Physically Tightly Coupled, Logically Loosely Coupled, Near-Memory BNN Accelerator (PTLL-BNN)

Yun-Chen Lo, Yu-Chun Kuo, Yun-Sheng Chang, Jian-Hao Huang, Jun-Shen Wu, Wen-Chien Ting, Tai-Hsing Wen, and Ren-Shuo Liu
Department of Electrical Engineering, National Tsing Hua University, Hsinchu, Taiwan

Abstract—In this paper, a physically tightly coupled, logically loosely coupled, near-memory binary neural network accelerator (PTLL-BNN) is designed and fabricated. Both architecture-level and circuit-level optimizations are presented. From the perspective of processor architecture, the PTLL-BNN includes two new design choices. First, the proposed BNN accelerator is placed close to the SRAM of the embedded processors (i.e., physically tightly coupled and near-memory); thus, the extra SRAM cost that is incurred by the accelerator is as low as 0.5 KB. Second, the accelerator is a memory-mapped IO (MMIO) device (i.e., logically loosely coupled), so all embedded processors can be equipped with the proposed accelerator without the burden of changing their compilers and pipelines. From the circuit perspective, this work employs four techniques to optimize the power and costs of the accelerator. First, this design adopts a unified input-kernel-output memory instead of separate ones, which many previous works adopt. Second, the data layout that this work chooses increases the sequentiality of the SRAM accesses and reduces the buffer size of storing the intermediate values. Third, this work innovatively proposes to fuse the max-pooling, batch-normalization, and binarization layers of the BNNs to significantly reduce the hardware complexity. Finally, a novel methodology of generating the scheduler hardware of the accelerator is included. We fabricate the accelerator using the TSMC 180 nm technology. The chip measurement results reach 91 GOP/s on average (307 GOP/s at peak) at 200 MHz. The achieved GOP/s per million logic gates and GOP/s per KB SRAM are 2.6 to 237 times greater than that of previous works, respectively. We also realize an FPGA system to demonstrate the recognition of CIFAR-10/100 images using the fabricated accelerator.

I. INTRODUCTION

Convolutional neural networks (CNNs) have recently emerged as a promising and successful technique to tackle important artificial intelligence (AI) problems such as computer vision. For example, state-of-the-art CNNs can recognize a thousand categories of objects in the ImageNet dataset both faster and more accurately (e.g., 97.75% accuracy in [9]) than humans (94.9% [8]).

Among the many CNN techniques, Binary CNNs (BNNs for short) [7], [13] are the most suitable for embedded devices such as those of the internet of things (IoT). The multiplications of BNNs are equivalent to logic XNOR operations, which are much simpler and consume much lower power than full-precision integer or floating-point multiplications. Meanwhile, open hardware and open instruction set architecture (ISA) have also attracted great attention recently. For example, many RISC-V solutions ([1], [4] and [14] to name a few) have become available and popular in the past very few years.

In view of the BNN, IoT, and RISC-V trends, developing an architecture that integrates embedded processors with BNN acceleration is unsurprisingly an important topic. Figure 1 compares our proposed physically tightly coupled, logically loosely coupled, near-memory BNN accelerator (PTLL-BNN) architecture with two other commonly used alternative architectures that we refer to as the vector processor (VP) and the peripheral engine (PE).

The design philosophy of the VP is to make the BNN accelerator tightly coupled to the processors. More specifically, it integrates vector instructions into processor cores [12], [2], and thus it offers good programmability to support general-purpose workloads. The inevitable and biggest downside of developing the VP architecture lies in the significant costs of developing toolchains (i.e., compilers) and hardware (i.e., the pipeline datapath and control). Another disadvantage is that even with vector instructions, moving data between SRAM and processor registers (i.e., load and store) and loops (e.g., branch) incur power and performance costs.

Abstract—In this paper, a physically tightly coupled, logically loosely coupled, near-memory binary neural network accelerator (PTLL-BNN) is designed and fabricated. Both architecture-level and circuit-level optimizations are presented. From the perspective of processor architecture, the PTLL-BNN includes two new design choices. First, the proposed BNN accelerator is placed close to the SRAM of the embedded processors (i.e., physically tightly coupled and near-memory); thus, the extra SRAM cost that is incurred by the accelerator is as low as 0.5 KB. Second, the accelerator is a memory-mapped IO (MMIO) device (i.e., logically loosely coupled), so all embedded processors can be equipped with the proposed accelerator without the burden of changing their compilers and pipelines. From the circuit perspective, this work employs four techniques to optimize the power and costs of the accelerator. First, this design adopts a unified input-kernel-output memory instead of separate ones, which many previous works adopt. Second, the data layout that this work chooses increases the sequentiality of the SRAM accesses and reduces the buffer size of storing the intermediate values. Third, this work innovatively proposes to fuse the max-pooling, batch-normalization, and binarization layers of the BNNs to significantly reduce the hardware complexity. Finally, a novel methodology of generating the scheduler hardware of the accelerator is included. We fabricate the accelerator using the TSMC 180 nm technology. The chip measurement results reach 91 GOP/s on average (307 GOP/s at peak) at 200 MHz. The achieved GOP/s per million logic gates and GOP/s per KB SRAM are 2.6 to 237 times greater than that of previous works, respectively. We also realize an FPGA system to demonstrate the recognition of CIFAR-10/100 images using the fabricated accelerator.

I. INTRODUCTION

Convolutional neural networks (CNNs) have recently emerged as a promising and successful technique to tackle important artificial intelligence (AI) problems such as computer vision. For example, state-of-the-art CNNs can recognize a thousand categories of objects in the ImageNet dataset both faster and more accurately (e.g., 97.75% accuracy in [9]) than humans (94.9% [8]).

Among the many CNN techniques, Binary CNNs (BNNs for short) [7], [13] are the most suitable for embedded devices such as those of the internet of things (IoT). The multiplications of BNNs are equivalent to logic XNOR operations, which are much simpler and consume much lower power than full-precision integer or floating-point multiplications. Meanwhile, open hardware and open instruction set architecture (ISA) have also attracted great attention recently. For example, many RISC-V solutions ([1], [4] and [14] to name a few) have become available and popular in the past very few years.

In view of the BNN, IoT, and RISC-V trends, developing an architecture that integrates embedded processors with BNN acceleration is unsurprisingly an important topic. Figure 1 compares our proposed physically tightly coupled, logically loosely coupled, near-memory BNN accelerator (PTLL-BNN) architecture with two other commonly used alternative architectures that we refer to as the vector processor (VP) and the peripheral engine (PE).

The design philosophy of the VP is to make the BNN accelerator tightly coupled to the processors. More specifically, it integrates vector instructions into processor cores [12], [2], and thus it offers good programmability to support general-purpose workloads. The inevitable and biggest downside of developing the VP architecture lies in the significant costs of developing toolchains (i.e., compilers) and hardware (i.e., the pipeline datapath and control). Another disadvantage is that even with vector instructions, moving data between SRAM and processor registers (i.e., load and store) and loops (e.g., branch) incur power and performance costs.
In comparison, the design philosophy of the PE architecture makes the BNN acceleration *loosely coupled* to the processor cores [6], [18], [11], [17] using a system bus such as an AHB (Advanced High-performance Bus). In contrast to the VP, most IC design companies are familiar and comfortable with the PE architecture, which avoids the abovementioned compiler and pipeline development costs. In addition, without load, store, and loop costs, the PE can potentially achieve higher performance than the VP. The downside of the PE architecture is that it utilizes private SRAM instead of sharing the available SRAM of the embedded processors. Typically, embedded processors for IoT devices [3] are equipped with approximately 64 to 160 KB of tightly coupled memory (TCM) that is made of SRAM and can support concurrent code executions and data transfers. TCM is also known as tightly integrated memory [4], scratchpad memory, or local memory [1].

Clearly, neither the VP nor the PE is the best solution. The former is superior at minimizing the SRAM overhead while the latter provides potentially higher performance and zero compiler and pipeline involvement. Therefore, this paper proposes the PTLL-BNN architecture. As shown in Figure 1, a PTLL-BNN accelerator is designed to be placed near the SRAM TCM of an embedded RISC core (i.e., physically tightly coupled and near-memory) and to be controlled through a memory mapped IO (MMIO) interface (i.e., logically loosely coupled). By doing so, the PTLL-BNN achieves the best of both worlds. The design choice of placing the accelerator near the memory is also inline with the recently emerged computing-in-memory (CIM) trend [5], [10], [15], [16].

The rest of this paper is organized as follows. Section II describes the PTLL-BNN architecture in more detail and further presents four circuit-design technologies: 1) unified input-kernel-output SRAM, 2) sequential data layout and partial-sum reuse schedule, 3) fused MaxPooling-BatchNorm-Binarization, and 4) scheduler generation from nested scheduling loops. Section III presents the chip measurement results. Section IV concludes this work.

II. ARCHITECTURE AND CIRCUIT DESIGN

A. PTLL-BNN Architecture

Figure 2 illustrates the proposed PTLL-BNN architecture. The RISC processor core and accelerator share the SRAM TCM. The MMIO interface controls the accelerator and multiplexes the address, data, read-enable, and write-enable signals of the SRAM between the RISC processor core and accelerator. By adopting such a design, the PTLL-BNN architecture is compatible with most (if not all) embedded processors, which usually support SRAM TCM and MMIO.

Figure 3 illustrates the PTLL-BNN in more detail. The TCM we implemented is single-port 22 KB (5632 × 32) SRAM, which is big enough to accommodate an input layer, eight kernels, and a pooled output layer of a BNN for CIFAR-10/100 image recognition. A scheduler fetches input feature maps and kernel weights from the TCM, sends them to the internal registers, performs bitwise dot-products (i.e., XNOR and popcount), and accumulates the dot-product results in the partial-sum buffer, which is dual-port 0.5 KB (32 × 128) SRAM. A Fused MaxPooling-BatchNorm-Binarization block converts partial-sums into output feature maps, which are stored back into the TCM.

B. Unified Input-Kernel-Output SRAM

Unlike previous works [17], [6], [18], [11] that utilize multiple SRAM ports and banks for inputs, outputs, and/or kernels, the PTLL-BNN intentionally utilizes only a single SRAM port and bank. By doing so, the PTLL-BNN can share the SRAM TCM of most embedded processors. With only a single SRAM bank available, as shown in Figure 4, the accelerator employs three pointers to store the base addresses of the inputs, kernels, and outputs. The processor can configure the three pointers through the MMIO, and the accelerator accesses the TCM according to the pointers for the inputs, kernels, and outputs.

C. Sequential data layout and partial-sum reuse schedule

The input and output feature maps of BNNs are binary values, but the partial sums are integers, which occupy much
SRAM. Therefore, instead of increasing the data reuses of inputs or kernels, which produce more intermediate partial sums, our design schedules operations to exhaust the data reuses of the partial sums and then to convert the partial sums back to the TCM as binary values. This strategy helps to minimize the required amount of partial-sum SRAM.

The data layout is designed to increase sequential memory accesses, which help to reduce power. Figure 5 shows an example data layout that places 8×8×64-bit inputs and eight 3×3×64-bit kernels in the TCM. To produce the first output row, the kernel registers are first filled with words 48–71 (relative to the kernel pointer). Then, the 3-tap input registers are filled with words 0–7 (relative to the input pointer) to perform a convolution with 8 dot products, and each dot product involves 96 multiply-and-accumulates (MACs). The results are accumulated in 64 variables (8 outputs×8 channels) in the partial-sum SRAM. Similar procedures are applied to input words 8–15, 64–71, and 72–79 and the corresponding kernels.

**Algorithm 1: Nested scheduling loops of the implemented BNN accelerator**

```plaintext
for(Yo := 0 to height(output_fmap)-1){
    for(Xo := 0 to width(output_fmap)-1){
        initialize partial_sum[Xo] to be the adjusted biases
    }
}
for(Xi := 0 to channel(input_fmap)/2){
    for(Yk := 0 to height(kernel)/2){
        for(Ci := 0 to channel(input_fmap)/32-1){
            for(Nk := 0 to number(kernels)-1){
                load kernelterrorism for[Yo][Ci][Nk][Xk] from the SRAM TCM
            }
        }
        for(Xi2 := 0 to width(input_fmap)-1){
            load input_fmap[Ci][Wow][Xi2] from the SRAM TCM
        }
    }
    for(Xi := 0 to width(input_fmap)-1){
        send partial_sum[Xi2] to the Fused-Max-BN-Bin block
    }
}
```

**Fig. 5. Data layout and computation scheduling**

**Fig. 6. Fused MaxPooling-BatchNorm-Binarization**

**D. Fused MaxPooling-BatchNorm-Binarization**

BNNs employ max pooling, batch normalization, and binarization steps in series to convert integer partial sums into binary outputs [7]. Figure 6(a) shows a naïve realization, which involves 7 integer operations (1 ADD, 4 SUB, 1 MULT, and 1 DIV) and 3 integer multiplexers to produce one output. This work proposes and implements a novel Fused MaxPooling-BatchNorm-Binarization approach, which produces an identical result but uses significantly fewer hardware resources, i.e., only a 4-input AND and a 2-input XNOR (Figure 6(b)). The key is to adjust the biases according to the batch normalization and perform binarization prior to max pooling. Note that adjusting the biases is done offline and does not incur costs at the run time, as explained in Figure 6(c).

**Fig. 7. Hardware generated from the nested scheduling loops**

**E. Scheduler Generation from Nested Scheduling Loops**

The MAC units of neural-network accelerators usually possess a regular structure, but their schedulers are not always trivial, which results in hidden development costs. To tackle this issue, we implement a Python generator that converts nested scheduling loops, as shown in Algorithm 1, into hardware, as shown in Figure 7. With this methodology, the PTLL-BNN can be parameterized and generated along with processor generators [4].
III. Measurement Results

We fabricate the accelerator mentioned above using 180 nm CMOS technology, which is also adopted by several RISC-V processor chips such as [4]. Figure 8 shows the die photo and chip specs. Figure 9 shows a shmoo plot of core Vdd vs. operating frequency and energy efficiency.

Figure 10 compares the PTLL-BNN with three related works: [17], BNN-FPGA [18], and TinBiNN [2]. Among these four works, [17] supports ImageNet while all other three support CIFAR-10/100. Please note that our chip is in 180 nm while others are in 28 and 40 nm. The 22 KB TCM is shared with a processor. Therefore, the amount of SRAM cost demanded by the PTLL-BNN is 0.5 KB, which is 0.22% that of [17] (Figure 10). The number of logic gates for the accelerator (excluding SRAM) is 0.027 million, which is 1.5% as large as the number from [17] (Figure 10).

To demonstrate the advantages of the PTLL-BNN, we compare the GOP/s per million logic gates and GOP/s per KB SRAM. These two figures of merit are relatively independent of the benchmarks (e.g., ImageNet vs. CIFAR) and process technologies. The PTLL-BNN is 2.6× to 237× better than [17] and BNN-FPGA [18].

The highest frequency of our chip is 200 MHz at 1.95 V, which can support CIFAR-10/100 recognition at 75 frames/s using the BNN model in [7]. The peak energy efficiency is 1.18 TOP/s/W (35 mW at 1.2 V and 90 MHz). In Figure 11, CIFAR-10 recognition using the BNN [7] is performed using an FPGA demo system, and the accuracy (87.8%) matches our software-based results.

IV. Conclusions

A physically tightly coupled, logically loosely coupled, near-memory BNN accelerator (PTLL-BNN) is proposed and fabricated in this work. The PTLL-BNN is designed to achieve two critical objectives for embedded processors. The first is to minimize hardware cost. Here, the extra SRAM and logic gates are only 0.5 KB and 0.027 million, respectively. The second is to minimize the development costs. Here, no compiler and pipeline change is needed. The measurement results reach 91 GOP/s on average at 200 MHz, which corresponds to 2.6× to 237× greater GOP/s per million logic gates and per KB SRAM than previous works.

ACKNOWLEDGMENTS

This work is supported in part by MOST under grants 108-2218-E-007-023 and 108-2218-E-007-021 and in part by the NOVATEK Fellowship. We thank the TSRI and TSMC for the chip fabrication and the NCHC for the computer time and facilities.

REFERENCES