BrainTTA: A 28.6 TOPS/W Compiler Programmable Transport-Triggered NN SoC

Maarten J. Molendijk[*], Floran A.M. de Putter[*], Manil Dev Gomony[*], Pekka Jääskeläinen[§], Henk Corporaal[*]
[*]Electrical Engineering department, Eindhoven University of Technology, the Netherlands
{m.j.molendijk, f.a.m.d.putter, m.gomony, h.corporaal}@tue.nl
[§] Faculty of Information Technology and Communication Sciences, Tampere University, Finland
pekka.jaaskelainen@tuni.fi

***Abstract*—Accelerators designed for deep neural network (DNN) inference with extremely low operand widths, down to 1-bit, have become popular due to their ability to significantly reduce energy consumption during inference. This paper introduces a compiler-programmable flexible System-on-Chip (SoC) with mixed-precision support. This SoC is based on a Transport-Triggered Architecture (TTA) that facilitates efficient implementation of DNN workloads. By shifting the complexity of data movement from the hardware scheduler to the exposed-datapath compiler, DNN workloads can be implemented in an energy efficient yet flexible way. The architecture is fully supported by a compiler and can be programmed using C/C++/OpenCL. The SoC is implemented using 22nm FDX technology and achieves a peak energy efficiency of 28.6/14.9/2.47 TOPS/W for binary, ternary, and 8-bit precision, respectively, while delivering a throughput of 614/307/77 GOPS. Compared to state-of-the-art (SotA), this work achieves up to 3.3x better energy efficiency compared to other programmable solutions.**

I. Introduction

Edge computing is a rising computing paradigm with the ability to overcome privacy, latency, and energy issues that are currently being faced in the deployment of neural networks (NNs) on embedded devices. While modern neural networks can solve complex tasks in fields such as Computer Vision (CV) and Natural Language Processing (NLP), the sheer size of these networks prevents them from being deployed directly on embedded devices, which typically have limited storage capacity. Moreover, the compute power also interferes with the deployment of such networks due to the energy constraints typically imposed on embedded hardware. To overcome these challenges, several approaches have been explored that optimize models for low-power hardware. These include Hardware-aware Neural Architecture Search (NAS) [1], model compression in the form of pruning [2], quantization [3] and exploiting efficient data reuse [4].

In parallel, research has been performed on creating highly specialized accelerators for neural network inference, exploiting the aforementioned model compression techniques. While these architectures perform great in terms of energy efficiency, the datapath structure is often not flexible and programmability is limited to some assembly dialect if programmable at all. Contrary, programmable solutions for neural network inference have been proposed; although the flexibility is better, the overhead cost due to data movement degrades the energy efficiency of these solutions.

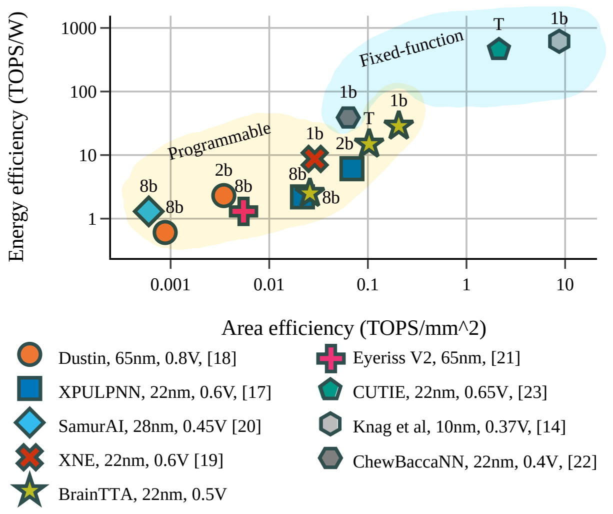


Fig. 1: Energy efficiency vs. area efficiency of programmable and fixed-function works.

In summary, current efforts towards low-power accelerators suffer from limited flexibility to easily and efficiently support different layers with varying sizes and varying parameter precision, or incur a large overhead to retain programmability. In this paper, we present BrainTTA, the first high-level programmable neural network SoC that features an exposed-datapath architecture [5] with mixed-precision support. We showcase the flexibility of this architecture and the energy trade-off when running layers with different bit-widths. The contributions of this paper are threefold:

- **BrainTTA**: an energy-efficient SoC for neural network inference. BrainTTA supports multiple precisions (binary, ternary and 8-bit) and utilizes its transport-triggered architecture [5] to reduce data movement. BrainTTA achieves an energy efficiency of 28.6/14.9/2.47 TOPS/W for binary, ternary, and 8-bit operands, respectively. (Section III & IV).
- **Energy consumption analysis**: a thorough analysis of the system's energy consumption for various operand bit-widths. We demonstrate that leveraging the TTA approach enables energy-efficient scaling to low-precision operands (Section V).

- **Comparison to State-of-the-Art**: We compare BrainTTA to state-of-the-art programmable architectures that focus on optimizing neural network inference for quantized neural networks. BrainTTA demonstrates up to 3.3x improvement in energy efficiency compared to other programmable solutions. Fig. 1 summarizes how BrainTTA compares to SotA for different operand precisions in area and energy efficiency (see also Section VI & VII).

The remainder of this paper is organized as follows. Section II discusses the background knowledge. Afterwards, Section III provides an overview of the full system architecture. The mapping of the network onto the proposed architecture is described in Section IV. The results are presented in Section V, followed by a comparison with respect to state-of-the-art architectures in Section VI and VII. Finally, Section VIII concludes the paper.

II. Background Information

To relieve the burden on the memory and reduce the arithmetic hardware complexity, quantization can be applied. Quantization can be applied down to 8-bit without significant loss of accuracy [6]. However, even with 8-bit quantization, the storage requirements of modern neural networks are not in line with the storage size typically found in embedded hardware. Therefore, a push towards even lower bit-width quantization was made.

A. Binary and Ternary Quantization

Binary quantization restricts the weights and activations to a single bit; therefore the weights and activations are $w, a \in \{-1, +1\}$ whereas ternary *trits* can additionally represent zero. This low operand precision introduces several advantages: the memory footprint is drastically reduced, the computations can be simplified, and the required bandwidth decreases sharply [7][8]. When both weights and activations are binarized or ternarized, the computations can be simplified by replacing the `MAC` (Multiply-Accumulate) operation with `XNOR` and `popcount` for binary and `Gated-XNOR` and `popcount` operations for ternary omitting the need for expensive multiplication operations.

While these extreme forms of quantization introduce significant energy and area savings, there is no such thing as a free lunch. There can be a significant gap in accuracy between 8-bit quantized networks and binary/ternary variants [3][9]. Furthermore, some layers are more resilient to quantization than others [10]. This varying quantization penalty motivates the use of an architecture that supports mixed-precision.

B. Transport-Triggered Architecture

Transport-Triggered Architecture (TTA) [5] is a kind of VLIW processor, however, it is programmed by specifying data movements instead of arithmetic operations. This means that the movement of data is exposed to the compiler; the TTA is an explicit-datapath architecture, this enables new optimizations. DNN workloads are heavily data-centric. The cost of retrieving data increases with every level in the hierarchy (RF $\rightarrow$ SRAM $\rightarrow$ DRAM) and should be avoided whenever possible. With the data path exposed to the compiler, optimization like Register File (RF) bypassing and dead result elimination can be applied such that the pressure on the RF is decreased allowing buffering of more DNN feature maps and weights or decreasing the area occupied by the RF. The explicit-datapath architecture essentially moves complexity from the hardware domain (scheduler) to the software domain (compiler) while retaining full flexibility. The downside is that the compiler complexity is increased, but this is circumvented by reusing existing frameworks, as discussed in Section III.

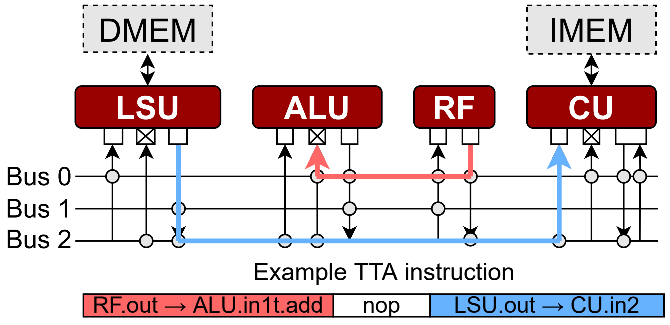


Fig. 2: An example TTA instance and instruction, the square blocks denote *input-* and *output-ports*. A cross denotes a *trigger-port*. The colored arrows drawn on the architecture illustrate the *moves* inside the example instruction. This TTA is a 3-issue processor, i.e., it handles 3 moves per cycle.

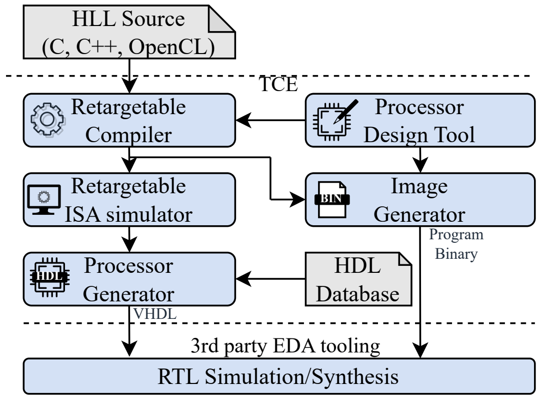
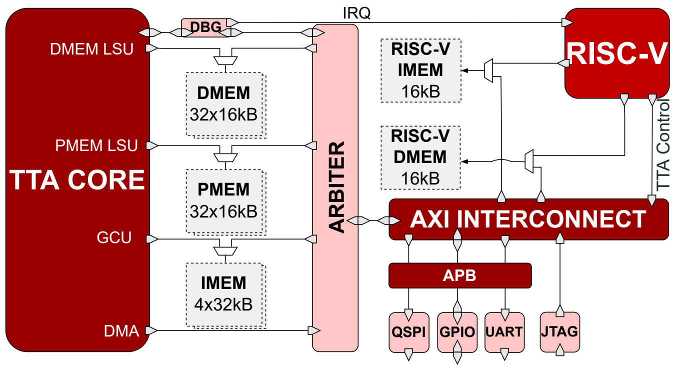
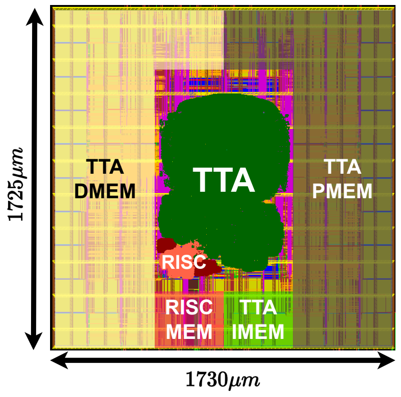


Fig. 3: TCE toolfow that is used to create the BrainTTA instantiation and compile the workload from a high-level language (HLL).

A basic exemplary instance of a TTA is displayed in Fig. 2. The TTA contains a Control Unit (CU) used for instruction fetching and decoding, RFs for temporary storage, and Load-Store Units (LSUs) to access the memories. The connectivity (represented by gray circles) is design-time configurable and can be tailored to be generic or application-specific as desired; more connectivity is at the expense of larger instruction size and more switching activity in the interconnect. In [11], several ways to mitigate the cost of increased connectivity by reducing



(a) Block diagram of the BrainTTA SoC, the arbiter forms the border between the RISC and TTA part of the SoC.



(b) Layout (excluding pads); RISC logic and TTA logic are highlighted.

Fig. 4: Overview of the BrainTTA SoC. The acronyms PMEM, DMEM, and IMEM stand for Parameter Memory, Data Memory, and Instruction Memory, respectively.

the instruction overhead, such as instruction compression are presented.

III. Architectural overview

The proposed full system architecture is displayed in Fig. 4 and the TTA core in Fig. 5. In this paper, the TTA-based Co-design Environment (TCE) [12] is used to create the TTA instance.

This is an open-source toolchain that provides full compiler support. An overview of the main components of the TCE framework is shown in Fig. 3. It allows creation of any Functional Unit (FU) with arbitrary functionality and number of input and output ports using the processor design tool. The FUs can implement custom instructions that can be added into the TCE compiler; the instructions can be inferred by the compiler or directly called using intrinsics. The compiled program can then be simulated using the ISA simulator which can provide useful statistics to optimize the function units and their connectivity. Finally, to deploy the TTA design, HDL code can be generated together with the binary images to be loaded into the TTA memory.

The main system components (as shown in Fig. 4) are:

- *RISC-V host processor* [13], to start/stop the execution on the TTA core, initialize the memories and send and receive information via the external interfaces.
- *TTA core*, used to perform the mixed-precision inference, more details will follow in the next paragraph.
- *SRAM*, with separate memories for the RISC and TTA core. Memories are banked to allow efficient access of smaller bit-widths.
- *Debugger (DBG)*, can halt the execution on the TTA core and signal the completion of a task to the RISC-V host.
- *AXI interconnect*, used for on-chip communication between the RISC and TTA-core and interfaces with the peripherals (APB) for off-chip communication.

At the heart of the SoC is the TTA core. This core is used for neural network inference and is based on a TTA as explained in Section II-B. The instantiation of the TTA core used in this paper is shown in Fig. 5. It contains different FUs. They are interconnected via buses, with scalar buses (buses 0-5) and vector buses (buses 6-11). The core consists of the following main units:

Control Unit (CU); it contains the logic to fetch and decode instructions and steers the other units to execute the correct operations. Furthermore, the CU contains a hardware loop-buffer. Since the network layers are essentially described by multiple nested loops (listing 1), having a hardware loopbuffer can greatly cut-down instruction fetch costs.

Vector Multiply-Accumulate (vMAC/vTMAC/vBMAC) unit is the workhorse of our BrainTTA. `MAC` operations are performed for 8-bit, ternary, and binary operands. The unit multiplies two 1024-bit vectors with 32 entries of 32-bits each. Thus the vMAC contains 32 reduction trees each 32-bit wide, where each tree has four 8-bit, 16 ternary, or 32 binary inputs.

The number of inputs to the adder tree grows for smaller bit widths, this is needed to amortize the accumulator cost with respect to the multiplier cost; since the multiplication operation is cheaper for smaller bit-widths, the adder tree needs to become wider [14]. To reduce switching activity, the inputs of the unused MAC units are gated to save power. Input data reuse is exploited by broadcasting the input feature map to multiple units with different weights. The fixed vector width implies that for each bit-width, different vectorization factors are applied; this is further explained in Section IV-B.

Vector Add (vADD) is used to add two (either 512- or 1024-bit) vectors, this can be used to support residual layers. The vector width that is supported corresponds to the output bit-widths of the MAC operation as shown in Section IV-A. For instance, when performing a MAC operation on a convolutional layer using 8-bit operands, the output of the MAC unit will have 32 entries of 32-bit each, thus resulting in a 1024-bit wide vector.

Vector Operations (vOPS) performs the non-arithmetic operations. This FU can additionally perform quantization,

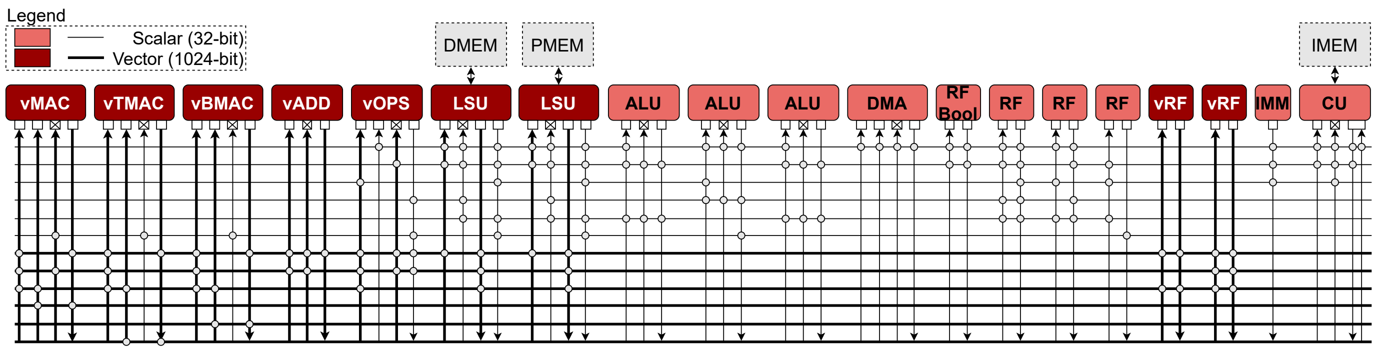


Fig. 5: BrainTTA core; thicker lines denote 1024-bit vector buses, thinner lines 32-bit scalar buses.

apply activation functions, e.g. `ReLU`, and pooling functions such as max pooling. This FU also supports scalar element insertion and extraction on vectors.

(Vector) Register Files (vRFs/RFs) to store intermediate values or buffer weights to increase data reuse.

Load-Store Units (LSUs) are the interface to the SRAM. There are two LSUs, one mainly used to load weights (from PMEM) and the other to load and store feature maps (DMEM).

Scalar Units are primarily used for address calculations that are needed as input to the LSUs, and for scalar code.

```
for h in [0, H - R + 1]:                          Output feature map height
  for w in [0, W - S + 1]:                         Output feature map width
    for tm in [0, M/32]:                          Ouput channels (v_M = 32)
      acc = bias[tm]
      for tc in [0, C/4]:                          Input channels (v_C = 4)
        for r in [0, R]:                                      Kernel height
          for s in [0, S]:                                     Kernel width
            acc += in_buffer[h + r][w + s][c] *
                   weights[n][r][s][tm]
      out_buffer[h][w][tm] = acc
```

Listing 1: An example of a convolutional layer with an output-stationary schedule, where v_C and v_M are the vectorization factors used for the input- and output-channels.

IV. Application Mapping

The different layers in a neural network can generally be described in terms of nested for-loops, see listing 1 for an example. Since applications for the TTA can be programmed in e.g. C, the schedule can easily be altered for each layer separately. Changing the schedule simply boils down to applying loop transformations (e.g. unroll, interchange, tile). This scheduling freedom, in combination with the exposed-datapath operating principle, allows the creation of an efficient schedule on a per-layer basis with minimized data movements.

A. Layer Support

Different neural networks constitute different layer types. Among the layer types that are supported in BrainTTA are:

1) Convolutional layer	(8b in, 32b out)
2) Binary convolutional layer	(1b in, 16b out)
3) Ternary convolutional layer	(2b in, 16b out)
4) Depth-wise convolutional layer	(8b in, 32b out)
5) Fully-connected layer	(8b in, 32b out)
6) Residual addition	(16/32b in, 16/32b out)
7) Requantization	(to 8b, 2b or 1b)

An energy breakdown of the first three layers in the list above is given in Section V.

Below some details about this layer support:

Convolutional layers are supported with three different bit-widths: 8-bit, ternary and binary. Depending on the bit-width of the convolutional layer, the 1024-bit weight and input vector are split in different ways (more information in Section IV-B). Since different output feature maps use the same input feature maps, input broadcasting is possible for data reuse.

Depth-wise convolutional layers are supported by changing the scalar-vector product used in convolutional layers to vector-vector products which is required since each weight kernel is bound to a single input channel; in other words, input broadcasting is not possible.

Fully-connected layers execution is similar to that of convolutional layers, however, the kernel size is now 1x1.

Residual addition is adding two higher bit-width values, but to support this, the scaling factor of the values added together needs to match. The latter is called **requantization**.

To reduce the overhead that comes with the flexibility and programmability of BrainTTA, parallelism is introduced in several scheduling dimensions (i.e. dimensions that are shown in listing 1); in the next paragraph, the choice of vectorization dimensions to achieve this parallelism will be elaborated.

B. Vectorization

Vectorization is visualized in Fig. 6. The choice of vectorization dimensions is based on three observations. First, modern networks typically have more feature maps (higher C, M) but the size of each individual feature map is smaller (lower W, H) [15][16]. This means that to populate a large vector, one should not restrict vectorization to W and H. Secondly, the `MAC`, binary and ternary `popcount` operations produce an intermediate output value with a much higher bit-width than the quantized value; requantization should happen as soon as possible to reduce movement of large intermediate values. Therefore, the final value of a single pixel in the

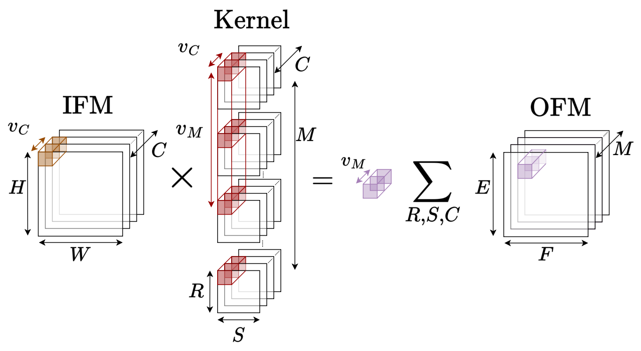
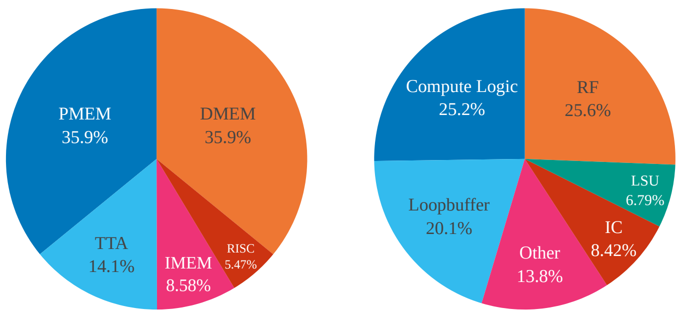


Fig. 6: A convolutional layer where the Input Feature Map (IFM), Kernel and Output Feature Map (OFM) vectorization is visualized; v_C is vectorization over the IFM channel dimension, v_M over the OFM channel dimension.



(a) SoC area breakdown. **(b)** TTA core area breakdown.

Fig. 7: BrainTTA area breakdown.

output feature map should be calculated as early as possible, which makes an output-stationary schedule favorable. Lastly, the `popcount` outputs the sum of its inputs. Therefore, the inputs supplied to the popcount module should contribute to the same output pixel. This means that the inputs supplied to a single popcount module (which corresponds to a single 32-bit vector element), should either have different *W&H* indices in the same receptive field or from a different input channel *C*.

V. Experimental setup and Results

The design shown in Fig. 4a is synthesized using GlobalFoundries 22nm FDX technology using an operating voltage of 0.5V while targeting the typical corner. After synthesis, the layout is created, as shown in Fig. 4b.

A. Experimental setup

The flow that is used to go from RTL to layout consists of Cadence Genus 21.10 for the logic synthesis and Cadence Innovus 21.11 for the back-end implementation. These tools are also used to obtain energy numbers. The energy numbers are obtained by annotating the switching activity found during the post-layout simulation in order to gain the most accurate energy figures possible.

B. Post-layout simulation results

The area of the SoC is 2.98mm^2 excluding IO pads as can be seen in Fig. 4b. The largest part of the floorplan is dedicated to the data (DMEM) and parameter (PMEM) memory of the TTA core, holding the input/output feature map and weights respectively; the breakdown of the on-chip area between the logic and memories is depicted in Fig. 7a. Both DMEM and PMEM are made by combining 32 16kB banks, resulting in a combined data storage capacity of 1MB. A breakdown of the area usage in the TTA core is shown in Fig. 7b, a large part of the area is used for instructions (loopbuffer) and for weight and activation storage (RF). Being memory dominated in both the overall area ($> 80\%$), as well as the TTA core ($> 45\%$), implies that BrainTTA has a disadvantage on the area efficiency metric. However, it therefore also has the ability to reuse more data at lower memory levels, thereby potentially increasing its energy efficiency. Note that the area efficiency of BrainTTA *still* beats the competition, see Fig. 1.

Fig. 8 shows the energy required to perform three convolution layers, with binary, ternary, and 8-bit operands. A `MAC` is counted as two operations. It can be seen that the energy per operation difference between the binary and ternary convolution is nearly a factor of 2. Furthermore, the breakdowns for the binary and ternary layers are very similar with the exception of the instruction memory and the parameter memory.

The reason for this similarity is that the compute unit (vMAC) circuitry and usage of the binary and ternary convolution are very alike, and their utilization of the other components is identical *but* the amount of computations per second is halved since the ternary digits (trits) take up twice the space of single binary digits, hence the doubled energy per operation. The difference in energy consumption of the parameter memory is simply due to the lower toggle count on the data lines in the binary benchmark compared to the ternary and 8-bit benchmark (i.e. the learned neural network weights have more consecutive logic 0s and 1s for the chosen temporal schedule for the binary benchmark).

The breakdown of the 8-bit convolution shows that a larger percentage of the energy is spent in the vMAC unit because the arithmetic operations cannot be simplified like the binary and ternary variants; the energy per operation grows faster than the bit-width. The energy used by the vMAC unit scales superlinearly with the bit-width whereas the other components only scale linearly; therefore, to achieve high energy efficiency for low bit-width operands, the energy consumption of the linearly scaling components should be minimized. This can be achieved by, for example, maximizing data reuse, and minimizing instruction overhead.

Furthermore, the interconnect (IC) of the TTA core takes second place in energy usage in the logic after the vMAC. This is one of the architectural characteristic components where a price is paid for flexibility. The routing flexibility in BrainTTA in combination with its freedom to implement any FU (and retain compiler support) makes it possible to run more complex networks like ResNet and even non-DNN workloads independently of the general-purpose processor.

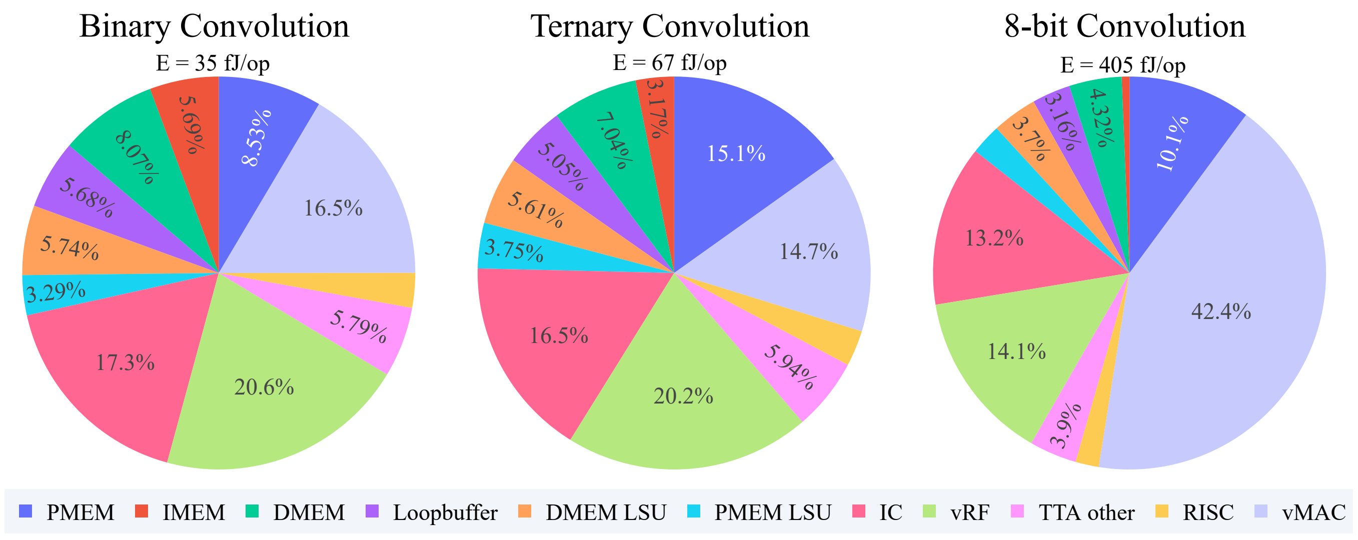


Fig. 8: Energy breakdown for the convolutional layers (GF 22nm FDX, 300 MHz, 0.5V, R=S=3, M=C=128 and W=H=16).

VI. Related work

Recently, several architectures have been proposed to accelerate neural network inference under stringent power constraints. These architectures can be categorized into two groups: programmable solutions and fixed-function accelerators. Programmable solutions provide flexibility in the temporal execution schedule of a neural network layer, allowing for adaptation to different layer types and dimensions while maintaining energy efficiency. On the other hand, fixed-function accelerators spatially map (complete) layers, limiting the temporal scheduling freedom. While fixed-function accelerators offer excellent energy efficiency, their support for layer types and dimensions is fixed during design, which significantly hampers the execution efficiency of different layer types and layers with different dimensions than supported by their spatial design.

A. Programmable solutions

We categorize programmable solutions into three types: compiler-programmable, assembly dialect, and run-time configurable network-on-chip (NoC).

Compiler-programmable solutions includes architectures such as XPULPNN [17] and Dustin [18]. These architectures are based on extensions of a RISC-V processor architecture. Dustin, for instance, combines a RISC-V host processor with a 16-core cluster consisting of RISC-V-based processing units optimized for operands of different precisions. The 16-core cluster can be configured to run in SIMD mode.

Assembly dialect supported architectures include XNOR-Neural Engine (XNE) [19] and SamurAI ([20]). XNE features a binary neural network accelerator that supports the previously described reduced arithmetic complexity introduced by binary quantization. XNE's SoC [19] allows running neural networks with the help of a configurable microcontroller unit (MCU) that is programmable using an assembly dialect. Similar to XNE, SamurAI ([20]) is an SoC consisting of a DNN accelerator in combination with a RISC-V processor. The accelerator features 64 8-bit processing elements spread over two clusters in groups of four.

Configurable NoC architectures include the EyeRiss v2 [21] accelerator. EyeRiss v2 introduces a configurable NoC that is able to adapt to the various dimensions of DNN layers. The hierarchical mesh network can be configured for unicast mode (low spatial reuse), broadcast mode (high spatial reuse), or a combination of both.

B. Fixed-function accelerators

Next to the programmable solutions, research has been done on fixed-function accelerators. Compared to programmable solutions, fixed-function accelerators are less flexible, as many of the neural network dimension parameters are hard-wired into the SoC design.

ChewBaccaNN [22] is an accelerator that supports binary operands. The kernel-size is hard-wired and data reuse is exploited by utilizing shift-registers. In [23] a ternary accelerator called CUTIE is presented. By spatially unrolling a complete 4D kernel (R, S, C and M dimensions, see Listing 1), it achieves significant data reuse if the loop iterators align with the hardware design. However, spatial unrolling this much directly constrains the network layer sizes that can be efficiently processed, thereby sacrificing flexibility. Another binary accelerator, [14], is implemented based on the computation-near-memory principle in 10nm FinFet technology. All kernel sizes are hard-wired, resulting in little to no flexibility.

In [24], the concept of having scheduling flexibility in neural network inference is explored. The need for scheduling flexibility is motivated by the change in layer parameters from the first layers into deeper layers. Two different schedules are implemented in hardware, both relying on feature map parallelism. One for the shallow layers ($W, H > C$) and one for the deeper layers ($C > W, H$). Although the addition of scheduling flexibility improves throughput, power numbers for

ASIC implementation are not provided, and thus, this paper is not considered in our actual comparison.

VII. Comparison to state-of-the-art

We compare BrainTTA not only to similar programmable architectures, but also to fixed-function accelerators. This comparison helps us identify the energy and area efficiency gap between these two categories. Table I presents a detailed comparison of BrainTTA with state-of-the-art programmable architectures. Fig. 1 provides an overall comparison of BrainTTA to both programmable and fixed-function architectures, showcasing energy and area efficiency.

From Fig. 1, we observe that BrainTTA achieves the best area and energy efficiency among all programmable works for all bit-widths supported. Although BrainTTA's area efficiency is negatively affected by its larger memory capacity compared to other works, it still maintains a lead in area efficiency. There is, however, still (at most) an order-of-magnitude gap between the fixed-function accelerators and the programmable solutions in both area and energy efficiency. However, performance of a fixed-function accelerator may reduce dramatically if a DNN layer does not match the design point (i.e. spatial unrolling) of the processing hardware, or can even not be executed at all.

When compared to the configurable NoC EyeRiss v2, BrainTTA demonstrates a 10x higher energy efficiency, and a 4.7x higher area efficiency. EyeRiss v2 prioritizes parallel MACs over memory in terms of area allocation, and therefore has a 2x higher throughput. BrainTTA can easily be scaled for higher throughput, e.g. by adding more vMAC units; TTAs are extremely scalable in this respect [5].

Furthermore, when compared to assembly dialect architectures like XNE and SamurAI, BrainTTA outperforms them in terms of energy and area efficiency. BrainTTA achieves 3.3x better energy efficiency than XNE for binary operands and 1.9x better energy efficiency than SamurAI for 8-bit operands. In both cases this is due to better data reuse in BrainTTA, using a combination of broadcasting in the vMAC units together with parameter buffering in the vector register files.

Finally, when compared to compiler-programmable works like XPULPNN and Dustin, we observe BrainTTA has a slight advantage on 8-bit operands. It is important to note that for a fair comparison, the results of Dustin as displayed in Table I should be technology-scaled (65nm vs. 22nm); in this comparison we assume linear scaling with a factor 3. After accounting for technology scaling, BrainTTA achieves a modest increase in energy efficiency of 14% and 39% over XPULPNN and Dustin respectively for 8-bit operands, primarily due to the lower operating voltage.

However, the key advantage of BrainTTA lies in its superior scaling to low bit-width operands. When decreasing the operand width from eight to two bits, XPULPNN and Dustin increase their energy efficiency by a factor 2.7 and 3.8. In other words, their energy efficiency scales sub-linear. Whereas in BrainTTA, the energy efficiency scales with a factor 11.4x and 5.96x when decreasing the operand width by a factor 8 and 5 respectively (ternary operand is considered as 1.6 bits [23]); i.e. operand scaling in BrainTTA yields a superlinear energy efficiency increase. This significant improvement in scaling can be attributed to the optimized balance between compute, data movement, and overhead energy in BrainTTA (compute energy scales quadratic, while the others scale linear with bit-width). The programmable exposed-datapath TTA architecture of BrainTTA allows for minimal data movement by 1) utilizing explicit register file bypassing and 2) the ability to use various temporal execution schedules. These features are paramount to achieve high energy efficiency for inference with low-precision operands. The overhead in Dustin and XPulpNN apparently limits their ability to efficiently support small bit-widths as it offsets the super-linearly decreasing compute cost for the smaller bit-width operands. Note that although Fig. 1 does not accommodate for differences in technology size or voltage, the advantage of improved scaling with the bit-width still holds.

Overall, BrainTTA showcases superior energy and area efficiency compared to state-of-the-art programmable architectures, highlighting the effectiveness of the TTA architecture in optimizing energy consumption and performance for various operand widths.

VIII. Conclusion

A novel TTA-based SoC for neural network inference that seamlessly combines flexibility with efficiency is presented. This SoC is able to perform operations at 28.6/14.9/2.47 TOP-S/W for binary, ternary and 8-bit operands, respectively. Still, it is highly flexible and can easily adapt to different types of networks such that it can advance together with the algorithmic inventions in the area of heavily quantized neural networks. The support for mixed-precision allows BrainTTA to mitigate accuracy loss in layers that are most adversely affected by low bit-width quantization (typically the first and last layer of the network). The programmable exposed-datapath architecture allows for temporal scheduling freedom in combination with explicit register file bypassing thus enabling BrainTTA to achieve superior scaling in terms of energy efficiency to low bit-width operands.

The mixed-precision and compiler support, in combination with the exposed-datapath, enable efficient execution of DNN layers with low bit-width operands making this architecture very versatile while beating the energy efficiency of other programmable architectures.

A. *Future work*

We see various options for future improvements such as:

Ternary data compression. In the current architecture one ternary symbol is stored in a 2-bits format, whereas we could store five ternary symbols in 8-bits to effectively store one symbol in 1.6-bits. This comes at the cost of compression and decompression logic.

Multiple vMAC units. The area efficiency of the current architecture could easily be raised by including more vMAC units in the design, since the current architecture is dominated by memory. Additionally, energy can than be saved using direct FU-to-FU communication by bypassing the RF.

TABLE I: Comparison between BrainTTA and state-of-the-art programmable architectures as discussed in Section VI, a MAC operation is counted as two operations. The energy efficiency is obtained under the same operating conditions as in Fig. 8.

	EyeRiss v2 [21]	**XNE** [19]	**SamurAI** [20]	**XPULPNN** [17]	**Dustin** [18]	**This Work**
Technology node	65nm	22nm	28nm	22nm	65nm	22nm
Programmability	Configurable	ASM[1]	ASM	Compiler	Compiler	Compiler
Supply voltage [V]	-	0.6	0.45	0.6	0.8	0.5
Inference precision	8	1	8,16,32	2,4,8,16,32	2,4,8,16,32	1,T[2],8
Energy efficiency	252 GOPS/W (8b)[3]	8.7 TOPS/W (1b)	1.3 TOPS/W (8b)	2.2 TOPS/W (8b) 6.1 TOPS/W (2b)	606 GOPS/W (8b) 2304 GOPS/W (2b)	**2.5 TOPS/W (8b)** **14.9 TOPS/W (T)** **28.6 TOPS/W (1b)**
Throughput[4]	154 GOPS (8b)[5]	67 GOPS (1b)	2.8 GOPS (8b)	22.8 GOPS (8b) 74.2 GOPS (2b)	8.8 GOPS (8b) 34.6 GOPS (2b)	77 GOPS (8b) 307 GOPS (T) 614 GOPS (1b)
Memory capacity [kB]	246	520	464	640	80	1024
Core area [mm^2]	28.1[6]	2.32	4.5	1.05	10	2.98
Area eff. [GOPS/mm^2]	5.5 (8b)	28.9 (1b)	0.6 (8b)	21.7 (8b) 70.7 (2b)	0.9 (8b) 3.46 (2b)	25.8 (8b) 103.0 (T) 206.0 (1b)

[1] ASM = Assembly dialect.
[2] T = `ternary`.
[3] Average on AlexNet.
[4] Throughput at most energy efficient operating point unless specified otherwise.
[5] Peak throughput, not throughput at most efficient operating point.
[6] Area is estimated by scaling the area of EyeRiss v1 (12.25mm2, 1176k NAND2-gates) by the gate-count difference with EyeRiss v2 (2695K NAND2-gates).

Automated scheduling exploration. We could intertwine the compiler with a hardware-aware scheduling tool such as ZigZag [25] to come up with better temporal schedules.

ACKNOWLEDGEMENT

Author Pekka Jääskeläinen thanks for the funding from Academy of Finland (decisions #331344 and #353199).

REFERENCES

[1] M. Tan *et al.*, "MnasNet: Platform-Aware Neural Architecture Search for Mobile," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2019, pp. 2815–2823.

[2] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag, "What is the State of Neural Network Pruning?" in *Proc. Machine Learning and Systems (MLSys)*, 2020, pp. 129–146.

[3] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A Survey of Quantization Methods for Efficient Neural Network Inference," *arXiv preprint arXiv:2103.13630*, 2021.

[4] L. Waeijen, S. Sioutas, M. Peemen, M. Lindwer, and H. Corporaal, "ConvFusion: A Model for Layer Fusion in Convolutional Neural Networks," *IEEE Access*, 2021.

[5] H. Corporaal, *Microprocessor Architectures: from VLIW to TTA*. John Wiley & Sons, Inc., 1997.

[6] B. Jacob *et al.*, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2018, pp. 2704–2713.

[7] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in *Proc. European Conf. Computer Vision (ECCV)*, 2016, pp. 525–542.

[8] L. Deng, P. Jiao, J. Pei, Z. Wu, and G. Li, "GXNOR-Net: Training Deep Neural Networks with Ternary Weights and Activations Without Full-precision Memory under a Unified Discretization Framework," *Neural Networks*, pp. 49–58, 2018.

[9] A. Bulat and G. Tzimiropoulos, "XNOR-Net++: Improved Binary Neural Networks," *Proc. British Machine Vision Conf. (BMVC)*, 2019.

[10] S. Gluska and M. Grobman, "Exploring Neural Networks Quantization via Layer-Wise Quantization Analysis," *arXiv preprint arXiv:2012.08420*, 2020.

[11] J. Multanen, "Energy-Efficient Instruction Streams for Embedded Processors," Ph.D. dissertation, Tampere University, 2021.

[12] P. Jääskeläinen, T. Viitanen, J. Takala, and H. Berg, "HW/SW Co-design Toolset for Customization of Exposed Datapath Processors," in *Computing Platforms for Software-Defined Radio*. Springer, 2016, pp. 147–164.

[13] M. Gautschi *et al.*, "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, pp. 2700–2713, 2017.

[14] P. C. Knag *et al.*, "A 617-TOPS/W All-Digital Binary Neural Network Accelerator in 10-nm FinFET CMOS," *IEEE J. Solid-State Circuits (JSSC)*, 2021.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2016, pp. 770–778.

[16] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," in *Proc. AAAI Conf. Artificial Intelligence*, 2017, p. 4278–4284.

[17] A. Garofalo, G. Tagliavini, F. Conti, D. Rossi, and L. Benini, "XpulpNN: Accelerating Quantized Neural Networks on RISC-V Processors Through ISA Extensions," in *Proc. Des. Autom. Test Eur. (DATE)*, 2020, pp. 186–191.

[18] G. Ottavi *et al.*, "Dustin: A 16-Cores Parallel Ultra-Low-Power Cluster With 2b-to-32b Fully Flexible Bit-Precision and Vector Lockstep Execution Mode," *IEEE Trans. Circuits Syst. I (TCAS-I)*, pp. 1–14, 2023.

[19] F. Conti, P. D. Schiavone, and L. Benini, "XNOR Neural Engine: A Hardware Accelerator IP for 21.6-fJ/op Binary Neural Network Inference," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD)*, pp. 2940–2951, 2018.

[20] I. Miro-Panades *et al.*, "SamurAI: A 1.7MOPS-36GOPS Adaptive Versatile IoT Node with 15,000× Peak-to-Idle Power Reduction, 207ns Wake-Up Time and 1.3TOPS/W ML Efficiency," in *Proc. IEEE Symp. VLSI Circuits Dig. Tech. Pap.*, 2020, pp. 1–2.

[21] Y. H. Chen, T. J. Yang, J. S. Emer, and V. Sze, "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," *IEEE J. Emerg. Sel. Top. Circuits Syst. (JETCAS)*, pp. 292–308, 2019.

[22] R. Andri, G. Karunaratne, L. Cavigelli, and L. Benini, "ChewBaccaNN: A Flexible 223 TOPS/W BNN Accelerator," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2021, pp. 1–5.

[23] M. Scherer, G. Rutishauser, L. Cavigelli, and L. Benini, "CUTIE: Beyond PetaOp/s/W Ternary DNN Inference Acceleration With Better-Than-Binary Energy Efficiency," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD)*, pp. 1020–1033, 2022.

[24] J. Cho, Y. Jung, S. Lee, and Y. Jung, "Reconfigurable Binary Neural Network Accelerator with Adaptive Parallelism Scheme," *Electronics (Switzerland)*, pp. 1–13, 2021.

[25] L. Mei, P. Houshmand, V. Jain, S. Giraldo, and M. Verhelst, "ZigZag: Enlarging Joint Architecture-Mapping Design Space Exploration for DNN Accelerators," *IEEE Trans. Comput. (TC)*, pp. 1160–1174, 2021.