A Near-Data Processing Server Architecture and Its impact on Data Center Applications

Xiaojia Song[1], Tao Xie[1], and Stephen Fischer[2]

[1] San Diego State University, 5500 Campanile Dr, San Diego, CA 92182, USA
{xsong2,txie}@sdsu.edu
[2] Samsung Semiconductor, 3655 N 1st St, San Jose, CA 95134, USA
sg.fischer@samsung.com

Abstract. Existing near-data processing (NDP) techniques have demonstrated their strength for some specific data-intensive applications. However, they might be inadequate for a data center server, which normally needs to perform a diverse range of applications from data-intensive to compute-intensive. How to develop a versatile NDP-powered server to support various data center applications remains an open question. Further, a good understanding of the impact of NDP on data center applications is still missing. For example, can a compute-intensive application also benefit from NDP? Which type of NDP engine is a better choice, an FPGA-based engine or an ARM-based engine? To address these issues, we first propose a new NDP server architecture that tightly couples each SSD with a dedicated NDP engine to fully exploit the data transfer bandwidth of an SSD array. Based on the architecture, two NDP servers ANS (ARM-based NDP Server) and FNS (FPGA-based NDP Server) are introduced. Next, we implement a single-engine prototype for each of them. Finally, we measure performance, energy efficiency, and cost/performance ratio of six typical data center applications running on the two prototypes. Some new findings have been observed.

Keywords: Near data processing · data center server · FPGA · ARM embedded processor · data-intensive · compute-intensive.

1 Introduction

A spectrum of near-data processing (NDP) work [1,3,6,10,17,16,24,18,27,29,30,24] have been proposed recently. Although they target data at different levels of the memory hierarchy, they share a common idea: deploying some hardware data processing accelerators (hereafter, NDP engines) such as FPGAs and embedded processors in or near memory devices to process data locally. NDP is a one-stone-two-birds approach. It largely reduces the pressure of data transfer as the size of processed data is normally smaller than that of raw data. Also, it alleviates the burden of host CPUs by offloading part or all computations to NDP engines.

However, existing NDP techniques might not be able to provide a satisfying solution to a data center server, which usually needs to perform a diverse range of applications from data-intensive to compute-intensive. This is mainly because they only aim at improving performance and energy-efficiency for some specific data-intensive applications such as databases [11,15,17,27], word count [3], linear regression [3], and scan [3]. It is understandable that they concentrate on data-intensive applications. After all, the major incentive of NDP is to reduce the increasingly heavy data transfer pressure of data-intensive applications. Recently, the processing power of NDP engines has been substantially increased [20]. For example, ARM Cortex-A53, a state-of-the-art embedded processor, is a quad-core 64-bit processor operating at 1.1 GHz [9]. The Xilinx VCU1525 FPGA board released in 2017 is equipped with 16 GB DDR4 memory and a Kintex UltraScale FPGA chip, which has 5k DSP slices, 1M logic cells, and 75.9 Mb block RAM [28]. We argue that NDP now also has a potential to benefit compute-intensive applications by considerably alleviating the computational burden of host CPUs as well as reducing their data movement. Thus, building a versatile NDP server that can benefit a wide range of data center applications becomes feasible. Unfortunately, such NDP server is not available yet. Besides, a good understanding of the impact of NDP on data center applications is still missing. For example, whether a compute-intensive application can also benefit from NDP remains an open question. In addition, FPGAs [6,27,29] and embedded processors (e.g., ARM processors) [3,24] are two main types of NDP engines. Which type of NDP engine is a better choice for an NDP server? In order to answer this question, a quantitative comparison between the two types of NDP engines in terms of performance, energy efficiency, and cost/performance ratio is required. Still, it cannot be found in the literature.

To address these issues, in this research we first propose a new versatile NDP server architecture (see Figure 1a), which employs an array of NDP engines between host CPUs and an SSD array. Based on the architecture, two NDP servers called FNS (**F**PGA-based **N**DP **S**erver) and ANS (**A**RM-based **N**DP **S**erver) are then introduced. In both ANS and FNS, there are multiple SSDs with each having its corresponding NDP engine. Next, we implement a single-engine prototype for each of them based on a conventional data center server (hereafter, conventional server). While SANS (**S**ingle-engine **ANS**) utilizes an ARM Cortex-A53 processor [9] as its NDP engine, SFNS (**S**ingle-engine **FNS**) employs FPGA logic as its NDP engine (see Section 4). Further, we measure performance, energy efficiency, and cost/performance ratio for six typical data center applications (see Section 4.3) on the two prototypes. Finally, we obtain some new findings after analyzing our experimental results. To the best of our knowledge, this is the first study that provides a quantitative comparison between the two major types of NDP engines. Also, this research is the first investigation on the impact of NDP on compute-intensive applications.

2 Related Work

According to the location of NDP engines in the memory hierarchy, existing NDP techniques can be generally divided into three groups: in-storage comput-

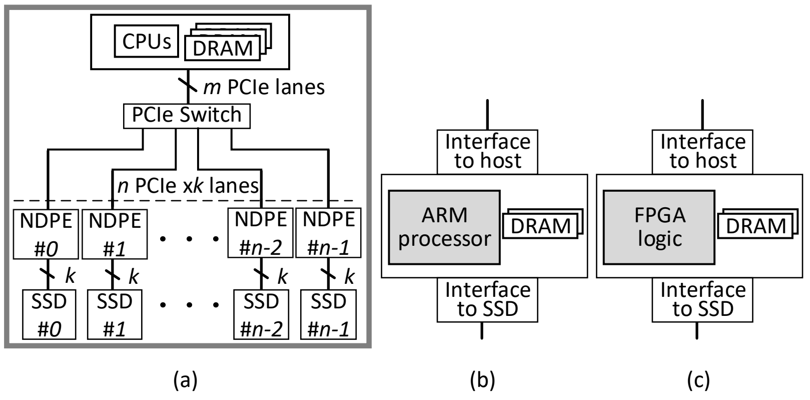


Fig. 1: (a) NDP server; (b) ARM-based NDP engine; (c) FPGA-based NDP engine.

ing (ISC), in-memory computing (IMC), and near-storage computing (NSC). Although various ISC and IMC techniques have shown their strength in the laboratory, so far none of them is publicly available. NSC, however, is more practical as one can develop an NSC-based computer using some commodity products (e.g., a server and FPGA). Thus, in this research we employ NSC to study the impact of NDP on data center applications.

NSC techniques usually insert computing devices on the path between storage devices (e.g., SSDs) and host CPUs to accelerate data processing. Ibex [27] is developed as an FPGA-based SQL engine that accelerates relational database management systems, whereas Netezza [5] builds a server equipped with one FPGA between main memory and storage to extract useful data. Firebox [2] consists of many fine-grained components of SoCs and memory modules connected with high-radix switches. Hewlett-Packard utilizes configurable fine-grained processing cores and memory pools to build a "machine" by connecting them with a photonic network [26]. Interconnected-FPGAs [29] proposes to build a computer system with one FPGA-based NDP engine to accelerate join operations in a database. While all existing NSC techniques only utilize one NDP engine in a server, ANS/FNS adds an NDP engine for each SSD of an SSD array to fully exploit the parallelism among the SSDs.

3 NDP Server Architecture

In this section, we first introduce the architecture of a conventional server. Next, we propose a versatile NDP server architecture, which inspires ANS and FNS.

3.1 The architecture of a conventional server

The architecture of a conventional server with all flash storage can be envisioned from Figure 1a by removing the NDP engine (i.e., NDPE) array. Its main components include one or multiple multi-core CPUs, DRAM, PCIe bus, PCIe switch, an array of SSDs, and network interface. The PCIe bus provides a high bandwidth data path between the CPUs and SSDs. There are k PCIe lanes ($k \geqslant 1$) for each SSD. A PCIe switch is in charge of the data path between CPUs and SSDs. The host CPUs concurrently access all SSDs through the PCIe switch.

Two limitations exist in a conventional server architecture. First, data transfer bandwidth provided by an array of SSDs is underutilized because the number of PCIe lanes from SSDs to the PCIe switch (i.e., $n \times k$) is usually much larger than that number from the PCIe switch to CPUs (i.e., m). Consequently, data transfer may become a performance bottleneck for a data-intensive application as the full bandwidth of the SSD array cannot be exposed to the CPUs. Another limitation is that for a compute-intensive application a performance bottleneck could occur on the CPUs. It seems that allocating engines (e.g., FPGAs) near host CPUs could also alleviate this limitation. However, doing so would require all raw data to be transferred from SSDs to the DRAM of host CPUs through the PCIe bus and PCIe switch (see Figure 1a), which decreases the performance and energy-efficiency. Besides, deploying engines close to host CPUs is useless for a data-intensive application as its data transfer bottleneck cannot be solved.

To address the two limitations, we propose a new NDP server architecture that employs an NDP engine array between the PCIe switch and the SSD array so that each SSD is coupled with an NDP engine (see Figure 1a). The rationale behind the new architecture is that deploying data processing engines near SSDs could benefit both data-intensive and compute-intensive applications. In addition, tightly coupling one SSD with one NDP engine enables an NDP server to fully exploit the storage bandwidth. Also, it makes the server scale well. Based on this new architecture, two NDP servers (i.e., ANS and FNS) are introduced.

3.2 The new NDP server architecture

Figure 1a shows the architecture of our proposed NDP server. The only difference between a conventional server and an NDP server based on the new architecture is that the latter has an extra layer of NDP engines. Each NDP engine consists of four key components: a processing element (PE), DRAM, an interface to host, and an interface to SSD. For an ANS, a PE is simply an embedded processor like an ARM Cortex-A53 (see Figure 1b). For an FNS, the FPGA logic used by an application kernel (i.e., a partition of an FPGA chip) serves as a PE (see Figure 1c). This is because an FPGA chip is relatively expensive. Multiple SSDs sharing one FPGA chip is more practical than each SSD owning an FPGA chip. Note that application kernels generated from one FPGA chip can concurrently process data from distinctive SSDs. The DRAM stores metadata. Also, it works as a buffer for data movement among an SSD, an NDP engine, and host CPUs.

A data processing procedure is always launched by host CPUs, which are in charge of the following tasks: (1) managing the operating system of the server; (2) monitoring the statuses of all NDP engines; (3) executing the host-side application; (4) offloading the application kernel to all NDP engines; (5) writing

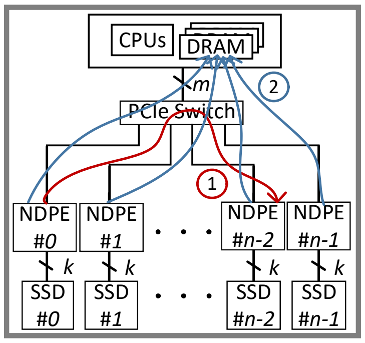
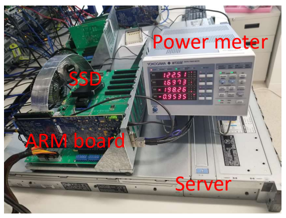


Fig. 2: Data transfer in an NDP server.

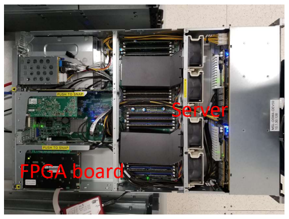
the arguments to an application kernel in an NDP engine and then enabling it to read and process the data from its corresponding SSD.

In the NDP architecture proposed by [29], each server (called computing node in [29]) only has one NDP engine and NDP engines belong to different servers are interconnected in order to reduce the communication cost caused by data exchange between different NDP engines. In our proposed NDP server architecture, however, NDP engines are not directly connected to each other because doing so will make hardware connection routing very complicated considering that each NDP server proposed in this research can have dozens of NDP engines.

Instead, when an NDP engine has a need to transfer data to one of its peers, it leverages a PCIe peer-to-peer (P2P) communication strategy [19], which is a part of the PCIe specification. The PCIe P2P communication enables regular PCIe devices (i.e., NDP engines in our case) to establish direct data transfer without the need to use host memory as a temporary storage or use the host-CPU for data movement. Thus, data transfer from a source NDP engine to a destination NDP engine can be accomplished through the PCIe switch in a DMA (Direct Memory Access) manner. PCIe P2P communication significantly reduces the communication latency and does not increase hardware design complexity. The data path 1 shown in Figure 2 illustrates this process. After all NDP engines finish their data processing, the results from each NDP engine will be aggregated at the host-DRAM for a further processing in CPU. The data path 2 shown in Figure 2 clarifies this case. Compared to the NDP architecture proposed in [29], our proposed NDP server architecture lays a burden on the host when an application needs to frequently exchange data or messages between different NDP engines. However, our architecture has two advantages: (1) the design complexity is greatly reduced; (2) it is more compatible with a conventional server. In addition, the fine-grained coupling of NDP engines with SSDs (i.e., each SSD has an exclusive NDP engine) in a shared-nothing architecture leads to a very high degree of parallelism in data transfer and data processing. It also delivers a very good scalability to the proposed server architecture.



(a)



(b)

Fig. 3: (a) The SANS prototype; (b) The SFAN prototype.

4 Implementations

In this section, we first explain our implementation methodology. Next, we describe how we implement the two single-engine NDP server prototypes SANS and SFNS, which are all extended from a state-of-the-art server with two 18-core Intel Xeon CPUs and 36 PCIe SSDs [22]. Finally, we provide implementation details of six data center applications.

4.1 Implementation methodology

To develop two NDP servers (i.e., ANS and FNS) based on our proposed architecture shown in Figure 1a, 36 Fidus Sidewinder-100 boards [9] and 36 Xilinx VCU1525 FPGA boards [28] are needed. In addition, each board needs two separate PCIe interfaces to connect an SSD and the PCIe switch, respectively. The high hardware cost and massive hardware revision are beyond our capacity. Fortunately, the major goal of this research is to understand the impact of NDP on data center applications instead of building two fully-fledged NDP servers. Therefore, we only build one NDP engine for each of the two proposed NDP servers shown in Figure 3. Six applications are executed on the two single-engine NDP server prototypes, and then, the results are extrapolated to the case of the two full-size NDP servers (i.e., ANS and FNS), respectively.

The procedure of data processing in an NDP server can be divided into four steps: (1) SSD: data transfer from SSDs to NDP engines; (2) NDP: data processing in NDP engines; (3) NDP2CPU: data transfer from NDP engines to host-DRAM; (4) CPU: data processing in host-CPU. These four steps are organized in a pipelined fashion. If the data throughput bandwidth of each step is denoted as *BW (* is SSD, NDP, NDP2CPU, or CPU), then the system performance of the NDP server is determined by:

$$\text{Min}\{\text{SSDBW, NDPBW, NDP2CPUBW}*\alpha\text{, CPUBW}\} \tag{1}$$

, where α is equal to the size of NDP engine input data divided by the size of NDP engine output data. An example of using this equation can be found in Section 5.1. Based on our tests, the read bandwidth of an SSD is approximately 3 GB/s and NDP2CPUBW is about 36 GB/s (see Table 1). When all 36 SSDs work concurrently the SSDBW is equal to 108 GB/s (i.e., 36×3 GB/s). The NDPBW is

Table 1: Platform setup

	Specifications
Server	Two CPU sockets; 36 SSDs m = 48; n = 36; k=4 (see Figure 1a)
CPU [14]	Xeon 6154 : 64bit, 3.0 GHz, 18 cores, 36 threads
PCIe	48 lanes attached to host CPUs (36 GB/s) 144 lanes attached to SSDs (108 GB/s)
SSD	PCIe×4; 3 GB/s
ARM Platform	Quad-core Cortex-A53: 64-bit, 1.1 GHz [9]
FPGA Platform	Xilinx VCU1525 platform [28]

equal to NDPEBW$\times n$, where NDPEBW denotes the data processing bandwidth of one NDP engine and n is the total number of NDP engines. Obviously, the values of NDPEBW and CPUBW depend on the characteristics of applications. These values of applications will be measured in Section 5. We will use Equation 1 to calculate the performance of the six applications running on ANS/FNS.

4.2 Implementation of SANS and SFNS

The PE of each NDPE of an ANS is a quad-core Cortex-A53 ARM processor embedded in a Fidus Sidewinder-100 SoC board [9]. Table 1 summarizes the specifications of the conventional server we used and the Fidus board. The board's PCIe Gen3 NVMe interfaces enable the ARM cores to directly read data from an attached SSD. Its PCIe $\times$ 8 host interface and 1 Gigabit Ethernet interface provides a channel for data movement and communication from/to the host CPUs. In our experiments, an application is first compiled by a cross-platform compiler aarch64-linux-gnu-g++. Next, the executable file is offloaded from host CPUs to the ARM cores in an NDPE. Finally, a data processing procedure is launched by the ARM cores. For each application, we measure its performance, energy efficiency and cost/performance ratio.

In an SFNS, the PE of an NDPE is built by FPGA logic (see Figure 1c). We use a Xilinx VCU1525 FPGA board [28] to implement that NDPE. The specifications of the FPGA board are presented in Table 1. The FPGA board is plugged into a PCIe slot of the server (see Figure 3b). The six applications are implemented in C++ and then compiled into binary files using Vivado High-Level Synthesis (HLS) [28] tool chain. The OpenCL framework is employed for a general management of the kernel running on NDP engines, which includes programing the device, setting arguments for the kernel, and launching the kernel. The pseudo code of the management is shown as below.

An SDAccel [28] development environment is used to evaluate the applications on FNS. It includes a system compiler, RTL level synthesis, placement, routing, and bitstream generation [28]. The system compiler employs underlying tools for HLS. The VCU1525 FPGA board is plugged into a PCIe Gen3 $\times$ 8 slot of the conventional server (see Figure 3b).

```
for $kernel \leftarrow 0$ to $N-1$ do
    cl::Program::Program(context, devices, binaryfile)
    //create a program project
    cl::Program::Kernel(program, kernel_name)
    //create a kernel object
    cl_int cl::Kernel::setArg()
    //set argument and workload for kernel
    cl::CommandQueue::enqueueTask(kernel)
    //program the FPGA and launch the kernel
end
```

In the SDAccel development environment, the OpenCL (Open Computing Language) [13] standard is used for parallel programming. It provides a programming language and runtime APIs to support the development of applications on the OpenCL platform model, which includes the host CPUs and FPGAs. Details of SDAccel and OpenCL can be found in [28]. Note that the data flow of SFNS is different from that of the OpenCL framework, which is shown in Figure 4. The primary benefit of NDP comes from reducing data movement by directly reading/processing data from where they are stored (i.e., SSDs in the case of FNS). Thus, an FNS engine is expected to be able to fetch data from an SSD to the DRAM of an NDP engine (step 1 in Figure 4a). And then, the data are transferred to the FPGA to be processed, after which the results are sent back to the DRAM in the NDP engine (step 2 in Figure 4a). Finally, the results will be transferred to the host-CPU (step 3 in Figure 4a). Unfortunately, the proposed NDP engine in an SFNS is built in the OpenCL framework, which always starts data processing from host-CPU. When there is a need to execute an application kernel on the FPGA board, the host-CPU first reads data from an SSD to the host-DRAM (step 1 in Figure 4b). Next, the host-CPU writes the data to the DRAM in an NDP engine (step 2 in Figure 4b). After the data have been processed in the NDP engine (step 3 in Figure 4b), they are eventually transferred to the host-DRAM (step 4 in Figure 4b).

Since it is difficult, if not impossible, to change the data flow of the OpenCL framework to the way that an SFNS desires, we find a workaround to bypass this issue. In particular, we use the steps 2-4 in Figure 4b to emulate the steps 1-3 in Figure 4a in our experiments in order to estimate an application's wall

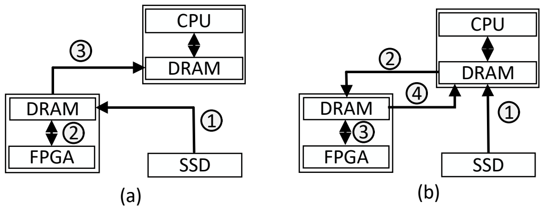


Fig. 4: (a) Data flow in SFNS; (b) Data flow in OpenCL framework.

time when it is running on an SFNS. The only difference between these two sets of steps lies in where to fetch the raw data. While SFNS is expected to achieve this by reading data from an SSD to the DRAM of an NDP engine (step 1 in Figure 4a), the OpenCL framework actually accomplishes this task by transferring raw data from host-DRAM to the DRAM of an NDP engine (step 2 in Figure 4b). However, the VCU1525 FPGA board can deliver a 10 GB/s data transfer bandwidth [28] in step 2 shown in Figure 4b, which is much higher than the 3 GB/s data transfer bandwidth provided by an SSD (see Table 1) in step 1 shown in Figure 4a. Therefore, a delay is injected to deliberately lower the data transfer bandwidth from 10 GB/s to 3 GB/s, by which we achieve our goal. To balance the workload among all NDP engines, the data set is equally split across the SSD array. In fact, the amount of workload for each kernel is set on the host program during its argument stetting phase for a kernel.

4.3 Implementation of the applications

Six applications with distinct characteristics are chosen to study the impact of NDP on data center applications. They are run on CNS, SANS, and SFNS, respectively.

Linear Classifier (LC) In the field of machine learning, a linear classifier achieves statistical classification by making a classification decision based on the value of a linear combination of the features [23]. If the input feature vector to the classifier is a real vector $\boldsymbol{x}$, then the output score is $y = f(\boldsymbol{w} \cdot \boldsymbol{x}) = f\left(\sum_j w_j x_j\right)$. In our experiments, j is set to 8, which makes LC a data-intensive application. A parallel implementation of this algorithm can be found at [23]. The size of the dataset used for this application is 37 GB. Since the classification for each data point is independent from the other points, the classifying of each point can be parallelized among the 36 NDP engines.

Histogram Equalization (HE) Histogram equalization is a computer image processing technique used to improve contrast in images. Histogram equalization transforms pixel intensities so that the histogram of the resulting image is approximately uniform. This allows for areas of lower local contrast to gain a higher contrast [23]. A parallel implementation of this algorithm can be found at [23]. The dataset size used for this application is 3.4 GB. The execution of histogram equalization on different pictures can be done concurrently across the 36 NDP engines.

k-NN_2, k-NN_6, and k-NN_8 Given a set S of n reference data points in a dimensional space and a query point q, the k-NN algorithm [21] returns the k points in S that are closest to point q. Main steps of k-NN include: (1) computing n squared Euclidean distances between the query point q (x1, x2, ..., xi) and the n reference points of the set S (s1, s2, s3, ..., si);

$$distance = (x1 - s1)^2 + (x2 - s2)^2 + ... + (xi - si)^2 \tag{2}$$

(2) sorting the n distances while preserving their original indices specified in S. The k nearest neighbors would be the k points from the set S corresponding to the k lowest distances of the sorted distance array. The dimension of the data in our experiments is set to 9. Since the distance calculation for each point in the database is independent, step 1 can be executed concurrently among all 36 NDP engines. In step 2, the calculated distances are aggregated and then sorted in order to discover the k nearest points of the query point. This step is carried out in the host CPUs after all results from step 1 are aggregated to the host-DRAM. The computational complexity of k-NN depends on the number of features of each data point. The number behind the word k-NN represents the number of features of each data point. For example, k-NN_8 stands for a k-NN algorithm with each data point having 8 features. A larger number of features for each data point implies a more complex k-NN problem. The CPU and ARM codes start from a parallel implementation of the k-NN algorithm from the Rodinia library [21]. The dataset used for this application is totally 130 GB [21].

FFT FFT is an algorithm that samples a signal over a period of time (or space) and then divides it into its frequency components. It is probably the most ubiquitous algorithm employed to analyze and manipulate digital or discrete data. It is also a well-recognized compute-intensive application [12]. The algorithm consists of two 1D FFTs, i.e., a row-wise FFT and a column-wise FFT. Note that for each picture its three color (i.e., R, G, B) values can be processed in parallel as shown in Figure 5. To obtain the best performance on CNS, we employ MKL (Math Kernel Library) [25] for the implementation of 2D FFT on the Xeon CPUs. A 2D FFT implementation on FPGA adopts 1D FFT IP core from Xilinx and it mainly consists of a 256 x 256 size row-wise 1D FFT module, buffer, and column-wise 1D FFT module (see Figure 5). This design is implemented at the RTL level. We run 2D FFT on 800 colorful pictures with total size of 238.54 MB [7].

Among the six applications, LC is the most data-intensive, whereas FFT is the most compute-intensive. Their data processing complexity increases in the following order: LC, HE, k-NN_2, k-NN_6, k-NN_8, FFT. In the same order, they become less data-intensive, which can be seen from the "System BW" columns shown in Table 3 and Table 7. While LC, HE, and FFT only rely on NDP engines to process their input data, the three KNN applications require both NDP engines and host CPUs to accomplish the data processing task. In Section 5, we will run these six applications with distinct data processing complexities

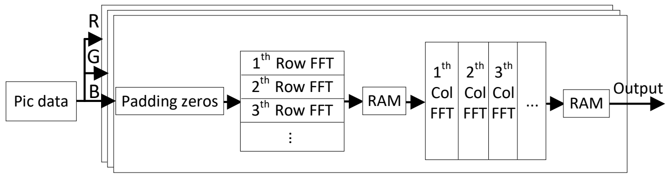


Fig. 5: 2D FFT on FPGA.

on CS, SANS, and SFNS separately. After that, the impact of proposed NDP server on them will be talked.

5 Evaluation

In this section, we measure the performance, energy efficiency, and cost/performance ratio of the six applications running on the conventional server (hereafter, CS), SANS, and SFNS, respectively. The results of ANS and FNS are extrapolated from real measurements of SANS and SFNS, respectively. In ANS and FNS, 36 NDPEs and 36 SSDs are assumed to be employed. Performance is defined as data processing bandwidth of an application when it is running on a server. Energy efficiency is represented by the amount of data that can be processed per joule (i.e., MB/joule). Cost/performance ratio is defined as a server's cost divided by its data processing bandwidth (i.e., dollar/(MB/second)). Obviously, an NDP server is more expensive than a CS because it is equipped with an array of NDP engines, which do not exist in a CS. However, it can deliver a higher performance. Therefore, measuring their cost/performance ratios is a fair method to compare their cost-effectiveness.

5.1 Evaluation of FFT&LC&HE

Performance evaluation: Table 2 and Table 3 show the performance of FFT, LC, and HE on the five servers in terms of wall time and data processing bandwidth, respectively. While "App" shown in Table 2 is a shorthand for "application", "BW" shown in Table 3 is an abbreviation of "data processing bandwidth" (see Equation 1). "All" stands for "all three applications". Since there is no NDP engine in CS, "NA" (i.e., not applicable) is used for the three applications' "Wall time of NDP" and "BW of NDP" columns. Besides, since the three applications are entirely implemented and executed in the NDP engines, there is no computing task for host CPUs. Thus, their values of "Wall time of CPU" and "BW of CPU" are "0" and "$+\infty$", respectively.

The BW of either an NDP engine or host CPUs is equal to the size of dataset divided by wall time. The wall time of ANS/FNS is derived by the wall time of SANS/SFNS divided by 36 as 36 NDP engines can work in parallel assuming that

Table 2: Performance of six applications

	Wall time (s)					
	NDP/CPU					
App	LC	HE	FFT	k-NN_2	k-NN_6	k-NN_8
CS	NA/0.98	NA/0.13	NA/2.05	NA/17.22	NA/31.33	NA/47.45
SANS	16.67/0	1.98/0	132.60/0	825.50/0.62	1843/0.62	2354/0.62
SFNS	13.45/0	1.14/0	3.02/0	36.31/0.62	34.39/0.62	32.83/0.62
ANS	0.46/0	0.06/0	3.68/0	22.93/0.62	51.21/0.62	65.40/0.62
FNS	0.37/0	0.03/0	0.08/0	1.01/0.62	0.96/0.62	0.91/0.62

Table 3: Performance of LC & HE & FFT

	BW (GB/s)					System BW (GB/s)		
	NDP/CPU			SSD(s)	NDP2CPU			
App	LC	HE	FFT	All	All	LC	HE	FFT
CS	NA/37.76	NA/26.15	NA/0.11	108	36	36	26.15	0.11
SANS	2.22/+∞	1.72/+∞	$1.80e^{-3}$/+∞	3	36	2.22	1.72	$1.80e^{-3}$
SFNS	2.75/+∞	2.98/+∞	0.08/+∞	3	36	2.75	2.98	0.08
ANS	80.43/+∞	56.67/+∞	0.06/+∞	108	36	80.43	36	0.06
FNS	100.00/+∞	113.33/+∞	2.91/+∞	108	36	100.00	36	2.91

Table 4: Energy efficiency of LC & HE & FFT

	NDP (Watt)			Server Only (Watt)		Energy Consumption (Joule)			Energy Efficiency (MB/Joule)		
				active	idle						
App	LC	HE	FFT	All	All	LC	HE	FFT	LC	HE	FFT
CS	0	0	0	613.70	30.33	601.43	79.78	1258.10	63.00	43.64	0.19
ANS	207.50	207.50	207.50	613.70	30.33	109.40	14.27	875.21	346.33	243.98	0.27
FNS	421.20	743.40	259.56	613.70	30.33	167.07	23.21	23.19	226.78	150.00	10.29

the dataset has been evenly distributed among the 36 SSDs. "System BW" of an application is derived by Equation 1. It represents the application's performance.

Take LC for example, its execution wall time on the NDP engine of SFNS is 13.45 seconds (see Table 2). Since the size of its dataset is 37 GB, its NDPBW is 2.75 (GB/s) (i.e., 37/13.45, see Table 3). Meanwhile, its SSDBW, NDP2CPUBW, CPUBW are 3 GB/s (only one SSD is used in SFNS), 36 GB/s (see Table 1), and "+∞", respectively. Based on Equation 1, the performance of LC on SFNS is 2.75 GB/s. Unlike FFT and HE whose size of dataset is unchanged after NDP engine processing, the size of dataset of LC is reduced by 8 times (i.e., $\alpha = 8$) after NDP engine processing [23]. To saturate NDP2CPUBW (i.e., 36 GB/s), its NDPBW should be at least 36 x 8 = 288 GB/s, which is much higher than 80.43 GB/s and 100 GB/s (i.e., NDPBW of LC using ANS and FNS). That is why the performance of LC on ANS and FNS is decided by NDPBW rather than NDP2CPUBW.

Energy efficiency: Table 4 summarizes energy consumption and energy efficiency of the three servers. All values of power (Watt) in this table are measured by the power meter shown in Figure 3a. The "Server Only" column provides the power of the CS server. In the CS, since there is no NDPE, a "0" shows up in the "NDP (Watt)" column for the three applications. The total energy consumption of a server is the sum of energy consumption of NDPEs, energy consumption of CPU in active status, and energy consumption of CPU in idle status. For example, the energy consumption of ANS running LC is (207.5+30.33)x0.46 = 109.40 joules. Thus, its energy efficiency is (37x1024)/109.4 = 346.33 MB/joule.

Cost/performance ratio: Cost/performance ratios of the three servers are provided in Table 5. Although in ANS and FNS the host CPUs are not involved

Table 5: Cost/performance ratios of FFT & LC & HE

System	Cost ($)			Cost/Performance ($/(MB/s))		
APP	FFT	LC	HE	FFT	LC	HE
CS	3,543	7,086	7,086	31.45	0.19	0.26
ANS	3,723	7,266	7,266	60.60	0.09	0.20
FNS	5,863	7,504	13517	1.97	0.07	0.37

Table 6: FPGA utilization of 36 NDPEs in FNS

App/Board	LUT	REG	BRAM	DSP slices
FPGA [28]	1,182,240	2,364,480	2,160	6,840
LC	108,108 (9.14%)	125,568 (5.31%)	72 (3.33%)	180 (2.63%)
HE	1,673,352 #(140.54%)	1,961,604 (82.96%)	*6300 0	0 0
k-NN_2	129,816 (10.98%)	172,512 (7.30%)	72 (3.33%)	432 (6.32%)
k-NN_6	227,016 (19.20%)	364,140 (15.40%)	288 (13.33%)	1,440 (21.10%)
k-NN_8	272,916 (23.08%)	444,960 (18.81%)	288 (13.33%)	1,944 (28.42%)
FFT	481,932 (40.81%)	643,608 (27.21%)	180 (8.28%)	3,456 (50.50%)

* Off-chip DRAM used for the overfilled BRAM;
#Larger than 100% means more than one FPGA chip needed.

in data processing, their costs are still taken into account as we are calculating the cost of an entire system. The prices of CPUs, ARM, and FPGA can be found at [14], [4], and [28], respectively. We will take FFT as an example to show how to obtain its cost/performance ratio on CS, ANS, and FNS, respectively. Since MKL [25] recommends using just one thread per host CPU core for FFT to achieve the best performance, we divide the total CPUs' price by two, which is \$3,543. So, the cost/performance ratio of FFT on CS is \$3,543/0.11 GB/s (see Table 3) = 31.45 \$/(MB/s). In an ANS, the total price of the server is the sum of the price of CS and 36 ARM processors[4]. In an FNS, most resources that the 36 NDPEs consume are DSP slices, which account for 50.5% of the FPGA resources (see Table 6). Thus, the cost of FFT on an FNS is the price of host CPUs (i.e., \$3,543) plus a 50.5% of FPGA price (i.e., \$4,593.75 [8]), which is equal to \$5,863. Since the performance of FFT on FNS is 2.91 GB/s (see Table 3), its cost/performance ratio is \$5,863/2.91 GB/s = 1.97 \$/(MB/s).

5.2 Evaluation of the three k-NN applications

Performance evaluation: Table 2 and Table 7 show the performance of k-NN_2, k-NN_6, and k-NN_8 on the five servers in terms of wall time and data processing bandwidth separately. Unlike FFT, LC and HE, an execution of a k-NN application on an NDP server (i.e., ANS or FNS) involves both host CPUs

Table 7: Performance of k-NN_2 & k-NN_6 & k-NN_8

	BW (GB/s)					System BW		
	NDP/CPU			SSD(s)	NDP2CPU	(GB/s)		
k-NN_	2	6	8	All	All	2	6	8
CS	NA/7.55	NA/4.15	NA/2.74	108	36	7.55	4.15	2.74
SANS	0.16/209.68	0.07/209.68	0.06/209.68	3	36	0.16	0.07	0.06
SFNS	3.58/209.68	3.78/209.68	3.96/209.68	3	36	3	3	3
ANS	5.67/209.68	2.54/209.68	1.99/209.68	108	36	5.67	2.54	1.99
FNS	128.71/209.68	136.42/209.68	142.86/209.68	108	36	108	108	108

Table 8: Energy efficiency of k-NN_2 & k-NN_6 & k-NN_8

	NDP (Watt)			Server Only (Watt)		Energy Consumption (Joule)			Energy Efficiency (MB/Joule)		
				active	idle						
KNN_	2	6	8	All	All	2	6	8	2	6	8
CS	0	0	0	613.70	30.33	10568	19227	29120	12.60	6.92	4.57
ANS	207.50	207.50	207.50	613.70	30.33	5815.10	12541	15916	22.89	10.61	8.36
FNS	420.53	429.85	431.87	613.70	30.33	817.06	803.46	782.29	162.93	165.68	170.17

and NDPEs. There are two steps in an execution of a k-NN application. While the first step (i.e., computing n squared Euclidean distances between the query point q and the n reference points of the set S) is performed in NDPEs, the second step (i.e., sorting the n distances while preserving their original indices specified in S) is carried out in host CPUs. Note that in the first step distance calculations can be performed in parallel as there is no data dependency among the distances. While the values in the "NDP" column show the performance of step 1 in NDP engines, the values in the "CPU" column demonstrate the performance of step 2 in host CPUs (see Table 7). The only difference among the three k-NN applications is the number of features used for the distance calculation in step1.

Energy efficiency and **Cost/performance ratio** of the three k-NN applications are summarized in Table 8 and Table 9.

5.3 Impact of NDP on data center applications

Figure 6 summarizes experimental results from Table 3 $\sim$ Table 9. A quantitative comparison between CS and the two NDP servers (i.e., ANS and FNS) is also given in Table 10. Based on Figure 6 and Table 10, several new findings on the impact of NDP on data center applications can be obtained.

Table 10 shows that in terms of performance ANS outperforms CS by 2.23× and 1.38× for LC and HE, respectively. **Finding 1:** *For data-intensive but compute-light applications, ANS can provide performance benefits by offloading computation from host CPUs to NDP engines that are close to data.* The performance benefits stem from ANS' capability of exploiting the full bandwidth of

Table 9: Cost/performance ratios of three k-NNs

System	Cost ($)			Cost/Performance ($/(MB/s))		
k-NN_	2	6	8	2	6	8
CS	7,086	7,086	7,086	0.92	1.67	2.53
ANS	7,266	7,266	7,266	1.25	2.79	3.57
FNS	7,591	7,968	8,390	0.07	0.07	0.08

Table 10: Comparisons among the three servers in three metrics.

Comparison with CS		LC	HE	k-NN_2	k-NN_6	k-NN_8	FFT
ANS	Performance	2.23x	1.38x	0.75x	0.61x	0.73x	0.55x
	Energy Efficiency	5.49x	5.59x	1.81x	1.53x	1.83x	1.42x
	Cost/performance ratio	2.11x	1.30x	0.74x	0.60x	0.71x	0.50x
FNS	Performance	2.78x	1.38x	14.30x	26.02x	39.42x	26.45x
	Energy Efficiency	3.6x	3.44x	12.93x	23.94x	37.23x	54.16x
	Cost/performance ratio	2.71x	0.70x	13.14x	23.86x	31.63x	15.46x

the SSD array, and thus, avoiding the data transfer bottleneck on the path from the PCIe switch to host CPU DRAM (see Figure 1 a). Table 10 also shows that ANS is inferior to CS in terms of performance for k-NN_2 (0.75x), k-NN_6 (0.61x), k-NN_8 (0.73x), and FFT (0.55x), which all have a data processing complexity higher than that of LC and HE. **Finding 2:** *ANS cannot benefit compute-intensive applications in terms of performance.* Although offloading computation to near-data processing engines enables ANS to enjoy the high data throughput bandwidth of the SSD array, for compute-intensive applications these benefits cannot offset the significant discrepancy in computational capacity between a 1.1 GHz embedded processor and a 3.0 GHz Xeon CPU. The trend shown in Finding 1 and Finding 2 can also be observed in Figure 6.

Can a compute-intensive application also enjoy the benefits of NDP? The answer is yes, which is confirmed by the Performance row of FNS in Table 10. **Finding 3:** *FNS can offer performance benefits not only for data-intensive applications but also for compute-intensive applications.* The FPGA's hardware-level acceleration capability in FNS remedies the weakness of an embedded processor in ANS. This advantage of FNS and the benefits brought by NDP (i.e., fully exploiting the high data throughput of the SSD array, and thus, reducing data movement) together explain this finding. The performance benefits cannot be gained by simply putting data processing engines at the host CPU side because data transfer from the PCIe switch to host CPU could become a system performance bottleneck if doing so (see NDP BW 142.86 GB/s and NDP2CPU BW 36 GB/s of k-NN_8 in Table 7).

From Table 10, we obtain the following findings. **Finding 4:** *FNS offers more benefits in terms of performance and energy efficiency for applications with a higher data-processing complexity, which is contrary to ANS.* **Finding 5:** *Compared with CS both ANS and FNS can deliver a higher energy efficiency for all six applications.* The reason is that host CPUs are not energy efficient in

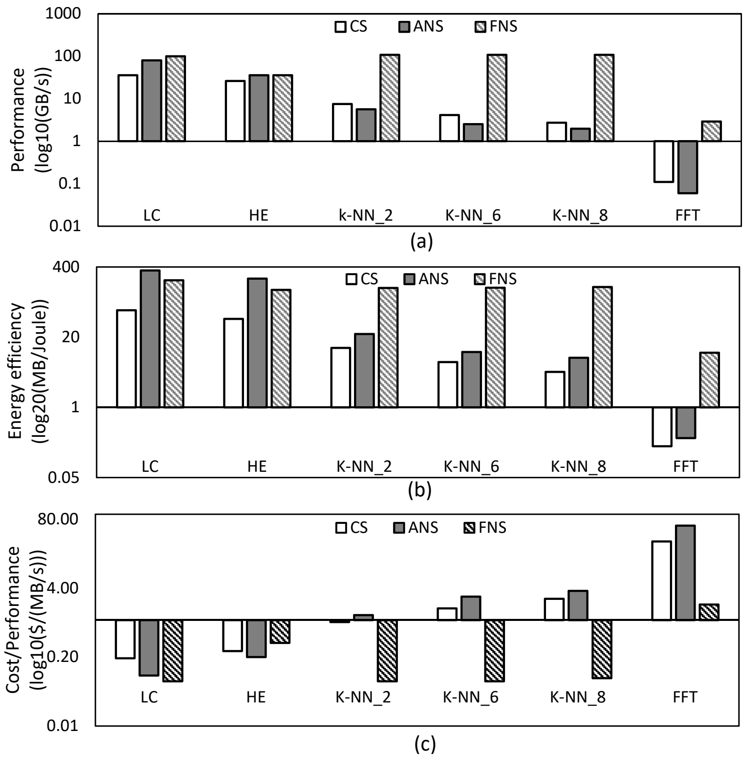


Fig. 6: Comparisons among the three servers in three metrics.

nature. Offloading more computational load to an NDP engine not only relieves the computational burden of host CPU but also reduces the data movement, which can better improve energy efficiency of the entire system. Table 10 shows that ANS can provide a better cost/performance ratio for LC and HE only compared with CS. For the rest four applications, it offers a worse cost/performance ratio. However, FNS can improve cost/performance ratio for all applications except HE. This is because HE consumes too many LUTs (see Table 6). **Finding 6:** *FNS can improve cost/performance ratio for a diverse range of data center applications, whereas ANS can do so only for some compute-light applications.* The conclusion is that FNS is better than ANS in terms of cost-effectiveness.

6 Conclusions

In this paper, we first propose a new NDP server architecture for data center applications. The goal of the new architecture is to benefit a wide range of data center applications from data-intensive to compute-intensive. Next, we implement two single-engine NDP server prototypes. Finally, we evaluate six typical data center applications on a conventional data center server and the two prototypes. Based on our experimental results, several new findings have been obtained. These findings answer some open questions about how NDP impacts data center applications. Now we understand that a compute-intensive application can also benefit from NDP in the three metrics when FPGA-based NDP engines are employed. In addition, we find that compared with an ARM-based NDP engine an FPGA-based NDP engine is more capable of benefiting a wide range of data center applications. Currently, the main merit of an ARM-based NDP engine is to improve energy efficiency and reduce cost/performance ratio for some data-intensive but compute-light applications. For most applications, an FPGA-based NDP engine is superior to an ARM-based NDP engine, and thus, it should be considered first when NDP is applied to a data center server.

7 ACKNOWLEDGMENT

This research was supported by Samsung Memory Solution Laboratory (MSL). We thank our colleagues from MSL who provided insight and expertise that greatly assisted the research. This work is also partially supported by the US National Science Foundation under grant CNS-1813485.

References

1. Ahn, J., Hong, S., Yoo, S., Mutlu, O., Choi, K.: A scalable processing-in-memory accelerator for parallel graph processing. ACM SIGARCH Computer Architecture News **43**(3), 105–117 (2016)
2. Asanovic, K., Patterson, D.: Firebox: A hardware building block for 2020 warehouse-scale computers. In: USENIX FAST. vol. 13 (2014)
3. Cho, S., Park, C., Oh, H., Kim, S., Yi, Y., Ganger, G.R.: Active disk meets flash: A case for intelligent ssds. In: Proceedings of the 27th international ACM conference on International conference on supercomputing. pp. 91–102. ACM (2013)
4. CNXSoft: Allwinner a64 a quad core 64-bit arm cortex a53 soc for tablets (2015)
5. Davidson, G.S., Cowie, J.R., Helmreich, S.C., Zacharski, R.A., Boyack, K.W.: Data-centric computing with the netezza architecture. Tech. rep., Sandia National Laboratories (2006)
6. De, A., Gokhale, M., Gupta, R., Swanson, S.: Minerva: Accelerating data analysis in next-generation ssds. In: FCCM. pp. 9–16. IEEE (2013)
7. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: CVPR. IEEE (2009)
8. Digikey: Price of vcu1525 board. `https://www.digikey.com/products/en?keywords=VCU1525` (2018)
9. Fidus Systems, I.: Fidus sidewinder-100. `https://www.xilinx.com/products/boards-and-kits/1-o1x8yv.html` (2017)
10. Gao, M., Ayers, G., Kozyrakis, C.: Practical near-data processing for in-memory analytics frameworks. In: PACT, 2015 International Conference on. pp. 113–124
11. Gu, B., Yoon, A.S., Bae, D.H., Jo, I., Lee, J., Yoon, J., Kang, J.U., Kwon, M., Yoon, C., Cho, S., et al.: Biscuit: A framework for near-data processing of big data workloads. In: ISCA. pp. 153–165. IEEE (2016)
12. He, H., Guo, H.: The realization of fft algorithm based on fpga co-processor. In: Intelligent Information Technology Application, 2008. IITA'08. Second International Symposium on. vol. 3, pp. 239–243. IEEE (2008)

13. Inc, T.K.G.: The open standard for parallel programming of heterogeneous systems. https://www.khronos.org/opencl/ (2018)
14. Intel: Intel® xeon® gold 6154 processor. https://ark.intel.com/products/120495/Intel-Xeon-Gold-6154-Processor-24_75M-Cache-3_00-GHz
15. István, Z., Sidler, D., Alonso, G.: Caribou: intelligent distributed storage. Proceedings of the VLDB Endowment **10**(11), 1202–1213 (2017)
16. Jo, I., Bae, D.H., Yoon, A.S., Kang, J.U., Cho, S., Lee, D.D., Jeong, J.: Yoursql: a high-performance database system leveraging in-storage computing **9**(12), 924–935 (2016)
17. Jun, S.W., Liu, M., Lee, S., Hicks, et al.: Bluedbm: An appliance for big data analytics. In: Computer Architecture (ISCA). pp. 1–13 (2015)
18. Koo, G., Matam, K.K., Narra, H., Li, J., Tseng, H.W., Swanson, S., Annavaram, M., et al.: Summarizer: trading communication with computing near storage. In: Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture. pp. 219–231. ACM (2017)
19. Mayhew, D., Krishnan, V.: Pci express and advanced switching: evolutionary path to building next generation interconnects. In: High Performance Interconnects, 2003. Proceedings. pp. 21–29
20. Nurvitadhi, E., Sheffield, D., Sim, J., Mishra, A., Venkatesh, G., Marr, D.: Accelerating binarized neural networks: comparison of fpga, cpu, gpu, and asic. In: FPT. pp. 77–84. IEEE (2016)
21. Rodinia: Rodinia:accelerating compute-intensive applications with accelerators (2009)
22. Samsung: Mission peak ngsff all flash nvme reference design. http://www.samsung.com/semiconductor/insights/tech-leadership/mission-peak-ngsff-all-flash-nvme-reference-design/ (2017)
23. Talbot, J., Yoo, R.M., Kozyrakis, C.: Phoenix++: modular mapreduce for shared-memory systems. In: Proceedings of the second international workshop on MapReduce and its applications. pp. 9–16. ACM (2011)
24. Tiwari, D., Boboila, S., Vazhkudai, S.S., Kim, Y., Ma, X., Desnoyers, P., Solihin, Y.: Active flash: towards energy-efficient, in-situ data analytics on extreme-scale machines. In: FAST. pp. 119–132 (2013)
25. Wang, E., Zhang, Q., Shen, B., Zhang, G., Lu, X., Wu, Q., Wang, Y.: Intel math kernel library. In: High-Performance Computing on the Intel® Xeon Phi™, pp. 167–188. Springer (2014)
26. Whitman, M., Fink, M.: Hp labs: The future technology. hp discover las vegas. https://news.hpe.com/content-hub/memory-driven-computing/ (2014)
27. Woods, L., István, Z., Alonso, G.: Ibex: an intelligent storage engine with support for advanced sql offloading. Proceedings of the VLDB Endowment **7**(11), 963–974 (2014)
28. Xilinx: Xilinx xilinx virtex ultrascale+ fpga vcu1525. https://www.xilinx.com/products/boards-and-kits/vcu1525-a.html (2017)
29. Yoshimi, M., Oge, Y., Yoshinaga, T.: Pipelined parallel join and its fpga-based acceleration. TRETS **10**(4), 28 (2017)
30. Zhang, D., Jayasena, N., Lyashevsky, A., Greathouse, J.L., Xu, L., Ignatowski, M.: Top-pim: throughput-oriented programmable processing in memory. In: HPDC. pp. 85–98. ACM (2014)