A Binary, 2048-dim. Generic Hyper-Dimensional Processor

Sohum Datta

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2019-19
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-19.html

May 1, 2019

A Binary, 2048-dim. Generic Hyper-Dimensional Processor

by

Sohum Datta

Submitted to the Department of Electrical Engineering and Computer Science,
University of California Berkeley
in partial fulfillment of requirements for the degree of
Master of Science in Computer Science

Abstract

The last decade witnessed a simultaneous withdrawal of Moore's scaling and emergence of learning-based workloads. While these trends have seriously challenged traditional paradigms, novel computing methods based on randomness can be leveraged for continued performance. Hyper-dimensional (HD) computing, a **bio-inspired** paradigm defined on random high-dimensional vectors, shows promise.

This work describes a **2048-dimensional HD processor**. Its simple architecture follows naturally from basic HD operations; and its massively parallel, shallow datapath resembles in-memory computing. The architecture also **supports scalability**: multiple such processors can be connected in parallel to increase effective HD dimension. The design was synthesized in 28nm HKMG process and benchmarked on 9 supervised classification tasks with varying complexity (such as language recognition and human face detection). The simulated chip exhibits accuracy close to conventional machine learning algorithms for most simple tasks with **energy efficiency better than 2 uJ/prediction** at less than **2.5 ns. cycle time**.

As a **first complete design** working with high dimensional stochastic signals, the main architectural decisions for similar systems is established. An improved system harnessing variability in emerging devices (eg. CNFET and RRAM) could be deployed for ultra low-power ubiquitous computing in future.

Research Adviser: Jan M. Rabaey
Title: Donald O. Pederson Distinguished Professor of EECS

Second Reader: Bruno Olshausen
Title: Professor at Helen Wills Neuroscience Institute and School of Optometry

Acknowledgements

This project would not be possible without the continuous support and encouragement from my adviser, Prof. Jan Rabaey. While working with him, I have learned the value of principled thinking, originality and perseverance in doing research. Prof. Bruno Olshausen gave great suggestions and feedback to help me learn the subject. With my best efforts, I hope I can continue to follow their inspiration and do good work.

I am very grateful for the love and support from all my friends and family during the last two years. What I am today is because of all the wonderful things that they have taught and shared throughout my life.

I am also thankful to E2CDA-NRI for sponsoring this project. E2CDA-NRI is a funded center of NRI, a Semiconductor Research Corporation (SRC) program sponsored by NERC and NIST. I would like to thank Pentti Kanerva (UC Berkeley) for his suggestions, comments and support, and especially Mohsen Imani (UC San Diego) for readily providing data-files used for training and classification of a few benchmarks. Last but not the least, I am grateful to TSMC for providing the technology libraries used for this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The impact of computers on our civilization has been historically unprecedented. It has affected all aspects of our society: from health-care to marriage patterns. The main driver behind this digital revolution is our ability to scale transistor dimensions relentlessly, while maintaining power consumption per unit area to a constant.

Known informally as **Moore's law** [1], it had a tremendous effect on the post-war world economy. An estimated $3 trillion has been added to the global Gross Domestic Product (GDP), with an additional $9 trillion of indirect value in the last 20 years, all due to the pace of innovation facilitated by it [2]. The total value added is about 15% of the 2018 world GDP; more than that of the four largest EU economies: Germany, United Kingdom, France and Italy, combined!

The *semiconductor revolution* also assisted advances in scientific and computational theory. The Church-Turing thesis, positing that all mechanisms underlying human creativity can be manifested mechanically or electrically [3], provided the theoretical foundation for rapid increase in complexity of machines made feasible by Large Scale Integration (LSI) in 1970s. This encouraged further efforts to conceive and build the *"thinking machine"*, and over the next three decades led to the golden age of Machine Learning (ML). New paradigms of computing, including the now-popular multi-layered neural networks, were discovered and perfected in waves. These advances coupled with the sheer speed of modern computers led to breakthroughs in automating tasks requiring considerable human intelligence and careful cultivation

of skill. Indeed, the computer beat the world chess champion in 1997 [4], drove 131 miles through a desert in 2005 [5], defeated the champions of quiz show *Jeopardy!* in 2011 [6], and won against the best *Go* and *Dota 2* players in 2016 [7] and 2017 [8].

Such spectacular progress has been achieved largely because of increased awareness of the applicability of classical ML algorithms to a wide variety of fields [9], abundance of data [10] and cheap computational power. The fundamental challenges to ML theory, mainly from statistics and optimization, remain unresolved [11]. Moreover, the current workhorse optimization technique, *back-propagation*, is unlikely to continue scaling as neural network size increases [12]. Today's increasingly complex problems based on high-dimensional data pose new challenges to conventional statistical estimation and inference [13].

At the same time, non-ideal behavior in transistors of length close to that of silicon lattice foretells the end of Moore's law. As transistor dimensions shrank to the nanometer regime, variability and reliability effects began to dominate its deterministic behavior [14]. To bridge the gap, architects switched to multi-core designs even at the cost of higher power consumption [15]. Lower layers of the computing stack were affected as well: new avenues of research into materials, semiconductor physics and organic chemistry have emerged [16].

Nevertheless, some fundamental impediments remain. Adapting emerging devices to the exact-computing paradigm is increasingly difficult [17]. As energy efficiency no longer scales with integration capacity, voltage reduction and near-threshold operation reduces power consumption at the expense of favourable signal-to-noise ratio (SNR) [18]. Finally, fewer applications today (including emerging domains such as pattern classification and data mining) have enough parallelism to completely utilize available hardware [19].

Just as the past half-century saw a fortuitous coincidence of growth in semiconductor technology and machine learning, the coming end of conventional progress in both these spheres pose a serious challenge. The entire computing stack must adapt:

hardware engineers must diversify the device inventory to find a *viable* replacement for planar transistors, and software developers must adapt to new programming models. Hence, a *concerted* effort harnessing the best algorithms that are practical on emerging devices is required.

While challenges of using unreliable components has long been known [20], biology offers the most concrete inspiration. For example, our brain processes massive data (3.6×10^{15} synaptic ops./s) with very slow and *diverse* neurons (typical firing rates are 10 - 100 Hz) while exhibiting tremendous energy-efficiency (total power about 12 W) [21]! Consequently, novel brain-inspired computing approaches could provide the required robustness and scalability for continued performance.

Hyper-Dimensional Computing (HDC) is one such nano-scalable paradigm [22]. It is a theoretical model of cognitive reasoning [23, 24], motivated by the fact that brains compute by transforming activation patterns of a *large population* of neurons. Hence, tolerance to variability is inherent: changes in activation of a few neurons does not affect the overall functioning.

Robustness and energy efficiency of HD computing has been demonstrated for language recognition [25, 26] and tested on fabricated systems based on emerging devices: a hybrid of carbon nanotube field-effect transistors (CNFETs) and resistive RAM (RRAM) memory in [27], and CMOS/vertically-integrated RRAM (VRRAM) implementation in [28]. While data-paths specific to an application have been proposed [29, 30, 31, 32, 22], a general HD system is yet to be developed ([22] presents only a brief outline).

A comprehensive architectural exploration of such a general HD processor is the main aim of this work.

Chapter 2

Hyper-Dimensional Computing

Hyper-Dimensional Computing (HDC) emerges from a theoretical model of memory and cognition developed by Pentti Kanerva [24]. It is based on the fact that human brains compute by transforming activation patterns of a large mass of neurons. The set of activations are modeled as points in very high dimensional spaces ($D > 1000$), and neural processing as transformations in this space.

It turns out that non-intuitive mathematical properties of High-Dimensional (HD) spaces, which pose a challenge [33] to common ML routines such as nearest-neighbor search [34], clustering [35] and regression [36], can be used to explain human cognitive functions like association of concepts, learning and recalling by analogy. Simple operations such as superposition, binding, permutation and their inverses form an algebraic field, giving (in principle) the same universality as algebra with numbers [37].

The HDC formalism is really a mathematical abstraction of memory functions exhibited by the human brain, similar to McCullouch and Pitts' *artificial neurons* formulated in 1943 [38]. Examples of directly related paradigms include Holographic Reduced Representation [37], Binary Spatter Code [39], Random Indexing [40], and Semantic Pointer Architecture (SPAUN) [41]; collectively referred to as Vector Symbolic Architectures [42].

2.1 Orthogonality in High Dimensions

HD computing defines random high-dimensional vectors ($D > 1000$) as its fundamental datatype [23, 22]. It is a **holographic** computing framework: unlike arithmetic over numbers, no vector component contains more information than any other. Though vectors with elements from any algebraic field can be used (including from the real line or the complex plane [22]), we will focus on binary vectors.

To compare vectors, a distance metric must be defined. *Hamming distance* (denoted by $d_H(a, b)$) is the number of dissimilar elements between vectors a and b. Two binary vectors x and y of dimension D are said to be **orthogonal** if $d_H(x, y) = D/2$. This definition is more familiar in bipolar code, with 0-valued elements replaced with integer -1: orthogonal vectors x and y have zero inner-product, $\langle x, y\rangle = 0$.

The underlying principle of HDC is **almost certain orthogonality** in high-dimensional spaces. For a rigorous demonstration, begin by considering that if x and y are chosen independently and uniformly from $\{0, 1\}^D$ (i.e. probability of any bit being 1 is $p = 1/2$), then $d_H(x, y)$ is binomially distributed ($x, y \sim Bin(D, p = 1/2)$). Fig. 2-1 plots a histogram of $d_H(x, y)$ normalized by dimension D for $10,000$ randomly-generated pairs (x, y). It also plots the Normal Approximation $N(Dp, Dp(1 - p))$ of the binomial distribution $Bin(D, p)$ scaled to have an area equal to histogram sample size of $10,000$. The Normal Approximation helps in plotting and is very accurate: using Berry-Essen bound (Theorem 10.4 in [43]) for $X \sim Bin(D, p), Y \sim N(Dp, Dp(1 - p)), p = 1/2$ and dimension $D \geq 2500$, the maximum error in cumulative distribution ($max_{0<t<1}|\Pr(X \leq t) - \Pr(Y \leq t)|$) is 0.016.

Then, it can be shown that (Theorem 1 of [44]):

$$\Pr\left[\left|\frac{d_H(x, y)}{D} - \frac{1}{2}\right| \geq \epsilon\right] < 2e^{-2D\epsilon^2} \tag{2.1}$$

Since $\epsilon \leq 1/2$ and $2e^{-1/2} > 1$, only high dimensions ($D > 1000$) result in a meaningful right-hand side in Eq. 2.1 [45]. Then random vectors x and y have normalized distance very close to 0.5. The *exponential* drop in probability beyond ϵ-deviation from the

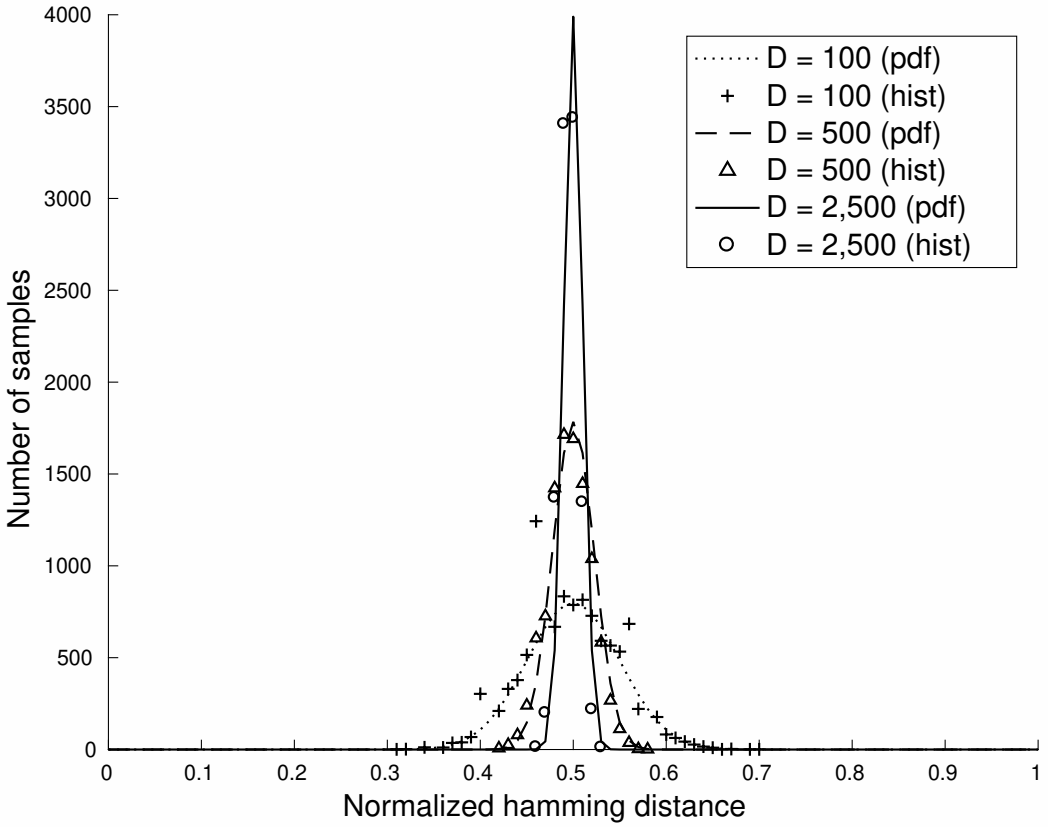


Figure 2-1: **Orthogonality in High Dimensions**: shown histogram (hist) and scaled probability density function (pdf) of hamming distances for $10,000$ pairs of random binary vectors with varying dimension D. For ease of plotting, the normal approximation of the binomial probability distribution is used. All vectors are generated uniformly and distance is normalized by D. Note the sharper concentration around 0.5 as D increases.

mean is the crucial property exploited here.

This surprising phenomenon is a consequence of concentration of functions in high-dimensional geometry. For the interested reader, [46] provides an excellent treatment from a non-asymptotic viewpoint.

2.2 The Binary HDC Subset

All high-dimensional binary vectors used in a given computation will be called **hyper-vectors**. When the context is obvious, hyper-vectors and vectors will be used interchangeably.

Though a large variety of HDC models exist (see [22]), the simple **Multiply-Add-Permute (MAP)** framework is most suited for a first general processor. These **encoding operations** allow representation of complex structures such as sequences, lists and trees [37, 24], and are fundamental to the algorithm. The notation used through the rest of this document is also introduced below.

- **Multiplication or Binding** is useful for forming *associations* among related vectors. X and Y are *bound* together to form $C = X \oplus Y$ orthogonal to both its constituents and is implemented by element-wise XOR.

- **Addition or Superposition** is the primary *conjunctive* operation. Based on Hebbian learning [37], the goal is to find a hyper-vector z representing the set of operand hyper-vectors $\{x_1, x_2, ...x_n\}$ such that $x_{1 \leq i \leq n}$ are much closer to z than a random hyper-vector ([47] provides a rigorous account). It is denoted by $z = [x_1 + x_2 + ... + x_n]$ and implemented as component-wise majority of operands.

- **Permutation** is a *unary operation* such that a permuted hyper-vector (denoted by $\rho(x)$) is orthogonal to initial hyper-vector x. A *derangement* is a permutation where no element ends up in its starting position. For orthogonality, the permutation operation must be a derangement in components of x, such as a circular shift. x permuted n times is denoted as $\rho^n(x)$.

As Section 2.1 shows, it is very rare for random hyper-vectors to deviate much from orthogonality. The MAP operations above (esp. Addition) can generate non-orthogonal vectors from operand vectors. This allows us to encode meaning, as significant deviation from orthogonality implies common membership or dependency. Therefore, an **Associative Search** to find the closest match of the MAP-encoded hyper-vector to stored *class hyper-vectors* is a crucial operation.

2.3 Example: Language Recognition

Language recognition from text is an ideal example for illustrating HDC in supervised classification, as the state-of-the-art algorithm can be directly mapped to this framework [26]. A corpus of 21 Indo-European languages transliterated to English forms the training set, and new sentences are queried as tests.

Baseline (n-gram character model): A language is modeled as a *probability distribution* on character sequences of length n (also called n-gram). More sophisticated models such as dictionary of words, phrases, etc., increase complexity with negligible gains [48, 49].

While training a language, raw n-gram frequency counts are generated from a large corpus of text and iteratively smoothened [50] to remove outlier artifacts. The resulting n-gram distribution is the trained language model. The steps are repeated for a test query, and the trained model with the *closest distribution* is the language prediction.

HDC Setup and Algorithm: To make use of HD computing, the first step is to map random hyper-vectors to meaningful entities. In this case, characters from Latin alphabet are assigned to uniformly generated hyper-vectors for dimension $D = 10,000$. The HD algorithm uses them to encode the training data and generate a **single hyper-vector** for each language.

A direct equivalent of frequency counting is the *superposition* of hyper-vectors

representing each occurring n-gram in the text. *Permutation* and *binding* is used to generate n-gram vector from constituent letter hyper-vectors. For example, the text

hello world!

could be modeled as the set of all occurring 3-grams (here white space is denoted by "_")

{hel, ell, llo, lo_, o_w, _wo, wor, orl , rld, ld!}

Once the characters are assigned to random hyper-vectors, let V_z denote hyper-vector representing character z. Then, the 3-gram "abc" is encoded as $V_{abc} \triangleq V_a \oplus \rho(V_b) \oplus \rho^2(V_c)$, and similarly for all other 3-grams. Due to properties of HD operations (Section 2.2), *all* n-gram and character hyper-vectors are orthogonal to each other. Hence, all possible 3-grams gets mapped to a set of approximately mutually-orthogonal hyper-vectors, similar to a basis set in vector spaces. Finally, the text is encoded as super-position of all n-gram hyper-vectors occurring in it. Hence, $V_{hello} \triangleq [V_{hel} + V_{ell} + V_{llo}]$. For each language in the dataset, a large corpus of text is converted similarly to a single hyper-vector that represents that language. Therefore, at the end of training, one is left with the random hyper-vectors mapped to each character and a set of 21 hyper-vectors, one for each language.

The test hyper-vector is computed similarly from the query text transliterated to Latin alphabet, and the language with *closest class hyper-vector* is returned as the prediction. Since the superimposed language hyper-vector is in the linear space spanned by basis n-gram vectors, the class with the closest n-gram distribution from *baseline* equivalently has the *smallest distance* in HDC.

For $D = 10,000$, HDC has an accuracy of *96.7 %* against a baseline of *97.1 %* [26]. However, it is an online algorithm **requiring only one pass**. The deviations from orthogonality (Fig. 2-1) during encoding operations automatically smoothen the superimposed multi-set of raw n-grams. Finally, HDC model size (one hyper-vector per language) is fixed with n-gram size, but grows *exponentially* in the baseline. For $n = 4$, the HDC model is **20X smaller** than baseline!

Application	Encoding	HDC	Known State-of-the-Art Algorithm
LANG	4-gram	90.6 %	97.1 %, n-gram-based Nearest Neighbors [26]
FlexEMG	2-stage	**95.8** %	89.7 %, Support Vector Machine [51]
DNA	60-feature	**96.2** %	93.7 %, knowledge-based Neural Networks [52]
CARDIO	21-feature	**90.6** %	90.6 %, Support Vector Machine [53]
PAGE	10-feature	**91.6** %	85.9 %, Hyperplane Separation [54]
UCIHAR	561-feature	76.7 %	89.3 %, Support Vector Machines [55]
ISOLET	617-feature	75.9 %	97.1 %, boosted k-Nearest neighbors [56]
FACE	608-feature	66.0 %	96.1 %, HOG-based boosted Decision Trees [57]
MNIST	784-feature	75.4 %	99.7 %, Deep Convolution Neural Network [58]

Table 2.1: The HDC Benchmark Suite

9 supervised classification tasks with varying complexity were chosen to evaluate the generic HD processor. See Section 2.4 for descriptions. HDC Accuracy for a *single* processor (D = 2048) is reported. *Multi-processor* configuration increases effective dimension and improves accuracy significantly (see Fig. 5-4(a) and Section 4.4).

2.4 Benchmark Applications

HD computing has been applied to a large variety of problems [22, 37]. The following 9 applications were chosen to evaluate the processor designed in this work.

Language Recognition (LANG) is described in Section 2.3 [26]. *EMG Hand-Gesture Recognition (FlexEMG)* classifies 64-channel electromyography signals recorded from a subject's hand into a set of hand-gestures [51]. *DNA Sequencing (DNA)* predicts the presence of Exon/Intron or Intron/Exon boundaries in a 60-character strand of DNA [52]. *Fetal State classification (CARDIO)* uses measurements of heart-rate and uterine pressure during pregnancy to classify fetal condition before delivery [53]. *Page-block classification (PAGE)* finds all blocks of the page layout in a document that has been detected by a segmentation process [54]. *UCI Human-activity Recognition (UCIHAR)* classifies recordings of 30 subjects performing activities of daily living while carrying a waist-mounted smartphone with embedded inertial sensors [55]. *Spoken Letter Classification (ISOLET)* predicts the English letter spoken from voice recordings of subjects. *Face Detection (FACE)* determines whether a human face is present within a given picture frame [57]. *MNIST Digit Recognition (MNIST)* predicts the digit from images of drawn digits [58].

Table 2.1 compares the accuracy of HDC with state-of-the-art machine learning models for each of these applications. **Boldface** numbers indicate equal or better performance of HDC over the best known ML algorithm for the dataset in the literature. These applications were chosen to represent the current state of the art of HDC as a paradigm. A special effort was made to have the set of ML models compared against as diverse as possible. Although it is not exhaustive, Table 2.1 attempts to give a balanced overview.

HDC is better or equal to the baseline for 4 applications in the benchmark (Table 2.1). *Language Recognition* can be further improved to the baseline accuracy by using a *multi-processor* system described in Section 4.4).

Chapter 3

Toward a General-Purpose Architecture

A versatile HDC machine must be able to handle a variety of input data-types and scale with an increasing number of channels without deteriorating fidelity. It must be able to map application-specific data, after suitable pre-processing, to a *general* architecture and perform required classification. The first step towards a general data-path is to *abstract essential elements* of HD algorithms. For supervised classification, a clear structure emerges.

3.1 Properties of HD Programs

3.1.1 Value Representation

To allow consumption by a discrete-time (clocked) digital system, the input data must be quantized into discrete states and sampled with a finite frequency. The choice of quantization scheme and sampling rates are important [59, 60] and is assumed to be pre-determined by a domain expert. Therefore, a **common symbol set** representing values in the feature space is created.

A multi-channel input stream can be serialized with a suitable policy. For example, values $x_1, x_2, x_3, ...$ from Channel 1 could be merged with $y_1, y_2, y_3, ...$ from Channel

2 to form $I_{serial} = \{x_1, y_1, x_2, y_2, x_3, y_3, ...\}$. Serialization order usually depends on data acquisition order and buffering capacity; for streaming applications only minor shuffling on raw data are feasible. Therefore, the input may be modeled as a **single time-series** without losing generality.

Once the set of classes, representation space, sampling rate, quantization and channel ordering are established, a supervised classification task is ready to be processed in HD.

To begin processing, symbols from the common symbol set are assigned to *randomly-generated* hyper-vectors called **items**. During training, the input data from each class is represented by the assigned items and transformed using HD *multiply, permute* and *superposition* to generate a **class hyper-vector**. During testing, a **test hyper-vector** is computed similarly from test data, and a **nearest-neighbor** search through all class hyper-vectors is performed. As shown previously, only the correct class hyper-vector will be close to test hyper-vector with others being nearly orthogonal.

3.1.2 Stages in HD Algorithms

The input stream is denoted by a *finite* sequence of hyper-vectors $\mathbb{I} \triangleq \{X_t | t = 1, 2...., T\}$. Let $[n] \triangleq \{1, 2, ...n\}$ for any number n. Any collection of input values can be specified by a set of their positions in $\mathbb{I}$. Since only supervised classification is considered, a given $\mathbb{I}$ belongs to a single class to be trained or tested. For both cases, the exact same algorithm is applied for processing.

To discern the basic properties of HD algorithms, it is important to consider its inherently **symbolic** nature. In fact, its remarkable power for analogical and hierarchical reasoning was among the first to be discovered [61, 37].

Testing set membership is a fundamental operation of any symbolic inference system, and HD can perform this very robustly using superposition. For the set of hyper-vectors $\mathbb{S} \triangleq \{X, Y, Z\}$, the hyper-vector $S \triangleq [X + Y + Z]$ is similar to each

X, Y, Z and nearly orthogonal to all non-members of $\mathbb{S}$. Along with *permutation* and *multiply*, this principle can be used to build complex data structures such as sequences, lists, and trees (see [37] for a detailed account). Since a complex structure (eg. a class) is *usually a set* of multiple objects with some common properties, the final step in HD processing is a superposition of their representations. Therefore, processing of hyper-vectors in HD classification algorithms **end with a superposition**. The final vector class hyper-vector I encoding the input sequence $\mathbb{I}$ can be written as superposition of K intermediate *term vectors*.

A **single-stage algorithm** is defined as any HD algorithm where superposition is used only once. All term vectors are then products of inputs and their permutations only. In other words, $I = [\sum_{i=1}^{K} f_i(\mathbb{I})]$, where i^{th} term is

$$\begin{aligned} f_i(\mathbb{I}) = (X_{p_1} \oplus X_{p_2} ... \oplus X_{p_m}) \\ \oplus (\rho^{u_1}(X_{q_1}) \oplus \rho^{u_2}(X_{q_2}) ... \oplus \rho^{u_n}(X_{q_n})) \end{aligned} \tag{3.1}$$

Each term $f_i(\mathbb{I})$ depends on specific input values: some occurring as is (denoted by the set of positions $P_i \triangleq \{p_1, p_2, ...p_m\} \subseteq [T]$ in $\mathbb{I}$), and some *with permutations* (denoted by $Q_i \triangleq \{q_1, q_2, ...q_n\} \subseteq [T]$) where permutation powers $(u_1, u_2, ..u_n)$ are positive integers. Note that few inputs may occur both with and without permutation in the term (i.e. $P_i \cap Q_i \neq \phi$).

A **dual-stage algorithm** has terms composed of products of inputs, outputs of a single-stage algorithm and their permutations. Similarly, one can build *any multi-stage* HD program by hierarchically combining outputs of smaller-stage algorithms. For the benchmark (see Table 2.1), most algorithms are single-stage except *FlexEMG* which is dual-stage.

3.1.3 The "*Generic*" Model for single-Stage HD Algorithms

Clearly, the main complexity is generation of K term vectors $f_i(\mathbb{I})$. Each term requires specific inputs $A_i = P_i \cup Q_i$, usually a small part of of the entire stream $\mathbb{I}$. In the most general case, expressions for distinct term may have very different inputs and result expressions, and separate hardware would be dedicated to each of them. However, all known HD algorithms (including those in Table 2.1) have a much simpler form. More precisely, following conditions are satisfied:

1. The number of dependent inputs $|A_i|$ is constant for all terms $i \in [K]$. Let this be L.

2. All K terms have the **same HD expression**. If $f_i(x_1, x_2, ..., x_L)$ denotes the i^{th} term in L input variables $x = (x_1, x_2, ...x_L)$, then $f_i(x) = f_j(x)\ \forall i, j \in [K]$. This common expression will be called $f(x)$.

3. The set of dependent inputs for i^{th} term are **translations of a fixed subsequence** of input stream. That is, for some increasing sequence $t_i \in [T]$ with $t_1 = 0$; the i^{th} input dependency set is $A_i = A_1 + t_i$. Here, $A_1 + t$ denotes the set obtained by adding t to each element of A_1. Note that the first term inputs A_1 captures the essential pattern of input dependencies for *all* terms.

These conditions limit the possibilities of single-stage HD expressions. An architecture designed to handle all such expressions shall be called **generic** as opposed to "*general-purpose*" or "*general*". Since input stream is a time-series, the entire programming complexity of such a machine is only for computing $f(x)$. Property 3 above ensures a *sequential* generation of all K term vectors: assuming a pipelined hardware for $f(x)$ with fixed latency and throughput equal to input rate, the i^{th} term is produced t_i steps after the first term. The *final superposition* $I = [\sum_{i=1}^{K} f_i(\mathbb{I})]$ can be computed by *accumulating* these terms $f_i(\mathbb{I})$ at the required time-steps $(t_1, t_2, ...)$. A T-bit register could mark these time-steps when the accumulator is to be enabled.

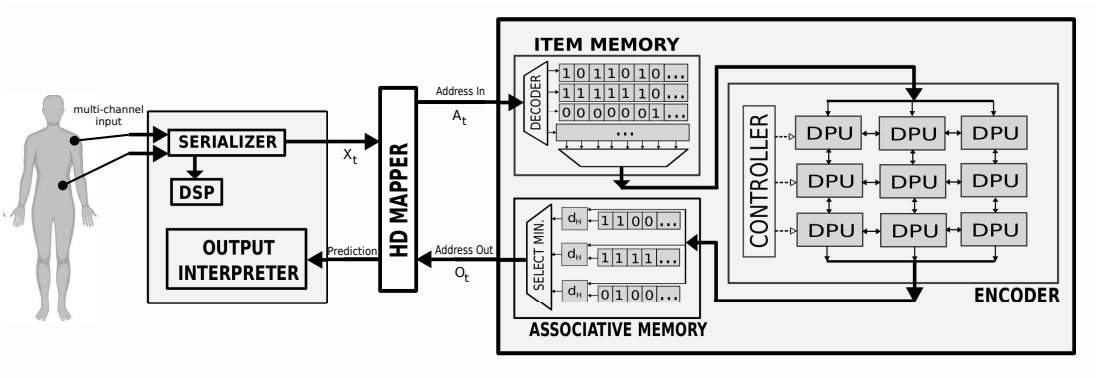


Figure 3-1: **The *Generic* HD Processor**: A body-sensing application is illustrated with two-channel input streams, and major components of the processor alongwith the dataflow during testing are shown. All application-specific peripherals are to the left of *HD Mapper*. Encoder is the only programmable component, and systolic array is the most suitable architecture.

3.2 Common Algorithmic Kernels

Only a few basic expressions are used repeatedly in most HD algorithms. In particular, all applications in the benchmark in Table 2.1 requires the following 2 kernels.

- **n-gram Sequence Encoding**: As mentioned in Section 2.3, n-grams or the multi-set of n-subsequences can be a very useful for modelling sequences. Applied widely for DNA sequencing and text analysis, this multi-set is enough to reconstruct long chunks of original data [62, 63], which is especially useful for classification.

  Characters in an *symbol set* $\mathbb{A} \triangleq \{a_1, a_2, ..., a_N\}$ are mapped to random hyper-vectors $Y_{a_1}, Y_{a_2}, ..., Y_{a_N}$ and the n-subsequence $x_1, x_2, ...x_n$ (where $x_i \in \mathbb{A}$) is encoded as $Y_{x_1} \oplus \rho(Y_{x_2}) \oplus \rho^2(Y_{x_3}) ... \oplus \rho^{n-1}(Y_{x_n})$. Finally, all occurring n-subsequences in the input are encoded and superimposed to form the final hyper-vector.

- **Feature Superposition**: This is used to map input feature vectors into a hyper-vector. Let the feature vector of d dimensions be $V = (v_1, v_2, ..., v_d)$. For each *vector position* $i \in [d]$, a random hyper-vector C_i is generated and as discussed in Section 3.1.1, all possible input values v are assigned hyper-vectors

Y_v. Then V is encoded as $[\sum_{i=1}^{d}(C_i \oplus Y_{v_i})]$; and a collection of n samples (a matrix U with sample vectors as rows) is encoded as the superposition $[\sum_{i=1}^{n}\sum_{j=1}^{d}(C_j \oplus Y_{U_{ij}})]$.

These kernels may be combined to perform dual-stage encoding as well. *EMG Hand-Gesture Recognition* in Table 2.1 is the only dual-stage algorithm in the benchmark, computing 4-grams of 64-feature samples.

3.3 Organization of the HD Processor

A *generic* HDC processor for supervised classification requires **three major components** corresponding to the major steps of processing illustrated in Section 2.3

- **Item Memory** stores a repertoire of random hyper-vectors (**items**). A sufficiently large collection of such vectors can be re-used for many applications. This storage requirement *cannot be avoided* as the map of symbols to items *must be the same* during training and testing.

- **Encoder** combines the input hyper-vector sequence according to a pre-specified algorithm to form **single hyper-vector** for each class.

- **Associative Memory** stores the trained class hyper-vectors. During testing, the class hyper-vector closest to the encoded test hyper-vector is returned as final prediction.

Fig. 3-1 shows a diagram for the complete system. All application specific preprocessing, sampling and quantization is done before serializing the input streams. The peripheral **HD Mapper** assigns incoming symbols to an *item* in Item Memory and class labels to *Associative Memory addresses*. This mapping is *retained* for all sessions of training and testing. Therefore, the *actual input* to the generic processor is a time-series of *Item Memory addresses*. Fig. 3-1 also shows the operation during test. The Item Memory fetches input hyper-vectors and Encoder generates the test hyper-vector. The Associative Memory *returns the address* of the closest class vector.

The actual label is substituted back by *HD Mapper* for further consumption.

When the processor is abstracted in this manner, two *crucial* properties emerge:

1. **Uni-directional Data-flow:** For all applications during training and testing, input hyper-vectors always flow from Item Memory to Encoder and end in Associative Memory. No class vector from Associative Memory needs to be loaded into the Encoder. There are **no iterations** over the input sequence as well.

2. **Single programmable component:** Only the Encoder needs to be programmed for an application. The operation of *both* memories *always remain the same*.

Therefore, all major architectural decisions principally concern the Encoder. Since it only performs HD operations, it is important to note the parallelism in each of them. For *superposition* and *multiply*, an element of the result vector depends *only on corresponding elements* of its operands. *Permutation* is the only operation with *dependency across vector elements*, where a result element depends on a neighbouring operand element. Clearly, *data-parallel* architectures are the most suitable candidates.

To begin with, any *fixed-width* architecture with width less than HD dimension D is inefficient. An internal memory for storing intermediate hyper-vectors is very expensive due to high dimensionality. Therefore, such an encoder needs to *iterate over entire input* for each sub-word and consequently must store the *full sequence* $\mathbb{I}$. For some applications, input data (esp. for training) are very long ($T \approx 1$ million for *Language Recognition*) and required storage (≈ 1 MB) can exceed size of memory components (a 1000-item 2048-dim. Memory needs 256 KB). Furthermore, permutation creates **intra-word dependencies** which leads to *redundant computations*, and *multiple sub-words* may be required for intermediate values (see Fig. 3-2).

Vector and *SIMD* architectures with data-width D are expensive as well. HD algorithms are too small to extract significant run-time parallelism to justify the over-

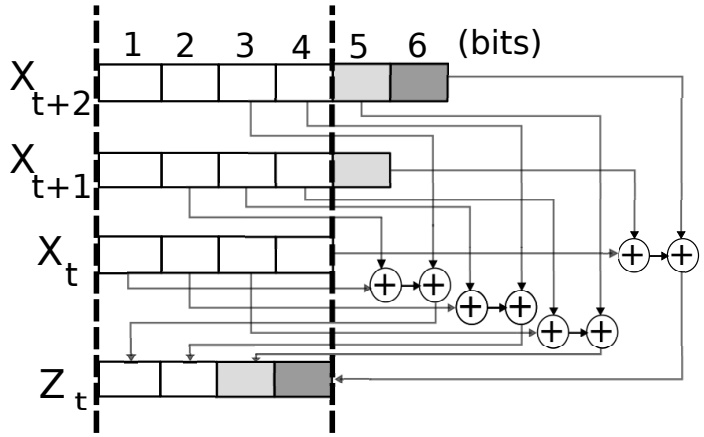


Figure 3-2: Encoding steps required for 3-gram Z_t of hyper-vectors X_t, X_{t+1}, X_{t+2} are shown. Sub-words are 4 bits wide (within dashed lines), computed one at a time. The module $\oplus$ performs *binding*. Values from external sub-words (bits 5 and 6) and dependent outputs are grey-coded.

head. Finally, such architectures require wide register-files and complex control logic, contradicting the advantages of performing only simple bit-wise operations *permute*, *multiply* and *superposition*.

Considering all factors described above, **data-flow based Array architecture** is the most suitable choice (see Fig. 3-1). Here, the encoder is comprised of a regular network of simple **Data Processing Units** (DPU), and inter-DPU communication for program dependencies is restricted to neighbors no more than fixed distance away [64]. Though several attempts have been made to map common workloads to DPUs [65, 66, 67, 68, 69], only few of them where dependency patterns can be expressed as a regular graph have been successful [64, 70, 71] . HD algorithms fit perfectly to these conditions. All HD operations can be implemented with a few gates. The sequential input model and the *generic abstraction* developed in Section 3.1.3 enables us to map algorithms to DPUs explicitly. Section 4.2 describes the Encoder implementation and programming in detail.

Chapter 4

Implementation and Programmability

This chapter describes the hardware implementations of each major component in detail. *Multi-processor scalability* is also described.

4.1 Item Memory

Choice of Item Hyper-vectors: The first step is to chose the set of item hyper-vectors used by the processor. There is a non-zero (though *vanishingly* small) probability of random hyper-vectors being very close. In fact, the small variations from orthogonality (Fig. 2-1) results in slight deviations (<0.1 %) in application accuracy with different choice of items. Therefore it is a good practise to generate multiple sets of items and test them on some token applications first. For this work, 20 sets of item vectors were generated and the one with best average accuracy for benchmark applications was selected.

***Continuous* Item Generation:** Representing integers or values from *any ordered set* by assigning orthogonal vectors may not be appropriate. Ideally, two close numbers should have a corresponding small distance in their hyper-vectors. A possible solution [72]is to assign points on a line connecting two *exactly* orthogonal vectors.

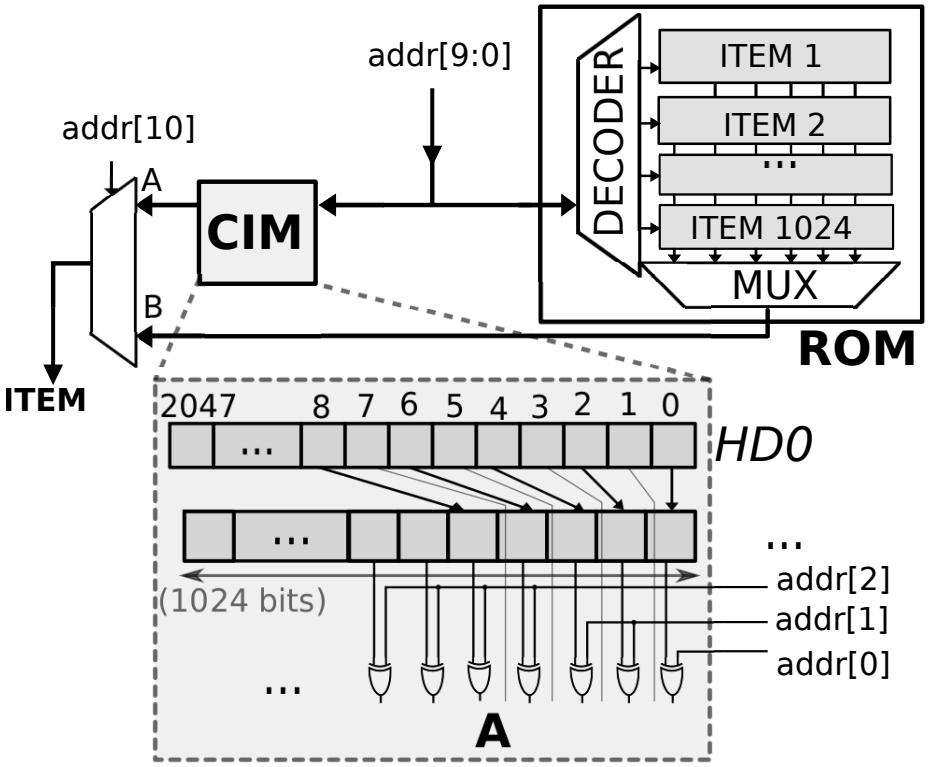


Figure 4-1: **Item Memory implementation**: The main block of orthogonal vectors are stored in a ROM. The Continuous Item Memory (CIM) module generates *correlated* hyper-vectors for representing integers.

Property	**Value**
Type	High-density, Via-programmed
Area	3020 sq. μm
Minimum t_{CLK}	0.71 ns
Clock to Valid Q (t_{cd})	0.51 ns
Dynamic Power (Read)	4 μA/MHz
Dynamic Power (Sleep)	59 nA/MHz
Leakage Current	6.55 μA

Table 4.1: Characterization of ROM1024×8 at (0.9V, 25°C, TT) corner.

Then any collection of 3 hyper-vectors A, B, C (representing integers $a \leq b \leq c$) will satisfy the triangle law $d_H(A,B)+d_H(B,C) = d_H(A,C)$. Equivalently, one can begin with a randomly generated *origin vector* `HD0` and a direction vector `Y` with $(D/2)$-bits being 1. The smallest integer, usually 0, is mapped to `HD0` and the largest integer, say M, is mapped to `HD0` $\oplus$ `Y`. For all other integers n, flip $D/2/M$ additional bits along direction `Y` from the hyper-vector assigned to $n-1$. The only restriction is M divides $D/2$.

A **Read-Only Memory (ROM)** was chosen to store the constant item hyper-vectors generated offline (Fig. 4-1). This is the simplest possible implementation suited for this preliminary design. An alternative is to use pseudo-random generators to generate them *online*, but their storage cannot be avoided because the same items must be used for training and testing. For the applications in the benchmark (Table 2.1), 1024 orthogonal items suffice. Item hyper-vectors are divided into 8-bit sub-words and stored separately in 256 instances of $ROM1024 \times 8$. Table 4.1 presents its characteristics.

Fig. 4-1 shows a 2048-dim Item Memory with 11-bit address (`addr`). Items in lower address-space (`addr`$[10] = 0$) are stored in ROM and are mutually orthogonal. The upper address space (`addr`$[10] = 1$) is used by *Continuous Item Memory (CIM)* to translate integer inputs. `HD0` is loaded from ROM during setup. For simplicity, `Y` was chosen so that only odd bits are 1. Since $M = 1024$ each integer vector requires $D/2048 = 1$ extra flips along Y (i.e. the *next odd bit* of `HD0`) from the previous integer vector.

4.2 Encoder

The Encoder is crucial for overall programmability of the processor, and has the largest activity and wiring complexity. As outlined earlier, this component must be organized into multiple redundant computing elements. (Sec. 3.3, Fig. 3-1).

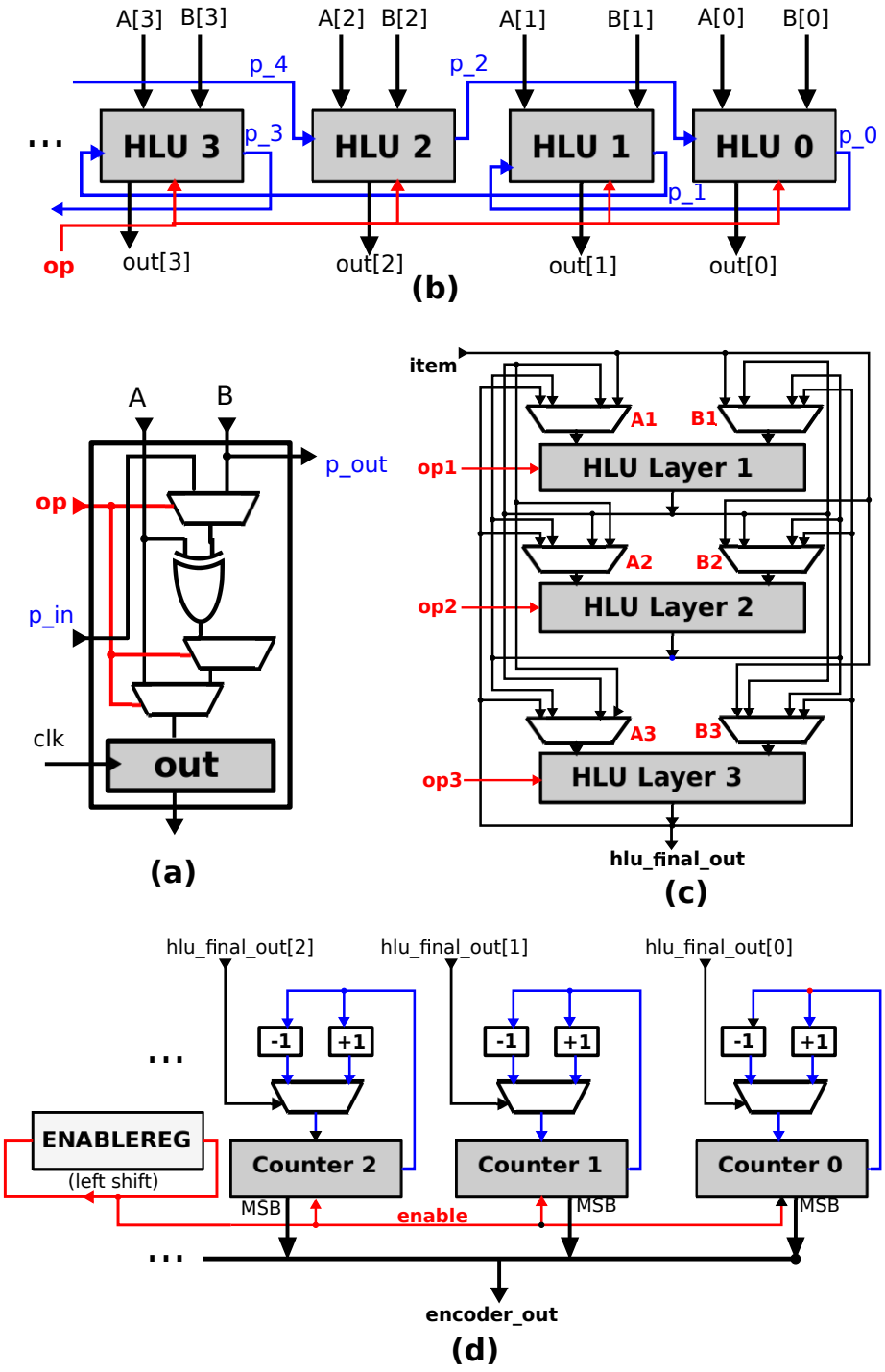


Figure 4-2: **Encoder organization**: (Clock-wise from left) (a) Hyper-Dimensional Logic Unit (HLU), (b) connecting single-bit HLUs to create HLU Layer, (c) Interconnecting multiple HLU layers, and (d) Accumulator for super-position

4.2.1 Organization of the Encoder

Generic algorithms can be decoupled into generation of *terms* and super-position. Clearly, the Data Processing Unit *(DPU) network* need only generate all necessary terms, hence it suffices to only implement *multiply* amd *permute* in them.

Fig. 4-2(a) shows the **Hyper-dimensional Logic Unit (HLU)**, the simplest such DPU possible. HLU takes two single-bit operands A and B and can **multiply** ($C = A \oplus B$), **permute** ($C = \rho(A)$), **delay** ($C = A$) and **permute-and-multiply** ($C = A \oplus \rho(B)$).

Since *permute* has intra-word dependencies, D HLUs can be connected together to form a module operating on entire hyper-vector (Fig. 4-2(b)). This coherent unit will be called **HLU Layer**. Each constituent HLU performs the *same* operation on the input hyper-vector. *Permute* is a single-cycle derangement, hence any *Hamiltonian* path connection through `p_in` and `p_out` visiting all HLUs is valid. Fig. 4-2(b) illustrates a scheme where alternate HLUs (except first and last) are connected to minimize length of longest wire.

HLU Layers can be interconnected among themselves, generating an overall output `hlu_final_out` by transforming the stream of inputs (`item`) from the Item Memory (Fig. 4-2(c)).

Finally, a simple array of **Accumulators** perform the *super-position* (Fig. 4-2(d)). A two's-complement counter at a hyper-vector position *increments* or *decrements* according to corresponding bits of `hlu_final_out` being 1 or 0. When encoding completes, the vector formed by MSB of counters is the required superposition. Note that *accumulator* takes in hyper-vectors (`hlu_final_out`) calculated by the *last HLU Layer*. In Fig. 4-2(d) the shift-register `ENABLEREG` is programmed to mark the cycles of arrival of required terms and enable the counters for accumulation (see Section 3.1.3). Its contents are shifted as the design encodes, accumulating *only the required* terms. For the benchmark applications, a 256-bit register is sufficient.

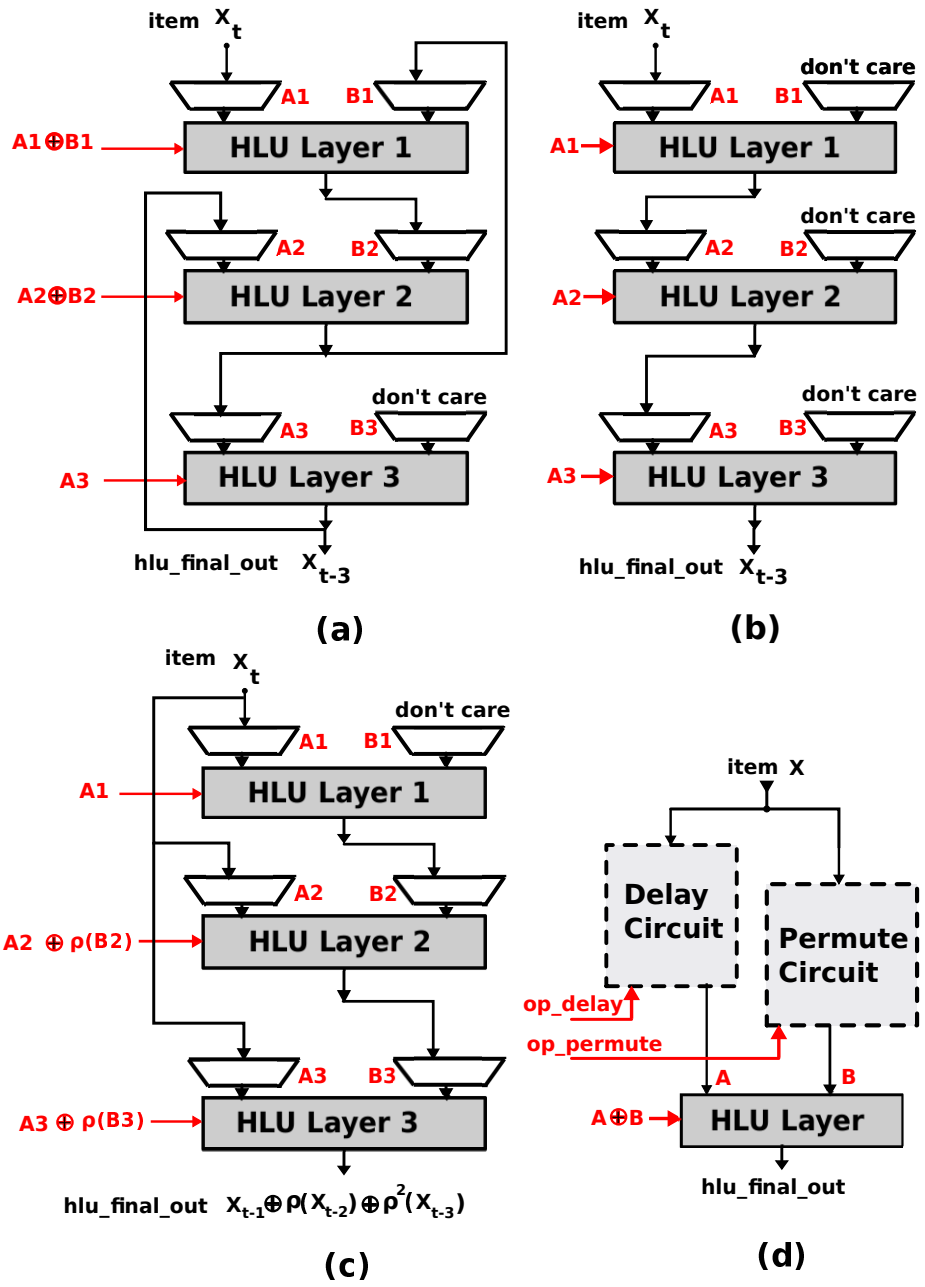


Figure 4-3: **Examples of HLU Interconnection**: (a) A feedback implementation of input X_t delayed by 3 cycles. (b) A feed forward implementation of input X_t delayed by 3 cycles. (c) A feed forward implementation of 3-gram. (d) In general, *delay partial terms* and *permute partial terms* can be separately computed and combined to get a *term expression*.

4.2.2 Programmability

A crucial factor for array architectures is the choice of interconnection network. More general networks allow efficient mapping of algorithms at the cost of increased hardware complexity and power consumption.

For the Encoder, HLU Layers are the only independent processing elements. They *compute coherently* and have two hyper-vector operands. Therefore, the **major decision** here is *the set of allowed operands* to each HLU Layer. The most general network would allow output of any HLU Layer or Item Memory to be *either operand* for *all* HLU Layers.

Fig. 4-2(c) provides an example Encoder with 3 layers. Signals `op1, op2, op3` program the operation carried out by HLU Layer 1, 2, 3. For *generic* programs *term expression* is a constant, hence these operation signals stay the same throughout the encoding. Operand-select signals `A1, B1, A2, B2, A3, B3` decide the actual interconnections of HLU Layers. Note that in this setting **feedback** is allowed: an HLU Layer's output may even be its own input in the next cycle.

Feedback offers greater variety for most term expressions. Fig. 4-3(a) and 4-3(b) are two HLU networks with feed-back and feed-forward configurations respectively. Both of them transform the input sequence X_t to the *same* final output `hlu_final_out` $= X_{t-3}$. Finding equivalent networks with feedback usually requires exhaustive search through all possible interconnections.

However, wiring complexity grows *quadratically* with number of HLU Layers, making general interconnections infeasible for large designs. It also complicate overall pipeline control as flushing or filling a pipeline with arbitrary feed-back connections is non-trivial. Fortunately, the benchmark applications do not need such large designs. Feed-forward implementations for term expressions are easy to find and the *generic* abstraction from Section 3.1.3 helps. Eq. 3.1 separates a *term* $f_i(\mathbb{I})$ into two products, one **delay partial term** with inputs and their delayed versions only $(X_{p_1} \oplus X_{p_2} ... \oplus X_{p_m})$ and another **permute partial term** with their permuted versions

$(\rho^{u_1}(X_{q_1}) \oplus \rho^{u_2}(X_{q_2}) ... \oplus \rho^{u_n}(X_{q_n}))$. They can be implemented by separate interconnections of HLU **Delay Circuit** and **Permute Circuit** respectively, and combined to give the final output (Fig. 4-3(d)).

4.3 Associative Memory

Associative Memory stores the learned vectors for a later comparison and retrieval (see Fig. 4-4). During training, the `write_en` signal enables only the address location pointed by `label_in` for writing. The output from Encoder (`encoder_out`) is saved into the corresponding address. During testing, the `encoder_out` contains the test hyper-vector whose *hamming distance* is computed to each class vector *in parallel*. The number of mis-matches is computed by an element-wise XOR followed by counting the number of 1s using `popcount` logic.

A simple module was designed that can store 32 classes and each class vector has a separate `popcount` logic attached. The number of 1s in 256-bit sub-words is counted in a single cycle by `popcount`; 8 cycles overall for 2048-dim hyper-vectors. Since the amount of storage required is small (32× 2048-dim hyper-vectors is about 8 KB), flip-flops were used instead of SRAMs.

4.4 Multi-processor Configuration

A simple configuration of multiple processors results in increased effective dimension. To demonstrate this, consider *Language Recognition* using 27 input characters (lowercase alphabets and white-space). Suppose two instances of the 2048-dim processor (with *same* items in Item Memory) are used for *Language Recognition*. The 27 input symbols are randomly mapped to 27 of 1024 available items in the first processor's *HD Mapper 1*. The second processor's *HD Mapper 2* chooses a *new* map of 27 symbols to 1024 items. On average, the two maps have about $27^2/1024 = 0.71$ common symbol-item pairs. Hence, they almost certainly compute with *completely different* items which is equivalent to computation on 4096-dim data-path with *freshly generated*

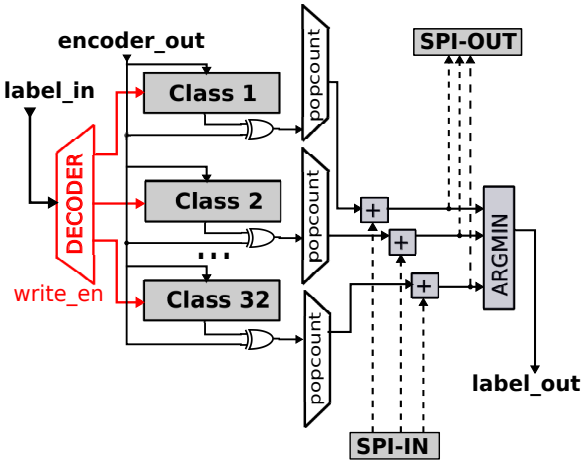


Figure 4-4: Associative Memory: the implemented design with capacity of 32 classes is shown. Additional logic for handling *multi-processor* mode is shown as well. When configured in *multi-processor* mode, distances from *previous processor* are added to local distances and sent to the *next processor*.

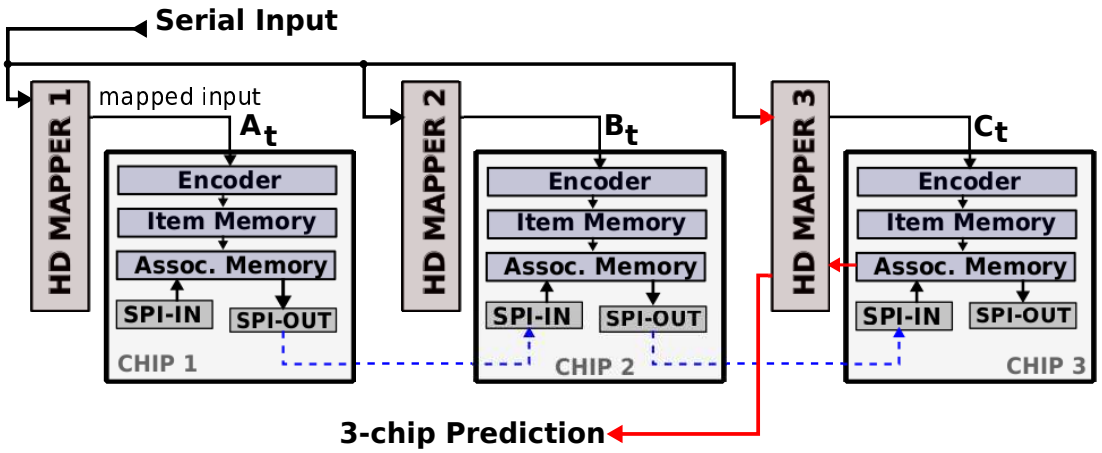


Figure 4-5: *Mult-processor* Operation: 3 instances of the 2048-dim processor (with identical Item Memory) are connected to improve effective dimension to $D_{eff} = 6144$.

items.

Fig. 4-5 shows a system of three 2048-dim processors interconnected to simulate $D_{eff} = 6144$. Class hyper-vectors are trained and stored locally in each processor. Note that *label-address* map for Associative Memory **must be the same** for all three HD Mappers. During testing, the test-vector and distances to stored class hyper-vectors are computed *locally* by each processor. For each class, these local distances are added to generate a *total distance*. The class with smallest *total distance* is returned as prediction.

A *Serial Peripheral Interface* (SPI) module is used for transmitting local distances to adjacent processors (see Fig. 4-5). The associative memory adds local distances to those from its SPI *slave* (`SPI-IN`) before transmitting to its SPI *master* (`SPI-OUT`) (see Fig. 4-4). As the processors are connected linearly, the *last* processor computes *total distances* and gives the final prediction.

Chapter 5

Experimental Results

5.1 Data-path Overview

Fig. 5-1 provides a simplified block-diagram of the entire 2048-dim processor. The Item Memory has 11-bit address with 1024 randomly generated items stored in ROM. The Associative memory can store up to 32 classes.

The designed Encoder has only as much resources as required by the benchmark applications. A total of 7 HLU layers are split into two groups `G1` and `G2`. HLU Layers 1 and 2 form group `G1`, where *Accumulator* 1 performs superposition of Layer 2 outputs. HLU Layers 3 to 7 form group `G2` where *Accumulator* 2 performs superposition of Layer 7 outputs. `ENABLEREG-1` and `ENABLEREG-2` are 256-bit shift registers implementing the control signals for enabling Accumulator 1 and Accumulator 2 respectively.

To allow **dual-stage encoding**, `ENABLEREG-1` from group `G1` can provide a **global enable** signal for all logic in `G2` as well. Finally, note that the HLU Layer interconnections are not fully general. The item hyper-vector can go only to Layer 1; Layers 1 and 2 are fully-connected; and Layers 3 to 7 are fully-connected with Layer 2 and Accumulator 1.

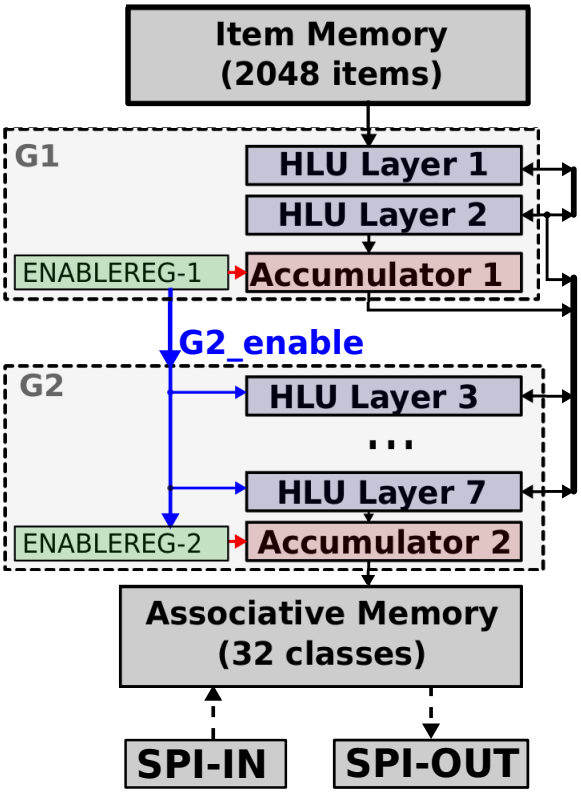


Figure 5-1: Summary of resources in the implemented 2048-dim processor. Can handle all applications requiring up to 2,048 items and 32 classes. The Encoder has 7 HLU Layers with 2 accumulators allowing *dual-stage* computation.

5.2 Hardware Complexity

A System-Verilog RTL for the processor was trained and tested for all applications in the benchmark. The data-path was synthesized with 28nm *High-K/Metal-gate* (HKMG) cells provided by TSMC under a clock-cycle constraint of $t_{CLK} \leq 2.4$ ns. The power consumption was estimated at (0.8 V, 25°C, TT) corner. Table 5.1 lists the main results of the designed hardware. The primary goal is to quantify the effects of the general architecture alone on the processor performance and efficiency. Hence no specialized library cells or circuits were utilized to optimize for area or power consumption.

Fig. 5-3 shows the component-wise breakdown of area and power consumption. Item Memory is the largest component as it stores *over a thousand* hyper-vectors. The principal contributor to the Encoder's size are the integer counters in its two

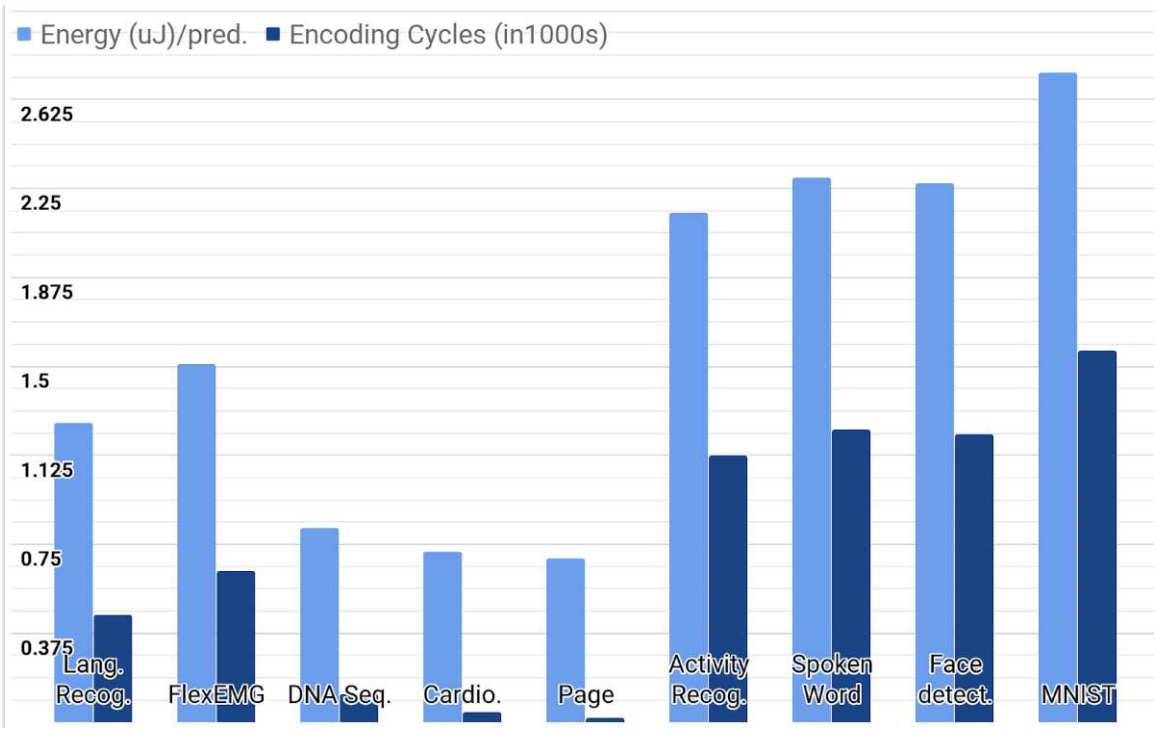


Figure 5-2: **Single-chip Energy Efficiency**: shown energy consumed [μJ] and number of cycles (in *thousands*) spent in Encoder for testing one sample on *single* processor.

Accumulators. Each component of the Accumulator hyper-vector needs a 22-bit register compared to single-bit registers elsewhere. The major contribution in Associative Memory is the dedicated `popcount` logic for each class, adding to overall area and power consumption. Most importantly, the two **memory components** are the **main contributors** to overall power and area. More efficient implementation, such as analog content-addressable memory [25], would reduce the overall cost significantly.

5.3 Energy Efficiency

To estimate the energy cost of a prediction, the configuration providing highest accuracy for the application must be considered. Fig. 5-4(a) shows improvements in HDC accuracy as number of processors connected in *multi-chip mode* increases. Results for *EMG Hand-Gesture Recognition (FlexEMG), DNA Sequencing (DNA)* and *Page-Block Classification (PAGE)* are *already* better than baseline on a single proces-

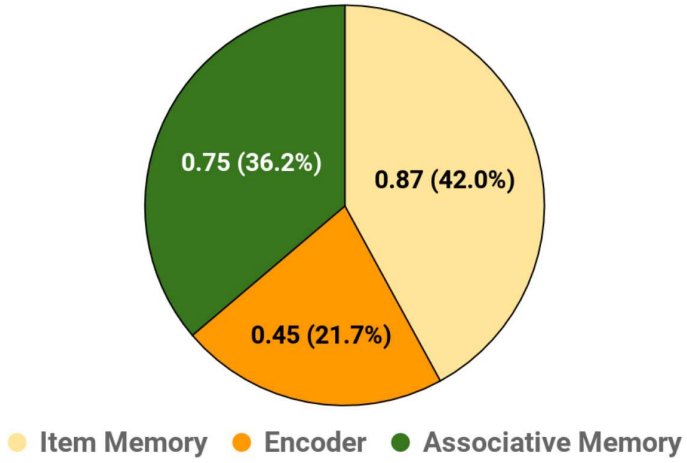
sor (Table 2.1). They improve negligibly with *multi-processor* scaling.

Fetal state Classification (CARDIO) has constant accuracy equalling the baseline for almost all configurations. *Language Recognition (LANG)* accuracy improves the most when 5 chips are interconnected, but is *slightly* lower than baseline. Finally, results for *UCI Human-activity Recognition (UCIHAR), Spoken Letter Classification (ISOLET), Face detection (FACE)* and *MNIST Digit Recognition* are lower than baseline for all configurations. Connecting more than 5 chips together ($D_{eff} > 10240$) increases energy consumption with negligible accuracy gains.

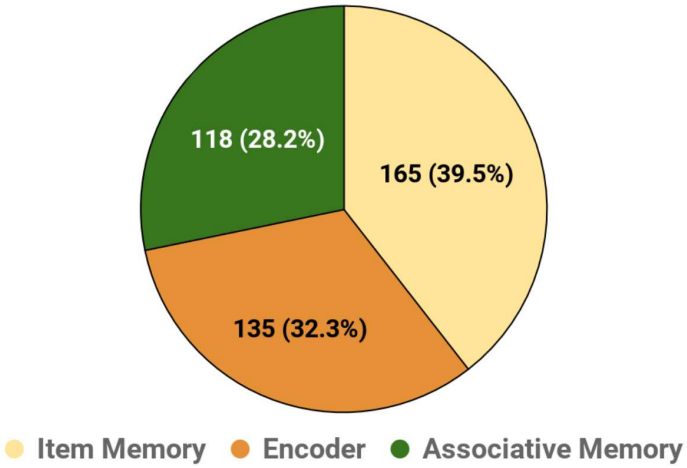
Fig. 5-2 shows the energy per prediction on a single-processor system. This suffices for applications FlexEMG, DNA, CARDIO and PAGE, with energy efficiency better than **1.6 μJ/pred**. LANG consumes about 6μJ/pred. on the 5-chip system. For comparison, [73] implements a $10,000$-dim HD algorithm for EMG on the 4-core PULPv3 processor [74], operated at 0.7 V, to give 25.6μJ/pred. A 5-chip implementation of EMG on this processor ($D_{eff} = 10240$) consumes only 7.55μJ/pred (3.3$\times$ improvement).

Property	Value
Technology	TSMC 28 HPM
Total Cell Area	2.07 sq. mm.
t_{CLK}	2.4 ns
Total estimated power	418 mW

Table 5.1: QoR report for the synthesized 2048-dim Generic HD Processor. The power is estimated for (0.8 V, 25° C, TT) corner.

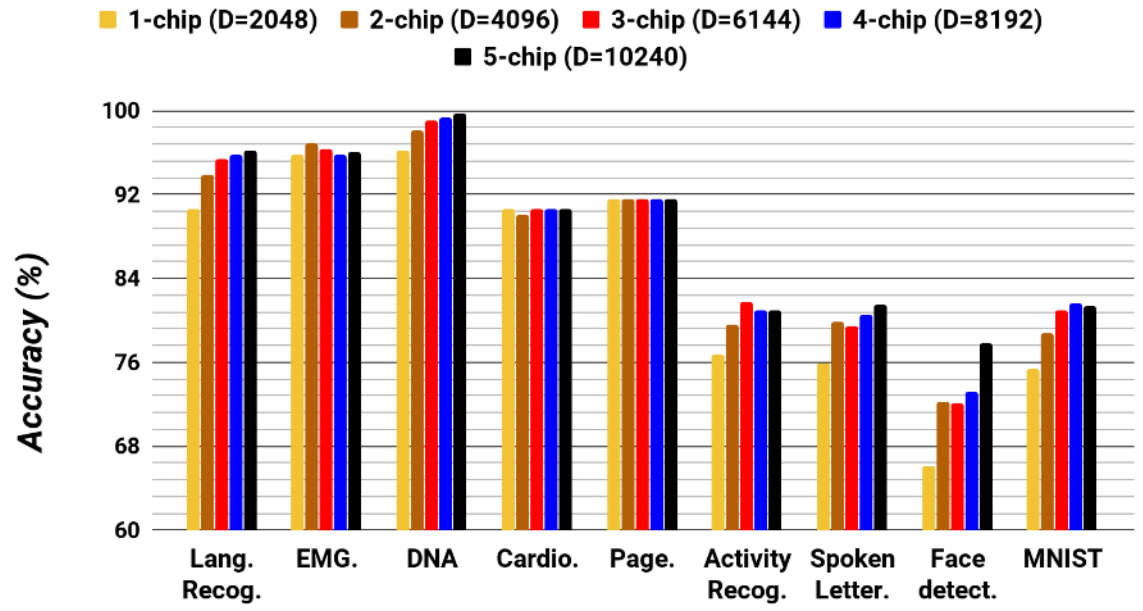


(a) Area [mm^2]

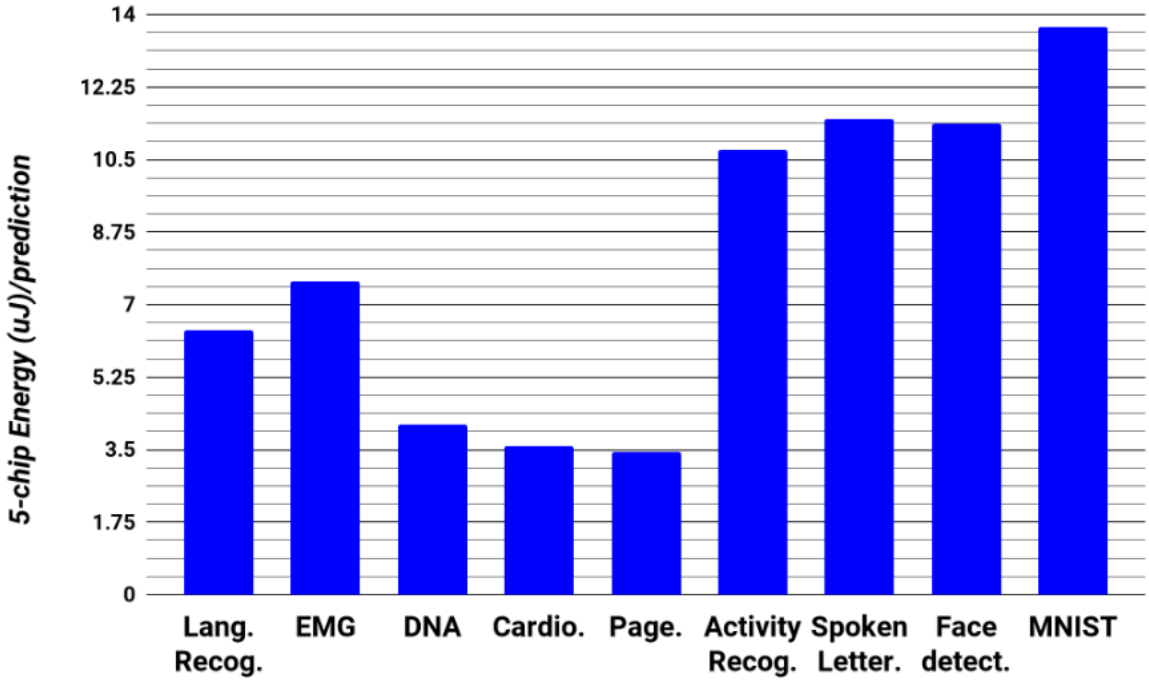


(b) Power [mW]

Figure 5-3: **(a) Area Breakdown**: the mapped area of major components is shown. **(b) Power Breakdown**: the power consumption of major components is shown.



(a) Benchmark Accuracy with *multi-processor* configuration



(b) 5-processor Energy [μJ] per Prediction

Figure 5-4: **(a) Accuracy in Multi-processor configuration**: up to five 2048-dim processors are connected together to increase effective dimension, thereby improving accuracy. **(b) Multi-processor Energy Efficiency**: shown energy consumed [μJ] for testing one sample using *five* processors configured in *multi-processor* mode.

Chapter 6

Conclusion

The basic components of a *generic* Hyper-dimensional processor were developed and the major choices required for implementation were elaborated. The data path was synthesized in 28nm HKMG, exhibiting an energy efficiency better than **13.9 μJ/pred**. for all benchmark applications. The 4 classification tasks with highest 1-processor HDC accuracy exhibit energy efficiency better than **1.6 μJ/pred**.

The *generic abstraction* (Section 3.1.3) allows efficient mapping of HD expressions to the developed architecture. For the benchmark applications chosen, an Encoder with only 7 HLU Layers suffices. However, the architecture allows design of Encoders with more HLU Layers and Accumulators as well. A future direction would be to extend this architecture to perform hierarchical HD computing [30].

The choice of distance metric is another crucial factor determining overall performance. An advanced data-path using **cosine similarity** instead of *hamming distance* would improve accuracy significantly [56, 52].

As a preliminary design of the generic processor, the energy efficiency for some applications are encouraging. An optimized system could be deployed for ultra low-power computing in the future.

Bibliography

[1] C. A. Mack, "Fifty years of moore's law," *IEEE Transactions on Semiconductor Manufacturing*, vol. 24, pp. 202–207, May 2011.

[2] "Celebrating the 50th anniversary of moore's law," *IHS Markit Techincal Report*, 2015.

[3] A. M. Ben-Amram, "The church-turing thesis and its look-alikes," *SIGACT News*, vol. 36, pp. 113–114, Sept. 2005.

[4] B. Pandolfini, *Kasparov and Deep Blue: The Historic Chess Match Between Man and Machine*. 1997.

[5] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, "Stanley: The robot that won the darpa grand challenge: Research articles," *J. Robot. Syst.*, vol. 23, pp. 661–692, Sept. 2006.

[6] J. Markoff, "Computer wins on 'jeopardy!': Trivial, it's not," *The New York Times*, Feb. 2016.

[7] N. S. J. I. Wong, "Google's ai won the game go by defying millennia of basic human instinct," *Quartz Magazine*, Mar. 2016.

[8] T. C. Sottek, "The world's best dota 2 players just got destroyed by a killer ai from elon musk's startup," *The Verge*, Aug. 2017.

[9] R. Kurzweil, *The Singularity Is Near: When Humans Transcend Biology*. 2006.

[10] C. Snijders, U. Matzat, and U.-D. Reips, "" big data": big gaps of knowledge in the field of internet science," *International Journal of Internet Science*, vol. 7, no. 1, pp. 1–5, 2012.

[11] J. D. Lafferty and L. Wasserman, "Challenges in statistical machine learning," *Statistica Sinica*, vol. 16, p. 307, 2006.

[12] S. LeVine, "Artificial intelligence pioneer says we need to start over," *Axios*, Sept. 2015.

[13] J. Fan and R. Li, "Statistical challenges with high dimensionality: Feature selection in knowledge discovery," *arXiv preprint math/0602133*, 2006.

[14] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, pp. 10–16, Nov 2005.

[15] R. Courtland, "Transistors could stop shrinking in 2021," *IEEE Spectrum*, vol. 53, no. 9, pp. 9–11, 2016.

[16] T. N. Theis and H.-S. P. Wong, "The end of moore's law: A new beginning for information technology," *Computing in Science & Engineering*, vol. 19, no. 2, pp. 41–50, 2017.

[17] Y.-B. Kim, "Challenges for nanoscale mosfets and emerging nanoelectronics," *Transactions on Electrical and Electronic Materials*, vol. 11, no. 3, pp. 93–105, 2010.

[18] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, pp. 253–266, Feb 2010.

[19] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, pp. 365–376, June 2011.

[20] J. Von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata studies*, vol. 34, pp. 43–98, 1956.

[21] R. Sarpeshkar, "Analog versus digital: extrapolating from electronics to neurobiology," *Neural computation*, vol. 10, no. 7, pp. 1601–1638, 1998.

[22] A. Rahimi, S. Datta, D. Kleyko, E. P. Frady, B. Olshausen, P. Kanerva, and J. M. Rabaey, "High-dimensional computing as a nanoscalable paradigm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2508–2521, 2017.

[23] 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.

[24] P. Kanerva, *Sparse distributed memory*. MIT press, 1988.

[25] 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)*, pp. 445–456, Feb 2017.

[26] 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*, ISLPED '16, (New York, NY, USA), pp. 64–69, ACM, 2016.

[27] T. F. Wu, H. Li, P. C. Huang, A. Rahimi, J. M. Rabaey, H. S. P. Wong, M. M. Shulaker, and S. Mitra, "Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pp. 492–494, Feb 2018.

[28] H. Li, T. F. Wu, A. Rahimi, K. S. Li, M. Rusch, C. H. Lin, J. L. Hsu, M. M. Sabry, S. B. Eryilmaz, J. Sohn, W. C. Chiu, M. C. Chen, T. T. Wu, J. M. Shieh, W. K. Yeh, J. M. Rabaey, S. Mitra, and H. S. P. Wong, "Hyperdimensional computing with 3d vrram in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition," in *2016 IEEE International Electron Devices Meeting (IEDM)*, pp. 16.1.1–16.1.4, Dec 2016.

[29] M. Imani, J. Hwang, T. Rosing, A. Rahimi, and J. M. Rabaey, "Low-power sparse hyperdimensional encoder for language recognition," *IEEE Design Test*, vol. 34, pp. 94–101, Dec 2017.

[30] M. Imani, C. Huang, D. Kong, and T. Rosing, "Hierarchical hyperdimensional computing for energy efficient classification," in *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, (New York, NY, USA), pp. 108:1–108:6, ACM, 2018.

[31] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *2017 IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1–8, Nov 2017.

[32] M. Imani, T. Nassar, A. Rahimi, and T. Rosing, "Hdna: Energy-efficient dna sequencing using hyperdimensional computing," in *2018 IEEE EMBS International Conference on Biomedical Health Informatics (BHI)*, pp. 271–274, March 2018.

[33] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, "On the surprising behavior of distance metrics in high dimensional space," in *International conference on database theory*, pp. 420–434, Springer, 2001.

[34] A. Hinneburg, C. C. Aggarwal, and D. A. Keim, "What is the nearest neighbor in high dimensional spaces?," in *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, (San Francisco, CA, USA), pp. 506–515, Morgan Kaufmann Publishers Inc., 2000.

[35] H.-P. Kriegel, P. Kröger, and A. Zimek, "Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering," *ACM Trans. Knowl. Discov. Data*, vol. 3, pp. 1:1–1:58, Mar. 2009.

[36] I. M. Johnstone and D. M. Titterington, "Statistical challenges of high-dimensional data," 2009.

[37] T. A. Plate, "Holographic reduced representations," *IEEE Transactions on Neural networks*, vol. 6, no. 3, pp. 623–641, 1995.

[38] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[39] P. Kanerva, "The spatter code for encoding concepts at many levels," in *ICANN'94*, pp. 226–229, Springer, 1994.

[40] P. Kanerva, J. Kristoferson, and A. Holst, "Random indexing of text samples for latent semantic analysis," in *Proceedings of the Annual Meeting of the Cognitive Science Society*, vol. 22, 2000.

[41] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen, "A large-scale model of the functioning brain," *science*, vol. 338, no. 6111, pp. 1202–1205, 2012.

[42] R. W. Gayler, "Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience," *arXiv preprint cs/0412059*, 2004.

[43] A. DasGupta, *Normal Approximations and the Central Limit Theorem*, pp. 213–242. New York, NY: Springer New York, 2010.

[44] M. Okamoto, "Some inequalities relating to the partial sum of binomial probabilities," *Annals of the institute of Statistical Mathematics*, vol. 10, no. 1, pp. 29–35, 1959.

[45] P. Kanerva, "Some properties of the space $\{0, 1\}^n$," in *Sparse distributed memory*, MIT press, 1988.

[46] S. Boucheron, G. Lugosi, and P. Massart, *Concentration inequalities: A nonasymptotic theory of independence*. Oxford university press, 2013.

[47] E. P. Frady, D. Kleyko, and F. T. Sommer, "A theory of sequence indexing and working memory in recurrent neural networks," *Neural Computation*, vol. 30, no. 6, pp. 1449–1513, 2018. PMID: 29652585.

[48] T. Vatanen, J. J. Väyrynen, and S. Virpioja, "Language identification of short text segments with n-gram models.," in *LREC*, 2010.

[49] J. F. D. Silva and G. P. Lopes, "Identification of document language is not yet a completely solved problem," in *2006 International Conference on Computational Inteligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce (CIMCA'06)*, pp. 212–212, Nov 2006.

[50] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," *Computer Speech & Language*, vol. 13, no. 4, pp. 359–394, 1999.

[51] A. Moin, A. Zhou, A. Rahimi, S. Benatti, A. Menon, S. Tamakloe, J. Ting, N. Yamamoto, Y. Khan, F. Burghardt, L. Benini, A. C. Arias, and J. M. Rabaey, "An emg gesture recognition system with flexible high-density sensors and brain-inspired high-dimensional classifier," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, May 2018.

[52] M. Imani, T. Nassar, A. Rahimi, and T. Rosing, "Hdna: Energy-efficient dna sequencing using hyperdimensional computing," in *2018 IEEE EMBS International Conference on Biomedical Health Informatics (BHI)*, pp. 271–274, March 2018.

[53] N. Chamidah and I. Wasito, "Fetal state classification from cardiotocography based on feature extraction using hybrid k-means and support vector machine," *2015 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pp. 37–41, 2015.

[54] A. Bagirov and J. Ugon, "An algorithm for computation of piecewise linear function separating two sets," 08 2018.

[55] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *International workshop on ambient assisted living*, pp. 216–223, Springer, 2012.

[56] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *2017 IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1–8, Nov 2017.

[57] Y. Kim, M. Imani, and T. Rosing, "Orchard: Visual object recognition accelerator based on approximate in-memory processing," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 25–32, Nov 2017.

[58] J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, (Washington, DC, USA), pp. 3642–3649, IEEE Computer Society, 2012.

[59] A. Coates and A. Y. Ng, "The importance of encoding versus training with sparse coding and vector quantization," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 921–928, 2011.

[60] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[61] P. Kanerva, "What we mean when we say" what's the dollar of mexico?": Prototypes and mapping in concept space.," in *AAAI fall symposium: quantum informatics for cognitive, social, and semantic processes*, pp. 2–6, 2010.

[62] P. E. Compeau, P. A. Pevzner, and G. Tesler, "How to apply de bruijn graphs to genome assembly," *Nature biotechnology*, vol. 29, no. 11, p. 987, 2011.

[63] M. Gallé and M. Tealdi, "Reconstructing textual documents from n-grams," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, (New York, NY, USA), pp. 329–338, ACM, 2015.

[64] H. V. Jagadish, S. K. Rao, and T. Kailath, "Array architectures for iterative algorithms," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1304–1321, 1987.

[65] U. Eckhardt and R. Merker, "Hierarchical algorithm partitioning at system level for an improved utilization of memory structures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 1, pp. 14–24, 1999.

[66] S. V. Rajopadhye, "Synthesizing systolic arrays with control signals from recurrence equations," *Distributed Computing*, vol. 3, no. 2, pp. 88–105, 1989.

[67] S. Borkar, R. Cohn, G. Cox, S. Gleason, and T. Gross, "Warp: An integrated solution of high-speed parallel computing," in *Proceedings of the 1988 ACM/IEEE Conference on Supercomputing*, Supercomputing '88, (Los Alamitos, CA, USA), pp. 330–339, IEEE Computer Society Press, 1988.

[68] K. K. Parhi, C.-Y. Wang, and A. P. Brown, "Synthesis of control circuits in folded pipelined dsp architectures," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 1, pp. 29–43, 1992.

[69] P. Cappello, "A processor-time-minimal systolic array for cubical mesh algorithms," *IEEE transactions on parallel and distributed systems*, vol. 3, no. 1, pp. 4–13, 1992.

[70] T. Komarek and P. Pirsch, "Array architectures for block matching algorithms," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 10, pp. 1301–1308, 1989.

[71] H. Jagadish and T. Kailath, "A family of new efficient arrays for matrix multiplication," *IEEE Transactions on computers*, vol. 38, no. 1, pp. 149–155, 1989.

[72] 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 *Rebooting Computing (ICRC), IEEE International Conference on*, pp. 1–8, IEEE, 2016.

[73] F. Montagna, A. Rahimi, S. Benatti, D. Rossi, and L. Benini, "Pulp-hd: Accelerating brain-inspired high-dimensional computing on a parallel ultra-low power platform," in *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, (New York, NY, USA), pp. 111:1–111:6, ACM, 2018.

[74] D. Rossi, I. Loi, A. Pullini, C. Müller, A. Burg, F. Conti, L. Benini, and P. Flatresse, "A self-aware architecture for pvt compensation and power nap in near threshold processors," *IEEE Design Test*, vol. 34, pp. 46–53, Dec 2017.