TiC-SAT: Tightly-coupled Systolic Accelerator for Transformers

Alireza Amirshahi, Joshua Alexander Harrison Klein, Giovanni Ansaloni, David Atienza
{alireza.amirshahi,joshua.klein,giovanni.ansaloni,david.atienza}@epfl.ch
Embedded Systems Laboratory (ESL), École Polytechnique Fédérale de Lausanne (EPFL)
Switzerland

ABSTRACT

Transformer models have achieved impressive results in various AI scenarios, ranging from vision to natural language processing. However, their computational complexity and their vast number of parameters hinder their implementations on resource-constrained platforms. Furthermore, while loosely-coupled hardware accelerators have been proposed in the literature, data transfer costs limit their speed-up potential. We address this challenge along two axes. First, we introduce tightly-coupled, small-scale systolic arrays (TiC-SATs), governed by dedicated ISA extensions, as dedicated functional units to speed up execution. Then, thanks to the tightly-coupled architecture, we employ software optimizations to maximize data reuse, thus lowering miss rates across cache hierarchies. Full system simulations across various BERT and Vision-Transformer models are employed to validate our strategy, resulting in substantial application-wide speed-ups (e.g., up to 89.5X for BERT-large). TiC-SAT is available as an open-source framework[1].

CCS CONCEPTS

• Computer systems organization → Neural networks; **Systolic arrays**; **• Computing methodologies → Natural language processing**; **• Hardware → Hardware-software codesign**.

KEYWORDS

Systolic Array, Tightly-coupled Accelerators, Transformers

ACM Reference Format:
Alireza Amirshahi, Joshua Alexander Harrison Klein, Giovanni Ansaloni, David Atienza. 2023. TiC-SAT: Tightly-coupled Systolic Accelerator for Transformers. In *28th Asia and South Pacific Design Automation Conference (ASPDAC '23), January 16–19, 2023, Tokyo, Japan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3566097.3567867

1 INTRODUCTION

Transformers, first conceived to tackle Natural Language Processing (NLP) tasks [19], are state-of-the-art solutions in many Artificial Intelligence (AI) scenarios. They now are used for question answering, sentiment analysis, image classification, clinical note analysis, and speech-to-text generation [1, 2, 5, 21].

[1]https://github.com/gem5-X/TiC-SAT

However, the massive size and the large number of parameters of typical transformer implementations pose a computational challenge. Transformer architectures are composed of several layers, each embedding many large matrices of parameters. A typical transformer such as BERT-large [1] usually requires hundreds of million 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. In this context, Systolic Array (SA) architectures [15] are the focus of renewed research interest [3]. SAs enable parallel execution of GEMMs on a 2-dimensional mesh of processing elements, computing outputs in linear time.

Recently, a number of research efforts have focused on devising effective strategies to integrate SAs into computing systems. Most (as discussed in Section 2) assume that SAs are interfaced on the system bus, employing scratchpad memories to store working sets [17]. We instead take a different approach, investigating the benefits of integrating small-scale functional units with dedicated SAs in the processor pipeline. We name such systolic arrays TiC-SATs: **Ti**ghtly-**C**oupled **S**ystolic array **A**ccelerators for **T**ransformers.

Our strategy has two main advantages. First, as TiC-SATs are integrated into CPUs, they are frugal from a resource perspective because they do not require dedicated scratchpads. Second, they do not incur significant overheads for data transfers to/from the accelerators. Contrarily, TiC-SATs can leverage data-reuse optimization strategies across the cache hierarchy. They also do not disrupt locality when transitioning from accelerated to non-accelerated computation segments.

Seeing the potential benefits of TiC-SAT, we implement it as a parametric module in the gem5-X full system simulation environment [16]. TiC-SAT instances are governed at run-time by custom instructions, which we define by extending the ARMv8 instruction set. We conduct comprehensive explorations to gauge the benefits of TiC-SAT across various SA sizes and transformer applications.

The contributions of our work are summarized as follows:

- We introduce a novel strategy for tightly coupling SAs as custom functional units governed by dedicated instructions.
- We showcase how SA accelerators can be integrated into computing systems, enabling full-system and application-wide explorations.
- We highlight how tight-coupling lightweight SAs can aptly exploit software optimizations that increase data locality to take advantage of available resources in cache hierarchies.
- We assess the benefit of small-scale, tightly-coupled SAs when accelerating inference in transformer models, considering different TiC-SAT sizes and benchmark applications.

2 RELATED WORKS

One of the earliest works investigating systolic array accelerators designed explicitly for transformers is [14]. The work implements a large SA accelerator using a hardware description language (HDL) and evaluates it on a Xilinx FPGA, considering a single transformer model. In [11], again, a systolic accelerator is implemented at the HDL level. In this hardware/software codesign work, the BERT model is fully quantized to 8 bits and 4 bits to reduce the latency and computation. However, these two works do not consider the integration of accelerators in computing systems.

Similarly to us, the work [17] 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 assuming a loosely-coupled interface which hinders the accelerator from utilizing the cache hierarchy.

The papers describing the Gemmini [3] and the SMAUG [20] platforms also introduce accelerators based on systolic arrays, adopting a full system simulation abstraction level akin to the one we consider in this paper. Nonetheless, Gemmini and SMAUG are integrated as loosely-coupled accelerators, requiring large scratchpad memories to store local data. Indeed, scratchpads account for 52.9% of the total area in Gemmini and 79.1% in SMAUG. Further resources are dedicated to the orchestration of data movements between memory and accelerator.

Another loosely-coupled accelerator for transformers is described in [22], 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 [3], 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.

3 BACKGROUND

3.1 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 a “self-attention" significance metric [19]. Self-attention values state the relevance of each input elements. BERT [1] is a highly successful family of transformer networks based on the self-attention concept, dedicated to language processing. BERT networks consist of three different parts. First, an embedding layer translates a sequence of input tokens (e.g., words or syllables) into numerical values. The second part of a BERT model implements the main transformer functionality, and is composed of encoder transformer blocks. Depending on the BERT version, different numbers of blocks of identical size are employed. Transformer outputs are derived by a final, application-specific linear layer. For instance, for text classification, a simple representation reduction is applied. A similar structure is implemented in VisionTransformer (ViT) models [2], which target image interpretation tasks. In particular, both BERT and ViT employ multi-head attention (MHA) in encoder blocks to increase robustness.

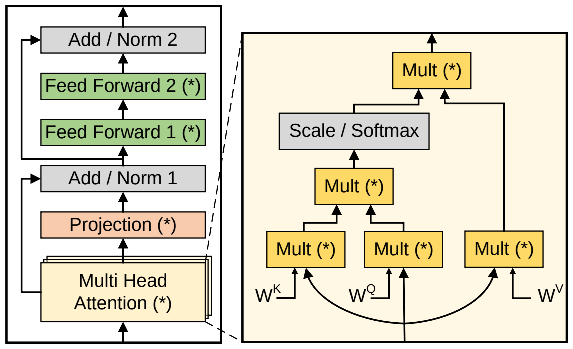


Figure 1: Left: A representation of a transformer block. Right: A single head from the multi-head attention layer. Asterisks (*) denote layers whose execution is dominated by matrix-matrix multiplication and hence are a target for acceleration.

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 block's structure. Their first component is devoted to the computation of multi-head attention (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 structures, 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 dimension 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 in an MHA layer. In it, the input X is multiplied with three weights matrices W_i^Q, W_i^K, and W_i^V to obtain Q_i, K_i, and V_i (named the Query, Key and Value matrices, respectively). The output of this single-head attention layer is then computed by applying non-linear softmax and scale functions and multiplying to V_i.

3.2 Systolic Arrays

SAs are composed of sparsely-interconnected Processing Elements (PEs) which process an input stream to produce an output stream.

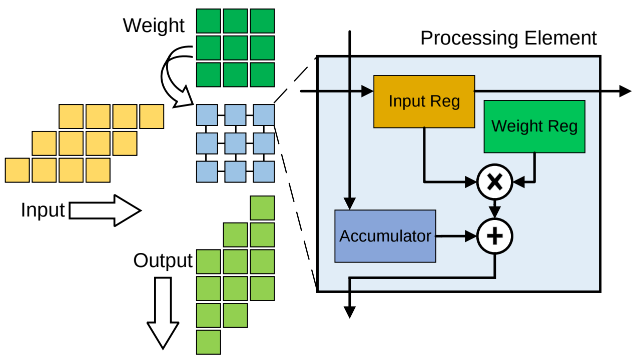


Figure 2: Left: architecture of a 3x3 weight-stationary systolic array. Right: detailed view of the PE structure.

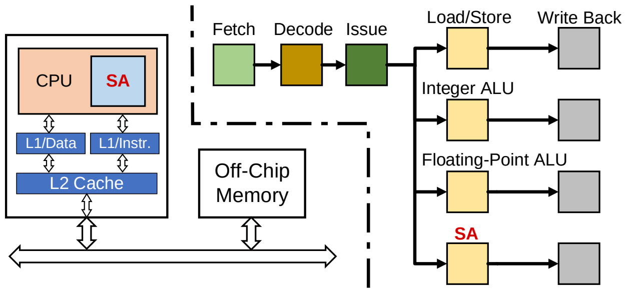


Figure 3: Left: architecture of systems integrating a tightly-coupled TiC-SAT accelerator. Right: TiC-SAT as a custom functional unit enriching a CPU pipeline.

Each PE embeds arithmetic units and storage. Recently, SAs have been the focus of renewed interest, as 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). 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 4.

Figure 2 illustrates the structure of a 3x3 weight-stationary SA architecture. In a weight stationary SA, input and outputs propagate in an array along two orthogonal directions (e.g., 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. Figure 2 shows that 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.

4 TIC-SAT: 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 Instruction Set Architecture (ISA) extensions, and how applications can effectively map matrix multiplications on available TiC-SAT resources.

4.1 TiC-SAT integration

An example system featuring a TiC-SAT accelerator as a specialized Functional Units (FUs) 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, no data movements across cache levels are required to provide inputs and retrieve outputs from/to the array. Moreover, data locality can be exploited even for computation patterns that involve both the SA and other functional units.

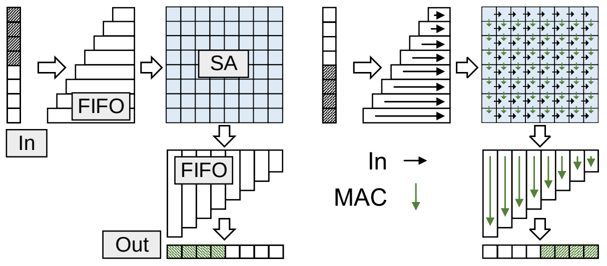
A processor pipeline featuring TiC-SAT is depicted in Figure 3-right. TiC-SAT is accessed by employing dedicated instructions. Similar to instructions defining integer, floating-point, load/store, and other operations, custom instructions governing the SA are first fetched and then decoded. In the latter stage, the instruction parameters are identified. In the case of TiC-SAT, the instructions have indirect addresses. Therefore, the effective address is read from memory in the decode step and set into a register. Then, the TiC-SAT-specific instructions are executed, directing operations on the systolic array. Finally, results are written back in the last stage of the pipeline.

4.2 TiC-SAT Custom Instructions

TiC-SAT-specific operations, extending an ARMv8 ISA, are composed of an operation code (opcode) field and data/address operands. During the instruction decode stage, the field of the instruction referring to the indirect address in the operand is set to a register. 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, 12] 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, SAs with different multiple-of-four size (4*4, 8*8, etc.) can be parametrically defined. In the common case in which the SA 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 these two instructions.

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



(a) SA_IO **(b) SA_IOC**

Figure 4: (a) The SA_IO instruction enqueues four 8-bit values at the TiC-SAT input, and reads back four 8-bit values at its output. b) 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.

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. As shown in Figure 4b, when SA_IOC executes, the PEs inside the TiC-SAT 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 SA. 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 SA 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.

Considering an SA with a size of k*k, the weight initialization requires $\frac{k^2}{4}$ SA_LD operations because, in every operation, four 8-bit weights are transferred via a 32-bit register. The streaming in/out of an SA row employs $\frac{k}{4}$ operations, from which $\frac{k}{4} - 1$ are SA_IO to fill the input buffer and read the output buffer, and one operation is SA_IOC to perform the computation step as well as the in/out streaming. Since the input is set and output is read in the same cycle, the most performant configuration is when the kernel dimensions are equal, i.e., for square SAs. Indeed, additional SA_IO instructions must be issued solely for loading inputs if the input dimension exceeds the output dimension. Dually, SA_IO instructions which solely read output values, are required if the SAs have more columns than rows.

In order to guarantee correctness, the input of weight stationary SAs must be skewed row-wise (see Figure 4). 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 TiC-SAT, 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.

4.3 Executing large GEMMs in constrained TiC-SATs

Transformers rely on very large matrices, which cannot be executed as-are in a realistically-sized TiC-SAT. Hence, our approach relies on the tiling [4] of input, weight, and output matrices to divide them according to the SA dimension. Tiling has been shown to be highly beneficial for GEMM even in the absence of hardware acceleration [10] because, by increasing data locality, it can minimize cache miss rates and the consequent clock cycles penalties. Indeed, a hierarchy of tiles of increasing dimensions can be adapted to conform to different cache levels. Herein, we introduce a further hierarchical tiling layer, targeting the maximization of data reuse for TiC-SAT systems.

If we consider a system comprising of one TiC-SAT of size k*k, and an L1 cache, our strategy divides data into three sub-matrices (containing inputs, outputs, and weights, respectively), which can fit in the L1 cache. Sub-matrices are further divided into tiles of dimensions equal to that of the SA. At run-time, tiles perform iterative computations inside sub-matrices, and sub-matrices iterate over the working set of a GEMM computation.

Algorithm 1 details the inner loop of such run-time behaviour, i.e., the iteration of tiles inside the sub-matrices. In the algorithm, the sizes for the input, weight, and output sub-matrices are indicated as $R^I_{L1} \times C^I_{L1}$, $R^W_{L1} \times C^W_{L1}$, and $R^I_{L1} \times C^W_{L1}$, respectively. Note that the column size of a sub-matrix in the input must be equal to the row size of the weight matrix, hence $C^I_{L1} = R^W_{L1}$. Line 1 and 2 iterate over the weight sub-matrix in steps of the tile dimensions k.

Algorithm 1 L1 cache-optimized management of TiC-SAT acceleration

```
Input: $W_{L1}$: Sub-matrix of size $R^W_{L1} \times C^W_{L1}$ in the weight matrix
Input: $I_{L1}$: Sub-matrix of size $R^I_{L1} \times C^I_{L1}$ in the input matrix
Output: $O_{L1}$: Sub-matrix of size $R^I_{L1} \times C^W_{L1}$ in the output matrix
1: for $m$ from 1 to $R^W_{L1}$ by $k$ do
2:     for $n$ from 1 to $C^W_{L1}$ by $k$ do
3:         Initialize the tile $W_{m,n} \in \mathbb{R}^{k \times k}$ into SA (SA_LD)
4:         for $p$ from 1 to $R^I_{L1}$ by $k$ do
5:             Stream the tile $I_{p,m} \in \mathbb{R}^{k \times k}$ to SA, and
               Obtain the tile $O_{p,n} \in \mathbb{R}^{k \times k}$ from SA
               (SA_IO and SA_IOC)
6:         end for
7:     end for
8: end for
```

Line 3 performs the weight initialization in the SA for a tile with size k*k. The innermost loop of the algorithm (line 4) reports the iteration along the rows of a tile. We choose this loop ordering to minimize the number of weight initialization (SA_LD) operations.

In Line 5, the input tile is streamed to the SA, and the output is read out.

We empirically choose the sub-matrix sizes as follows:

$$R^I_{L1} > C^I_{L1}, R^I_{L1} > C^W_{L1}$$
$$R^I_{L1} \times C^I_{L1} + C^I_{L1} \times C^W_{L1} + R^I_{L1} \times C^W_{L1} < \textbf{L1} \text{ size}$$

The rationale for this choice lies in the minimization of the overhead for loading weights in the weight-stationary SA, which makes a slightly larger R^I_{L1} desirable. However, the total sub-matrix sizes cannot exceed the L1 cache size. Since the cache space available for sub-matrices does not correspond to the entire cache, the sum of the tile size should be slightly less than the cache size.

4.4 System simulation

TiC-SAT instances are realized as gem5-X modules, allowing the evaluation of their benefits when accelerating transformer applications from a full-system perspective [16]. We employed gem5-X to explore various performance indicators besides run-time, including hits/misses in different levels of the memory hierarchy. gem5-X is based on the popular gem5 framework [13], 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 TiC-SAT custom instructions. The behavioural model of the SA, including the functionality of each defined instruction, is modeled in C++.

Applications can access TiC-SATs 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 1 is encapsulated in a library of higher-level functions performing the programming of weights and the streaming of data to/from the systolic array.

Table 1: Use of ISA extensions for TiC-SAT.

Load Weight	Input, Output	Input, Output, Compute
`MOV R1, col`	`MOV R1, col`	`MOV R1, col`
`MOV R2, row`	`LW R3, $(in_addr)`	`LW R3, $(in_addr)`
`LW R3, $(w_addr)`	`SA_IO R1, R3, R4`	`SA_IOC R1, R3, R4`
`SA_LD R1, R2, R3`	`SW R4, $(out_addr)`	`SW R4, $(out_addr)`

5 EXPERIMENTS AND RESULTS

5.1 Target system and transformer applications

We investigate the run-time performance of our TiC-SAT systems considering multiple transformer applications belonging to the BERT and VisionTransformers (ViT) families. We adopt the different sizes mentioned in [2] for VisionTransformers and in [18] for the BERT models. Benchmarks, therefore, range from very small implementations (BERT-tiny) having 4 million parameters, up to ViT-huge, which requires 632 million parameters.

Experiments are carried out on a system with in-order CPU core clocked at 1 GHz, 32 KB of L1 data and instruction cache and 1MB of L2 cache size. The main memory is a 4GB DDR4. The considered baseline (non-TiC-SAT) solutions are:

- *Non-optimized models (N.O.)*: Conventional matrix to matrix multiplication implemented as a triple-nested loop iterating on the input rows, weight columns, and input columns.
- *Optimized cache utilization (C-Opt.)*: In this baseline, the input, weight, and output matrices are divided into sub-matrices that fit inside the L1 cache memory.

TiC-SAT systems embed SAs of varying k*k sizes. We constrained our exploration to small-scale arrays that can realistically be integrated as tightly-coupled processors. We synthesize TiC-SAT using Synopsys Design Compiler with TSMC 28 *nm* library and a 1*n* clock constraint. The design shows that a 16*16 SA consumes 38 516μm^2, and the FIFOs implemented for data alignment in input and output require 3 031μm^2 and 2 836μm^2, respectively. The total cell area for TiC-SAT is only 44 384μm^2. On the other hand, in Gemmini [3], as a loosely-coupled accelerator, the components with 22 *nm* technology consume much more area, namely, 858 000μm^2, which is 19.3x larger than TiC-SAT. Also, in [20], the SMAUG accelerator consumes an area of 1.44 mm^2 characterized with 16 *nm* FinFET technology. The dedicated area for this accelerator is 32.5x times larger than TiC-SAT.

5.2 Run-time analysis of TiC-SAT-enabled systems

Figure 5 showcases the run-time required to execute a block (as shown in Figure 1) of the BERT-large model on the system. Results are shown for different baseline implementations, as well as for systems featuring TiC-SATs of increasing size. In each case, stacked bars detail the execution time for every layer inside the transformer block. Figure 5 also displays the corresponding speed-ups with respect to the non-optimized baseline.

The graphs highlight that the software optimization enhancing data locality in caches is very effective. TiC-SAT tightly-coupled accelerators can effectively speed-up software-optimized baselines. When considering a small 4*4 SA, an additional speed-up exceeding 2X (with respect to *C-Opt.*) is obtained. Further reductions in run-time are achieved when increasing the array size.

This figure also shows that the most time-consuming layers in a transformer block are the feed-forward (FF) ones. These layers almost entirely consist of GEMM operations, and hence can be accelerated by employing TiC-SATs. As mentioned in Section 3, GEMMs are not nonetheless exclusive to feed-forward layers. Indeed, the only layers that do not comprise GEMMs are “Scale/Softmax" and “Add/Norm". Non-GEMM layers are not, however, the computational bottlenecks of transformers, as shown in Figure 6. This consideration stands even for TiC-SAT-accelerated systems, where they account for only 3.1% of the execution time at most.

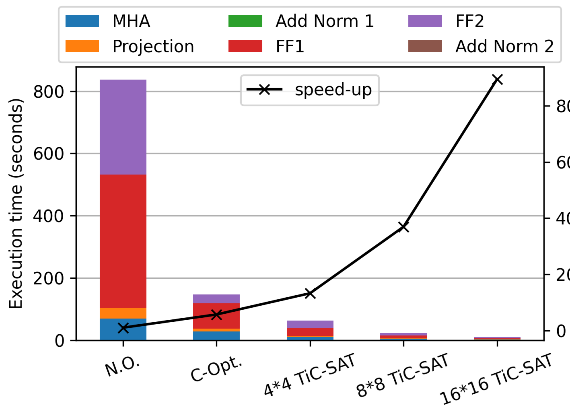
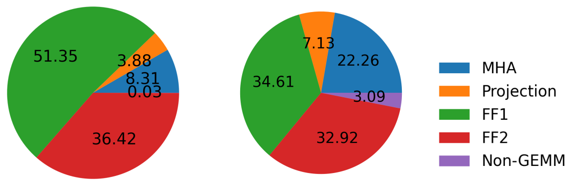


Figure 5: The run-time and speed-up of the BERT-large encoder block executing with and without TiC-SAT accelerator.



(a) N.O. baseline **(b) TiC-SAT**

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 TiC-SAT.

5.3 Impact of TiC-SAT acceleration on data locality

TiC-SAT increases performance both by parallelizing computation, and by increasing the locality of memory accesses, hence lowering the burden on the cache hierarchy. We herein focus on the latter aspect, again investigating the use-case of BERT-large.

Figure 7a shows the number of accesses in different memory hierarchy levels for different baselines and TiC-SAT-based instances. As shown in this figure and as expected, L2 cache (and main memory) accesses are reduced by applying software optimization. Such benefit comes at the cost of an increase in L1 instruction accesses, as tiling introduces a more complex loop structure, hence more instructions are executed.

When the TiC-SAT is used to execute the transformer block, accesses across the memory hierarchy are drastically decreased (Figure 7a). The reason for this behaviour is two-fold: first, a single SA_IOC instruction triggers k*k MAC operations, hence lowering the number of required instructions. Second, k*k parameters are resident during computation in the SA, and must not be re-read at each use, thus reducing the size of the working set. Both effects become more prominent for larger TiC-SAT sizes.

Figure 7b and 7c show the number of read and write accesses to main memory in different layers of the BERT-large model. Results highlight that the large matrices employed in the feed-forward layers result in a considerable amount of replacements and evictions of L2-cache lines. Such trashing effect is countered by C-Opt. Further reductions in read accesses are observed when employing TiC-SAT, due to the reduction in the instruction count of the application, as discussed above.

Table 2: Transformer benchmark characteristics and speed-up of 16*16 TiC-SAT with respect to the N.O. baseline.

Model	d_{seq}	d_{model}	h	params	speed-up
ViT-base/16	197	768	12	86M	69.4
ViT-base/32	50	768	12	86M	48.8
ViT-large/16	197	1024	16	307M	82.5
ViT-large/32	50	1024	16	307M	57.2
ViT-huge/14	257	1280	16	632M	82.7
BERT-tiny	512	128	2	4M	20.3
BERT-mini	512	256	4	11M	38.2
BERT-medium	512	512	8	41M	58.3
BERT-base	512	768	12	110M	69.3
BERT-large	512	1024	16	340M	89.5

5.4 TiC-SAT run-time on different benchmarks

Table 2 reports the benchmark characteristics and the speed-up in ten different transformer applications. The speed-up refers to the executions of the entire block with the Non-optimized baselines and ones where a 16*16 TiC-SAT is employed.

Results show that double-digit (up to 89.5X) speed-ups are achieved in all applications. Such outcomes originate from a combination of software optimization and hardware acceleration. They highlight the benefit of employing tightly-coupled acceleration, as it can effectively leverage data reuse in the memory hierarchy, in the presence of both accelerated as well as non-accelerated layers.

5.5 Comparison with a loosely-coupled implementation

We compare our TiC-SAT tightly-coupled approached with the SMAUG loosely-coupled accelerator [20]. Both implementations extend the gem5 system simulation framework [13], which allow to perform a fair comparison. As shown in Table 3, we considered the same baseline system in terms of clock frequency, cache size, and CPU type. Then, we performed an iso-area comparison, targeting a 64*64 TiC-SAT and a SMAUG system with 3x8 KB scratchpads. Notice that this setting results in a slightly bigger TiC-SAT array, but also that SMAUG models a smaller technology node (28*nm* vs. 16*nm*). Simulations on a BERT-Medium feedforward layer highlight that both accelerators can effectively speed-up the computation, but that TiC-SAT outperforms SMAUG by more than 2.5X.

6 CONCLUSION

In this work, we have showcased TiC-SAT, an architecture and framework for tightly-coupled systolic arrays devoted to accelerating transformer applications. We designed a model for systolic array acceleration in the gem5-X full system simulator, and defined

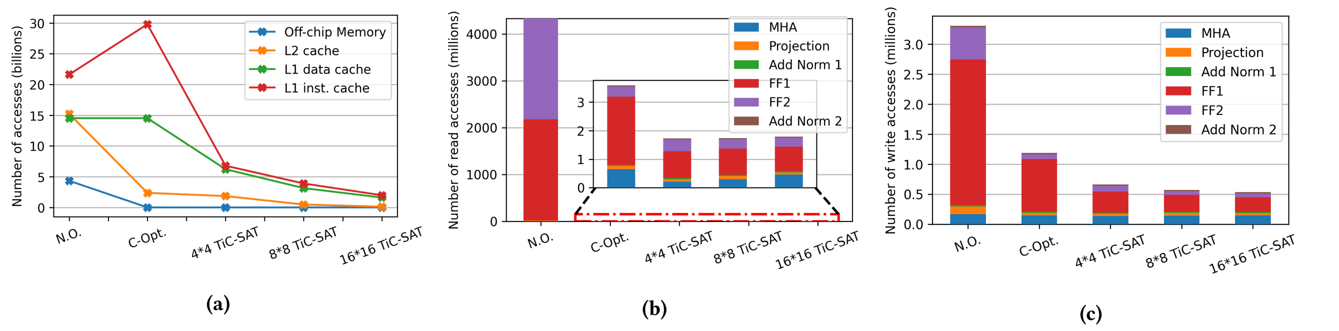


Figure 7: a) Memory accesses at different levels of cache and memory. b) Read accesses and c) write accesses to main memory for different transformer layers.

Table 3: Comparison of TiC-SAT and SMAUG [20] performance

System	64*64 TiC-SAT	SMAUG [20]
CPU Frequency	1 GHz	1 GHz
L1 Cache Size	32 KB	32 KB
L2 Cache Size	1 MB	1 MB
CPU type	out-of-order	out-of-order
Data type	Int8	Float16
Technology	28nm	16nm
Area (mm^2)	0.70	0.61
Speed-up wrt baseline	234x	88x

its interface with custom extensions to the ARMv8 instruction set. Our approach overcomes traditional data transaction bottlenecks associated with other accelerated systems for transformers. Therefore, targeting various BERT and VisionTransformer models, we tested embedding systolic arrays of varying dimensions defined in gem5-X. We examined the speed-ups and memory behaviour relative to the reference and optimized applications. Our exploration of transformer architectures and computer organizations have demonstrated substantial speed-ups, including up to 89.5X for the BERT-large transformer using a 16*16 TiC-SAT accelerator.

ACKNOWLEDGMENTS

This work has been supported by the EC H2020 WiPLASH (GA No. 863337) and the EC H2020 FVLLMONTI (GA No. 101016776) projects.

REFERENCES

[1] 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).
[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).
[3] Hasan Genc, Seah Kim, Alon Amid, Ameer Haj-Ali, 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.
[4] Gene H Golub and Charles F Van Loan. 2013. *Matrix computations.* JHU press.
[5] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. 2021. Hubert: Self-supervised speech representation learning by masked prediction of hidden units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 29 (2021), 3451–3460.
[6] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, 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, et al. 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] Monica D Lam, Edward E Rothberg, and Michael E Wolf. 1991. The cache performance and optimizations of blocked algorithms. *ACM SIGOPS Operating Systems Review* 25, Special Issue (1991), 63–74.
[11] 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.
[12] 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).
[13] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, et al. 2020. The gem5 simulator: Version 20.0+. *arXiv preprint arXiv:2007.03152* (2020).
[14] 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.
[15] Patrice Quinton and Yves Robert. 1991. *Systolic algorithms & architectures.* Prentice Hall.
[16] 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.
[17] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. Scale-sim: Systolic cnn accelerator simulator. *arXiv preprint arXiv:1811.02883* (2018).
[18] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962* (2019).
[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, et al. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
[20] Sam Xi, Yuan Yao, Kshitij Bhardwaj, Paul Whatmough, Gu-Yeon Wei, and David Brooks. 2020. SMAUG: End-to-End Full-Stack Simulation Infrastructure for Deep Learning Workloads. *ACM Transactions on Architecture and Code Optimization* 17 (2020), 1–26.
[21] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems* 32 (2019).
[22] 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).