Design and Implementation of Data-Intensive Application using Memory Expansion Device

HooYoung Ahn, SeonYoung Kim, Yoo-mi Park, and Woojong Han
Supercomputing Technology Research Center
Artificial Intelligence Computing Research Laboratory
Electronics and Telecommunications Research Institute
{ahnhy, seonyoung8436, parkym, woojong.han}@etri.re.kr

***Abstract*—Modern intelligent applications utilizing data-driven analysis have gained significant attention. As these applications rely on analyzing large-scale data for improved performance, they demand substantial high-performance memory. However, current computer architectures often lack the capacity to provide enough fast memory, leading to performance degradation in such applications. To address these limitations, memory expansion devices are being developed that connect to host servers through the new cache coherent interconnects like CXL. These devices offer additional memory and hardware acceleration capabilities to enhance performance.**

In this paper, we propose a novel method to accelerate data-intensive MPI applications in a CPU-FPGA heterogeneous computing environment, employing memory expansion devices called MEX. Our approach leverages MEX and MPI to enable near-memory processing by utilizing MEX as shared computing resources. To the best of our knowledge, this is the first study to combine memory expansion devices and MPI for accelerating data-intensive MPI applications. We implement a content-based image similarity search system using MPI and MEX and verify the feasibility of our proposed methods.

***Index Terms*—Message Passing Interface, Memory Expansion Device, Data-Intensive Application**

I. Introduction

With the proliferation of the artificial intelligence and big data, a data-driven approach is utilized in numerous domains such as weather forecasting, protein structure analysis, autonomous driving, and XR (eXtended Reality). The latest intelligent applications using the data-driven approach utilize massive data for their own purpose. For instance, deep learning applications leverage large amounts of data to improve the accuracy and reduce the overfitting.

Since data-intensive applications handle a huge amount of data, the capacity of memory on a computing node directly affects application performance. Unfortunately, however, there is a clear limit on the memory capacity due to the architectural limitations of the hardware. To overcome these limitations, leading memory vendors such as Samsung reveal their own FPGA (Field Programmable Gate Array) based memory expansion devices [1] [2]. These memory expansion devices employ cache-coherent interconnect such as CXL [3], CCIX [4], Gen-Z [5] instead of the PCIe to achieve higher bandwidth and lower latency. In addition, they provide hardware acceleration that leverages near-memory processing (NMP) through FPGAs to increase the computing capability of computing nodes [6].

Another representative memory expansion device is MEX (**M**emory **EX**pander) [7] connected to a computing node via CXL. MEX is an expansion card that provides additional memory named MEMEM (**ME**x **MEM**ory) and accelerator for hardware acceleration named MEACC (**ME**x **ACC**elerator). In this paper, we propose a novel method to improve the performance of k-NN (K-Nearest Neighbor) which is the key operation of the typical data-intensive application, on a CPU-FPGA heterogeneous computing environment adopting a prototype version MEX.

We also employ MPI (Message Passing Interface) to improve the performance of large-scale similarity search applications. MPI is a type of inter-process communication method and a standard specification for message passing between processes in distributed memory architecture [8], [9]. It increases the parallel processing performance and the scalability of processors in a multi-node cluster. We use MVAPICH [10] which is the optimized implementation for InfiniBand to parallelized k-NN in a multi-node cluster.

We implement the k-NN accelerator which uses our proprietary MPI-MEX library. MPI-MEX library is newly devised to reduce the network communication overhead by replacing the send and receive operations with the read and write operations of MEMEM. It also increases the processing speed of MPI operations by parallel processing the distance computation and sort using the MEACC. The proposed method could be a solution to improve the system scalability of HPC environment.

The major contributions of our work are as follows:

- We survey various data-intensive applications and select k-NN operation as the target use case for MEX.
- We newly devise and implement MPI-MEX, a library that improves MPI communication performance with Near Memory Processing using MEX. It uses MEMEM and MEACC usefully for the communication and operations of MPI.
- We implement the k-NN accelerator which can be used in various data-intensive applications using the MPI-MEX library.
- We implement the content-based image similarity search system using the MPI-MEX library and k-NN accelerator and verify the the feasibility of the proposed method.

The rest of this paper is organized as follows. In Section II, we present the background of our work. In Section III, we introduce related work. Section IV describes the proposed MPI-MEX library and the k-NN accelerator. The implementation details of the content-based image similarity search system are presented in Section V. Finally, Section VI concludes this paper.

II. Background

A. Memory Expansion Device

As more and more cores are integrated into a single processor chip to fulfill the demand for more powerful computing capability, the required memory bandwidth and capacity are also increasing. However, as the hardware limitations of each computing node impose certain limits on the memory bandwidth and capacity, there is a demand for a new memory interface that breaks the existing memory hierarchy.

To meet this demand, the next-generation interconnects such as CXL and CCIX have emerged that allow access to the expanded memory located between storage and main memory while maintaining the cache coherence. These interconnects provide the ability for host CPUs to maintain cache coherence between heterogeneous accelerators as well as memory, and are in the spotlight as a technology that efficiently utilizes computing resources provided by heterogeneous accelerators. As interconnects continue to advance, there is an emergence of memory expansion devices that make use of these improved interconnects. Representative examples of memory expansion devices include CXL Memory Expander [2] and MEX [7]. They not only merely provide expanded memory but also provide the hardware acceleration by leveraging the concept of near-memory processing.

B. K-Nearest Neighbor

k-NN is a key operation of the similarity search [11]. It is widely used in deep learning, machine learning [12], and data mining [13]. Image recognition [14], anomaly detection, text classification, bioinformatics are representative examples of applications that utilize k-NN operation [15]. To find the similar items, it first measures the similarity between the target item and the candidate items based on a distance metric. After computing all the distances, it sorts the distances and selects Top-k items. In this way, k-NN makes a large workload during the similarity search process. Since the k-NN is quite compute- and data-intensive operation, the performance of similarity search is highly dependent on the k-NN processing performance.

III. Related Work

In this section, we review the state-of-the-art technologies to improve the performance of data-intensive applications and the researches on the near-memory processing. Chen *et al.* [16] proposed pattern matching accelerators for the genome analysis platform to improve the performance. They made the sequence searching operations be hardware-friendly by mapping the original sequences into the high-dimensional space. Peng *et al.* [17] proposed the similarity search accelerator to improve the performance of molecular databases. They devised FPGA-based graph traversal engine and increased the search speed and accuracy. Kalantar *et al.* [18] presented a FPGA-accelerator performing time series similarity prediction using deep learning models. They improve the performance of real-time analysis system by implementing both the edge and cloud accelerators.

Singh *et al.* [19] tried to tackle the data movement bottleneck between the processor and memory using FPGA with high-bandwidth memory. By leveraging the FPGA, they accelerated the genome analysis and the weather modeling. Herruzo *et al.* [20] proposed the memory-centric architecture that reduces the unpredictable memory access which cannot benefit from the cache hierarchies in processors. Asgari *et al.* [21] proposed the solution for reducing the irregular random memory accesses in scientific computing and graph analytics. The proposed approach minimized data movement by maximizing the parallelism of the memory accesses while eliminating redundant memory accesses for the large embedding table lookup. Ke *et al.* [22] proposed the NMP platform to improve the performance of recommendation systems. They improved the performance of embedding operations using the specialized FPGA board. Lee *et al.* [6] improved the performance of the scan operation of database management systems using the NMP solution for data analysis.

IV. Proposed Method

In this section, we introduce the MPI-MEX library, which enables the sharing of memory and accelerators of the MEX device among MPI ranks. Furthermore, we present the k-NN accelerator developed using the proprietary MPI-MEX library.

A. MPI-MEX Library

MPI-MEX is the library that provides MPI communication method using MEX memory which is named MEMEM as a communication buffer.It also offloads several MPI operations (MPI-OPs) to the MEX accelerator named MEACC.

1) Point-to-Point Communication: Fig. 1 shows the comparison of point-to-point (P2P) communication of the conventional MPI and MPI-MEX. As shown Fig. 1 (a), the sender P_0 puts its data 5 in an envelope and sends it to the receiver P_1 through the network communication methods. Fig. 1 (b) shows the P2P communication method of MPI-MEX. Currently, the *Send* and *Recv* of MPI-MEX provides the functionality of P2P MPI communication by write and read MEMEM. This feature has been implemented by managing the address of MEMEM as a shared memory resource. ① The sender P_0 sends the data by writing data into the MPI-Data-Envelope of the MEMEM that is used for storing data and envelope information of MPI ranks. ② The sender P_0 receives the pointer *ptr of the above mentioned MEMEM and saves it into the shared memory. ③ The receiver P_1 attaches itself to shared memory and finally reads the data after checking the information in the envelope.

2) Collective Communication: MPI-Reduce is an operation in which the root rank P_0 receives the computation result by performing MPI-OPs on the data of the MPI ranks in the same communicator. This function is designed as shown in Fig. 2.

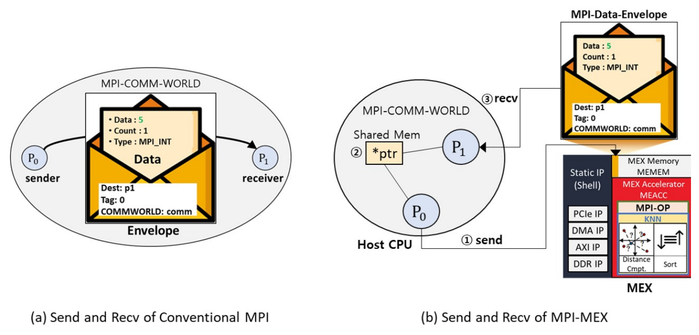
Fig. 1: P2P Communication of MPI-MEX

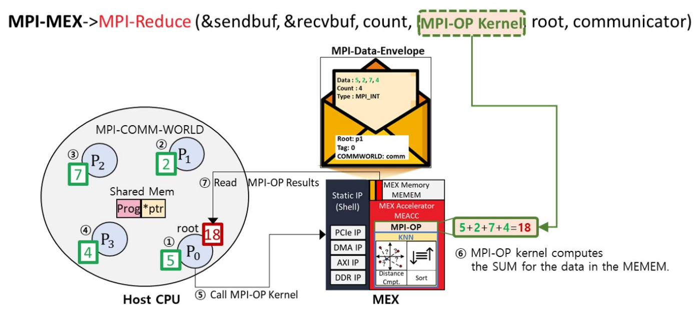
Fig. 2: Collective Communication of MPI-MEX

MPI-Reduce of MPI-MEX is similar to MPI-Reduce of the conventional MPI, but the biggest difference is that the MPI-OP kernel's binary file name is placed in the parameter position where the MPI-OP is placed. In this figure, we show that each of the four MPI ranks has one number, and the sum of these numbers is computed using the MPI-OP kernel developed in MEACC not in the CPU. First of all, P_0 prepares two shared memory. One is for storing the pointer *ptr of MPI-Data-Envelope, and the other is for the monitoring variable $Prog$ that is used for checking the write progress of MPI ranks. MPI-Reduce of MPI-MEX goes through seven stages. ① P_0 writes data 5 stored in its send buffer to MEMEM using the shared memory pointer *ptr in the yellow box. Then it assigns 3 which is the number excluding itself in the total number of ranks to shared variable $Prog$ in the pink box. After that, it monitors $Prog$ variable until it becomes 0 to see if there are any remaining write operations. ② P_1 writes data 2 stored in its send buffer to MEMEM. Then it subtracts 1 from $Prog$ variable to indicate that its write operation is completed. ③ P_2 writes data 7 stored in its send buffer to MEMEM. Then it subtracts 1 from $Prog$ variable to indicate that its write operation is completed. ④ P_3 writes data 4 stored in its send buffer to MEMEM. Then it subtracts 1 from $Prog$ variable to indicate that its write operation is completed. ⑤ When the $Prog$ variable becomes 0, P_0 confirms that all MPI ranks have finished their write operations and calls the MPI-OP kernel of MEACC. ⑥ Then the MPI-OP kernel computes the sum of data of P_0 to P_3 stored in the MEMEM in a parallel manner and derives 18 as the sum of 5, 2, 7, 4 and stores the result in MEMEM area which is marked in red color. ⑦ Finally, P_0 reads and copies the computation result to the host memory.

Fig. 3 shows the sequence diagram of MPI-Reduce in the heterogeneous computing environment. We represent the sequence diagram assuming an environment using Xilinx Runtime (XRT) and PCIe shell. When the main program calls MPI-Reduce, P_0 requests and receives a shared memory and a semaphore from the OS and attached to the shared memory. After that, P_0 requests the XRT to allocate an array in MEMEM. The request is delivered to the shell of MEX, and the shell allocates an array in MEMEM and informs the XRT of the physical address of the array. Then XRT converts the physical address of MEMEM into the logical address and informs the pointer *ptr of the array to P_0. In addition, *ptr is stored in a shared memory to share the address of the array in MEMEM with other ranks P_1, P_2, and P_3. P_0 initializes the $Prog$ variable and uses it to monitor whether other ranks have written all data to MEMEM. The initialization value is set to a number excluding itself from the size of MPI-COMM-WORLD. In this figure, since four MPI ranks are in one MPI-COMM-WORLD, the corresponding variable value is initialized to 3. After this initialization step, MPI-Barrier is called once to make other ranks start processing after the initialization of P_0. After the MPI-Barrier, P_0 writes its own data to the 0^{th} index of the array in MEMEM. After that, the progress of the write operation of other ranks is monitored using $Prog$ variable.

P_1, P_2 and P_3, which are not root ranks, wait until the initialization step of P_0 is finished. After recognizing that initialization is completed, they attach themselves to the shared memory created by P_0, and gets a semaphore to manage the critical section of the shared memory from Operating Systems. After that, P_1 writes data stored in its send buffer to the 1^{st}, P_2 writes its data to 2^{nd}, and P_3 writes its data to 3^{rd} of the array in MEMEM. At this time, when the write operation for each rank is completed, they subtract 1 from the $Prog$ variable respectively. P_0 continues to monitor the $Prog$ variable, and when the value becomes 0, it confirms that all ranks have finished the write operations. Then P_0 calls the MPI-OP kernel to compute the data of P_0 to P_3 stored in MEMEM and finally reads the computation result from MEMEM.

Algorithm 1 describes the MPI-Reduce using MEX. The input parameters of the algorithm is as follows. s is the send buffer, r is the receive buffer, op is the type of MPI operation, $root$ is the id of the root process. Line 2 is the stage of initializing the result variable r and write progress check variable $Prog$. Lines 3–11 is the process of root rank. The root rank creates shared memory in MEMEM, allocates $*ptr$ and initializes the $Prog$ variable to the MPI-COMM-WORLD size minus 1. Then, it call MPI-Barrier and writes its data from the send buffer to $*ptr$. Afterward, it waits for the other MPI ranks to complete their write operations and invokes MEACC's MPI-OP kernel when all ranks have finished their write operations. Lines 12–19 is the process of non-root rank. Non-root ranks attach to the shared memory and write data to

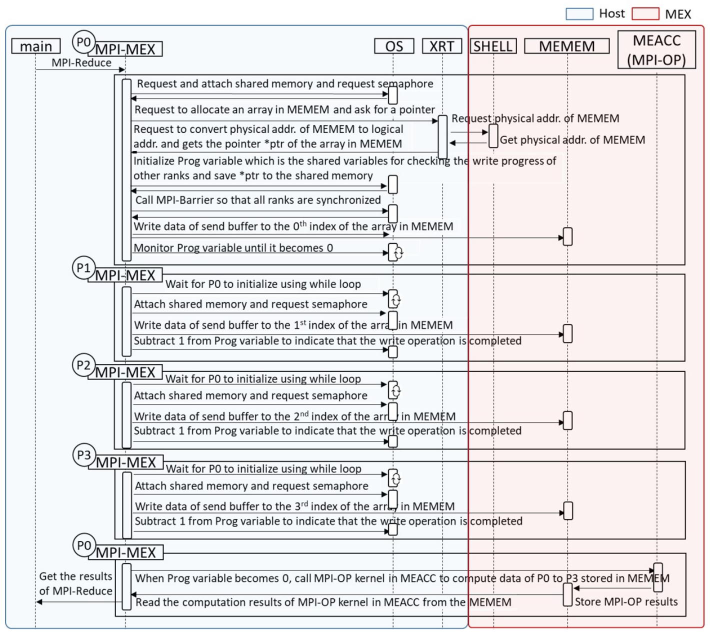
Host
MEX
main
P0
MPI-MEX
OS
XRT
SHELL
MEMEM
MEACC (MPI-OP)
MPI-Reduce
Request and attach shared memory and request semaphore
Request to allocate an array in MEMEM and ask for a pointer
Request physical addr. of MEMEM
Request to convert physical addr. of MEMEM to logical addr. and gets the pointer *ptr of the array in MEMEM
Get physical addr. of MEMEM
Initialize Prog variable which is the shared variables for checking the write progress of other ranks and save *ptr to the shared memory
Call MPI-Barrier so that all ranks are synchronized
Write data of send buffer to the 0th index of the array in MEMEM
Monitor Prog variable until it becomes 0
P1
MPI-MEX
Wait for P0 to initialize using while loop
Attach shared memory and request semaphore
Write data of send buffer to the 1st index of the array in MEMEM
Subtract 1 from Prog variable to indicate that the write operation is completed
P2
MPI-MEX
Wait for P0 to initialize using while loop
Attach shared memory and request semaphore
Write data of send buffer to the 2nd index of the array in MEMEM
Subtract 1 from Prog variable to indicate that the write operation is completed
P3
MPI-MEX
Wait for P0 to initialize using while loop
Attach shared memory and request semaphore
Write data of send buffer to the 3rd index of the array in MEMEM
Subtract 1 from Prog variable to indicate that the write operation is completed
P0
MPI-MEX
Get the results of MPI-Reduce
When Prog variable becomes 0, call MPI-OP kernel in MEACC to compute data of P0 to P3 stored in MEMEM
Read the computation results of MPI-OP kernel in MEACC from the MEMEM
Store MPI-OP results

Fig. 3: Sequence Diagram of MPI-Reduce

the location offset by itself from $*ptr$ and subtract 1 from the $Prog$ variable. The final result of MPI-Reduce is returned in Line 20.

Algorithm 1 MPI-Reduce using MEX

```
 1: procedure MPI-REDUCE($s, r, op, root$)
 2:     $r \leftarrow \emptyset$ $Prog \leftarrow -999$
 3:     if myRank is $root$ then
 4:         $*ptr \leftarrow$create Shared-MEMEM
 5:         $Prog \leftarrow sizeof$(MPI-COMM-WORLD)-1
 6:         call MPI-Barrier
 7:         write $s$ to $*ptr$ of Shared-MEMEM
 8:         while $Prog \neq 0$ do
 9:             ;                       // wait for all write operations
10:         end while
11:         $r \leftarrow$ call MPI-OP kernel in MEACC
12:     else
13:         while $Prog$ is -999 do
14:             ;                       // wait for initialization of root
15:         end while
16:         attach Shared-MEMEM
17:         write $s$ to $*ptr$ + $sizeof(s)$ * myRank
18:         $Prog$--
19:     end if
20:     return $r$
21: end procedure
```

B. k-NN Accelerator using MPI-MEX

The typical data process of a similarity search application is as follows [23]. In the first step, **Feature Extraction**, the feature vectors are extracted from candidate items. The second step is **Query Generation**, this step extracts feature vectors from the query item. The third step is the **k-NN search**. In this step, the k-NN operator computes the distance among feature vectors of dataset and the query item using the distance metric. The fourth step is the **Reverse Lookup** step. This step finds actual items and shows the final Top-k item to the user.

The proposed k-NN accelerator of this paper finds Top-k items using MPI and MEX as follows. The MPI ranks receive a set of feature vector C and a query vector q as inputs and compute k-NN in a distributed and parallel manner using MEMEM and MEACC. The data processing flow consists of four main parts. In part I, which is the **Scatter and Broadcast** stage, it scatters candidate vectors and their indexes, and broadcasts the query vector to all the MPI ranks in the same MPI communicator. In part II which is the **Compute Distance** stage, each MPI rank computes and sorts the distance of candidate feature vectors and query vector using the distance computation and sort kernel of MEACC. In part III, each MPI rank sorts the distance in descending order. At this time, the important thing is to sort the index together in the order in which the distances are sorted. We named this stage as **Local Sort** because MPI ranks only sort the data in their own memory.

Part IV, the **Global Sort** stage, is the step to derive the global sort results by collecting and sorting the distances of different MPI ranks. In this stage, the MPI ranks communicate using the shared MEMEM as their communication buffer, and sort the distances using MEACC. For example, P_0 and P_1 write the local sorted distance to the shared MEMEM created by P_0 and P_0 reads the distance stored in Shared-MEMEM$_{01}$. Then P_0 performs the sorting by merging distances using the sort kernel of MEACC. P_2 and P_3 write the local sorted distance to the shared MEMEM created by P_2 in order to merge and sort their data, and P_2 reads the distance stored in Shared-MEMEM$_{23}$. Then P_2 performs the sorting by merging distances using the sort kernel of MEACC. Then P_0 and P_2 write the local sorted distance to the shared MEMEM created by P_0 in order to merge and sort their data, and P_0 reads the distance stored in Shared-MEMEM$_{02}$. Finally, P_0 performs the sorting by merging distances of the two MPI ranks and the global sort is completed.

V. IMPLEMENTATION

To demonstrate the effectiveness of our method in real data-intensive applications, we developed a content-based image similarity search system. As a hardware configuration, we use one x86 host server with AMD Xilinx Alveo U280 FPGA card which is the MEX prototype connected by PCIe bus. MPI ranks use the MEX memory as a communication buffer to communicate with each other by reading and writing that memory and they use MEX accelerator for the several MPI-OPs, distance computation, and sort. As a software configuration, we use MPI-MEX which is the custom MPI library for MEX and OpenCL for the communication between the host CPU and MEX. For the demonstration, we use 128 RGB images in 10 categories as the dataset. We made the dataset by collecting images with the large color difference since we used the color histogram as a feature vector in the demonstration. We implement the C++ based OpenCL kernel for the acceleration logic of the MEX accelerator. Given a query image, our system finds the Top-k similar images and displays the answer images on the monitor screen.

VI. Conclusion

In this paper, we propose a novel method to improve the performance of data-intensive application using MPI and MEX. To achieve our objectives, we first devise and develop the MPI-MEX library that leverages the memory of MEX as a communication buffer and the accelerators of MEX for accelerating various MPI operations, distance computation, and sort. The library greatly helps reduce MPI communication and computation costs.

Furthermore, we implemented the k-NN accelerator on a CPU-FPGA heterogeneous computing environment using the MPI-MEX library. Then, we implemented a content-based image similarity search system using the k-NN accelerator to demonstrate the feasibility of our approach and verified its effectiveness. The proposed method is expected to serve as a cornerstone for near-memory processing solutions in MPI.

Acknowledgment

This research was partly supported by the Super Computer Development Leading Program of the National Research Foundation of Korea(NRF) funded by the Korean government (Ministry of Science and ICT(MSIT)) (No. 2021M3H6A1017683) and Electronics and Telecommunications Research Institute (ETRI) grant funded by the Korean government. (No. 23RS1100, System Integration and Test for Highly Scalable Exascale MPI Using Emerging Memory Hierarchy)

References

[1] Sijung Yoo, Donghoon Kim, Yoon Mo Koo, Sujee Kim Wooju Jeong, Hyungjoon Shim, Won-Jun Lee, Beom Seok Lee, Seungyun Lee, Hyejung Choi, Hyung Dong Lee, et al. Structural and device considerations for vertical cross point memory with single-stack memory toward cxl memory beyond 1x nm 3dxp. In *2022 IEEE International Memory Workshop (IMW)*, pages 1–4. IEEE, 2022.

[2] SJ Park, H Kim, K-S Kim, J So, J Ahn, W-J Lee, D Kim, Y-J Kim, J Seok, J-G Lee, et al. Scaling of memory performance and capacity with cxl memory expander. In *2022 IEEE Hot Chips 34 Symposium (HCS)*, pages 1–27. IEEE Computer Society, 2022.

[3] Stephen Van Doren. Hoti 2019: compute express link. In *2019 IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 18–18. IEEE, 2019.

[4] Brad Benton. Ccix, gen-z, opencapi: Overview & comparison. In *OpenFabrics Workshop*, 2017.

[5] Won-ok Kwon, Song-Woo Sok, Chan-ho Park, Myeong-Hoon Oh, and Seokbin Hong. Gen-z memory pool system implementation and performance measurement. *ETRI Journal*, 44(3):450–461, 2022.

[6] Donghun Lee, Jinin So, Minseon Ahn, Jong-Geon Lee, Jungmin Kim, Jeonghyeon Cho, Rebholz Oliver, Vishnu Charan Thummala, Ravi shankar JV, Sachin Suresh Upadhya, et al. Improving in-memory database operations with acceleration dimm (axdimm). In *Data Management on New Hardware*, pages 1–9. 2022.

[7] Seon Young Kim, Hoo Young Ahn, and Yoomi Park. Mex: Cxl-based memory expander with hardware acceleration.

[8] Young Woo Kim, Myeong-Hoon Oh, and Chan Yeol Park. Multi-communication layered hpl model and its application to gpu clusters. *ETRI Journal*, 43(3):524–537, 2021.

[9] Seon-Young Kim WooJong Han HooYoung Ahn, YooMi Park. Research trends for improving mpi collective communication performance. *[ETRI] Electronics and Telecommunications Trends*, 37(6), 2022.

[10] DK Panda. Mvapich2: A high performance mpi library for nvidia gpu clusters with infiniband. In *GPU technology conference*, volume 2, page 2, 2013.

[11] Thomas Seidl and Hans-Peter Kriegel. Optimal multi-step k-nearest neighbor search. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 154–165, 1998.

[12] Byoung-Jun Park, Eun-Hye Jang, Myung-Ae Chung, and Sang-Hyeob Kim. Design of prototype-based emotion recognizer using physiological signals. *ETRI Journal*, 35(5):869–879, 2013.

[13] Jinyoung Moon, Youngrae Kim, Hyungjik Lee, Changseok Bae, and Wan Chul Yoon. Extraction of user preference for video stimuli using eeg-based user responses. *ETRI Journal*, 35(6):1105–1114, 2013.

[14] Mohammad Fazel Younessy Ghadikolaie, Ehsanolah Kabir, and Farbod Razzazi. Sub-word based offline handwritten farsi word recognition using recurrent neural network. *ETRI Journal*, 38(4):703–713, 2016.

[15] Jang-Hee Yoo and Mark S Nixon. Automated markerless analysis of human gait motion for recognition and classification. *Etri Journal*, 33(2):259–266, 2011.

[16] Hanning Chen and Mohsen Imani. Density-aware parallel hyperdimensional genome sequence matching. In *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 1–4. IEEE, 2022.

[17] Hongwu Peng, Shiyang Chen, Zhepeng Wang, Junhuan Yang, Scott A Weitze, Tong Geng, Ang Li, Jinbo Bi, Minghu Song, Weiwen Jiang, et al. Optimizing fpga-based accelerator design for large-scale molecular similarity search (special session paper). In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–7. IEEE, 2021.

[18] Amin Kalantar, Zachary Zimmerman, and Philip Brisk. Fpga-based acceleration of time series similarity prediction: From cloud to edge. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 2022.

[19] Gagandeep Singh, Mohammed Alser, Damla Senol Cali, Dionysios Diamantopoulos, Juan Gómez-Luna, Henk Corporaal, and Onur Mutlu. Fpga-based near-memory acceleration of modern data-intensive applications. *IEEE Micro*, 41(4):39–48, 2021.

[20] Jose M Herruzo, Ivan Fernandez, Sonia González-Navarro, and Oscar Plata. Enabling fast and energy-efficient fm-index exact matching using processing-near-memory. *The Journal of Supercomputing*, 77(9):10226–10251, 2021.

[21] Bahar Asgari, Ramyad Hadidi, Jiashen Cao, Sung-Kyu Lim, Hyesoon Kim, et al. Fafnir: Accelerating sparse gathering by using efficient near-memory intelligent reduction. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 908–920. IEEE, 2021.

[22] Liu Ke, Xuan Zhang, Jinin So, Jong-Geon Lee, Shin-Haeng Kang, Sukhan Lee, Songyi Han, YeonGon Cho, Jin Hyun Kim, Yongsuk Kwon, et al. Near-memory processing in action: Accelerating personalized recommendation with axdimm. *IEEE Micro*, 42(1):116–127, 2021.

[23] Xiucai Ye and Tetsuya Sakurai. Robust similarity measure for spectral clustering based on shared neighbors. *ETRI journal*, 38(3):540–550, 2016.