Accelerating Data Analytics near Memory: A k-NN Search Case Study

MEMORY FOREST

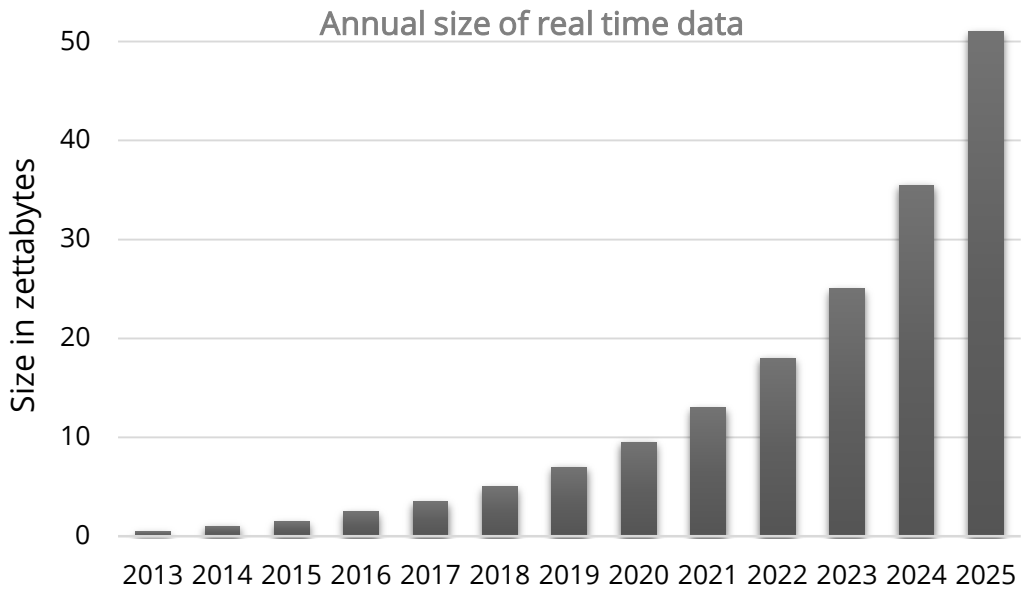
Minho Ha, Joonseop Sim, Jungmin Choi, Donguk Moon, Taeyoung Ahn, Byungil Koh, Eui-Cheol Lim, and Kyoung Park

We Do Technology | SK hynix

Motivation

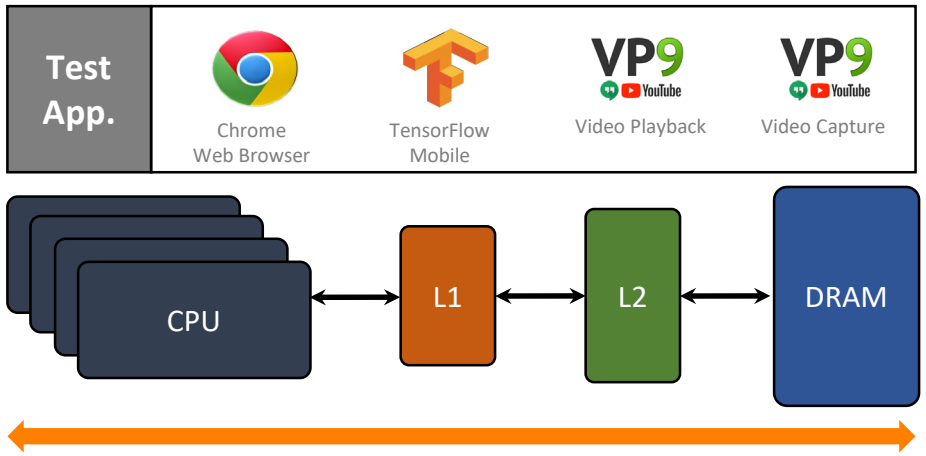
- **Growing calls for an efficient approach to process massive volumes of data**

Data is growing explosively



Source: https://www.statista.com/statistics/949144/worldwide-global-datasphere-real-time-data-annual-size/

Processor-centric architecture suffers from data movement



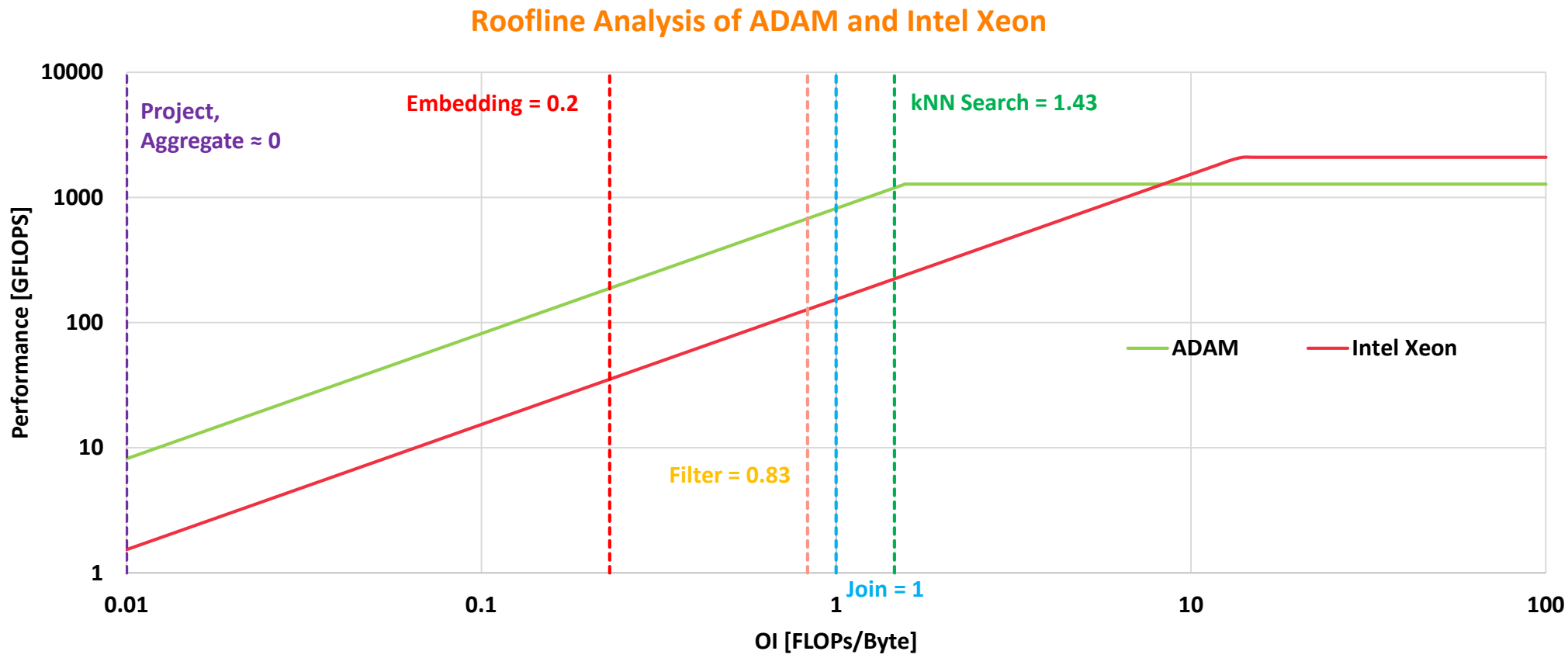
62.7% of the total system energy is spent on data movement
→ Performance Bottleneck

Source: Amirali Boroumand, "GoogleWorkloads for Consumer Devices: Mitigating Data Movement Bottlenecks," ASPLOS`18

We need to develop a domain-specific system suitable for processing memory-intensive workload!

Data Analytics Application: SQL & k-NN Search

- Data analytics is a typical example of memory-intensive workload
  - Roofline analysis result shows that data analytics operations, SQL operations & kNN, have very low operational intensity
  - High-bandwidth memory support is required to handle these operations

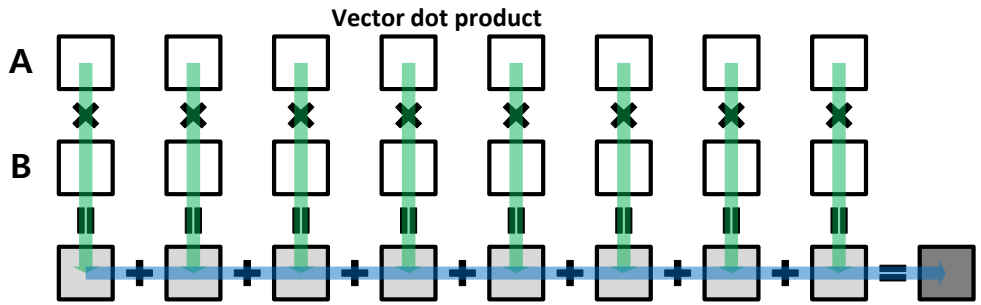


Case Study: k-NN Search

- k-NN search is an algorithm that finds the k nearest items
- HW characteristics of k-NN search: *Need large memory capacity & high memory BW*
  - It requires a lot of memory access while the operation is simple
  - Correlation between performance and memory BW is high
  - Experimental observation shows that performance is saturated due to memory BW bound
    → Performance can be improved linearly by expanding the memory channel
  - Hundreds of GB or more are required per server to store the DB used in the industry

k-NN search formula

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$



k-NN search performance evaluation results (AVX-512)

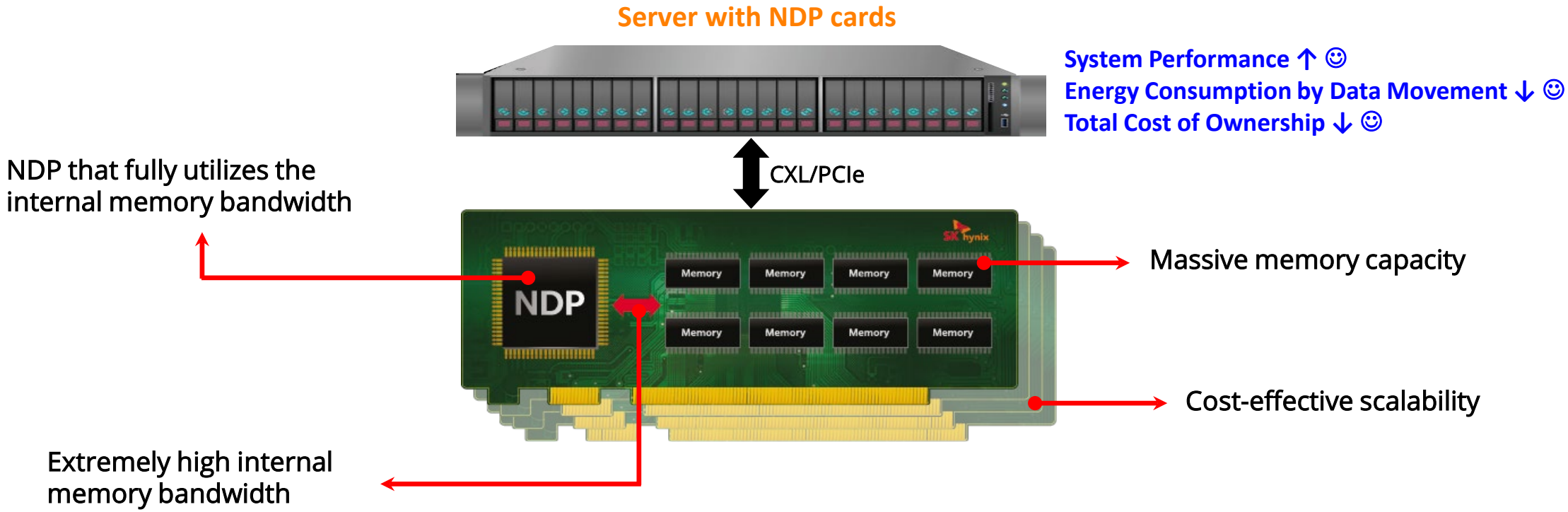
Thread #	Measured Value		Correlation: 0.993	
	Performance (ms)	Max Memory BW (GB/s)	Performance	Memory BW
1	130.06	15.2	**1**	**1**
4	41.52	51.93	**3.13**	**3.42**
16	16.58	133.91	**7.84**	**8.81**
32	14.23	156.26	**9.14**	**10.28**
64	16.17	155.48	**8.04**	**10.23**

* HW environment
- CPU: 2x Intel Icelake Xeon Gold 6338 2.0GHz 32 Core
- DRAM: 1TB DDR4 (16x 64GB DDR4-3200)
- SSD: 2x SKH PE8010 1.92TB, 2x SS PM1733 1.92TB (PCIe Gen4)
- XILINX Sidewider-100 (ARM Cortex A53 4 Core, 32GB DRAM, PCIe Gen3x8)

* SW parameters
- Dimension: 256
- Batch size: 128
- DB size: 1.6M
- Precision: 4Bytes

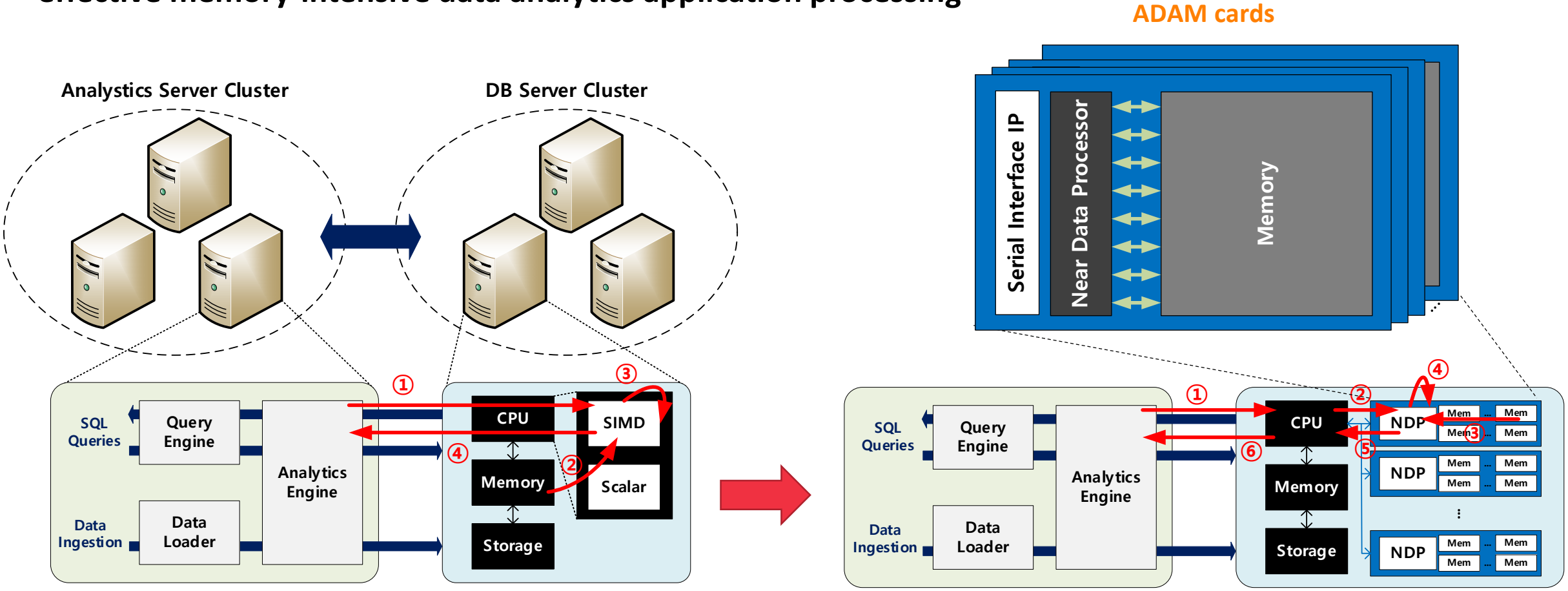
Possible Solution: Near Data Processing (NDP)

- **NDP is a concept of processing data right next to memory where the data is located**



Accelerating Data Analytics Near Memory (ADAM)

- We propose *Accelerating Data Analytics Near Memory (ADAM)*, which takes the NDP concept for effective memory-intensive data analytics application processing



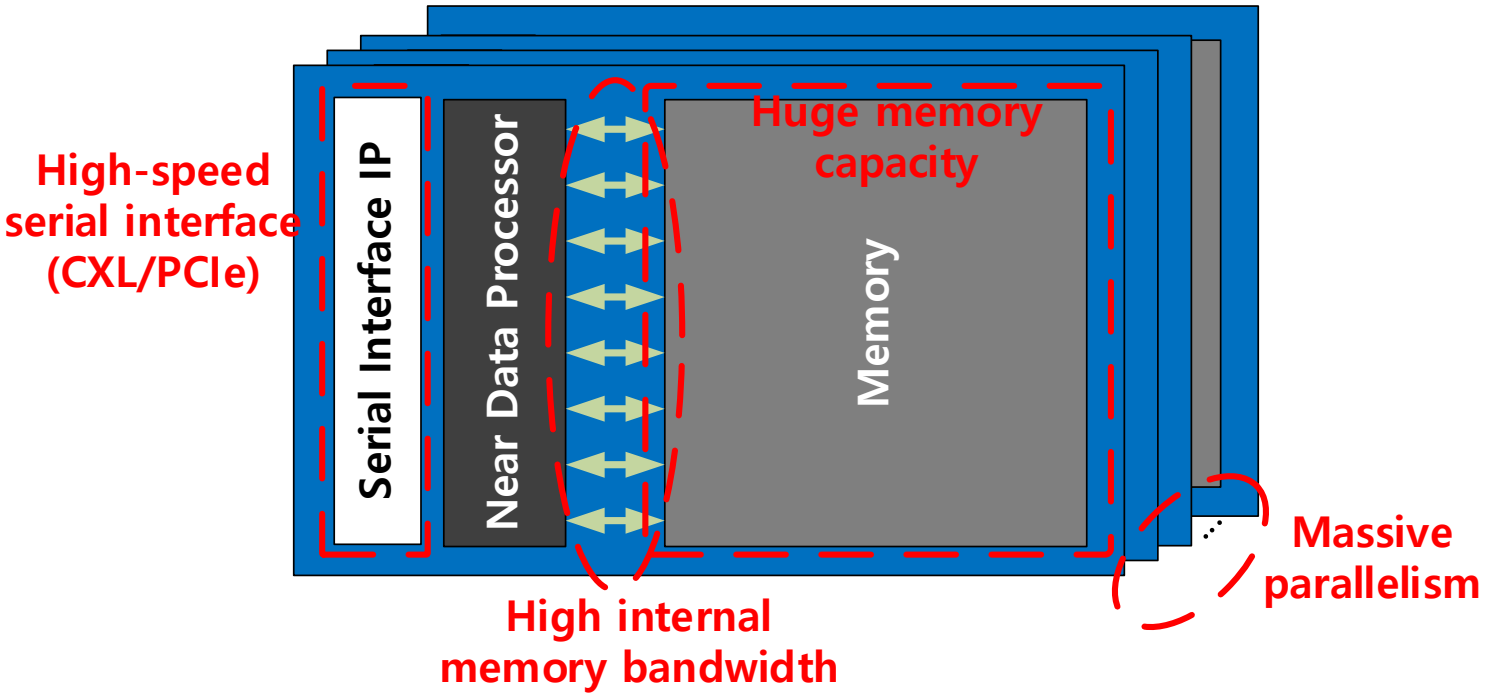
Baseline k-NN acceleration system

ADAM-augmented k-NN acceleration system

ADAM Card Architecture

- **ADAM is an ultimate solution for memory-intensive workloads**
  - **Benefit1) High performance for memory-intensive workload**
  - **Benefit2) Low cost & power**

ADAM cards



Specs

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

* Above specifications are performance assumptions at the future product level

ADAM API

- **ADAM cards communicate with the Host through the ADAM APIs**

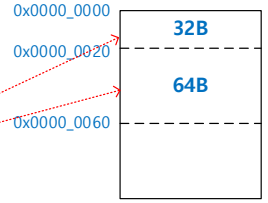
Memory management APIs

```
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);
```

< Pseudo code >

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

< ADAM memory map >



< ADAM memory >

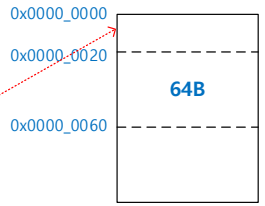
ADAM_malloc()

```
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);
```

< Pseudo code >

Pointer	Physical address	Size
~~0~~	~~0x0000_0000~~	~~32B~~
1	0x0000_0020	64B

< ADAM memory map >



< ADAM memory >

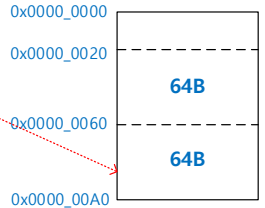
ADAM_free()

```
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);
```

< Pseudo code >

Pointer	Physical address	Size
0	~~0x0000_0000~~ 0x0000_0060	~~32B~~ 64B
1	0x0000_0020	64B

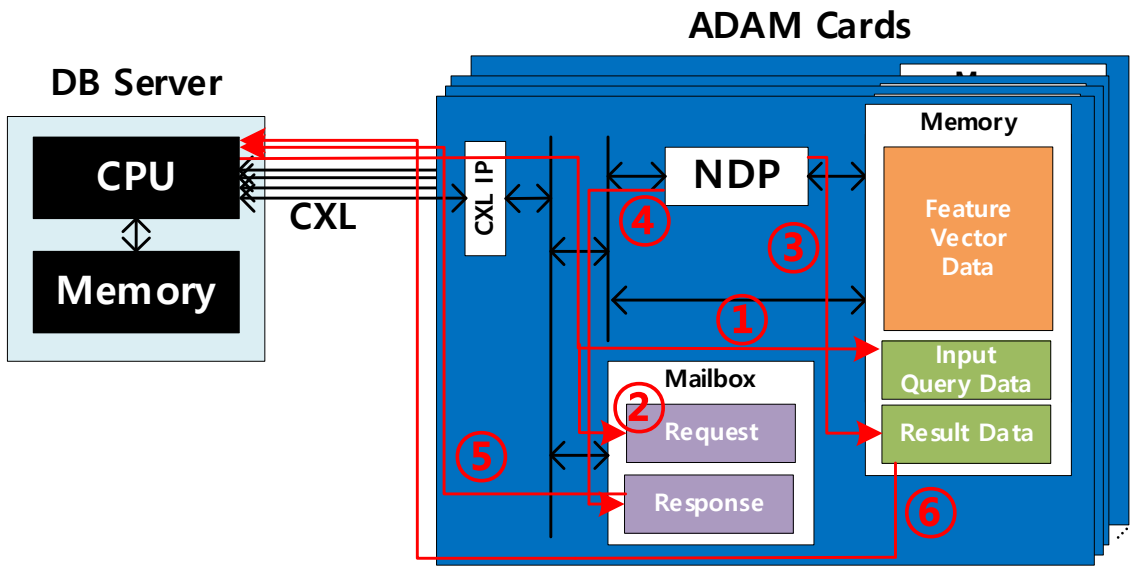
< ADAM memory map >



< ADAM memory >

ADAM_realloc()

k-NN Search Acceleration API (KNN_scan())

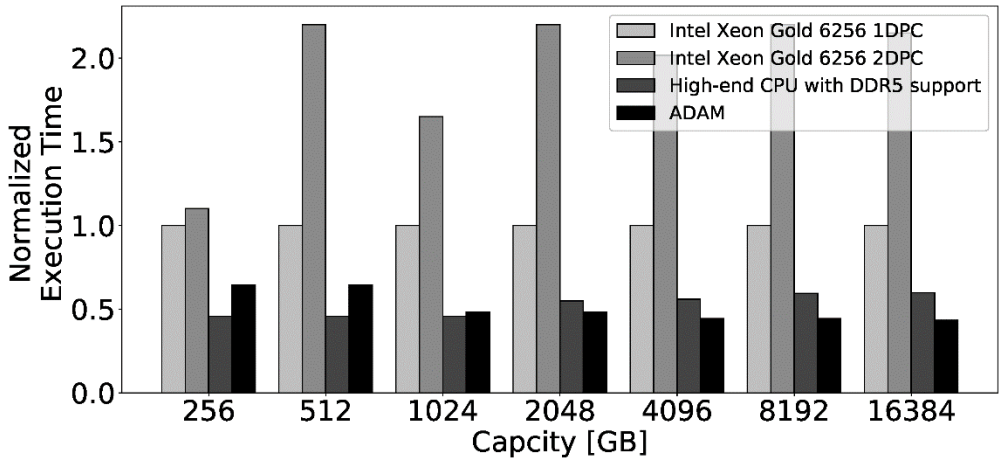


API functions	Format
Memory allocation	`ADAM_malloc (int size)`
Free memory	`ADAM_free (float *index)`
Re-allocation	`ADAM_realloc (float *index, int size)`
Request k-NN search and read results	`KNN_scan (int table_ID, int feature_ID, int K, float* array, float *index)`

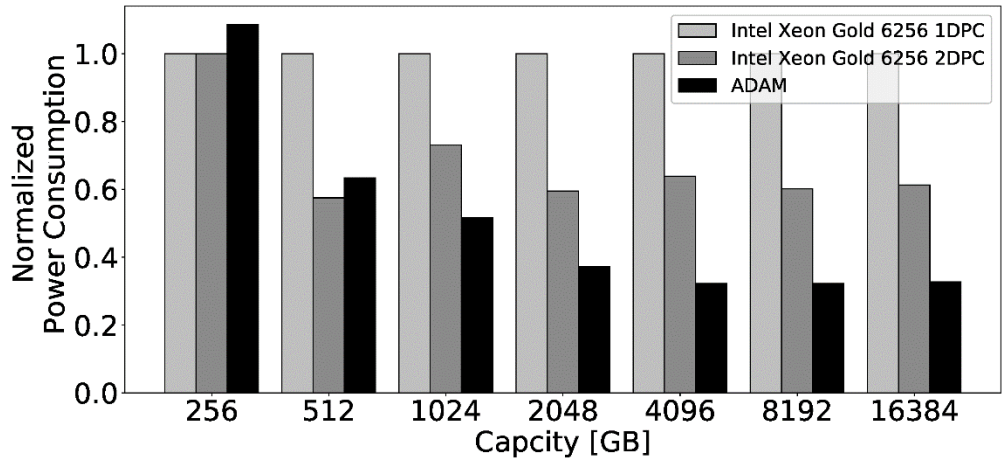
Evaluation Results

- **System-level simulation 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**

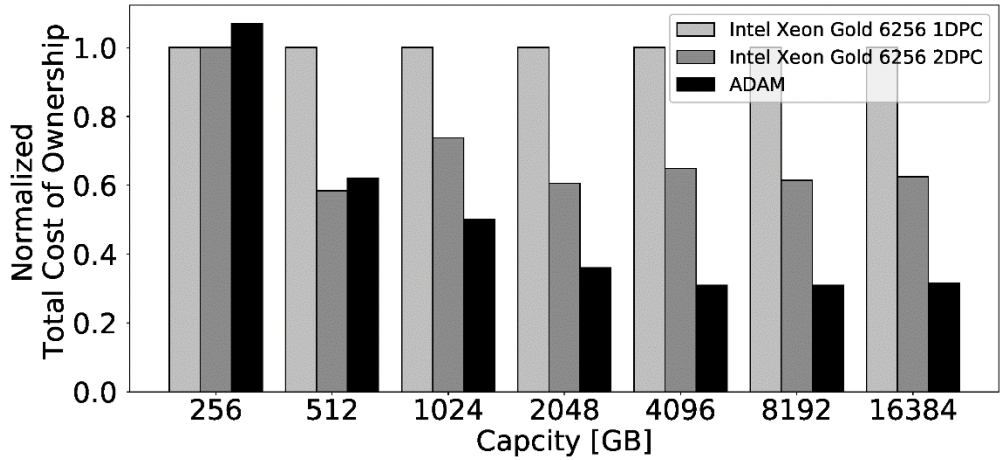
Performance



Power



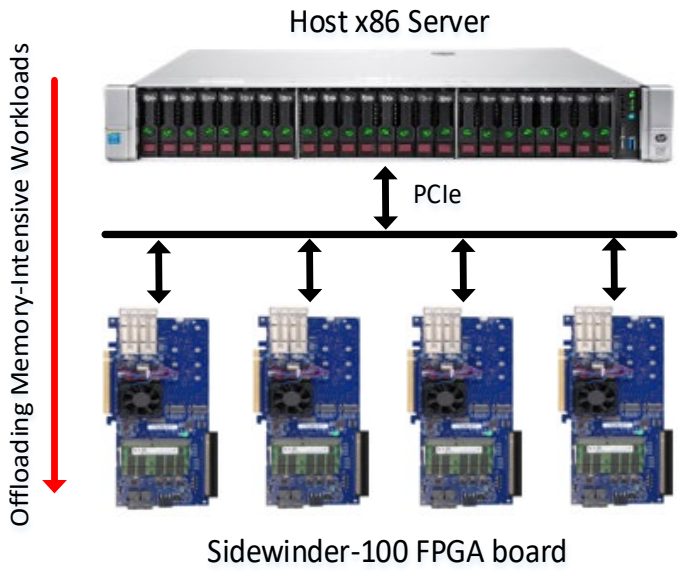
Cost



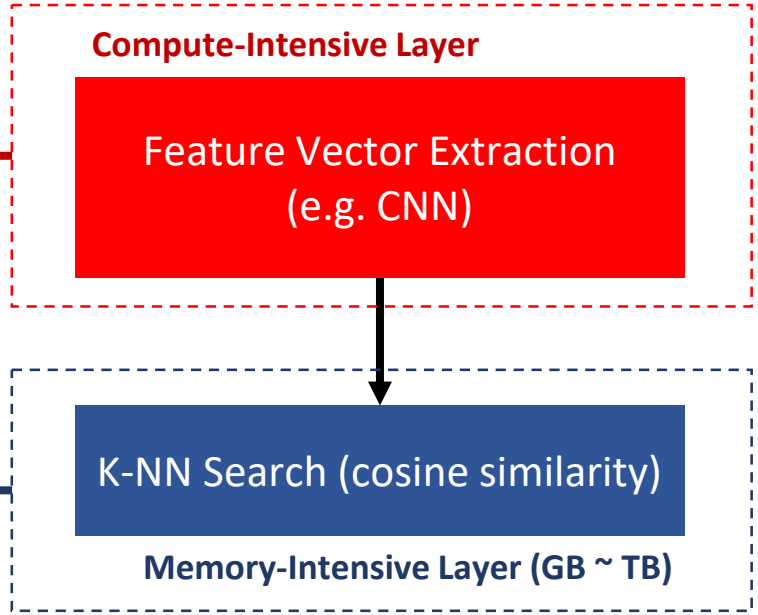
1st Proof of Concept

- We implemented a commercial FPGA-based PoC and actually ran the k-NN search acceleration
  - Host CPU manages the entire applications and offloads memory-intensive workloads to FPGA PoCs
  - Offloaded memory-intensive workloads are accelerated by optimized HW and kernels
  - The overall performance improves as the number of FPGA PoC cards increases up to 6.52x

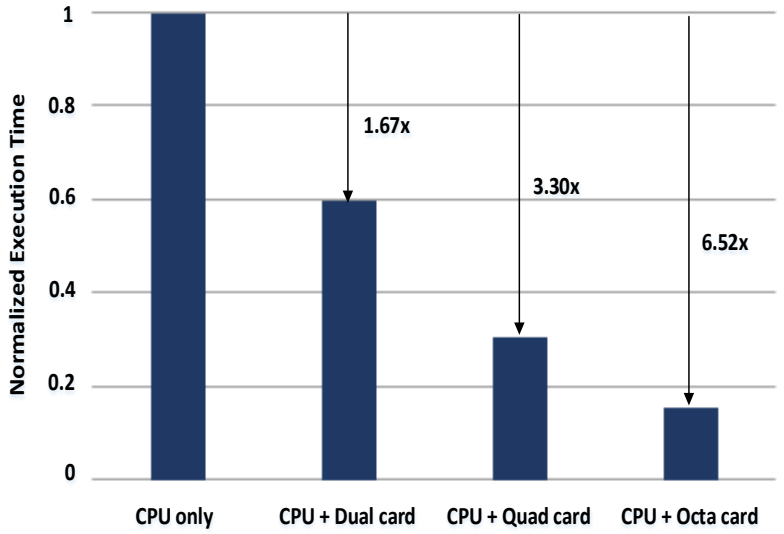
Demo system with FPGA PoC



K-NN search workload



PoC Performance



Future Plan

- In addition to k-NN search, we plan to increase the coverage of NDP solution with *other machine learning functions, Apache Spark SQL operations (aggregate/filter) and data compression/decompression*

- *Additional hardware accelerator modules* are being planned to improve processing performance

- Considering the explosive growth of processed data, we plan to ultimately maximize memory capacity and processing performance by loading multiple NDP solution cards in the *Disaggregated Memory Pool*

Conclusion

- A domain-specific system is required to effectively handle memory-intensive workloads such as data analytics applications
- Accordingly, we propose *Accelerating Data Analytics near Memory (ADAM)*, a solution using the near data processing (NDP) concept
- As a result of accelerating k-NN search, a representative data analytics application, through a system equipped with ADAM, performance is improved by 56.7%, power consumption by 67.3%, and cost by 68.5% compared to the existing CPU SIMD acceleration
- We implemented the first FPGA-based PoC and plan to further improve it