EcoFlex-HDP: High-Speed and Low-Power and Programmable Hyperdimensional-Computing Platform with CPU Co-processing

Yuya Isaka[†], Nau Sakaguchi[*], Michiko Inoue[†], and Michihiro Shintani[‡]
[†]*Nara Institute of Science and Technology, Ikoma, Japan*
[*]*San José State University, San José, USA*
[‡]*Kyoto Institute of Technology, Kyoto, Japan*
Email: isaka.yuya.iw6@is.naist.jp, shintani@kit.ac.jp

***Abstract*—Hyperdimensional computing (HDC) can efficiently perform various cognitive tasks efficiently by mapping data to hyperdimensional vectors with thousands to tens of thousands of dimensions. However, the primary operations of HDC—Bind, Permutation, and Bound—need to be executed more efficiently on a standard CPU platform. This study introduces a novel computational platform, EcoFlex-HDP, specifically designed for HDC. EcoFlex-HDP exploits the parallelism and high memory access efficiency of HDC operations to achieve low computation time and energy consumption, outperforming the CPU. Furthermore, it can work cooperatively with a CPU, enabling integration with existing software, providing flexibility to apply new algorithms, and contributing to the development of an HDC ecosystem. Through experimental evaluations with a Cortex-A9 processor, HDC operations were shown to be accelerated by a maximum of 169 times. Furthermore, EcoFlex-HDP was confirmed to improve the energy–delay product by up to 13,469 times when training an image recognition task. All source codes for our platform and experiments are available at https://github.com/yuya-isaka/EcoFlex-HDP.**

***Index Terms*—Hyperdimensional-computing, Domain-specific architecture, Computing platform**

I. Introduction

Hyperdimensional computing (HDC) constitutes a learning paradigm inspired by the information processing mechanisms of the human brain, encapsulating data in hyperdimensional vectors spanning thousands to tens of thousands of dimensions [1]. This methodology emulates the capacity of the human brain to process multidimensional information. By leveraging this attribute, HDC can preserve the richness and complexity of information when utilizing it in classification tasks [2]. Specifically, HDC calculates the degree of similarity between the hyperdimensional vectors of the input data and those of the pretrained classes. The system selects the class with the highest similarity for classification. A salient advantage of HDC is its computational efficiency; it requires fewer computational resources than conventional machine learning techniques such as deep neural networks or support vector machines. Consequently, it facilitates the rapid and energy-efficient execution of tasks. Furthermore, HDC algorithms are inherently conducive to parallelization [3]. This property makes the HDC well-suited for acceleration through multicore architectures and power optimization via circuit miniaturization. Consequently, HDC exhibits exceptional performance in cognitive tasks, even in environments with limited computational resources. This versatility enables its application across various fields [4]–[8].

Despite the many documented benefits of HDC, it still faces challenges that require resolution. HDC employs specialized arithmetic operations, henceforth referred to as HDC operations, that differ from standard arithmetic calculations and are not efficiently executed on a general-purpose CPU [9]. To facilitate the execution of HDC operations, an emulation strategy must be employed that combines multiple instructions from a conventional instruction set, resulting in increased computational load and energy consumption. In addition, because HDC primarily operates on hyperdimensional vectors, a mechanism for vector computation is indispensable. However, in energy-efficient devices, employing vector extension instructions such as arm-scalable vector extension instructions are not generally feasible. To address these challenges, research is being conducted to develop dedicated processors that can efficiently and expeditiously execute HDC operations. Various approaches exist, including processors optimized for specific recognition tasks and those capable of handling diverse tasks [4], [10]–[13]. However, these approaches are insufficient when viewed from the perspective of integration with existing software and adaptability to novel algorithms. Consequently, the onus falls on developers to implement additional mechanisms for pre- and post-processing, thereby increasing the developmental burden.

Against these challenges, we propose a computational platform named EcoFlex-HDP that surpasses CPU processing speed and energy efficiency, is compatible with existing software, and is adaptable to emerging algorithms. By implementing EcoFlex-HDP, the development of products that combine existing software assets with HDC becomes substantially easier, thereby promoting HDC-enabled software development. This promotion contributed to the growth of HDC ecosystems. Furthermore, by establishing an EcoFlex-HDP infrastructure capable of accommodating novel algorithms, the continuous incorporation and implementation of cutting-edge HDC research methodologies become feasible, catalyzing the research and development of innovative algorithms and specific application examples.

The CPU plays a crucial role in enabling the seamless integration of existing software assets. Accordingly, this study

979-8-3503-4859-0/DATE24/© 2024 EDAA

proposes a computational environment to facilitate smooth data transfer between CPUs and HDC-specific processors. Simultaneously, we aim to establish an efficient and rapid environment for executing HDC operations by designing and constructing a Hyperdimensional Processing Unit (HPU) architecture that is specifically tailored for hyperdimensional computation. This computational framework where the HPU and CPU operate in synchronized harmony is what we propose as the EcoFlex-HDP.

The main contributions of this study are summarized below:

- We designed an HPU architecture equipped with computational instructions optimized for HDC operations. An HPU is a minimal processor engineered to execute HDC computations more efficiently and quickly than a CPU.
- We constructed the EcoFlex-HDP, a platform where the CPU and HPU operate in coordinated harmony. Through EcoFlex-HDP, tasks can be executed flexibly and efficiently, either by utilizing existing software or by sharing the workload between the CPU and HPU.
- We developed a software stack that facilitates collaboration between the CPU and the HPU. This stack allows software developers to easily employ the libraries and assemblers provided to facilitate cooperative task processing between the HPU and CPU.
- We verified the capability of EcoFlex-HDP to integrate with existing software and its adaptability to new algorithms. Specifically, we confirmed that both the CPU and HPU could collaboratively execute three types of recognition tasks: image, voice, and language.
- We evaluated the performances of various EcoFlex-HDP implementations on an FPGA platform, using various metrics such as execution speed and power consumption. We discovered that EcoFlex-HDP exhibited notably improved energy–delay product (EDP) compared to commercial CPUs such as M1 Max and Cortex-A9. Specifically, the EDP of EcoFlex-HDP was approximately 13,469 and 538 times better than those of Cortex-A9 and M1 Max, respectively.

The remainder of this paper is organized as follows. Sec. II provides an overview of HDC, existing dedicated processors, and their challenges. Sec. III details the EcoFlex-HDP architecture. Sec. IV evaluates EcoFlex-HDP and commercial processors from the perspectives of execution speed, power consumption, and EDP. Finally, Sec. V concludes the paper.

II. HYPERDIMENSIONAL COMPUTING

The fundamental concept of HDC involves encoding low-dimensional input data into hyperdimensional vectors [2]. HDC defines unique operations on these hyperdimensional vectors and combines these operations to facilitate the input data learning and inference processes [14]. The elements of the hyperdimensional vectors employed in HDC can range from natural to imaginary numbers and binary values. In this study, we utilized binary hyperdimensional vectors primarily because of the ease with which they can be optimized through hardware implementation [12]. HDC is structured into two distinct phases: learning and inference, both of which enable the processing of a diverse range of recognition problems. During the learning phase, hyperdimensional vectors that capture the characteristics of each class are generated from the input data and stored in memory. In the inference phase, new test data are encoded into hyperdimensional vectors, and their similarity to the stored class data is computed. The principal advantages of the HDC are its simplicity and efficiency. Compared with conventional machine-learning algorithms, both the learning and inference processes in HDC are straightforward. This simplicity enables the rapid and efficient capture of salient features inherent to the data. Encoding operations are subject to various algorithms tailored for specific recognition tasks in both the learning and inference phases. Numerous studies have reported the efficacy and utility of these algorithms [4], [10], [15], [16]. These are designed to capture the relevant features from the data for each class. Consequently, their applicability is strongly tied to target applications such as image and speech recognition. Therefore, the algorithm must be modified appropriately for each application [2].

HDC manipulates hyperdimensional data through three fundamental operations: Bind, Permutation, and Bound, thereby capturing their interrelationships and patterns [1]. The Bind operation elucidates the correlation between hyperdimensional vectors. Utilizing bitwise exclusive OR (XOR), a new vector $\boldsymbol{H} = \boldsymbol{X} \oplus \boldsymbol{Y}$ is generated from vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$. The resulting vector $\boldsymbol{H}$ is approximately orthogonal to both $\boldsymbol{X}$ and $\boldsymbol{Y}$. The Permutation operation is a unary operation that modifies the sequence of vector elements. Typically, circular shifts are employed, and a new vector $\rho(\boldsymbol{X})$ is generated, which is also approximately orthogonal to $\boldsymbol{X}$. The Bound operation amalgamates multiple vectors to generate a new vector $\boldsymbol{Z} = [\boldsymbol{X}_1 + \boldsymbol{X}_2 + ... + \boldsymbol{X}_n]$. Using a majority function $M(\cdot)$, the operation summarizes data by selecting the most frequent value at each bit position expressed as $M(p_1, p_2, ...p_n) = \lfloor \frac{1}{2} + ((\sum_{i=1}^{n} p_i) - \frac{1}{2})/n \rfloor$. The resulting vector retains the attributes of the original vectors. This Bound operation is generally executed in the final stage of computational processing primarily because the other two operations, Bind and Permutation, demonstrate distributive properties concerning the Bound operation [9].

Although these operations contribute significantly to the efficient processing of cognitive tasks, computing them on a general-purpose CPU is more efficient, resulting in numerous proposals for designing processors specialized for HDC application [4], [10]–[13]. Existing studies have indicated that such specialized processors exhibit superior computational efficiency and lower energy consumption. They primarily optimize the manipulation of hyperdimensional vectors, thereby reducing the computational overhead incurred by the CPU. However, most existing studies focus on optimizations exclusively for specific HDC algorithms, resulting in poor adaptability to new algorithms or encoding schemes. Although some studies have proposed specialized processor designs capable of executing arbitrary algorithms [9], [17], they do not offer CPU integration, hindering the utilization of existing software resources. To address these challenges, a novel design approach that focuses

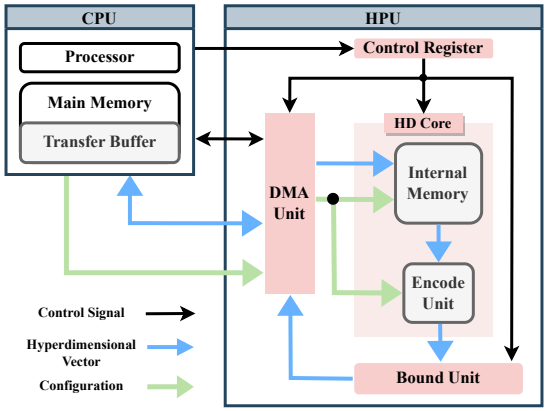

Fig. 1. EcoFlex-HDP Overview.

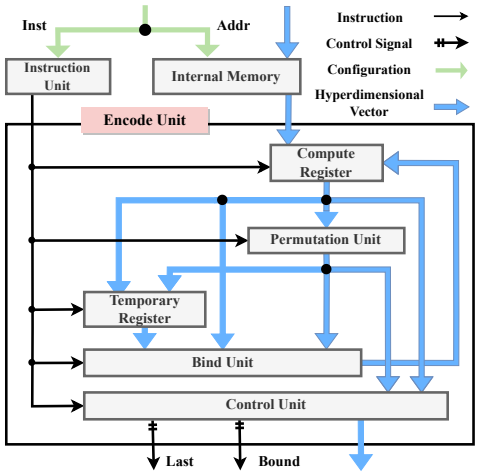

Fig. 2. HD Core Overview.

on the diversity and extensibility of algorithms and seamless hardware-software integration is required.

III. Proposed Platform

The proposed EcoFlex-HDP addresses two primary challenges. First, a low-power, high-speed HPU architecture is introduced to mitigate CPU inefficiencies in HDC operations. Second, we aim to create a computational environment that supports new algorithms, while ensuring compatibility with existing software. In this section, we discuss our design strategy, outline the four design goals, and describe the mechanism that enables cooperative operation between the HPU and CPU.

A. Design Strategies and Goals

We developed a unique design strategy for EcoFlex-HDP, where the HPU specializes in handling computationally intensive HDC operations—Bind, Permutation, and Bound—whereas the CPU is responsible for other general tasks. Based on this strategy, EcoFlex-HDP was designed to leverage the strengths of both types of processors. Specifically, the HPU efficiently handles HDC operations with low power consumption and high speed, whereas the CPU offers the flexibility to adapt to new algorithms and more complex tasks. In designing EcoFlex-HDP, we focused on the following four primary design goals (DGs):

DG1: Make it faster than CPU
DG2: Make it use less power than CPU
DG3: Make it programmable for various HDC algorithms
DG4: Enable efficient collaboration between CPU and HPU

B. EcoFlex-HDP: Hyperdimensional-Computing Platform

The EcoFlex-HDP structure shown in Fig. 1 was designed for the collaborative execution of HDC tasks between the CPU and HPU. The CPU component comprises a processor and main memory, with a portion of the main memory functioning as a transfer buffer. The HPU was composed of four units. The HD core and bound unit specialize in HDC operations, whereas the control register and DMA unit manage the communication between the CPU and the HPU.

Here, we describe the role of each unit in an HPU. The control register adjusts the HPU operating mode, toggling between preparation and execution modes. Data can be pre-generated because the hyperdimensional vectors used for HDC operations are fixed for each task. In preparation mode, the HPU pre-arranges these hyperdimensional vectors, and in execution mode, the HDC operations are carried out collectively, as instructed by the CPU. The DMA unit transfers hyperdimensional vectors and configuration signals between the CPU and the HPU. This configuration uses 16-bit data that specifies both the instructions for the encoding unit and the addresses in the internal memory. Once the DMA unit receives transfer commands from the CPU via memory-mapped I/O, it begins transferring the data sequentially allocated to the transfer buffer. Furthermore, the calculation results in the HPU were transferred and received via the DMA unit to the transfer buffer using the burst transfer function. The HD core performs HDC operations and comprises seven units, as shown in Fig. 2. The instruction unit receives commands from the CPU and instructs each unit on detailed operations. The internal memory stores hyperdimensional vector data and sends it from the address specified in the configuration signal to the encoding unit. The encoding unit performs basic Bind and Permutation operations. Computation and temporary registers hold the operands and results of the HDC operations. The control unit outputs the last signal to halt the HPU operations or activate the Bound unit and sends the computational results. The Bound unit receives the output from the HD core and the Bound signal. At the time indicated by the Bound signal, a Bound operation is performed, generating a new hyperdimensional vector. This newly generated hyper-dimensional vector can be transferred to a CPU via a DMA unit.

EcoFlex-HDP operates in two main phases to execute the HDC tasks: learning and inference. During the learning phase, the CPU prepares hyper-dimensional vector data from various sources and stores them in the transfer buffer within the main memory. The HPU then receives these data and the accompanying configuration signal, which includes operational instructions, via the DMA unit. Encoding and class data generation are performed using the HD core and Bound unit. The resulting data are then returned to the CPU. In the inference

phase, the process for encoding the test data is the same as that for the learning data. The results of the HDC operations are sent back to the CPU, where they are compared with the class data generated during the learning phase for classification.

C. Design Goal

1) DG1: There are three main technical challenges to achieving higher speeds. First, data transfer between the CPU and HPU can result in a memory bottleneck. Second, the internal read speed of HPU memory may limit its processing speed. Third, the computing speed of HDC operations has not been improved to its maximum potential.

We introduced internal memory and operating modes to address the first issue. Because hyperdimensional vectors are fixed for each HDC task, they can be pre-stored in the internal memory. This eliminates the need to wait for data from the CPU during operation. In addition, we leveraged burst transfers to eliminate random access to the main memory. We removed the cache and memory hierarchy to address the second issue. As shown in Fig. 2, the internal SRAM is placed adjacent to the arithmetic unit. This eliminates cache misses and reduces the data transfer latency. To address the third issue, we designed a minimal configuration for HPU. As shown in Fig. 2, the HDC operations were achieved using only two registers. Each operation can be executed in a single cycle. The Bind operation is based on XOR and requires no special hardware. The Permutation operation uses a small multiplexer and a sequential circuit to achieve a low-latency circular shift. The latency introduced by using a sequential circuit in the Permutation is masked by introducing multithreading. The Bound operation implements a method for sequentially adding results to a signed counter, which is also used in existing methods [9], [12].

2) DG2: There are three main technical challenges to achieving low power consumption. First, registers are required to store hyperdimensional vectors. Second, the wiring within the HD core is complex. Third, most voting operations in the Bound operation are complex, which increases power consumption.

To address the first and second challenges, we limited the number of registers to only two and fixed their roles. Specifically, the compute register is the only one that can receive data from the internal memory. It also has the exclusive privilege of transferring data to a temporary register, Permutation unit, and control unit. However, the temporary register is only used as a temporary storage location for Bind operations. By restricting its use, the amount of wiring was reduced. Additionally, to tackle the third challenge, we designed a simplified Bound unit that uses a signed counter and references only the most significant bit for the computation result, a method also used in existing approaches [9], [12]. This allowed us to reduce the circuit area required for complex majority-voting operations.

3) DG3: To achieve programmability in HPU, it is essential to freely combine the three primary HDC operations. As noted in Sec. II, the Bound operation is performed by leveraging the distributive property. We focused on this characteristic and designed a system to separate the Bound unit from the HD core, as illustrated in Fig. 1. This allows the Bound operation

TABLE I
HPU INSTRUCTION SET

Instruction	Action
Load	Load data from internal memory onto the compute register
Permutation	Permute values in the compute register
Bind	Bind compute and temporary registers
P.Bind	Bind temporary register with the Permutation result
Bound	Transfer values from the compute register to the Bound unit
P.Bound	Transfer Permutation results to the Bound unit
Move	Copy values from the compute register to temporary register
P.Move	Copy Permutation result values to the temporary register
Last	Terminate HPU operations

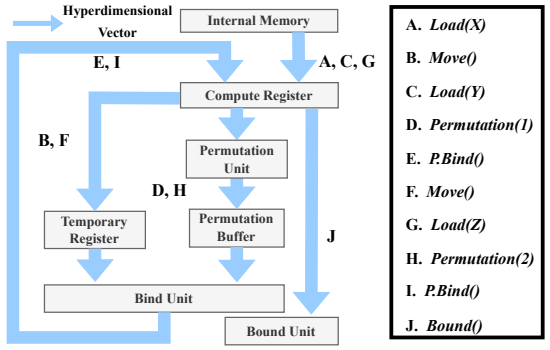

Fig. 3. Execution flow of the HDC algorithm using the dedicated library.

to be executed at the end of HDC operations. In addition, a free combination of Bind and Permutation operations is made possible by utilizing two registers and units within the HD core. As shown in Table I, an instruction set was devised to enable developers to combine these operations programmatically. Developers can freely issue instructions to the HD core and Bound unit using these ten basic instructions. The advantage of this design is its ability to accommodate new algorithms by flexibly combining the basic instructions. Consequently, the HPU can achieve high efficiency and flexibility across a broad range of HDC tasks.

4) DG4: To achieve seamless cooperation between the CPU and the HPU, it is necessary to establish the appropriate interfaces and mechanisms to enable mutual communication, which is achieved through the DMA and the memory-mapped I/O units, as shown in Fig. 1. In addition, abstraction is essential to efficiently manage HPU from CPU. Therefore, we developed a software stack that allows developers to run the HPU using languages with which they are already familiar. This software stack, provided as a dedicated library and assembler, abstracts the complexities of HPU operations. In such an environment, developers can focus on HDC tasks without worrying about memory management or resource-access violations.

Fig. 3 shows an example of a 3-gram HDC algorithm implementation, which was proposed in [10], using the EcoFlex-HDP. This algorithm encodes hyperdimensional vectors at addresses X, Y, and Z by combining the Bind and Permutation operations. The algorithm is realized by invoking functions (A)–(J), as shown in Fig. 3, from the library using the CPU. These functions correspond to the instructions listed in Table I. Memory management and directives are abstracted by the library, allowing developers to call functions and send instructions from the CPU to the HPU. Although Fig. 3 only

TABLE II
FPGA DEVICE RESOURCE UTILIZATION

	Resources used	Resources available	Utilization rate (%)
LUT	47122	53200	88.58
LUTRAM	527	17400	3.03
Flip-Flop	50707	106400	47.66
Block RAM	92	140	65.71
Global Buffer	2	32	6.25

TABLE III
EXPERIMENTAL ENVIRONMENTS

	No. of cores	Freq. (MHz)	OS	GCC (O2)
HDP	2	125	Debian11.3	10.2.1
M1 Max	10	3000	macOS	12.2.0
Cortex-A9	1	666	Debian11.3	10.2.1

shows the code used to implement library calls, this implementation can also be seamlessly integrated with the control instructions for looping and branching. Moreover, developers can assemble instructions in various ways to implement the desired algorithm.

IV. EXPERIMENTAL

A. Setup

In this study, EcoFlex-HDP was designed and evaluated using a Zynq XC7Z020 SoC, wherein both the FPGA and Cortex-A9 are integrated into a single chip, on a Xilinx PYNQ-Z1 board [18]. The HPU was implemented within the FPGA and controlled through the Cortex-A9 processor. This design allows processing 1,024 hyperdimensional vectors and storing 512 of them using Block RAM as the internal memory. We used Xilinx Vivado [19] as the design tool, and the device resource utilization is outlined in Table II. The communication between the CPU and the HPU was facilitated through 32-bit signal lines and AXI4-Stream and AXI4-Lite interfaces. Data were transferred through the Xilinx AXI DMA unit and accelerator coherency port of Zynq, which incorporate burst transfer functionalities. The number of cores and operating systems used in the evaluated platforms are summarized in Table III. The HPU allows two HD cores to operate in parallel, and the internal memory of each core stores identical vectors to ensure data integrity. Moreover, the HPU library can automatically generate configurations that are compatible with multiple HD cores, allowing developers to program without worrying about environmental differences.

This experiment evaluated the performance of EcoFlex-HDP, Cortex-A9, and M1 Max. The evaluation was divided into the microbenchmarking and application tests. In the former, three basic operations (Bind, Permutation, Bound) are executed on randomly generated test patterns, and their speeds are measured. We used the rand() function of stdlib.h, which is a standard C-library, for random number generation. We used a library developed exclusively for HDC to implement each CPU. The Permutation was evaluated as a right circular shift. Application tests were used to evaluate the performances of the processors for language, speech, and image recognition using three real-word datasets: the Wortschatz Corpora [20], ISOLET dataset [21], and MNIST dataset [22]. The learning and inference performance for each recognition task was evaluated

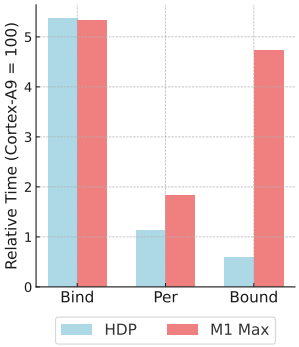

Fig. 4. Execution times for Bind, Permutation, and Bound operations with Cortex-A9 execution time = 100.

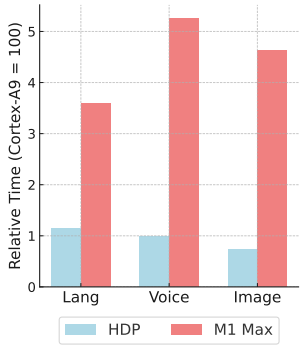

Fig. 5. Execution times for language, voice, and image recognition tasks with Cortex-A9 execution time = 100.

TABLE IV
SPEED IMPROVEMENTS OF ECOFLEX-HDP OVER M1 MAX AND CORTEX-A9

	vs. M1 Max	vs. Cortex-A9
Bind	0.99×	18.61×
Permutation	1.61×	88.80×
Bound	8.00×	169.11×

using the HDC algorithms proposed in previous studies [4], [10], [15]. The datasets were read with the fread() function in C using the Wortschatz Corpora [20], ISOLET dataset [21], and MNIST dataset [22]. In addition, the power consumption and EDP were measured in the experiments to account for energy efficiency; EDP was calculated as the square of the product of power consumption and processing time The power consumptions of the HPU and Cortex-A9 were measured using a USB power meter attached to the PYNQ-Z1 board, and the power consumption of the M1 Max was measured using the Powermetrics utility of macOS.

B. Results

1) Microbenchmarking test: In the initial experiments, we evaluated the performance of basic HDC operations (Bind, Permutation, Bound) using microbenchmarking tests. Table IV shows how many times faster EcoFlex-HDP can process tasks compared with M1 Max and Cortex-A9. In Fig. 4, we present the relative execution times for EcoFlex-HDP and M1 Max, with the execution time for Cortex-A9 set to 100 as a reference. Notably, these computational times exclude the time required for test pattern generation. Table IV and Fig. 4 demonstrate that EcoFlex-HDP performs remarkably well in basic HDC operations. Particularly in the Bound operation, EcoFlex-HDP is significantly faster than both M1 Max and Cortex-A9. The Bind and Permutation operations are also faster than Cortex-A9 and are comparable to M1 Max. Considering these results, EcoFlex-HDP exhibits excellent potential for achieving high performance in various HDC operational tasks.

2) Recognition tasks: The performance of the EcoFlex-HDP was evaluated for language, speech, and image recognition applications. First, we evaluated the inference accuracy for each recognition task. Consequently, we achieved accuracies of approximately 96% for language recognition, 92% for speech recognition, and approximately 71% for image recognition.

TABLE V
EXECUTION TIMES OF THE THREE PROCESSORS FOR EACH RECOGNITION TASK DURING THE LEARNING PHASE

	Language	Voice	Image
HDP	0.345 s	0.380 s	3.452 s
M1 Max	1.076 s	2.024 s	22.035 s
Cortex-A9	29.951 s	38.496 s	475.201 s

TABLE VI
POWER CONSUMPTION FOR EACH RECOGNITION TEST

	Language	Voice	Image
HDP	2.4 W	2.4 W	2.4 W
M1 Max	30.4 W	27.8 W	31.6 W
Cortex-A9	1.7 W	1.7 W	1.7 W

TABLE VII
EDP FOR EACH REGOCNITION TEST

	Language	Voice	Image
HDP	0.2	0.3	28.5
M1 Max	35.1	113.8	15343.1
Cortex-A9	1525.0	2519.3	383887.1

These inference accuracies were comparable to those of existing HDC algorithm-based methods, confirming the accurate implementation of the algorithm.

Next, we evaluated the execution speed. Table V represents the execution time for each application during the learning phase. Fig. 5 shows the relative execution time on the other platforms when the execution time on the Cortex-A9 is set to 100 as the reference. The "execution time" here is the processing time excluding the data loading time from storage. Fig. 5 shows that EcoFlex-HDP has significantly shorter execution times than Cortex-A9 and M1 Max. In particular, its image recognition performance is approximately 137 times faster than that of the Cortex-A9 and 6.3× faster than that of the M1 Max.

Finally, we evaluated the power consumption and EDP. Tables VI and VII present the power consumption and EDP during the execution of each application, respectively. EcoFlex-HDP consumes approximately 1.4× more power than Cortex-A9 owing to the inclusion of power consumed by the FPGA, which houses both Cortex-A9 and HPU. Nevertheless, the significant reduction in execution time allows the PYNQ Z1 board to achieve higher power efficiency, resulting in a significantly improved EDP. In particular, the EDP improved by approximately 13,469 times for image recognition. Compared to M1 Max, the power consumption was reduced by 0.07× for some tasks, and an improvement in the EDP of up to approximately 538× was observed.

V. CONCLUSION

In this study, we focused on HDC and developed the EcoFlex-HDP that allows the HPU and CPU to work together and process HDC operations quickly and efficiently. Additionally, the HPU exhibited a clear advantage over existing CPUs, particularly across three key metrics: speed, power consumption, and programmability. Moreover, EcoFlex-HDP allows integration with existing software and can support new algorithms. Furthermore, we proposed a mechanism for abstracting the HPU behavior by building a software stack; this is expected to promote the research and development of HDC and an ecosystem of software development using HDC. Evaluation of the computing speed and power consumption of EcoFlex-HDP and commercial processors showed that EcoFlex-HDP is significantly faster than Cortex-A9, with an EDP improvement of up to 13,469×. Furthermore, the speed and EDP were improved by 6.3× and 538×, respectively, compared to M1 Max. This performance shows that EcoFlex-HD can be particularly useful for edge computing. Additionally, these results demonstrate the potential of EcoFlex-HDP to efficiently perform a wide range of cognitive tasks for IoT devices and highlight its significant utility.

ACKNOWLEDGMENT

This work was partially supported by JSPS KAKENHI Grant, No. 22K11954.

REFERENCES

[1] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, pp. 139–159, 2009.
[2] L. Ge and K. K. Parhi, "Classification using hyperdimensional computing: A review," *IEEE CAS Magazine*, vol. 20, no. 2, pp. 30–47, 2021.
[3] A. Rahimi *et al.*, "High-dimensional computing as a nanoscalable paradigm," *IEEE Trans. CAS-I*, vol. 64, no. 9, pp. 2508–2521, 2017.
[4] M. Imani *et al.*, "VoiceHD: hyperdimensional computing for efficient speech recognition," *Proc. ICRC*, 2017.
[5] A. Rahimi *et al.*, "Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of exg signals," *Proc. IEEE*, vol. 107, pp. 123–143, 2019.
[6] M. Imani, S. Bosch, M. Javaheripi, B. Rouhani, X. Wu, F. Koushanfar, and T. Rosing, "SemiHD: Semi-supervised learning using hyperdimensional computing," *Proc. ICCAD*, pp. 1–8, 2019.
[7] M. Imani, Y. Kim, T. Worley, S. Gupta, and T. Rosing, "HDCluster: an accurate clustering using brain-inspired high-dimensional computing," *Proc. DATE*, pp. 1591–1594, 2019.
[8] A. Moin *et al.*, "An EMG gesture recognition system with flexible high-density sensors and brain-inspired high-dimensional classifier," *Proc. ISCAS*, pp. 1–5, 2018.
[9] S. Datta *et al.*, "A programmable hyper-dimensional processor architecture for human-centric iot," *IEEE J. Emerg. Sel.*, vol. 9, no. 3, pp. 439–452, 2019.
[10] A. Rahimi *et al.*, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," *Proc. ISLPED*, pp. 64–69, 2016.
[11] S. Salamat *et al.*, "Accelerating hyperdimensional computing on FPGAs by exploiting computational reuse," *IEEE Trans. on Computers*, vol. 69, no. 8, pp. 1159–1171, 2020.
[12] M. Imani *et al.*, "A binary learning framework for hyperdimensional computing," *Proc. DATE*, 2019.
[13] F. Montagna *et al.*, "PULP-HD: Accelerating brain-inspired high-dimensional computing on a parallel ultra-low power platform," *Proc. DAC*, pp. 1–6, 2018.
[14] M. Imani *et al.*, "Exploring hyperdimensional associative memory," *Proc. HPCA*, 2017.
[15] A. X. Manabat *et al.*, "Performance analysis of hyperdimensional computing for character recognition," *Proc. ISMAC*, 2019.
[16] M. Imani *et al.*, "BRIC: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing," *Proc. DAC*, pp. 1–6, 2019.
[17] M. Eggimann *et al.*, "A 5 μw standard cell memory-based configurable hyperdimensional computing accelerator for always-on smart sensing," *IEEE Trans. CAS-I*, pp. 1–13, 2021.
[18] "Zynq-7000 SoC data sheet: Overview," Xilinx, Inc. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ds190-Zynq-7000-Overview
[19] *Vivado Design Suite*, Xilinx, Inc., 2018, [Online] Available: https://www.xilinx.com/products/design-tools/vivado.html].
[20] U. Quasthoff *et al.*, "Corpus portal for search in monolingual corpora," *Proc. LREC*, 2006.
[21] "ISOLET," UCI machine learning repository. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/isolet
[22] "MNIST handwritten digit database," ATT Labs, 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist