Optimizing Self-Organizing Maps for Bacterial Genome Identification on Parallel Ultra-Low-Power Platforms

Optimizing Self-Organizing Maps for Bacterial Genome Identification on Parallel Ultra-Low-Power Platforms

Seyed Ahmad Mirsalari*, Saba Yousefzadeh†, Giuseppe Tagliavini*, Dimitrios Stathis†, Ahmed Hemani†
*University of Bologna, Bologna, Italy, †KTH Royal Institute of Technology, Stockholm, Sweden

***Abstract*—Pathogenic bacteria significantly threaten human health, highlighting the need for precise and efficient methods for swiftly identifying bacterial species. This paper addresses the challenges associated with performing genomics computations for pathogen identification on embedded systems with limited computational power. We propose an optimized implementation of Self-Organizing Maps (SOMs) targeting a parallel ultra-low-power platform based on the RISC-V instruction set architecture. We propose two mapping methods for implementing the SOM algorithm on a parallel cluster, coupled with software techniques to improve the throughput. Orthogonally to parallelization, we investigate the impact of smaller-than-32-bit floating-point formats (smallFloats) on energy savings, precision, and performance. Our experimental results show that all smallFloat formats exhibit a 100% classification accuracy. The parallel variants achieve a speed-up of 1.98×, 3.79×, and 6.83× on 2, 4, and 8 cores, respectively. Comparing our design with a 16-bit fixed-point implementation on a coarse grain reconfigurable architecture (CGRA), the FP8 implementation achieves, on average, 1.42× energy efficiency, 1.51× speedup, and a 50% reduction in memory footprint compared to CGRA. Furthermore, FP8 vectorization increases the average speed-up by 2.5×.**

***Index Terms*—Self-organizing maps, parallel ultra-low-power platform, RISC-V, approximate computing, smallFloat data types, genome identification**

I. Introduction

Pathogenic bacteria pose a significant threat to human health. Consequently, there is a growing demand for precise and efficient methods for swiftly identifying bacterial species. Developing a portable, low-cost, and fast genomics edge computational platform enables quick identification and control of fast-spreading epidemics in remote areas, allows for widespread genome testing and personalized treatment, facilitates frequent monitoring of genomic changes, and ensures privacy by eliminating the need for data transfer to cloud-based storage [1]. However, severe challenges are associated with performing genomics computations on embedded systems. Architectures for edge Artificial Intelligence (AI) computing are a promising target, but they are typically limited in terms of computational power and memory capacity. Traditional genomics algorithms are designed for high-performance computers and may exhibit slow and unscalable performance on edge devices [1]. Pathogen identification requires processing a large amount of data, depending on the identification method and complexity of the pathogen.

Artificial Neural Network (ANN) algorithms hold the potential to address the challenges above, offering the prospect of improved speed, accuracy, and reduced bias compared to traditional approaches. Self-Organizing Maps (SOMs) are a class of unsupervised learning models that reduce data dimensions and facilitate the clustering of similar data points [2]. In our context, SOMs encapsulate a compressed representation of genomic data, able to distinguish between different pathogens without processing the entire sampled DNA data. For instance, the algorithm proposed in [3] uses 40k random fragments of the DNA sequence of two strains of E. Coli bacteria to train two SOM networks to classify subsequent sequences of the bacterial strains. One of the core benefits of this approach is its ability to work with fragments of the DNA sequence instead of requiring fully assembled DNA, as has been attempted by [4] in the past.

Approximate computing techniques can significantly enhance energy efficiency for applications that can tolerate some reduction in output quality [5], trading off a slight degradation in the accuracy of computations for notable gains in overall system performance [6]. In power-constrained platforms, leveraging fixed-point implementations is a common method to enhance energy efficiency and speed for applications performing computations on real numbers. This approach involves manually adjusting the range and precision of operands based on the processing chain's requirements. However, implementing fixed-point optimizations can be complex and demanding since it requires a comprehensive understanding of the target algorithms [7]. In scenarios where computational accuracy and a wide dynamic range are critical requirements, adopting Floating-Point (FP) arithmetic is highly beneficial. Tagliavini et al. [8] conducted a study showing that utilizing smaller-than-32-bit FP formats (smallFloats) can result in considerable energy savings. This outcome is due to the simplification of arithmetic circuitry and reduction in memory bandwidth requirements for data transfer between memory and registers through the enablement of vectorization techniques.

This paper presents an optimized implementation of SOMs targeting a parallel ultra-low-power platform (PULP) based on the RISC-V instruction set architecture (ISA). The main contributions of our work are the following:

- Proposing two mapping methods for implementing the SOM algorithm on a parallel computing cluster (Sec. IV-A).
- Designing an algorithm to support various network sizes by employing tiling and double buffering to transfer

data between different levels of the memory hierarchy (Sec. IV-A).
- Introducing algorithmic variants to use smallFloat data types for the SOM algorithm (Sec. IV-B).
- Enabling vectorization techniques to improve the performance of smallFloat data types (Sec. IV-C).
- Assessing the impact of smallFloat data types, specifically the FP8 data type, on the accuracy, memory footprint, and performance metrics (Sec. V).
- Comparing the results with a CGRA fabric that uses a 16-bit fixed-point format (Sec. V).
- Providing an open-source repository[1] that includes the implementation of the SOM algorithm and the integration of the FP8 data type in the PyTorch.

In our experimental findings, we have observed that our approach utilizing FP8 yields several benefits: Firstly, we achieve accurate classification of DNA sequences, ensuring reliable and precise identification. Secondly, we successfully reduced the memory footprint by a factor of 2 compared to 16-bit representations. Furthermore, our FP8 implementation demonstrates an impressive speedup w.r.t. the FP32 implementation, up to $18.7\times$ in large SOM networks. In contrast, the CGRA-based implementation achieves a speedup of $11.05\times$. Additionally, the FP8 implementation exhibits a lower energy consumption than the 16-bit CGRA implementation [3], thanks to dedicated algorithmic optimizations. We achieve a remarkable $6.72\times$ improvement in energy efficiency compared to FP32, while the CGRA achieves a $3.15\times$ improvement in large SOM networks.

The rest of the paper is structured as follows. In section II, we review the related work in the field. Section III provides a brief background on the SOM algorithm, the PULP architecture, and smallFloats. In section IV, we explain our implementation approaches. Section V presents the experimental results, and finally, the paper is concluded in section VI.

II. Related Work

Leveraging hardware acceleration is an alternative approach to enhance the efficiency of computationally intensive algorithms. Previous research conducted by [3] has proposed the utilization of SOM for accelerating genome identification processes. The CGRA implementation [9] observed a less than 1% quality loss when training SOM networks using 16-bit fixed-point number representations, compared to 32-bit FP implementation. The authors of [9] also demonstrated that low-bit fixed-point formats, such as 8-bit, are inadequate for genome identification experiments, as they fail to meet the required accuracy and dynamic range for successful identification. A current trend in computing is to customize the precision of floating-point arithmetic in applications to match their specific requirements and constraints within their respective domains [10]. With the increasing computational requirements and dynamic nature of target algorithms, there is a growing need for hardware support for arithmetic operations in smaller-than-32-bit FP formats to enhance system energy efficiency while maintaining the desired level of accuracy [7].

In the field of low-power near-sensor computing, the GAP8 [11] system-on-chip (SoC) from the parallel ultra-low power (PULP) family has been widely utilized for its powerful and energy-efficient processing capabilities. Recently, a new SoC called GAP9[2] has been introduced by GreenWaves technologies. This GAP9 offers significant advancements compared to its predecessor. Notably, GAP9 demonstrates a remarkable increase in power efficiency, with a consumption of only 0.33 milliwatts per giga operation (GOP) for each RISC-V core. Additionally, it boasts improved memory capacity and supports FP operations on 32 and 16-bit data types, making it a highly desirable option for energy-efficient computing tasks. This work leverages the features of the GAP9 SoC proposing a portable and low-power solution for genome identification.

III. Background

This section overviews the implemented SOM algorithm, the PULP platform, and the smallFloats data types.

A. SOM algorithm

Algorithm 1: The computational kernel of the SOM training; N: Number of neurons, I: Input-Vector, W: Weight Matrix

for $I_k \in IS$ **do**
  $dist_{min} = \min_{j=1...N}(\sum_{i=1}^{M} |I_{k,i} - W_{i,j}|)$;
  $j_{min} = j$ where $dist_j = dist_{min}$;
  for $j \in 1...N$ **do**
    $dist = ||j - j_{min}| - \frac{N}{2}|$; // toroid distance
    $W_j = W_j - \frac{\beta}{2^{dist}}(W_j - I_k)$;
  end
  $\beta = \max(\beta * decay\_factor, \beta_{min})$; //decay β
end

This paper uses a circular SOM for bacterial genome recognition described in [3]. Algorithm 1 summarizes the main computational steps. The goal is to capture the genetic signature of a particular bacteria as a specific SOM; this implies that we train one specific SOM for every bacterial strain we are interested in. By comparing the DNA sequences of unknown bacteria with trained SOM networks, we aim to identify the closest correlation between the SOM network trained on the same bacteria and the unknown test bacteria. This correlation allows us to determine the identity of the unknown bacteria.

A SOM is represented by a weight matrix, denoted as W, with dimensions N × M, where N represents the number of neurons, and M represents the number of weights for each neuron. Each bacterial strain has its dedicated SOM, trained using strain-specific sequences (IS) composed of M-element vectors, with each element representing a nucleotide (C, G, T,

[1] https://github.com/ahmad-mirsalari/SOM-on-PULP

[2] *https://greenwaves-technologies.com/*

or A). These vectors correspond to the short reads obtained from sequencing machines. During training, a random fragment is selected from the sequence fragments corresponding to a bacterial genome. The distance of this fragment from all neurons is computed using the equation stated in line 7. Subsequently, the weights of the winning neuron and its neighbouring neurons are updated based on lines 9-12.

B. PULP

Edge AI nodes require high throughput and energy efficiency within a power limit of a few milliwatts. Parallel Ultra-Low-Power (PULP) [12] is an open-source system-on-chip (SoC) performing parallel computing in a near-threshold operating point. PULP includes a microcontroller unit (MCU) and a programmable multi-core cluster. The SoC also includes an on-chip SRAM memory (L2) to store resident code and application data. The multi-core cluster consists of a variable number of identical RISC-V cores sharing a Tightly Coupled Data Memory (TCDM) organized into multiple banks. This design allows the cores to access data in a single clock cycle through a low-latency logarithmic interconnect. The software manages data transfers between the L2 and the TCDM using a dedicated Direct Memory Access (DMA) controller. This controller orchestrates data transfers in parallel with core computations, thereby reducing the impact of memory access latency and improving overall system performance. In addition, a third memory level is available in the system, namely an off-chip L3 memory with virtually unlimited capacity.

The cluster cores share four floating-point units (FPUs) that support FP32, FP16, bfloat16, and FP8 arithmetic, including cast operations between different formats and cast-and-pack instructions to improve the effectiveness of packed Single-Instruction Multiple-Data (SIMD) vector operations [13]. These FPUs are connected through a low-latency interconnect, which handles access contention in hardware, allowing multiple cores to share a single FPU seamlessly from a software perspective.

C. SmallFloats

Adopting smaller-than-32-bit FP formats has become essential in modern computing systems that require both accuracy and a wide dynamic range. The authors in [14] reveal that FP8 training yields similar outcomes to FP16 or bfloat16 training in a wide range of tasks, neural network model architectures, and sizes without requiring modifications to the model or optimizer hyperparameters. Moreover, due to its nonlinear sampling of real numbers, FP8 may offer advantages over int8 for inference tasks.

Recently, there has been a growing interest in utilizing FP8 formats for training neural networks within the deep learning community [15]. Notably, Nvidia has introduced an FP8 format in their Transformer Engine[3] software designed for the latest Hopper architecture GPUs [16]. Additionally, an IEEE working group is actively exploring the possibility of standardizing FP8 for training deep learning networks in cloud-based environments [17]

In [18], a set of Instruction Set Architecture (ISA) extensions are presented for RISC-V 32-bit processors, which support scalar and SIMD operations designed explicitly for smallFloat formats on embedded processors. This study demonstrates that, on average, manual vectorization achieved a speedup of $1.5\times$ compared to scalar operations, with a peak speedup of $1.91\times$. Furthermore, when utilizing FP8 types, manual vectorization led to average and peak speedups of $2.35\times$ and $3.58\times$, respectively. It is important to note that employing smallFloats without vectorization results in the same number of cycles required as using FP32.

The advantages of SmallFloats in edge devices stem from two key aspects. Firstly, they reduce the memory footprint for storing data and parameters. Secondly, they accelerate execution by leveraging SIMD vectorization. These benefits also positively impact energy consumption in algorithms. We widely discuss the adoption of SmallFloat types and related vectorization techniques in Sec. V.

D. Power and Energy Characterization of PULP

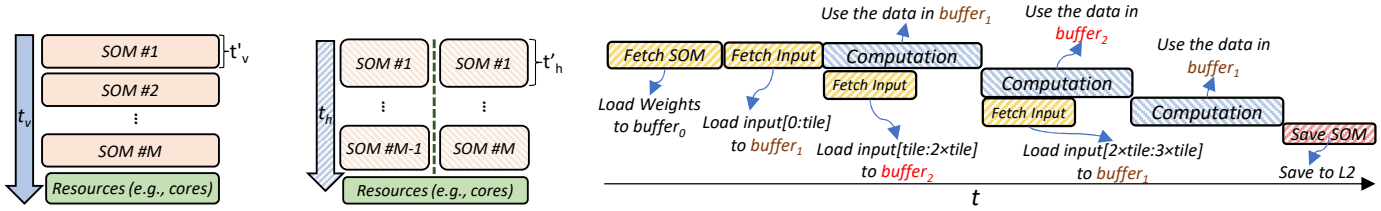
To explain the experimental results, we analyzed the effects of architectural design choices on the energy and power consumption of the GAP9 platform, considering cores, memory, and interconnects. Table I presents the outcomes of this analysis executing a compute-intensive kernel (i.e., a matrix multiplication) on different configurations.

Sharing an FPU between two cores results in a $1.03\times$ and $1.02\times$ increase in cycles and energy consumption compared to the non-shared scenario. These values are roughly equivalent when comparing the power consumption of the code using FP instructions to the one using integer operations on a single core. However, when utilizing 8 cores, FP32 exhibits increased cycles, energy consumption, and average power by $1.09\times$, $1.25\times$, and $1.22\times$, respectively. This result is due to the higher circuit complexity of the FPU.

Moreover, we evaluated the impact of vectorization and parallelization. FP16 vectorization reduces the cycles and energy consumption by $0.56\times$ and $0.46\times$, respectively. Vectorization achieves this by minimizing the load and store operations, thereby reducing the energy expended during a load-execute-store process for each set of input operands. However, adopting vectorization in real applications incurs additional overheads due to conversion from other formats or part of the algorithms that are inherently scalar, preventing an ideal reduction of cycles (by $0.5\times$). To examine the impact of parallelization without interference from the FPU sharing, we compared the use of 8 cores and 1 core using INT32 computations. The results reveal that employing 8 cores decreases $0.13\times$ and $0.19\times$ cycles and energy consumption, respectively, compared to using only 1 core.

Finally, to evaluate power-saving policies when the cores are idle, we executed the benchmark on 8 cores assigning the entire computation to a single core while the others are clock-

[3] *https://github.com/NVIDIA/TransformerEngine*



(a) The vertical pattern (b) The horizontal pattern (c) Example of tiling and double buffering

Fig. 1: Mapping patterns

TABLE I: Average energy, cycles, and power consumption of FP32, INT32, and vectorized FP16 on the PULP SoC.

	Cores	Baseline	Cycles	Energy	Average Power
Shared FPU	2	Non Shared FPU	1.03x	1.02x	0.99x
FP32	1	INT32	1.00x	1.00x	1.06x
FP32	8	INT32	1.09x	1.25x	1.22x
Vectorized FP16	1	FP32	0.56x	0.46x	1x
INT32	8	INT32, 1 core	0.13x	0.19x	1.44x
FP32	8 (7 idle)	FP32, 1 core	0.99x	0.99x	1x

gated. The results in the two cases are close, demonstrating the effectiveness of the power-saving policies.

IV. Methodology

This section presents the details of our methodology.

A. Computation Mapping

Figure 1 depicts two mapping strategies, namely Vertical and Horizontal, used to map SOM to GAP9 SOC. In the Vertical pattern, the execution of SOM networks occurs sequentially, where all available resources (i.e., cores and memory) are dedicated to a single SOM under training. This approach yields satisfactory performance when the total workload, determined by factors such as the input size (line 6), Number of Neurons (line 7), and the neighbourhood radius (line 9), can be efficiently parallelized across all cores. However, if the workload is insufficient, this method does not fully exploit the potential of utilizing multiple cores. Consequently, in the Horizontal pattern, the available resources are divided and shared between the two SOMs being trained. We verified that using more than two SOMs with 8 available cores increases the overheads without real benefits on performance. This approach allows for a more balanced utilization of resources and can potentially improve the overall efficiency of the training process. We will show the results of our experiments in Section V.

To facilitate the execution of large networks that do not fit within the TCDM, we support tiling and double buffering for both SOM parameters and input data. The tiling strategy divides the data into smaller fragments, enabling them to fit within the available amount of TCDM memory. Additionally, this strategy enables seamless data movement between memory levels by utilizing DMA-assisted double buffering. By employing this technique, the data for the next tile can be transferred in parallel with the computation on the current tile, leading to a significant reduction in the overhead associated with the memory hierarchy. Ideally, this technique allows for the complete overlap of data transfers and computation phases, as shown in Figure 1c. To simplify the explanation, considering TCDM capacity is enough to hold the SOM networks, we demonstrate the utilization of double buffering to efficiently transfer input data to the TCDM memory.

B. FP8

The FP8 format is gaining traction among hardware developers, considering its availability in the new NVIDIA Hopper architecture. However, there is no standardized definition for FP8 in the IEEE 754 standard, and different definitions are being reconciled within the industry [17]. The specific values for exponent bias and the encodings for infinity and NaN are not yet standardized. Two representations for FP8 are commonly used: E4M3 (4-bit exponent and 3-bit mantissa) and E5M2 (5-bit exponent and 2-bit mantissa) [14]. While the Nvidia FP8 introduction paper [14] mentions that the FP8-E5 format is primarily utilized for gradients in deep neural networks, we opted for the E5M2 format due to its availability on the PULP hardware platform. The E5M2 format adheres to the IEEE 754 conventions for the exponent and special values, making it similar to IEEE half-precision with a reduced number of mantissa bits. As a result, conversion between E5M2 and IEEE FP16 formats is straightforward and can be easily achieved [17].

We extended the functionality of the PyTorch backend by implementing support for FP8|e5m2 data type. The PULP architecture currently supports this format at the hardware design and compiler level. This modification enabled us to leverage the benefits of reduced memory footprint and accelerated computation offered by 8-bit FP precision. This implementation is available as open-source [4].

C. Vectorization

Despite the inherent limitations in vectorizing the SOM algorithm, we support SIMD vectorization for both the proposed computational patterns. In lines 7 and 8, SIMD vectorization is employed for calculating the distance between the input and weight values. In line 10, the toroid distance is computed

[4]https://github.com/ahmad-mirsalari/SOM-on-PULP

for each neuron relative to the winning neuron. This distance serves as the basis for subsequent operations, including the division operation $\frac{\beta}{2^{dist}}$. In the following step, we utilize a built-in function to generate a vector containing the division results. For each smallFloat format, the compiler provides a specific built-in function to create the corresponding vector with a minimum set of assembly instructions (1 for FP16/bfloat16, 2 for FP8). As a result, vectorization can be leveraged for all operations in line 11.

D. Algorithm and Architectural Optimizations

Given that the denominator in the division operation is a power of two, we optimized the computation by converting it into a multiplication operation; $\frac{\beta}{2^{dist}}$ is equal to $\beta \times 2^{-dist}$. We implemented an optimized version of the function computing the power of two with a bit-level manipulation of the exponent. To determine the exponent value for 2^{-dist} in the target format, we subtract the input value (x) from the bias. Then, we move this value to the exponent field position through a left-shift operation. These steps require only two cycles on the PULP architecture; since there is a single register file for integer and FP data, bit manipulation does not incur register copy overheads. Using a similar approach, the $fabs$ function has been optimized to efficiently compute the absolute value of a floating point variable x through a $bitwise$ AND operation with a $bitmask$ designed to clear the $sign$ bit. Both optimizations can be generalized to the vectorial case.

We skip the inner loop as an additional optimisation when the distance exceeds a predefined threshold value. This optimization is possible because the value of 2^{dist} becomes unrepresentable in certain cases. The range limitations of the selected FP data type determine this threshold value. In practical terms, this optimization guarantees that only the winning neuron and its neighbouring neurons are updated, preceding and following it in a circular SOM within a radius of 2 times the threshold.

V. Experimental Results

A. Experimental Setup

1) GAP9: GAP9 is the commercial embodiment of the Vega SoC [19]. It comprises 10 cores implementing the RISC-V RV32IMFC ISA, with the addition of an ISA extension for DSP processing. The compute cluster consists of 9 cores, with one core dedicated to orchestration and 8 serving as workers and sharing 4 FPUs. GAP9 is equipped with 128 kB of TCDM. The fabric controller (FC) has access to various peripherals and is equipped with 1.5 MB of L2 memory. The architecture incorporates adjustable dynamic frequency and voltage domains, enabling precise control over energy consumption based on specific requirements at any given moment. Under peak performance conditions, both the cluster and the fabric controller cores operate at a frequency of 400 MHz.

GAP9 does not natively support the FP8 data type: this feature will be evaluated for the next generation of the SoC. To assess the impact of FP8 operations, we get the performance counters from cycle-accurate simulations and the power numbers from a post-synthesis simulation.

2) Golden Model: We developed a Python reference based on PyTorch to establish a reliable golden model. This model supports various floating-point types, including FP32, FP16, bfloat16, and FP8. We introduced a set of dedicated flags into the golden model to simulate the behaviour of the instructions commonly found in ISA extensions for edge AI nodes. The *MAC* flag emulates the fused-multiply-add (FMA) operator that is typically available in DSP instruction sets for embedded devices. The *vector* flag simulates using SIMD vector instructions. Introducing these flags aims to account for the precision changes that occur in intermediate results due to rounding effects caused by using the respective instructions (MACs and vector operations) instead of the original ones (mul+add and scalar operations). As a result, the golden model considers the impact of these processor-level operations to ensure accurate and reliable results.

We implemented the SOM algorithm in plain C code, including a set of preprocessor directives to enable vectorization, multi-core processing, and performance monitoring. The full project codebase is available in an open-source repository[5].

3) CGRA fabric: We compare the SOM implementation on GAP9 with a CGRA that targets dense linear algebra applications, including SOM [3]. The authors employed two CGRA fabrics, namely Dynamically Reconfigurable Resource Array (DRRA) [20] for dense linear algebra and Distributed Memory Architecture (DiMArch) [21] for streaming scratchpad memory, connected through a configuration network-on-chip (NoC), as described in [3]. Each DRRA cell includes a 16-bit fixed-point arithmetic Data Processing Unit (DPU), a register file, and a sequencer responsible for cell configuration. The arrangement consists of multiple columns, with each column comprising two DRRA cells and two DiMarch cells. The study observed that increasing the number of columns led to a corresponding increase in speedup. The details of SOM mapping, timing, and cell configuration can be found in [3]. As part of their extended work, the authors investigated the impact of different fixed-point bit-widths on the performance of the SOM algorithm [9].

The SOM algorithm was implemented with four different dimensions: 128, 256, 512, and 1024 neurons, and each neuron has 8 weights. This setup allowed for a fair comparison between the PULP implementation and the CGRA. The CGRA fabrics lack architectural features like vectorization, double buffering, and near-threshold computation. Further, the CGRA-based SOM implementation lacks the algorithmic optimization described in Sec. IV-D.

B. Accuracy

We trained 10 different SOMs, each with a distinct DNA sequence corresponding to a different bacteria. For each SOM, we used a set of 20,000 training vectors. After the training

[5]https://github.com/ahmad-mirsalari/SOM-on-PULP

phase, we utilized the trained networks to identify 1000 unknown DNA sequences. To evaluate the performance of the networks, we use the classification error metric [9]. This metric is calculated by dividing the number of times the network misclassified the bacterial strain (C_{false}) by the total number of tests performed (C_{total}):

$$classification\_error = \frac{C_{false}}{C_{total}} \times 100\%$$

Our experimental results demonstrate that the FP16 and bfloat16 formats successfully passed the experiment across different neuron counts, maintaining the same accuracy as the FP32 baseline. However, when utilizing the FP8 format, we observed a classification error of less than 9% when employing 128 neurons. Notably, this error goes to zero by increasing the number of neurons to 256 or higher.

Considering fixed-point data formats, the 8-bit one proved ineffective for all different networks, yielding unsatisfactory results in the classification task. The 12-bit format exhibits a classification error of 39%. Finally, the 16-bit and 32-bit formats successfully pass the experiment with 0% error.

C. Hardware Results

In the hardware experiments, we trained two SOM networks using four dimensions: 128, 256, 512, and 1024 neurons. These networks were trained with a dataset of 40,000 training vectors representing two strains of E. coli bacteria.

1) Memory footprint and silicon area: Table II reports the memory footprint required for storing the weights of two SOM networks and the corresponding silicon area requirements. Employing FP8 can substantially reduce memory requirements, up to $2\times$ compared to 16-bit formats, for networks with 256, 512, and 1024 neurons while maintaining accuracy. Additionally, opting for FP8 with 128 neurons allows for further reduction, although at the cost of a 9% decrease in SOM accuracy.

To ensure a fair assessment, comparing the cluster domain of GAP9 (including the TCDM) with the CGRA fabric (excluding the DRAM) is more appropriate. This means we do not consider the L2 memory in GAP9 and the DRAM in the CGRA fabric, as they are responsible for storing the input data when measuring the area. In our study, we provided the area measurements for the CGRA fabric with the 8 columns. We can consider the CGRA fabric with a 16-bit fixed-point format and 8 columns to guarantee a fair comparison in terms of accuracy and maximum speedup. Since CGRA fabric is implemented using 28 nm, we scaled the area number by 0.6. As a result, GAP9 achieves a $1.13\times$ reduction in the area compared to the CGRA fabric. This increase in area in the CGRA fabric is because each cell has its own sequencer with dedicated memory.

2) Parallelization and vectorization performance: Fig. 2 illustrates the speed-ups achieved executing on 1, 2, 4, and 8 cores, combining the benefits deriving from parallelism and vectorization w.r.t the FP32 baseline. The suffixes *Vec*, *V*, and *H* indicate the execution using vectorization, Vertical mapping, and Horizontal mapping variants, respectively. In the same chart, we provide the speed-up values for the CGRA fabric with 1, 2, 4, and 8 columns. The baseline cycle counts for the FP32 data type are as follows: 425,180,568 cycles for 128 neurons, 738,577,694 cycles for 256 neurons, 1,364,078,880 cycles for 512 neurons, and 2,613,552,914 cycles for 1024 neurons. On average, we achieved $1.99\times$, $3.879\times$, and $7.01\times$ speed-ups on 2, 4, and 8 cores with no vectorization using Horizontal mapping.

TABLE II: Memory required for storing the weights of two SOM neurons in byte and Area in mm^2

Weight Memory (Byte)					Area (mm^2)	
Neuron / Bit	**128**	**256**	**512**	**1024**	CGRA 8 columns 28nm	GAP9
8-bit	2048	4096	8192	16384		
16-bit	4096	8192	16384	32768	2.79	1.48
32-bit	8192	16384	32768	65536		

Notably, the Horizontal mapping approach cannot be utilized with only one core, as we divide the number of cores between two SOMs in this configuration. As depicted in Figure 2, the Horizontal method consistently achieves a superior speedup compared to the Vertical method, especially when scaling up to 8 cores. As outlined in Section IV-D, the determination of a threshold value plays a crucial role in ensuring an accurate representation of 2^{dist}, considering the data range limitations of the chosen FP data type. This threshold value is calculated using the expression $\log_2(max\_number)$, resulting in a maximum threshold value of 19 for FP32 or bfloat16. Notably, despite the different data ranges of FP16/FP8 and FP32/bfloat16, the same threshold value can be employed for these data types without compromising accuracy. Consequently, the maximum number of neurons to be updated is 39, encompassing the winning neuron and its neighbouring sides in the toroid topology (19 + 1 + 19). In the case of Vertical mapping, these neurons are distributed among 8 cores, leading to an inadequate workload w.r.t. the overhead associated with parallelization. Conversely, with Horizontal mapping, two SOMs are updated using this number of cores, resulting in a higher speedup.

As depicted in Figure 1, the Horizontal mode exhibits a longer runtime for each individual SOM (utilizing half of the available resources) compared to the Vertical mode ($\acute{t}_h > \acute{t}_v$). However, since we simultaneously execute two SOMs in the Horizontal mode, the overall time required is actually shorter than that of the Vertical mode ($t_h < t_v$). On average, vectorization yields a speed-up improvement of $1.45\times$ for FP16/bfloat16 and $2.49\times$ for FP8 compared to FP32. As discussed in Section IV-C, the SOM algorithm poses challenges in exploiting vectorization.

When comparing the performance of FP16/bfloat16 on GAP9 with the 16-bit fixed-point on CGRA fabric, we observed that in smaller networks, specifically with 128 and 256 neurons, CGRA consistently demonstrated a superior

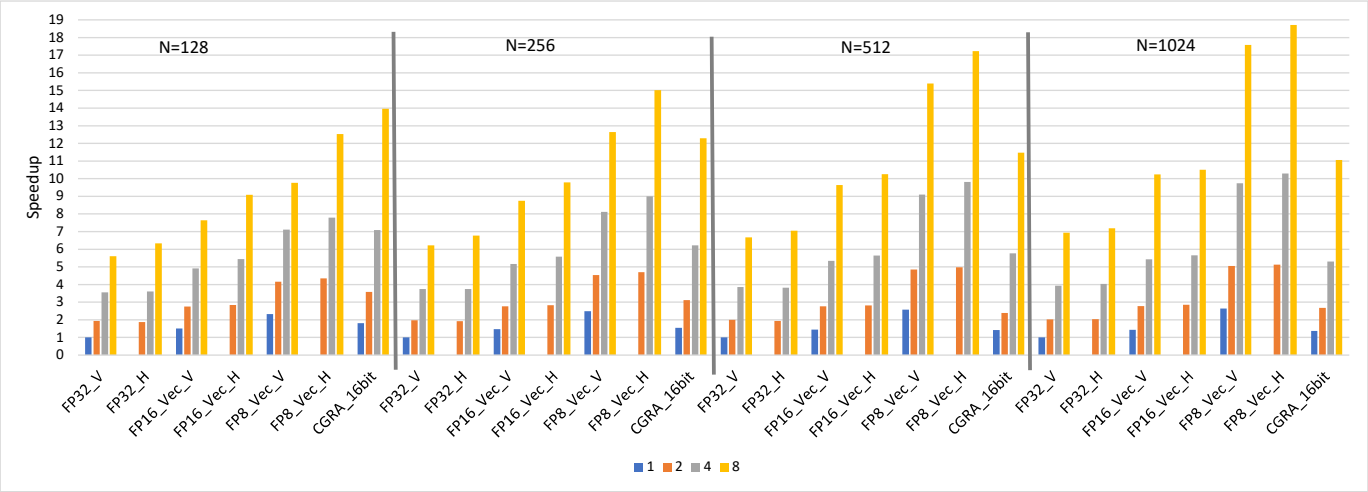


Fig. 2: Speed-up of parallel (2, 4, 8 cores), vectorized (FP16/bfloat16, FP8), and CGRA fabric (1, 2, 4, and 8 columns) w.r.t. the FP32 baseline. V and H represent vertical and horizontal mapping, respectively.

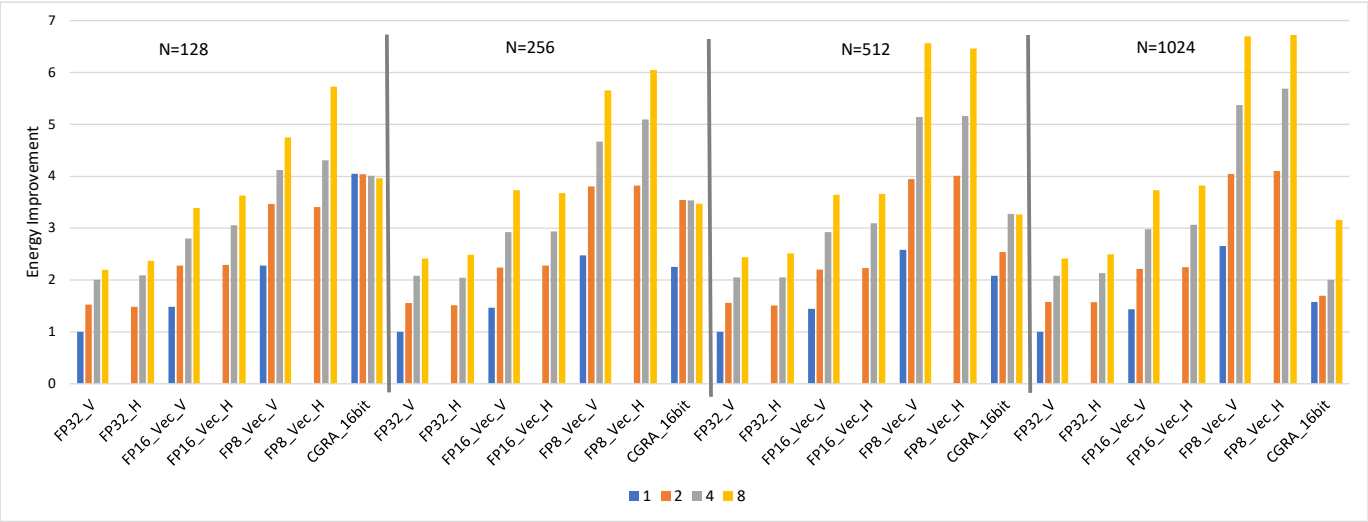


Fig. 3: Energy improvement of parallel (2, 4, 8 cores), vectorized (FP16/bfloat16, FP8), and CGRA fabric (1, 2, 4, and 8 columns) w.r.t. the FP32 baseline. V and H represent vertical and horizontal mapping, respectively.

speedup across various column configurations. For example, when using 8 columns, CGRA achieved a speedup of $12.3\times$, while GAP9 achieved a speedup of $9.78\times$ with 8 cores.

However, as the network size increased, the competitive advantage of GAP9 became evident. For instance, using 1024 neurons, CGRA fabric achieved a speedup of $10.89\times$ with 8 columns, while GAP9 achieved a close speedup of $10.50\times$ with 8 cores. These differences in GAP9 and CGRA stem from the fact that for smaller SOMs, the architectural and algorithmic optimizations do not show up; for larger networks, these advantages play a significant role.

In the case of the FP8 data type, our analysis in Figure 2 revealed that CGRA outperformed FP8 in terms of speedup only for 128 neurons and 8 columns, where it achieved a speedup of $13.97\times$ compared to $12\times$ for FP8. However, FP8 consistently exhibited a higher speedup for other network sizes in all core configurations. For example, when using 256, 512, and 1024 neurons, CGRA achieved $12.3\times$, $11.47\times$, and $10.89\times$ with 8 columns, while GAP9 achieved speedups of $14.77\times$, $17.19\times$, and $18.60\times$ with 8 cores.

3) Energy: The energy estimation for the CGRA was based on post-layout synthesis simulations in 28 nm technology with 180 MHz frequency. The values are then scaled down to 22 nm for fair comparison using a factor of 0.7 based on [22]. The energy includes the DRAM accesses based on the [23].

The baseline energy consumption values for the FP32 data type are as follows: 32.17 mJ for 128 neurons, 56.33 mJ for 256 neurons, 104.05 mJ for 512 neurons, and 200.75 mJ for 1024 neurons.

As depicted in Figure 3, in the case of a small network, we find that FP16 on GAP9 exhibits a comparable energy improvement when utilizing 8 cores, compared to the 16-bit

fixed-point on CGRA fabric. Specifically, FP16 achieves a notable energy improvement of $3.67\times$ using 8 cores, which is very close to the energy improvement achieved by the 16-bit fixed-point at $3.47\times$ in a network with 256 neurons.

However, as the network size increases, FP16 showcases its advantage by achieving a higher energy improvement with fewer cores. For instance, in the case of 1024 neurons, FP16 achieves a significant energy improvement of $3\times$ with only 4 cores on the GAP9 platform, surpassing the performance of the 16-bit fixed-point which achieves a $2\times$ improvement on the CGRA fabric.

In the case of FP8, we observe consistently higher energy improvements across all configurations compared to the use of the 16-bit fixed-point implementation on the CGRA fabric. This holds true for networks with 256, 512, and 1024 neurons, and FP8 even surpasses the 16-bit fixed-point approach in the case of 128 neurons when employing only 4 cores.

For instance, with a single core, FP8 achieves energy improvements of $2.47\times$, $2.58\times$, and $2.65\times$ in networks with 256, 512, and 1024 neurons, respectively. In contrast, the CGRA fabric yields energy improvements of $2.25\times$, $2.08\times$, and $1.57\times$ when using 1 column in the aforementioned neurons. Considering 8 cores, FP8 demonstrates substantial energy improvements of $6\times$, $6.56\times$, and $6.72\times$ in networks with 256, 512, and 1024 neurons, respectively. Conversely, the CGRA fabric provides energy improvements of $3.47\times$, $3.26\times$, and $3.15\times$ when utilizing 8 columns in the same networks.

VI. Conclusion and Future Work

This paper proposes an optimized implementation of SOMs for genomics computations on resource-constrained embedded systems. We explore the use of smallFloat formats to improve energy efficiency and performance. Our approach includes parallelization, data transfer optimization, and comparison with a CGRA, achieving state-of-the-art performance for bacterial identification. Our work contributes to efficient genomics computations on embedded systems, considering computational limitations and energy constraints. As a significant outcome, we demonstrated that a general-purpose parallel platform can achieve state-of-the-art results for genome identification. Other architectures, such as a CGRA, have the potential to achieve better results considering the same setup. However, the PULP architecture allows algorithm designers to obtain competitive results in many application contexts by performing a software exploration and using commercially available solutions.

Utilizing a scaling factor is a widely adopted approach to bring higher precision values within a range that aligns well with the representable range [14]. As the FP8 format did not achieve a 0% classification error with 128 neurons, we will explore the dynamic scaling techniques to enhance the accuracy of FP8 as future work. Furthermore, we will investigate the implementation of alternative encodings of FP8, such as FP|E4M3, to assess their impact on accuracy.

Acknowledgement

This work is conducted within the project APROPOS, funded by the European Union's Horizon 2020 (H2020) Marie Sklodowska-Curie Innovative Training Networks H2020-MSCA-ITN-2020 call, under the Grant Agreement no 956090.

References

[1] V. Gnanasambandapillai *et al.*, "MESGA: An MPSoC based embedded system solution for short read genome alignment," in *2018 23rd Asia and South Pacific Design Automation Conf. (ASP-DAC)*, pp. 52–57.

[2] T. Kohonen, "The self-organizing map," *Proc. of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[3] Y. Yang *et al.*, "RiBoSOM: rapid bacterial genome identification using self-organizing map implemented on the synchoros SiLago platform," *Proc. of the 18th Int. Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, 2018.

[4] P. Liu *et al.*, "3D-stacked many-core architecture for biological sequence analysis problems," *International journal of parallel programming*, vol. 45, pp. 1420–1460, 2017.

[5] Q. Xu *et al.*, "Approximate Computing: A Survey," *IEEE Design & Test*, vol. 33, pp. 8–22, 2016.

[6] A. Yazdanbakhsh *et al.*, "AxBench: A Multiplatform Benchmark Suite for Approximate Computing," *IEEE Design & Test*, vol. 34, pp. 60–68, 2017.

[7] S. Mach *et al.*, "A Transprecision Floating-Point Architecture for Energy-Efficient Embedded Computing," in *2018 IEEE Int. Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–5.

[8] G. Tagliavini *et al.*, "A Transprecision Floating-Point Platform for Ultra-Low Power Computing," IEEE, pp. 1051–1056, 2018.

[9] D. Stathis *et al.*, "Approximate Computing Applied to Bacterial Genome Identification using Self-Organizing Maps," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2019, pp. 560–567.

[10] P. Micikevicius *et al.*, "Mixed Precision Training," 2018.

[11] GreenWaves Technologies Corp., *GAP8 Hardware Reference Manual*, Grenoble, FR, Jan. 2019, [Online]. [Online]. Available: https://gwt-website-files.s3.amazonaws.com/gap8_datasheet.pdf

[12] M. Gautschi *et al.*, "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices, year=2017," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713.

[13] S. Mach *et al.*, "FPnew: An Open-Source Multiformat Floating-Point Unit Architecture for Energy-Proportional Transprecision Computing," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 4, pp. 774–787, 2021.

[14] P. Micikevicius *et al.*, "FP8 Formats for Deep Learning," 2022.

[15] B. Noune *et al.*, "8-bit Numerical Formats for Deep Neural Networks," 2022.

[16] M. Andersch *et al.* (2022) Nvidia Hopper Architecture In-Depth. 2, 4, 6. [Online]. Available: https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

[17] M. van Baalen *et al.*, "FP8 versus INT8 for efficient deep learning inference," 2023.

[18] G. Tagliavini *et al.*, "Design and Evaluation of SmallFloat SIMD extensions to the RISC-V ISA," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 654–657.

[19] D. Rossi *et al.*, "Vega: A Ten-Core SoC for IoT Endnodes With DNN Acceleration and Cognitive Wake-Up From MRAM-Based State-Retentive Sleep Mode," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 1, pp. 127–139, 2022.

[20] M. A. Shami *et al.*, "Partially reconfigurable interconnection network for dynamically reprogrammable resource array," in *2009 IEEE 8th Int. Conference on ASIC*, 2009, pp. 122–125.

[21] M. A. Tajammul *et al.*, "NoC Based Distributed Partitionable Memory System for a Coarse Grain Reconfigurable Architecture," in *2011 24th Int. Conference on VLSI Design*, 2011, pp. 232–237.

[22] H. Esmaeilzadeh *et al.*, "Dark silicon and the end of multicore scaling," in *Proc. of the 38th Annual Int. Symposium on Computer architecture*, 2011, pp. 365–376.

[23] M. O'Connor *et al.*, "Fine-grained DRAM: Energy-efficient DRAM for extreme bandwidth systems," in *Proc. of the 50th Annual IEEE/ACM Int. Symposium on Microarchitecture*, 2017, pp. 41–54.