Optimizing Foundation Model Inference on a Many-Tiny-Core Open-Source RISC-V Platform

Optimizing Foundation Model Inference on a Many-tiny-core Open-source RISC-V Platform

Viviane Potocnik, Luca Colagrande, Tim Fischer, Luca Bertaccini, Daniele Jahier Pagliari, Alessio Burrello, Luca Benini

***Abstract*—Transformer-based foundation models have become crucial for various domains, most notably natural language processing (NLP) or computer vision (CV). These models are predominantly deployed on high-performance GPUs or hardwired accelerators with highly customized, proprietary instruction sets. Until now, limited attention has been given to RISC-V-based general-purpose platforms. In our work, we present the first inference results of transformer models on an open-source many-tiny-core RISC-V platform implementing distributed Softmax primitives and leveraging ISA extensions for SIMD floating-point operand streaming and instruction repetition, as well as specialized DMA engines to minimize costly main memory accesses and to tolerate their latency. We focus on two foundational transformer topologies, encoder-only and decoder-only models. For encoder-only models, we demonstrate a speedup of up to 12.8× between the most optimized implementation and the baseline version. We reach over 79% FPU utilization and 294 GFLOPS/W, outperforming State-of-the-Art (SoA) accelerators by more than 2× utilizing the HW platform while achieving comparable throughput per computational unit. For decoder-only topologies, we achieve 16.1× speedup in the Non-Autoregressive (NAR) mode and up to 35.6× speedup in the Autoregressive (AR) mode compared to the baseline implementation. Compared to the best SoA dedicated accelerator, we achieve 2.04× higher FPU utilization.**

***Index Terms*—Foundation Models, Transformers, RISC-V, Multi-Core Platforms**

I. Introduction

In the last few years, the field of artificial intelligence (AI) has seen a paradigm shift towards transformer-based models such as Large Language Models (LLMs) [1], Vision Transformers (ViTs) [2], and, more in general, Foundation Models (FMs) [3]. Transformer-based FMs are large-scale models based on the so-called attention mechanism [4], trained on enormous generic datasets in a self-supervised fashion. FMs can then be fine-tuned to perform specialized downstream tasks [5]. The big breakthrough in FMs came with the Generative Pretrained Transformer (GPT), which exploited the transformer architecture for question-answering problems with natural language. After the immense success of GPT-3 [6], transformer-based FMs have then been adapted to all aforementioned domains, with models such as ViTs [2] for computer vision or Whispers [7] for Automatic Speech Recognition (ASR).

The broad adoption of FMs makes their efficient execution a high-priority target. However, the computational patterns at the core of attention layers, i.e., the fundamental building blocks of FMs, exhibit several key bottlenecks. For each input element, these layers compute an *attention score* with respect to every other input obtained through a Softmax-normalized dot-product operation [4]. Attention complexity scales quadratically with the input sequence length, limiting both the practical size of the models and the length of the inputs they can handle. Moreover, the internal Softmax is crucial for normalizing the attention scores to a pseudo-probabilistic distribution, but it introduces significant computational challenges. Namely, the computation of exponentials for each attention score prior to normalization can be expensive and susceptible to numerical stability issues, particularly under large-scale or low-precision arithmetic conditions.

At the algorithmic level, these bottlenecks can be addressed with innovations such as sparse and linearized attention mechanisms [8], [9], as well as approximations and alternatives to the Softmax function [10]. All these approaches aim to mitigate the computational overhead and enhance the model's capacity to handle longer sequences more effectively, thereby extending the practical applications of FMs without compromising their performance.

On the HW side, a variety of platforms have been engineered from scratch or enhanced to provide the computational power and memory requirements essential for supporting this new class of models. The most commonly used hardware platforms to train and run FMs are graphics processing units (GPUs), such as NVIDIA's A100 and H100 [11], [12], coupled with efficient software libraries to limit memory transfers and efficiently distribute the workload. In a quest to surpass the efficiency of GPUs and primarily address the inference workloads, many new companies have focused on designing specialized AI accelerators, such as Cerebras [13] and Groq [14], which employ highly parallel architectures and fully specialized processing elements (typically large arrays of multiply-accumulate units), together with tailored closed-source kernel libraries. However, accelerators often lack flexibility and cannot compete with GPUs in rapidly adapting to new workloads.

This work presents the first FM deployment flow based entirely on an open-source Instruction Set Architecture (ISA), open hardware and open-source software. We target a many-tiny-core general-purpose RISC-V architecture with specialized RISC-V extensions and advanced direct memory access (DMA) engines, allowing for a highly flexible AI-oriented dataflow.

Viviane Potocnik, Luca Colagrande, Tim Fischer, Luca Bertaccini, L. Benini are with the Integrated Systems Laboratory (IIS) of ETH Zürich, ETZ, Gloriastrasse 35, 8092 Zürich, Switzerland (e-mail: name.surname@iis.ee.ethz.ch).

A. Burrello, L. Benini are with the Department of Electrical, Electronic and Information Engineering, University of Bologna, 40136 Bologna, Italy. E-mail: firstname.firstsurname@unibo.it

A. Burrello, D. Jahier Pagliari are with the Department of Control and Computer Engineering, Politecnico di Torino, 10129, Turin, Italy. E-mail: name.surname@polito.it

Manuscript received xxx 2024; revised xxx 2024.

This platform adopts a scalable, hierarchical structure that organizes cores into groups of parallel compute clusters. Advanced DMA engines facilitate high bandwidth heterogeneous memory movements, including core-to-core, cluster-to-cluster, and cluster-to-High Bandwidth Memory (HBM) communications, both one-dimensional (1D) and two-dimensional (2D), effectively hiding the latency of costly memory operations [15]. Each cluster [16] comprises eight compute cores with custom ISA extensions for SIMD-capable floating-point (FP) operand streaming and instruction repetition, as well as one DMA core. Compute cores are coupled to a 64-bit wide SIMD-capable floating-point unit (FPU) supporting `FP64`, `FP32`, `FP16`, `BrainFloat16` (`BF16`) [17], `FP8` (E5M2), and `FP8ALT` (E4M3) [18] formats. At the software level, we exploit these hardware capabilities to implement an optimized kernel library for FMs, including advanced attention implementations such as FlashAttention-2, fused layers combining head concatenation and input space projection in a logarithmic tree fashion distributed over compute clusters, and low-precision SIMD-based kernels. Our library is available open-source at https://github.com/viv-eth/snitch_cluster/tree/llm/end-to-end/sw/dnn.

We benchmark our work on five different FMs, including decoder-only LLMs and encoder-only ViT models, to demonstrate the flexibility of our HW platform and our SW kernels. Furthermore, we compare against both commercial and academic SoA architectures. We focus on the conventional (quadratic) attention, since algorithmic optimizations (such as linearized attention) are not accuracy-neutral at scale. Furthermore, quadratic attention allows direct comparisons with other state-of-the-art architectures.

The key contributions of this work are the following:

- We provide a comprehensive open-source FM library that supports encoder- and decoder-only models, leveraging the HW features and ISA extensions of a scalable RISC-V-based open-source multi-core platform. To improve the performance of our kernels, we leverage advanced DMA engines and demonstrate the benefits of using cluster-to-cluster data transfers, enabling layer fusion and reduction in main memory accesses.
- We provide a detailed ablation study showing how specialized RISC-V ISA extensions for latency-tolerant operand streaming and instruction repetition boost the performance by a factor of up to 35.6× on GPT AR mode, 16.1× on the GPT NAR mode and 12.8× for the ViT model class, respectively, compared to the base instruction set architecture (ISA), achieving FPU utilization above 79% in NAR mode.
- We explore the scalability of attention kernels among data precision and the number of available clusters on our HW target. We benchmark our FM library at `FP64`, `FP32`, `FP16`, and `FP8`. We also show how different attention block hyperparameters scale compared to the number of cores in terms of throughput (images/s) on all ViT models.
- To the best of our knowledge, we provide the first fully open-source software deployment of ViT and LLM models on an open-source RISC-V hardware architecture.
- Through comprehensive benchmarking, we demonstrate that our inference engine outperforms SoA platforms regarding HW utilization while maintaining the full flexibility of a many-core RISC-V architecture. We outperform SoA accelerators by up to 8× in terms of FPU utilization, with a minimum speedup of 1.81× compared to the best competitor.

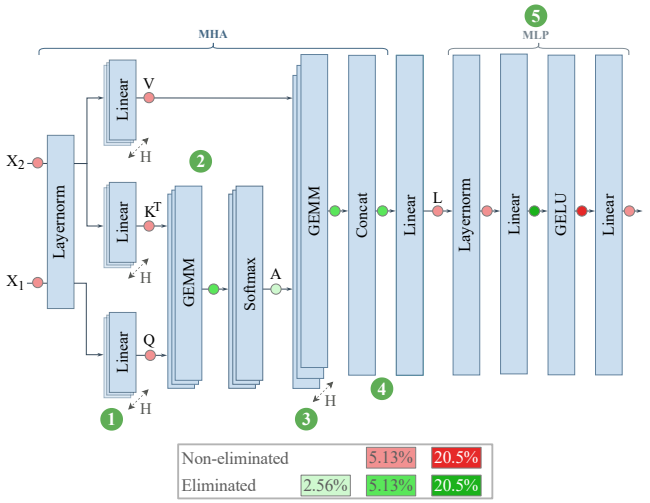


Fig. 1: Block topology of the basic Attention layer. The arrows are annotated with the percentage of the memory transfers needed for the specific tensor over the total number of transfers needed by the block, considering the GPT-j model in NAR mode and a sequence length of 2048. Red dots represent data reads from HBM of our implementation. Green ones are transfers done only at the cluster level.

The remainder of the manuscript is organized as follows. Sections II and III provide background and review of related work in full-stack FM deployment flows. Section IV offers an in-depth description of our target hardware platform and Section V details our software library. Section VII presents comprehensive findings and the results of our benchmarking experiments. Finally, Section VIII summarizes our study with concluding remarks.

II. Background

A. Attention Kernel

Transformers-based FMs are constructed by stacking multiple *attention* blocks, which correlate each element of a sequence $\mathbf{X_1}$ (e.g., language tokens or image patches) with all elements of another sequence $\mathbf{X_2}$ [4], [19]. Notably, in many cases, the two sequences coincide ($\mathbf{X_1} = \mathbf{X_2}$), and the operation is denoted as *self-attention*, as opposed to *cross-attention* when they differ. The ability of attention layers to dynamically *focus* on different input data segments has been shown to improve performance across diverse tasks, thus becoming a fundamental component in the architecture of modern SoA neural networks.

A scheme depicting a basic attention block is shown in Figure 1. The vectors that compose the input sequences organized as rows of the $\mathbf{X_1}$ and $\mathbf{X_2}$ tensors are first projected through learned linear transformations (1) to form three new matrices: the Queries (Q), obtained from $\mathbf{X_1}$, and the Keys (K) and Values (V), derived from $\mathbf{X_2}$. Mathematically:

$$\mathbf{Q} = \mathbf{X_1}\mathbf{W}_\mathrm{Q},\ \mathbf{K} = \mathbf{X_2}\mathbf{W}_\mathrm{K},\ \mathbf{V} = \mathbf{X_2}\mathbf{W}_\mathrm{V} \tag{1}$$

Where $\mathbf{W}_{Q,\,K,\,V}$ have dimensions E $\times$ P, which represent the Embedding (E) space (i.e., the number of input features) and the Projection (P) space, respectively [4].

The attention weights (or scores) $\mathbf{A}$ are then calculated as the scaled dot product of matrix Q and K, normalized using a Softmax function to convert them into pseudo-probabilities (2). The scaling factor $\sqrt{\mathbf{P}}$, where $\mathbf{P}$ corresponds to the dimension of the key vector, ensures that the dot products do not become excessively large, which aids in maintaining numerical stability during training. The normalized scores represent the degree to which each element of $\mathbf{X_1}$ should attend to all elements of $\mathbf{X_2}$, directing the network's *focus* towards more relevant parts of the input. Accordingly, a new sequence is produced in which each element is a weighted sum of the V vectors, where the weights are the Softmax-normalized scores. Formally:

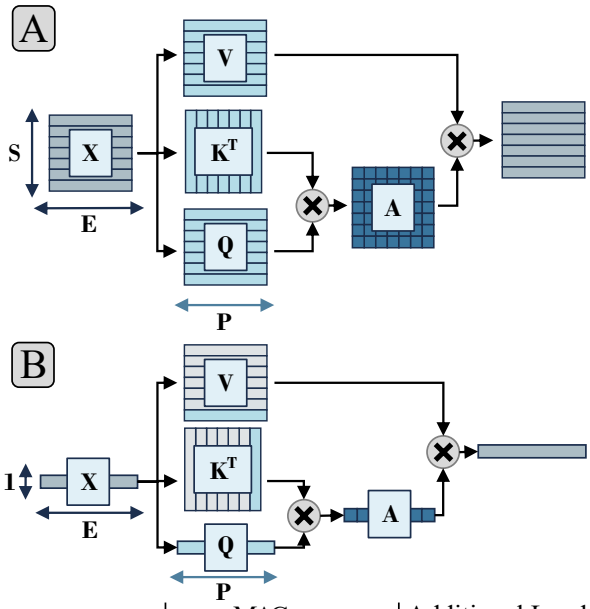
$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) \doteq \mathbf{AV} \doteq Softmax_{\text{over keys}}\left(\frac{\mathbf{QK}^{\mathrm{T}}}{\sqrt{\mathbf{P}}}\right)\mathbf{V} \tag{2}$$

The commonly used Multi-Head Attention (MHA) extends this basic layer to enhance the model's capability to process information from multiple representation subspaces simultaneously [4]. It allows the model to capture various aspects of the input data in parallel *heads*, each performing the attention process independently with different learned linear transformations. After each head computes its attention outputs independently, the results are concatenated and once again linearly transformed (4). This step combines the different representations of each head, forming a unified output. The output vectors of the MHA block are normalized with a Layernorm operation and then processed by a multi-layer perceptron (MLP) (5). The first dense layer of the MLP expands the input feature dimension to the hidden dimension. The subsequent Gaussian Error Linear Unit (GELU) activation function introduces non-linearity, which is essential for learning more complex patterns. The final linear layer projects the expanded representations back to the original input space E.

While new attention block variants are arising, such as the Grouped-Query Attention (GQA) [20], we describe and employ classical attention for our experiments to enable fair comparisons on the same workload with SoA hardware. However, the findings of our work can be applied to any other Transformer-based architecture.

B. Foundation Models

First transformer architectures revolutionized NLP [4] and later, CV [2] and many other domains. Nowadays, transformers are the dominant architecture in terms of accuracy in most complex AI tasks: for instance, OmniVec, a multi-modal model based on popular ViT and Bidirectional Encoder Representations from Transformers (BERT) transformer architectures, reaches the SoA accuracy in many fields such as image recognition, video object recognition, audio recognition or text summarization [21]. Moreover, new megatrends such as the one towards generative AI have been opened thanks to this new network topology, leading to the creation of transformer-based models, e.g., ChatGPT, which are used in daily life by millions of users [22].



	MACs	Additional Loads
Enc.-block	3EPSH + 2PS^{2}H	0
Dec.-block	3EPH + 2PSH	2P(S - 1)H

Fig. 2: Operation performed by the fundamental ViT and GPT blocks.

However, to reach this remarkable performance, transformers require huge architectures, that are difficult to train from scratch for each new task; coupled with the effectiveness of newly developed label-free training schemes, FMs [5] have emerged as the most natural paradigm for training and deploying transformers, leveraging vast amounts of data [2], [23]. FMs, such as the previously mentioned OmniVec, undergo extensive pre-training on large and diverse datasets, often through self-supervised procedures, which helps them develop a generalized and wide-range understanding of language patterns, visual information, audio, and many more mixed modalities, depending on the nature of training data. Once pre-trained, these models can be fine-tuned at a much lower cost (in terms of data and computing) to perform specific downstream tasks accurately.

The large majority of SoA FMs belong to one of two big classes. *Encoder-only* models, which include classifiers such as ViTs, are feed-forward neural networks that simultaneously receive the entire input sequence and generate their output in a single inference pass. *Decoder-only* models, in contrast, are mainly used for generative tasks, typically in an autoregressive fashion. In this case, the final output is generated through multiple iterations of the network, each fed with the output of the previous one. In our work, we describe a general FM kernel library and a flexible hardware platform able to optimize both model families. We use the dominant ViT and GPT as representatives of the two categories to demonstrate our results.

ViT-based FMs The ViT encoder-only architecture exploits a first convolutional layer with a stride equal to the filter dimension to create a linearized series of patches used as the

input sequence. The heart of the network constitutes Multi-Head Self-Attention (MHSA) blocks that correlate all the input tokens with themselves, therefore having $\mathbf{X_1} = \mathbf{X_2}$. Fig. 2-A depicts the matrix operations executed and the tensor dimensions for the key blocks of the MHSA section. The operations are quadratically proportional to the input sequence length S and linearly proportional to P and E. All involved operations are on multi-dimensional matrices, and a single pass through all the network blocks produces the desired classification. As reference models for this family, we considered ViT-{Base, Large, Huge}, denoted with suffixes -B, -L, and -H in the following, given their maturity and widespread adoption [2]. For more details on the three architectures, we refer the reader to [2]. Although more recent models have been developed, the basic building blocks remained similar, and, therefore, our library can be identically employed for their deployment [24], [25].

GPT-based FMs Decoder-only GPT models belong to the LLMs family. Their primary task is text generation: given an input text prompt, the model generates the most likely continuation. Fundamentally, the input prompt is *tokenized* [26], and the sequence of tokens is fed to the model for the so-called pre-fill phase. Then, the text continuation is generated one token at a time, iteratively invoking the model in an autoregressive fashion, after concatenating the generated tokens to the input prompt. A more detailed description of GPT model execution can be found in [23]. Computationally, there are two crucial differences compared to the encoder model: First, the attention is causal, which means that the new tokens can only attend to the previous ones in the sequence and not to subsequent ones. Second, a single new token and not the entire output is generated at each pass through the network (see Fig. 2-B). For instance, to generate the reply to the question *What is your name?*, the model is first provided with the tokenized question as a prompt. In the first iteration, the initial token of the continuation (e.g., *I*) is generated. Then, such a token is appended to the input, and the network is invoked again, producing the following output token (e.g., *am*). This procedure repeats until a special end-of-sentence token is produced or another stopping criterion is met (e.g., a maximum length). Compared to encoder-only models, we observe a shift from matrix-matrix operations towards matrix-vector operations, which are less compute-intensive. In fact, except for the initial network invocation (prompt processing or pre-fill step), a single new query vector per layer is processed at each subsequent step, computing its attention weights against all previous keys and values. Note that the K and V projections involve just a vector-matrix multiplication since the keys and values relative to tokens processed in previous inference passes can be stored in memory to avoid re-computation in the so-called *KV cache* [27]. These differences will be analyzed in detail in Sec. VII.

C. Transformer Optimization

In the domain of FMs, optimizing for computational efficiency without compromising performance is a crucial challenge, given the high amount of resources required by these models and the recent trends towards embedding them on constrained mobile or edge devices [28].

An important category of optimizations in the literature looks at the network topology, either changing the attention block's structure or simplifying the overall network. For instance, a series of works [29], [30] proposed simplifying the attention computation by either low-rank approximation or fixing a dimension, reducing the complexity with respect to S from quadratic to linear. Other approaches focus on sharing the K and V matrices across multiple heads, obtaining the so-called GQA found in more recent LLaMa models [20], or applying sparsity to the projection weights to reduce the overall memory occupation. Some works specifically focus on LLMs by reducing the KV-cache size, either by compression or by evicting some unimportant tokens [31]. All these optimizations are orthogonal to our work, and indeed, the flexibility of our architecture enables adopting all these algorithmic optimizations once they are fully proven to be robust and competitive in accuracy.

Consequently, we focus our analysis on optimizations concerning canonical attention, the most widespread ViT and GPT transformer architecture, and the corresponding data precision. Despite many studies exploring the possibility of using low-precision *integer* formats for FMs, such as 8-bit [23] or 4-bit [32], these quantization schemes inevitably lead to a non-negligible loss in accuracy. In contrast, the research on `FP8` quantization, as documented in [18] by NVIDIA researchers' testing across various GPT, encoder-decoder, and BERT models, shows that low bit width floating point formats can often maintain identical performance without any loss, while significantly enhancing computational efficiency. Hence, in this work, our efforts have been concentrated on testing the performance scalability of our hardware and software stack considering precisions ranging from 64-bit down to 8-bit floating-point. A second key optimization knob to improve FM performance consists in reducing the number of costly accesses to the main memory. To this end, we exploit the *FlashAttention-2* implementation [10] and a fused linear concatenation layer that leverages binary cluster-to-cluster sum reduction to concatenate the attention heads and project them to the original input space dimension. Both techniques minimize frequent communication between the main HBM and the on-chip scratchpad memory (SPM) during the processing of attention blocks and can be applied to any Transformer-based FM. While both have already been explored in literature, we adapt them to our HW platform, exploiting its unique features.

III. Related Work

While GPUs are still the most commonly used hardware platforms for AI workloads [11], [33], [38], various companies have designed their specialized accelerators in the last few years [34], [35], [37], [45]. Table I presents an overview of some of the most recent platforms specifically designed for AI training and inference, often focusing on LLMs and FMs.

A common denominator of those architectures is the replication of a base compute cluster combining multiple processing elements (PEs) with an L1 cache or SPM via a low-latency interconnect. The replicated clusters are interconnected by a latency-tolerant global Network-on-Chip (NoC) and rely either

TABLE I: State-of-the-art platforms for FMs execution.

Platform	Technlogy	Max. Frequency	Memory	Cores	Data Format	Special Features
Commercial Platforms						
A100 [11]	7 nm N7 TSMC	1.41 GHz	164 kB SPM + 40 GB HBM	6912 FP32 + 432 TCs	FP8/FP16/FP32/ FP64/INT8/INT4	TCs
H100 [33]	4 nm N4 TSMC	1.78 GHz	256 kB SPM + 80 GB HBM	16896 FP32 + 528 TCs	FP8/FP16/FP32/ FP64/INT8/INT4	Transformer Engine, Tensor Cores
SambaNova DataScale SN30 [34]	7 nm N7 TSMC	-	640 MB SRAM + 1 TB DRAM	1280 PCUs	BF16/FP32/INT32/ INT16/INT8	Units (PCUs)
Cerebras CS-2 [35]	7 nm N7 TSMC	1.1 GHz	40 GB SRAM + 1 TB DRAM	850'000	FP16/FP32	WSE
GROQ [14]	14 nm GF	900 MHz	2.3 TB SRAM	10440 TSPEs	FP16/FP32/INT8	LPU
Habana Gaudi 2 [36]	7 nm Intel	-	48 MB SRAM + 96 GB HBM	24 TPC + 2 MME	FP8/FP16/FP32	Tensor Processor Core
Habana Gaudi 3 [37]	5 nm Intel	-	96 MB SRAM + 128 GB HBM	64 TPC + 8 MME	FP8/FP16/FP32	Tensor Processor Core
AMD MI250 [38]	6 nm FinFET TSMC	1.6 GHz	8 MB SRAM + 128 GB HBM	4096 Stream Processors	FP16/FP32/FP64/ INT8/INT4	AMD CDNA 2
Academic Platforms						
AccelTran [39]	14 nm FinFet Intel	700 MHz	13 MB 3D RRAM + 16 GB LP-DDR3	64 PEs	20-bit fixed-point	DynaTran + dynamic data flow
Wang et al. [40]	28 nm CMOS	510 MHz	336 kB SRAM	4 Cores (32 PE Lines)	INT12	Approx. Attention
Kim et al. [41]	28 nm CMOS TSMC	200 MHz	500 kB SPM	HDSC + HMAU OSSU, IWGU	INT8	SNN/DNN transformer Big-little network
FreFlex [42]	28 nm CMOS TSMC	1.1 GHz	64 kB SRAM	32 x 16 PE array	INT8	SDFM
Ayaka [43]	28 nm CMOS TSMC	430 MHz	544 kB SRAM	RPAS unit + transformer core	INT8/INT16	CSP + HDPE
Tambe et al. [44]	12 nm FinFET N/A	717 MHz	647 kB SRAM	STP	FP4/FP8	MP MAC unit
Ours	12 nm FinFET GF	1 GHz	128 kB SPM + 16 GB HBM	16 clusters 9 cores/cluster	FP8/FP16/ FP32/FP64	MiniFloat ISA, cluster2cluster communication stream semantic registers (SSRs)

entirely on external main memory or on some distributed on-chip cluster memories.

For example, Nvidia's A100 [11] groups multiple Texture Processing Clusters with 108 streaming multiprocessors (SMs) into GPU Processing Clusters. Each SM combines four warps with an L1-data cache and a shared memory of 164 kB, whereas each warp combines a tensor core, 16 INT32 and FP32 cores, and eight FP64 cores with a shared 64 kB register file. The tensor core supports FP16, BF16, FP32, FP64, INT8, INT4, and binary formats, and its 32-thread granularity enables 256 FP16/FP32 fused multiply-accumulate (FMA) operations per cycle, corresponding to the computation of an 8×4×8 mixed-precision matrix multiplication.

The H100 [33] GPU doubles the HBM memory of the A100 and increases the number of total cores from 6912 to 16896, and introduces support for FP8. Further, it includes a Transformer Engine (TE) optimized for handling multi-trillion parameter models, significantly enhancing performance for FMs. The engine supports Automatic Mixed Precision (AMP), which dynamically adjusts the precision of computations during training and inference. Additionally, the TE supports fused operations, combining multiple computational steps into a single kernel. The architecture of the TE is designed to scale efficiently across multiple GPUs, allowing for parallel training and inference of large models, further reducing training times. This scalability is complemented by an advanced software stack optimized for transformer workloads, including libraries like NVIDIA CUDA, cuBLAS, and cuDNN.

SambaNova DataScale SN30 [34] uses second-generation Reconfigurable Dataflow Units (RDU) as PEs. It uses a 7 nm process node and supports BF16 and FP32 data formats. Cerebras has scaled its accelerator architecture to a wafer-sized chip called wafer-scale engine (WSE) [35]. The second generation, WSE-2, with 46 000 mm^2, is the largest processor ever built and combines 850k AI cores with a wafer-scale high-bandwidth, low-latency fabric arranged in a 2D mesh. Each general-purpose core combines four 16-bit fused multiply-accumulate (FMAC) units, 8× 6 kB static random-access memory (SRAM) banks (total 48 kB), and 256 B of software-managed cache for often accessed data structures such as accumulators.

GROQ's Language Processing Unit (LPU) architecture [14], taped out in 14 nm, employs a linear array of tensor streaming processing elements (TSPEs) interleaved with memory units. It is designed for deterministic performance and eliminates control flow variability to maximize data throughput. The LPU supports FP32, FP16, and INT8. Each LPU comprises 220 MB of on-chip SRAM, divided across superlanes, enabling high

performance without the need for external DRAM. The LPU can compute more than 400000 integer multiply–accumulate (MAC) operations per cycle, achieving up to 1000 TOPS. By having the compiler explicitly control all aspects of program execution, including instruction scheduling, data movement, and resource management, runtime variability is eliminated.

The Habana Gaudi 2 [36] and its successor Habana Gaudi 3 [37] processors feature two/eight Matrix Multiplication Engines (MME), 24/64 Tensor Processor Cores (TPCs), and 96/128 GB of HBM, respectively, providing robust platforms for efficient scale training. They support `FP8`, `FP16`, and `FP32` data formats. Lastly, the AMD MI250 [38] utilizes a 6 nm process with 128 GB of HBM2e, focused on high bandwidth and large buffer capacities essential for handling large datasets efficiently. It supports `FP16`, `FP32`, and `FP64`, together with low-precision integer formats, aligning with the needs of diverse AI training and inference workloads.

In contrast to large industrial products, academic research has focused on smaller-scale hardware and models, investigating specific optimizations. For instance, AccelTran [39] focuses on sparse transformers, using a dynamic inference scheme to prune activations at runtime. In most academic works [39]–[43], the authors primarily focus on specific hardware optimizations to support different integer quantization schemes that can limit the applicability to high-accurate LLMs. Additionally, they propose architectures tailored to specific models, such as fusions between Transformers and DNNs or Spiking Neural Networks (SNNs) [41]. Tambe et al. [44] are the only ones employing floating point data types. However, the authors combine floating-point with a series of other software optimizations that could limit the utilization of their work for huge LLMs. These optimizations include token entropy-based early exit, Mixed-Precision (MP) (`FP4`/`FP8`) prediction, and sparsity. Until now, no open-source general-purpose academic hardware platforms have demonstrated full encoder/decoder execution of widespread foundation models like GPT and ViT.

To fill this gap, in this work, we explore an *open-source*, scalable architecture comprising a hierarchical grouping of four RISC-V multi-core Snitch compute clusters [46], achieving an inter-cluster bandwidth of 64B/cycle while sharing 64B/cycle bandwidth to another hierarchy stage of cluster grouping. We exploit on-chip interconnect to enable flexible cluster-to-cluster direct communication without passing through main memory. This capability is not fully supported even in the most advanced commercial platforms, where direct on-chip communication is either restricted to a static peer-to-peer dataflow dataflow (e.g., Groq [14]) or limited to an expensive shuffle (e.g., in NVIDIA H100 [12]).

IV. Many-Core RISC-V Hardware Platform

In this work, we optimize FM inference on an open-source, scalable, many-core architecture optimized for FP-intensive workloads with Machine Learning (ML)-centric ISA extensions. The foundation of the hardware platform is the *Snitch* cluster [16] described in Section IV-A and shown in Figure 3, which is replicated and implemented in a hierarchical multi-cluster configuration, as discussed in Section IV-B and shown in Figure 4.

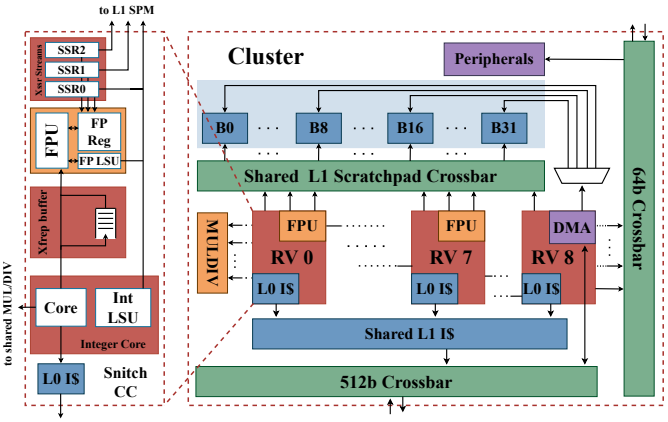


Fig. 3: Architecture of the RISC-V compute cluster with ISA extension `Xfrep` and `Xssr`.

A. Compute Cluster Architecture

The compute cluster is the basic block of our scalable HW architecture [16]. Figure 3 illustrates the architecture of the cluster, which is composed of 9 tiny 32-bit in-order integer RISC-V cores called Snitch [16], each coupled with an L0 instruction cache and a 64-bit wide single instruction, multiple data (SIMD)-capable FPU, supporting a wide range of FP formats and mixed-precision operations [47]. Additionally, the cores share an integer division/square-root unit and a two-way 8 kB L1 instruction cache (I$). To achieve FPU utilization rates above 50% with Snitch's lean single-issue pipeline, the architecture integrates two extensions [16]: `Xssr` and `Xfrep`. The `Xssr` augments three FP register file entries to SSRs with capabilities for address generation, enabling latency-tolerant data prefetching. The `Xfrep` implements an instruction repetition buffer called FP repetition (FREP), reducing the pressure on the L1 instruction cache and preventing the overhead of loop handling. The joint effect of these extensions is that the inner loop, e.g., a dot product operator, only contains the FMA instruction. Hence, FPU utilization can rise to the 90%-region.

Eight of these cores are in charge of parallel workloads, while the ninth core is used for cluster coordination and controls a DMA unit with hardware support for 1D and 2D data transfers. All cores share a tightly coupled 128kB L1 SPM via a single-cycle interconnect. The SPM itself is organized as 32× 64-bit interleaved memory banks. The DMA unit can access eight banks in parallel via a dedicated 512-bit interconnect and shares a duplex 512-bit Advanced eXtensible Interface (AXI) crossbar to the higher-level memory system with the I$. The cores can access the global memory system over a duplex 64-bit AXI crossbar. Furthermore, all cores support atomic operations and use hardware barriers for synchronization. In Sec. V, we will describe how the peculiar HW features are exploited in our kernel library for FMs. Notice that the cluster structure is not explored in this work but is only used as a basic building block of our scalable architecture. The interested reader is referred to [16], [47] for in-depth explorations of the cluster architecture.

1) Low and Mixed-Precision Floating-Point Support: Low-precision data types allow for reduced memory footprints and

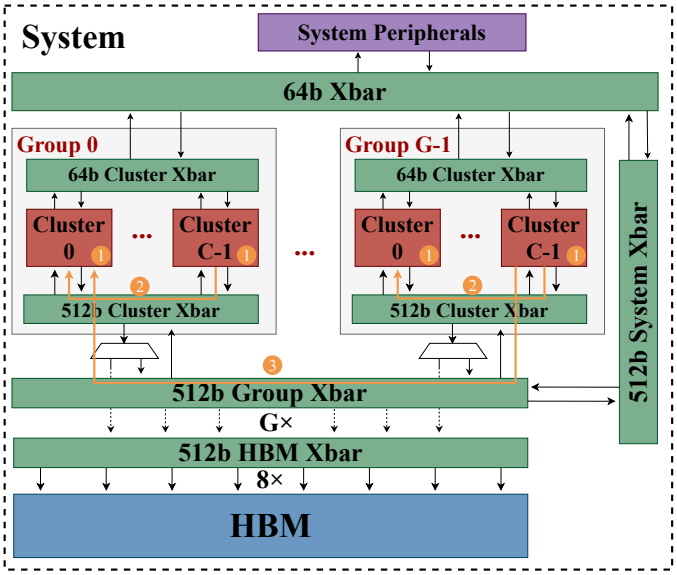


Fig. 4: Scalable multi-cluster architecture with hierarchical heterogeneous memory interconnect.

more energy-efficient inferences for ML models. For these reasons, Snitch's FPU supports a wide set of FP formats: `FP64`, `FP32`, two 16-bit FP data types (`FP16`, `BF16`), and two 8-bit FP data types (`FP8`, `FP8ALT`). The `FP8` and `FP8ALT` formats include a 2-bit/3-bit mantissa field and a 5-bit/4-bit exponent field, respectively. By leveraging the wide set of FP formats, the computation precision can be tuned to the specific application requirements, achieving higher performance and energy efficiency. Each Snitch's FPU is pipelined to achieve a throughput of 1 instruction per cycle. Since the FPU supports low-precision SIMD computation, twice the number of operations per cycle can be achieved when halving the data width. The Snitch cluster achieves, therefore, a theoretical peak performance of 16, 32, 64, and 128 FLOP/cycle, respectively for `FP64`, `FP32`, `FP16`/`BF16`, and `FP8`/`FP8ALT`, when parallelizing the computation over 8 cores (where 1 fused multiply-add = 2 FLOP). Furthermore, Snitch's FPU supports SIMD short widening dot-product instructions [47] working on 8 or 16-bit inputs and accumulating with 16 or 32 bits, respectively. These mixed-precision instructions compute the equivalent of two cascaded widening fused multiply-add. They ensure the speedup enabled by the lower-precision inputs even when accumulating at higher precision while retaining a higher accuracy in long accumulations (for example, in GEMM kernels), as previous works have shown that it is beneficial [48], [49]. In our work, we explore the utilization of different data precisions, showing the speedup achievable with a dedicated FM library capable of exploiting all the SIMD operations and maintaining higher precision where needed~~(e.g., softmax)~~, as will be detailed in section V-A2.

B. Scalable Multi-Cluster Architecture

The compute cluster described in the previous section can be scaled up to a multi-cluster architecture as proposed in [50] and silicon-proven in [46]. A block diagram is shown in Figure 4. The first level of scalability is achieved with a *Group* of C compute clusters. The clusters of a group are connected via a narrow 64-bit crossbar for fast synchronization and a wide 512-bit AXI crossbar for efficient and high-bandwidth inter-cluster access. The next level of scale-up is achieved by connecting G Groups. Similar to the cluster level, a group-level AXI crossbar allows fast access between different groups. Additionally, the groups access eight HBM channels over a wide HBM crossbar for high-bandwidth access to the main memory.

This type of memory hierarchy enables increasingly higher bandwidth with each level while retaining scalability (see Figure 4). The first level, the cluster-to-SPM interconnect (1), has a peak bandwidth of 256 GB/s. The all-to-all *Cluster* and *Group* crossbars have a $C \times 64$ GB/s and $G \times 64$ GB/s peak bandwidth for inter-cluster (2) and inter-group (3) communication, respectively. The last level, the groups-to-HBM connection, has a bandwidth of 410 GB/s [51] that can be fully exploited for configurations where G is at least equal to the number of HBM channels. In our work, we design our FM library to exploit each memory level and the heterogenous interconnect maximally: we primarily maximize the accesses to the local SPM memory. Then, we rely on cluster-to-cluster communication to efficiently transfer the data among clusters without intermediate copies to HBM. The access to the HBM is limited to few accesses for loading and storing the input and output tensors, whereas intermediate tensors are stored in cluster memories.

V. Foundation Model Library

This section introduces our software stack to deploy Transformer-based FMs efficiently onto the platform introduced in the previous section. First, we describe our single-layer optimizations on the General Matrix Multiplication (GEMM), Layernorm, GELU kernels and our MHA layer implementation inspired by [10]. Then, we explain how we exploit layer fusion and the hierarchical interconnect of our hardware to maximize data reuse and minimize transfers from and to the HBM. Our FM library supports `FP64`, `FP32`, `FP16`, and `FP8` layer variants and includes examples of full network execution.

A. Layer optimization

1) GEMM: In Transformer-based models, GEMM operations are fundamental components of the MHA, projection layers, and MLP layers. The GEMM operation computes $\mathbf{C} = \alpha\mathbf{A} \times \mathbf{B}$, where $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are matrices of size $M \times K$, $K \times N$ and $M \times N$, respectively, and α is an optionally present scalar multiplying factor. We optimize three crucial knobs in our implementation: the spatial and temporal tiling (i.e., the partitioning of the layer computation between clusters and between successive iterations of the same cluster), the intra-cluster parallelization, and the GEMM's innermost loop.

Matrix tiling is a key component of our implementation strategy. Tiles are first spatially distributed between clusters and then temporally distributed over iterations. As shown in Figure 5, we support spatial tiling in the M and K dimensions (Figure 5-A). Additionally, we support temporal tiling in all

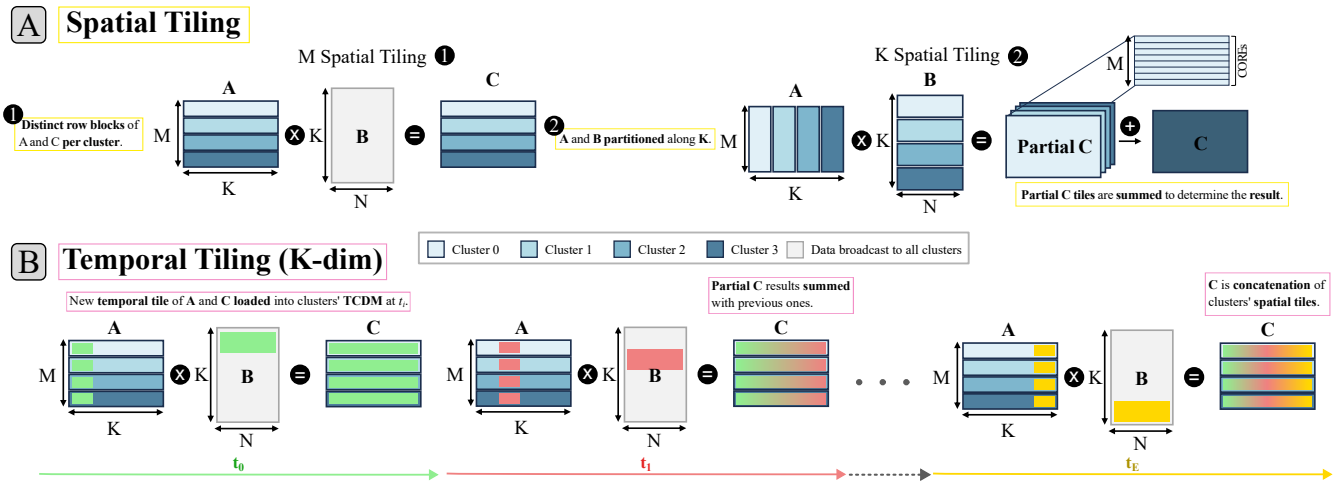


Fig. 5: Illustration of the spatio-temporal GEMM Tiling. A) Spatial tiling in the M and K dimensions: clusters process distinct row blocks by partitioning matrices A and C, while matrix B is broadcast to all clusters. B) Combined spatial (M) and temporal (K) tiling: at each time step (t_0, t_1, ..., t_E), a single temporal tile (represented by green, red, and yellow rectangles) is loaded into the cluster memory.

matrix dimensions (an example of K-tiling is shown in Figure 5-B) to support all kernel sizes in our target FMs, where spatial and temporal tiling along one or two dimensions alone would not suffice to fit tiles in the cluster-local L1 SPMs. Spatial tiling in the M dimension maximizes utilization and repetitive memory access to the same data: each portion of the A matrix is loaded by a single cluster only, while the matrix B is broadcasted to all clusters with a single access to the external memory. At each time-step i, we only load a single temporal tile of the matrices **A** and **C** in the cluster memories. We spatially tile on dimension M instead of N since the data are stored contiguously in memory with an MN layout, and M (which usually corresponds to the sequence dimension for FMs) is always large enough for tiling. Figure 5-B shows the process over time: at each time step, a different portion of each spatial tile is loaded in each cluster memory, and a partial C matrix result is produced and summed together with the previous ones generated. At the last time step t_E, the last partial C spatial tiles are generated and summed with the previous ones (e.g., green plus red partial results in step t_1). The result of the GEMM is the concatenation of the spatial C tiles from the different clusters.

Spatially tiling the K dimension is generally not optimal since it generates partial sums in each cluster that need to be reduced. However, K-tiling becomes useful when the required input tiles of matrices **A** and **B** are already in the corresponding clusters because they have been produced by a previous operation. In this situation, employing the tiling on the M dimension would cause memory transfers between clusters proportional to the dimension of the input matrices. The MHA layer is the most notable example of this case. As the first GEMM operations ($Softmax(^{QK^T}/_{\sqrt{P}})V$) are split over the heads (see Sec. V-A2), the following linear layer can be tiled on the heads dimension, corresponding to the internal K dimension in our generic notation. When K is spatially tiled, the clusters' locally computed partial results are aggregated through a tree-wise reduction process, described in Sec. V-B.

Once each cluster is assigned a spatial-temporal tile, as described above, the computation is further parallelized at intra-cluster granularity. Namely, the computation of each matrix tile is parallelized over the M dimension by distributing distinct rows of the output matrix to different cores of the cluster (see Figure 5).

At the level of every single core, the GEMM's loop execution is optimized using the custom ISA extensions of the Snitch processor, i.e., `Xfrep` and `Xssr`. We map the innermost loop to the `Xfrep` instruction, therefore improving the latency thanks to the elimination of indexing and branching instructions. The innermost loop computes the dot product between a row of the **A** matrix tile and a column of the **B** matrix tile. To further reduce the overall latency of the loop, we map the **A** and **B** tiles to SSRs 0 and 1, exploiting the `Xssr` operation to remove load latency. In this way, we can stream operands to FPU instructions in all cores without having to issue additional load instructions. Furthermore, we unroll the innermost loop by a factor of 8 to hide the read-after-write (RAW) stalls due to the FPU pipeline latency. As a last optimization, our single- and low-precision GEMM kernels are designed to exploit the SIMD ISA extensions of Snitch's 64-bit wide FPU for further parallelization, increasing the throughput of the innermost loop by a factor of 2, 4, 8 for `FP32`, `FP16`, and `FP8`, respectively.

Data movement is double-buffered at the cluster level using the DMA engine. Data are copied from/to the main memory or from/to the SPM memory of clusters.

2) FlashAttention-2: The MHA block comprises two GEMM operations, a Softmax operation, and a final linear projection. We exclude the Q, K, and V projections from this paragraph since those are simple GEMM operations and can be optimized as discussed above. To optimize MHA, we draw inspiration from the forward pass of the *FlashAttention-2* algorithm proposed by Dao [10]. This algorithm efficiently computes the fused scaled dot product attention by dynamically generating a tiled Softmax on the fly. We implement the same dataflow proposed in the original FlashAttention-2 paper [10].

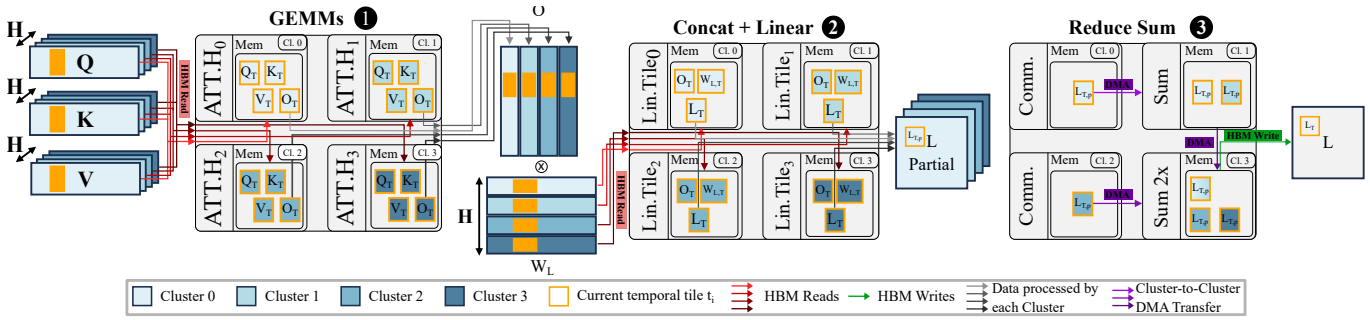


Fig. 6: Example of MHA block mapping on an architecture with 4 clusters. Red/green arrows represent reads/writes from/to HBM, gray arrows indicate the total data seen per cluster in TCDM, and purple yellows correspond to DMA transfers between clusters. The orange data chunks represent the current data in the clusters' SPM in a generic time step t_i.

We draw inspiration from the parallelization scheme presented therein, with the difference that *we map H attention heads to H distinct Snitch clusters*, potentially spanning multiple groups (indicated with different blues in Figure 6) and that *every cluster processes all the tiles* in the FlashAttention-2 dataflow in a time-iterative fashion in contrast to distributing tiles across GPU warps (in Figure 6 we depict the execution of a single temporal tile at the generic time t_i, for each MHA sub-operation.) All GEMMs involved in the individual attention head computations are parallelized exclusively within the associated cluster along the M dimension, as described in Section V-A1. Similarly, we parallelize the calculation of the row statistics for the online Softmax across the cores in a cluster. As noted by Dao, this strategy is embarrassingly parallel, so all clusters operate independently without the need for inter-cluster synchronization and communication. As anticipated in Section V-A1, the final linear layer is parallelized on the heads (corresponding to the K dimension), exploiting a logarithmic reduction to accumulate partial results. Figure 6 depicts the complete MHA kernel mapping. In orange, we show the data loaded from the HBM to the memory of each cluster in a generic time step t_i. Notice that at each time step, only a portion of the L partial matrices is generated and stored by each cluster; then, a reduction-sum is performed over all clusters to generate a single temporal tile of the final L matrix.

The optimized `FP32`, `FP16`, and `FP8` FlashAttention-2 layer implementations leverage the respective optimized GEMM kernels to accelerate the computation. In particular, we quantize the input operands of the compute-intensive GEMM operations. All other operations, particularly the non-linear operations involved in the Softmax calculation, are performed in single-precision FP arithmetic in all implementations, following the SotA practice [18], [52]. Conversion operations from low- to single-precision format are inserted at the output of the $\mathbf{Q} \times \mathbf{K^T}$ GEMM operation and before the final GELU. Vice versa, single- to low-precision conversions are performed before the final $\mathbf{A} \times \mathbf{V}$ GEMM operation.

3) Layernorm: The Layernorm operation is spatially tiled on the output matrix row dimension. Then, if the individual tiles do not fit the clusters' L1 SPM, they are temporally tiled on the column dimension. The Layernorm is first parallelized within a cluster on the eight compute cores. Then, it is further optimized thanks to the SSRs in combination with the `Xfrep` extension to reduce the number of instructions in the innermost loop, similarly to what is done for the GEMM kernel. Lower precision formats exploit the SIMD arithmetic capabilities to parallelize normalization operations further.

4) GELU: To avoid costly division operations and the computation of the $tanh$ function, we approximate the GELU with the i-GELU polynomial described by Kim et al. [53], which allows us to retain identical accuracy in all tasks benchmarked in the original paper.

B. Memory Accesses Optimization

Besides optimizing individual kernels, efficiently implementing attention layers requires careful consideration of memory access. In particular, maximizing data reuse in the higher levels of the memory hierarchy is crucial to reducing bandwidth requirements of the external HBM and avoiding main memory access, which carries a high cost in terms of energy and latency.

To pursue this goal, we apply *Layer Fusion* techniques in the MLP and MHA blocks. No intermediate buffers are copied back to the HBM memory. More specifically, in the MLP, we fuse the computation of the element-wise *GELU* activation function with the preceding *Linear* layer. In the *FlashAttention-2* layer, we fuse the MHA block with the subsequent Concat and Linear layers (see Figure 6). As explained below, this requires an additional reduction operation. As can be noted in Fig. 1, thanks to our layer fusion and our hierarchical interconnect, we can reduce the total reads from the HBM by up to 1.6× (removing the memory transfers marked in red in the figure) for the GPT-J model, stepping from 624 MB down to 384 MB. Note that this reduction directly translates to latency and energy gains.

The right part of Figure 6 shows the fusion details for the MHA. As the output of the GEMM block ❶, every cluster produces a different output matrix per head. In an un-fused implementation, these blocks would have to be written back to the main memory to be subsequently loaded again by the following Concat layer. Instead, we use the locally computed matrixes without going through the HBM. To this end, the $\mathbf{W_L}$ weights of the final linear layer are loaded in row-wise tiles into the different clusters, layer's GEMM along the H dimension, producing many partial output matrices L_c, each of dimensions $S \times P$ ❷.

Aggregating these partial results in a traditional shared-memory system would unavoidably mandate going through the

TABLE II: Foundation Models benchmarked.

	ViT-B	ViT-L	ViT-H	GPT3-XL	GPT-J
Blocks	12	24	32	40	28
Params	86M	307M	632M	1.3B	6B
E	768	1024	1280	2048	4096
P	64	64	80	128	256
S	197	197	197	[128-2048]	[128-2048]
FF	3072	4096	5120	8192	16384
H	12	16	16	16	16

main memory or the first shared cache level. As a consequence, accesses from all clusters would be serialized. Instead, thanks to the cluster-to-cluster and group-to-group connections in our hierarchical interconnect, we can implement the reduction in parallel and take advantage of the inherent data locality of nearby clusters without sending data to HBM. This is achieved by accumulating in a logarithmic reduction fashion among the clusters and groups ❸. The depth d of the binary reduction tree is determined as $d = \log_2(C \cdot G)$, where $C \cdot G$ is the total number of clusters to which a tile of heads is mapped. Based on the current level of the binary tree, we determine whether a cluster is active and whether it will send or receive a tile of the partial result matrix. The sending cluster's DMA engine performs the transfer. If the cluster is designated as a *receiver*, it will perform the sum reduction of the two tiles and proceed to the next level in the tree. The reduction is first performed among clusters in a group ❷ and then among clusters in different groups ❸ (see Figure 4). Finally, the last cluster stores the fully reduced L matrix in HBM.

In summary, this additional reduction step translates the SPH stores and SPH loads from the external memory needed before the Concat layer of the MHA block into SPH cluster-to-cluster transfers to aggregate the Linear layer results. As detailed, besides being less energy-hungry and faster, given that we do not have to access external memory, these transfers are also executed in parallel thanks to the logarithmic reduction tree, further reducing the overall MHA latency.

1) Double buffering: We employ *double buffering* in all kernels to minimize the memory transfer overhead. The kernel process initiates with a priming phase, where the initial data set is loaded into the cluster's shared L1 SPM. Subsequently, the kernel computation begins. Concurrently, the dedicated DMA core preloads the next data chunk into the L1, preparing it for the next computation cycle, either from the main memory or from one other cluster L1 memory. This method ensures a seamless transition between computation and data movement phases without idle time, as data unloading and new computation iterations occur simultaneously. Therefore, we hide the latency associated with data transfers in and out of the memory.

VI. Experimental Setup

A. Foundation Models benchmarked

We considered five models to analyze our comprehensive deployment flow encompassing the FM software library and the scalable multi-core HW platform. All model hyperparameters are reported in Table II. *Params* denote the number of weights of the model, and *FF* is the number of neurons of the first linear layers of the MLP block. The first three models are variants of the encoder-only ViT [2], characterized by a different number of repeated transformer blocks, a different number of heads (H), and different values for the embedding and projection dimensions (E and P, respectively). The backbone of all three models is a series of repeated transformer blocks that include an MHSA followed by a feed-forward MLP. The input image is first divided into fixed-size patches by a convolutional layer; then, the patches are fed to the transformer network. The output of the final transformer layer is passed through a linear classifier head that maps the learned features to class predictions. For this model family, we consider the image/s as the crucial benchmarking metric, i.e., the number of classifications produced per second. We use batch size (B) = 1 for model execution.

The other two models, GPT-J and GPT3-XL, represent decoder-only LLMs. The two models are characterized by their large scale and capacity to handle a wide range of tasks, from generating human-like text to solving complex coding problems. Differently from the ViT variants, for these models, we consider two different execution modes: first, we report the results in **non-autoregressive (NAR) mode**, i.e., the mode used during the *prefill* stage, also known as prompt processing. In this mode, the model produces a number of outputs equal to the input length, processing S tokens (all readily available before starting) in a single forward pass. In generative inference applications, this step is necessary to prefill the so-called KV-cache, which is used as the initial context for the following output generation phase. Moreover, the computations involved in this operating mode are also similar to those of an LLM's *training forward pass* (with batch size = 1). The main difference is that, in our benchmarks, we do not explicitly exploit *causal masking*, as done in [10]. Causal masking is used during training to avoid information leakage from future tokens to the past, matching the inference-time scenario, where the model produces each new token, looking only at the past history. Note that ignoring causal masking makes our results conservative since its usage could lead to skipping computations, leading to additional speedups [10]. Nevertheless, we highlight that our models are functionally identical to those exploiting causal masking. We use this operating mode to obtain a fair comparison with SoA platforms.

The second mode considered for decoder-only models is the **autoregressive (AR) mode**, adopted during the actual *generative inference* phase, after the prompt processing. In this case, only a single next token is produced for each network invocation, using the previous part of the generated sequence (plus the previously processed prompt) as input. The primary metric used to benchmark decoder-only models for both modes is the number of tokens produced per second (tokens/s). For all models, we also use GFLOPS as a general comparison metric.

Notice that our work focuses on attention workload and the full execution of encoder/decoder architecture. However, we do not include ViT/LLMs frontends (Patcher/Tokenizer), which are necessary to execute a model end-to-end but whose latency is negligible compared to the rest of the architecture [54]. We

point the readers to their open-source implementation[1].

B. Simulation setup

We set the Snitch cluster at a frequency of 1 GHz in a 12 nm technology. We considered an implementation of our architecture, including 16 clusters and 16 GiB of HBM2E. The software kernels are compiled with a customized LLVM 12 toolchain for Snitch with -O3 optimizations. All performance results are obtained by cycle-accurate register-transfer level (RTL) simulations with QUESTASIM 2022.3. The access time to the external HBM memory is modeled using *DRAMSys* [x]. Thanks to extensive simulations with different access patterns and memory tiles, we obtained an average roundtrip access time of 88 ns [55]. To ensure realistic memory throughput, latency, and scheduling, we also conducted extensive dedicated simulations with tile dimensions employed in our networks. On average, we achieve a bandwidth with a four-cluster per group configuration of 56 B/cycle per cluster for both reads and writes. These measurements exclude on-chip interconnects that are simulated in RTL. We measured a DMA transfer setup time of 27 ns. Combining the results, we obtained a total static data transfer overhead of 115 ns per transfer, which is combined with DRAMsys simulation to obtain the latency of each single HBM-to-TCDM transfer.

We run power analyses over a placed-and-routed netlist of our architecture. We modeled the power consumption from the contributions of the system's major components, i.e., the Snitch cluster and its major sub-block (FPU, Snitch cores, scratchpad memory, ...), and the cluster-to-cluster interconnect, considering the FPU utilization to model the FPU power consumption correctly. Notice that our measurements led to comparable results (average 10.5% error) to the ones obtained on a scaled-up silicon implementation of the same open-source architecture [46] (normalizing their power numbers to take into account for the lower number of Snitch clusters of our architecture).

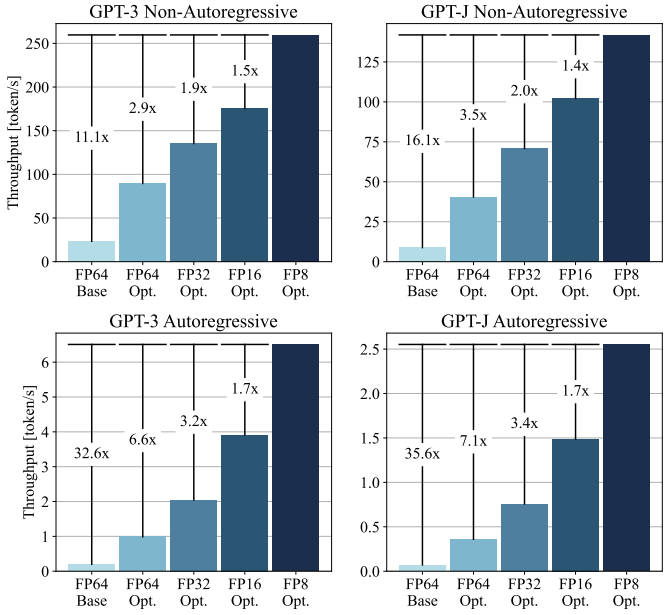


Fig. 7: Impact of SW optimizations on the throughput of the GPT-3XL and GPT-J models with $S = 1024$ in the NAR and AR modes.

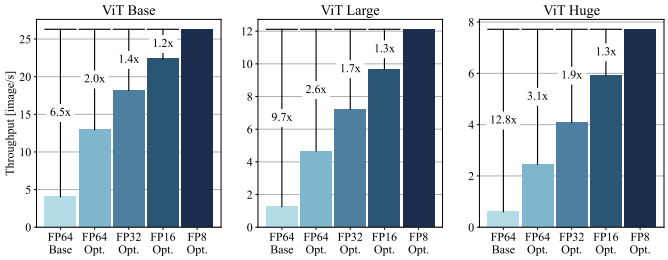


Fig. 8: Impact of SW optimizations on the throughput of the ViT model class.

VII. EXPERIMENTAL RESULTS

A. Impact of Software Optimizations

In Fig. 7 and Fig. 8, we show the impact of the optimizations defined in Sec. V. We evaluate the throughput of the GPT-3XL and GPT-J in terms of generated output tokens per second on both the AR and NAR operating modes. Furthermore, we investigate the impact of our optimizations on all three ViT models. The baseline implementation uses `FP64` precision and does not exploit the `Xssr` and `Xfrep` extensions. Furthermore, the cluster-to-cluster communication of the hierarchical interconnect is not exploited. By adding these features, we achieve an initial speedup of up to 5.0× for the GPT AR mode, 4.6× for the GPT NAR mode, and 4.1× for the ViTs. Then, on top of the `FP64` implementation, we show the additional impact of lower-precision formats that exploit SIMD parallelism. In particular, going to `FP32`, we observe an additional speedup of up to 1.8× (NAR), 2.1× (AR) for the GPT, and 1.6× for ViT models, respectively. Interestingly, for the GPT-J model, the improvement is even higher than the expected ideal one of 2× due to the SIMD only. This improvement is possible thanks to the lower memory occupation of the `FP32` tensors. The reduced memory footprint allows for more data to be loaded into TCDM. In turn, this leads to bigger TCDM tiles, which increase the overall data reuse, reducing the overall memory transfer time by more than 2×. Further, bigger tiles allow for better workload parallelization among the cores, leading to fewer idle cores. For instance, considering the parallelization of Fig.5, if M is lower than the cores' number, there will be idle ones. On the other hand, increasing M, each of the cluster cores will process data. Going to 16-bit floating-point, we observe an additional speedup of 1.4× (GPT NAR), 2× (GPT AR) and 1.5× (ViT). With an `FP8` data type, we achieve *overall* speedups of up to 16.1× for the GPT models in NAR mode, 35.6× in AR mode, and 17.9× for the ViT models. These correspond to 260/142 tokens/s in NAR mode and 6.5/2.6 tokens/s in AR mode for GPT3-XL and GPT-J, respectively. For the three ViT models, we achieve 26, 12, and 8 image/s, respectively, for Base, Large and Huge models.

[1]e.g., https://github.com/karpathy/llama2.c

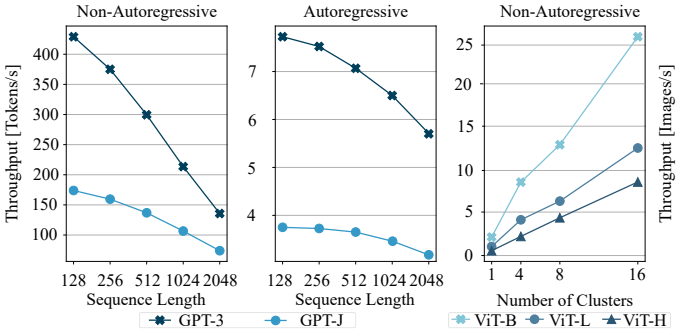


Fig. 9: On the left, the impact of scaling the sequence length in the GPT model class. On the right, scalability of the ViT models with increasing number of clusters.

B. Model & HW Scalability

Fig. 9 (leftmost two panes) shows the scaling of our performance with respect to the input sequence length S. For this experiment, we consider GPT3-XL and GPT-J in AR and NAR modes since ViTs are not normally fed with varying-length inputs. In NAR mode, changing S causes a quadratic increase in the computation complexity of the attention and a linear increase for the feed-forward layers. The number of tokens the network produces per invocation also varies (equal to S). Overall, we observe that the performance degrades with an almost constant slope proportional to the increase in operations, maintaining constant TFLOPS. Consequently, even when the sequence length increases, no overheads arise, such as expensive last-level memory spilling or lower tensor reuse. Increasing the sequence length from 128 to 2048, the performance of GPT3-XL decreases from 429 tokens/s to 136 tokens/s, and GPT-J from 174 tokens/s to 74 tokens/s.

In AR mode, the attention computation complexity grows linearly, while the linear layers' computation is independent of the sequence length, given that previous K and V elements are saved in the KV-cache. We observe this exact behavior when measuring individual operators' latencies. Overall, for full AR passes, performance ranges from 7.9 tokens/s to 5.8 tokens/s and from 3.8 tokens/s to 1 token/s for the GPT3-XL and the GPT-J models, respectively.

In both cases, GPT3-XL exhibits a more pronounced increase in processing time per unit of sequence length, indicating that the computational load, and hence, resource utilization, can be distributed more effectively for larger models like GPT-J.

Fig. 9 (rightmost pane) shows the images/s produced by the ViT models when increasing the number of clusters (and, therefore, of cores) in our scalable hardware platform. When the number of clusters is lower than the heads, we apply temporal tiling to this dimension. When the number of clusters is identical or higher, we exploit this dimension for inter-cluster parallelization as described in section V. Compared to one cluster, we obtain a speedup of $\{(4\times, 6\times, 12\times), (4\times, 6\times, 11.9\times), (4\times, 7.9\times, 15.8\times)\}$ for 4, 8, and 16 clusters on ViT-{B, L, H}, respectively. The progression of speedup nearly doubles with each step, which indicates a close-to-perfect scalability of our hardware and software stack. Namely, we are able to maintain similar hardware utilization despite the complexity of the additional inter-cluster communication.

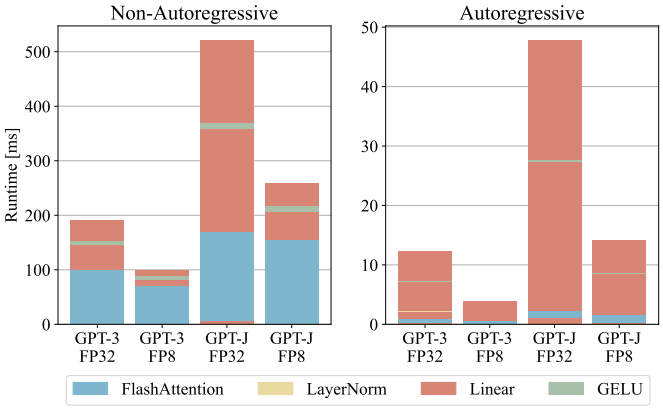


Fig. 10: `FP32` and `FP8` kernel breakdown latency analysis in NAR and AR modes.

C. Kernel Latency Breakdown

Figure 10 shows the latency breakdown for the GPT-J and GPT3-XL models in `FP32` and `FP8` precision. While the breakdowns differ for the two operating modes, the effect of precision reduction is identical. Overall, we observe a significant latency reduction from `FP32` to `FP8`, as also shown in Fig. 7. This result can be analyzed in more depth by looking at the three main latency contributors.

First, we observe that activation layers, i.e., Layernorm and GELU, have a limited latency impact, implying that they are not the primary bottlenecks within the model's architecture and that their scaling does not impact the overall layer latency. Most latency is spent in GEMM operations – for GPT-J, 66% in `FP32` and 36% in `FP8` of the total in NAR mode, 97% in `FP32`, and 89% in `FP8` in AR mode. Conversely, the last latency contributor, the FlashAttention-2 kernel, has a more pronounced relative impact on the `FP8` latency compared to `FP32`. This behavior is attributed to the Softmax exponential, still being executed in `FP32`, which does not fully capitalize on the latency reduction potential offered by `FP8`. Furthermore, we need unpacking and packing operations moving from `FP8` to `FP32` and vice-versa, which incurs additional latency in the `FP8` version of the FlashAttention-2 kernel. Executing the Softmax in `FP32` allows higher numerical stability and avoids quantization errors that could negatively impact the model's performance. Exploring the execution of the Softmax at lower precision is out of the scope of our work.

D. FPU Utilization & Power Measurements

Table III summarizes the power, FPU Utilization, and GFLOPS/W obtained by our FM library, considering the different precisions on the GPT-J model with a sequence length of 1024 in both NAR and AR modes.

The first thing to notice is the utilization of the AR mode. The AR mode consistently achieves $< 10\%$ utilization of the FPU. This low utilization is caused by the fact that only a single token is processed while the previous ones are loaded from the KV cache. This strongly reduces the overall latency but at the cost of low utilization of the computational units. On the other hand, when employing NAR mode, we consistently achieve

utilization higher than 65% when processing S output tokens in parallel. In particular, the wider the data type, the higher the utilization, given that the impact of non-arithmetic operations such as memory ones is lower. Furthermore, as previously mentioned in section V-A2, in FP8 and FP16 kernels, there are additional conversion operations to execute activation layers in FP32.

In terms of power consumption, we observe a power proportional to the FPU utilization, with a max of 5.2 W for the FP32 NAR attention block. In terms of efficiency, we reach the top performance of 294 GFLOPS/W with the FP8 kernel in NAR mode. Nevertheless, even in the AR mode, we can achieve the best performance of 65.5 GFLOPS/W thanks to the overall energy efficiency of the platform.

TABLE III: Power measurements and Efficiency on NAR and AR workloads in different precisions.

Mode	Implementation	Power [W]	GFLOPS/W	FPU Util [%]
NAR	FP64 Optimized	5.0	38.8	76.3
	FP32 Optimized	5.2	78.8	79.7
	FP16 Optimized	4.8	151	70.6
	FP8 Optimized	4.5	294	65.2
AR	FP64 Optimized	2.1	10.0	8.32
	FP32 Optimized	2.2	20.1	8.46
	FP16 Optimized	2.1	38.3	7.89
	FP8 Optimized	2.0	65.6	6.39

E. Comparison with State-of-the-Art

Table IV compares our fully open deployment flow and platform against SoA commercial accelerators. We use the NAR mode of the GPT class of models for this comparison to be compatible with SoA data. Emani et al. [56] collected data relative to the training forward pass (identical to our NAR mode) of GPT2-XL model running on many SoA accelerator platforms. All SoA platforms employ FP16. We also report our numbers in FP16 to have a fair comparison. Nonetheless, we can achieve even higher throughput by utilizing FP8 precision. Given that commercial accelerators have a much larger scale compared to our platform, we do not benchmark only the total throughput but also the throughput divided by the number of compute units (CUs), the FPU Utilization, computed as the ratio between the throughput achieved and the ideal maximum throughput of the platform, and the energy efficiency in terms of operation per time unit per W (GFLOPS/W).

TABLE IV: Comparison with SoA accelerators of the GPT NAR mode (GPT2-XL for SoA, GPT3-XL for our work) in FP16. SoA numbers are extracted from [56]. CU: Compute Unit

	A100	MI250	SN30	Gaudi2	Ours
Compute Units	6912 +432	13312 +208	1280	24 +2	128
Throughput (TFLOPS)	5.63	3.75	13.8	11.3	0.72
Throughput per CU (GFLOPS)	0.8	0.3	10.7	432.7	5.6
GFLOPS/W	18.7	7.5	-	23	**151**
FPU Util. [%]	14.4	7.8	16.0	34.6	**70.6**

TABLE V: Comparison with SoA academic accelerators.

	Model	Precision	#Params	Efficiency
Sparse & Integer Accelerators				
Wang et al. [40] ∼ 90% structured	GPT-2	INT12	774M	3.98 TOPS/W
Kim et al. [41] ∼ 40% pruning	GPT-2	INT8	774M	33.4 TOPS/W
FreFlex [42] ∼ 90% weight + act.	BERT-base	INT8	110M	0.75 TOPS/W
Ayaka [43] ∼ 90% act. + QK pruning	BERT-base	INT8	110M	49.7 TOPS/W
Floating-Point Accelerators (low- and mixed-precision)				
AccelTran [39] ∼ 50% weight + act.	BERT-Tiny	20b QFP	4M	0.012 TOPS/W
Tambe et al. [44]	BERT-base	FP4	110M	0.14 TFLOPS/W
Ours	GPT-J	FP8	6B	0.3 TFLOPS/W

Despite using general-purpose RISC-V cores with limited specialization for data-parallel compute patterns, our architecture achieves a throughput efficiency per compute unit comparable with the SoA of 0.0056 TFLOPS employing FP16 kernels. Only Gaudi2 and SN30 show impressively higher throughput; however, their CUs are either complex tensor processing units or matrix multiplication engines. When comparing FPU Utilization, our architecture achieves the best result of 70.6%, significantly outperforming SoA accelerators. For instance, the *A100* and *MI250* exhibit FPU utilization of 14.42% and 7.81%, respectively. Even the *Gaudi2*, which stands out among the compared platforms with an FPU utilization of 34.62%, reaches 2.04× lower utilization than ours. In terms of energy efficiency, again, the Gaudi 2 platform stands out as the best competitor, achieving 23 GFLOPS/W. At the same FP precision, we achieve 151 GFLOPS/W outperforming Gaudi 2 by 6.56×. We also compare our results to those of the two most powerful SoA competitors, Nvidia's *H100* GPU and the Gaudi 3 platform [37]. For H100 platform, while not having precise data on either AR or NAR execution for decoder class models, the *MLPerf* benchmarks [57] report a peak performance of 2683 samples/s on the (HF) ViT-large model in FP8 at 670W of power consumption, leading to an efficiency of 4 samples/s/W. Given the 17424 compute units (CUs) in H100, the throughput for CU corresponds to 0.15 samples/s/CU. In comparison, our architecture delivers a total of 27 samples/s for the same model in FP8 at a power consumption of 4.5 W, corresponding to 0.2 samples/s/CU and 6 samples/s/W, outperforming the *H100* by a factor of 1.3× in terms of throughput per CU and 1.5× in terms of efficiency. For the Gaudi 3 platform, while not having data on similar architectures to the ones we used, in [37], Intel

reports an average energy efficiency gain on three different decoder models (LLama-7B, LLama 70B, and Falcon-180B) of 1.4× compared to H100, being therefore comparable with our architecture.

On the other hand, given that academic accelerators run different workloads, such as sparse or dynamic transformers, and mostly use different data types, doing a direct fair comparison is not trivial. Table V provides a comparative analysis of SoA academic accelerators against our architecture. Accelerators that achieve a higher energy efficiency than our platform are characterized by deep modifications of the baseline models to leverage sparsity (up to 90%), integer quantization, and dynamic inference. For instance, the two most efficient accelerators, [41] and [43], aggressively leverage model sparsity, as well as adaptive inference through big-little architectures and QK pruning together with INT8 quantization to simplify the execution and improve efficiency. Needless to say all these architectures would require significant effort in model re-training and distillation. We can more directly compare with accelerators using FP (low bit-width) or closely related representations without enforcing model sparsity: AccelTran uses a not-so-aggressive 50% weight sparsity but employs a 20b fake quantized floating point data type to execute a BERT-tiny model, and [44], which executes a classical BERT model without deep model modification but using a very low bit-width FP4 data format. AccelTran performs inference within a power envelope of 24.04 W, reaching 0.28 TFLOPS as throughput. In NAR mode, with a sequence length of $S = 1024$, we operate in a power envelope of 4.5 W, reaching a throughput of 1.32 TFLOPS in FP8 format. Using almost the same number of bits, i.e., FP16, we still run in a power envelope of 4.8 W, with a throughput of 0.72 TFLOPS, therefore outperforming AccelTran by 2.6× in terms of throughput and by 12.6× in terms of energy efficiency.

The academic accelerator, developed by Tambe et al., utilizes 8-bit and 4-bit MP *floating-point* computation for inference. They test their accelerator on the 12-layer 12-head BERT-base encoder model, achieving a minimum latency of 489 ms with an efficiency of 0.14 TFLOPs/W (normalized to 1 GHz to run at the same frequency of our architecture). We compared this number with our `FP8` ViT-Base FP8 execution, considering that the models' parameters are almost identical (same H, Blocks, E, FF). On this model, despite the larger data type (FP8 vs FP4), we achieve 38 ms inference time, outperforming Tambe et al.'s work by a factor of 12.8× in terms of latency. In terms of efficiency, we outperform the accelerator by 2.1×.

VIII. Conclusion

In this work, we systematically explored the deployment of FMs on a multi-core scalable RISC-V platform. Our experimental analysis included three ViT models and two GPT models (GPT3-XL and GPT-J). We achieve superior FPU utilization compared to SoA commercial accelerators by tailoring our kernels to specific hardware resources, and memory interconnect characteristics. In particular, by implementing specialized kernels for the attention mechanisms inherent in FMs, we have shown speedups of over 35× compared to the baseline implementation. In our future work, we will explore additional SW and HW innovations to support FM execution, such as multi-chiplet scalability. We aim to create a more adaptable, efficient, and powerful flow encompassing SW and HW components that can handle the increasing complexity and size of emerging FMs in an open-source ecosystem.

Acknowledgments

This work has been supported in part by 'The European Pilot' project under grant agreement No 101034126, which receives funding from EuroHPC-JU as part of the EU Horizon 2020 research and innovation program. This work has also received funding from the Key Digital Technologies Joint Undertaking (KDT-JU) under grant agreements No 101095947 and No 101112274.

References

[1] W. X. Zhao *et al.*, "A Survey of Large Language Models," Nov. 2023, *arXiv preprint arXiv*:2303.18223.
[2] A. Dosovitskiy *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *arXiv preprint arXiv:2010.11929*, Oct. 2020.
[3] C. Zhou *et al.*, "A Comprehensive Survey on Pretrained Foundation Models: A History from BERT to ChatGPT," May 2023, *arXiv preprint arXiv*:2302.09419.
[4] A. Vaswani *et al.*, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.
[5] R. Bommasani *et al.*, "On the Opportunities and Risks of Foundation Models," Jul. 2022, *arXiv preprint arXiv*:2108.07258.
[6] T. Brown *et al.*, "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.
[7] OpenAI, "Whisper," https://openai.com/research/whisper, 2024, accessed: 2024-04-10.
[8] L. Ribar *et al.*, "SparQ Attention: Bandwidth-Efficient LLM Inference," Mar. 2024, *arXiv preprint arXiv*:2312.04985.
[9] A. Katharopoulos *et al.*, "Transformers are RNNs: Fast autoregressive transformers with linear attention," in *Proceedings of the 37th International Conference on Machine Learning*, ser. ICML'20, vol. 119. JMLR.org, Jul. 2020, pp. 5156–5165.
[10] T. Dao, "FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning," Jul. 2023, *arXiv preprint arXiv*:2307.08691.
[11] J. Choquette *et al.*, "NVIDIA A100 Tensor Core GPU: Performance and Innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, Mar. 2021.
[12] J. Choquette, "NVIDIA Hopper H100 GPU: Scaling Performance," *IEEE Micro*, vol. 43, no. 3, pp. 9–17, 2023.
[13] N. Dey *et al.*, "Cerebras-GPT: Open Compute-Optimal Language Models Trained on the Cerebras Wafer-Scale Cluster," Apr. 2023, *arXiv preprint arXiv*:2304.03208.
[14] D. Abts *et al.*, "A Software-Defined Tensor Streaming Multiprocessor for Large-Scale Machine Learning," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*. New York New York: ACM, Jun. 2022, pp. 567–580.
[15] T. Benz *et al.*, "A High-Performance, Energy-Efficient Modular DMA Engine Architecture," *IEEE Transactions on Computers*, vol. 73, no. 1, pp. 263–277, Jan. 2024.
[16] F. Zaruba *et al.*, "Snitch: A Tiny Pseudo Dual-Issue Processor for Area and Energy Efficient Execution of Floating-Point Intensive Workloads," *IEEE Transactions on Computers*, vol. 70, no. 11, pp. 1845–1860, Nov. 2021.
[17] D. Kalamkar *et al.*, "A Study of BFLOAT16 for Deep Learning Training," Jun. 2019, *arXiv preprint arXiv*:1905.12322.
[18] P. Micikevicius *et al.*, "FP8 Formats for Deep Learning," Sep. 2022, *arXiv preprint arXiv*:2209.05433.
[19] J. Devlin *et al.*, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein *et al.*, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186.

[20] H. Touvron *et al.*, "Llama 2: Open Foundation and Fine-Tuned Chat Models," Jul. 2023, *arXiv preprint arXiv*:2307.09288.
[21] S. Srivastava *et al.*, "Omnivec: Learning robust representations with cross modal sharing," in *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, 2024, pp. 1236–1248.
[22] T. Wu *et al.*, "A brief overview of chatgpt: The history, status quo and potential future development," *IEEE/CAA Journal of Automatica Sinica*, vol. 10, no. 5, pp. 1122–1136, 2023.
[23] T. Dettmers *et al.*, "GPT3.int8(): 8-bit Matrix Multiplication for Transformers at Scale," *Advances in Neural Information Processing Systems*, vol. 35, pp. 30 318–30 332, Dec. 2022.
[24] B. Wu *et al.*, "Visual Transformers: Token-based Image Representation and Processing for Computer Vision," Nov. 2020, *arXiv preprint arXiv*:2006.03677.
[25] H. Wu *et al.*, "CvT: Introducing Convolutions to Vision Transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 22–31.
[26] T. Kudo *et al.*, "Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," 2018. [Online]. Available: https://arxiv.org/abs/1808.06226
[27] R. Pope *et al.*, "Efficiently Scaling Transformer Inference," *Proceedings of Machine Learning and Systems*, vol. 5, Mar. 2023.
[28] Z. Liu *et al.*, "MobileLLM: Optimizing Sub-billion Parameter Language Models for On-Device Use Cases," Feb. 2024, *arXiv preprint arXiv*:2402.14905.
[29] X. Ma *et al.*, "Luna: Linear Unified Nested Attention," in *Advances in Neural Information Processing Systems*, vol. 34. Curran Associates, Inc., 2021, pp. 2441–2453.
[30] S. Wang *et al.*, "Linformer: Self-Attention with Linear Complexity," Jun. 2020, *arXiv preprint arXiv*:2006.04768.
[31] Z. Liu *et al.*, "Scissorhands: Exploiting the Persistence of Importance Hypothesis for LLM KV Cache Compression at Test Time," *Advances in Neural Information Processing Systems*, vol. 36, pp. 52 342–52 364, Dec. 2023.
[32] T. Dettmers *et al.*, "QLoRA: Efficient Finetuning of Quantized LLMs," *Advances in Neural Information Processing Systems*, vol. 36, pp. 10 088–10 115, Dec. 2023.
[33] "NVIDIA Data Center Deep Learning Product Performance AI Inference," https://developer.nvidia.com/deep-learning-performance-training-inference/ai-inference.
[34] S. Systems, "SambaNova DataScale® | The AI Platform for Innovation," https://sambanova.ai/products/datascale.
[35] S. Lie, "Cerebras Architecture Deep Dive: First Look Inside the Hardware/Software Co-Design for Deep Learning," *IEEE Micro*, vol. 43, no. 3, pp. 18–30, May 2023.
[36] "Intel Gaudi 2 neural network deep learning inference processor." [Online]. Available: https://habana.ai/products/gaudi2/
[37] "Intel Gaudi 3 AI Accelerator," https://www.intel.com/content/www/us/en/content-details/817486/intel-gaudi-3-ai-accelerator-white-paper.html.
[38] A. Smith *et al.*, "AMD Instinct™ MI200 Series Accelerator and Node Architectures," in *2022 IEEE Hot Chips 34 Symposium (HCS)*, Aug. 2022, pp. 1–23.
[39] S. Tuli *et al.*, "AccelTran: A Sparsity-Aware Accelerator for Dynamic Inference With Transformers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 4038–4051, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10120981/
[40] Y. Wang *et al.*, "An Energy-Efficient Transformer Processor Exploiting Dynamic Weak Relevances in Global Attention," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, pp. 227–242, Jan. 2023.
[41] S. Kim *et al.*, "20.5 C-Transformer: A 2.6-18.1μJ/Token Homogeneous DNN-Transformer/Spiking-Transformer Processor with Big-Little Network and Implicit Weight Generation for Large Language Models," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2024, pp. 368–370. [Online]. Available: https://ieeexplore.ieee.org/document/10454330/
[42] S. Shanmuga Sundaram *et al.*, "FreFlex: A High-Performance Processor for Convolution and Attention Computations via Sparsity-Adaptive Dynamic Frequency Boosting," *IEEE Journal of Solid-State Circuits*, vol. 59, no. 3, pp. 855–866, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10371341/
[43] Y. Qin *et al.*, "Ayaka: A Versatile Transformer Accelerator With Low-Rank Estimation and Heterogeneous Dataflow," *IEEE Journal of Solid-State Circuits*, pp. 1–15, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10530252/
[44] T. Tambe *et al.*, "22.9 A 12nm 18.1TFLOPs/W Sparse Transformer Processor with Entropy-Based Early Exit, Mixed-Precision Predication and Fine-Grained Power Management," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 342–344. [Online]. Available: https://ieeexplore.ieee.org/document/10067817/
[45] S. Knowles, "Graphcore," in *2021 IEEE Hot Chips 33 Symposium (HCS)*, Aug. 2021, pp. 1–25.
[46] G. Paulin *et al.*, "Occamy: A 432-core 28.1 dp-gflop/s/w 83% FPU utilization dual-chiplet, dual-HBM2E RISC-V-based accelerator for stencil and sparse linear algebra computations with 8-to-64-bit floating-point support in 12nm FinFET," *VLSI Symposium*, 2024.
[47] L. Bertaccini *et al.*, "MiniFloats on RISC-V Cores: ISA Extensions with Mixed-Precision Short Dot Products," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–16, 2024.
[48] G. Henry *et al.*, "Leveraging the bfloat16 artificial intelligence datatype for higher-precision computations," in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, 2019, pp. 69–76.
[49] N. Wang *et al.*, "Training deep neural networks with 8-bit floating point numbers," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 7686–7695.
[50] F. Zaruba *et al.*, "Manticore: A 4096-Core RISC-V Chiplet Architecture for Ultraefficient Floating-Point Computing," *IEEE Micro*, vol. 41, no. 2, pp. 36–42, Mar. 2021.
[51] "HBM2E," https://www.micron.com/products/memory/hbm/hbm2e.
[52] "Gtc: What's new? grace, grace hopper, infiniband, arm hpc software ecosystem," https://www.cmc.ca/wp-content/uploads/2023/05/Griffin-Lacey.pdf.
[53] S. Kim *et al.*, "I-BERT: Integer-only BERT quantization," *arXiv preprint arXiv:2101.01321*, 2021.
[54] Y. Li *et al.*, "Efficientformer: Vision transformers at mobilenet speed," in *Advances in Neural Information Processing Systems*, S. Koyejo *et al.*, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 12 934–12 949. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/5452ad8ee6ea6e7dc41db1cbd31ba0b8-Paper-Conference.pdf
[55] P. Scheffler *et al.*, "Sparse Stream Semantic Registers: A Lightweight ISA Extension Accelerating General Sparse Linear Algebra," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 12, pp. 3147–3161, Dec. 2023.
[56] M. Emani *et al.*, "A Comprehensive Performance Study of Large Language Models on Novel AI Accelerators," Oct. 2023, *arXiv preprint arXiv*:2310.04607.
[57] "Benchmarking Large Language Models on NVIDIA H100 GPUs with CoreWeave (Part 1)," https://www.databricks.com/blog/coreweave-nvidia-h100-part-1, Thu, 04/27/2023 - 09:09.

Viviane Potocnik received her BSc and MSc degree in Electrical Engineering and Information Technology from ETH Zurich in 2020 and 2022. She is currently pursuing a PhD in the Digital Circuits and Systems group of Prof. Benini. Her research focuses on heterogeneous architectures for energy-efficient multi-modal AI fusion and the exploration of innovative data representation strategies to enhance the computational efficiency and adaptability on devices at the extreme edge, ranging from high-performance to resource-constrained environments.

Luca Colagrande received his BSc degree from Politecnico di Milano in 2018 and his MSc degree from ETH Zurich in 2020. He is currently pursuing a PhD in the Digital Circuits and Systems group of Prof. Benini. His research focuses on the co-design of energy-efficient general-purpose manycore accelerators for machine learning and high-performance computing applications.

Tim Fischer received his BSc and MSc in "Electrical Engineering and Information Technology" from the Swiss Federal Institute of Technology Zurich (ETHZ), Switzerland, in 2018 and 2021, respectively. He is currently pursuing a Ph.D. degree at ETH Zurich in the Digital Circuits and Systems group led by Prof. Luca Benini. His research interests include scalable and energy-efficient interconnects for both on-chip and off-chip communication.

Luca Bertaccini received the M.Sc. degree in Electronic Engineering from the University of Bologna in 2020. He is currently pursuing a Ph.D. degree at ETH Zurich in the Digital Circuits and Systems group led by Prof. Luca Benini. His research interests include heterogeneous systems-on-chip, energy-efficient hardware accelerators, computer arithmetic, and transprecision computing.

Daniele Jahier Pagliari received the M.Sc. and Ph.D. degrees in computer engineering from the Politecnico di Torino, Turin, Italy, in 2014 and 2018, respectively. He is currently an Assistant Professor with the Politecnico di Torino. His research interests are in the computer-aided design and optimization of digital circuits and systems, with a particular focus on energy-efficiency aspects and on emerging applications, such as machine learning at the edge.

Alessio Burrello is currently a research assistant at Politecnico di Torino. He received his Ph.D. degrees in Electronic Engineering at the University of Bologna in 2023. His research interests include parallel programming models for embedded systems, machine and deep learning, hardware-oriented deep learning, and code optimization for multi-core systems. He has published over 90 papers in peer-reviewed international journals and conferences.

Luca Benini holds the chair of digital Circuits and systems at ETHZ and is Full Professor at the Universita di Bologna. He received his Ph.D. from Stanford University. Dr. Benini's research interests are in energy-efficient parallel computing systems, smart sensing micro-systems, and machine learning hardware. He is a Fellow of the IEEE, of the ACM, and a member of the Academia Europaea.