for cluster analysis in data mining [17]. Binarization is an algorithm used in image processing to convert each pixel of an image into one bit (i.e., '1' or '0') [18]. Sobel is used in image processing and computer vision, particularly within edge detection algorithms where it creates an image emphasising edges [18]. We select the three applications because they are widely used algorithms in big data analysis and their data processing complexities (i.e., CPB) are quite different (see Table II). All experimental results are obtained by plugging the parameters shown in

Table II, Table III, and Table IV into the validated models presented in Section IV.

## A. Performance evaluation

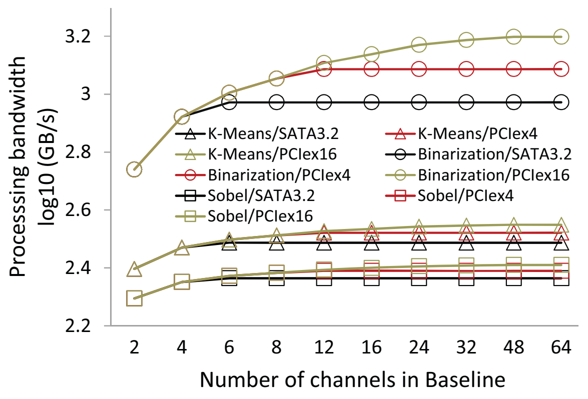


Figure 3: Impact of host I/O bandwidth on Baseline.

We first examine how the host I/O bandwidth and number of NVM channels impact the performance of Baseline. Next, we measure the sensitivity of RISP performance to various resources (e.g., the number of NVM channels, FPGA clock frequency, etc.). Finally, performance comparisons between Baseline and RISP are provided.

To understand the impact of host I/O bandwidth on the performance of Baseline, we utilize three I/O interfaces (i.e., SATA3.2, PCIex4, and PCIx16) to test their impact on Baseline when the number of channels varies from 2 to 64. The results are shown in Fig. 3. The general trend is that the processing bandwidth of Baseline increases when the number of channels goes up. However, after a certain number of channels (e.g., 12 for Binarization with a PCIe-4G interface) Baseline processing bandwidth stops increasing. The reason is that each I/O interface has its own data transfer upper bound (e.g., SATA3.2 is 1.97 GB/s and PCIex4 is 3.938 GB/s [4]). After the internal bandwidth reaches the data transfer upper bound of a host I/O interface, further increasing NVM channels cannot enlarge Baseline processing bandwidth as at this point the host I/O bandwidth becomes a bottleneck.

Fig. 4 also exhibits the same trend. In this figure, SATA3.2 is used in all scenarios. The impact of the host I/O bandwidth bottleneck on performance can be easily observed in Fig. 4a. When the number of channels is smaller than six, the performance of the three applications all improves with an increased number of NVM channels. However, after that none of them can noticeably improve performance. From Fig. 4b we can see that when the number of channels is smaller than six the external bandwidth is decided by the internal bandwidth. However, when it is more than six, the external bandwidth is immediately limited by the upper bound of the host I/O interface bandwidth, which is 1.97 GB/s.

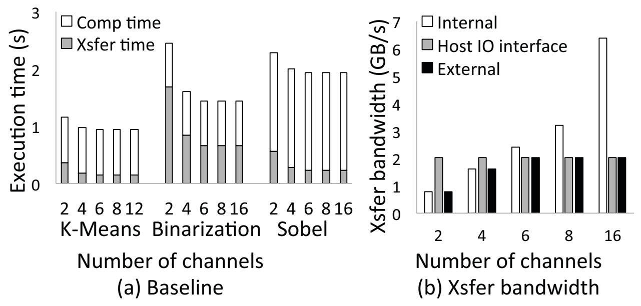


Figure 4: Impact of host I/O and internal bandwidth.

Another observation is that Binarization is more sensitive to the host I/O bandwidth. From Fig. 3 we can see that using different types of host I/O interfaces Baseline can deliver processing bandwidth ranging from 940 MB/s to 1,580 MB/s for Binarization. However, for K-Means and Sobel, it can only offer bandwidth in the ranges of 300-350 MB/s and 230-260 MB/s, respectively. Further, the variances of processing bandwidth for Binarization, K-Means, and Soble are 640 MB/s, 50 MB/s, 30 MB/s, respectively. This is because compared with K-means and Sobel who have a higher *CPB* value, Binarization is an I/O-bound application in which data transfer time is a major part of its execution time. Fig. 4a shows that the data transfer time for Binarization accounts for 50% of its execution time even when 16 NVM channels are used. Thus, increasing data transfer bandwidth between host and SSD can improve the performance of Binarization more obviously than the other two applications.

Next, we examine the impact of the number of NVM channels and FPGA clock frequency on the performance of RISP in Fig. 5. In particular, we measure the performance of the three applications running on RISP at two different FPGA clock frequencies: 200 MHz (Fig. 5a) and 500 MHz (Fig. 5b). Note that this clock only affects data analysis rate, which determines the computing time (i.e., "Comp time" in Fig. 5) of an application running on RISP. As we explained in Section IV, RISP fully leverages pipelining for both data transfer and data analysis. Take Binarization with four RISP channels as an example (see Fig. 5a), its

Table II: System and application parameters [4][6]

| | | | |
|---|---|---|---|
| $n_{ch}$ | 4-64 | $f_{RU}$ | 100-500 MHz |
| $b_{ch}$ | 400 MB/s | $f_{hostCPU}$ | 3.10 GHz |
| $b_{DRAM}$ | 15 GB/s | $n_{CPU}$ | 8 |
| $b_{hostIO}$ | SATA3.2 (1.97 GB/s), PCIex4 (3.938 GB/s) PCIex16 (15.754 GB/s) | | |

| K-Means [17], Binarization [18], Sobel [18] | | | |
|---|---|---|---|
| $D$ | 0.3, 1.4, 0.5 (GB) | $\alpha$ | 9, 1, 1 (Byte) |
| $\beta$ | 3.6466, 3, 1 | $n_{delay}$ | 15, 2, 4 (clock cycle) |
| $CPB$ | 72.3757, 16.4822, 106.5680 (for CPU) | | |

data analysis takes a much longer time (grey bar) than its data transfer (white bar) does. Thus, its execution time (i.e., ”Exec time” in black bar) is only decided by its data analysis time (i.e., ”Comp time”). Several observations can be made. First, we notice that when the number of RISP channels is increased both data transfer time and data processing time decrease, which is quite different from Baseline where only data transfer time declines (see Fig. 4a). The explanation of the difference is that each RISP channel consists of a processing cell and an NVM channel. Thus, bandwidths of data transfer and data analysis are both increased with an increased number of RISP channels. Secondly, we can see that the performance of the three applications running on RISP continuously improves with the increment of channels (see Fig. 5). However, these performance improvements almost do not exist in Baseline (see Fig. 4a). In fact, after the number of channels exceeds six, there is no further execution time decrease in Baseline because the data transfer bandwidth hits the ceiling of the host I/O interface bandwidth. The conclusion is that RISP has a good scalability. The last observation is that when computing time determines the execution time of an application, optimization of RU (e.g., increasing the FPGA clock frequency) helps improve the application’s performance. For instance, the execution time of Binarization with four RISP channels decreases by 52.5% when the FPGA clock frequency is increased from 200 MHz to 500 MHz. However, when execution time of an application is decided by its data transfer time (see Binarization in Fig. 5b), the NVM channel interface becomes a bottleneck. So, further increasing FPGA clock frequency would incur more energy consumption without any performance improvement.

Finally, we compare the performance of RISP and Baseline using the three applications with each processing a particular data set (see Table II). Fig. 6a shows performance improvements of the three applications compared with Baseline when the FPGA clock frequency is set to 100 MHz. The general trend is that all applications continuously improve their performance when the number of channels goes up. When the number of channels is 16 (i.e., a common case for modern SSDs), Sobel, K-Means, and Binarization reduce execution time by a factor of 6.6, 3.9, and 1.6, respectively. Among the three applications, Binarization always requires more channels in order to achieve a performance improvement similar to that of K-Means and Sobel (see Fig. 6a). This is because it is an I/O-bound application, and thus, adding more channels can reduce its execution time in Baseline more than the other two applications. Hence, its performance improvement is reduced. The three applications follow the order of Sobel $>$ K-Means $>$ Binarization in terms of CPB. Since an application with a larger CPB can receive a higher performance improvement due to an increase of channels, Sobel always achieves the highest performance improvements. We also test the performance improvements of Binarization when the FPGA clock frequency varies from 100 MHz to 500 MHz (see Fig. 6b). Clearly, a higher FPGA clock frequency always leads to a better performance for Binarization.

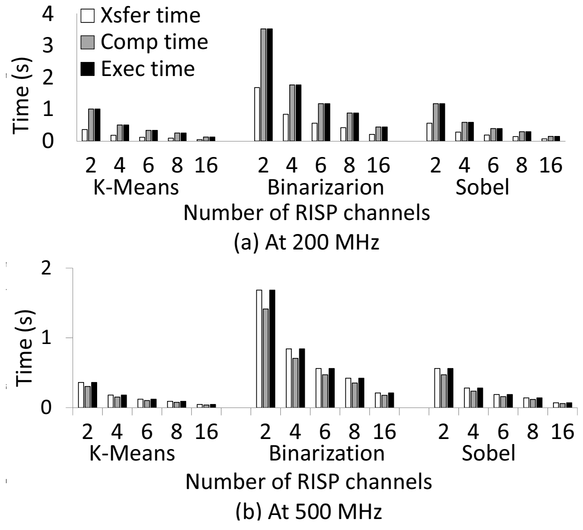


Figure 5: Performance of Binarization on RISP.

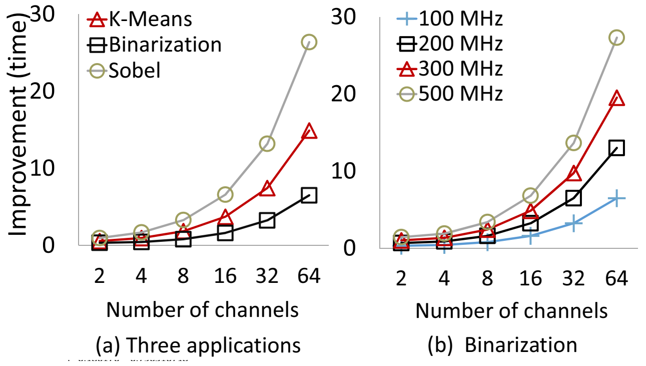


Figure 6: Performance improvements.

## B. Energy consumption evaluation

We evaluate how much energy can be saved if the three applications are running on RISP instead of Baseline. We also measure energy-saving due to the reconfigurability of RISP. Table.III contains all energy-related parameters.

An energy consumption comparison between Baseline and RISP without enabling RISP’s reconfigurability is examined first. Fig. 7a shows the breakdowns of energy consumption of the three applications in Baseline. The energy consumption of each application in Baseline is a function of the $D$ and $CPB$. Data analysis with a larger data set and higher $CPB$ costs more energy. We can see that energy consumed by the

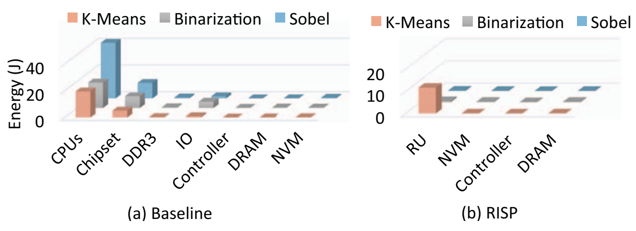


Figure 7: Energy consumption.

CPU(s) takes the largest portions, which are 74.37%, 56.5%, and 74.28% for K-Means, Binarization, and Sobel, respectively. Due to the great differences between CPU and FPGA in computational architecture, the computational complexity and energy consumption of an application running on a host CPU are quite different from running on FPGA.

Fig. 7b shows the breakdowns of the energy consumption of the three applications when they are executed on RISP. Table II and Table IV demonstrate the differences in terms of computational complexity for the three applications running on Baseline and RISP. An application's FPGA utilization indicates its computational complexity on RISP. The FPGA utilization of an application is mainly determined by the number of slices, LUTs (lookup table), and DSP48E (digital signal processing 48E) that have been used by the application. All values shown in Table IV are obtained by running the Vivado [23] tool for the three applications. By comparing these two tables, we can see that Soble has the highest CPB value when it runs on Baseline. However, K-Means has the highest computational complexity when it runs on RISP because its FPGA utilization is the highest (see Table IV). Energy consumption of an application is also correlated to FPGA utilization when it is running on RISP. Thus, K-Means has the highest power as shown in Table IV.

For K-Means, RU consumes the most energy. For Binarization and Sobel, their energy consumption taken by RU only accounts for around 50% of total energy consumption. By comparing Fig. 7a and Fig. 7b, we find that compared to Baseline processing the same data set with 16 channels RISP could reduce energy consumption by 2.2× , 32×, and 161× for K-Means, Binarization, and Sobel respectively. We can find the values of $CPB$ from Table II and derive FPGA utilization from Table IV. From these two tables, we notice that Sobel has the highest $CPB$ value but lowest FPGA utilization, which implies that more energy consumption can be saved by using RISP. That is why Sobel enjoys the highest energy efficiency. This also explains why K-Means receives the lowest energy consumption reduction (i.e., 2.2x).

## C. Reconfigurablity of RISP

In this section, we demonstrate how RISP can judiciously select an appropriate number of channels to serve an application under a constraint of power or a bandwidth bottleneck of the host interface. Based on the performance model for RISP (see Eq. (8)), we can see that for each application there is a correlation between the number of channels used and data processing (including both data transfer and data analysis) bandwidth, which is equal to $D$ (i.e., the size of raw data) divided by $T_{RISP}$ (see Eq. (8)). Similarly, based on the energy consumption model (see Eq. (9)) and Table III, for each application running on RISP there is a correlation between power of RISP and the number of channels used. The power of RISP for an application is defined as the total energy consumed by the application divided by its execution time $T_{RISP}$. Hence, for a particular application running on RISP, the following two general functions can be used to describe the two correlations.

$$Power = f_{power}(n_{ch}). \tag{11}$$

$$Processing\_bandwidth = f_{pb}(n_{ch}). \tag{12}$$

When the host CPU packages all information of an application into a request (see Step 1 in Fig. 1a), the two functions are also included. The embedded processor then configures RU (see Step 2 in Fig. 1) based on the two functions to decide an appropriate number of channels to use in order to satisfy either a power constraint or a host interface bandwidth bottleneck. To discover two specific functions for a particular application to replace the two general functions, we first calculate data processing bandwidth in terms of GB/second and power in terms of Watt for the three applications running on RISP at FPGA clock frequency of 100 MHz based on the performance model (i.e., Eq. (8)) and energy consumption model (i.e., Eq. (9)). Fig. 8 shows the results of these two metrics when the number of channels varies from 4 to 64. For each application, from Fig. 8 one can see that there exist two hidden curves: one is formed by the values of data processing bandwidth (i.e., the heights of data processing bandwidth bars in Fig. 8a) and the other is created by the values of power (i.e., the heights of power bars in Fig. 8b).

Therefore, two linear regression functions can be obtained by curve-fitting. For example, the two linear regression functions for Sobel are presented in Eq. (13) and Eq.

Table III: Energy parameters [8][11]

| | | | |
|---|---|---|---|
| $P_{NVM\_active}$ | 66 mW | $P_{NVM\_idle}$ | 16.5 mW |
| $P_{DRAM\_active}$ | 500 mW | $P_{DRAM\_idle}$ | 350 mW |
| $P_{controller\_active}$ | 150 mW | $P_{controller\_idle}$ | 75 mW |
| $P_{DDR}$ | 360.4 mW | $P_{hostChipset}$ | 6 W |
| $EPC$ | 0.9 nJ/Ins | $P_{RUidle}$ | 24 mw |
| $P_{RUactive}$ | See Table IV | | |
| $P_{IO\_active}$ | SATA: 7 W, PCIex4: 35 W | | |

Table IV: FPGA utilization and power for 16 channels

| Applications | Slices | LUTs | DSP48E | Power (w) |
|---|---|---|---|---|
| $K-Means$ | 23728 | 76368 | 0 | 30.8864 |
| $Binarization$ | 144 | 96 | 48 | 1.20016 |
| $Sobel$ | 144 | 256 | 0 | 1.20016 |

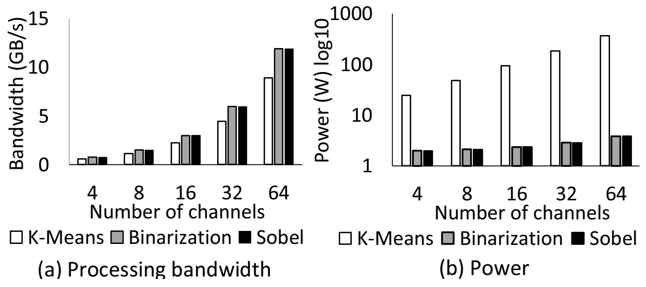


Figure 8: Processing bandwidth and power.

(14), respectively. Now, we use Sobel as an example to illustrate how a 64-channel RISP-based SSD leverages its reconfigurability to meet the power constraint. Base on our experimental results, for Sobel the power of RISP varies from 1.9 to 3.8 Watts when RU is running at 100 MHz. Since the power budget of a typical modern SSD is normally 3 Watts [20], RISP will adjust the number of channels so that the power constraint will not be violated. From Eq. (13) we can derive that the number of channels should be no more than 36 in order not to exceed the 3-Watt constraint. After 36 is plugged into Eq. (14), we find that the maximal data processing bandwidth of RISP is 6.9 GB/s. Thus, RISP only activates 36 channels while powering off all rest 28 channels to satisfy the power constraint.

$$Power = 1.858 + 0.032 * n_{ch} \tag{13}$$

$$Processing\_bandwidth = 0.19 * n_{ch} \tag{14}$$

If power is not a concern for an application, is it a good idea to use all RISP channels to run an application? We find that the answer is no. The reason is that if the host interface bandwidth becomes a performance bottleneck using all RISP channels for a low data processing complexity application (e.g., Sobel) could waste energy without any performance gains. Assume that RISP has 64 channels and the host interface in Fig. 1a is SATA3.2 whose bandwidth is 1.97 GB/s (see Table II). With 64 RISP channels, based on Eq. (14) RISP could deliver a data processing bandwidth of 12.16 GB/s for Sobel, which is denoted as $PB$. Then the output bandwidth of the results should be $PB/\beta$ = 12.16 GB/s ($\beta$=1 for Sobel, see Table II), which is higher than the host interface bandwidth (i.e., 1.97 GB/s). Consequently, some RISP channels will be in an idle status, which simply wastes energy without any performance improvement. In this scenario, RISP only needs to provide a data processing bandwidth that matches the host interface bandwidth. Thus, based on Eq. (14), it only needs to activate 10 out of 64 channels for Sobel to reach the host interface bandwidth bottleneck. Therefore, RISP only configures the 10 processing cells on the 10 channels. In other words, only the FPGA fabric resources (i.e., LUTs, DSP, etc.) of the 10 channels are deployed. Under this situation, the 10 processing cells

Table V: The impact of reconfigurability on energy

| | *Deactive RF* | | *Active RF* | | *Reduction* |
|---|---|---|---|---|---|
| | $n_{ch}$ | $E(J)$ | $n_{ch}$ | $E(J)$ | |
| *K-Means* | 64 | 11.7001 | 51 | 11.6801 | 0.17% |
| *Binarization* | 64 | 0.7183 | 31 | 0.4274 | 40.50% |
| *Sobel* | 64 | 0.624 | 10 | 0.1425 | 77.17% |

need to process all data from the 64 channels and the public processing cell is in charge of switching a group of channels to a particular processing cell. For example, processing cell on RISP channel 0 processes data from channel 0 to channel 5, processing cell on RISP channel 6 takes care of data from channel 6 to 11, and so forth.

Based on Eq. (13) and Eq. (9), RISP provides an additional 77.2% energy saving for Sobel due to its reconfigurability. Under the same host interface bandwidth bottleneck and settings, the reconfigurability of RISP could save additional energy by 40.5% (31 out of 64 channels) for Binarization and 0.2% (51 out of 64 channels) for K-Means. Table V summarizes energy savings due to the reconfigurability of RISP. RF in Table V stands for reconfigurability.

## VI. Conclusion

Big data analysis increasingly demands a high performance data processing architecture. Although existing ISP techniques can substantially improve data processing performance, they unnecessarily use all NVM channels for all applications all the time. In this paper, we propose a reconfigurable ISP technique called RISP, which can save energy without any performance degradation due to its reconfigurability. In the future work, we plan to investigate how to utilize features of emerging NVM technologies like byte addressability to further improve ISP efficiency.

## VII. Acknowledgments

This work is sponsored by the U.S. National Science Foundation under grant CNS-1320738.

## References

[1] A. Acharya, M. Uysal, and J. Saltz. Active disks: Programming model, algorithms and evaluation. *ACM SIGPLAN Notices*, 33(11):81–91, 1998.

[2] A. M. Caulfield, T. I. Mollov, L. A. Eisner, A. De, J. Coburn, and S. Swanson. Providing safe, user space access to fast, solid state disks. *ACM SIGARCH Computer Architecture News*, 40(1):387–400, 2012.

[3] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach. Accelerating compute-intensive applications with gpus and fpgas. In *Application Specific Processors, 2008. SASP 2008. Symposium on*, pages 101–107. IEEE, 2008.

[4] S. Cho, C. Park, H. Oh, S. Kim, Y. Yi, and G. R. Ganger. Active disk meets flash: A case for intelligent

ssds. In *Proceedings of the 27th international ACM conference on International conference on supercomputing*, pages 91–102. ACM, 2013.

[5] S. Cho, C. Park, Y. Won, S. Kang, J. Cha, S. Yoon, and J. Choi. Design tradeoffs of ssds: From energy consumptions perspective. *ACM Transactions on Storage (TOS)*, 11(2):8, 2015.

[6] K. Czechowski, V. W. Lee, E. Grochowski, R. Ronen, R. Singhal, R. Vuduc, and P. Dubey. Improving the energy efficiency of big cores. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 493–504. IEEE Press, 2014.

[7] A. De, M. Gokhale, R. Gupta, and S. Swanson. Minerva: Accelerating data analysis in next-generation ssds. In *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*, pages 9–16. IEEE, 2013.

[8] E. Grochowski and M. Annavaram. Energy per instruction trends in intel microprocessors. *Technology@ Intel Magazine*, 4(3):1–8, 2006.

[9] B. Gu, A. S. Yoon, D.-H. Bae, I. Jo, J. Lee, J. Yoon, J.-U. Kang, M. Kwon, C. Yoon, S. Cho, et al. Biscuit: A framework for near-data processing of big data workloads. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 153–165. IEEE, 2016.

[10] G. Hong et al. Analysis of peak current consumption for large-scale, parallel flash memory. In *Workshop for Operating System Support for Non-Volatile RAM (NVRAMOS 2011 Spring)(Jeju, Korea, April 2011)*, 2011.

[11] Intel. Intel ssd data center tool. https://downloadcenter.intel.com/download/23931/Intel-Solid-State-Drive-Data-Center-Tool, 2016.

[12] Z. István, D. Sidler, and G. Alonso. Caribou: intelligent distributed storage. *Proceedings of the VLDB Endowment*, 10(11):1202–1213, 2017.

[13] I. Jo, D.-H. Bae, A. S. Yoon, J.-U. Kang, S. Cho, D. D. Lee, and J. Jeong. Yoursql: a high-performance database system leveraging in-storage computing. *Proceedings of the VLDB Endowment*, 9(12):924–935, 2016.

[14] S.-W. Jun, M. Liu, S. Lee, J. Hicks, J. Ankcorn, M. King, S. Xu, et al. Bluedbm: An appliance for big data analytics. In *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*, pages 1–13. IEEE, 2015.

[15] G. Koo, K. K. Matam, H. Narra, J. Li, H.-W. Tseng, S. Swanson, M. Annavaram, et al. Summarizer: trading communication with computing near storage. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 219–231. ACM, 2017.

[16] A. Nafkha and Y. Louet. Accurate measurement of power consumption overhead during fpga dynamic partial reconfiguration. In *Wireless Communication Systems (ISWCS), 2016 International Symposium on*, pages 586–591. IEEE, 2016.

[17] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary. Minebench: A benchmark suite for data mining workloads. In *Workload Characterization, 2006 IEEE International Symposium on*, pages 182–188. IEEE, 2006.

[18] I. Pitas. *Digital image processing algorithms and applications*. John Wiley & Sons, 2000.

[19] J. Reinders. *VTune performance analyzer essentials*, volume 14. Intel Press, 2005.

[20] Samsung. Solid state drives. http://www.samsung.com/us/computing/memory-storage/solid-state-drives/s/_/n-10+11+hv22y+zq29m/, 2017.

[21] D. Tiwari, S. Boboila, S. S. Vazhkudai, Y. Kim, X. Ma, P. Desnoyers, and Y. Solihin. Active flash: towards energy-efficient, in-situ data analytics on extreme-scale machines. In *FAST*, pages 119–132, 2013.

[22] J. Wang, D. Park, Y.-S. Kee, Y. Papakonstantinou, and S. Swanson. Ssd in-storage computing for list intersection. In *Proceedings of the 12th International Workshop on Data Management on New Hardware*, page 4. ACM, 2016.

[23] Xilinx. https://www.xilinx.com/products/design-tools/vivado.html.

[24] Xilinx. Xilinx virtex-7 fpga vc707 evaluation kit. https://www.xilinx.com/products/boards-and-kits/ek-v7-vc707-g.html#hardware.

[25] Xilinx. 7 series fpgas configuration user guide. https://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf, 2017.