# Exploiting Free Silicon for Energy-Efficient Computing Directly in NAND Flash-based Solid-State Storage Systems

Peng Li
*Seagate Technology*
*Shakopee, MN, 55379*
*benjamin.p.li@seagate.com*

Kevin Gomez
*Seagate Technology*
*Shakopee, MN, 55379*
*kevin.gomez@seagate.com*

David J. Lilja
*University of Minnesota, Twin Cities*
*Minneapolis, MN, 55455*
*lilja@umn.edu*

***Abstract*—Energy consumption is a fundamental issue in today's data centers as data continue growing dramatically. How to process these data in an energy-efficient way becomes more and more important. Prior work had proposed several methods to build an energy-efficient system. The basic idea is to attack the memory wall issue (i.e., the performance gap between CPUs and main memory) by moving computing closer to the data. However, these methods have not been widely adopted due to high cost and limited performance improvements. In this paper, we propose the *storage processing unit* (SPU) which adds computing power into NAND flash memories at standard solid-state drive (SSD) cost. By pre-processing the data using the SPU, the data that needs to be transferred to host CPUs for further processing are significantly reduced. Simulation results show that the SPU-based system can result in at least 100 times lower energy per operation than a conventional system for data-intensive applications.**

***Keywords*-SSD, NAND Flash, Parallel computing, OpenCL.**

## I. Introduction

High-performance computers have been widely used in many fields, such as weather prediction, human genome studies, and so on. Their performance has been driven by endless quests for more and more computing power in such applications. However, as data continue to grow dramatically, energy consumption of those high-performance computers becomes a limiting issue. In fact, more than 50% of today's data centers' budgets go to energy costs [1]. Thus, how to design an energy-efficient computer system becomes more and more important.

A fundamental energy-efficiency issue is the performance gap between CPUs and main memories, i.e., the memory wall [2]. In 1994, Wulf and McKee [3] pointed out the implications of processor and memory performance progressing exponentially but with differing rates (50% per year for processors vs. 7% per year for memory). Most complex operations, such as floating-point multiplications, take only a few CPU clock cycles [4]. However, accessing the main memory can take hundreds of CPU clock cycles [4]. Thus, for those data-intensive applications whose dataset is much larger than the CPU cache size, more than 90% of the energy is consumed by the data transfer instead of computing. If the dataset is even larger than the main memory size, accessing the data in storage devices will consume even more energy and futher extend the processing time [5], [6], [7]. The exponentially increasing gap between the CPUs and the main memories has also led to the end of single thread processor performance progress by 2008. Although multi-core CPUs had been proposed to further improve the system performance through the memory wall, the energy-efficiency was not improved. The IEEE rebooting computing working group has set up a goal to completely rethink computing, from devices to circuits to architecture and software [8].

A well-known solution to the memory wall issue is moving computing closer to the data. For example, Gokhale et al [2] proposed a processor-in-memory (PIM) chip by adding a processor into the main memory for computing. Riedel et al [9] proposed an active disk by using the processor inside the hard disk drive (HDD) for computing. With the evolution of other non-volatile memories (NVMs), such as phase-change memory (PCM) and spin-transfer torque (STT)-RAM, researchers also proposed to use these NVMs as the main memory for data-intensive applications [10] to improve the system energy-efficiency. However, these methods are not widely adopted due to the high cost and the limited performance improvements.

In this paper, we attack the memory wall issue by adding computing power directly into NAND flash memories. Compared to prior work, our approach has the following advantages: (1) Compared to its nearest competing technology (such as the PCM and the STT-RAM), the NAND flash-based solid-state devices (SSDs) have been widely used in the storage market thanks to their low-cost and high-density. To enable the NAND flash memories to scale and stay orders of magnitude cheaper than its nearest competing technology, manufacturers have integrated hardware into a die inside the NAND flash package for error-correcting logic to deal with the reliability issue as the NAND flash cell size continues to shrink. We believe that this is the first time in history that a significant chunk of silicon (more than 1M gates) is called for right on the storage media touching the data stream to deal with the degrading signal-to-noise ratio (SNR). If we add a computing unit into the same die, especially in a pad-limited case (i.e., the die area is driven by the number of I/O pins rather than gate counts), the cost will be negligible or zero. (2) The NAND flash is a block addressable storage device which can tolerate latencies due to sophisticated signal processing. This is not feasible in byte-addressable main memory such as DDR SDRAM or 24 byte addressable PCM since it would significantly impact latency. (3) Compared to the conventional HDD, the NAND flash-based SSDs have high performance and low power consumption. In addition, compared to the active disk approach, which suffers from the single channel of the spinning disk recording head/media system when applied to data intensive compute applications, the NAND flash-based storage devices have multiple independent channels.

We call the proposed architecture a *storage processing unit* (SPU). The basic idea is adding a coprocessor in the pad-limited die inside the NAND flash package to perform parts of the computation required for data-intensive applications. The SPU benefits the system in the following aspects: (1) By moving computing closer to the data, the system saves energy on data transfers without a loss of performance. This is because we can use pipeline techniques to design the coprocessors, loading data from the NAND flash memories and processing these data using the coprocessor simultaneously. (2) By pre-processing the data inside the storage devices, the host CPU will work with a smaller dataset thereby producing better energy-efficiency because cache miss rates are reduced. (3) The coporcessors do not need to be high-performance processors due to the speed limitation of the NAND flash I/O interface. Thus, they can be implemented using low operating power (LOP) technology as defined by the International Technology Roadmap for Semiconductors (ITRS) [11]. These low power coprocessors are more energy-efficient than the high-performance host CPUs [12]. (4) Hardware for error-correcting coding (ECC) and a general purpose processor for data management have already been integrated into the die inside the NAND flash package, and the die area is pad-limited. Thus, the cost for adding a coprocessor into this die is negligible. (5) The SSD's multi-channel architecture is highly scalable. By adding computing power into the NAND flash memories, we can take advantage of parallel computing.

We evaluate the proposed SPU using a facial recognition algorithm [13] and a restricted Boltzmann machine (RBM) algorithm [14]. The facial recognition algorithm is a proxy algorithm for content-based image retrieval algorithms, and the RBM algorithm is a proxy algorithm for many machine learning and data-intensive scientific computing algorithms [15]. Simulation results show that the SPU-based system is at least 100 times more energy-efficient than the conventional system for data-intensive applications. The remainder of this paper is organized as follows. Section II briefly reviews the background of the NAND flash-based SSDs. Section III describes the proposed SPU in detail. Section IV presents the simulation results. Conclusions are drawn in Section V.

## II. Background

Demand for NAND flash-based SSDs has grown dramatically in recent years thanks to their high performance and low power consumption. As shown in Fig. 1, a conventional NAND flash-based SSD normally consists of an SSD controller and multiple NAND flash channels. Each of these channels has multiple NAND flash packages, and each of these NAND flash packages contains multiple NAND flash dies. The NAND flash die has multiple planes, each plane has multiple blocks, and each block has multiple pages which are the basic storage units. The SSD controller communicates with the host and implements a flash translation layer (FTL) to emulate a block device to the host. The functions in the FTL include block management, wear leveling, and garbage collection. In addition, the SSD controller also needs to implement error-correction coding (ECC) due to the reliability issues of the NAND flash memories.

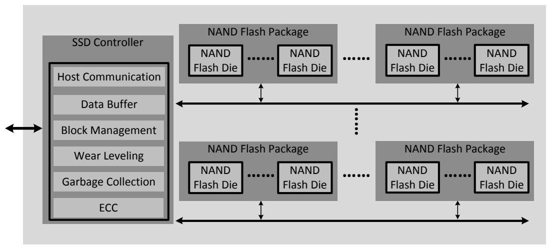

Figure 1. The block diagram of a typical SSD.

To increase the density of the NAND flash memories, manufacturers have applied two main techniques [16]. One is the standard process node and lithography shrinks, making each memory cell and the associated circuitry smaller. The other technique is storing more than one bit per cell. Early NAND devices, the single-level cell (SLC), could store one of two states in a memory cell. Recently, multi-level cell (MLC) and tri-level cell (TLC) techniques have been used to further increase the density. Compared to SLC, which is the original 2 levels per cell and stores 1 bit of information per cell, MLC uses 4 levels and stores 2 bits, and TLC uses 8 levels and stores 3 bits [16]. Although MLC and TLC increase the density, they have come at the cost of introducing more errors. The more bits stored per cell, the more errors are introduced.

To solve the reliability issues of MLC and TLC, more complicated ECC has to be used. For example, the MLC NAND flash memories require more than 16 bits of ECC per 512 bytes [16]. Advanced ECC operations, like low-density parity-check (LDPC) coding, are computationally expensive. Many solutions implement specific hardware to support ECC, since these operations will take many CPU cycles from other important tasks if they are implemented only using the SSD controller [16]. For example, both the Micron ClearNAND and the Toshiba embedded multimedia card (eMMC) have integrated the hardware ECC into a die inside the NAND flash package. In addition, since the die area is pad-limited, manufacturers like Micron and Toshiba also have integrated a general purpose processor into the die to implement parts of the FTL functions, such as block management, to further increase the SSD performance. A block diagram of such an architecture is shown in Fig. 2 (a).

In fact, even with the integrated general purpose processor and the hardware ECC, the die still has available area [16]. Thus, we can integrate more logic units without any additional cost. In this paper, we propose the SPU architecture, which adds computing elements directly into the die and enables the storage devices to support parallel computing. The SPU design will be presented in the next section.

## III. Design of the SPU

Based on the block diagram shown in Fig. 2 (a), it can be seen that a conventional SSD is actually a small computer that contains multiple processors and storage media. Besides the basic read/write operations, we can also use it to perform computing. In addition, the flexibility of the SSD multiple channel architecture provides substantial opportunities to improve the system performance. A block diagram of the proposed SPU is shown in Fig. 2 (b). It can be seen that, besides the hardware ECC and the general purpose processor, the SPU also adds a coprocessor into the pad-limited die inside the NAND flash package. In this section, we first discuss the coprocessor design. Then we introduce a data allocation scheme that fully utilizes the throughput of the multi-level parallelism. Finally, we show how this SPU can be utilized for parallel computing using a standard programming environment such as OpenCL [17].

### A. *Coprocessor Design*

Fig. 3 shows the trend of the NAND flash process node and the SSD controller process node based on the ITRS [11]. For example, the controller process node will be 13nm and the NAND flash

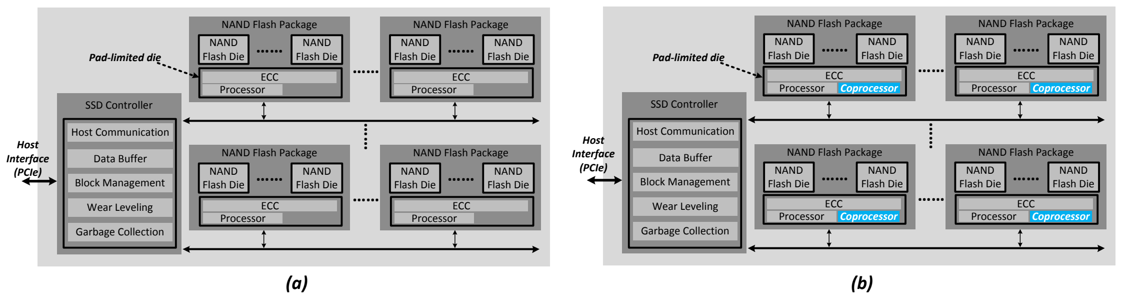

Figure 2. The block diagram of SSD systems with integrated functions in the NAND flash package: (a) a typical SSD with ECC function and a general purpose processor integrated into the NAND flash package; (b) our proposed SPU.

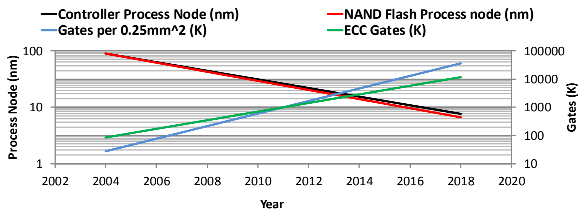

Figure 3. Trends in ECC requirements for NAND flash show that the additional gates provided by Moore scaling outpace the area needed for the hardware ECC in 2011, which enables an exponentially increasing gate count available to provide essentially free computational resources for enhanced operations, especially in a pad-limited case [18].

process node will be 12nm by the year 2015. To maintain the system error rate, the gate count of the hardware ECC needs to be increased when the process node continues shrinking. However, as the transistor size continues shrinking, we also have more available gates besides the ones used for the hardware ECC. As shown in Fig. 3, by the year 2015 the hardware ECC needs 4M gates but the available gate counts per 0.25mm$^2$ will be 8M. Thus, it will be feasible to increase the die utilization by adding computing functionalities without increasing the die area. Compared to the hardware ECC and the general purpose processor, the gate counts required for the coprocessor can be much smaller. This is because the computing kernels of most data-intensive applications are all basic operations, such as multiplication, addition, and comparison. The multiplier, adder, and comparator can be implemented using less than 10K gates. Especially in the pad-limited case, the additional cost is negligible [18].

Furthermore, because the NAND flash package is designed based on the Open NAND Flash interface (ONFi) specification, the speed of the coprocessor should match the NAND flash I/O throughput. For example, based on ONFi specification version 3.1 [19], the NAND flash interface supports transfer rates up to 400MB/s. In this case, the coprocessor can work under 400MHz, and use pipeline techniques for computing to match its I/O throughput. Thus, the coprocessor does not need to be high-performance, and we can use the LOP technology as defined by the ITRS to design the coprocessor [11]. Compared to the high performance (HP) technology defined by the ITRS, the speed of the LOP technology is half as fast, but its dynamic power and leakage power is only 60% and 5% of the HP technology, respectively. Thanks to the LOP technology, a 100K-gate coprocessor consumes only 0.3mW more dynamic power than a 1K-gate coprocessor at 400MHz.

Due to these conditions, we should design the coprocessor using as much available die area as possible, because the additional cost is negligible in terms of both die area and power consumption. Using more gates to design the coprocessor can support more computing kernels. For example, if we use 10K gates to design the coprocessor, it will support basic operations such as multiplication, addition, and comparison. If we can use 100K gates to design the coprocessor, it will also support computing kernels, such as random number generation, which are widely used in Monte Carlo simulations [4]. If we can use 1M gates to design the coprocessor, it will additionally support floating-point computation, which is widely used in science and engineering applications [4].

## *B. Data Allocation*

As we described in Section II, the SSD has multi-level (channel-level, package-level, die-level, and plane-level) parallelism. In fact, Hu et al [20] have shown that the priority order of this multi-level parallelism is different in terms of the I/O throughput. To achieve the highest I/O throughput for the conventional SSD, the optimum priority order should be channel-level, die-level, plane-level, and package-level parallelism.

Since we have both the general purpose processor and the coprocessor inside each NAND flash package, we have multiple control and compute units in each flash channel. To fully utilize the parallel computing capability of the SPU, we design our data allocation scheme based on the one proposed by Hu et al [20] with modifications. We change the priority order to channel-level, package-level, die-level, and plane-level parallelism. The channel-level and package-level parallelism are managed by the SSD controller, while the die-level and the plane-level parallelism is managed by the general purpose processor inside each of the NAND flash packages.

For example, let us consider vector to matrix multiplication, which is one of the computing kernels in the basic linear algebra subprograms (BLAS) widely used in data-intensive applications [4]. We allocate the matrix to the NAND flash memory based on its columns. Since the channel-level parallelism has the highest priority, the columns are first allocated to different channels. Then, when all the channels have been allocated at least one column, the next column will be allocated to a new NAND flash package in the flash channels. A similar principle is applied to the die-level and plane-level parallelism. Table I shows an example of how to allocate a 16-column matrix to a 2-channel, 2-package, 2-die, 2-plane SPU. For example, the 11th column of the matrix is allocated to channel 1, package 2, die 1, and plane 2.

Table I
AN EXAMPLE OF THE PROPOSED DATA ALLOCATION SCHEME.

| Columns | Channel-Package -Die-Plane | Columns | Channel-Package -Die-Plane |
|---|---|---|---|
| 1 | 1-1-1-1 | 2 | 2-1-1-1 |
| 3 | 1-2-1-1 | 4 | 2-2-1-1 |
| 5 | 1-1-2-1 | 6 | 2-1-2-1 |
| 7 | 1-2-2-1 | 8 | 2-2-2-1 |
| 9 | 1-1-1-2 | 10 | 2-1-1-2 |
| 11 | 1-2-1-2 | 12 | 2-2-1-2 |
| 13 | 1-1-2-2 | 14 | 2-1-2-2 |
| 15 | 1-2-2-2 | 16 | 2-2-2-2 |

### C. Parallel Programming Model

Because the SPU has a general purpose processor in each of the NAND flash packages, its firmware can be designed to support parallel programming standards, such as OpenMP, MPI, and OpenCL. In this section, we describe how the SPU can be used as an OpenCL-based computing device, since the OpenCL programming standard has become more popular. It defines four models, which are the platform model, the execution model, the memory model, and the programming model [17]. In this subsection, we discuss how the SPU can be mapped into these models.

The OpenCL platform model defines the interface between the host and the compute devices. Because all of the current OpenCL-based compute devices do not contain storage media, they need a high throughput interface to reduce the data transfer time. In fact, this is the current bottleneck of these compute devices. Thus, they adopt the high throughput PCIe interface. Since the SPU is also a storage device, the data transfer time will not be an issue. Although the SPU can adopt the PCIe interface, it can also use other interfaces such as SATA, SAS, or even USB to reduce cost.

The OpenCL execution model defines the concepts of work group and work item [17]. As shown in Fig. 2 (b), a single flash channel of the SPU can serve as an OpenCL work group, and the coprocessors in the flash channel can serve as the OpenCL work items. If the SPU has $c$ flash channels and each channel has $p$ NAND flash packages, then the SPU will have $c$ work groups and $p$ work items per work group.

The OpenCL memory model defines three different kinds of memories that can be accessed by the compute device—the global/constant memory, the local memory, and the private memory [17]. In the SPU, the SSD controller's main memory can be used as the global/constant memory. The SRAM inside the SSD controller can be divided into multiple groups and used as the local memory of the work group. The SRAM of the general purpose processor inside each NAND flash package can be used as the private memory.

The OpenCL programming model supports both the data parallel programming model and the task parallel programming model. The SPU can use the multiple coprocessors to support the data parallel programming model. In addition, thanks to its internal processor hierarchy (coprocessors and the SSD controller), the SPU can use this processor hierarchy to support the task parallel programming model, i.e., some tasks can be run using the coprocessors and the others can be run using the SSD controller. How to allocate these tasks depends on their data dependencies and the corresponding computing load. For example, the SSD controller is more suitable for the tasks with small datasets and a light computing load.

## IV. SIMULATION RESULTS

In this section, we compare the SPU-based system to a conventional system in terms of processing time and energy consumption. Both systems are implemented using SystemC transaction level modeling. Two applications, the restricted Boltzmann machine (RBM) [14] and facial recognition [13], are used for evaluation. The size of the dataset of the RBM is between the host CPU cache size and the host main memory size, which can be used to study the memory bus as the system bottleneck. The size of the dataset of the facial recognition application is much larger than the host main memory size, which can be used to further study the storage device as the system bottleneck. The computing kernels of these two applications are vector multiplication and addition, which is widely used in data mining and machine learning. The datasets and the computing kernels of the two applications can represent many other applications. For example, the facial recognition algorithm is the proxy algorithm for content-based image retrieval algorithms, and the RBM algorithm is the proxy algorithm for many machine learning and data-intensive scientific computing algorithms [4].

The model of the conventional system consists of the CPU, the main memory (DDR SDRAM), the PCIe interface, and the conventional SSD. To run an application, the conventional system first loads data from the SSD to the main memory via the PCIe interface. If the dataset is smaller than the main memory, the data will be loaded only once. We use different values of CPU cycles per instruction (CPI) to study the memory wall. In addition, we use different numbers of CPU cores to study scalability. If the dataset is larger than the main memory, the CPU will load the data from the SSD multiple times. In this case we implement a direct memory access (DMA) controller between the main memory and the SSD. For example, when the CPU is performing computing on the first sub-dataset which has been loaded into the main memory, the DMA controller is loading the second sub-dataset from the SSD to the main memory. The SSD uses the data allocation scheme proposed by Hu et al [20] to maximize its throughput.

The SPU-based system consists of the CPU, the main memory, the PCIe interface, and the SPU. To run the application, the host CPU issues computing commands to the SPU via the PCIe bus. We assume the data have been allocated based on the scheme introduced in Section III-B, so that the parallel computing capacity of the SPU can be fully utilized. In addition, we assume the coprocessor's CPI is one for the basic operations like multiplication and addition. This can be achieved by using the same clock frequency as the NAND flash bus and pipelining. The SPU sends only the results to the host CPU for further processing.

Table II lists the main parameters used in our models. We assume the gate count of a single CPU core is 200M based on multi-core CPUs from different manufacturers, such as the IBM Power7 8-core CPU and the Intel Xeon E5 8-core CPU[1]. The gate count of the SSD controller is estimated using the ARM A5 embedded general purpose processor [19]. The gate count of the SPU coprocessor is estimated using the basic arithmetic logic units such as the multiplier and the adder [18]. The power of the CPU is calculated based on the high performance (HP) technology defined by ITRS [11]. Its dynamic power is the product of the clock frequency, the gate count, and the power of a single gate

[1]http://en.wikipedia.org/wiki/Transistor_count

defined by the HP technology. Its leakage power is the product of the gate count, the operating voltage, and the leakage current of a single gate defined by the HP technology. The power of the SSD controller and the SPU coprocessor is based on the low operating power (LOP) technology, since their speeds are at most half of the CPU speed. The power of the NAND flash die (including the general purpose processor and the hardware ECC) and the power of the main memory are based on Micron's devices [19].

Table II
THE MAIN PARAMETERS USED IN THE SIMULATION MODEL.

| Parameters | Values |
|---|---|
| **CPU clock frequency** | 2GHz |
| **CPU gate counts per core** | 200M |
| **CPU dynamic/leakage power per core** | 5.04W/0.34W |
| **Main memory dynamic/leakage power** | 0.44W/0.09W |
| **PCIe interface speed** | 24Gb/s |
| **PCIe dynamic/leakage power per GB** | 37.5mW/0 |
| **SSD controller clock frequency** | 1GHz |
| **SSD controller gate counts** | 20M |
| **SSD controller dynamic/leakage power** | 156mW/1.3mW |
| **NAND flash dynamic/leakage power per die** | 40mW/3mW |
| **NAND flash page read to register** | 75us |
| **NAND flash bus speed** | 400MHz (or 2.5ns per byte) |
| **SPU coprocessor clock frequency** | 400MHz |
| **SPU coprocessor gate counts** | 1K |
| **SPU coprocessor dynamic/leakage power** | 3.12uW/67nW |

Fig. 4 shows the energy consumption (per classification) of the RBM algorithm using the conventional system. In the simulation, we change the assumed CPIs from 100 to 0.1 and change the number of CPU cores from 1 to 16. Note that the average CPI values include all sources of delay, including cache miss delays. In the conventional system, we assume that the performance of the multi-core CPU is highly scalable. For example, the 16-core CPU will always be 16 times faster than the 1-core CPU for any computing task. As we know, this is not true in reality. However, even in this ideal case, we noticed that increasing the number of the cores will decrease the system energy-efficiency. For example, when CPI=100, the system using 16 cores consumes 11.7% more energy than the system using a single core CPU. When CPI=0.1, the system using 16 cores consumes 14 times more energy than the system using one core. In addition, reducing the CPI can significantly increase the system energy-efficiency. For example, with one CPU core, the energy consumption of the system with CPI=0.1 is only 1.8% of the one with CPI=100. If the number of cores is increased to 16, the energy consumption of the system with CPI=0.1 is only 23.7% of the one with CPI=100. However, it is very hard to reduce the CPI in reality for data-intensive applications due to cache misses and other delays.

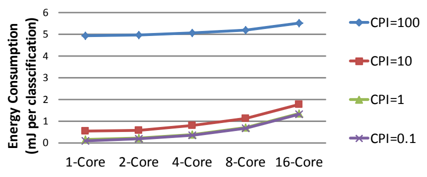

Figure 4. Energy consumption per classification of the RBM algorithm using the conventional system.

Fig. 5 shows the energy consumption of the RBM algorithm using the SPU-based system. In the simulation, we change the number of flash channels from 4 to 32 (each channel has a single NAND flash package), and change the gate count of the SPU coprocessors from 1K to 1M. In the SPU-based system, we assume the host CPU has CPI=100 and the host CPU has only a single core. Note that when we increase the gate count of the coprocessor, it means that the coprocessor can implement more computing kernels. We noticed that by increasing the gate count of the coprocessor from 1K to 1M, the energy consumption is increased by less than 4%. Thus, as long as the die area has available space, adding more logic units into the coprocessor has little or no cost in terms of manufacturing and power consumption, and it can benefit more applications. We also noticed that, by increasing the number of channels, the energy consumption is decreased. This is because the ratio between the memory/storage and the coprocessor is a constant as the number of channels increases. Thus, compared to the conventional system, the SPU is a highly scalable architecture in terms of energy-efficiency. The minimum energy consumption of the conventional system shown in Fig. 4 is 0.09mJ when CPI=0.1 with a single core CPU. The maximum energy consumption of the SPU-based system shown in Fig. 5 is 0.037mJ when the coprocessor gate count is 1M and the SPU uses 4 channels. Even in this worst-case comparison, the SPU-based system is still 2.4 times more energy-efficient than the conventional system.

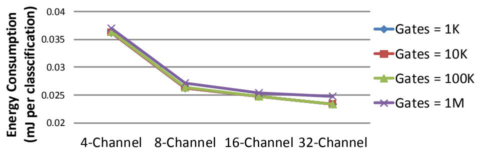

Figure 5. Energy consumption per classification of the RBM algorithm using the SPU-based system.

Fig. 6 shows the energy consumption of the facial recognition algorithm using the conventional system. Compared to the RBM algorithm, the size of the dataset is much larger than the main memory. Thus, the main memory needs to load data from the SSD multiple times. In this case, we noticed that when CPI=100, using more CPU cores can decrease the system energy consumption. However, the most efficient way to improve the system energy-efficiency is still to reduce the CPI.

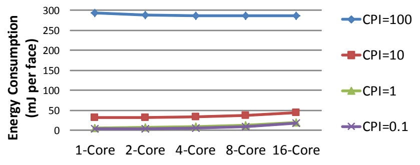

Figure 6. Energy consumption per face of the facial recognition algorithm using the conventional system.

Fig. 7 shows the energy consumption of the facial recognition algorithm using the SPU-based system. The maximum energy consumption of the SPU-based system is only 7% of the minimum energy consumption of the conventional system.

Based on these simulation results, for the more general cases of data-intensive applications, when the host's CPI is 100, the SPU-based system is at least 100 times more energy-efficient than the conventional system. For example, Fig. 8 compares the energy consumption of the conventional quad-core system and the 16-channel

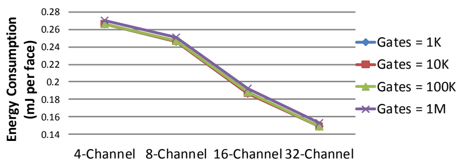

Figure 7. Energy consumption per face of the facial recognition algorithm using the SPU-based system.

SPU-based system. When the host's CPI is 100 in the conventional system, the SPU-based system is 204 times more energy-efficient for the RBM algorithm and 1521 times more energy-efficient for the facial recognition algorithm. The improved energy-efficiency is mainly from the energy savings for data transfers and using multiple low-power coprocessors for parallel computing instead of the high-performance host CPU. In addition, the SPU-based system also has better performance. For example, for the RBM algorithm, the processing time of the 16-channel SPU-based system (0.05ms) is only half of the conventional quad-core system with CPI=10 (0.1ms). For the facial recognition algorithm, the processing time of the 16-channel SPU-based system (0.1ms) is only 3% of the conventional quad-core system with CPI=10 (3.4ms).

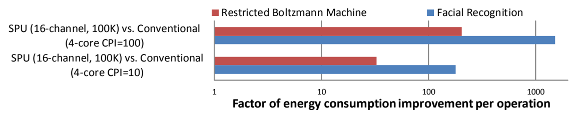

Figure 8. Energy consumption comparison of the conventional quad-core system and the 16-channel SPU-based system.

## V. Conclusion

This paper proposes the *storage processing unit* (SPU) architecture to improve system energy-efficiency by adding computing power into the NAND flash-based SSDs. Compared to the conventional system, the SPU-based system reduces energy consumption significantly by substantially reducing the data transfer cost. This allows for significantly more compute resources within a given system power budget. In summary, the proposed SPU will significantly benefit data-intensive applications in datacenters in terms of energy-efficiency without sacrificing performance.

## Acknowledgment

This work was supported in part by the Center for Research in Intelligent Storage (CRIS), which is supported by National Science Foundation grant no. IIP-0934396 and member companies, and NSF grant no. IIP-1127829. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF. This work was performed when Peng Li was a Ph.D. candidate at the University of Minnesota. The authors would like to thank Laurent Isenegger from Intel for his help on the CoFluent tool, and Prof. David Du from University of Minnesota for his valuable suggestions on this project.

## References

[1] M. Poess and R. O. Nambiar, "Energy cost, the key challenge of today's data centers: a power consumption analysis of tpc-c results," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1229–1240, 2008.

[2] M. Gokhale, B. Holmes, and K. Iobst, "Processing in memory: The terasys massively parallel pim array," *Computer*, vol. 28, no. 4, pp. 23–31, 1995.

[3] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.

[4] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, *et al.*, "The landscape of parallel computing research: A view from berkeley," tech. rep., Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.

[5] Y. Yang and V. Prasanna, "Robust and scalable string pattern matching for deep packet inspection on multi-core processors," *Parallel and Distributed Systems, IEEE Transactions on*, 2012.

[6] W. Harrod, "A journey to exascale computing," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pp. 1702–1730, 2012.

[7] J. S. Vetter, *Contemporary High Performance Computing: From Petascale Toward Exascale*. Chapman & Hall/CRC, 2013.

[8] K. Pretz, "The future of computing," *The Institute, IEEE*, 2013.

[9] E. Riedel, C. Faloutsos, and D. F. Nagle, "Active disk architecture for databases," tech. rep., Carnegie Mellon University, 2000.

[10] B. Van Essen, R. Pearce, S. Ames, and M. Gokhale, "On the role of NVRAM in data-intensive architectures: An evaluation," in *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pp. 703–714, IEEE, 2012.

[11] "Process Integration, Devices, and Structures (PIDS)," *The International Technology Roadmap for Semiconductors*, 2012.

[12] Z. Ou *et al.*, "Energy-and cost-efficiency analysis of arm-based clusters," in *Cluster, Cloud and Grid Computing (CCGrid), 12th IEEE/ACM International Symposium on*, IEEE, 2012.

[13] M. Turk and A. Pentland, "Eigenfaces for recognition," *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71–85, 1991.

[14] G. Hinton, "A practical guide to training restricted boltzmann machines," *Momentum*, vol. 9, 2010.

[15] S. Reinhardt, "Discovery in big data using a graph analytics appliance," Tech Talk, San Diego Supercomputer Center, 2013.

[16] D. Allred and A. Gaurav, "Software and hardware challenges due to the dynamic raw NAND market," *EE Times*, 2011.

[17] J. Kowalik and T. Puzniakowski, *Using OpenCL: Programming Massively Parallel Computers*. IOS Press, 2012.

[18] K. Gomez and P. Li, "Quantum tunneling through the memory wall," Big Data Workshop, Center for Research in Intelligent Storage, 2013.

[19] "Open NAND flash interface specification 3.1," *www.onfi.org*, 2012.

[20] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and C. Ren, "Exploring and exploiting the multilevel parallelism inside ssds for improved performance and endurance," *Computers, IEEE Transactions on*, vol. 62, no. 6, pp. 1141–1155, 2013.