European Commission

FVLLMONTI

Call: **H2020-FETPROACT-2020-01**

Grant Agreement no. **101016776**

Deliverable D5.3 – Architecture library, multi-objective trade-offs and calibrated thermal models -V1

Start date of the project: 1st January 2021
Duration: 50 months
Project Coordinator: Cristell MANEUX - University of Bordeaux
Contact: Cristell MANEUX - cristell.maneux@ims-bordeaux.fr

FVLLMONTI

DOCUMENT CLASSIFICATION

Title	Architecture library, multi-objective trade-offs and calibrated thermal models - V1
Deliverable	D5.3
Estimated Delivery	31/10/2022 (M20+2)
Date of Delivery Foreseen	31/10/2022 (M20+2)
Actual Date of Delivery	31/10/2022 (M20+2)
Authors	Giovanni Ansaloni – P5 – EPFL, Alireza Amishahi – P5 – EPFL
Approver	Cristell Maneux (UBx)
Internal reviewers	Alberto Bosio (ECL), Jean-Luc Rouas (UBx)
Work package	WP5
Dissemination	PU
Version	V1.1
Doc ID Code	D5.3_FVLLMONTI_P5-EPFL-20221020
Keywords	Computer architectures, Full system simulation, Domain-specific acceleration

DOCUMENT HISTORY

VERSION	PUBLICATION DATE	CHANGE
0.1	05.08.2022	Initial version from EPFL (G. Ansaloni, A. Amirshahi)
0.2	09.08.2022	References, conclusion added (G. Ansaloni)
0.3	28.09.2022	Document sent to reviewers (G. Ansaloni)
1.0	11.10.2022	Feedback from internal review (J.L. Rouas)
1.1	20.10.2022	Feedback from internal review (A. Bosio)

DOCUMENT ABSTRACT

This document presents the progresses of the partners in WP5 (specifically, in T5.2, led by the partner EPFL), regarding the definition of the system-level evaluation framework for N2C2-accelerated computing platforms executing transformer applications. Our development effort resulted in the implementation of a novel tightly-coupled accelerator model, integrated in the gem5-X simulation environment. The accelerator can be flexibly characterized to accommodate the results from WP1-4. It can also be customized to leverage the application-level optimizations been explored in T5.3.

Achievements can be summarized as follows:
1 - identification of the computational kernels of transformer applications
2 - development of the behavioural model of the N2C2-based accelerator
3 - definition of the accelerator interface and of the required instruction set architecture extensions
4 - integration of the module in the gem5-X system simulation environment
5 - preliminary exploration of speed-ups of N2C2-accelerated systems over baseline solutions

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101016776.

TABLE OF CONTENT

LIST OF FIGURES AND TABLES

LIST OF ACRONYMS / GLOSSARY

AI:	Artificial Intelligence
ASR:	Automatic Speech Recognition
BERT:	Bidirectional Encoder Representations from Transformers
D:	Deliverable
FF:	Feed-Forward
FIFO:	First-In First-Out
FU:	Functional Unit
GEMM:	GEneral Matrix-to-matrix Multiplication
ISA:	Instruction Set Architecture
M:	Month of the project
MAC:	Multiply ACcumulate
MHA:	Multi-Head Attention
MT:	Machine Translation
N2C2:	Neural Network Compute Cube
PE:	Processing Elements
PU:	Public
SA:	Systolic Array
ST:	Speech Translation
ViT:	VisionTransformer
VNWFET:	Vertical Nanowire Field Effect Transistors
WP:	Work Package

1. Introduction

Transformers are state-of-the-art solutions in many Artificial Intelligence (AI) scenarios, including that of Speech Translation (ST) and Automatic Speech Recognition (ASR), targeted by the Fvllmonti project. However, the massive size and the large number of parameters of typical transformer implementations pose a computational challenge. Transformer architectures are composed by several layers, each embedding many large matrices of parameters. A typical transformer such as BERT-large [2] usually requires hundreds of millions of parameters. Such characteristics call for the hardware acceleration of inference in transformer models, in order to reduce their run-time and/or enable their execution in resource-constrained devices.

Hardware accelerators for transformers usually target GEneral Matrix to Matrix multiplication (GEMM), which dominates the run time of this class of applications, as shown in Section 4. In this context, Systolic Array (SA) architectures [20] are the focus of renewed research interest [4]. SAs enable parallel execution of GEMMs on a 2-dimensional mesh of processing elements, computing outputs in linear time. In the context of Fvllmonti, they are particularly attractive solutions, as their spatial parallelism can be well captured by the Neural Network Compute Cube (N2C2) architecture being designed in WP4, while their high computational intensity makes ideal test vehicles for disruptive technologies such as the Vertical Nanowire Field Effect Transistors (VNWFET) object of the research endeavors in WPs 1-3.

We herein describe our strategy to integrate N2C2-based accelerators in computing systems, and the full-system simulator environment developed for this purpose. Our approach is based on tight coupling of accelerators as extensions to the processor pipelines. Run-time execution is then governed by custom instructions, enriching the Instruction Set Architecture (ISA) of systems. Such stance has two main advantages. First, because accelerators are integrated into CPUs, they are frugal from a resource perspective, as they do not require dedicated scratchpads to store input, outputs or intermediate data. Second, they do not incur significant overheads for data transfers to/from the accelerators. On the contrary, tightly-coupled N2C2s can potentially leverage data-reuse optimization strategies performed at the system level, such as multi-tier tiling across the cache hierarchy. Finally, tight-coupling do not disrupt locality when transitioning from accelerated to non-accelerated computation segments.

To enable the quantitative assessment of these benefits, we implemented a parametric module emulating tightly-coupled N2C2 cells in the gem5-X full system simulation environment [17]. Preliminary results indicate run-time speedups exceeding 80X when executing BERT-Large Transformer model.

2. Background

I. TRANSFORMERS

Transformers are deep learning models composed of multiple blocks, where the outputs of each block are produced based on the inputs, weighted according to an "attention" significance metric. Attention values state the relevance of each input elements.

Early attention-based models derived attentions from an auxiliary channel. A major breakthrough in transformer performance was the introduction of "self-attention" by Vaswani et al. [19], which showcased that attention values can be effectively derived from the input data itself.

BERT [2] is a prototypical transformer model based on the self-attention concept and dedicated to language processing. The network consists of three different parts. First, an embedding layer translates a sequence of input tokens (e.g., words or syllables) into numerical values. Then, the main transformer functionality is implemented. This is composed of encoder transformer blocks. Depending on the BERT version, different numbers of blocks of identical size are employed. Finally, outputs are derived by an application-specific linear layer. For instance, for text classification, a simple representation reduction is applied. A similar structure is implemented in other transformer implementations, such as VisionTransformer (ViT) models [3], which target image interpretation tasks. In particular, both BERT and ViT employ multi-head attention (MHA) in encoder blocks to increase robustness.

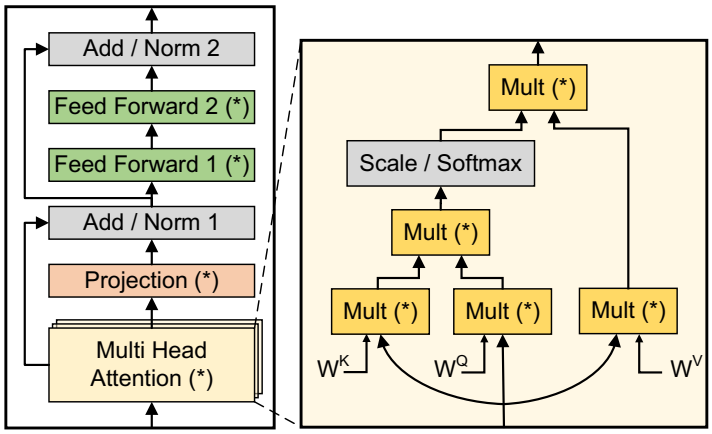

Figure 1 : Left: A representation of a transformer block. Right: A single head from the multi-head attention layer. Asterisks (*) denote layers whose execution is dominated by matrix-matrix multiplication.

Encoders dominate the workload of transformer-based models. The optimization of encoder blocks is, therefore, key from an application-wide perspective. Figure 1-left illustrates the encoder block structure. Their first component is devoted to the computation of MHA values. It applies the input matrix $X \in \mathbb{R}^{d_{seq} \times d_{model}}$ to h number of heads, where d_{seq} denotes the input sequence length, and d_{model} denotes the

vector length for each input in the sequence. The heads have identical dimensions, but because they employ different weights, they produce different values from the same input. Then, in the subsequent "Projection" layer, MHA outputs are concatenated and transformed to a lower dimension using a further rectangular weight matrix. The output of this layer has the dimensions of the encoder block input, so that both can be fed to the subsequent "Add & Norm" layer. As the name implies, in this stage the inputs of an encoder block are added to the Projection outputs, and the resulting values are normalized to have zero mean and unit variance. The final stages of the transformer block employ two position-wise Feed-Forward (FF) transformations to increase the dimension to d_{FF} and decrease it again to d_{model}. The FFs are followed by a further "Add & Norm" operation.

Figure 1-right shows the details of the computation of one head ($i \in \{0,..,h-1\}$) in an MHA layer. In it, the input X is multiplied with three weights matrices W_i^Q, W_i^K, W_i^V to obtain Q_i, K_i, V_i (named the Query, Key and Value matrices, respectively). The output of this single-head attention layer is then computed as follows:

$$H_i = Softmax\left(\frac{Q_i \times K_i^T}{\sqrt{d_q}}\right) \times V_i\,, \qquad i \in \{0,..,h-1\}$$

In the equation above, d_q is the dimension of the query, key, and value matrices. Then, the softmax layer scales matrix values in the range $[0, 1]$. H_i is the output of i-th head, which is the input (along with the output of all other attention heads) of the projection layer.

Stage	MHA	Projection	Add Norm 1	FF1	FF2	Add Norm 2
run-time (sec)	69.769738	32.482152	0.020244	429.476127	304.602903	0.020434

Table 1 : Run-time of the different stages of a BERT transformer block. Data gathered when executing on a 1GHz ARM CPU, with 32KB L1 data and instruction caches, and 1MB unified L2 cache.

As shown in Table 1, the dominant stages in transformer blocks (from a run-time perspective) implement GEMM operations: MHA, Projection and FF. In contrast, the non-GEMM Add-Norm stage only marginally contribute to run-time. In turn, GEMM-based layers can be effectively sped-up by the dedicated N2C2 cells, whose functionality and integration is detailed in the following.

II. SYSTOLIC ARRAYS

SAs are composed of sparsely-interconnected Processing Elements (PEs) which process an input stream to produce an output stream. Each PE embeds arithmetic units and storage [11]. Crucially, 2-dimensional SA grids can be specialized to spatially distribute the computation of GEMM algorithms [6].

SA designs for GEMM can stream the two input operands and have an output value being computed on each cell (output-stationary SA). Alternatively, one input may be stationary, while the other and the computed output values are streamed to/from the array (weight-stationary SA [1]). We focus on the latter choice, as it guarantees a better degree of data reuse for transformer applications when weights are considered as stationary inputs. In turn, a high degree of data reuse is key to coping with bandwidth constraints in tightly-coupled accelerators, as we discuss in detail in Section III.

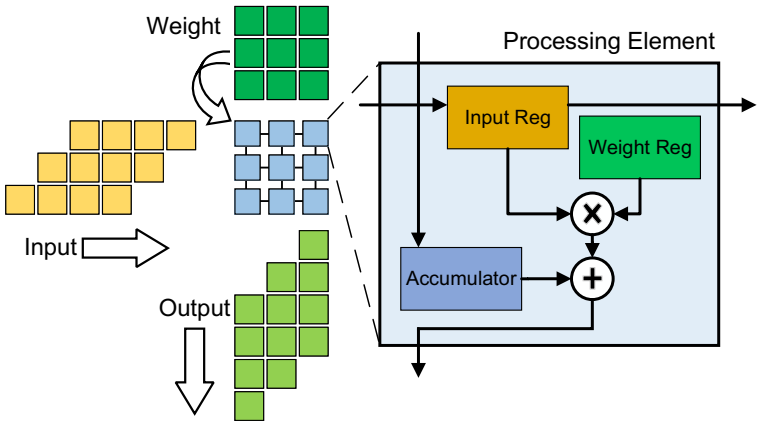

Figure 2 : Left: architecture of a 3x3 weight-stationary systolic array. Right: detailed view of the PE structure.

Figure 2 illustrates the structure of a 3x3 weight-stationary SA architecture. Input and outputs propagate in the array along two orthogonal directions (in the example, inputs stream left-to-right, outputs top-to-bottom). Weights are initialized before the start of computation and are resident in PEs. Then at every clock cycle, inputs are moved from one PE to the next unmodified, while outputs accumulate the results of the multiplication of inputs and weights. To produce a correct result, both inputs and outputs must be skewed along a diagonal. In our implementation, such skewing is performed with First-In First-Out (FIFO) queues of appropriate sizes that act as delay elements.

3. N2C2 as a tightly coupled Systolic Array

In this section, we describe how SAs can be effectively integrated into computing platforms as tightly-coupled accelerators, how their capabilities are exposed to software through ISA extensions, and how applications can effectively map matrix multiplications on available N2C2 resources.

I. N2C2 INTEGRATION

An example system featuring a N2C2 accelerator as a specialized Functional Unit (FU) is presented in Figure 3-left. From a resource requirements perspective, such an approach has the advantage that the SA, similarly to other FUs (e.g., devoted to integer, floating-point, or load/store operations), is interfaced to the cache hierarchy of the processor. Hence, there is no need for large dedicated scratchpads to host inputs and outputs. As discussed below, only shallow FIFOs are required to implement the proper data alignment.

A processor pipeline featuring N2C2 acceleration is depicted in Figure3-right. The N2C2 is accessed by employing dedicated instructions. Similar to instructions defining integer, floating-point, load/store, and other operations, custom instructions governing the SA are first fetched and then decoded. In the issue stage, the instruction operands are identified. Then, the N2C2-specific instructions are executed, directing operations on the systolic array. Finally, results are written back in the last stage of the pipeline.

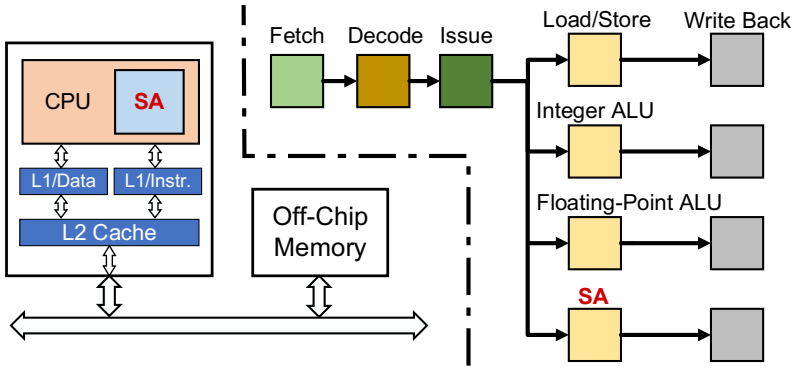

Figure 3 : Left: architecture of systems integrating a tightly-coupled N2C2 accelerator (indicated as a Systolic Array, SA). Right: SA as a custom functional unit embedded in a CPU pipeline.

II. N2C2 CUSTOM INSTRUCTIONS

In the developed implementation, N2C2 dedicated operations extend the ARMv8 ISA. They are composed of an operation code (*opcode*) field and data/address operands. Operands are assumed to be 32 bits.

Such bitwidth, while common in computing systems, is not required to represent data (both weights and intermediate values) in transformer models. Indeed, it is shown in the literature [7, 10, 13, 23] that 8-bit quantized transformer models incur a negligible accuracy drop. Therefore, to optimize run-time performance, we allow the transfer of four 8-bit data elements as different bit-fields of the same 32-bit register. Then, N2C2s with different multiple-of-four size (4*4, 8*8, etc.) can be parametrically defined. In the common case in which the N2C2 size exceeds 4*4, multiple instructions are required to complete the transfer of a matrix row. We hence use two different instructions to a) transfer four data items to/from the accelerator and b) transfer four data items to/from the accelerator *and* activate the MAC computations on the array. Figure 4 shows the behavior of these two instructions.

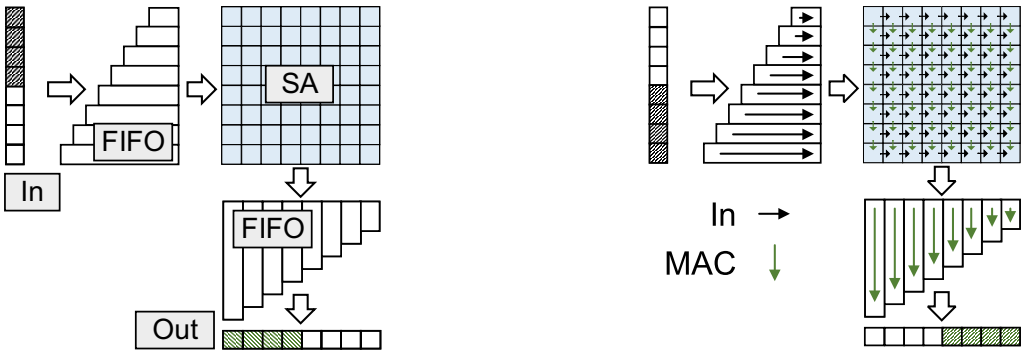

Figure 4 : Left: The *SA_IO* instruction enqueues four 8-bit values at the N2C2 input, and reads back four 8-bit values at its output. Right: In addition to performing I/O, *SA_IOC* also activates the FIFOs at the periphery of the SA and the PEs performing the computation of MACs in the array.

In more details, the custom instructions are defined as follows:

- ***SA_LD***. This instruction loads the weight inside PEs. It has three operands: two operands identify a PE cell by row and column address. The third operand is the weight value that should be stored in the PE. *SA_LD* transfers four weights at each invocation: that of the cell at the indicated row and column, and the three cells after it on the same row.
- ***SA_IOC***. This instruction is for Input, Output, and Computation (IOC) in the systolic array. It has two 32-bit operands. The first one dictates the input data, expressed as four concatenated 8-bit values. The second operand indicates in which position of the input row the values should be stored. When *SA_IOC* executes, the PEs inside the N2C2 array perform the computation of MAC operations in all columns, and inputs are propagated in all rows. Outputs are then produced at the bottom of the N2C2. The four 8-bit values corresponding to the position indicated as the second operand are forwarded to the writeback stage of the processor pipeline.
- ***SA_IO***. In contrast to the *SA_IOC*, in *SA_IO*, the N2C2 does not perform the computation because a row of inputs has not fully loaded. Similarly to *SA_IOC*, the instruction has two operands, determining the input data and the data position. In the same cycle, a 32-bit of output is read, again determined by the second instruction operand.

Note that in an N2C2 with 4*4 PEs, only the *SA_LD* and *SA_IOC* instructions are required. To this end, four *SA_LD* are used to program the weight values of the 16 registers, and then, the input data is streamed to the N2C2 using *SA_IOC* (at position '0'). The values read back by this operation correspond to four concatenated 8-bit N2C2 outputs.

In order to guarantee correctness, the input of weight stationary SAs must be skewed row-wise (see Figure 2). The proper input matrix shape could be arranged in software, i.e., by explicitly inserting '0' values in the upper-left region of the input matrix. This solution would unnecessarily increase memory requirements.

In our N2C2 model, as shown in Figure 4, we instead address this issue by employing row-wise FIFOs of increasing size at the SA periphery. Further column-wise FIFOs of appropriate size are used to enforce the correct alignment of output data. Data in FIFOs is advanced in response to an *SA_IOC* instruction.

III. SYSTEM SIMULATION

N2C2 instances are realized as gem5-X modules, allowing the evaluation of their benefits when accelerating transformer applications from a full-system perspective [17]. gem5-X is based on the popular gem5 framework [14], adding enhancements to support architectural extensions and advanced features such as guest/host shared spaced and fine-grained check-pointing.

We targeted systems based on the ARMv8 ISA, using gem5-X to extend the instruction set using unallocated opcodes, which we assigned to the N2C2 custom instructions. The behavioral model of accelerators, including the functionality of each defined instruction, is modeled in C++.

Applications can access N2C2s by inserting in-line assembly calls in their code. Three such code snippets are exemplified in Table 1. In the first column, two values are loaded in registers with a PE row and column index. Then, a third register is programmed with a weight value. Finally, the weight is transferred to the indexed PE using *SA_LD*. The second and third columns illustrate the use of *SA_IO* and *SA_IOC*, respectively. In both cases, data and position registers are set, the custom instruction is called, and the result is stored in memory. For convenience, in our implementation assembly code such as the one in Table 2 is encapsulated in a library

of higher-level functions performing the programming of weights and the streaming of data to/from the systolic array.

Load Weight	Input, Output	Input, Output, Compute
1. MOV R1, col 2. MOV R2, row 3. LW R3, $(w_addr) 4. SA_LD R1, R2, R3	1. MOV R1, col 2. LW R3, $(w_addr) 3. SA_IO R1, R3, R4 4. SW R4, $(out_addr)	1. MOV R1, col 2. LW R3, $(w_addr) 3. SA_IO R1, R3, R4 4. SW R4, $(out_addr)

Table 2 : Use of ISA extensions for governing N2C2 instances at run-time.

4. Preliminary results

As a first assessment of the benefit of employing the developed accelerator, we considered as a test vehicle a system with a single in-order core, defined in gem5-X. The system features 32 KB L1 instruction and data caches and a 1MB L2 cache. It also integrates an N2C2 accelerator with a size of 16*16 PEs. Performance was compared with that of an identical system, but without N2C2 acceleration.

We targeted as a benchmark a transformer block of the BERT-Large model, executing under Ubuntu Linux 16.04. The application characteristics are listed in the table below, where the listed parameter follow the definition provided in Section 2.

d_{seq}	d_{model}	d_{ff}	d_q	h	# weights
512	1024	4096	64	16	$340 * 10^6$

Table 3 : BERT-Large model parameters and number of weights.

As shown Figure 5, the use of an N2C2 of a moderate 16*16 size resulted in a very relevant speedup of 89.5X. Such encouraging result originate from multiple benefits deriving from the use of tightly-coupled systolic accelerators:

1. The presence of parallel hardware resources allows to speed-up computation, enabling the computation of a 16*16 tile of a GEMM in constant time.
2. The N2C2 internal register cache the weight values of a tile, reducing the pressure on the cache hierarchy.
3. The regular N2C2 structure results in tiled computation patterns, which leads to an increase in data reuse.

Ongoing research efforts are elaborating on these finding, in order to provide a detailed analysis of the contribution of the different factors outlined above, as well as to investigate a vast collection of benchmarks and target architectures.

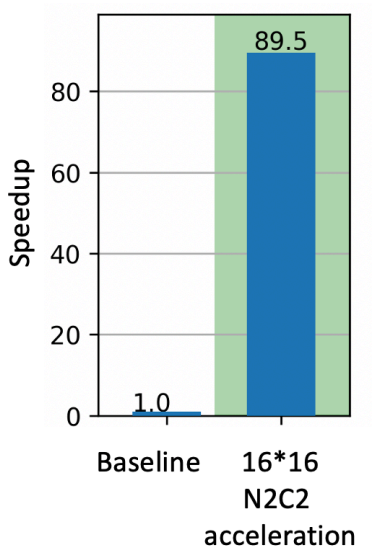

Figure 5 : Speed-up of the BERT-large encoder block executing with and without N2C2 acceleration.

The pie charts below report the breakdowns of the run-time along the different phases of the investigated transformer blocks. The right graph refers to the baseline system, while the right one to the accelerated one (again, considering a 16*16 N2C2). Crucially, it shows that, even for accelerated cases, GEMM is still the most intensive computational kernel, hinting at the scalability of our approach to even more performant accelerator, having a larger number of PEs, which we are investigating in collaboration with WP4.

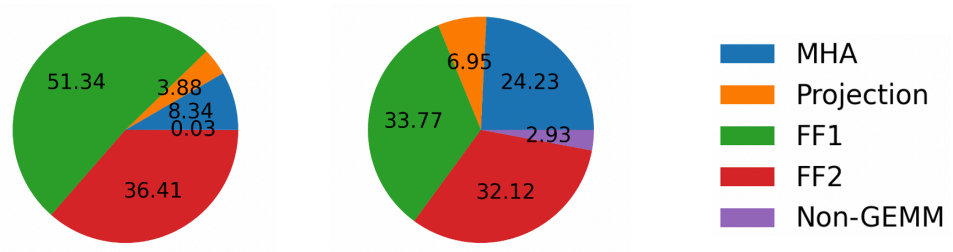

Figure 6 : Proportion of non-GEMM operations in a BERT-large block executing (a) as a non-optimized implementation and (b) accelerated by a 16*16 N2C2.

5. Related Works

The developed N2C2 system model is instrumental for evaluating the benefits of the technological advances sought in WP1-4 and the novel application optimization strategies in T5.3. Indeed, it serves as a necessary bridge between tasks performed at the technology and application levels within the Fvllmonti project.

In addition, the system-level simulation framework introduced in this deliverable D5.3 is an important research outcome in its own right, because the performed approach combining tight coupling and systolic acceleration, tailored to the acceleration of Transformer, tangibly innovates on the state of the art.

Indeed, few works explore systolic arrays for transformers. One of the earliest works on this topic is [15]. The paper implements a large SA accelerator using a hardware description language (HDL) and evaluates it on a Xilinx FPGA, considering a single transformer model. In [12], again, a systolic accelerator is implemented at the HDL level. However, these two works do not consider the integration of accelerators in computing systems.

Similarly to us, the work [18] proposes a systolic array accelerator simulated in a cycle-accurate platform for deep learning models, including transformers. However, the architecture in this work is designed in a loosely-coupled interface which hinders the accelerator from efficiently utilizing the cache hierarchy to maximize data reuse.

Our approach is related to the works in [4], which proposes the dual-core Gemmini platform: a loosely-coupled accelerator for deep learning models based on SAs. More than half of the area of Gemmini is dedicated to the scratchpads managing local data, and further resources are dedicated to the orchestration of data movements between memory and accelerator. Our tightly-coupled approach waives the need for these hardware elements, potentially resulting in highly energy- and area-efficient implementations.

Another loosely coupled accelerator for transformers is described in [21], which also introduces a hardware/software codesign to leverage acceleration opportunities. First, an algorithm is proposed to dynamically identify the most critical weights in the first layer of a transformer. Then, a hardware weight filtering unit is employed to recognize these weights at run-time. As in [4], a vast part of the area of their accelerator (58.6%) is employed to buffer input-output data, an overhead that we entirely avoid in our approach.

The authors of [9] employ Analog In-Memory Computing (AIMC) crossbars based on resistive memories to speed-up transformers. Their solution employs a multiplicity of crossbars interfaced to content addressable memories. A tightly-coupled solution for AIMC integration is introduced in ALPINE [8]. The authors of this paper focus on different applications with respect to us: multi-layer perceptrons, recurrent neural networks, and convolutional neural networks, where matrix-vector multiplications (as opposed to the GEMMs) are the main computational bottleneck.

Recently, a tightly-coupled accelerator for deep learning models, including transformers, has been introduced by [20], relying on custom extensions to the RISC-V instruction set. As opposed to ours, their approach is based on vector operations on SIMD units (instead of systolic arrays), which require explicit masking and moving operations among vector registers. The authors do not report achieved speed-ups on transformer benchmarks.

6. Conclusions and Future works

This deliverable presented the system architecture work carried out in the first phase of the project. In particular, it illustrates the developed transformer accelerator, motivating the rationale for its design and detailing its structure as well as the approach undertaken for its integration in overall computing systems.

Preliminary results highlight the potential benefit in accelerating the execution of transformers, while requiring a paucity of resources. We plan to confirm such preliminary results by exploring performance trends while varying the size of the systolic array, the capabilities of the overall system (operating frequencies, cache sizes, etc...) and the requirement of applications. In this regard, our exploration is following two complementary axes. On one side, we plan to investigate performance on entire transformer benchmark suites, to complement the results on BERT-Large presented in this deliverable. On the other, we will assess the effect of algorithmic optimizations presented in D5.5 in N2C2-accelerated systems.

Moreover, we plan to parametrize the N2C2 gem5-X module from an area, timing and area perspective, following up on the activities in WP1-4. Hence, our aim is to provide a hardware-proven realistic model based on the characterization of VNWFET gates (WP3), and their synthesis in the N2C2 HDL description (WP4). Our preliminary investigation resulted in a conference paper [22], accepted as a full research manuscript at the Asia and South Pacific Design Automation Conference 2023 (ASP-DAC23), where it will be presented in January 2023.

REFERENCES

[1] Bahar Asgari, Ramyad Hadidi, and Hyesoon Kim. 2020. *Proposing a fast and scalable systolic array for matrix multiplication.* In 2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE, 204–204.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. *Bert: Pre-training of deep bidirectional transformers for language understanding.* arXiv preprint arXiv:1810.04805 (2018).

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. *An image is worth 16x16 words: Transformers for image recognition at scale.* arXiv preprint arXiv:2010.11929 (2020).

[4] Hasan Genc, Seah Kim, Alon Amid, Ameer Haj-Ali, Vighnesh Iyer, Pranav Prakash, Jerry Zhao, Daniel Grubb, Harrison Liew, Howard Mao, et al. 2021. *Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration.* In 2021 58th ACM/IEEE Design Automation Conference (DAC). IEEE, 769–774.

[5] Gene H Golub and Charles F Van Loan. 2013. *Matrix computations.* JHU press.

[6] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. *In-datacenter performance analysis of a tensor processing unit.* In Proceedings of the 44th annual international symposium on computer architecture. 1–12.

[7] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2021. *I-bert: Integer-only bert quantization.* In International conference on machine learning. PMLR, 5506–5518.

[8] Joshua Alexander Harrison Klein, Irem Boybat, Yasir Mahmood Qureshi, Martino Dazzi, Alexandre Levisse, Giovanni Ansaloni, Marina Zapater Sancho, Abu Sebastian, and David Atienza Alonso. 2022. *ALPINE: Analog In-Memory Acceleration with Tight Processor Integration for Deep Learning.* arXiv preprint arXiv:2205.10042 (2022).

[9] Ann Franchesca Laguna, Mohammed Mehdi Sharifi, Arman Kazemi, Xunzhao Yin, Michael Niemier, and Sharon Hu. 2022. *Hardware-Software Co-Design of an In-Memory Transformer Network Accelerator.* Frontiers in Electronics 3 (2022).

[10] Yang Lin, Tianyu Zhang, Peiqin Sun, Zheng Li, and Shuchang Zhou. 2021. *FQ-ViT: Fully Quantized Vision Transformer without Retraining.* arXiv preprint arXiv:2111.13824 (2021).

[11] Richard J Lipton and Daniel Lopresti. 1985. *A systolic array for rapid string comparison. In Proceedings of the Chapel Hill Conference on VLSI.* Chapel Hill NC, 363–376.

[12] Zejian Liu, Gang Li, and Jian Cheng. 2021. *Hardware acceleration of fully quantized bert for efficient natural language processing.* In 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 513–516.

[13] Zhenhua Liu, Yunhe Wang, Kai Han, Wei Zhang, Siwei Ma, and Wen Gao. 2021. *Post-training quantization for vision transformer.* Advances in Neural Information Processing Systems 34 (2021).

[14] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann, Srikant Bharadwaj, et al. 2020. *The gem5 simulator: Version 20.0+.* arXiv preprint arXiv:2007.03152 (2020).

[15] Siyuan Lu, Meiqi Wang, Shuang Liang, Jun Lin, and Zhongfeng Wang. 2020. *Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer.* In 2020 IEEE 33rd International System-on-Chip Conference (SOCC). IEEE, 84–89.

[16] Patrice Quinton and Yves Robert. 1991. *Systolic algorithms & architectures.* Prentice Hall.

[17] Yasir Mahmood Qureshi, William Andrew Simon, Marina Zapater, David Atienza, and Katzalin Olcoz. 2019. *Gem5-X: A Gem5-based system level simulation framework to optimize many-core platforms.* In 2019 Spring Simulation Conference (SpringSim). IEEE, 1–12.

[18] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. *Scale-sim: Systolic CNN accelerator simulator.* arXiv preprint arXiv:1811.02883 (2018).

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. *Attention is all you need.* Advances in neural information processing systems 30 (2017).

[20] En-Yu Yang, Tianyu Jia, David Brooks, and Gu-Yeon Wei. 2021. *FlexACC: A Programmable Accelerator with Application-Specific ISA for Flexible Deep Neural Network Inference*. In 2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP). IEEE, 266–273.

[21] Zhe Zhou, Junlin Liu, Zhenyu Gu, and Guangyu Sun. 2022. *Energon: Towards Efficient Acceleration of Transformers Using Dynamic Sparse Attention.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2022).

[22] Alireza Amirshahi, Joshua Alexander Harrison Klein, Giovanni Ansaloni, David Atienza, 2023. *TiC-SAT: Tightly-coupled Systolic Accelerator for Transformers.* Accepted paper, to be presented at Asia and South Pacific Design Automation Conference (ASP-DAC 2023).

[23] L. Ben Letaifa and J.-L. Rouas, *Transformer Model Compression for End-to-End Speech Recognition on Mobile Devices*. In 2022 European Signal Processing Conference (EUSIPCO).