BSTC: A Novel Binarized-Soft-Tensor-Core Design for Accelerating Bit-Based Approximated Neural Nets

Ang Li
Pacific Northwest National Lab
ang.li@pnnl.gov

Tong Geng
Boston University
tianqi@bu.edu

Tianqi Wang
Boston University
tgeng@bu.edu

Martin Herbordt
Boston University
herbordt@bu.edu

Shuaiwen Leon Song
Pacific Northwest National Lab
shuaiwen.song@pnnl.gov

Kevin Barker
Pacific Northwest National Lab
kevin.barker@pnnl.gov

ABSTRACT

Binarized neural networks (or BNNs) promise tremendous performance improvement over traditional DNNs through simplified bit-level computation and significantly reduced memory access/storage cost. In addition, it has advantages of low-cost, low-energy, and high-robustness, showing great potential in resources-constrained, volatile, and latency-critical applications, which are critical for future HPC, cloud, and edge applications. However, the promised significant performance gain of BNN inference has never been fully demonstrated on general-purpose processors, particularly on GPUs, due to: (i) the challenge of extracting and leveraging sufficient fine-grained bit-level-parallelism to saturate GPU cores when the batch size is small; (ii) the fundamental design conflict between bit-based BNN algorithm and word-based architecture; and (iii) architecture & performance unfriendly to BNN network design. To address (i) and (ii), we propose a binarized-soft-tensor-core as a software-hardware codesign approach to construct bit-manipulation capability for modern GPUs and thereby effectively harvest bit-level-parallelism (BLP). To tackle (iii), we propose intra- and inter-layer fusion techniques so that the entire BNN inference execution can be packed into a single GPU kernel, and so avoid the high-cost of frequent launching and releasing. Experiments show that our Singular-Binarized-Neural-Network (SBNN) design can achieve over 1000× speedup for raw inference latency over the state-of-the-art full-precision BNN inference for AlexNet on GPUs. Comparisons with CPU, GPU, FPGA and Xeon-Phi demonstrate the effectiveness of our design. SBNN is opensourced and available at *https://github.com/uuudown/SBNN*.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; *Parallel algorithms*; • **Computer systems organization** → **Single instruction, multiple data**; **Neural networks**.

ACM Reference Format:
Ang Li, Tong Geng, Tianqi Wang, Martin Herbordt, Shuaiwen Leon Song, and Kevin Barker. 2019. BSTC: A Novel Binarized-Soft-Tensor-Core Design for Accelerating Bit-Based Approximated Neural Nets. In *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '19), November 17–22, 2019, Denver, CO, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3295500.3356169

SC '19, Nov 17–22, 2019, Denver, CO, USA

ACM ISBN 978-1-4503-6229-0/19/11…$15.00
https://doi.org/10.1145/3295500.3356169

1 INTRODUCTION

Binarized neural networks (BNNs) evolved from conventional binarized weight networks (BWNs) [11, 32, 70] when it was observed that, if weights could be binarized into ±1, then floating-point mul could be degraded to add (mul +1) and sub (mul -1). It was then observed that if both the weights and inputs could be binarized, then floating-point add and sub could be further degraded to logical bit operations (i.e., xnor and popc) [12, 32, 70].

BNNs have several advantages over conventional DNNs: (I) *Low computation demand.* Each 32 floating-point or integer mul-add operations can be aggregated into a single popc-xnor operation, significantly reducing latency and hardware design complexity. (II) *Low storage demand.* The entire memory hierarchy (e.g., registers, caches, scratchpad, main memory, flash, etc.) can sustain 32× or more storage and bandwidth compared with full-precision DNNs. (III) *Low energy consumption.* Due to reduced hardware complexity and smaller chip-area, BNN-based devices are much more energy efficient. (IV) *Reliability.* BNNs have recently been shown to have greater robustness against adversarial attacks than normal DNNs [22], and also can be encoded exactly for formal verification and property analysis [9, 61, 63].

The primary concern about BNNs is accuracy, which is becoming a popular topic for DNN algorithm research. Recently, with advanced training techniques being introduced [14, 32, 40, 74], BNN accuracy has improved significantly. The top-1 training accuracy for BNN-based AlexNet and ResNet on ImageNet have been enhanced from 27.9% and 42.2% [12, 32, 74] to 46.1% and 53% [14] within three years, with respect to 56.6% and 69.3% [28, 39] for the full-precision designs. With boosting enabled, BNN accuracy can reach 54.3% and 61% [80]. Although BNN accuracy is still slightly inferior, it shows great advantages in almost all the other aspects, e.g., low-cost, low-latency, low-energy, and high-robustness.

While BNNs are unlikely to completely replace full-precision DNNs, for many practical uses including edge (e.g., smart sensor [18, 60], mobile devices [59]), cloud (e.g., online recommendation [13, 29], AI as microservices [27, 35]), and HPC (e.g., high-energy physics data triggering [4, 16, 55], and optimizing search spaces [37, 58, 71]), when a certain accuracy bar is surpassed, other essential metrics such as real-time processing (particularly for large inputs),

resources consumption, cost, and power become more significant. BNNs and their variants, such as B-LSTM [19, 30], can thus provide attractive solutions. There has also been an effort to map BNN-like learning structures to quantum annealers with the goal of demonstrating quantum supremacy [3].

While most existing research either focuses on improving BNN accuracy, or exploring the design space for ASIC/FPGA implementation, the tremendous potential performance gains of BNN inference (32× memory and 10× computation reduction) has generally not been achieved on general-purpose processors such as GPUs. This is due to the following reasons. First, it is challenging to extract sufficient fine-grained parallelism for small batch or non-batching cases, leading to extremely poor utilization of GPU resources. These can be as low as 1% [64]). Second, there is a fundamental design conflict between bit-based algorithms and word-based architectures, which lack the full complement of bit-manipulation support. Third, current BNN network designs are mainly proposed from algorithm and accuracy perspectives and have therefore not benefited from potential contributions from architecture-aware performance improvements. For example, only convolution and fully-connect layers in BNN models are binarized; the remaining layers, such as batch-normalization and pooling, all consist of expensive full-precision floating-point calculation and memory access.

These challenges lead to inefficient BNN designs and implementations on general-purpose accelerators, significantly reducing their adoption even for scenarios that require high real-time and resource constraints. Meanwhile, the majority of BNN algorithm research has been conducted via full-precision simulations in high-level programming environments (e.g., Tensorflow and PyTorch). While modern general-purpose many-core accelerators such as CPUs and GPUs provide great computation power, due to the poor hardware utilization and extreme real-time latency constraints for inference, conventional wisdom suggests that they are incapable of bringing significant speedups for BNN designs applied to real-world/production scenarios.

In this paper, we address the three challenges above by proposing the *Singular-Binarized-Neural-Network* (SBNN). To systemically build the bit-manipulation capability for modern GPUs, while keeping the hardware highly utilized, we adopt a software-hardware co-design approach. A virtual bit-core, the Binarized-Soft-Tensor-Core (BSTC), is constructed on the GPU's Streaming Multiprocessors (SMs). It leverages native hardware instructions to obtain high efficiency while offering desired bit-manipulation APIs to the application layer for harvesting the bit-level-parallelism (BLP) that have recently emerged as the key optimization target in various domains [2, 6, 7, 20, 21, 67, 72, 77].

To create a hardware-friendly and performance-oriented design, we propose several BNN network adjustments. For instance, intra- and inter-layer fusion techniques, which can merge the entire BNN inference process into a single GPU kernel, can help achieve up to *6115*× latency reduction for MLP on MNIST, and *1429*× latency reduction for AlexNet on ImageNet, when comparing with full-precision simulated BNN in a high-level DNN environment (i.e., TensorFlow via cuDNN) running on an NVIDIA Tesla-P100 GPU. In this design, there is no warp-divergence, no shared memory bank conflicts, no non-coalesced global memory accesses, and no additional workspace requirement; this leads to excellent computation efficiency and resource utilization. Because of the whole-network fusion, all GPU kernel invocation and release overheads from the conventional design are eliminated. As a result, SBNN can complete non-batched AlexNet inference within 1ms. Finally, the proposed design is purely software-based with no dependence on any external libraries. It can be directly deployed on existing HPC and embedded GPU platforms with no need to adjust or reconfigure when the network or input size changes.

2 RELATED WORK

The emergence of deep neural networks has brought significant challenges and opportunities for application-specific system and architecture design [5, 73]. Since BNNs were first proposed in 2016 [12, 70], the subsequent research around BNNs can be classified into two major categories: *algorithms* and *implementations*.

Algorithms. The main objective is to improve BNN training accuracy [12, 56, 60, 70, 74, 79], especially for large datasets. XNOR-Net [70] applied BNNs on ImageNet, reporting top-1 accuracies of up to 51.2% for full binarization and 65.5% for partial binarization. DoReFa-Net [79] reported best-case ImageNet top-1 accuracies of 43% for full and 53% for partial binarization. Another study [74] reported 46.6%/71.1% top-1/5 accuracy for AlexNet on ImageNet by taking advantage of their new observation on learning-rate, activation function, and regularizer. Recent studies such as ABC-Net [56], Self-Binarizing [40], BNN+ [14], and BENN [80] reported even better BNN training accuracy.

Implementation. The goal is to build high-performance BNNs to satisfy the stringent real-time inference constraints of delay-critical applications in cloud and edge domains, and with as little area and energy cost as possible [23, 24, 31, 54, 60, 64, 75, 78]. The majority of these studies are FPGA-based [23, 24, 54, 64, 75, 78] due to FPGA's flexible and powerful bit-manipulation capability. Recently, a CPU-based BNN implementation was proposed [31]. This work relies on bit-packing and AVX/SSE vector instructions to derive good bit computation performance. However, their implementation mainly focuses on Bit-Matrix-Multiplication (BMM); Bit-Convolution (BConv) is converted to BMM through the conventional flatten or unfold approach with expensive pre/post-processing needed for padding. They do not discuss intra- and inter-layer fusion for whole network optimization. Although recent studies have mainly focused on optimizing individual layers, we believe that inter-layer and whole-network optimization are becoming more essential. For example, the TensorFlow XLA [41] and Tensor Comprehensions [76] recently compiled entire neural network graphs at once, performing various transformations and achieving 4x speedup over manually tuned individual layers. Our design follows this emerging research trend by merging all the layers into a single GPU kernel, achieving significant inference speedups.

3 BINARIZED-SOFT-TENSOR-CORE

As shown in Figure 1, a binarized-soft-tensor-core (BSTC) is defined as *a 2D bit-tile with orthogonal dimensions corresponding to GPU warp lanes and bitwidth per lane.* A BSTC is executed purely via a single warp, with each lane contributing 32 bits or 64 bits, forming a 32×32 or 32×64 matrix; and each matrix element represents a binary

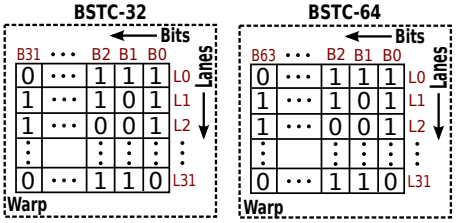

Figure 1: Binarized-Soft-Tensor-Core in 32 and 64 bits mode. Bits are indexed from right to left since in an operand, MSB is usually on the left, while LSB on the right.

state. For BSTC-32, the data type is unsigned int; for BSTC-64, it is unsigned long long int. BSTCs stay in the register file.

BSTC design follows our recently proposed *warp-consolidation* programming model [45], which unifies a thread block and a warp: each thread block contains only a single warp – a BSTC here. This model shows six advantages over the traditional CUDA model: (i) no thread block level synchronization; (ii) independent on-chip resource allocation and release; (iii) simplified kernel configuration and design space; (iv) register-shuffle-based fast inter-lane communication; (v) flexible fine-grained thread cooperation, and (vi) extended register space. Both BSTC and SBNN exploit these features to boost performance.

3.1 BSTC Operations

BSTC relies on fine-grained, highly-efficient inter-lane and inter-bit communication and cooperation mechanisms to design the required functionalities and achieve high performance. Figure 2 illustrates the general primitives to operate a BSTC:

(A): AND, OR and XOR are executed on each bit of the data along the rows, so the communication pattern is *intra-row*. XOR is very useful for ±1 constructing bit-dot-product. OR and AND are used for `bit-max` and `bit-min` operations.

(B): __popc() and __popcll(), also known as population-count, return the number of bits whose binary value equals to 1 in a 32/64-bit data. This primitive offers fast accumulation along the BSTC rows. They are the key operations to map from BSTC's binary space to normal full-precision space. popc has been widely adopted for aggregating `bit-dot-product` results.

(C): __brev() and __brevll() reverse the bit sequence of a 32/64 bit row. They offer the ability to fast rotating a bit-row for 180°. The communication pattern is intra-row. They are often used together with __ballot(), see (H).

(D): __any() and __all() are the warp voting operations. They are executed along bit-columns – merging 32 bits of a bit-column into a single bit, and broadcasting this bit to all the 32 thread lanes. __any() returns 1 if any entry of the column is 1 while __all() returns 1 when all of them are 1s.

(E): __shfl() is to exchange bit-rows in a BSTC. Shuffle has four variants: __shfl() performs flexible general bit-row exchanging function. __shfl_up() and __shfl_down() are to rotate bit-rows up and down by a certain interval. __shfl_xor() conducts butterfly bit-row exchanging [65]. The shuffle operations are quite useful for bit-row communication, achieving register-level data-reuse [6, 45].

(F): Left and right shifting are logical operations. We separate them out to highlight their communication patterns as inter-columns. They are often adopted to extract, exchange and merge bit-columns.

(G): The most interesting operation here perhaps is __ballot(). It returns a 32-bit integer with its Nth bit (from LSB to MSB) contributed as a predicate (1bit) from the Nth lane of the warp. In other words, it offers the ability to convert a bit-column into a bit-row. Recall that in a BSTC, lanes are indexed from top to bottom, while bits are indexed from right to left (Figure 1), __ballot() essentially rotates a bit-column by 90° clockwise to a bit-row.

(H): This operation is the conjugate of (G). In order to rotate a bit-column 90° anticlockwise to a bit-row, one has to combine __ballot() and __brev(). Note that there is no direct operation for rotating from bit-rows to bit-columns for existing hardware; one has to broadcast an entire row to all the lanes via __shfl() and then extract the required bit-column(s), which are expensive.

Having these BSTC operations, we can flexibly combine them to design versatile bit-based functions and communication patterns, accelerating various emerging bit-based algorithms [6, 7, 20, 21, 67, 72, 77] on GPUs.

4 BNN BIT FUNCTIONS

We discuss the bit functions in BNN algorithm, where BSTC operations can be applied. Figure 3 shows the network structure for BNN-based LeNet [43]. Note that only the inputs and weights for convolution and fully-connected layers are binarized and their outputs are in full-precision. Other kernels still rely on full-precision calculation and data access.

We first review the differences between BNNs and conventional DNNs/CNNs. As shown in Figure 3, BNNs have a binarization function (i.e., *Bin*) and their own versions of convolution (i.e., *BConv*) and fully-connected functions (i.e., *BMM*). Other functions (e.g., *BN*, *Actv*, *Pool*) are the same as in DNNs/CNNs. The binarization function *Bin* in BNNs is:

$$x^b = sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases} \quad (1)$$

Note that the "*sign()*" function in some high-level languages such as Python results not in binarization but in trinarization, as it returns 0 when x is 0. This is not an issue when simulating BNNs through a full-precision implementation (as "$x+0=x$" while "$x\times 0=0$"), but becomes a big problem with padding in low-level implementation. Therefore, using Eq (1) in Python for BNN training may lead to inconsistent results. We solve this issue by adding a very tiny ϵ to x before applying Eq (1) for binarization in Python-based BNN training (e.g, TensorFlow and PyTorch).

For *BConv* and *BMM*, the basic operation is bit dot-product:

$$v = \vec{a} \cdot \vec{b} = popc(\vec{a} \,\&\, \vec{b})$$

where $\vec{a}$ and $\vec{b}$ are bit-vectors, "&" refers to logical-and, and *popc* refers to population count. However, this is for a conventional 0/1 dot-product; bit-0 in BNNs is not 0 but -1. Consequently, the dot-product for -1/+1 becomes:

$$v = \vec{a} \cdot \vec{b} = n - 2 \times popc(\vec{a} \oplus \vec{b}) = 2 \times popc(\vec{a} \oplusbar \vec{b}) - n \quad (2)$$

where n is the length of $\vec{a}$ and $\vec{b}$; "$\oplus$" refers to *exclusive-or* or XOR, and "$\oplusbar$" refers to *exclusive-nor* or XNOR. As the XNOR gate is widely utilized in digital fabrication, most existing BNN algorithmic studies (e.g., [12, 70]) and FPGA/ASIC based implementations (e.g., [64, 75]) apply XNOR-based design (i.e., $2 \times popc(\vec{a} \oplusbar \vec{b}) - n$). But since XNOR is not directly supported on GPUs, we choose to apply

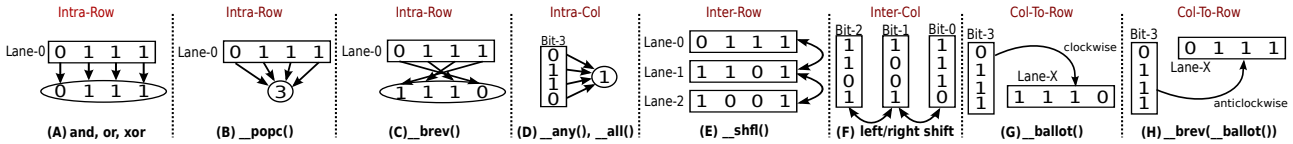

Figure 2: BSTC row-wise and column-wise operations and communication patterns.

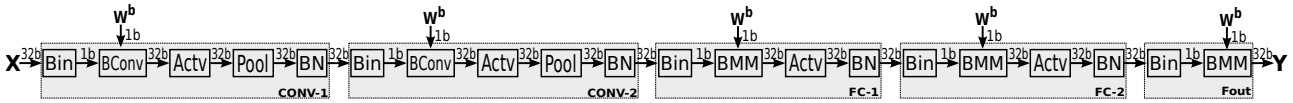

Figure 3: LeNet BNN network structure for inference. W^b refers to binarized weights. "1b" refers to 1-bit, "32b" refers to 32-bit full precision. "X" is the input. "Y" is the output. "Bin" refers to binarization. "BConv" refers to bit-convolution. "Actv" refers to activation, which is ReLU in this paper. "Pool" refers to pooling. "BN" refers to batch-normalization. "BMM" refers to bit-matrix-multiplication or fully-connected layer. Only BMM and BConv are binarized.

XOR approach instead (i.e., $n - 2 \times popc(\vec{a} \oplus \vec{b})$). Note that finding the right n is the key for excluding padding bits to ensure the correctness. We will discuss this in detail later.

Overall, the functional differences between BNNs and DNNs fall into four functionalities regarding to bit operations:

Bit-Packing: This corresponds to *Bin* in Figure 3. Following Eq (1), if one lane of a warp reads a datum into its register R0, we then use R0>=0 to generate a bit-column (1 bit per lane), and rely on BSTC op-(H) to rotate the bit-column anticlockwise to a bit-row and distributed it to all 32 lanes of the warp. Here, we use op-(H) rather than op-(G) because the result of __ballot() is little-endian, i.e., the first lane corresponds to the LSB of the result. Therefore, we need an additional __brev() to reverse the bit-sequence so that the first lane corresponds to MSB of the 32-bit result. For 64-bit binarization by a warp, __ballot() is repeated twice. After that, the two output unsigned ints are concatenated into a 64-bit unsigned long long int via the following embedded PTX:

```
asm volatile("mov.b64 %0, {%1,%2};":"=l"(l0):"r"(r0),"r"(r1)); //low,high
```

and then reverse as a whole using __brevll(l0).

Bit-Computation: This includes bit-dot-product and other bit operations. Bit-dot-product is used extensively in *BMM* and *BConv*. We rely on BSTC op-(A) and (B) to realize bit-dot-product. OR and AND in op-(A) can be quite useful when dealing with bit-based max- and min-pooling.

Bit-Communication: BSTC op-(E), (F) and (G) are particularly important for this purpose, which realize bit-row exchanging/broadcasting, bit-column extraction/adjustment/concatenation, bit-row rotation/merging, respectively.

Bit-Unpacking: Although this function is not directly used in BNN inference, it is very useful for debugging as it provides a way to obtain a human-readable result to compare against the correct outputs of a particular layer. Bit-unpacking is the reverse of bit-packing. Op-(F) is adopted here: each lane (corresponding to a bit-row) shifts and extracts a bit, and converts it to full-precision. For the 64-bit version, we can repeat the following routine while altering 31 to 63 for the second pass.

```
C = 2*(int)((R0>>(31-laneid))&0x1)-1;
```

5 SBNN SINGLE LAYER DESIGN

We discuss how to efficiently implement BMM and BConv based on BSTCs.

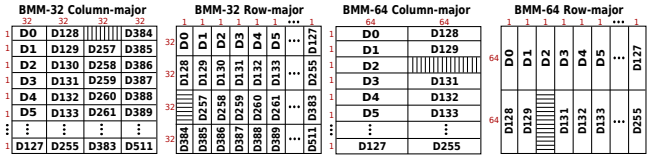

Figure 4: Row-major and Column-major Bit-Packing Format for BMM

```
1  template <typename T>
2  __global__ void PackTo32Col(const T* A,
3  unsigned* B, int A_height, int A_width){
4    unsigned Bval, laneid;//fetch lane-id
5    asm("mov.u32 %0, %%laneid;":"=r"(laneid));
6    #pragma unroll
7    for (int i=0; i<WARP_SIZE; i++){
8      T f0 = A[(blockIdx.x*WARP_SIZE+i)*A_width
9        +blockIdx.y*WARP_SIZE+laneid];
10     //rotate anticlockwise
11     unsigned r0=__brev(__ballot(f0>=0));
12     if (laneid == i) Bval = r0;
13   }
14   B[blockIdx.y*A_height+blockIdx.x*WARP_SIZE
15     +laneid] = Bval;
16 }
```

Listing 1: Binarize into col-major format

```
1  template <typename T>
2  __global__ void PackToC32Row(const T* A,
3  unsigned* B, int A_height, int A_width){
4    unsigned laneid;//fetch lane-id in a warp
5    asm("mov.u32 %0, %%laneid;":"=r"(laneid));
6    unsigned Bval = 0;
7    #pragma unroll
8    //process 1 bit per iteration
9    for (int i=0; i<WARP_SIZE; i++){
10     T f0 = A[(blockIdx.x*WARP_SIZE+i)*A_width
11       +blockIdx.y*WARP_SIZE+laneid];
12     //concatenate a bit on the right
13     Bval = (Bval<<1) | (f0>=0);
14   }
15   B[bx*A_width+by*WARP_SIZE+laneid]=Bval;
16 }
```

Listing 2: Binarize into row-major format

5.1 Bit Matrix Multiplication (BMM)

Matrix-multiplication comprises both vertical and horizontal data access. To gain performance we propose two bit-packing formats: *row-major* and *column-major*. Figure 4 shows how a 128×128 data block is packed into the two data formats:

- **Column-major**: Each of 32 consecutive raw data from a row of the original data matrix are binarized into 32 0/1 bits via Eq 1 and packed as an *unsigned int* (e.g., D0 in Figure 4), forming a BSTC per warp. These unsigned ints are then organized into a column-major order.
- **Row-major**: Each of 32 consecutive raw data from a column of the original data matrix are binarized and packed as an *unsigned int*, forming a BSTC per warp. These unsigned ints are then organized into a row-major order.

Listing 1 and 2 show the code for binarization into column-major and row-major 32-bit data formats, respectively. Line 11 in Listing 1 binarizes by a warp: 32 lanes cooperatively generate a complete 32-bit binarized result as a bit-row via op-(H) per each iteration, and save the bit-row to corresponding lane in Line 12, building a complete BSTC-32. Line 13 in Listing 2 shows another approach. Each lane concatenates a single bit per iteration to its own bit-row and the whole BSTC-32 is constructed after 32 iterations. Note that Listing 1 stores data into B by column-major order while Listing 2 stores data into B by row-major order. The two bit conversion methods have similar delay. The 64-bit versions are analogous.

```
1 __global__ void BMM32_Arow_Brow(...){
2 ...
3 register unsigned Cm[32] = {0};
4 for (int i = 0; i < A_width; i++){
5  unsigned r0 = Asub[i+A_height+laneid];
6  unsigned r1 = Bsub[i+WARP_SIZE+laneid];
7  #pragma unroll
8  for (int j=0; j<WARP_SIZE; j++){
9    //crop a bit-column of B
10   unsigned r2=__brev(__ballot((r1>>j)&0x1));
11   //each lane dot-product with the column of B
12   Cm[j] += __popc(r0 ^ r2);
13 }} ...
```

Listing 3: BMM by multiplying each A row with each B column

```
1 __global__ void BMM32_Arow_Bcol(...){
2 ...
3 register unsigned Cm[32] = {0};
4 for (int i = 0; i < A_width; i++){
5  unsigned r0 = Asub[i+A_height+laneid];
6  unsigned r1 = Bsub[i+B_width+laneid];
7  #pragma unroll
8  for (int j=0; j<WARP_SIZE; j++){
9    //broadcast a bit-row of B from lane-j
10   unsigned r2 = __shfl(r1, j);
11   //each lane dot-product with the row of B
12   Cm[j] += __popc(r0 ^ r2);
13 }} ...
```

Listing 4: BMM by multiplying each A row with each B row

```
1 register unsigned Cm[64] = {0};
2 for (int i=0; i<A_width; i++){
3  unsigned long long a0 = Asub[i+A_height+laneid]); //1st half of A
4  unsigned long long a1 = Asub[i+A_height+WARP_SIZE+laneid]);//2nd half of A
5  unsigned long long b0 = Bsub[i+B_width+laneid]);//1st half of B
6  unsigned long long b0 = Bsub[i+B_width+WARP_SIZE+laneid]);//2nd half of B
7  #pragma unroll
8  for (int j=0; j<WARP_SIZE; j++){
9    unsigned long long l0 = __shfl(b0, j); //broadcast 64 bits
10   unsigned long long l1 = __shfl(b1, j); //broadcast 64 bits
11   Cm[j] += (__popcll(a0^l0)<<16) | __popcll(a1^l0); //dot-product and register packing
12   Cm[j+WARP_SIZE] = (__popcll(a0^l1)<<16) | __popcll(a1^l1); }}
```

Listing 5: 64-bits Bit-Matrix-Multiplication Kernel (BMM-64)

Having binarized the matrices, we are ready to perform BMM. To simplify the discussion, we first make three assumptions: *(a) A and B are bit matrices stored in row-major data format. (b) A and B are not transposed, and (c) the sizes of A and B are divisible by 32.* With these assumptions, we first partition A and B into 32×32 bit-blocks, corresponding to two BSTCs in two registers. Then, the block-wise BMM can be realized by fetching a bit-column of BSTC in register B, rotating anticlockwise by 90° via op-(H), then dot-producting with a bit-row of BSTC in register A via op-(A)/(B), and storing the result to C as a full-precision element (Figure 3). Listing 3 shows the major part of the code. Line 10 shows how to crop a bit-column from Bsub (B is row-major). Each lane contributes one bit per iteration, which is then dot-producted with bit-rows from Asub held by R0 of each lane.

Relaxing Assumption-(a): Consider that if B is stored in column-major data-format, we can avoid the bit-column cropping operation by broadcasting a bit-row of Bsub via __shfl() of op-(E). Listing 4 shows the code: if B is column-major, then rather than cropping a bit-column, we now only need to broadcast a bit-row of B per iteration. It is interesting to see that the format transition from row-major to column-major for B is equivalent to each 32×32 bit-tile in B undertaking a tile-wise transpose. Similarly, if A is stored in column-major, B in row-major, or if both A and B are stored in column-major, there are two more compositions. To summarize, there are four alternative approaches (one per combination of data formats):

- $A \rightarrow A^b_{col\text{-}major}, B \rightarrow B^b_{col\text{-}major}, BMM_{row\text{-}by\text{-}col} \rightarrow C$
- $A \rightarrow A^b_{col\text{-}major}, B \rightarrow B^b_{row\text{-}major}, BMM_{row\text{-}by\text{-}row} \rightarrow C$
- $A \rightarrow A^b_{row\text{-}major}, B \rightarrow B^b_{row\text{-}major}, BMM_{col\text{-}by\text{-}row} \rightarrow C$
- $A \rightarrow A^b_{row\text{-}major}, B \rightarrow B^b_{col\text{-}major}, BMM_{col\text{-}by\text{-}col} \rightarrow C$

where A^b refers to the binarization of A. We have evaluated all these four combinations and found that the second choice (Listing 4) shows the best performance (on a Tesla-P100, the delays are 28K, 20K, 28K, 56K cycles, respectively). An explanation is that, compared with __ballot(), communicating through __shfl() appears to be more efficient. Among the four options, only the second does not use __ballot().

Relaxing Assumption-(b): We find that transposition (A, B, or both) can be realized through a different composition of the data format (i.e., row-major, col-major) and BMM approaches (i.e., row-by-row, row-by-col, col-by-row, col-by-col). For example, if A is to be transposed before BMM, it is equivalent to binarizing A and B into row-major and performing row-by-row BMM. For each scenario of transposition (i.e., $A \times B$, $A \times B^T$, $A^T \times B$, $A^T \times B^T$), we measured all 16 combinations of data-formats and BMM approaches and found that row-by-row based tiled-BMM (in Listing 4) always outperforms the other three. Therefore, the optimal design summarizes as:

- $A \times B$: $A^b_{col\text{-}major}, B^b_{row\text{-}major}, BMM_{row\text{-}by\text{-}row} \rightarrow C$
- $A^T \times B$: $A^b_{row\text{-}major}, B^b_{row\text{-}major}, BMM_{row\text{-}by\text{-}row} \rightarrow C$
- $A \times B^T$: $A^b_{col\text{-}major}, B^b_{col\text{-}major}, BMM_{row\text{-}by\text{-}row} \rightarrow C$
- $A^T \times B^T$: $A^b_{row\text{-}major}, B^b_{col\text{-}major}, BMM_{row\text{-}by\text{-}row} \rightarrow C$

Relaxing Assumption-(c): In the case where matrix size is not a factor of 32 (e.g., in the output layer), padding is necessary. For full-precision GEMM, padding with zeros does not affect the correctness. However, for BMM, padding with zeros is equivalent to padding with -1s, so we need an approach to eliminate the effect of padding. Two important observations have been made: (1) Eq 2 essentially has a nice feature: if $\vec{a}$ and $\vec{b}$ are both padded with t bits and all the $2 \times t$ bits remain the same (either 0 or 1), it does not impact $popc(\vec{a} \oplus \vec{b})$. So as long as n is the actual length before padding (i.e., n rather than $n + t$), the padded bits will not affect the correctness of the result. (2) When writing to C, we must avoid writing outside the output matrix boundary.

BMM-64: The major differences between BMM-32 and BMM-64 include: (a) Although in BMM-64 the bitwidth becomes 64-bits, a warp still only has 32 lanes. Therefore, to process a 64×64 bit-block (corresponding to two BSTC-64), a warp has to process it twice: one round for each half of the bit-block (i.e., a BSTC-64). Segmentation and concatenation from/to one 64-bit datum to/from two 32-bit data are required. (b) In Line 3 of Listing 3 and 4, we use per-lane registers to temporarily buffer C's partial accumulation result Cm, with a demand of 1024 registers per warp. However, such a strategy cannot be applied to BMM-64 as it requires too many registers [49]. To address this issue, we propose *register packing*: 16 bits is already sufficient to hold C's partial results, as shown in Listing 5.

Extracting Fine-Grained Parallelism: In the current design, each warp processes a 32×32 or 64×64 subblock. However, the latest GPUs contain dozens of streaming multiprocessors (SMs). For example, in order for Tesla-V100 GPUs to be fully-loaded, with the current occupancy (i.e., 0.5 due to register constraints), it requires the matrix size to be at least 1620×1620 ($32\text{bits} \times \sqrt{80\text{SMs} \times 32\text{warps}} = 1620$) for BMM-32 and 3240×3240 for BMM-64. This is too much for small BNN fully-connected layers and can easily lead to workload imbalance, especially when the batch size is small. Therefore, we propose a lightweight batched version of BMM-32 and BMM-64, which spreads out the loop in Line 7-8 of Listing 4 and Listing 5 to different warps. This may sacrifice some register-level data reuse, but can effectively extract more fine-grained parallelism to feed all GPU SMs and alleviate potential workload imbalance. Such a more fine-grained design can bring significant speedups for small input matrices.

5.2 Bit Convolution (BConv)

For convolution, the basic operation is still bit dot-product. Again, we need to compact the data first. To be general, we adopt the following API as the interface of BConv, which is similar to Tensorflow's conv2d() API.

```
1 Bconv2d(x, W, stride_h, stride_w, padding, use64bit)
```

where x is the input image, which is a 4D tensor in shape *[batch, in_height, in_width, in_channels]* (or "*NHWC*"); W is the filter, a 4D tensor in shape *[filter_height, filter_width, in_channels, out_channels]*; *stride_w* and *stride_h* are the strides along height and width dimensions; padding option can be "*SAME*" or "*VALID*". The task of BConv is to binarize x and W into x^b and W^b with Eq 1, and convolve x^b with W^b to generate a full-precision 4D output image tensor in the same format as input *[batch, out_height, out_width, out_channels]*:

$$x^b[s,t] * W^b[s,t] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x^b[m,n] \cdot W^b[s-m, t-n]$$

For a high-performance implementation, identifying the optimal mapping from algorithm parallelism to hardware parallelism is essential, i.e., with the most data reuse. Regarding BConv, since the only connection between x and W is "in_channels", to gain the maximum spatial benefit, we binarize along "in_channels" (except for the input layer). Binarization is similar to BMM following Eq 1. In bit convolution, to gain the most fine-grained parallelism we use each warp to process one element of the output. Therefore, for an image of size 224×224 (batch=1) from ImageNet, we spawn $224 \times 224 \times 1 = 50176$ warps, which is sufficient to feed all SMs. Since every output image element is produced by dot-producting in size of [filter_height, filter_width, in_channels, out_channels], the hardware parallelism, which is warp_size × 32/64 bit = 1024/2048, needs to spread along filter_height×filter_width×in_channels×out_channels. For filter_height and filter_width, the difficulty is the need to handle padding at the boundary; these two dimensions are thus processed sequentially. The remaining issue then becomes how to spread 1024/2048 along in_channels×out_channels more efficiently.

First, let's suppose in_channel is a factor of 32 or 64, Listing 6 shows the BConv code. The design is a direct convolution: it does not follow the implementations of cuDNN [10] or Caffe [34], which flatten the input image first and then conduct GEMM [8]. Therefore, we do not require dozens of different strategies to efficiently handle different scenarios as done by cuDNN [10]. In addition, we do not require extra GPU memory as specialized workspace to buffer the flattened matrix, which can be huge in size. As shown in Listing 6, *out_height* is mapped to block-grid Y-dimension (Line 9); *out_width* is mapped to block-grid X-dimension (Line 8); *batch* is mapped to block-grid Z-dimension (Line 10), which is 1 (or very small) for inference. *filter_height* and *filter_width* are mapped to loop r and s, respectively, for sequential execution. For a single warp or thread block (following the warp-consolidation model), we map in_channels to the bit-width (32 or 64), and out_channels to warp-lanes (32). In Line 24-26, each warp fetches 32 coalesced full-precision data from the input image, binarizes them as an unsigned int (i.e., R0) and broadcasts R0 to all 32 lanes through __ballot(). Then in Line 28, each lane fetches 32bits (i.e., R1) with different out_channel ids from the binarized filter. After that, R0 and R1 are dot-producted in Line 29. If in_channel is larger than 32/64, we

```
1  template <typename T>
2  __global__ void Conv2d_32bit(const T* __restrict__ input, const unsigned* __restrict__ bitfilter ,
3  T* output, const int in_channels, const int out_channels, const int in_width,
4   const int in_height, const int filter_width, const int filter_height,
5   const int batch, const int stride_vertical, const int stride_horizontal,
6   const int out_width, const int out_height, const int pad_h, const int pad_w) {
7     unsigned laneid; asm("mov.u32 %0, %%laneid;":"=r"(laneid));//fetch lane-id in a warp
8     const int bx = blockIdx.x; //over out_width
9     const int by = blockIdx.y; //over out_height
10    const int bz = blockIdx.z; //over batch
11    const int ins = (in_channels >> 5);//number of steps in in_channels
12    extern __shared__ unsigned Csub[];
13    const int ax0 = bx*stride_horizontal-pad_w;//from output image idx to input image index
14    const int ay0 = by*stride_vertical-pad_h;
15    int exclude = 0; //track the number of filter entries that are masked off
16    for (int i=laneid; i<out_channels; i+=WARP_SIZE) Csub[i] = 0;
17    for (int r=0; r<filter_height; r++) { //load a window of data from Input
18      for (int s=0; s<filter_width; s++){
19          const int ay = ay0 + r; //y-coord in Input
20          const int ax = ax0 + s; //x-coord in Input
21          if ( (ay>=0) && (ay<in_height) && (ax>=0) && (ax<in_width) ) {//within Input frame
22              for (int c=0; c<ins; c++) {//per 32-bit in in_channels
23                  //Input[batch, in_height, in_width, in_channels]
24                  T f0 = input[bz*in_width*in_height*in_channels
25                      +(ay*in_width+ax)*in_channels+c*32+laneid];
26                  unsigned r0 = __brev(__ballot(f0>=0));//binarize
27                  for (int i=laneid; i<out_channels; i+=WARP_SIZE) {
28                    unsigned r1 = bitfilter [(r*filter_width+s)*ins*out_channels+c*out_channels+i];
29                    Csub[i] += __popc(r0 ^ r1);}
30          }}
31          else { //not in frame, so track
32              exclude++; //accumulate
33          }}}
34    for (int i=laneid; i<out_channels; i+=32)
35      output[(bz*out_height*out_width*out_channels) //batch
36            + (by*out_width*out_channels)//out_height
37            + (bx*out_channels) + i] //out_width
38            = in_channels*filter_width*filter_height //flatten matrix size
39              - exclude*in_channels //eliminate padding distoration
40              - (2*Csub[i]); } //n-2popc(a^b) for 0/1 to simulate sum(a*b) for +1/-1
```

Listing 6: 32-bits Bit Convolution Kernel (BConv-32)

iterate along in_channel (loop c in Line 22) as a step of 32/64. If out_channel is larger than warp_size, we iterate along out_channel (loop i in Line 27) at a step of warp_size. Note that writes to Csub in shared memory does not incur bank conflicts. Finally, at Line 33, we traverse out_channels by the whole warp to perform efficient coalesced writing.

Padding: The more interesting part of BConv lies in padding. When the padding option is "*VALID*", no padding is needed; but when it is "*SAME*", padding is required. As mentioned in Section 4.2, simply padding 0 is not feasible as 0 is already encoded for -1. For BMM, a unified boundary condition to exclude the padding elements solves the problem, but for BConv, it is more difficult: the input image and filter are not necessarily aligned. This is especially the case for the corner elements and when the stride is not 1. Divergent from existing CPU-based BConv implementations [31], which require pre-processing and/or post-processing, we handle the padding issue on the fly with no extra memory cost. As shown in Line 32, we declare an *exclude* variable to track when the convolving element falls out of the input image frame (Line 21), calculate the actual vector length (the padded bits should not be counted as effective bits, recall Eq 2 and Section 4.2), and make corresponding amendments (i.e., $-exclude*in\_channels$) when saving the results in Line 38-40. Note that this simple strategy is only feasible for direct convolution rather than the widely-used flatten&GEMM approach [8, 10, 31, 34]. This is because we need to convey the padding information of each element from the flatten kernel to the BMM kernel in order to distinguish a padding 0 from a normal 0. As a consequence, extra space cost and/or pre-/post-processing are inevitable.

BConv-64 BConv-64 is similar as BConv-32 except that each lane of a warp processes a 64-bit in_channel per iteration of c. In addition, we need two op-(G) for binarization and 64-bit op-(B) for bit accumulation.

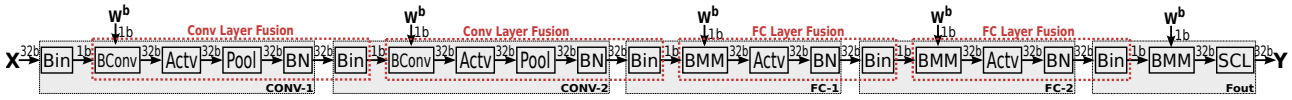

Figure 5: New Layer Partition for BNN LetNet Network.

6 SBNN INFERENCE IN A KERNEL

In this section, we present how to fuse the whole SBNN network model into a single GPU kernel. First, we discuss why it is possible for conducting all-layer fusion in our SBNN design. Then, we focus on presenting the intra-layer fusion strategy. Finally, we show how to fuse across layers.

6.1 Design Prerequisite

In this section, we propose the novel idea of fusing all DNN layers into a single GPU kernel. First, we will discuss some prerequisites for a successful all-layer fusion:

(1) Hardware support for GPU global synchronization. Prior to the Pascal architecture and CUDA Runtime-9.0, global synchronization among thread blocks is not feasible [52]. Recently, with the introduction of *cooperative-groups*, programmers can declare the grid-group through grid_group grid=this_grid() and use grid.sync() to synchronize across the kernel grid. To enable this functionality, a kernel has to be launched by cudaLaunchCooperativeKernel() and compiled with *-rdc=true* option [65]. Global synchronization is necessary for ensuring data consistency in our SBNN cross-layer fusion.

(2) Novel layer partition design for minimal cross-layer data movement. Figure 5 shows our proposed new layer partition method for SBNN, in comparison to the original design in Figure 3. The dotted-red-boxes mark the new LeNet layer boundaries, where we move binarization (i.e., Bin) of the next layer into the previous layer. In this way, we partition the SBNN network into two major types of layers: FC layers and Conv layers. This new partition approach brings the following benefits. First, only the data movement between Bin and BConv/BMM are binarized; others are full-precision types. Thus, the data write and read between layers can be minimized according to this layer partition. Second, although Actv, Pool, BN, Bin are all element-wise functions, BConv and BMM are not. As workload per Conv/FC layer can be significantly different, to achieve high occupancy for a small batch, we have to re-balance workload as evenly as possible among SMs per layer. As there is no inter-SM communication network, the image data forwarded across layers have to be stored and re-fetched per layer. To avoid potential memory dependency violations, a global synchronization is often required for SMs to stay synchronized.

(3) Unified GPU kernel configuration across fused layers. To fuse two GPU kernels, a basic requirement is that they share the same kernel configuration, i.e., *gridDim* and *blockDim*. "*gridDim*" is often related to input data (e.g, the volume of tasks), and "*blockDim*" is strongly correlated to programming models and occupancy, which may greatly impact performance. Traditional CUDA based designs assign each layer function (e.g., conv2d, gemm, pool, batch-normalization, etc) its individual ideal *blockDim* and *gridDim* for archiving the best performance for the targeted workload, resulting in multiple kernels. On the contrary, since our BSTC-based SBNN kernel design (e.g., BMM, BConv) follows the warp-consolidation model [45], *blockDim* is always 1D with 1 warp (or 32 threads). Regarding to *gridDim*, we adopt elastic kernel [66] or warp-delegation [50] to unify this value across layers. In this way, a unified GPU kernel can be formed across all the fused layers, avoiding multi-kernel launching and release. Specifically, we allocate the maximum number of warps (or thread blocks) the GPU resources can sustain simultaneously as elastic agents, which iteratively fetches the thread block jobs from a task list. A runtime throttling scheme on thread blocks is also provided to control concurrency for best performance.

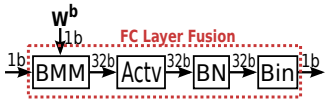

Figure 6: SBNN FC Layer Fusion

6.2 Intra-Layer Fusion

We show how to fuse the functions inside the red-dotted-boxes in Figure 5 into a single GPU kernel.

(a) FC Intra-Layer Fusion. Here, we adopt a backward fusing strategy, as shown in Figure 6. First, we fuse binarization (Bin) with batch-normalization (BN). BN has already been discussed in Eq (1). BN [33] is shown below:

$$y_{i,j} = \left(\frac{x_{i,j} - \mathbb{E}[x_{*,j}]}{\sqrt{Var[x_{*,j}] + \epsilon}} \right) \cdot \gamma_j + \beta_j \tag{3}$$

where $\mathbb{E}$ is the mean value across the batch. Var is the variance. ϵ is a small scalar to avoid zero division. γ and β are the learned scaling factor and bias. i iterates over the batch, j iterates over each output data element (i.e., output_channels×output_height×output_width). Combining Eq (3) and Eq (1), and given the fact that γ is a scaling factor being positive [1], we have

$$y_{i,j} = \begin{cases} 1 & \text{if } x_{i,j} \geq \mathbb{E}_{*,j} - \frac{\beta_j \cdot \sqrt{Var[x_{*,j}]+\epsilon}}{\gamma_j} \\ -1 & \text{otherwise} \end{cases} \tag{4}$$

where $y_{i,j}$ is the output of Bin which is in binary while $x_{i,j}$ is the input of BN which is in full-precision. Then, we integrate Actv. For DNN or BNN, Actv is often ReLU, or $y = max(x, 0)$. Therefore, the combination becomes

$$y_{i,j} = \begin{cases} -1 & \text{if } max(x_{i,j}, 0) < \mathbb{E}_{*,j} - \frac{\beta_j \cdot \sqrt{Var[x_{*,j}]+\epsilon}}{\gamma_j} \\ 1 & \text{otherwise} \end{cases} \tag{5}$$

where $y_{i,j}$ is the output of Bin which is in binary. $x_{i,j}$ is the output of BMM. FC intra-layer fusion is essentially to apply Eq 5 on each output of BMM before storing into binarized output matrix. The threshold $\mathbb{E}_{*,j} - \frac{\beta_j \cdot \sqrt{Var[x_{*,j}]+\epsilon}}{\gamma_j}$ is a floating-point number. But since $max(x_{i,j}, 0)$ in BNN is an integer, this offers opportunities to round up the threshold to an integer. Thus, most parameters in our implementation are integers, except the final output layer, where

[1] This is true for Tensorflow but not necessarily for PyTorch. In that case, we can inverse the corresponding channels of the weight to ensure γ converts to positive since bias=0 for BConv and BMM. Or, we can constraint γ to be always positive during training.

```
unsigned c=0; //amendment to BMM-32
if (bx*32+laneid<A_input){ //after padding, if height is in boundary
  for(int i=0; i<32; i++){//concatenate bits
    c <<=1;// left shift to leave space for the next bit
    if (by*32+i<B_width){//after padding, if width is in boundary
      int t = max( A_width - 2 * Cm[i], 0); //ReLU
      c |= (t < threshold[by*32+i]); }}} //Batch-normalization and Binarization
Csub[laneid] = c; // coalesced writing
```

Listing 7: FC Layer Fusion Amendment to BMM-32

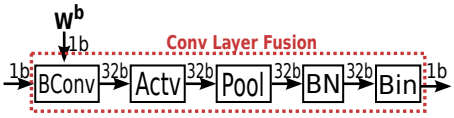

Figure 7: SBNN Conv Layer Fusion

```
__device__ __inline__ void ConvPool32Layer(Conv32LayerParam* p){
  extern __shared__ int Cs[];
  volatile int* Csub = (int*)&Cs;
  for (int bid = blockIdx.x; bid < (p->output_height)*(p->output_width)*(p->batch);
    bid += gridDim.x) {
    int bz = bid / ((p->output_width)*(p->output_height));
    int by = bid % ((p->output_width)*(p->output_height)) / (p->output_width);
    int bx = bid % ((p->output_width)*(p->output_height)) % (p->output_width);
    ... // Original BConv-32 Kernel Call
}
```

Listing 8: Warp-Delegation for SBNN Inter-Layer Fusion

no further binarization can be fused. Listing 7 shows the necessary amendments to the original BMM-32 in Listing 4 when integrating Actv, BN and Bin. Also, the output is binarized data, reducing data writes by 32x.

(b) Conv Intra-Layer Fusion. Convolution intra-layer fusion is similar to FC, except that we have to integrate the pooling function Pool, as shown in Figure 7. Pooling function is typically to pick the maximum or minimum value among a $m \times m$ tile, where m is generally 2. If we move the Pool function after Bin, their combination essentially offers a new opportunity for *branch-pruning*: for max-pooling, if one of the 2×2 elements is shown to be 1 after going through BN and Bin, the computation of the remaining 3 elements in the pooling window can be pruned since Bin already has its maximum possible value of 1. Similar condition applies to min-pooling with value 0.

Since Pool is not strictly element-wise function, one approach is to perform a 2×2 thread coarsening. In other words, each warp handles 4 output elements sequentially. Thus, max-pooling can be achieved by

```
output[((bz*output_height+(by/2))*output_width +(bx/2))*(output_channel/32)+k] |= C;
```

Alternatively, to extract the most fine-grained parallelism, we still spread out a warp for each output image before pooling. To process pooling, we use bitwise OR and AND for max/min pooling when writing binarized results to the pooled output image. However, as the 2×2 data elements are processed by 4 independent warps, a race condition may occur. We apply atomicOr() and atomicAnd() before writing to output image (the latter option is often much faster than the former), e.g.,

```
atomicOr(output[((bz*output_height+(by/2))*output_width
  +(bx/2))*(output_channel/32)+k], C);
```

6.3 Inter-Layer Fusion

Now we are ready to fuse all the layers into a single kernel. Since BMM and BConv functions have been designed based on the warp-consolidation model, *blockDim* for all the layers are already unified to 32. Here we show how to unify *gridDim* across layers via elastic

```
#include <cooperative_groups.h>
// ================================ Inference Network ================================
__global__ void LeNet(InConv32LayerParam* bconv1,Conv32LayerParam* bconv2,Fc32LayerParam* bfc1,
  Out64LayerParam* bout){
  grid_group grid = this_grid();
  InConvPool32Layer(bconv1); //1st Convolution Layer with Pooling
  ConvPool32Layer(bconv2); //2nd Convolution Layer with Pooling
  Fc32Layer(bfc1); //1st Fully-Connnect Layer
  Out32Layer(bout);} //2nd Fully-Connnect Layer
// ===================================================================================
int numBlockPerSm; //maximum number of thread-blocks per SM under current occupancy
int numThreads = 32; //warp size following warp-consolidation model
cudaDeviceProp deviceProp;
cudaGetDeviceProperties(&deviceProp, GPU_dev); //GPU_dev is GPU device id
cudaOccupancyMaxActiveBlocksPerMultiprocessor(&numBlockPerSm, LeNet,
    numThreads, shared_memory);
void* args[] = {&bconv1, &bconv2, &bfc1, &bout};//set kernel function parameters
cudaLaunchCooperativeKernel((void*)LeNet, numBlockPerSm*deviceProp.multiProcessorCount,
  numThreads, args, shared_memory);//kernel invokation
```

Listing 9: SBNN Singular Kernel for LeNet Network

kernel [66] and warp-delegation [50]. Listing 8 shows an example on BConv-32 with pooling. We have developed a corresponding parameter object *Conv32LayerParam* to pack all the required parameters of that particular layer. In Line 4-5, we use all the allocated thread blocks to traverse the original *gridDim* space and calibrate the thread block id to its dedicated task in the original space (Line 6-8). After that, we invoke the original BConv-32 kernel for processing. In this way, all the layer kernels can have the identical *gridDim*, getting ready for fusion.

We then fuse all the layers into a single kernel, as shown in Listing 9. This kernel is to be written by the users to describe the network structure, with one parameter object per layer. "grid.sync()" is inserted at the end of each layer function as a global barrier. As can be seen, the fused kernel is very easy to write from users' perspective: an appropriate parameter object for each layer can be simply declared, followed by constructing the __global__() function as Listing 9. To increase hardware occupancy, by default we allocate the maximum number of thread blocks allowed under present resources consumption in Line 15. However, users can tune this value as needed. The kernel function parameters are configured in Line 17 and the fused-kernel is invoked in Line 18-19.

There are four design considerations in our fused approach. (1) After our new layer partition approach is applied (Section 5.1), the image data to be written by the former layer and immediately read by the following layer are condensed bit-data, which is possible to fit into on-chip memory, e.g., registers and shared memory (e.g., in Line 2-3 of Listing 8, Cs is shared by all the layers). Ideally, there is no global memory image data read and write in each layer. However, one complexity is that BMM and BConv may have data dependency across SMs (one SM may need data from another SM). Unless data are written into the global memory, SMs cannot share on-chip data – there is no inter-SM network for current GPUs. Thus, some global write and read as well as synchronization are necessary. Nevertheless, the overhead should be relatively small. This is because the volume of data has already been reduced by 32×; and since read is often immediately followed by write (of the same image-data), L2 cache rather than off-chip global memory is leveraged for buffering. So the data is essentially still kept on-chip. In fact, we observe very high L2 cache hit rate for the fused approach. (2) The first layer is the input layer, in which the input image is in full-precision and in_channel is generally small (e.g. 1 for gray image and 3 for RGB). In addition, for a large network, the input image is not binarized in the first layer to avoid significant information lost. In our fused design, we have developed special parameter objects and layer implementation for the input and output layers (i.e., the input layer

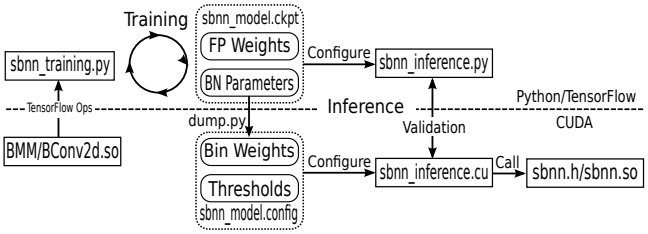

Figure 8: Framework for SBNN training and inference.

is typically convolution while the output layer is generally fully-connected). (3) For the fully-connected layers, the input image is binarized in column-major (weight in row-major) while for convolution layer, the input image is binarized along in_channel. Therefore, to connect the last convolution layer with the first fully-connected layer, the bit image data has to be transposed. We set a configurable option for the convolution layer to correctly write to the output image (i.e., transpose during writing) so that such transposition will not lead to additional overhead. (4) For newer network structures such as ResNet [28], we need to save the full-precision convolution result before binarization for subsequent merging.

6.4 SBNN Inference Framework

For validation and wide adaption, we have developed a framework to train an SBNN and configure its weights & thresholds for inference. As shown in Figure 8, we train SBNN in *TensorFlow*. We have encapsulated BMM-32/64 (Listing 4) and BConv-32/64 (Listing 6) as a dynamic library that can be called as TensorFlow operations [26] during training. The intention is to ensure that the behavior of training forward-pass is exactly the same as our SBNN inference. After an SBNN network is trained, a script *dump.py* is employed to extract the weights and batch-normalization parameters (i.e.,$Var, \mathbb{E}, \beta, \gamma$) from the TensorFlow dumped checkpoints (e.g., *sbnn_model.ckpt*), calculate the thresholds, and save weights & thresholds information into an SBNN configuration file (e.g., *sbnn_model.config*). The configuration file is then parsed by SBNN inference program during initialization. We have assigned a certain format for the configuration file and provided APIs to parse such a file in C/CUDA. For validation and debugging purposes, we also develop a TensorFlow inference script in addition to the training Python script (Figure 8). We have also developed a similar framework for *PyTorch*. There are two differences: (i) the image tensors from TensorFlow are in *NHWC* format, while from PyTorch are in *NCHW* format, so when converting from CONV layers to FC layers, a permutation may be required to obtain the correct weights for SBNN inference; (ii) in PyTorch, we observe that occasionally the trained scaling factor in batch-normalization (i.e., γ_j in Eq (3)) can be negative, which may bring trouble in the threshold method (i.e., Eq (5)). Therefore, in PyTorch training, we enforce γ_j to be positive.

7 EVALUATION

We evaluate our design of BMM, BConv, and SBNN on four state-of-the-art GPU platforms: an HPC-oriented NVIDIA Tesla-P100, V100, and edge-oriented Jetson-TX1 and TX2, as listed in Table 1. It is expected that the Tesla platforms will be used for cloud or HPC inference while the Jetson platforms will be used for edge inference in an embedded environment.

7.1 Bit-Matrix-Multiplication

First, we measure the latency of BMM with matrix size scaling from 64×64 to 32768×32768. We compare our design against *cuBLAS* (i.e., simulating +1/-1 BMM through full-precision SGEMM) and *bnn-baseline* (i.e., the BMM design from the original BNN work [12]). Note that the cuBLAS scheme does not include binarization. Besides, *bnn-baseline* implementation can only run correctly when the matrix size is a factor of 512 and it does not support transposition. There are 8 different schemes to be evaluated from this study, described in Table 2: *BMM-32* (the 32bit BMM in Listing 4), *BMM-64* (the 64bit BMM in Listing 5), *BMMS-32* (the variant of BMM-32 by extracting fine-grained parallelism, as discussed in Section 4.1), *BMMS-64* (the variant of BMM-64), *BMM-32B* (BMM-32 with its two input matrices already binarized before processing and its output matrix binarized before return; i.e., both inputs and output matrices are bit matrices, which is the condition of SBNN inference. Similarly, *BMM-64B*, *BMMS-32B*, and *BMMS-64B* are described in Table 2. We validate the correctness of results by comparing against cuBLAS. The reported figures are the average of multiple runs.

Figures 9, 10, 11 and 12 illustrate the measured latencies of BMM with increased matrix size on the four platforms in Table 1. In order to fit the curves into a single figure, we normalized the latency with respect to *cuBLAS*. On the Maxwell-based Jetson-TX1, *BMMS-64* and *BMMS-64B* failed to run due to insufficient hardware resources. Overall, *BMM-32B* on the P100 GPU shows the best performance, delivering ~ 10× speedup over cuBLAS simulated BMM, with 1K matrix size. In general, the 32-bit BMM schemes show better performance than their 64-bit counterparts. This is the trade-off [46, 51] between *workload balancing* for *data reuse* in registers. The 64-bit BMM achieves better data reuse than the 32-bit BMM, but when the matrix size is small or medium, it suffers from insufficient or unbalanced workload among SMs. The 64-bit BMM starts to show an advantage with large matrices (e.g., ≥ 16K×16K). The fine-grained versions, although they do not show an obvious advantage over the coarse-grained version here, appear to give significantly better performance in SBNN inference, especially for the 64-bit versions (as will be seen in Section 6.3). We have also evaluated BMMs with transposition. The results show that the transposition overhead is negligible since it is just an alternative selection of data format (see Section 3). If the matrix size S is not a factor of 32 for 32-bit BMM, or 64 for 64-bits BMM, padding is required and some computation will be wasted; the performance is then the same as when the matrix size is $\lceil S/32 \rceil \times 32$ or $\lceil S/64 \rceil \times 64$.

7.2 Bit Convolution

We then measure the performance of BConv. In contrast with matrix-multiplication, convolution needs many more parameters: *input_size*, *filter_size*, *input_channel*, *output_channel*, *batch*, *stride* and possibly *pooling*. Due to page limitation, we only show the curves with respect to input_channel and output_channel – these are frequently varied with layers of the same network. We set *input size*=64 (medium image size), *filter size*=3 (most frequently used), *stride*=1 (most frequently used), and *batch*=1 (for inference). Evaluation with respect to other parameters (e.g., batch, input_size, filter_size) are provided in the supplementary file. Since there is

Table 1: GPU SM resource configuration. "Reg" refers to the number of 4B register entries.

GPU	Arch	CC.	Freq.	Rtm.	SMs	CTAs/SM	Warps/SM	Warps/CTA	Reg/SM	Reg/CTA	Reg/Thd	Shared/SM	Shared/CTA
Tesla-P100	Pascal	6.0	1481 MHz	9.1	56	32	64	32	64K	64K	255	64KB	48KB
Tesla-V100	Volta	7.0	1530 MHz	9.0	80	32	64	32	64K	64K	255	96KB	96KB
Jetson-TX1	Maxwell	5.3	998MHz	9.0	2	32	64	32	64K	32K	255	64KB	48KB
Jetson-TX2	Pascal	6.2	1301MHz	8.0	2	32	64	32	64K	32K	255	64KB	48KB

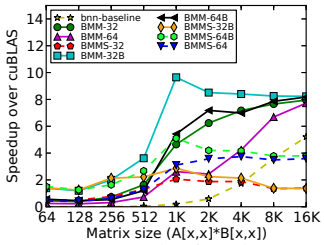

Figure 9: BMM speedups for Tesla-P100.

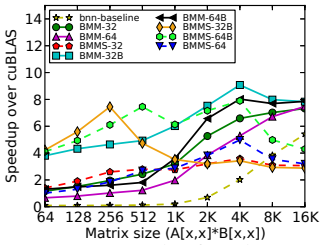

Figure 10: BMM speedups for Tesla-V100.

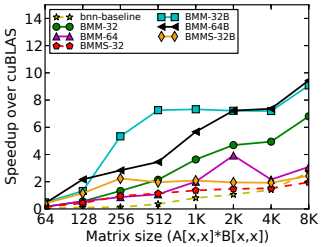

Figure 11: BMM speedups for Jetson-TX1.

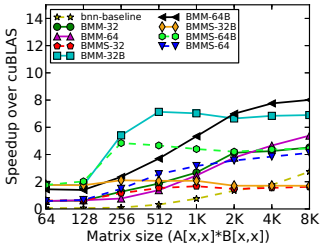

Figure 12: BMM speedups for Jetson-TX2.

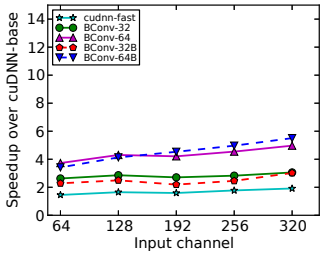

Figure 13: BConv speedups on Tesla-P100.

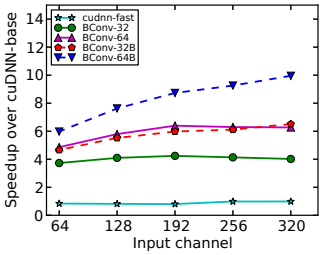

Figure 14: BConv speedups on Tesla-V100.

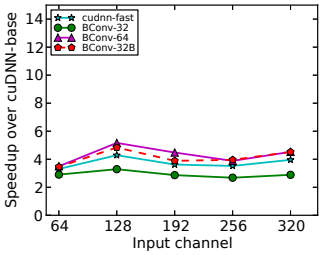

Figure 15: BConv speedups on Jetson-TX1.

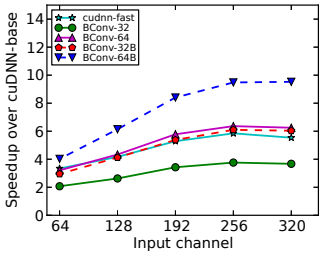

Figure 16: BConv speedups on Jetson-TX2.

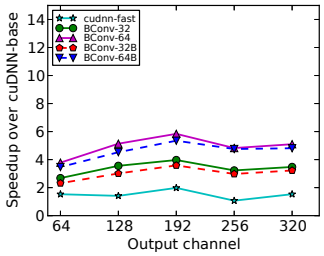

Figure 17: BConv speedups on Tesla-P100.

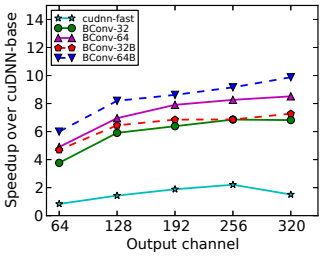

Figure 18: BConv speedups on Tesla-V100.

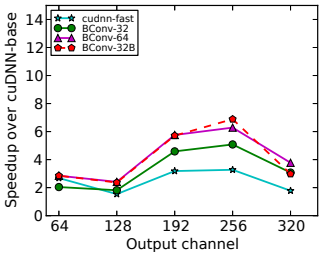

Figure 19: BConv speedups on Jetson-TX1.

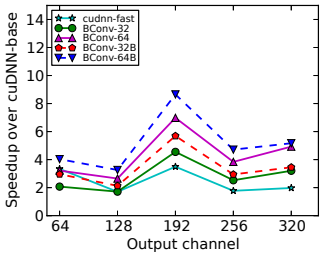

Figure 20: BConv speedups on Jetson-TX2.

Table 2: Performance Comparison. Input and Output refer to bit-width per data element for input data and output data.

Schemes	Description	Algorithm	Input	Output
cuBLAS	Simulating BMM via SGEMM	SGEMM	32bit	32bit
bnn-baseline	BMM from the BNN paper [12]	BMM	32bit	32bit
BMM-32	The 32bit BMM design	BMM	32bit	32bit
BMM-64	The 64bit BMM design	BMM	32bit	32bit
BMMS-32	Fine-grained parallelism BMM-32	BMM	32bit	32bit
BMMS-64	Fine-grained parallelism BMM-64	BMM	32bit	32bit
BMM-32B	Bit input/output BMM-32	BMM	1bit	1bit
BMM-64B	Bit input/output BMM-64	BMM	1bit	1bit
BMMS-32B	Fine-grained parallelism BMM-32B	BMM	1bit	1bit
BMMS-64B	Fine-grained parallelism BMM-64B	BMM	1bit	1bit

no existing BConv design for GPUs, we compare our approaches against FP-32 cuDNN simulated BConv.

There are two major implementations for convolution in cuDNN: the basic one without any workspace utilization (i.e., CUDNN_CONVOLUTION_FWD_NO_WORKSPACE, marked as *cuDNN-base*) and a fast one demanding large workspace (i.e., CUDNN_CONVOLUTION_FWD_PREFER_FASTEST, marked as *cuDNN-fast*). Our implementation includes four versions: *BConv-32* and *BConv-64* are 32-bit and 64-bit BConv with input, filter and output in 32-bit full-precision. *BConv-32B* and *BConv-64B* are 32-bit and 64-bit BConv with input, filter and output in bits (so, we avoid the binarization of input and filter, but add the binarization of output).

The results are normalized to *cuDNN-base*, shown in Figures 13, 14, 15, and 16 for *input-channels* (output_channel=64), and in Figures 17, 18, 19, and 20 for *output_channels* (input_channel=64) on the four platforms (Table 1), respectively. As with BMM, the 64-bit version cannot run properly on the Maxwell-based Jetson-TX1 platform due to resource limitations. From these figures, we can observe that, unlike those in BMM, the 64-bit BConv versions outperform the 32-bit versions. This is mainly because the BMM compacts bits along rows or columns of the 2D matrix, but BConv compacts bits along input or output channels of the 4D tensors, where more data reuse can be achieved through larger parallel granularity. For BMMs, the workload balance with smaller granularity outweighs the benefit from increased data reuse with larger granularity; but for BConv, this is reversed. As can be seen, our *BConv-64B* achieves a 10× speedup over the cuDNN implementation on the latest Volta GPU when input/output_channel = 320. In case the input or output_channel is not a factor of 32/64, padding is required when performing the binarization. This may sacrifice some performance, especially when the image_size and/or filter_size is large. However, most modern DNN input/outpu_channels are factors of 32/64 (e.g., AlexNet, VGG, and ResNet).

7.3 SBNN Inference

We evaluate our SBNN inference framework on three different datasets using three different network structures, as listed in Table 3. For MLP on MNIST, the input image for the first layer is also

Table 3: SBNN Inference Evaluation. "1024FC" refers to a fully-connected layer with 1024 neutrons. "2x128C3" refers to 2 convolution layer with 128 output channels and 3x3 filter. "MP2" refers to a 2x2 pooling layer with stride=2. "128C11/4" refers to a convolution layer with 128 output channels, 11x11 filter size and stride=4. "Input size" is in the format of *input_height*×*input_width*×*input_channels*. "Output" is the number of categories for the classification. "Ref" is short for references. "BNN" refers to state-of-the-art BNN training accuracy from existing work. "Our BNN" is the SBNN training accuracy we obtained from our own training implementation. Note, this is training accuracy, which is irrelevant to inference. Our SBNN does not cause accuracy loss once a trained BNN model is given. "Full-Precision" is the reference FP32 training accuracy.

Dataset	Ref	Network	Ref	Network Structure	Input Size	Output	BNN	Our BNN	Full-Precision
MNIST	[42]	MLP	[12]	1024FC-1024FC-1024FC-1024FC	$28 \times 28 \times 1$	10	98.6% [12]	97.6%	99.1% [12]
Cifar-10	[38]	VGG	[11]	(2x128C3)-MP2-(2x256C3)-MP2-(2x512C3)-MP2-(3x1024FC)	$32 \times 32 \times 3$	10	89.9% [12]	88.7%	90.9% [14]
ImageNet	[17]	AlexNet	[39]	(128C11/4)-MP2-(256C5)-MP2-(2x384C3)-(256C3)-MP2-(3x4096FC)	$224 \times 224 \times 3$	1000	75.7/46.1% [14]	74.2/44.7%	80.2/56.6% [14]
Cifar-10	[38]	ResNet-14	[70]	128C3/2-4x128C3-4x256C3-4x512C3-(2x512FC)	$32 \times 32 \times 3$	10	N/A	91.6%	N/A
ImageNet	[17]	ResNet-18	[70]	64C7/4-4x64C3-4x128C3-4x256C3-4x512C3-(2x512FC)	$224 \times 224 \times 3$	1000	73.2/51.2% [70]	72.7%/48.6%	89.2/69.3% [70]

Table 4: SBNN Inference Performance on NVIDIA Pascal-based P100 GPU

	MNIST-MLP				Cifar10-VGG				ImageNet-AlexNet			
Schemes	**Raw Latency**	**Speedup**	**Throughput**	**Speedup**	**Raw Latency**	**Speedup**	**Throughput**	**Speedup**	**Raw Latency**	**Speedup**	**Throughput**	**Speedup**
SBNN-32	0.277ms	1678×	3.16×10^6	1090×	1.502ms	76.3×	2.53×10^4	34.5×	3.238ms	674×	1615.3	6.6×
SBNN-32-Fine	0.084ms	5533×	1.20×10^6	416×	1.298ms	88.3×	2.47×10^4	41.5×	2.778ms	786×	1530.8	6.2×
SBNN-64	1.007ms	462×	1.19×10^5	41.2×	1.556ms	73.7×	2.81×10^4	38.4×	13.985ms	156×	1449.2	5.9×
SBNN-64-Fine	0.076ms	6115×	4.06×10^6	1410×	0.611ms	188×	3.73×10^4	50.8×	1.528ms	1429×	1910.3	7.8×
TensorFlow	464.74ms	1.0×	2.89×10^3	1×	114.65ms	1×	733.13	1×	2183.62ms	1×	245.83	1×

Table 5: SBNN Inference Performance on NVIDIA Volta-based V100 GPU

	MNIST-MLP				Cifar10-VGG				ImageNet-AlexNet			
Schemes	**Raw Latency**	**Speedup**	**Throughput**	**Speedup**	**Raw Latency**	**Speedup**	**Throughput**	**Speedup**	**Raw Latency**	**Speedup**	**Throughput**	**Speedup**
SBNN-32	0.183ms	1266×	4.39×10^6	1010×	0.994ms	132×	5.15×10^4	60.4×	2.226ms	523×	4.19×10^3	14×
SBNN-32-Fine	0.04ms	5790×	3.32×10^6	762×	0.833ms	157×	5.10×10^4	59.8×	1.576ms	739×	3.95×10^3	13.2×
SBNN-64	0.896ms	259×	1.88×10^5	43×	1.395ms	94×	4.22×10^4	49.5×	9.134ms	128×	2.87×10^3	9.6×
SBNN-64-Fine	0.04ms	5790×	8.98×10^6	2060×	0.466ms	281×	5.78×10^4	67.9×	0.979ms	1190×	4.40×10^3	14.7×
TensorFlow	231.6ms	1×	4.36×10^3	1×	131.09ms	1×	851.49	1×	1164.95ms	1×	298.44	1×

Table 6: SBNN ResNet Inference Performance on NVIDIA Pascal-based P100 GPU

	Cifar10-ResNet14				ImageNet-ResNet18			
Schemes	**Latency**	**Speedup**	**Throu**	**Speedup**	**Latency**	**Speedup**	**Throu**	**Speedup**
SBNN-32	4.255ms	227×	3.36×10^3	6.29×	5.898ms	157×	4.74×10^2	2.83×
SBNN-32-Fine	4.187ms	230×	3.49×10^3	6.54×	5.769ms	161×	4.80×10^2	2.86×
SBNN-64	1.924ms	501×	7.06×10^3	13.2×	3.485ms	266×	6.62×10^2	3.95×
SBNN-64-Fine	1.330ms	725×	7.19×10^3	13.5×	2.754ms	337×	6.65×10^2	4.0×
TensorFlow	964.777ms	1×	5.34×10^3	1×	927.4ms	1×	1.68×10^2	1×

Table 7: SBNN ResNet Inference Performance on NVIDIA Volta-based V100 GPU

	Cifar10-ResNet14				ImageNet-ResNet18			
Schemes	**Latency**	**Speedup**	**Throu**	**Speedup**	**Latency**	**Speedup**	**Throu**	**Speedup**
SBNN-32	2.784ms	342×	1.13×10^4	14.67×	3.293ms	285×	2.03×10^3	9.27×
SBNN-32-Fine	2.700ms	353×	1.23×10^4	15.94×	3.165ms	296×	1.94×10^3	8.85×
SBNN-64	1.584ms	602×	1.88×10^4	24.5×	2.532ms	370×	2.53×10^3	11.6×
SBNN-64-Fine	1.027ms	928×	2.04×10^4	26.5×	1.906ms	492×	2.88×10^3	13.2×
TensorFlow	953.294ms	1×	7.69×10^2	1×	937.11ms	1×	218.7	1×

binarized in both training and inference. This explains the slight accuracy degradation on our BNN training accuracy (98.6% to 97.4%). For VGG on Cifar10 and AlexNet on ImageNet, the input image of the first layer is in full-precision, but the weight is binarized, similar to BWN. This is due to accuracy consideration. Since there are only 3 input channels, the overhead of the first layer adopting full-precision input is not very high. Divergent from the original BNN design, the final output layer in our implementation is also binarized for both input and weight, followed by a batch normalization. We find the BN here, also cannot be aggregated as a threshold comparison as before (there is no further binarization), is vital to the high training accuracy for large dataset. Overall, after a training period of 2000, 2500 and 500 epochs for MLP, VGG and AlexNet, we achieve comparable or even superior BNN training accuracy with state-of-the-art BNN and full-precision DNN training accuracy. This demonstrates that our intra- and inter-layer fusion techniques will not affect the accuracy of the training process.

Table 4 and 5 list the inference performance results for the three networks (i.e., MLP on MNIST, VGG-like on Cifar10, and AlexNet on ImageNet) on Tesla-P100 and V100. *SBNN-32/64* refer to the 32/64-bit single-kernel SBNN inference. *SBNN-32/64-Fine* refers to the implementation that the FC layers are using fine-grained 32-bit BMM (i.e., *BMMS-32B*). The TensorFlow results which simulate BNN in 32-bit single-precision floating-point (FP32) via cuBLAS and cuDNN on the same GPUs are used as reference (which is also the general scenario for BNN algorithm design and validation), since to the best of our knowledge, there is no mature GPU-based BNN implementation. As can be seen in Table 4 and 5, our single-kernel SBNN inference achieves up to 6115× speedup over the TensorFlow baseline on P100 with a small dataset (i.e., MNIST) and 1429× speedup with a large dataset (i.e., ImageNet) for a single image BNN inference. An inference delay of 40μs for a small network and less than 1ms for a large network are sufficient to satisfy the timing constraints for most latency-critical DNN applications in the real world. We can also observe that the throughput (i.e., *images per second* or *img/s*) improvement does not match the significant raw latency reduction. In fact, for AlexNet on ImageNet, the throughput improvement is only about an order, implying that GPU is extremely underutilized with non-batched inference in current TensorFlow. Our design dramatically enhances GPU utilization (but not in a traditional way as no floating-point computation is involved). Comparing among various implementations, SBNN-64-Fine outperforms the others, demonstrating the optimal performance among all conditions. This is mainly because: (1) SBNN-64 has a better register data reuse thus a higher computation-to-memory ratio [46, 51] than SBNN-32; (2) For small-batched inference, extracting more fine-grained parallelism from the GPU kernel to feed all GPU SMs is clearly more crucial than better data reuse or cache efficiency.

Table 6 and 7 list the performance for more modern Residual neural network [28] on Tesla-P100 and V100. The implementation for ResNet-14 on Cifar10 and ResNet-18 on ImageNet follow existing ResNet implementation for BNNs [14, 32, 70]. As can be seen, SBNN achieves comparable speedups for more recent network models with residuals and shortcuts.

Table 8 and 9 further compare SBNN with other existing BNN works on various alternative platforms, such as FPGAs, CPU, and

Table 8: Comparing SBNN with FPGA works using AlexNet on ImageNet.

AlexNet/ImageNet	Platform	Raw Latency	Throughput
RebNet [25]	Xilinx Virtex VCU108 FPGA	1902 μs	521 img/s
FP-BNN [54]	Intel Stratix-V FPGA	1160 μs	862 img/s
O3BNN [24]	Xilinx Zynq ZC706 FPGA	774 μs	1292 img/s
SBNN	NVIDIA Tesla V100 GPU	979 μs	4400 img/s

Table 9: Comparing SBNN with CPU, GPU and FPGA using VGG-16 on ImageNet.

Vgg-16/ImageNet	Platform	Raw Latency	Throughput
BitFlow [31]	NVIDIA GTX1080	12.87 ms	78 img/s
BitFlow [31]	Intel i7-7700 HQ	16.10 ms	62 img/s
BitFlow [31]	Intel Xeon-Phi 7210	11.82 ms	85 img/s
O3BNN [24]	Xilinx Zynq ZC706 FPGA	5.626 ms	178 img/s
SBNN	NVIDIA Tesla V100 GPU	3.208 ms	312 img/s

Xeon-Phi, using AlexNet and VGG-16 on imageNet. As can be seen, for AlexNet, SBNN shows better or comparable raw latency, and much better throughput than specialized hardware accelerators using FPGAs. For VGG-16, SBNN shows much better latency and throughput than deeply-optimized (with vectorization and multi-core enabled [31]) CPU, GPU and Xeon-Phi, demonstrating the effectiveness of our BSTC and SBNN design.

8 DISCUSSION

In this section, we further discuss BNN accuracy, potential use cases for the SBNN inference framework, other optimizations we adopted in SBNN and future work.

BNN Accuracy: Despite the fact that the training accuracy of BNNs has improved dramatically during the past three years, there is still a gap with full precision methods (due to reduced model capacity and discrete parameter space). Recently the boosting and bagging methods have been applied to BNNs, enabling them to achieve accuracy comparable to full-precision networks with the same model structure (AlexNet Top-1: BENN6:54.3% vs DNN:56.6%, ResNet18 Top-1: BENN6:61.0% vs DNN: 69.3%) [80]. In addition, the BNN components of an EBNN can be parallel and independently executed by multiple GPUs of a modern scale-up GPU platform [47, 48] for advanced performance. On the other hand, searching a discrete space for global optimal parameters for BNN training is NP-hard. Current approaches focus on how to better converge to an appropriate discrete point from the continuous space in full-precision network training [14, 32, 40, 70, 74]. For this task, quantum annealing may play a role [3].

Practical Use Cases: Despite some accuracy loss, there are scenarios where other factors such as latency, cost, and power are critical, as long as a certain accuracy can be maintained. BNNs have the interesting attribute that they can be analyzed formally and verified through Boolean Satisfiability and Integer Linear Programming. This means that formal guarantees can be provided on certain properties of the network (defined by the user) [9, 37, 61, 62, 62]. This guarantee is important for security related contexts [22, 44].

With certain accuracy bar being surpassed, there are many use cases for SBNN (in addition to edge computing). For example, for cloud warehouse computing, DNN inference is integrated as a microservice [27, 35] for web applications (e.g., image processing, speech recognition, natural language processing, etc.). As a computationally efficient component executed with rigid latency constraint, BNNs can be an alternative choice. This is particularly the case when accuracy is not emphasized, such as with ad recommendations, fuzzy search, and indexing. For HPC, an ideal use case is online/offline processing of raw data. For example, since the 1990s, shallow neural networks have been widely used in high-energy physics (HEP) [15, 69], including for event classification [1, 36], object reconstruction [68], and triggering [16, 57]. The last of these is a particularly attractive application for BNNs: triggering (deciding whether an event is of interest or not) uses a huge amount of data (collected online by sensors/simulations or offline in huge data collections, PBs per observation) and accuracy is ensured by overall probability rather than single-event instance. These general methods are applicable more broadly in other HPC domains requiring preprocessing of large amounts of raw data, including those found in astronomy, physics, chemistry, biology, and environmental sciences.

Other Optimizations: we discuss some optimizations adopted in the designs: (i) *Cache bypassing* [53]. For BConv, when batch-size=1 and stride=1, the filter shows less intra-thread data reuse than input. Therefore, we can bypass the filter fetching in order to leave more cache space for input data. (ii) *Register compaction*, as already discussed in Section 5.2 and Listing 5. (iii) *Shared memory compaction*. We leverage shared memory to temporarily buffer partial output-channel results. When output-channel is large, shared memory may fall short to reach the maximum theoretical occupancy. To address this, we can declare shared memory output-channel buffer as *short int* rather than *int* to save space and achieve a higher occupancy, at the cost of a few shared memory bank conflicts. (iv) *Offline binarization for weights and filters*. Note that weights and filters are learned and binarized offline; it is not in the critical path of inference.

Future Work: we will mainly pursue three directions: (i) accelerating SBNN training process on GPUs; (ii) exploring state-of-the-art multi-GPU platforms such as DGX-1/2 on accelerating complex BNN models, particularly BENN [80]; (iii) we recently learnt that the *TensorCores* in the latest NVIDIA Turing GPUs start to support bit types and potentially bit computations, so it will be interesting to see how SBNN can be accelerated by these TensorCores.

9 CONCLUSION

In this paper, we propose binarized-soft-tensor-core to build a strong bit-manipulation capability for modern economy-of-scale general-purpose GPU architectures. We use BSTC to accelerate bit functions such as bit-packing, bit-matrix-multiplication, and bit-convolution of binarized-neural-networks. To further accelerate BNN inference, we propose intra- and inter-layer fusion techniques that can merge the whole BNN inference process into a single GPU kernel. For non-batched BNN inference tasks, our design demonstrates more than three orders of magnitude latency reduction compared with the state-of-the-art full-precision simulated BNN inference on GPUs.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable comments. This research was supported by PNNL's DeepScience-HPC and DMC-CFA LDRD projects. This research was supported by the U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, under award 66150: "CENATE - Center for Advanced Architecture Evaluation". This research was supported by PNNL's High Performance Data Analytics (HPDA) program.

REFERENCES

[1] Paulo Abreu, W Adam, T Adye, E Agasi, GD Alekseev, A Algeri, P Allen, S Almehed, SJ Alvsvaag, U Amaldi, et al. 1992. Classification of the hadronic decays of the Z0 into b and c quark pairs using a neural network. *Physics Letters B* 295, 3-4 (1992), 383–395.

[2] Saman Ashkiani, Martin Farach-Colton, and John D Owens. 2018. A dynamic hash table for the GPU. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 419–429.

[3] Carlo Baldassi and Riccardo Zecchina. 2018. Efficiency of quantum vs. classical annealing in nonconvex learning problems. *Proceedings of the National Academy of Sciences* (2018), 201711456.

[4] Pierre Baldi, Kyle Cranmer, Taylor Faucett, Peter Sadowski, and Daniel Whiteson. 2016. Parameterized machine learning for high-energy physics. *arXiv preprint arXiv:1601.07913* (2016).

[5] Tal Ben-Nun and Torsten Hoefler. 2018. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. *arXiv preprint arXiv:1802.09941* (2018).

[6] Eli Ben-Sasson, Matan Hamilis, Mark Silberstein, and Eran Tromer. 2016. Fast multiplication in binary fields on GPUS via register cache. In *Proceedings of the 2016 International Conference on Supercomputing*. ACM, 35.

[7] Benjamin Block, Peter Virnau, and Tobias Preis. 2010. Multi-GPU accelerated multi-spin Monte Carlo simulations of the 2D Ising model. *Computer Physics Communications* 181, 9 (2010), 1549–1556.

[8] Kumar Chellapilla, Sidd Puri, and Patrice Simard. 2006. High performance convolutional neural networks for document processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft.

[9] Chih-Hong Cheng, Georg Nührenberg, Chung-Hao Huang, and Harald Ruess. 2018. Verification of Binarized Neural Networks via Inter-neuron Factoring. In *Working Conference on Verified Software: Theories, Tools, and Experiments*. Springer, 279–290.

[10] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759* (2014).

[11] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*. 3123–3131.

[12] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* (2016).

[13] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. ACM, 191–198.

[14] Sajad Darabi, Mouloud Belbahri, Matthieu Courbariaux, and Vahid Partovi Nia. 2018. BNN+: Improved binary network training. *arXiv preprint arXiv:1812.11800* (2018).

[15] B Denby. 1988. Neural networks and cellular automata in experimental high energy physics. *Computer Physics Communications* 49, 3 (1988), 429–448.

[16] B Denby, M Campbell, Franco Bedeschi, N Chriss, C Bowers, and F Nesti. 1990. Neural networks for triggering. *IEEE Transactions on Nuclear Science* 37, 2 (1990), 248–254.

[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. Ieee, 248–255.

[18] US DOE. 2018. Sensor Technologies and Data Analytics. https://www.smartgrid.gov/document/Sensor_Technologies_and_Data_Analytics_2018.html

[19] Marcus Edel and Enrico Köppe. 2016. Binarized-blstm-rnn based human activity recognition. In *2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. IEEE, 1–7.

[20] Wenbin Fang, Mian Lu, Xiangye Xiao, Bingsheng He, and Qiong Luo. 2009. Frequent itemset mining on graphics processors. In *Proceedings of the fifth international workshop on data management on new hardware*. ACM, 34–42.

[21] Francesco Fusco, Michail Vlachos, Xenofontas Dimitropoulos, and Luca Deri. 2013. Indexing million of packets per second using GPUs. In *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 327–332.

[22] Angus Galloway, Graham W Taylor, and Medhat Moussa. 2017. Attacking Binarized Neural Networks. *arXiv preprint arXiv:1711.00449* (2017).

[23] Tong Geng, Tianqi Wang, Chunshu Wu, Chen Yang, Shuaiwen Leon Song, Ang Li, and Martin Herbordt. 2019. LP-BNN: Ultra-low-Latency BNN Inference with Layer Parallelism. In *Proceedings of the 30th IEEE International Conference on Application-specific Systems, Architectures, and Processors*. IEEE.

[24] Tong Geng, Tianqi Wang, Chunshu Wu, Chen Yang, Wei Wu, Ang Li, and Martin C Herbordt. 2019. O3BNN: an out-of-order architecture for high-performance binarized neural network inference with fine-grained pruning. In *Proceedings of the ACM International Conference on Supercomputing*. ACM, 461–472.

[25] Mohammad Ghasemzadeh, Mohammad Samragh, and Farinaz Koushanfar. 2018. ReBNet: Residual binarized neural network. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 57–64.

[26] Google. 2018. TensorFlow: Adding a New Op. http://www.tensorflow.org/extend/adding_an_op

[27] Johann Hauswald, Yiping Kang, Michael A Laurenzano, Quan Chen, Cheng Li, Trevor Mudge, Ronald G Dreslinski, Jason Mars, and Lingjia Tang. 2015. DjiNN and Tonic: DNN as a service and its implications for future warehouse scale computers. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 27–40.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[29] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 173–182.

[30] Lu Hou, Quanming Yao, and James T Kwok. 2016. Loss-aware binarization of deep networks. *arXiv preprint arXiv:1611.01600* (2016).

[31] Yuwei Hu, Jidong Zhai, Dinghua Li, Yifan Gong, Yuhao Zhu, Wei Liu, Lei Su, and Jiangming Jin. 2018. BitFlow: Exploiting Vector Parallelism for Binary Neural Networks on CPU. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 244–253.

[32] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *Advances in Neural Information Processing Systems*. 4107–4115.

[33] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[34] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 675–678.

[35] Ram Srivatsa Kannan, Lavanya Subramanian, Ashwin Raju, Jeongseob Ahn, Jason Mars, and Lingjia Tang. 2019. GrandSLAm: Guaranteeing SLAs for Jobs in Microservices Execution Frameworks. In *Proceedings of the Fourteenth EuroSys Conference 2019*. ACM, 34.

[36] Hermann Kolanoski. 1996. Application of artificial neural networks in particle physics. In *International Conference on Artificial Neural Networks*. Springer, 1–14.

[37] Svyatoslav Korneev, Nina Narodytska, Luca Pulina, Armando Tacchella, Nikolaj Bjorner, and Mooly Sagiv. 2018. Constrained image generation using binarized neural networks with decision procedures. In *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 438–449.

[38] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2014. The CIFAR-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html* (2014).

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[40] Fayez Lahoud, Radhakrishna Achanta, Pablo Márquez-Neila, and Sabine Süsstrunk. 2019. Self-Binarizing Networks. *arXiv preprint arXiv:1902.00730* (2019).

[41] Chris Leary and Todd Wang. 2017. XLA: TensorFlow, compiled. *TensorFlow Dev Summit* (2017).

[42] Yann LeCun, Corinna Cortes, and CJ Burges. 2010. MNIST handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist* 2 (2010).

[43] Yann LeCun, LD Jackel, Leon Bottou, A Brunot, Corinna Cortes, JS Denker, Harris Drucker, I Guyon, UA Muller, Eduard Sackinger, et al. 1995. Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*, Vol. 60. Perth, Australia, 53–60.

[44] Francesco Leofante, Nina Narodytska, Luca Pulina, and Armando Tacchella. 2018. Automated verification of neural networks: Advances, challenges and perspectives. *arXiv preprint arXiv:1805.09938* (2018).

[45] Ang Li, Weifeng Liu, Linnan Wang, Kevin Barker, and Shuaiwen Song. 2018. Warp-Consolidation: A Novel Execution Model for GPUs. In *Proceedings of the 2018 International Conference on Supercomputing (ICS)*.

[46] Ang Li, Shuaiwen Leon Song, Eric Brugel, Akash Kumar, Daniel Chavarria-Miranda, and Henk Corporaal. 2016. X: A comprehensive analytic model for parallel machines. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 242–252.

[47] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, Xu Liu, Nathan Tallent, and Kevin Barker. 2019. Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect. *arXiv preprint arXiv:1903.04611* (2019).

[48] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Xu Liu, Nathan Tallent, and Kevin Barker. 2018. Tartan: evaluating modern GPU interconnect via a multi-GPU benchmark suite. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 191–202.

[49] Ang Li, Shuaiwen Leon Song, Akash Kumar, Eddy Z Zhang, Daniel Chavarría-Miranda, and Henk Corporaal. 2016. Critical points based register-concurrency

autotuning for GPUs. In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*. EDA Consortium, 1273–1278.

[50] Ang Li, Shuaiwen Leon Song, Weifeng Liu, Xu Liu, Akash Kumar, and Henk Corporaal. 2017. Locality-aware cta clustering for modern gpus. *ACM SIGOPS Operating Systems Review* 51, 2 (2017), 297–311.

[51] Ang Li, YC Tay, Akash Kumar, and Henk Corporaal. [n. d.]. Transit: A visual analytical model for multithreaded machines. In *HPDC-15*. ACM.

[52] Ang Li, Gert-Jan van den Braak, Henk Corporaal, and Akash Kumar. 2015. Fine-grained synchronizations and dataflow programming on GPUs. In *Proceedings of the 29th ACM on International Conference on Supercomputing*. ACM, 109–118.

[53] Ang Li, Gert-Jan van den Braak, Akash Kumar, and Henk Corporaal. 2015. Adaptive and transparent cache bypassing for GPUs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 17.

[54] Shuang Liang, Shouyi Yin, Leibo Liu, Wayne Luk, and Shaojun Wei. 2018. FP-BNN: Binarized neural network on FPGA. *Neurocomputing* 275 (2018), 1072–1086.

[55] Tatiana Likhomanenko, Philip Ilten, Egor Khairullin, Alex Rogozhnikov, Andrey Ustyuzhanin, and Michael Williams. 2015. LHCb topological trigger reoptimization. In *Journal of Physics: Conference Series*, Vol. 664. IOP Publishing, 082025.

[56] Xiaofan Lin, Cong Zhao, and Wei Pan. 2017. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*. 345–353.

[57] Leif Lönnblad, Carsten Peterson, and Thorsteinn Rögnvaldsson. 1990. Finding gluon jets with a neural trigger. *Physical review letters* 65, 11 (1990), 1321.

[58] Chao Ma, Yulan Guo, Yinjie Lei, and Wei An. 2018. Binary volumetric convolutional neural networks for 3-d object recognition. *IEEE Transactions on Instrumentation and Measurement* 99 (2018), 1–11.

[59] Yinglan Ma, Hongyu Xiong, Zhe Hu, and Lizhuang Ma. 2018. Efficient Super Resolution Using Binarized Neural Network. *arXiv preprint arXiv:1812.06378* (2018).

[60] Bradley McDanel, Surat Teerapittayanon, and HT Kung. 2017. Embedded binarized neural networks. *arXiv preprint arXiv:1709.02260* (2017).

[61] Nina Narodytska. 2018. Formal Analysis of Deep Binarized Neural Networks.. In *IJCAI*. 5692–5696.

[62] Nina Narodytska, Shiva Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. 2018. Verifying properties of binarized deep neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[63] Nina Narodytska, Shiva Prasad Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. 2017. Verifying properties of binarized deep neural networks. *arXiv preprint arXiv:1709.06662* (2017).

[64] Eriko Nurvitadhi, David Sheffield, Jaewoong Sim, Asit Mishra, Ganesh Venkatesh, and Debbie Marr. 2016. Accelerating binarized neural networks: comparison of FPGA, CPU, GPU, and ASIC. In *Field-Programmable Technology (FPT), 2016 International Conference on*. IEEE, 77–84.

[65] NVIDIA. 2018. CUDA Programming Guide. http://docs.nvidia.com/cuda/cuda-c-programming-guide

[66] Sreepathi Pai, Matthew J Thazhuthaveetil, and Ramaswamy Govindarajan. 2013. Improving GPGPU concurrency with elastic kernels. In *ACM SIGPLAN Notices*, Vol. 48. ACM, 407–418.

[67] Martín Pedemonte, Enrique Alba, and Francisco Luna. 2011. Bitwise operations for GPU implementation of genetic algorithms. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*. ACM, 439–446.

[68] Carsten Peterson. 1989. Track finding with neural networks. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 279, 3 (1989), 537–545.

[69] Carsten Peterson, Thorsteinn Rögnvaldsson, and Leif Lönnblad. 1994. JETNET 3.0: A versatile artificial neural network package. *Computer Physics Communications* 81, 1-2 (1994), 185–220.

[70] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*. Springer, 525–542.

[71] Buser Say and Scott Sanner. 2018. Planning in Factored State and Action Spaces with Learned Binarized Neural Network Transition Models.. In *IJCAI*. 4815–4821.

[72] Jingkuan Song. 2017. Binary generative adversarial networks for image retrieval. *arXiv preprint arXiv:1708.04150* (2017).

[73] Ion Stoica, Dawn Song, Raluca Ada Popa, David Patterson, Michael W Mahoney, Randy Katz, Anthony D Joseph, Michael Jordan, Joseph M Hellerstein, Joseph E Gonzalez, et al. 2017. A berkeley view of systems challenges for ai. *arXiv preprint arXiv:1712.05855* (2017).

[74] Wei Tang, Gang Hua, and Liang Wang. 2017. How to train a compact binary neural network with high accuracy?. In *AAAI*. 2625–2631.

[75] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 65–74.

[76] Nicolas Vasilache, Oleksandr Zinenko, Theodoros Theodoridis, Priya Goyal, Zachary DeVito, William S Moses, Sven Verdoolaege, Andrew Adams, and Albert Cohen. 2018. Tensor Comprehensions: Framework-Agnostic High-Performance Machine Learning Abstractions. *arXiv preprint arXiv:1802.04730* (2018).

[77] Kefu Xu, Wenke Cui, Yue Hu, and Li Guo. 2013. Bit-parallel multiple approximate string matching based on GPU. *Procedia Computer Science* 17 (2013), 523–529.

[78] Ritchie Zhao, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng-Hau Lin, Mani Srivastava, Rajesh Gupta, and Zhiru Zhang. 2017. Accelerating binarized convolutional neural networks with software-programmable fpgas. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 15–24.

[79] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).

[80] Shilin Zhu, Xin Dong, and Hao Su. 2018. Binary Ensemble Neural Network: More Bits per Network or More Networks per Bit? *arXiv preprint arXiv:1806.07550* (2018).

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

We ran our Bit-Matrix-Multiplication, Bit-Convolution in the sizes mentioned in the paper on NVIDIA DGX-1P machine with Tesla P100 GPU and CUDA Toolkit-9.1, NVIDIA DGX-1V machine with Tesla V100 GPU and CUDA Toolkit-9.0. We ran our SBNN inference framework on the same two machines.

ARTIFACT AVAILABILITY

Software Artifact Availability: All author-created software artifacts are maintained in a public repository under an OSI-approved license.

Hardware Artifact Availability: There are no author-created hardware artifacts.

Data Artifact Availability: All author-created data artifacts are maintained in a public repository under an OSI-approved license.

Proprietary Artifacts: There are associated proprietary artifacts that are not created by the authors. Some author-created artifacts are proprietary.

List of URLs and/or DOIs where artifacts are available:

`https://github.com/uuudown/SBNN`

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: NVIDIA DGX-1P and DGX-1V

Operating systems and versions: Ubuntu 5.4.0-6ubuntu1 16.04.4

Compilers and versions: nvcc 9.0/9.1 and gcc 5.4.0

Key algorithms: binarized neural network

Input datasets and versions: MNIST, CIFAR10 and ImageNet

Output from scripts that gathers execution environment information.

```
SUDO_GID=34669
MAIL=/var/mail/USER
USER=USER
SUDO_UID=220292
LOGNAME=USER
TERM=screen.xterm-256color
USERNAME=USER
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/b⌋
↪ in:/sbin:/bin:/snap/bin
LANG=en_US.UTF-8
SUDO_COMMAND=./collect_environment.sh
KRB5CCNAME=FILE:/tmp/krb5cc_220292_EvbTPa
SHELL=/bin/bash
Distributor ID:        Ubuntu
Description:        Ubuntu 16.04.3 LTS
Release:        16.04
Codename:        xenial
Linux dgx-2 4.4.0-92-generic #115-Ubuntu SMP Thu Aug
↪ 10 09:04:33 UTC 2017 x86_64 x86_64 x86_64
↪ GNU/Linux
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                80
On-line CPU(s) list:   0-79
Thread(s) per core:    2
Core(s) per socket:    20
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 79
Model name:            Intel(R) Xeon(R) CPU E5-2698 v4
↪ @ 2.20GHz
Stepping:              1
CPU MHz:               3298.625
CPU max MHz:           3600.0000
CPU min MHz:           1200.0000
BogoMIPS:              4391.50
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              51200K
NUMA node0 CPU(s):     0-19,40-59
NUMA node1 CPU(s):     20-39,60-79
Flags:                 fpu vme de pse tsc msr pae mce
↪ cx8 apic sep mtrr pge mca cmov pat pse36 clflush
↪ dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx
↪ pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs
↪ bts rep_good nopl xtopology nonstop_tsc
↪ aperfmperf eagerfpu pni pclmulqdq dtes64 monitor
↪ ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr
↪ pdcm pcid dca sse4_1 sse4_2 x2apic movbe popcnt
↪ tsc_deadline_timer aes xsave avx f16c rdrand
↪ lahf_lm abm 3dnowprefetch epb intel_pt tpr_shadow
↪ vnmi flexpriority ept vpid fsgsbase tsc_adjust
↪ bmi1 hle avx2 smep bmi2 erms invpcid rtm cqm
↪ rdseed adx smap xsaveopt cqm_llc cqm_occup_llc
↪ cqm_mbm_total cqm_mbm_local dtherm ida arat pln
↪ pts
MemTotal:       528276048 kB
MemFree:        246554004 kB
MemAvailable:   522876632 kB
Buffers:         2189484 kB
Cached:         267688704 kB
SwapCached:            0 kB
Active:         182478716 kB
Inactive:       88420124 kB
```

```
Active(anon):    1192724 kB
Inactive(anon):   133860 kB
Active(file):   181285992 kB
Inactive(file): 88286264 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:               112 kB
Writeback:             0 kB
AnonPages:       1020740 kB
Mapped:           223192 kB
Shmem:            305936 kB
Slab:            8496620 kB
SReclaimable:    8150652 kB
SUnreclaim:       345968 kB
KernelStack:       20608 kB
PageTables:        20164 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:    264138024 kB
Committed_AS:   15096524 kB
VmallocTotal:   34359738367 kB
VmallocUsed:           0 kB
VmallocChunk:          0 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:    23095528 kB
DirectMap2M:    210624512 kB
DirectMap1G:    305135616 kB
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0 446.6G  0 disk
├─sda1   8:1    0   487M  0 part /boot/efi
└─sda2   8:2    0 446.2G  0 part /
sdb      8:16   0     7T  0 disk
└─sdb1   8:17   0     7T  0 part /raid
loop0    7:0    0         0 loop
loop1    7:1    0         0 loop
loop2    7:2    0         0 loop
loop3    7:3    0         0 loop
loop4    7:4    0         0 loop
loop5    7:5    0         0 loop
loop6    7:6    0         0 loop
loop7    7:7    0         0 loop
Wed Apr 10 11:46:42 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.81                 Driver Version: 384.81                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:06:00.0 Off |                    0 |
| N/A   33C    P0    41W / 300W |     10MiB / 16152MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:07:00.0 Off |                    0 |
| N/A   35C    P0    41W / 300W |     10MiB / 16152MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:0A:00.0 Off |                    0 |
| N/A   36C    P0    44W / 300W |     10MiB / 16152MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:0B:00.0 Off |                    0 |
| N/A   33C    P0    41W / 300W |     10MiB / 16152MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   4  Tesla V100-SXM2...  On   | 00000000:85:00.0 Off |                    0 |
| N/A   35C    P0    41W / 300W |     10MiB / 16152MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   5  Tesla V100-SXM2...  On   | 00000000:86:00.0 Off |                    0 |
| N/A   36C    P0    43W / 300W |     10MiB / 16152MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   6  Tesla V100-SXM2...  On   | 00000000:89:00.0 Off |                    0 |
| N/A   38C    P0    42W / 300W |     10MiB / 16152MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   7  Tesla V100-SXM2...  On   | 00000000:8A:00.0 Off |                    0 |
```

```
| N/A   36C    P0    42W / 300W |     10MiB / 16152MiB
↪  |      0%      Default |
+-------------------------------+----------------------┐
↪  ---+----------------------+

+-----------------------------------------------------┐
↪  --------------------------+
| Processes:
↪             GPU Memory |
|  GPU       PID   Type   Process name
↪              Usage      |
|=====================================================┐
↪  ==========================|
|  No running processes found
↪                           |
+-----------------------------------------------------┐
↪  --------------------------+
H/W path           Device       Class
↪  Description
=====================================================┐
↪  ======
                                system         DGX-1
                                ↪  with V100
                                ↪  (Default string)
/0                              bus            DGX-1
↪  with V100
/0/0                         memory        64KiB BIOS
/0/36                           memory         System
↪  Memory
/0/36/0                         memory         31GiB
↪  DIMM Synchronous 2400 MHz (0.4 ns)
/0/36/1                         memory         31GiB
↪  DIMM Synchronous 2400 MHz (0.4 ns)
/0/36/2                         memory         31GiB
↪  DIMM Synchronous 2400 MHz (0.4 ns)
/0/36/3                         memory         31GiB
↪  DIMM Synchronous 2400 MHz (0.4 ns)
/0/3c                           memory         System
↪  Memory
/0/3c/0                         memory         31GiB
↪  DIMM Synchronous 2400 MHz (0.4 ns)
/0/3c/1                         memory         31GiB
↪  DIMM Synchronous 2400 MHz (0.4 ns)
/0/3c/2                         memory         31GiB
↪  DIMM Synchronous 2400 MHz (0.4 ns)
/0/3c/3                         memory         31GiB
↪  DIMM Synchronous 2400 MHz (0.4 ns)
/0/42                           memory         System
↪  Memory
/0/42/0                         memory         31GiB
↪  DIMM Synchronous 2400 MHz (0.4 ns)
/0/42/1                         memory         31GiB
↪  DIMM Synchronous 2400 MHz (0.4 ns)
/0/42/2                         memory         31GiB
↪  DIMM Synchronous 2400 MHz (0.4 ns)
/0/42/3                         memory         31GiB
↪  DIMM Synchronous 2400 MHz (0.4 ns)
/0/48                           memory         System
↪  Memory
/0/48/0                         memory         31GiB
↪  DIMM Synchronous 2400 MHz (0.4 ns)
/0/48/1                         memory         31GiB
↪  DIMM Synchronous 2400 MHz (0.4 ns)
/0/48/2                         memory         31GiB
↪  DIMM Synchronous 2400 MHz (0.4 ns)
/0/48/3                         memory         31GiB
↪  DIMM Synchronous 2400 MHz (0.4 ns)
/0/4e                           memory         1280KiB
↪  L1 cache
/0/4f                           memory         5MiB L2
↪  cache
/0/50                           memory         50MiB L3
↪  cache
/0/51                         processor      Intel(R)
↪  Xeon(R) CPU E5-2698 v4 @ 2.20GHz
/0/52                           memory         1280KiB
↪  L1 cache
/0/53                           memory         5MiB L2
↪  cache
/0/54                           memory         50MiB L3
↪  cache
/0/55                         processor      Intel(R)
↪  Xeon(R) CPU E5-2698 v4 @ 2.20GHz
/0/4                            memory
/0/6                            memory
/0/100                          bridge         Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DMI2
/0/100/1                        bridge         Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D PCI Express Root
↪  Port 1
/0/100/1/0         enp1s0f0     network
↪  Ethernet Controller 10-Gigabit X540-AT2
/0/100/1/0.1       enp1s0f1     network
↪  Ethernet Controller 10-Gigabit X540-AT2
/0/100/2                        bridge         Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D PCI Express Root
↪  Port 2
/0/100/2/0                      bridge         PLX
↪  Technology, Inc.
/0/100/2/0/4                    bridge         PLX
↪  Technology, Inc.
/0/100/2/0/4/0     ib0          network        MT27700
↪  Family [ConnectX-4]
/0/100/2/0/8                    bridge         PLX
↪  Technology, Inc.
/0/100/2/0/8/0                  display        NVIDIA
↪  Corporation
```

```
/0/100/2/0/c                    bridge         PLX
↪  Technology, Inc.
/0/100/2/0/c/0                  display        NVIDIA
↪  Corporation
/0/100/3                        bridge         Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D PCI Express Root
↪  Port 3
/0/100/3/0                      bridge         PLX
↪  Technology, Inc.
/0/100/3/0/4                    bridge         PLX
↪  Technology, Inc.
/0/100/3/0/4/0                  display        NVIDIA
↪  Corporation
/0/100/3/0/8                    bridge         PLX
↪  Technology, Inc.
/0/100/3/0/8/0                  display        NVIDIA
↪  Corporation
/0/100/3/0/c                    bridge         PLX
↪  Technology, Inc.
/0/100/3/0/c/0     ib1          network        MT27700
↪  Family [ConnectX-4]
/0/100/3/0/10                   bridge         PLX
↪  Technology, Inc.
/0/100/3/0/11                   bridge         PLX
↪  Technology, Inc.
/0/100/3/0/12                   bridge         PLX
↪  Technology, Inc.
/0/100/3/0/13                   bridge         PLX
↪  Technology, Inc.
/0/100/5                        generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D
↪  Map/VTd_Misc/System Management
/0/100/5.1                      generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D IIO Hot Plug
/0/100/5.2                      generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D IIO RAS/Control
↪  Status/Global Errors
/0/100/5.4                      generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D I/O APIC
/0/100/11                       generic        C610/X99
↪  series chipset SPSR
/0/100/11.4                     storage        C610/X99
↪  series chipset sSATA Controller [AHCI mode]
/0/100/16                       communication  C610/X99
↪  series chipset MEI Controller #1
/0/100/16.1                     communication
↪  C610/X99 series chipset MEI Controller #2
/0/100/1a                       bus            C610/X99
↪  series chipset USB Enhanced Host Controller #2
/0/100/1a/1        usb1         bus            EHCI
↪  Host Controller
/0/100/1a/1/1                   bus            USB hub
/0/100/1a/1/1/2                 bus            Virtual
↪  Hub
/0/100/1a/1/1/2/4               input          Virtual
↪  Keyboard and Mouse
/0/100/1a/1/1/3                 bus            USB2.0 Hub
/0/100/1a/1/1/4                 bus            USB2.0 Hub
/0/100/1c                       bridge         C610/X99
↪  series chipset PCI Express Root Port #1
/0/100/1c/0                     bridge         AST1150
↪  PCI-to-PCI Bridge
/0/100/1c/0/0                   display        ASPEED
↪  Graphics Family
/0/100/1d                       bus            C610/X99
↪  series chipset USB Enhanced Host Controller #1
/0/100/1d/1        usb2         bus            EHCI
↪  Host Controller
/0/100/1d/1/1                   bus            USB hub
/0/100/1f                       bridge         C610/X99
↪  series chipset LPC Controller
/0/100/1f.2                     storage        C610/X99
↪  series chipset 6-Port SATA Controller [AHCI mode]
/0/100/1f.3                     bus            C610/X99
↪  series chipset SMBus Controller
/0/100/1f.6                     generic        C610/X99
↪  series chipset Thermal Subsystem
/0/7                            generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 0
/0/8                            generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 0
/0/9                            generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 0
/0/a                            generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 1
/0/b                            generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 1
/0/c                            generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 1
/0/d                            generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D R3 QPI Link 0/1
/0/e                            generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D R3 QPI Link 0/1
/0/f                            generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D R3 QPI Link 0/1
/0/10                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D R3 QPI Link Debug
/0/11                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/12                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/13                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/14                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/15                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
```

```
/0/16                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/17                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/18                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/19                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/1a                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/1b                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/1c                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/1d                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/1e                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/1f                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/20                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/21                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/22                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/23                           generic        Intel
↪  Corporation
/0/24                           generic        Intel
↪  Corporation
/0/25                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/26                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/27                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/28                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/29                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/2a                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/2b                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/2c                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D R2PCIe Agent
/0/2d                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D R2PCIe Agent
/0/2e                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Ubox
/0/2f                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Ubox
/0/30                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Ubox
/0/31                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 0
/0/32                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 0
/0/33                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 0 Debug
/0/34                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 1
/0/35                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 1
/0/37                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 1 Debug
/0/38                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Target Address/Thermal/RAS
/0/39                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Target Address/Thermal/RAS
/0/3a                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel Target Address Decoder
/0/3b                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel Target Address Decoder
/0/3d                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Broadcast
/0/3e                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Global
↪  Broadcast
/0/3f                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel 0 Thermal Control
/0/40                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel 1 Thermal Control
/0/41                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel 0 Error
/0/43                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel 1 Error
/0/44                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Interface
/0/45                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Interface
/0/46                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Interface
/0/47                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Interface
```

```
/0/49                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Target
↪  Address/Thermal/RAS
/0/4a                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Target
↪  Address/Thermal/RAS
/0/4b                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Channel Target
↪  Address Decoder
/0/4c                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Channel Target
↪  Address Decoder
/0/4d                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Broadcast
/0/56                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Global
↪  Broadcast
/0/57                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  1 - Channel 0 Thermal Control
/0/58                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  1 - Channel 1 Thermal Control
/0/59                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  1 - Channel 0 Error
/0/5a                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  1 - Channel 1 Error
/0/5b                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Interface
/0/5c                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Interface
/0/5d                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Interface
/0/5e                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Interface
/0/5f                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
/0/60                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
/0/61                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
/0/62                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
/0/63                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
/0/64                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
/0/65                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
/0/1                            bridge         Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D PCI Express Root
↪  Port 1
/0/1/0            scsi0        storage        MegaRAID
↪  SAS-3 3108 [Invader]
/0/1/0/0.8.0                    generic        DGX-1
/0/1/0/2.0.0      /dev/sda      disk           479GB
↪  MRROMB
/0/1/0/2.0.0/1                  volume         486MiB
↪  Windows FAT volume
/0/1/0/2.0.0/2    /dev/sda2     volume         446GiB
↪  EXT4 volume
/0/1/0/2.1.0      /dev/sdb      disk           7679GB
↪  MRROMB
/0/1/0/2.1.0/1    /dev/sdb1     volume         7151GiB
↪  EXT4 volume
/0/2                            bridge         Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D PCI Express Root
↪  Port 2
/0/2/0                          bridge         PLX
↪  Technology, Inc.
/0/2/0/4                        bridge         PLX
↪  Technology, Inc.
/0/2/0/4/0        ib2           network        MT27700
↪  Family [ConnectX-4]
/0/2/0/8                        bridge         PLX
↪  Technology, Inc.
/0/2/0/8/0                      display        NVIDIA
↪  Corporation
/0/2/0/c                        bridge         PLX
↪  Technology, Inc.
/0/2/0/c/0                      display        NVIDIA
↪  Corporation
/0/3                            bridge         Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D PCI Express Root
↪  Port 3
/0/3/0                          bridge         PLX
↪  Technology, Inc.
/0/3/0/4                        bridge         PLX
↪  Technology, Inc.
/0/3/0/4/0                      display        NVIDIA
↪  Corporation
/0/3/0/8                        bridge         PLX
↪  Technology, Inc.
/0/3/0/8/0                      display        NVIDIA
↪  Corporation
/0/3/0/c                        bridge         PLX
↪  Technology, Inc.
/0/3/0/c/0        ib3           network        MT27700
↪  Family [ConnectX-4]
/0/3/0/10                       bridge         PLX
↪  Technology, Inc.
/0/3/0/11                       bridge         PLX
↪  Technology, Inc.
```

```
/0/3/0/12                       bridge         PLX
↪  Technology, Inc.
/0/3/0/13                       bridge         PLX
↪  Technology, Inc.
/0/5                            generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D
↪  Map/VTd_Misc/System Management
/0/5.1                          generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D IIO Hot Plug
/0/5.2                          generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D IIO RAS/Control
↪  Status/Global Errors
/0/5.4                          generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D I/O APIC
/0/66                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 0
/0/67                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 0
/0/68                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 0
/0/69                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 1
/0/6a                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 1
/0/6b                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 1
/0/6c                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D R3 QPI Link 0/1
/0/6d                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D R3 QPI Link 0/1
/0/6e                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D R3 QPI Link 0/1
/0/6f                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D R3 QPI Link Debug
/0/70                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/71                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/72                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/73                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/74                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/75                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/76                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/77                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/78                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/79                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/7a                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/7b                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/7c                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/7d                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/7e                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/7f                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/80                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/81                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/82                           generic        Intel
↪  Corporation
/0/83                           generic        Intel
↪  Corporation
/0/84                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/85                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/86                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/87                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/88                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/89                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/8a                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent
/0/8b                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D R2PCIe Agent
/0/8c                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D R2PCIe Agent
/0/8d                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Ubox
/0/8e                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Ubox
/0/8f                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Ubox
/0/90                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 0
/0/91                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 0
/0/92                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 0 Debug
/0/93                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 1
/0/94                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 1
```

```
/0/95                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 1 Debug
/0/96                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Target Address/Thermal/RAS
/0/97                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Target Address/Thermal/RAS
/0/98                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel Target Address Decoder
/0/99                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel Target Address Decoder
/0/9a                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Broadcast
/0/9b                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Global
↪  Broadcast
/0/9c                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel 0 Thermal Control
/0/9d                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel 1 Thermal Control
/0/9e                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel 0 Error
/0/9f                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel 1 Error
/0/a0                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Interface
/0/a1                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Interface
/0/a2                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Interface
/0/a3                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Interface
/0/a4                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Target
↪  Address/Thermal/RAS
/0/a5                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Target
↪  Address/Thermal/RAS
/0/a6                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Channel Target
↪  Address Decoder
/0/a7                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Channel Target
↪  Address Decoder
/0/a8                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Broadcast
/0/a9                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Global
↪  Broadcast
/0/aa                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  1 - Channel 0 Thermal Control
/0/ab                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  1 - Channel 1 Thermal Control
/0/ac                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  1 - Channel 0 Error
/0/ad                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  1 - Channel 1 Error
/0/ae                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Interface
/0/af                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Interface
/0/b0                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Interface
/0/b1                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Interface
/0/b2                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
/0/b3                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
/0/b4                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
/0/b5                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
/0/b6                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
/0/b7                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
/0/b8                           generic        Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
/1                vethe9e72a8  network        Ethernet
↪  interface


SUDO_GID=1011
MAIL=/var/mail/USER
LANGUAGE=en_US:
USER=USER
SUDO_UID=1011
LOGNAME=USER
TERM=xterm-256color
USERNAME=USER
```

```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/b
↪ in:/sbin:/bin
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=0
↪ 1;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;0
↪ 1:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=3
↪ 4;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*
↪ .arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*
↪ .lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:
↪ *.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:
↪ *.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=
↪ 01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.bz2=01
↪ ;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;
↪ 31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;
↪ 31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;
↪ 31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;
↪ 31:*.rz=01;31:*.cab=01;31:*.jpg=01;35:*.jpeg=01;
↪ 35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;
↪ 35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;
↪ 35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01
↪ ;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=0
↪ 1;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=
↪ 01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v
↪ =01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv
↪ =01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb
↪ =01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv
↪ =01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=0
↪ 1;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=0
↪ 1;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=0
↪ 0;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=
↪ 00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=0
↪ 0;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=
↪ 00;36:*.xspf=00;36:
SUDO_COMMAND=./collect_environment.sh
SHELL=/bin/bash
Distributor ID:        Ubuntu
Description:        Ubuntu 16.04.5 LTS
Release:        16.04
Codename:        xenial
Linux dgx-1 4.4.0-116-generic #140-Ubuntu SMP Mon Feb
↪ 12 21:23:04 UTC 2018 x86_64 x86_64 x86_64
↪ GNU/Linux
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                80
On-line CPU(s) list:   0-79
Thread(s) per core:    2
Core(s) per socket:    20
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 79
Model name:            Intel(R) Xeon(R) CPU E5-2698 v4
↪ @ 2.20GHz
Stepping:              1
CPU MHz:               1200.031
CPU max MHz:           3600.0000
CPU min MHz:           1200.0000
BogoMIPS:              4391.52
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              51200K
NUMA node0 CPU(s):     0-19,40-59
NUMA node1 CPU(s):     20-39,60-79
Flags:                 fpu vme de pse tsc msr pae mce
↪ cx8 apic sep mtrr pge mca cmov pat pse36 clflush
↪ dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx
↪ pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs
↪ bts rep_good nopl xtopology nonstop_tsc
↪ aperfmperf eagerfpu pni pclmulqdq dtes64 monitor
↪ ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr
↪ pdcm pcid dca sse4_1 sse4_2 x2apic movbe popcnt
↪ tsc_deadline_timer aes xsave avx f16c rdrand
↪ lahf_lm abm 3dnowprefetch epb invpcid_single
↪ intel_pt retpoline kaiser tpr_shadow vnmi
↪ flexpriority ept vpid fsgsbase tsc_adjust bmi1
↪ hle avx2 smep bmi2 erms invpcid rtm cqm rdseed adx
↪ smap xsaveopt cqm_llc cqm_occup_llc cqm_mbm_total
↪ cqm_mbm_local dtherm ida arat pln pts
MemTotal:       528277508 kB
MemFree:        520118224 kB
MemAvailable:   524288976 kB
Buffers:         1471208 kB
Cached:          3338684 kB
SwapCached:            0 kB
Active:          3811984 kB
Inactive:        1158980 kB
Active(anon):     162564 kB
Inactive(anon):    78864 kB
Active(file):    3649420 kB
Inactive(file):  1080116 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:                60 kB
Writeback:             0 kB
AnonPages:        161252 kB
Mapped:           183048 kB
Shmem:             80328 kB
Slab:            1100952 kB
SReclaimable:     841496 kB
SUnreclaim:       259456 kB
KernelStack:       16768 kB
PageTables:         8888 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:    264138752 kB
```

```
Committed_AS:    1688052 kB
VmallocTotal:   34359738367 kB
VmallocUsed:           0 kB
VmallocChunk:          0 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:    24557612 kB
DirectMap2M:    34052096 kB
DirectMap1G:    480247808 kB
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0 446.6G  0 disk
├─sda1   8:1    0   121M  0 part /boot/efi
└─sda2   8:2    0 446.5G  0 part /
sdb      8:16   0     7T  0 disk
└─sdb1   8:17   0     7T  0 part /raid
loop0    7:0    0          0 loop
loop1    7:1    0          0 loop
loop2    7:2    0          0 loop
loop3    7:3    0          0 loop
loop4    7:4    0          0 loop
loop5    7:5    0          0 loop
loop6    7:6    0          0 loop
loop7    7:7    0          0 loop
Wed Apr 10 14:10:06 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.79       Driver Version: 410.79       CUDA Version: 10.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla P100-SXM2...  On   | 00000000:06:00.0 Off |                    0 |
| N/A   39C    P0    31W / 300W |     10MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla P100-SXM2...  On   | 00000000:07:00.0 Off |                    0 |
| N/A   36C    P0    33W / 300W |     10MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla P100-SXM2...  On   | 00000000:0A:00.0 Off |                    0 |
| N/A   37C    P0    31W / 300W |     10MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla P100-SXM2...  On   | 00000000:0B:00.0 Off |                    0 |
| N/A   32C    P0    31W / 300W |     10MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   4  Tesla P100-SXM2...  On   | 00000000:85:00.0 Off |                    0 |
| N/A   32C    P0    31W / 300W |     10MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   5  Tesla P100-SXM2...  On   | 00000000:86:00.0 Off |                    0 |
| N/A   35C    P0    33W / 300W |     10MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   6  Tesla P100-SXM2...  On   | 00000000:89:00.0 Off |                    0 |
| N/A   36C    P0    33W / 300W |     10MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   7  Tesla P100-SXM2...  On   | 00000000:8A:00.0 Off |                    0 |
| N/A   39C    P0    32W / 300W |     10MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
00:00.0 Host bridge: Intel Corporation Xeon E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D DMI2 (rev 01)
00:01.0 PCI bridge: Intel Corporation Xeon E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D PCI Express Root Port 1 (rev 01)
```

```
00:02.0 PCI bridge: Intel Corporation Xeon E7 v4/Xeon
↪  E5 v4/Xeon E3 v4/Xeon D PCI Express Root Port 2
↪  (rev 01)
00:03.0 PCI bridge: Intel Corporation Xeon E7 v4/Xeon
↪  E5 v4/Xeon E3 v4/Xeon D PCI Express Root Port 3
↪  (rev 01)
00:05.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D
↪  Map/VTd_Misc/System Management (rev 01)
00:05.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D IIO Hot Plug (rev
↪  01)
00:05.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D IIO RAS/Control
↪  Status/Global Errors (rev 01)
00:05.4 PIC: Intel Corporation Xeon E7 v4/Xeon E5
↪  v4/Xeon E3 v4/Xeon D I/O APIC (rev 01)
00:11.0 Unassigned class [ff00]: Intel Corporation
↪  C610/X99 series chipset SPSR (rev 05)
00:11.4 SATA controller: Intel Corporation C610/X99
↪  series chipset sSATA Controller [AHCI mode] (rev
↪  05)
00:16.0 Communication controller: Intel Corporation
↪  C610/X99 series chipset MEI Controller #1 (rev 05)
00:16.1 Communication controller: Intel Corporation
↪  C610/X99 series chipset MEI Controller #2 (rev 05)
00:1a.0 USB controller: Intel Corporation C610/X99
↪  series chipset USB Enhanced Host Controller #2
↪  (rev 05)
00:1c.0 PCI bridge: Intel Corporation C610/X99 series
↪  chipset PCI Express Root Port #1 (rev d5)
00:1d.0 USB controller: Intel Corporation C610/X99
↪  series chipset USB Enhanced Host Controller #1
↪  (rev 05)
00:1f.0 ISA bridge: Intel Corporation C610/X99 series
↪  chipset LPC Controller (rev 05)
00:1f.2 SATA controller: Intel Corporation C610/X99
↪  series chipset 6-Port SATA Controller [AHCI mode]
↪  (rev 05)
00:1f.3 SMBus: Intel Corporation C610/X99 series
↪  chipset SMBus Controller (rev 05)
00:1f.6 Signal processing controller: Intel
↪  Corporation C610/X99 series chipset Thermal
↪  Subsystem (rev 05)
01:00.0 Ethernet controller: Intel Corporation
↪  Ethernet Controller 10-Gigabit X540-AT2 (rev 01)
01:00.1 Ethernet controller: Intel Corporation
↪  Ethernet Controller 10-Gigabit X540-AT2 (rev 01)
03:00.0 PCI bridge: PLX Technology, Inc. Device 8764
↪  (rev ab)
04:04.0 PCI bridge: PLX Technology, Inc. Device 8764
↪  (rev ab)
04:08.0 PCI bridge: PLX Technology, Inc. Device 8764
↪  (rev ab)
04:0c.0 PCI bridge: PLX Technology, Inc. Device 8764
↪  (rev ab)
05:00.0 Infiniband controller: Mellanox Technologies
↪  MT27700 Family [ConnectX-4]
06:00.0 3D controller: NVIDIA Corporation Device 15f9
↪  (rev a1)
07:00.0 3D controller: NVIDIA Corporation Device 15f9
↪  (rev a1)
08:00.0 PCI bridge: PLX Technology, Inc. Device 8780
↪  (rev ab)
09:04.0 PCI bridge: PLX Technology, Inc. Device 8780
↪  (rev ab)
09:08.0 PCI bridge: PLX Technology, Inc. Device 8780
↪  (rev ab)
09:0c.0 PCI bridge: PLX Technology, Inc. Device 8780
↪  (rev ab)
09:10.0 PCI bridge: PLX Technology, Inc. Device 8780
↪  (rev ab)
09:11.0 PCI bridge: PLX Technology, Inc. Device 8780
↪  (rev ab)
09:12.0 PCI bridge: PLX Technology, Inc. Device 8780
↪  (rev ab)
09:13.0 PCI bridge: PLX Technology, Inc. Device 8780
↪  (rev ab)
0a:00.0 3D controller: NVIDIA Corporation Device 15f9
↪  (rev a1)
0b:00.0 3D controller: NVIDIA Corporation Device 15f9
↪  (rev a1)
0c:00.0 Infiniband controller: Mellanox Technologies
↪  MT27700 Family [ConnectX-4]
11:00.0 PCI bridge: ASPEED Technology, Inc. AST1150
↪  PCI-to-PCI Bridge (rev 03)
12:00.0 VGA compatible controller: ASPEED Technology,
↪  Inc. ASPEED Graphics Family (rev 30)
7f:08.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 0 (rev
↪  01)
7f:08.2 Performance counters: Intel Corporation Xeon
↪  E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 0 (rev
↪  01)
7f:08.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 0 (rev
↪  01)
7f:09.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 1 (rev
↪  01)
7f:09.2 Performance counters: Intel Corporation Xeon
↪  E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 1 (rev
↪  01)
7f:09.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 1 (rev
↪  01)
7f:0b.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D R3 QPI Link 0/1
↪  (rev 01)
```

```
7f:0b.1 Performance counters: Intel Corporation Xeon
↪  E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D R3 QPI Link 0/1
↪  (rev 01)
7f:0b.2 Performance counters: Intel Corporation Xeon
↪  E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D R3 QPI Link 0/1
↪  (rev 01)
7f:0b.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D R3 QPI Link Debug
↪  (rev 01)
7f:0c.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0c.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0c.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0c.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0c.4 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0c.5 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0c.6 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0c.7 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0d.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0d.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0d.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0d.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0d.4 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0d.5 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0d.6 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0d.7 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0e.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0e.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0e.2 Unassigned class [ffff]: Intel Corporation
↪  Device 6ff2 (rev 01)
7f:0e.3 Unassigned class [ffff]: Intel Corporation
↪  Device 6ff3 (rev 01)
7f:0f.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0f.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0f.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0f.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0f.4 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0f.5 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:0f.6 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
7f:10.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D R2PCIe Agent (rev
↪  01)
7f:10.1 Performance counters: Intel Corporation Xeon
↪  E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D R2PCIe Agent
↪  (rev 01)
7f:10.5 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Ubox (rev 01)
7f:10.6 Performance counters: Intel Corporation Xeon
↪  E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D Ubox (rev 01)
7f:10.7 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Ubox (rev 01)
7f:12.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 0 (rev
↪  01)
7f:12.1 Performance counters: Intel Corporation Xeon
↪  E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 0
↪  (rev 01)
7f:12.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 0 Debug
↪  (rev 01)
7f:12.4 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 1 (rev
↪  01)
```

```
7f:12.5 Performance counters: Intel Corporation Xeon
↪  E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 1
↪  (rev 01)
7f:12.6 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 1 Debug
↪  (rev 01)
7f:13.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Target Address/Thermal/RAS (rev 01)
7f:13.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Target Address/Thermal/RAS (rev 01)
7f:13.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel Target Address Decoder (rev 01)
7f:13.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel Target Address Decoder (rev 01)
7f:13.6 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Broadcast (rev 01)
7f:13.7 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Global
↪  Broadcast (rev 01)
7f:14.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel 0 Thermal Control (rev 01)
7f:14.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel 1 Thermal Control (rev 01)
7f:14.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel 0 Error (rev 01)
7f:14.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel 1 Error (rev 01)
7f:14.4 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Interface (rev 01)
7f:14.5 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Interface (rev 01)
7f:14.6 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Interface (rev 01)
7f:14.7 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Interface (rev 01)
7f:16.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Target
↪  Address/Thermal/RAS (rev 01)
7f:16.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Target
↪  Address/Thermal/RAS (rev 01)
7f:16.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Channel Target
↪  Address Decoder (rev 01)
7f:16.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Channel Target
↪  Address Decoder (rev 01)
7f:16.6 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Broadcast (rev 01)
7f:16.7 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Global
↪  Broadcast (rev 01)
7f:17.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  1 - Channel 0 Thermal Control (rev 01)
7f:17.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  1 - Channel 1 Thermal Control (rev 01)
7f:17.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  1 - Channel 0 Error (rev 01)
7f:17.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  1 - Channel 1 Error (rev 01)
7f:17.4 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Interface (rev 01)
7f:17.5 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Interface (rev 01)
7f:17.6 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Interface (rev 01)
7f:17.7 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Interface (rev 01)
7f:1e.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
↪  (rev 01)
7f:1e.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
↪  (rev 01)
7f:1e.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
↪  (rev 01)
7f:1e.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
↪  (rev 01)
7f:1e.4 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
↪  (rev 01)
7f:1f.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
↪  (rev 01)
7f:1f.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
↪  (rev 01)
80:01.0 PCI bridge: Intel Corporation Xeon E7 v4/Xeon
↪  E5 v4/Xeon E3 v4/Xeon D PCI Express Root Port 1
↪  (rev 01)
```

```
80:02.0 PCI bridge: Intel Corporation Xeon E7 v4/Xeon
↪  E5 v4/Xeon E3 v4/Xeon D PCI Express Root Port 2
↪  (rev 01)
80:03.0 PCI bridge: Intel Corporation Xeon E7 v4/Xeon
↪  E5 v4/Xeon E3 v4/Xeon D PCI Express Root Port 3
↪  (rev 01)
80:05.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D
↪  Map/VTd_Misc/System Management (rev 01)
80:05.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D IIO Hot Plug (rev
↪  01)
80:05.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D IIO RAS/Control
↪  Status/Global Errors (rev 01)
80:05.4 PIC: Intel Corporation Xeon E7 v4/Xeon E5
↪  v4/Xeon E3 v4/Xeon D I/O APIC (rev 01)
81:00.0 RAID bus controller: LSI Logic / Symbios Logic
↪  MegaRAID SAS-3 3108 [Invader] (rev 02)
82:00.0 PCI bridge: PLX Technology, Inc. Device 8764
↪  (rev ab)
83:04.0 PCI bridge: PLX Technology, Inc. Device 8764
↪  (rev ab)
83:08.0 PCI bridge: PLX Technology, Inc. Device 8764
↪  (rev ab)
83:0c.0 PCI bridge: PLX Technology, Inc. Device 8764
↪  (rev ab)
84:00.0 Infiniband controller: Mellanox Technologies
↪  MT27700 Family [ConnectX-4]
85:00.0 3D controller: NVIDIA Corporation Device 15f9
↪  (rev a1)
86:00.0 3D controller: NVIDIA Corporation Device 15f9
↪  (rev a1)
87:00.0 PCI bridge: PLX Technology, Inc. Device 8780
↪  (rev ab)
88:04.0 PCI bridge: PLX Technology, Inc. Device 8780
↪  (rev ab)
88:08.0 PCI bridge: PLX Technology, Inc. Device 8780
↪  (rev ab)
88:0c.0 PCI bridge: PLX Technology, Inc. Device 8780
↪  (rev ab)
88:10.0 PCI bridge: PLX Technology, Inc. Device 8780
↪  (rev ab)
88:11.0 PCI bridge: PLX Technology, Inc. Device 8780
↪  (rev ab)
88:12.0 PCI bridge: PLX Technology, Inc. Device 8780
↪  (rev ab)
88:13.0 PCI bridge: PLX Technology, Inc. Device 8780
↪  (rev ab)
89:00.0 3D controller: NVIDIA Corporation Device 15f9
↪  (rev a1)
8a:00.0 3D controller: NVIDIA Corporation Device 15f9
↪  (rev a1)
8b:00.0 Infiniband controller: Mellanox Technologies
↪  MT27700 Family [ConnectX-4]
ff:08.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 0 (rev
↪  01)
ff:08.2 Performance counters: Intel Corporation Xeon
↪  E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 0 (rev
↪  01)
ff:08.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 0 (rev
↪  01)
ff:09.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 1 (rev
↪  01)
ff:09.2 Performance counters: Intel Corporation Xeon
↪  E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 1 (rev
↪  01)
ff:09.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D QPI Link 1 (rev
↪  01)
ff:0b.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D R3 QPI Link 0/1
↪  (rev 01)
ff:0b.1 Performance counters: Intel Corporation Xeon
↪  E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D R3 QPI Link 0/1
↪  (rev 01)
ff:0b.2 Performance counters: Intel Corporation Xeon
↪  E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D R3 QPI Link 0/1
↪  (rev 01)
ff:0b.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D R3 QPI Link Debug
↪  (rev 01)
ff:0c.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0c.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0c.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0c.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0c.4 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0c.5 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0c.6 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0c.7 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0d.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
```

```
ff:0d.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0d.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0d.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0d.4 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0d.5 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0d.6 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0d.7 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0e.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0e.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0e.2 Unassigned class [ffff]: Intel Corporation
↪  Device 6ff2 (rev 01)
ff:0e.3 Unassigned class [ffff]: Intel Corporation
↪  Device 6ff3 (rev 01)
ff:0f.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0f.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0f.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0f.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0f.4 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0f.5 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:0f.6 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Caching Agent (rev
↪  01)
ff:10.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D R2PCIe Agent (rev
↪  01)
ff:10.1 Performance counters: Intel Corporation Xeon
↪  E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D R2PCIe Agent
↪  (rev 01)
ff:10.5 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Ubox (rev 01)
ff:10.6 Performance counters: Intel Corporation Xeon
↪  E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D Ubox (rev 01)
ff:10.7 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Ubox (rev 01)
ff:12.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 0 (rev
↪  01)
ff:12.1 Performance counters: Intel Corporation Xeon
↪  E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 0
↪  (rev 01)
ff:12.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 0 Debug
↪  (rev 01)
ff:12.4 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 1 (rev
↪  01)
ff:12.5 Performance counters: Intel Corporation Xeon
↪  E7 v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 1
↪  (rev 01)
ff:12.6 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Home Agent 1 Debug
↪  (rev 01)
ff:13.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Target Address/Thermal/RAS (rev 01)
ff:13.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Target Address/Thermal/RAS (rev 01)
ff:13.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel Target Address Decoder (rev 01)
ff:13.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel Target Address Decoder (rev 01)
ff:13.6 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Broadcast (rev 01)
ff:13.7 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Global
↪  Broadcast (rev 01)
ff:14.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel 0 Thermal Control (rev 01)
ff:14.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel 1 Thermal Control (rev 01)
ff:14.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel 0 Error (rev 01)
```

```
ff:14.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  0 - Channel 1 Error (rev 01)
ff:14.4 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Interface (rev 01)
ff:14.5 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Interface (rev 01)
ff:14.6 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Interface (rev 01)
ff:14.7 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 0/1
↪  Interface (rev 01)
ff:16.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Target
↪  Address/Thermal/RAS (rev 01)
ff:16.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Target
↪  Address/Thermal/RAS (rev 01)
ff:16.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Channel Target
↪  Address Decoder (rev 01)
ff:16.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Channel Target
↪  Address Decoder (rev 01)
ff:16.6 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Broadcast (rev 01)
ff:16.7 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Global
↪  Broadcast (rev 01)
ff:17.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  1 - Channel 0 Thermal Control (rev 01)
ff:17.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  1 - Channel 1 Thermal Control (rev 01)
ff:17.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  1 - Channel 0 Error (rev 01)
ff:17.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Memory Controller
↪  1 - Channel 1 Error (rev 01)
ff:17.4 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Interface (rev 01)
ff:17.5 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Interface (rev 01)
ff:17.6 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Interface (rev 01)
ff:17.7 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D DDRIO Channel 2/3
↪  Interface (rev 01)
ff:1e.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
↪  (rev 01)
ff:1e.1 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
↪  (rev 01)
ff:1e.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
↪  (rev 01)
ff:1e.3 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
↪  (rev 01)
ff:1e.4 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
↪  (rev 01)
ff:1f.0 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
↪  (rev 01)
ff:1f.2 System peripheral: Intel Corporation Xeon E7
↪  v4/Xeon E5 v4/Xeon E3 v4/Xeon D Power Control Unit
↪  (rev 01)
```

ARTIFACT EVALUATION

Verification and validation studies: The inference output generated by our SBNN is compared with Pytorch and Tensorflow implementation with the same input image and weights.

Accuracy and precision of timings: We use CUDA event for the time measurement related to CUDA implementation. We use the average time for 5 times' execution.