Accelerating the Permute and N-Gram Operations for Hyperdimensional Learning in Embedded Systems

Pere Vergés, Igor Nunes, Mike Heddes, Tony Givargis and Alexandru Nicolau

October 28, 2023

Accelerating Permute and N-gram Operations for Hyperdimensional Learning in Embedded Systems

Removed for blind review

***Abstract*—Hyperdimensional computing (HDC) is a novel computing framework that has gained significant attention for its ability to accelerate machine learning algorithms. Its fast learning and inference capabilities make it an ideal technique for various fields, including machine learning. HDC utilizes high-dimensional holographic vectors, which are vectors with independent and identically distributed dimensions, to represent information. This unique representation allows HDC to leverage highly parallelizable arithmetic operations such as *bundling*, *binding* and *permute*. These simple and highly optimizable operations make HDC an efficient framework for classification in embedded systems. HDC has demonstrated remarkable accuracy in learning patterns from sequenced data. In this paper, we propose a method to enhance the permute operation, which is crucial for maintaining the order of symbols or measures in real-time data. Our method enhances the efficiency of HDC's permute operations by a factor of 10×. Furthermore, by applying the same idea to n-gram encoding, we achieve a speedup of 14×, resulting in up to 26.8× speedup on a real application, compared to a state-of-the-art HDC prototyping library. To achieve this improvement, we utilized SIMD operations and shifted entire SIMD data blocks rather than individual elements. As a result, we demonstrate that real-time inference can be conducted rapidly in applications that are utilized in embedded systems with constrained computational and memory resources, such as those for recognizing emotions, gestures, and language.**

***Index Terms*—Embedded Systems, Machine Learning, Hyperdimensional Computing, Vector Symbolic Architecture, High-Performance Computing, SIMD**

I. Introduction

Hyperdimensional Computing (HDC) [1], also known as Vector Symbolic Architectures (VSA) [2], is an emerging brain-inspired computational framework [3], [4] that encodes data using high-dimensional vectors caller *hypervectors*. HDC has shown significant benefits in terms of reduced memory and computational resource usage, while maintaining high accuracy in machine learning tasks [5]. As a result, HDC has emerged as a viable solution for resource-constrained environments, including embedded systems with limited processing power, memory, and energy [6]. The inherent robustness of HDC against hardware errors [7], such as bit upsets, is due to its utilization of vectors with identically and independently distributed dimensions. This property is a crucial feature for internet of things (IoT) devices, which ensures that such errors and noise only have a negligible impact on the algorithm's accuracy [8], making HDC a robust option for IoT devices that are often subject to potential hardware failures [9]–[12]. HDC's high level of parallelism enables multi-threaded embedded devices to achieve significant speedups [13], [14]. Additionally, the utilization of SIMD vector operations can significantly enhance the speed of HDC by reducing the number of load and store operations performed [15]. Furthermore, HDC has allowed power-constrained IoT devices [16] to achieve real-time classification with low latency by exploiting its ease of high-speed parallel processing. As such, HDC has shown significant potential for being a powerful technique for accelerating machine learning algorithms in embedded systems. These properties make HDC a solid option for tackling problems such as speech recognition [17], image processing [18], and sensor data analysis [19], which are typically tasks carried out by embedded devices that require a rapid response time.

Hyperdimensional learning solves real-time classification tasks by using a learning model that relies on encoding data into hypervectors, achieved through three operations: *bundle*, *bind*, and *permute*. Among these operations, *permute* is particularly important for encoding temporal data, allowing the model capture the information's sequence or ordering. The permute operation is widely utilized in numerous applications in combination with the n-gram encoding [10], [19]–[24] (see Section IV-B), which is used to group various types of data, such as letters, words, phrases, and time-series, among others.

TABLE I: Execution time (ms) of permute(4) of size 10000.

Method	Time (speed up)
Shuffle (HDCC)	80
SIMD shift (OURS)	8 (**10**×)

In order to accelearte hyperdimensional learning, particularly the permute operation, we propose leveraging the innate multithreaded parallelism of HDC, as well as its ease to use Single Instruction to Multiple Data (SIMD) operations. Our research presents a novel approach for enhancing the efficiency of the permute operation and n-gram encoding. The permute operation is typically executed through circular shifting of the hypervector, which involves a significant number of load and store operations on memory. This process can become particularly burdensome due to the high dimensionality of the hypervectors. Our proposed solution reduces the number of load and store operations on the order of the SIMD vector size by shifting the SIMD blocks that constitute the vector, rather than a single element within the high dimensional vector. This technique achieves a speedup of 10×, compared to the conventional single element shift approach. Additionally, we apply this approach to the widely used n-gram encoding by utilizing the SIMD block shift in conjunction with the addition and multiplication of SIMD blocks. This methodology not only reduces memory consumption compared to the state-of-the-art prototyping library Torchhd[1] [25], but also increases the speed of the operation by 14×, as compared to the single element shift approach. The resulting acceleration of the permute operation and n-gram encoding, coupled with parallel and SIMD operations, significantly improves response time of machine learning in embedded systems.

TABLE II: Execution time (ms) of 4-gram of size 10240000.

Method	Time (speed up)
Naive	1246.7
Shuffle (HDCC)	346.7 (3.59×)
SIMD shift (OURS)	84.8 (**14.7**×)

II. Related Work

Previous works proposing hardware accelerations for Hyperdimensional computing include in-memory platforms [13], application-specific integrated circuit (ASIC) accelerators for binarized models [26] [27] and acceleration by exploiting computational reuse [28]. These approaches have utilized specific hardware implementations and have targeted specific platforms to accelerate HDC computing. In contrast, our work aims to accelerate the permute operation for embedded systems without focusing on any specific target hardware. Our motivation is to enhance the inference and training time of HDC applications used in resource-limited systems, which utilize the permute operation or n-gram encoding to represent sequenced data. HDC is widely used to solve classification problems involving sequences or time series, where the permute operation plays a crucial role. Some important examples are introduced below.

Emotion recognition has been a popular area of research for HDC. In [18], the authors demonstrated that HDC was capable of classifying the effective response of multiple individuals from electroencephalogram (EEG) data, with performance comparable to the state-of-the-art. The encoding used included a component that encoded temporal relations, achieved through the permute operation. Similarly, in [20], HDC was applied to solve the emotion recognition problem for embedded systems, utilizing the n-gram encoding to encode temporal signals.

Gesture recognition studies in HDC have shown that traditional machine learning techniques cannot provide real-time training and model updates for real-time electromyogram (EMG) analysis, and HDC helps bridge this performance gap. In previous works [19], [21], [22], [24], the best encoding approaches for this task utilized either the permute or n-gram operation to encode the temporal position of each sample.

Language recognition is another area where HDC has shown outstanding results. In [23], the authors used a classification approach to identify different languages based on letter n-grams, achieving an accuracy of 96.7%. The same approach was also applied in [29], for Arabic languages.

DNA pattern matching is yet another area where HDC has been used. The permute operation has been utilized as part of its pattern-matching strategy [10].

Additionally, a recent study [24] has demonstrated that HDC is a promising technique for efficient biosignal processing, especially when the data is noisy and non-stationary. The study showcases how different biosignals, such as EMG, EEG, and electrocorticography (ECoG), can be encoded using HDC to solve classification problems.

In all the above-mentioned cases, the encoding approach incorporates the temporal relationships between the samples, achieved by using the permute operation to encode n-grams.

III. Hyperdimensonal Computing

Hyperdimensional computing is a framework that utilizes high-dimensional vectors as its fundamental data type. These vectors are typically represented using 10,000 dimensions and are holographic by construction. Each vector component is independently and identically distributed, ensuring that every dimension carries an equal amount of information.

These vectors are manipulated and combined using three operations: *bundling*, *binding*, and *permuting*. Bundling combines two input hypervectors to create a new one that is similar to the two inputs, while binding associates two hypervectors to create a new one that is dissimilar to the inputs. Finally, permuting implements a circular shift and is used to give a sense of order to hypervectors, being typically used for creating sequences or encoding text by generating n-grams.

In the hyperspace $H, \in \{0,1\}^D$, where $D \approx 10,000$, a similarity metric is used to extract information from hypervectors. This is commonly usde to asses if a hypervector is contained in a another one or not. The most commonly used similarity metrics in HDC are the dot similarity, cosine similarity, and hamming distance.

Hyperdimensional computing is an efficient and effective approach for performing classification tasks due to its ability to perform single-pass learning. To achieve this, the algorithm used for learning begins by selecting an encoding scheme to map input data to the high-dimensional space. During training, the algorithm applies the encoding to every sample and bundles the resulting hypervector to the model's associative memory under the appropriate class. This process is repeated for all samples. During inference, the model encodes the input sample and compares its hypervector to the associative memory using a similarity measure to retrieve the predicted class.

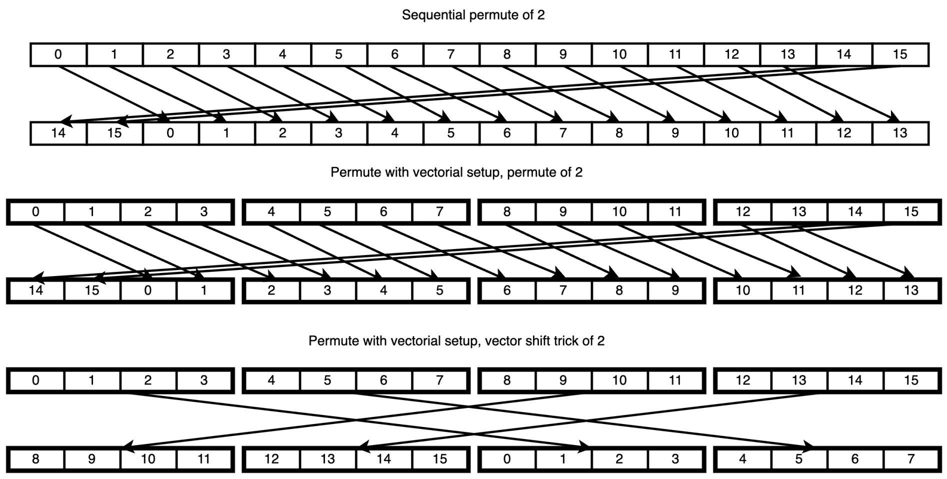
IV. Permute and N-gram acceleration

This section presents the proposed method for implementing the permute operation and n-gram encoding. To empirically demonstrate the effectiveness of our algorithm, we implemented our approach within the HDCC [30][2] compiler, which

[1]Torchhd: https://github.com/hyperdimensional-computing/torchhd

[2]HDCC: https://anonymous.4open.science/r/hdcc-5F7C/

Fig. 1: Permute representation. 1.Single element shift (Torchhd). 2.SIMD Shuffle shift (HDCC). 3 .SIMD Block shift (OURS).



produces self-contained C code that can be easily integrated into embedded systems without any compatibility issues or additional dependencies.

TABLE III: Tradeoff SIMD size and number of permutes.

SIMD size	HDCC	16	32	64	128	256
#Permutes	10240	1280	640	320	160	80
Time (ms)	0.584	0.393	0.297	0.224	0.188	0.131

A. Permutation

The primary operations in HDC are *bundle*, *binding*, and *permute*. Our method introduces a novel approach to the *permute*, which exploits SIMD operations to minimize the number of load and store operations on the order of the SIMD operation size. Typically the permute operation is implemented by performing a circular shift. However this operation is not very efficient to implement; thus, most programming languages, including C, do not provide a built-in implementation. Throughout the rest of the paper, we will assume without loss of generality that all permutations will consist of right-shifts. In the MAP (Multiply, Add, and Permute) vector symbolic architecture, permute refers to a circular shift of a hypervector, where the shift can be from one to *d-1* positions, with *d* being the hypervector's dimensionality. This operation is primarily used to establish order and represent sequences.

1) Naive permute implementation: The most naive method for implementing the permute operation (i.e., circular shift) involves reassigning every entry in the array to *n* positions to the right. This approach requires looping through every position in the array and storing the shifted value into additional data structures to prevent data loss. Figure 1 illustrates this implementation, where the first image shows the amount of data replacements required. In C, *memcpy* can also be used to copy regions of memory, but the performance achieved by assigning the elements one by one or using memcpy is virtually the same.

TABLE IV: N-gram sizes in the literature.

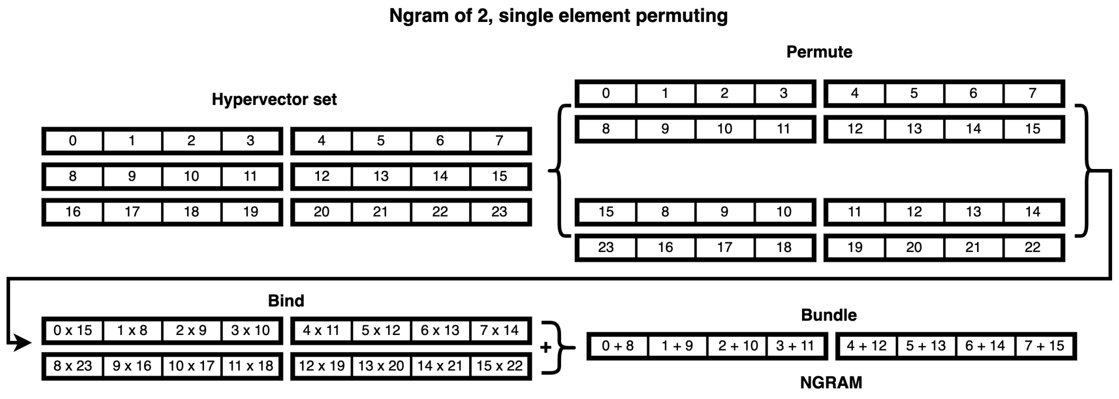
Dataset/Paper	Size (N)
EMG [24]	3
EMG Hand [22]	3
DEAP [18]	3-4
AMIGOS [20]	4
Language recognition [17]	3-5
News Classification [31]	5
Arab [29]	4-7
ERP EEG [24]	16-29

2) Shuffle permute implementation (SIMD): When performing SIMD operations in C, the use of intrinsics is necessary. C includes a built-in operation that allows shuffling an array by providing a shuffle mask, which can slightly enhance performance. However, copying between SIMD blocks of data is still required, and auxiliary data structures are necessary to store the shifted values. Figure 3 illustrates the operation of the built-in shuffle operation, while Figure 1 provides an illustration of the intuition of its use. It is worth noting that this operation is not universally implemented in all g++ or gcc compiler versions, which can result in compatibility issues depending on the compiler version and architecture.

3) Permute block operation (OURS): To enhance the efficiency of the permute operation and increase its generality while avoiding compatibility issues, we propose a more efficient approach that reduces the number of operations by leveraging the C SIMD intrinsics.

Our proposed implementation involves shifting the entire SIMD block instead of just a single element. Specifically, if we refer to the type defined in Figure 3, we would shift a

Fig. 2: Workflow of the Torchhd n-gram implementation. It is depicted how the vectors are permuted, bind and bundled to form the resulting n-gram



hyperdimensional array of f4si* in which one f4si is shifted instead of a single value. This approach significantly reduces the number of operations required, on the order of the SIMD operation size, improving its efficiency.

```
f4si __attribute__ ((vector_size (128)));
__builtin_shufflevector(v,v,5,6,7,0,1,2,3,4);
```

Fig. 3: Shuffle vector C builtin instruction

In hyperdimensional computing, permutations are used to represent ordered patterns. Typically, a permutation is achieved by shifting each element by a single position, with a permute of 1 each element in the vector is shifted one position. However, the inherent sequence information can be learned by shifting *n* elements at the same time. We leverage this property by using SIMD operations to reduce the number of load and stores required by, instead of readdressing single elements, readdressing entire SIMD blocks. Our approach is illustrated in Figure 1, where we show that instead of readdressing 16 individual elements, we only need to readdress 4 blocks.

To evaluate the effectiveness of our method, we compare the execution time of the permute operation on a hypervector of 10,000 dimensions using the HDCC implementation and our proposed method. We repeated this execution one million times and report the average time improvements achieved by our method in Table I.

B. N-gram

The n-gram encoding operation heavily relies on the permute operation. N-grams are a popular approach for encoding text and time series data. By utilizing our method, we can accelerate the n-gram operation and decrease its peak memory usage.

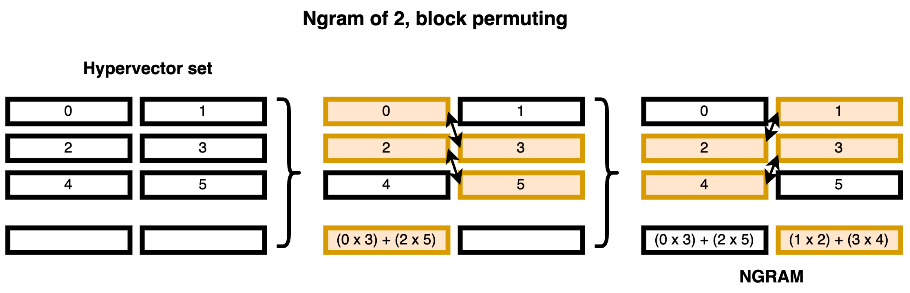
In hyperdimensional computing, the n-gram operation involves binding *n* hypervectors together. The first hypervector ($hv[0]$) is not permuted, the second hypervector ($hv[1]$) is permuted one position, and so on, up to the $(n-1)$-th hypervector being permuted by $n-2$ positions. Once each hypervector has been permuted, the resulting hypervectors are combined by bundling to form a single one. This process is repeated $m-n$ times, where m is the number of hypervectors in the hypervector set and n is the size of the n-gram. The proposed solution for the permute operation, along with the reduction in memory usage, can significantly speed up the n-gram encoding process.

Algorithm 1 SIMD Block permute n-gram encoding

```
1: procedure N-GRAM($X$, $n$)
2:     $encoding \leftarrow \{0, ..., 0\}_D$
3:     for $x$ in $X$ do
4:         $current = \{0, ..., 0\}_{Batches}$
5:         for $j$ in $x$ do
6:             for $k$ in $0, \ldots, n$ do
7:                 $block\_idx \leftarrow (j + k) \mod Batches$
8:                 $current \leftarrow current * x(j + block\_idx)$
9:             end for
10:            $encoding_j \leftarrow encoding_j + current$
11:        end for
12:    end for
13:    return $encoding$
14: end procedure
```

Figure 2 depicts the n-gram implementation approach employed in Torchhd [25], a state-of-the-art PyTorch-based library for hyperdimensional computing. In this approach, the hypervector set is permuted $n-1$ times, and the resulting permutations are then bound together to form a hypervector set that generates the final n-gram representation upon bundling. However, this approach suffers from two primary drawbacks. Firstly, it demands a considerable amount of memory, which scales with the size of the hypervector set (defined by the number of dimensions times the size of the set) multiplied by the size of the n-gram. Secondly, the approach involves shifting single elements, which incurs more operations than

Fig. 4: Workflow of our n-gram implementation, we show how the SIMD blocks are permuted and added to form the resulting n-gram.



shifting a SIMD block, resulting in higher computational costs.

Our proposed approach consists of permuting the whole SIMD block, which reduces the number of load and store operations. As shown in Listing 1, where X represents the input data as hypervectors, n is the size of the n-gram, $Batches$ is the number of SIMD blocks, and D is the dimensionality of the hyperspace, our algorithm processes the hypervector as by SIMD blocks, binding all corresponding blocks to the current one, and then bundling the result to the corresponding final n-gram hypervector. In this case, the amount of memory used is a constant n-gram hypervector with the size of the dimensions and the original hypervector set. Figure 4 illustrates its implementation.

Table II presents the time difference between using the shuffle operation and the block shifting approach for the permute operation in the n-gram encoding. We show the time taken to permute a hypervector set of 10,000 dimensions and size 1,024, with the operations repeated 1 million times.

C. Leveraging SIMD size

Using our approach has the drawback that the number of possible shifts is reduced by the size of the SIMD operation. We show in Table IV the n-gram and permute sizes used in various previous works. Most of these works employ n-grams with $n < 10$, with only one of the presented works using values up to 30. Using 10,000-dimensional vectors and 128-bit SIMD operations, the highest values for permute and n-gram size are 160, which still exceeds the maximum value used in all the examples shown in Table IV.

If an application requires a larger number for the permute and n-gram operation, one can reduce the size of the SIMD operation. Table III demonstrates how SIMD operation size affects performance. The execution time increases as the SIMD vector size decreases. However, even with smaller SIMD vector sizes, the execution time still outperforms the HDCC implementation, using single shift.

V. Experiments

This section is focused on demonstrating the speedup achieved by using our proposed SIMD permute and n-gram approach. We will compare our implementation to Torchhd [25], the current state-of-the-art hyperdimensional computing library, and HDCC implementations. We will also explore the speedups achieved by using SIMD and parallel against sequential and scalar implementations.

Initially, we will present a general comparison of the time and memory usage of the three implementations mentioned above, Torchhd, HDCC, and ours on a RaspberryPi. Subsequently, we will show an experiment that highlights the contribution of using scalar, SIMD operations, and parallel execution. Finally, we will present a study that examines how the HDCC parallelization differs from ideal parallelization.

TABLE V: Machines specifications.

Machine	RaspberryPi 4B	ThunderX
SoC	Cortex-A72 SoC	ThunderX 88XX
Memory	8GB DDR3	134 GB DDR3
Architecture	ARM v8 64bit	aarch64
Frequency	1.5 GHz	2.5GHz
CPUs	4	96
Threads per core	1	1

A. Setup

This section describes the machines and datasets used for our evaluation

1) Machines: Two machines were utilized in the experiments. We employed RaspberryPi B to showcase the performance of the three different implementations on embedded devices. It is worth noting that this RaspberryPi B model lacks support for hardware simultaneous multithreading. The second board used in the experiments was ThunderX 88XX, featuring 96 cores and was primarily used to demonstrate code parallelization. Table V presents an overview of the specifications for each board.

2) Datasets: This section provides an overview of the datasets utilized for evaluating our method. We selected four commonly used datasets in the HDC literature.

- ISOLET dataset [17] is utilized for speech recognition tasks, specifically for classifying audio recordings of the

Fig. 5: Execution comparison in a RaspberryPi of Torchhd, HDCC, and OUR implementations on all datasets using multiple dimensions. We show time(s) and Peak Memory usage in bytes

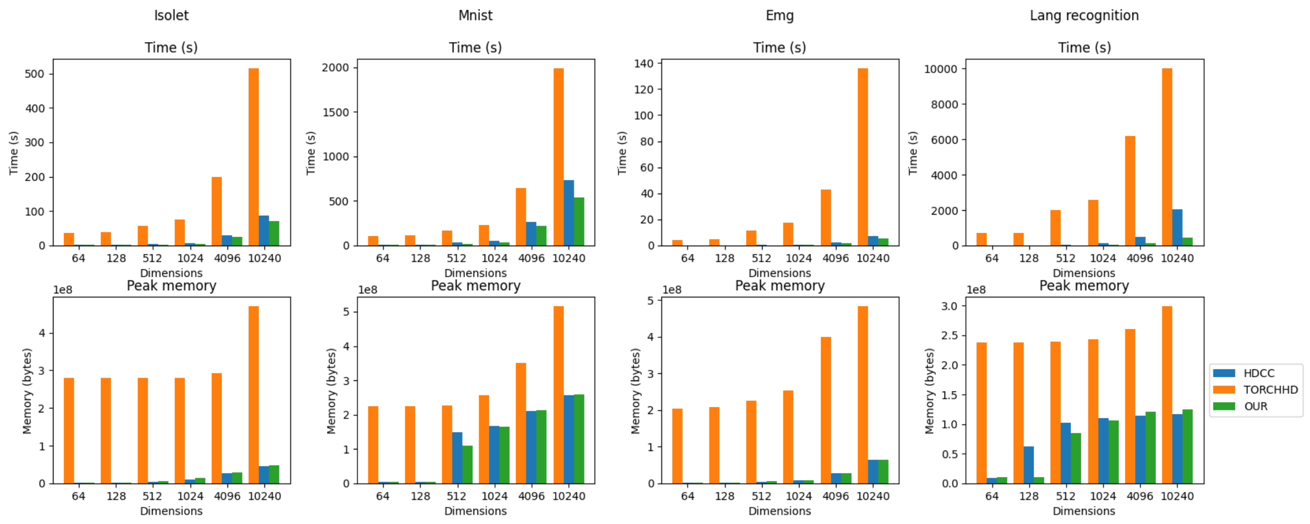


TABLE VI: Execution comparison of Torchhd, HDCC and OUR implementation in a RaspberryPi using 10000 dimensions.

Datasets	EMG	VOICEHD	LANGUAGES	MNIST
Torchhd	144.22	504.88	9869.06	1963.98
HDCC	7.43	86.10	2054.31	730.11
OUR	5.37 (**26.8**×, **1.38**×)	70.65 (**7.2**×, **1.21**×)	438.13 (**22.5**×, **4.68**×)	535.22 (**3.6**×, **1.36**×)

26 English alphabet letters. For this work, we encoded the dataset using the method proposed in VoiceHD [32], which is a hyperdimensional encoding technique for speech signals.

- EMG hand gesture recognition dataset contains recordings of the hand position of five subjects, with the goal of classifying the data into five recorded positions. The encoding used for this dataset was obtained from a previous study by Rahimi et al. in 2016, which utilized hyperdimensional computing for the classification task.
- Language recognition dataset is composed of sentences from 21 different European languages, as described in [17]. The sentences were sourced from the Wortschatz Corpora, which is a large collection of sentences in the respective languages.
- MNIST dataset [33] is a collection of images containing handwritten digits, with each image representing a number from 0 to 9. The objective of this task is to classify each image into one of the ten possible classes.

B. Experiments

This section describes the experiments carried out.

1) RaspberryPi Execution: In this experiment, we evaluated the performance of Torchhd, HDCC and our proposed implementation across all datasets using a RaspberryPi B embedded board. The results presented in Table VI and Figure 5 demonstrate that HDCC and our optimized implementation exhibit a significant speedup compared to Torchhd, while lowering the memory usage. Moreover, our optimized implementation achieves a higher speedup than HDCC across all datasets, with a maximum speedup of **4.7**× over HDCC and **22.5**× over Torchhd on the Languages dataset.

2) SIMD and parallel accelerations: This experiment aims to investigate the factors contributing the improvement of Hyperdimensional Learning performance. These factors include the parallelization of HDC [1] and the 10,000-dimensional vector operations. While we cannot achieve constant operations for vectors with 10,000 dimensions, we can leverage SIMD operations to narrow the gap. Table VII presents our findings on the contributions of parallelization and SIMD operations on Hyperdimensional Learning algorithms. The results were obtained by executing the Languages dataset on a Raspberry Pi using our optimized implementation.

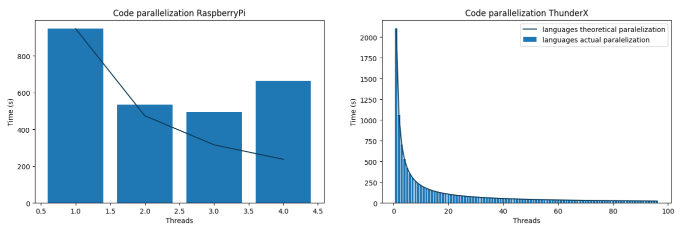
TABLE VII: Time execution comparison of Languages application using SIMD, scalar, sequential and parallel executions.

	Scalar	SIMD
Sequential	26710.4s	5770.23s (4.62×)
Parallel	8909.18s (2.99×)	425.17s (**62.8**×)

3) Parallelization study: This study aimed to evaluate the impact of parallelization on both embedded devices (RaspberryPi) and high-performance boards with large number of

CPUs (ThunderX 88XX). The results presented in Figure 6 demonstrate that on the Raspberry Pi, parallelization deviated from perfect parallelization when using more than two threads, due to oversubscription and OS management issues. However, on the ThunderX board, the parallelization times for different numbers of threads followed the theoretical parallelization, indicating that with the right implementation, almost ideal parallelization can be achieved in Hyperdimensional Learning. We tested this parallelized execution using the Languages dataset with our implementation.

Fig. 6: Parallelization evaluation of our implementation, on a RaspberryPi B and ThunderX board.



VI. Conclusion

In this study, we have shown that circular shift operations, also known as permute operations, can be accelerated using SIMD instructions by shifting entire SIMD blocks instead of single elements. Our implementation reduces the number of permute operations required and achieves speedups of 10x. Furthermore, we have implemented the n-gram operations using this approach, resulting in a 14.7× speedup. The performance of our implementation on real datasets achieves up to 22.5× speedup compared to Torchhd, the current state-of-the-art library for hyperdimensional computing, and the outperforms HDCC implementation by 4.68×. These findings highlight the potential of SIMD operations and parallelization for accelerating hyperdimensional learning algorithms, especially on high-performance boards with many CPUs. Additionally, we have demonstrated that the utilization of SIMD operations and parallelization can result in significant speedups, up to 62.8× faster than scalar and sequential code. It is evident that future work in this area could lead to even more significant improvements in the performance of hyperdimensional computing.

References

[1] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, no. 2, pp. 139–159, 2009.

[2] R. W. Gayler, "Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience," in *Joint International Conference on Cognitive Science (ICCS/ASCS)*, 2003, pp. 133–138.

[3] M. Imani, Y. Kim, S. Riazi, J. Messerly, P. Liu, F. Koushanfar, and T. Rosing, "A framework for collaborative learning in secure high-dimensional space," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, 2019, pp. 435–446.

[4] M. Imani, J. Morris, J. Messerly, H. Shu, Y. Deng, and T. Rosing, "Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[5] M. Schmuck, L. Benini, and A. Rahimi, "Hardware optimizations of dense binary hyperdimensional computing: Rematerialization of hypervectors, binarized bundling, and combinational associative memory," vol. 15, no. 4, 2019. [Online]. Available: https://doi.org/10.1145/3314326

[6] Z. Yan, S. Wang, K. Tang, and W.-F. Wong, "Efficient hyperdimensional computing," 2023. [Online]. Available: https://openreview.net/forum?id=9RQh6MOOaD

[7] S. Zhang, R. Wang, J. J. Zhang, A. Rahimi, and X. Jiao, "Assessing robustness of hyperdimensional computing against errors in associative memory : (invited paper)," in *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2021, pp. 211–217.

[8] M. Heddes, I. Nunes *et al.*, "Hyperdimensional hashing: A robust and efficient dynamic hash table," in *Design Automation Conference (DAC)*, 2022.

[9] P. Poduval, Z. Zou, H. Najafi, H. Homayoun, and M. Imani, "Stochd: Stochastic hyperdimensional system for efficient and robust learning from raw data," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 1195–1200.

[10] Y. Kim, M. Imani, N. Moshiri, and T. Rosing, "Geniehd: Efficient dna pattern matching accelerator using hyperdimensional computing," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 115–120.

[11] M. Imani, S. Pampana, S. Gupta, M. Zhou, Y. Kim, and T. Rosing, "Dual: Acceleration of clustering algorithms using digital-based processing in-memory," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 356–371.

[12] A. Hernández-Cano, N. Matsumoto, E. Ping, and M. Imani, "Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 56–61.

[13] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey, "Exploring hyperdimensional associative memory," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 445–456.

[14] P. Poduval, Z. Zou, X. Yin, E. Sadredini, and M. Imani, "Cognitive correlative encoding for genome sequence matching in hyperdimensional system," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 781–786.

[15] G. Mitra, B. Johnston, A. P. Rendell, E. McCreath, and J. Zhou, "Use of simd vector operations to accelerate application code performance on low-powered arm and intel platforms," in *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, 2013, pp. 1107–1116.

[16] Z. Zou, Y. Kim *et al.*, "Scalable edge-based hyperdimensional learning system with brain-like neural adaptation," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2021, pp. 1–15.

[17] R. Cole, Y. Muthusamy, and M. Fanty, *The ISOLET spoken letter database*. Oregon Graduate Institute of Science and Technology, 1990.

[18] A. Menon, A. Natarajan, R. Agashe, D. Sun, M. Aristio, H. Liew, Y. S. Shao, and J. M. Rabaey, "Efficient emotion recognition using hyperdimensional computing with combinatorial channel encoding and cellular automata," *Brain Informatics*, vol. 9, 2021.

[19] A. Moin, A. Zhou, A. Rahimi, A. Menon, S. Benatti, G. Alexandrov, S. Tamakloe, J. Ting, N. Yamamoto, Y. Khan, F. L. Burghardt, L. Benini, A. C. Arias, and J. M. Rabaey, "A wearable biosensing system with in-

sensor adaptive machine learning for hand gesture recognition," *Nature Electronics*, vol. 4, pp. 54–63, 2020.

[20] E.-J. Chang, A. Rahimi, L. Benini, and A.-Y. A. Wu, "Hyperdimensional computing-based multimodality emotion recognition with physiological signals," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2019, pp. 137–141.

[21] A. Moin, A. Zhou, S. Benatti, A. Rahimi, L. Benini, and J. M. Rabaey, "Analysis of contraction effort level in emg-based gesture recognition using hyperdimensional computing," in *2019 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2019, pp. 1–4.

[22] A. Rahimi, S. Benatti, P. Kanerva, L. Benini, and J. M. Rabaey, "Hyperdimensional biosignal processing: A case study for emg-based hand gesture recognition," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*, 2016, pp. 1–8.

[23] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ser. ISLPED '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 64–69. [Online]. Available: https://doi.org/10.1145/2934583.2934624

[24] A. Rahimi, P. Kanerva, L. Benini, and J. M. Rabaey, "Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of exg signals," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 123–143, 2019.

[25] M. Heddes, I. Nunes, P. Vergés, D. Desai, T. Givargis, and A. Nicolau, "Torchhd: An open-source python library to support hyperdimensional computing research," 2022.

[26] M. Imani, J. Messerly, F. Wu, W. Pi, and T. Rosing, "A binary learning framework for hyperdimensional computing," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 126–131.

[27] S. Gupta, M. Imani, and T. Rosing, "Felix: Fast and energy-efficient logic in memory," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–7.

[28] S. Salamat, M. Imani, and T. Rosing, "Accelerating hyperdimensional computing on fpgas by exploiting computational reuse," *IEEE Transactions on Computers*, vol. 69, no. 8, pp. 1159–1171, 2020.

[29] G. S. Rady, S. S. Mohamed, M. F. Mohamed, and K. F. Hussain, "High dimensional autonomous computing on arabic language classification," in *Computers and Electrical Engineering*, 2022, p. 108020.

[30] P. Vergés, M. Heddes, I. Nunes, T. Givargis, and A. Nicolau, "Hdcc: A hyperdimensional computing compiler for classification on embedded systems and high-performance computing," 2023.

[31] G. Karunaratne, M. L. Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, "In-memory hyperdimensional computing," *Nature Electronics*, vol. 3, pp. 327–337, 2019.

[32] M. Imani, D. Kong *et al.*, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *International Conference on Rebooting Computing (ICRC)*. IEEE, 2017, pp. 1–8.

[33] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.