# <u>A</u>ccelerating <u>D</u>ata <u>A</u>nalytics near <u>M</u>emory: A k-NN Search Case Study

Minho Ha, Joonseop Sim, Jungmin Choi, Donguk Moon, Taeyoung Ahn, Byungil Koh, Eui-Cheol Lim, Kyoung Park
*SK hynix, Icheon 17336, South Korea*
E-mail: minho1.ha@sk.com

***Abstract*—Data explosion driven by advances in big data and AI has caused a demand for huge increase in memory capacity and bandwidth, shifting the bottleneck of system performance from computing to memory side. In this paper, we propose a *Accelerating Data Analytics near Memory* (*ADAM*) applying near data processing approach to solve memory side bottleneck. With the Roofline analysis of k-NN search, one of the representative operations for data analytics application, we observed that the k-NN search is a memory-intensive workload, which is suitable for ADAM to process. To address the limitations of running k-NN search on existing computing system, we propose a system equipped with ADAM cards that accelerates k-NN search without the analytics servers involved in the acceleration. Our evaluation on various feature vector data sizes shows up to 56.7%, 67.3%, and 68.5% improvements in performance, power, and cost, respectively, compared to baseline system. As a result of comparing with upcoming high-end CPU model, performance improvement of up to 27.4% can be achieved in this case as well.**

## I. Introduction

Recently, the amount of data processed in data centers has been explosively increased [1]. To process this soaring amount of data in real time without inconvenience to users is a very important service of the data center. It is known that operations used for data analytics require relatively light computation compared to a large amount of memory access [2]. It implies that data analytics operations are memory-intensive workloads.

Researchers have recognized that CPU-centric architectures are inefficient to handle memory-intensive workloads. Data movement between CPU and memory for frequent data access has been a major cause of overall system performance degradation [3]. To effectively handle memory-intensive workloads where data movement occurs frequently, a specialized architecture is required.

Near data processing (NDP) approach can be a promising solution to address the data movement issue by processing some of the operations right next to the memory where the data is stored. It can fully utilize the internal memory bandwidth and data movement to the host CPU or other accelerators can be mitigated. There have been many prior research related to the NDP approach. However, they lacked prudence about the limited memory capacity for the processing unit. Also, many of them utilized the internal bandwidth of the existing DIMM module, they has a limit to scalability to handle large memory-intensive workloads.

In this paper, we propose *Accelerating Data Analytics near Memory (ADAM)*, a *card-type* memory solution with huge memory capacity, high memory bandwidth, and NDP core optimized for memory-intensive workloads. Since ADAM's NDP core offloads a memory-intensive part from the host operations and returns only the reduced result, it can minimize the amount of data movement while fully utilizing the high internal bandwidth of ADAM card. Also, since ADAM is a card-type solution plugging into serial interface, it can be easily scaled up. These features can lead to performance and power consumption improvement.

Based on the ADAM, we suggest a system to accelerate data analytics operations. We investigated a k-NN search, one of the representative operations of data analytics application, as a case study. First, we quantitatively confirm that k-NN search is a memory-intensive workload based on Roofline analysis [4]. And then, we precisely model baseline system and ADAM-augmented system running data analytics operations. In the baseline system, k-NN search is accelerated through host CPU's SIMD hardware in DB servers, and in the ADAM-augmented system, k-NN search is accelerated by the ADAM cards installed in DB servers. Simulation results show that performance, power, and cost are improved by 56.7%, 67.3%, and 68.5%, respectively, compared to the baseline system. Compared with an upcoming high-end CPU model that supports DDR5, ADAM-augmented system shows 27.4% performance improvement.

## II. Background

### A. Data Analytics Acceleration Examples

*1) Industrial Approach:* Recently, there are efforts to accelerate data analytics in industry. One effort is Advanced Query Accelerator (*AQUA*) for Amazon Redshift [5]. AQUA brings compute to storage by processing critical data in innovative cache. It uses Amazon web service (AWS)-designed processor and a scale-out architecture to accelerate data processing beyond what traditional CPUs can do. AWS reported that AQUA allows Amazon Redshift to automatically enrich certain types of queries to run up to 10x faster than other enterprise cloud data warehouses.

Another effort is the solution proposed by Bigstream [6]. Bigstream's solution offloads the data analytics operations suitable for hardware (FPGA or SmartSSD [7]) acceleration [8] to accelerators. According to Bigstream, users can achieve up to 10x performance improvement without code change. In

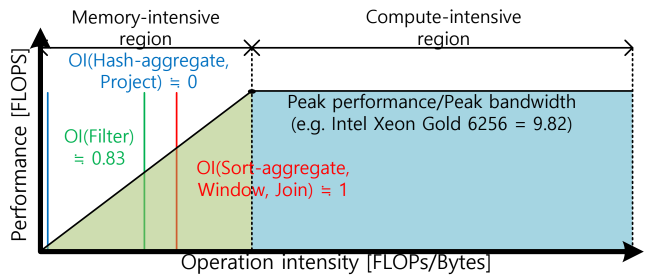


Fig. 1: Roofline plot.

addition to the above solutions, various methods have been proposed in industry.

*2) Academic Approach:* Academia has been also reporting various research to accelerate data analytics [9]–[11]. In [9], *Genesis* processing genomic data efficiently is proposed. Genesis provides up to 19.3x better performance and up to 15x better cost savings than traditional methods. In [10], FPGA engine for accelerating singular value decomposition (SVD) kernel widely used in data analytics is proposed. Through the scalable parallel SVD FPGA engine, computation performance improved by 80x to 300x compared to high-performance CPUs. In [11], a near storage accelerator for database sort, called *NASCENT* utilizing SmartSSD [7], is proposed. NASCENT is 147.2x faster and 131.4x more energy efficient than the CPU baseline as the number of storage devices increases.

## B. Roofline Analysis

Roofline analysis [4], which shows the maximum performance achievable under given hardware conditions, is run to identify memory-intensive workloads among various data analytics operations. Based on the Roofline analysis, we identify the relationship between operational intensity (OI) and performance (FLOPS) of the workload. Roofline can be obtained through Eq. (1).

$$Roofline = min(\pi, \beta \times OI) \tag{1}$$

In Eq. (1), $\pi$ and $\beta$ mean peak performance and peak bandwidth of given hardware, respectively. Roofline plot based on Eq. (1) is like Fig. 1. In Fig. 1, bend point of the Roofline is $\pi/\beta$ point. If the workload is located to the left of the bend point, it is a memory-intensive workload, and if it is located to the right, it is a compute-intensive workload. When Roofline analysis of Spark SQL [12], a representative data analytics operation, is run, a lot of Spark SQL operations are memory-intensive workloads.

## C. Near Data Processing

NDP is a concept of processing data right next to memory where the data is located. Unlike Processing-in-Memory (PiM) [13], which puts processing elements (PEs) *in* memory chip, NDP puts PEs *near* memory chip. If NDP approach is adopted, there is an advantage that internal memory bandwidth can be fully utilized because data is processed right next to the memory. Such NDP approach can reduce the data transfer size because the data is processed near the memory and then only output data is transferred to the host CPU. In other words, communication bottleneck can be alleviated.

To take full advantage of NDP, memory bandwidth should be fully utilized. It means that memory-intensive workloads whose OI is low are suitable workloads for NDP. Since data analytics operations are memory-intensive workloads, as mentioned in Section II-B, a great performance improvement can be expected when applying the NDP approach to data analytics operations.

## III. k-NN Search Acceleration

In this section, we will explain the features of *k-NN search*, which is considered as a case study, and why k-NN search is selected as a case study. And reference data analytics system that accelerates k-NN search will be described.

## A. Features of k-NN Search Algorithm

k-NN search is an algorithm finding $k$ vectors having the closest characteristics through similarity comparison between input vector and vectors to be compared. Although similarity comparison is possible in various ways, we will use *cosine similarity*, the most representatively used for k-NN search. Cosine similarity-based k-NN search is shown in Eq. (2).

$$Similarity(A,B) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2}$$

In Eq. (2), $A$ and $B$ are feature vector and $n$ is feature vector dimension. Although there are square root and division in k-NN search, it mainly consists of simple addition and element-wise multiplication for inner product. Eq. (2) is used for all feature vectors in the database. That is, for k-NN search, all feature vectors in the database must be accessed. In summary, k-NN search requires a lot of memory access while the amount of computation is small.

To quantitatively identify the computational features of k-NN search and make prioritized optimization decisions, we run a Roofline analysis [4]. Fig. 2 shows the Roofline analysis result of k-NN search. For Roofline analysis, Intel Xeon Gold 6256 [14] and 2933MHz DDR4 (6 channels per CPU) are assumed. k-NN search workload parameters for Roofline analysis are as follows: vector dimension is 128, each vector element is 32-bit floating point number, and the number of feature vectors in the DB is 1,900,000. Since k-NN search has a very low OI, approximately 1.43 in this case, it is located in the memory-bound region. It means that no matter how good the CPU performance is, k-NN search cannot be run properly due to the memory bandwidth bottleneck. To effectively process k-NN search, it is more important to solve the memory bandwidth bottleneck than CPU performance, and NDP can be the solution.

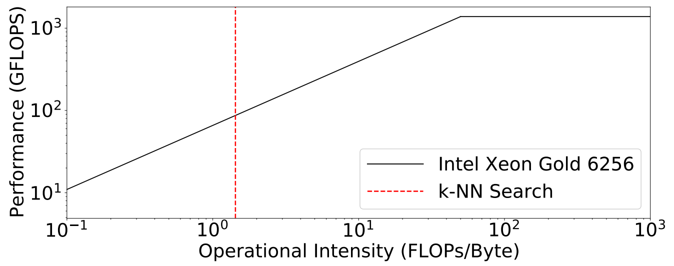


Fig. 2: Roofline analysis result of k-NN search. k-NN search is located in the memory-bound region (slope).

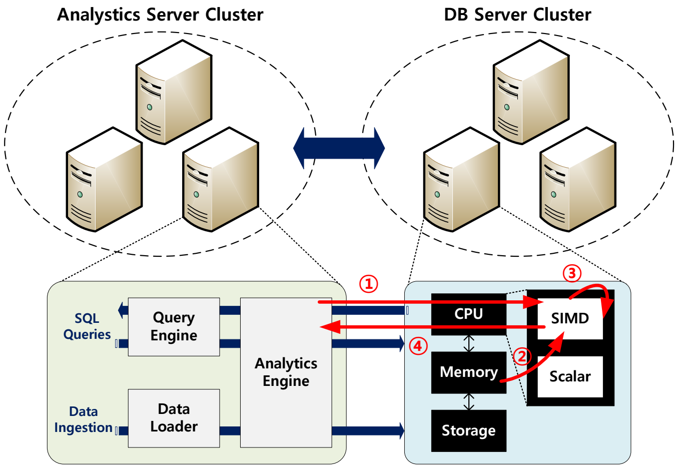


Fig. 3: System for k-NN search acceleration on DB servers & its operation flow: ① Offload k-NN search function from analytics engine to host CPU's SIMD hardware in the DB server, ② Read feature vectors in the DB, ③ Accelerate k-NN search using SIMD hardware, ④ Return results.

## B. k-NN Search Acceleration on DB Servers

We assume a baseline system consisting of an analytics server cluster and a DB server cluster, composed of in-memory DB and storage DB. In general, necessary data from the DB servers is brought to the analytics servers and processed in analytics servers. In addition to providing data to the analytics server cluster, our baseline DB server cluster is able to process selected analytics functions that are pushed down from the analytics server cluster. The push-down feature improves the overall data analytics performance by reducing data movement from the DB cluster to analytics cluster. This can also be applied to k-NN search. Analytic engine (such as Apache Spark [15]) offloads k-NN search function to the DB server, and the offloaded k-NN search is accelerated by SIMD engine of host CPU in the DB server. Although acceleration is possible through SIMD engine in the DB server, k-NN search has a memory-bound feature, so there is still a bottleneck due to host-side memory bandwidth. Therefore, a greater acceleration effect will be achieved by solving the memory bandwidth bottleneck through the NDP approach.

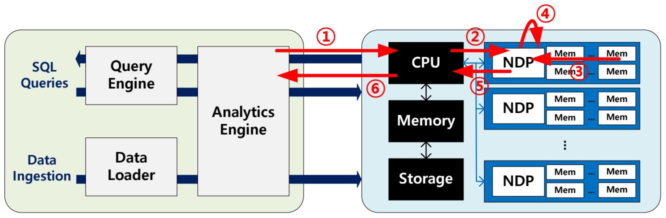


Fig. 4: Proposed system architecture for k-NN search acceleration & operation flow: ① Offload k-NN search function from analytics engine to DB server's host CPU, ② Offload k-NN search function from DB server's host CPU to ADAM's NDP cores, ③ Read feature vectors in the DB, ④ Accelerate k-NN search using NDP cores, ⑤ & ⑥ Return results.

## IV. Accelerating Data Analytics near Memory

### A. Architecture Overview

We propose ADAM to which NDP approach is applied for data analytics acceleration in DB servers. In the proposed system, shown in Fig. 4, ADAM cards are added to the DB servers. By adding ADAM cards with built-in NDP core, memory capacity and bandwidth expansion and calculation functions can be added at the same time. Up to 16, for which the physical size of one server is considered, ADAM cards can be installed in ADAM-augmented server. Like the baseline system, k-NN search is first offloaded to the host CPU in the DB server, but this operation is once more offloaded to the NDP core in the ADAM card (API related to offloading will be explained later). Unlike the baseline system with memory capacity and bandwidth limitations, the proposed ADAM-augmented DB server can provide large capacity and high bandwidth with computation capability, thereby maximizing k-NN search processing performance.

If the feature vector data size in DB server and miscellaneous data exceeds memory capacity of one ADAM card, k-NN search has to be processed in multiple ADAM cards (scale-up case, up to 16 cards per server). In this case, it should be processed hierarchically. After processing k-NN search for each card, one more k-NN search is processed. For the second k-NN search, only $k$ out of up to $16k$ needs to be selected, so it will not significantly affect the total number of operations and data movement. When scale-out is required, k-NN search needs to be additionally processed once more. However, like the scale-up case, the third k-NN search will not significantly affect the total number of operations and data movement. In the case of scale-out, data is exchanged through Ethernet (200Gbps for highend GPU server like DGX-A100 [16]), but the data going out of the server is only $k$ vectors, so there will be no network bottleneck.

### B. Details of ADAM Card

*1) ADAM Card Architecture:* ADAM card, whose architecture is shown in Fig. 5, is composed of NDP core optimized for data analytics processing and huge capacity and high internal bandwidth DDR5 memories. Since the NDP core needs to be

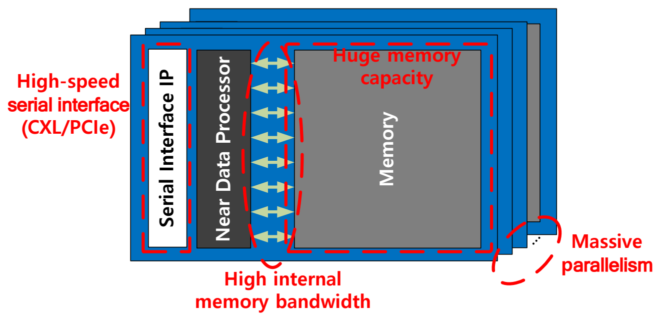


Fig. 5: ADAM card architecture.

TABLE I: ADAM card specification

| | | |
|---|---|---|
| **NDP core configurations** | Frequency [GHz] | 1 |
| | SIMD bit-width | 1024 |
| | # Cores per NDP | 10 |
| | Performance per core [GFLOPS] | 32 |
| | ADAM card performance [GFLOPS] | 320 |
| **Memory configurations** | # of channels per NDP | 8 |
| | Capacity per ADAM card [GB] | 256 |
| | Memory frequency [MHz] | 4800 |
| | Channel bandwidth [GB/s] | 38.4 |
| | ADAM card memory bandwidth [GB/s] | 307.2 |
| **Serial interface configurations** | CXL lanes per ADAM card | 4 |
| | CXL bandwidth [GB/s] | 16 |

TABLE II: Implemented APIs for k-NN search offloading

| **API functions** | **Format** |
|---|---|
| Memory allocation | `ADAM_malloc (int size)` |
| Free memory | `ADAM_free (float *index)` |
| Re-allocation | `ADAM_realloc (float *index, int size)` |
| Request k-NN search and read results | `KNN_scan (int table_ID, int feature_ID, int K, float* array, float *index)` |

able to process various data analytics operations, it is chosen as a lightweight CPU rather than a specialized accelerator. Since memory-intensive workloads require processing large amounts of data at one time rather than fast data processing, ADAM card's NDP is designed to support a wide SIMD bit-width with low operation frequency to maximally utilize high internal memory bandwidth. ADAM cards and the host CPU are connected by a high-speed serial interface such as compute express link (CXL) or PCIe. Up to 16 ADAM cards can be installed per server. It means that scale-up (in-server expansions) of a large capacity and high bandwidth memory solution becomes possible.

*2) ADAM Card Specification:* Specifications of ADAM card are summarized in Table I. NDP core used in the ADAM card is optimized for data analytics function acceleration. Computational performance of NDP itself is 320GFLOPS, which is inferior to Intel Xeon Gold 6256 [14], whose computational performance is 1382.4GFLOPS. However, since the ADAM card supports large capacity (256GB) and high internal bandwidth (307.2GB/s) memory, it is rather suitable for k-NN search, which is a memory-bound operation reported in III-A. As can be seen from Fig. 4, only the k-NN search results processed by NDP core are transmitted to the DB server's host CPU (⑤) and analytics server (⑥), so the serial interface, assuming CXL, between host CPU and ADAM cards can be covered with only a small number of lanes (4 lanes, 16GB/s). When $k$ is 10, for example, 5KB result data is transmitted via CXL. It takes hundreds of nanoseconds considering CXL link and switch latency. This is a negligible amount of time for end-to-end k-NN search perspective.

## C. APIs of k-NN Search Acceleration using ADAM

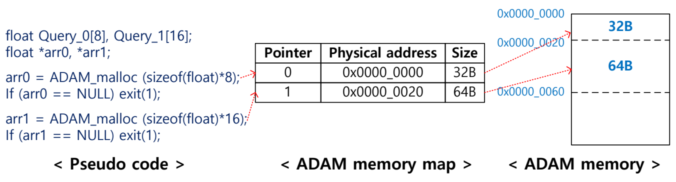
*1) API Overview:* To offload k-NN search to ADAM cards, we define and implement APIs between the host and ADAM cards. Table II summarizes the implemented APIs. APIs can be classified into memory management functions (`ADAM_malloc()`, `ADAM_free()`, and `ADAM_realloc()`) and k-NN search acceleration function (`KNN_scan()`).

For k-NN search, feature vector, input query, and result data are stored in memory of ADAM card (see Fig. 7). Feature vector data is used for cosine similarity calculation. We assume that the feature vector data is already loaded into memories in the ADAM cards. Input query data is data containing various information for conducting k-NN search. Result data is the result of k-NN search processed through the NDP core of ADAM card. Request and response between NDP core and host CPU are executed through mailbox in ADAM card (see Fig. 7). Request and response operate based on interrupt, which will be explained in detailed in Section IV-C3.

*2) Memory Management Functions:* Memory management functions are always required for ADAM cards, regardless of the application. Fig. 6 depicts memory management function APIs. `ADAM_malloc()` allocates the physical address area of the memory in the ADAM card and maps it to a virtual device address (pointer or index) and returns it. `ADAM_free()` frees the memory area allocated with `ADAM_malloc`. `ADAM_realloc()` changes the size of the area allocated with `ADAM_malloc`.

*3) k-NN Search Acceleration Function:* Operation flow of `KNN_scan()` is described in Fig. 7. ① When `KNN_scan()` is called, input query data is transferred to memory in ADAM card through DMA. ② Host CPU sends request message to mailbox and NDP core receives an interrupt from mailbox. ③ NDP core reads the request message and then processes the offloading function based on query data. After the operation is finished, the result is stored in memory. ④ NDP core sends response message to mailbox and host receives interrupt from mailbox. ⑤ When host CPU receives an interrupt from the mailbox, host CPU reads the response message. ⑥ k-NN search result data is transferred to host memory by direct memory access (DMA). In this paper, only k-NN search acceleration function API has been described, but other data analytics acceleration function APIs could be added.

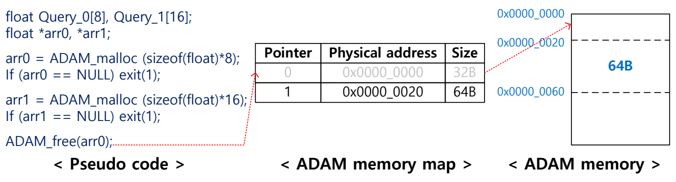
As shown in Fig. 7, not only feature vector data but also input query data and result data are stored in the memory of



```
float Query_0[8], Query_1[16];
float *arr0, *arr1;

arr0 = ADAM_malloc (sizeof(float)*8);
If (arr0 == NULL) exit(1);

arr1 = ADAM_malloc (sizeof(float)*16);
If (arr1 == NULL) exit(1);
```

| Pointer | Physical address | Size |
|---|---|---|
| 0 | 0x0000_0000 | 32B |
| 1 | 0x0000_0020 | 64B |

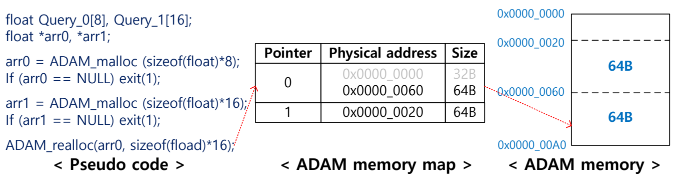
(a) `ADAM_malloc()`



```
float Query_0[8], Query_1[16];
float *arr0, *arr1;

arr0 = ADAM_malloc (sizeof(float)*8);
If (arr0 == NULL) exit(1);

arr1 = ADAM_malloc (sizeof(float)*16);
If (arr1 == NULL) exit(1);

ADAM_free(arr0);
```

| Pointer | Physical address | Size |
|---|---|---|
| 0 | 0x0000_0000 | 32B |
| 1 | 0x0000_0020 | 64B |

(b) `ADAM_free()`



```
float Query_0[8], Query_1[16];
float *arr0, *arr1;

arr0 = ADAM_malloc (sizeof(float)*8);
If (arr0 == NULL) exit(1);

arr1 = ADAM_malloc (sizeof(float)*16);
If (arr1 == NULL) exit(1);

ADAM_realloc(arr0, sizeof(fload)*16);
```

| Pointer | Physical address | Size |
|---|---|---|
| 0 | 0x0000_0000<br>0x0000_0060 | 32B<br>64B |
| 1 | 0x0000_0020 | 64B |

(c) `ADAM_realloc()`

Fig. 6: Memory management function APIs.

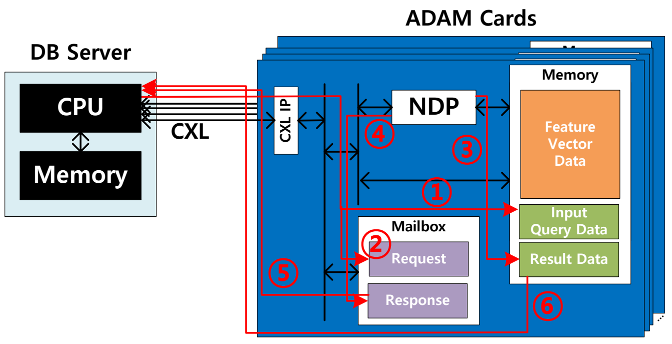


Fig. 7: Implemented `KNN_scan()` API flow.

ADAM cards. However, feature vector data occupies hundreds of GB, whereas input query data and result data occupies only tens to hundreds of KB. In this paper, we will assume that most of ADAM card's memory is used to store the feature vector.

## *D. Benefits of k-NN Search Acceleration using ADAM*

*1) Performance Improvement:* Using a ADAM card, the operation of Eq. (2) can be processed in parallel. In other words, it can be expected that the performance will be improved as much as the ADAM card is inserted. As mentioned above, when a hierarchical k-NN search is required, the second and higher-level k-NN search has little effect on the overall performance, and the network traffic caused by the k-NN search result is very small. So the performance can be improved in proportion to the number of ADAM cards.

*2) Power & Cost Improvement:* If k-NN search is processed using ADAM, great gains can be expected in terms of power and cost. Memory of host CPU is limited in scalability because it is constrained by the number of memory channels per CPU socket. That is, since there is a limit to adding memory DIMMs for expansion of memory capacity and bandwidth, high-cost scale-out, for which whole server should be added for expansion, is required. However, in the case of ADAM, simply plugging in additional CMS cards to serial interface, memory capacity and bandwidth can be easily expanded. It is possible to expand up to 4TB capacity and 4.8TB/s aggregate bandwidth by scale-up only. Until memory capacity of CMS-augmented system reaches 4TB, power and cost increase only in proportion to the number of CMS cards added.

TABLE III: Comparison of ADAM PoC Card and Simulator

| # ADAM cards | | 1 | 2 | 4 |
|---|---|---|---|---|
| **k-NN search performance** | **Measured by PoC** | 1 | 1.9 | 3.8 |
| | **Estimated by simulator** | 1 | 2 | 4 |

## V. Evaluation

### *A. Simulation Setup*

*1) Methodology:* We build an in-house simulator based on full-system modeling that mimics both the baseline system (Fig. 3) and ADAM-augmented system (Fig. 4). Both systems consist of analytics server cluster and DB server cluster. Among the data analytics operations, k-NN search is offloaded to be processed by the DB server. Scale-up or -out of DB server varies depending on the DB size. Sine performance of processor and memory constituting the servers and k-NN search offloading flow are precisely considered, our in-house simulator can accurately predict performance changes according to scale-up or -out. For the baseline system, two types of 1 DIMM per channel (DPC) and 2DPC are considered.

To verify our in-house simulator, we make an ADAM proof-of-concept (PoC) card using SIDEWINDER-100 FPGA [17]. k-NN search is run while increasing the number of ADAM PoC cards, and the simulaton is also run assuming the same number of cards as the PoC. Table III shows the results of the ADAM card PoC and the simulator. For fair comparison, simulation is run based on the ARM Cortex A53 specification mounted on SIDEWINDER-100 and resource utilization from measurement is considered. As shown in Table III, there is no significant difference (less than 5% error) between the experimental results of ADAM card PoC and the simulator. Through these results, the consistency of the in-house simulator can be seen.

*2) System Configuration:* Baseline DB server configuration is summarized in Table IV. In the case of the baseline system, the k-NN search is accelerated through the SIMD hardware of the host CPU in the DB server. Configuration in Table IV assumes 1DPC. If 2DPC is supported, total memory capacity will be doubled and the memory frequency will be reduced to 2666MHz. It is assumed that the ADAM-augmented DB server has the same CPU as the baseline DB server with two 64GB DDR memories. For the baseline system, when the size of DB data exceeds the DB server total memory capacity, the DB server is expanded (scale-out). For ADAM-augmented system, when the size of DB data exceeds the memory capacity of the

TABLE IV: Baseline DB server configuration

| | | |
|---|---|---|
| **CPU configurations** | Frequency [GHz] | 3.6 |
| | SIMD bit-width | 512 |
| | # of SIMD units per core | 2 |
| | # of Cores per CPU | 12 |
| | Performance per core [GFLOPS] | 115.2 |
| | CPU performance [GFLOPS] | 1382.4 |
| **Memory configurations** | # of channels per CPU socket | 6 |
| | Total memory capacity per DB server [GB] | 384 |
| | Memory frequency [MHz] | 2933 |
| | Channel bandwidth [GB/s] | 23.5 |
| | Total memory bandwidth per DB server [GB/s] | 140.8 |

TABLE V: k-NN search model parameters

| | | |
|---|---|---|
| **Benchmark parameters** | top-$k$ | 10 |
| | Feature vector dimension | 256 |
| | # of feature vectors | $2^{28}\sim2^{34}$ |
| | Memory requirements [GB] | 256∼16384 |

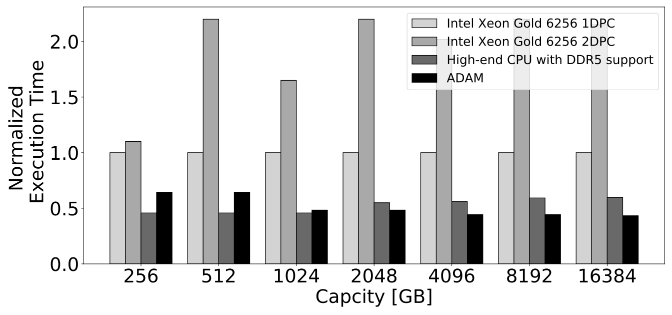
ADAM card, ADAM cards are added (scale-up). When the number of ADAM cards exceeds 16, the server is expanded.

*3) Workload:* As the simulation workload, k-NN search described in Section III will be used. Since performance, power, and cost vary according to the number of feature vectors, it is necessary to conduct experiments in various cases. k-NN search model parameters are summarized in Table V.

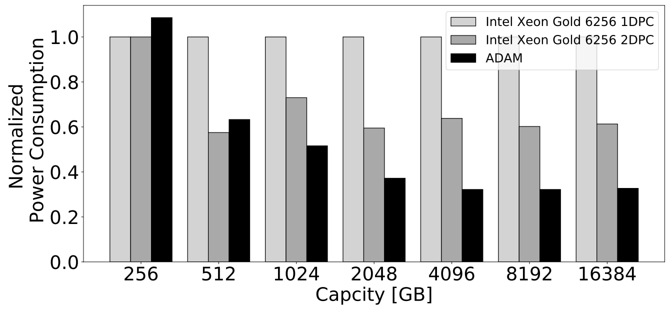
## B. Simulation Results

*1) Execution Time:* Not only the existing Intel Xeon Gold 6256 CPU [14] but also CPU supporting DDR5 to be released in the future are modeled and compared. Difference from the baseline is that the number of memory channels is increased to 8, and the frequency of memory is improved to 4800MHz, which is the DDR5 specification (Since CPU what we modeled for future one has not been released yet, there is no data related to power consumption or cost, so we did not compare power consumption and cost.). As shown in Fig. 8 (a), since ADAM can run k-NN search in complete parallel on a card-by-card basis, execution time does not increase significantly even if the feature vector data increases. When feature vector data size is small, high-end CPU supporting DDR5 memory, whose computing resources is more powerful than NDP, shows better performance. However, since ADAM is optimized for memory-intensive workload, the performance is reversed as the size of feature vector data increases, and it shows up to 27.4% better performance. ADAM improves performance by up to 56.7% and 79.8% compared to Intel Xeon Gold 6256 1DPC and 2DPC server, respectively.

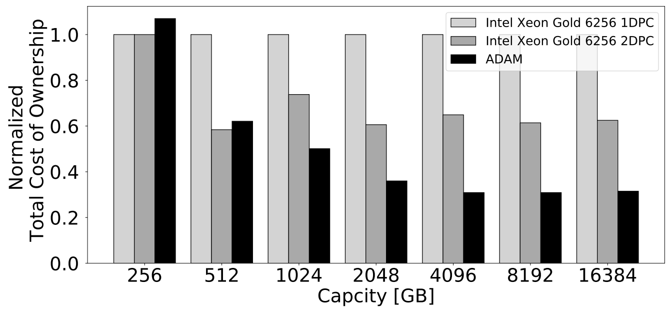
*2) Power Consumption:* As mentioned in IV-D, the baseline needs to scale-out for large capacity memory, but ADAM only needs to scale-up to a certain level. In the case of 2DPC, the memory frequency is low, but instead, the memory capacity can be doubled, so scale-out is possible with fewer servers compared to 1DPC. However, there is a performance degradation (see V-B1). As a scale-up solution, ADAM can achieve up to 67.3% and up to 46.6% less power consumption compared to the 2DPC baseline, respectively. If the k-NN feature vector



(a) Execution time



(b) Power consumption



(c) Total cost of ownership

Fig. 8: Simulation results according to feature vector data size change (smaller is better).

data size is small, there may be no benefits to using ADAM. For example, ADAM-augmented system consumes 8.6% more power consumption for 256GB case. However, the difference is very small, and the ADAM-augmented system consumes better power in other cases.

*3) Total Cost of Ownership:* Total cost of ownership (TCO) shows a similar tendency to power consumption. 2DPC with fewer servers required for scale-out shows better TCO than 1DPC. ADAM, which does not require scale-out until 16 ADAM cards are added, has obviously better TCO than both baselines. As in power consumption, when the k-NN search feature vector data size is small, TCO is relatively large because a ADAM card should be added to the existing server. ADAM can achieve up to 68.5% and up to 49.6% less TCO compared to the 2DPC baseline, respectively.

## VI. Conclusion

As the importance of data analytics acceleration is growing, this paper proposes a k-NN search, a representative data analytics operation, acceleration scheme using ADAM in DB server. Analysis shows that k-NN search turns out to be a memory-intensive workload suitable for ADAM processing. Based on the analysis, k-NN search is accelerated by inserting ADAM cards in the DB server. Simulations on various k-NN search feature vector data sizes report that the execution time, power consumption, and TCO are improved by 56.7%, 67.3%, and 68.5%, respectively, compared to the baseline system (1DPC case). When comparing a high-end CPU model supporting DDR5, even in this case, performance can be improved up to 27.4%. It is expected that various data analytics operations other than k-NN search can be accelerated through ADAM.

## References

[1] Z. D. Stephens *et al.*, "Big data: astronomical or genomical?" *PLoS biology*, vol. 13, no. 7, p. e1002195, 2015.

[2] B. Gu *et al.*, "Biscuit: A framework for near-data processing of big data workloads," *Proc. ISCA*, 2016.

[3] A. Boroumand *et al.*, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *Proc. ASPLOS*, 2018, pp. 316–331.

[4] S. Williams *et al.*, "Roofline: an insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[5] "AWS AQUA," https://aws.amazon.com/redshift/features/aqua/, 2021/11/22.

[6] "BigStream," https://https://bigstream.co/, 2021/11/22.

[7] "Samsung SmartSSD," https://samsungsemiconductor-us.com/smartssd/, 2021/11/22.

[8] "Hyperacceleration with Bigstream Technology," https://info.bigstream.co/hyperacceleration-with-bigstream-technology, 2021/11/22.

[9] T. J. Ham *et al.*, "Genesis: a hardware acceleration framework for genomic data analysis," in *Proc. ISCA*. IEEE, 2020, pp. 254–267.

[10] Y. Wang *et al.*, "A scalable fpga engine for parallel acceleration of singular value decomposition," in *Proc. ISQED*. IEEE, 2020, pp. 370–376.

[11] S. Salamat *et al.*, "Nascent: Near-storage acceleration of database sort on smartssd," in *Proc. FPGA*, 2021, pp. 262–272.

[12] "Spark SQL," https://spark.apache.org/sql/, 2021/11/22.

[13] J. Ahn *et al.*, "A scalable processing-in-memory accelerator for parallel graph processing," in *Proc. ISCA*, 2015, pp. 105–117.

[14] "Intel Xeon Gold 6256 Processor," https://ark.intel.com/content/www/us/en/ark/products/198655/intel-xeon-gold-6256-processor-33m-cache-3-60-ghz.html, 2021/11/22.

[15] "Apache Spark," https://spark.apache.org/, 2021/11/22.

[16] "DGX-A100 Datasheet," https://images.nvidia.com/aem-dam/Solutions/Data-Center/nvidia-dgx-a100-datasheet.pdf, 2021/11/22.

[17] "SIDEWINDER-100," https://fidus.com/wp-content/uploads/2019/01/Sidewinder$_{Data_Sheet.pdf}$, $2021/11/22$.