ARCHIVIO ISTITUZIONALE
DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

A Scalable Near-Memory Architecture for Training Deep Neural Networks on Large In-Memory Datasets

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:
Schuiki F., Schaffner M., Gurkaynak F.K., Benini L. (2019). A Scalable Near-Memory Architecture for Training Deep Neural Networks on Large In-Memory Datasets. IEEE TRANSACTIONS ON COMPUTERS, 68(4), 484-497 [10.1109/TC.2018.2876312].

Availability:
This version is available at: https://hdl.handle.net/11585/702703 since: 2019-10-18

Published:
DOI: http://doi.org/10.1109/TC.2018.2876312

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

08 September 2024

This is the post peer-review accepted manuscript of:

F. Schuiki, M. Schaffner, F. K. Gürkaynak and L. Benini, "A Scalable Near-Memory Architecture for Training Deep Neural Networks on Large In-Memory Datasets", in IEEE Transactions on Computers, vol. 68, no. 4, pp. 484-497, 1 April 2019. doi: 10.1109/TC.2018.2876312

The published version is available online at: https://doi.org/10.1109/TC.2018.2876312

© 2018 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works

A Scalable Near-Memory Architecture for Training Deep Neural Networks on Large In-Memory Datasets

Fabian Schuiki, Michael Schaffner, Frank K. Gürkaynak, and Luca Benini, *Fellow, IEEE*

Abstract—Most investigations into near-memory hardware accelerators for deep neural networks have primarily focused on inference, while the potential of accelerating training has received relatively little attention so far. Based on an in-depth analysis of the key computational patterns in state-of-the-art gradient-based training methods, we propose an efficient near-memory acceleration engine called NTX that can be used to train state-of-the-art deep convolutional neural networks at scale. Our main contributions are: (i) a loose coupling of RISC-V cores and NTX co-processors reducing offloading overhead by $7\times$ over previously published results; (ii) an optimized IEEE 754 compliant data path for fast high-precision convolutions and gradient propagation; (iii) evaluation of near-memory computing with NTX embedded into residual area on the Logic Base die of a Hybrid Memory Cube; and (iv) a scaling analysis to meshes of HMCs in a data center scenario. We demonstrate a $2.7\times$ energy efficiency improvement of NTX over contemporary GPUs at $4.4\times$ less silicon area, and a compute performance of $1.2\,\mathrm{Tflop/s}$ for training large state-of-the-art networks with full floating-point precision. At the data center scale, a mesh of NTX achieves above 95% parallel and energy efficiency, while providing $2.1\times$ energy savings or $3.1\times$ performance improvement over a GPU-based system.

Index Terms—Parallel architectures, memory structures, memory hierarchy, machine learning, neural nets

1 Introduction

Modern Deep Neural Networks (DNNs) have to be trained on clusters of GPUs and millions of sample images to be competitive [1]. Complex networks can take weeks to converge during which the involved compute machinery consumes megajoules of energy to perform the exa-scale amount of operations required. Inference, i.e. evaluating a network for a given input, provides many knobs for tuning and optimization. Substantial research has been performed in this direction and many good hardware accelerators have been proposed to improve inference speed and energy efficiency [2]. The training of DNNs is much harder to do and many of these optimizations do no longer apply. Stochastic Gradient Descent (SGD) is the standard algorithm used to train such deep networks [3]. Consider Figure 1 which shows the data dependencies when training a simple neural network. While inference is concerned only with finding y, training aims at finding the gradients ($\Delta\theta$) which introduces a data dependency that requires us to temporarily store the output x_i, y of *every* layer. This also prevents optimizations such as fusing activation or subsampling functions with the preceding layer.

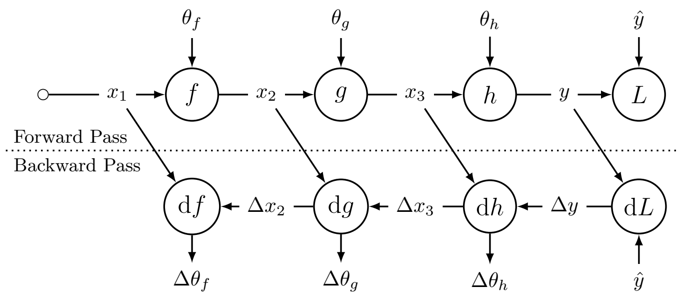

Figure 1. Data dependency graph of the forward pass (above) and backward pass (below). f, g, h are DNN layers, L is the loss function, x_1, x_2, x_3 and $\theta_1, \theta_2, \theta_3$ are the layer activations and parameters. The backward pass introduces a data dependency between the first node f and the last node $\mathrm{d}f$. Thus intermediate activations need to be stored.

While it has been shown that inference is robust to lowering arithmetic precision [2], the impact of fixed-point or reduced-precision floating-point (FP) arithmetic on training is not yet fully understood (see Section 5). Until additional light is shed on the topic, a training accelerator must support 32 bit FP arithmetic to be able to compete with the ubiquitous GPU. Existing accelerators require a significant amount of custom silicon and often additional memory and computational resources to function. In this paper we show that a processing system embedded in the Logic Base (LoB) of a Hybrid Memory Cube (HMC) is a competitive and scalable option for training DNNs in the data center. The proposed architecture is based on the earlier NeuroStream (NS) [4] inference engine which introduced the concept of streaming coprocessors based on nested hardware loops and address generators to HMC. We show that such corprocessors can be extended to training workloads and be made more efficient by increasing their level of autonomy. High overall data center-level energy efficiency can be achieved by distributing training over multiple such HMCs. The key contributions of this paper are:

1) A compute architecture featuring a few RISC-V cores loosely coupled with several NTX co-processors (1:8 ratio) capable of managing computation and L1 memory access. One RISC-V core can manage 8 NTX with a reduced number of instructions, hence the von Neumann

Manuscript received 18 Feb 2018; revised 4 Aug and 26 Sep 2018.

bottleneck is relaxed without compromising flexibility (Section 2).

2) An optimized data path in the NTX for high-precision convolutions and gradient propagation coupled with an effective TCDM/DMA data transfer hardware that eliminates area and power overhead of large caches, by leveraging the predictability of DNN memory patterns (Section 3).
3) Significant computational capabilities at no additional silicon area cost in the LoB of a HMC, which we show to outperform GPUs and other accelerators in terms of silicon and energy efficiency (Section 4).
4) A competitive scaling to meshes of HMCs that can replace existing GPU-based solutions in a data center setting, the improved efficiency of which translates to an increase in computational power and significant savings in power, cooling, and equipment cost (Section 4).

The remainder of this paper is organized as follows: Section 2 describes the proposed hardware architecture and Section 3 shows the execution model of DNN layers. Section 4 presents experimental results and comparisons to other accelerators. The remaining sections describe related and future work, and provide a conclusion.

2 Architecture

The LoB of a HMC offers a unique opportunity to introduce a Processor-in-Memory (PiM) as depicted in Figure 2. The memory dies are subdivided into vertically connected vaults, with individual memory controllers on the LoB. Traffic between the serial links and vaults is transported by the means of an efficient all-to-all network [5], [6]. Our architecture consists of multiple processing clusters attached to a crossbar, which thus gain full access to the entire memory space of the HMC. The architecture was chosen to offer the same bandwidth as in [4] such that the interconnect offers the full bandwidth required by the aggregate cluster ports. The memory cube is attached to a host CPU or other cubes via the four serial links. The on-chip network is responsible for arbitration of traffic between the serial links, the DRAM, and the PiM. This arbitration can be prioritized such that external memory accesses from the serial links are given priority over internal ones originating in the processing system. It also allows requests from the PiM to be routed to the serial links for inter-HMC communication.

2.1 Processing Cluster

We combine a general purpose RISC-V processor core [7] with multiple NTX FP streaming co-processors. Both operate on a $128\,\mathrm{kB}$ TCDM which offers a shared memory space with single-cycle access. The memory is divided into 32 banks that are connected to the processors via a low-latency logarithmic interconnect. These form a cluster which also contains a DMA engine that is capable of transferring two-dimensional planes of data between the TCDM and the HMC's memory space. This solution has proven to be more area and energy efficient than implicit caches, and the DMA can anticipate and time block data transfers precisely, thereby hiding latency. The RISC-V processors perform address calculation and control data movement via the DMA.

Table 1
Arithmetic error comparison between a conventional float32 FPU and the NTX 32 bit FMAC unit, for a full 3×3 convolution layer of GoogLeNet [1], with respect to a common baseline (64 bit float).

Implementation	RMSE	Relative Error	
		Maximum	Median
Intel CPU (float32)	$1.83 \cdot 10^{-7}$	$5.42 \cdot 10^{-3}$	$9.40 \cdot 10^{-8}$
NTX (32 bit FMAC)	$1.08 \cdot 10^{-7}$	$1.19 \cdot 10^{-7}$	$5.97 \cdot 10^{-8}$

Actual computation is performed on the data in the TCDM by the NTX co-processors which we describe in the next section.

Address translation is performed either in software or via a lean Memory Management Unit (MMU) with Translation Look-aside Buffer (TLB) as described in [5]. This allows the PiM to directly operate on virtual addresses issued by the host. If there are multiple HMCs attached to the host, care must be taken since the PiMs can only access the memory in the HMC that they reside in. An additional explicitly managed memory outside the clusters, labeled "L2" in Figure 2, holds the RISC-V binary executed by the processors and additional shared variables. The binary is loaded from DRAM.

2.2 Network Training Accelerator (NTX)

The computations involved in training DNNs are highly regular. To leverage this feature we developed NTX, a FP streaming co-processor that operates directly on the TCDM. Conceptually the NTX co-processor is similar to the one presented in [4], but it is a complete redesign optimized for performance and training. The streaming nature of the co-processor alleviates the need for a register file and corresponding load/store instructions. The architecture of NTX is depicted in Figure 3. It consists of four main blocks: (i) the FPU containing the main data path, (ii) the register interface for command offloading, (iii) the controller that decodes the commands and issues micro-instructions to the FPU, and (iv) the address generators and hardware loops.

2.3 FMAC and FPU

The FPU in NTX can perform fast FMAC operations with single-cycle throughput. It is based on a Partial Carry-Save (PCS) accumulator which aggregates the $48\,\mathrm{bit}$ multiplication result at full fixed-point precision ($\approx$300 bit). After accumulation the partial sums are reduced in multiple pipelined segments. In order to reach an operating frequency above $1.5\,\mathrm{GHz}$ in $28\,\mathrm{nm}$ (SS 125°C 1.0 V), two segments are sufficient. The employed format has been aligned with IEEE 754 32 bit floats. The wide accumulator and deferred rounding allows NTX to achieve higher precision than conventional floating-point units (FPUs) for reduction operations such as convolutions. Analysis has shown that in a full 3×3 convolution layer of GoogLeNet [1] the Root Mean Squared Error (RMSE) of NTX is $1.7\times$ lower than that of a 32 bit FPU, with respect to a common baseline (64 bit float). See Table 1.

The FMAC unit allows NTX to compute inner/outer product and vector addition/multiplication. An additional comparator, index counter, and ALU register enable various additional commands such as finding minima/maxima, ReLU, thresholding and masking, and copy/memset.

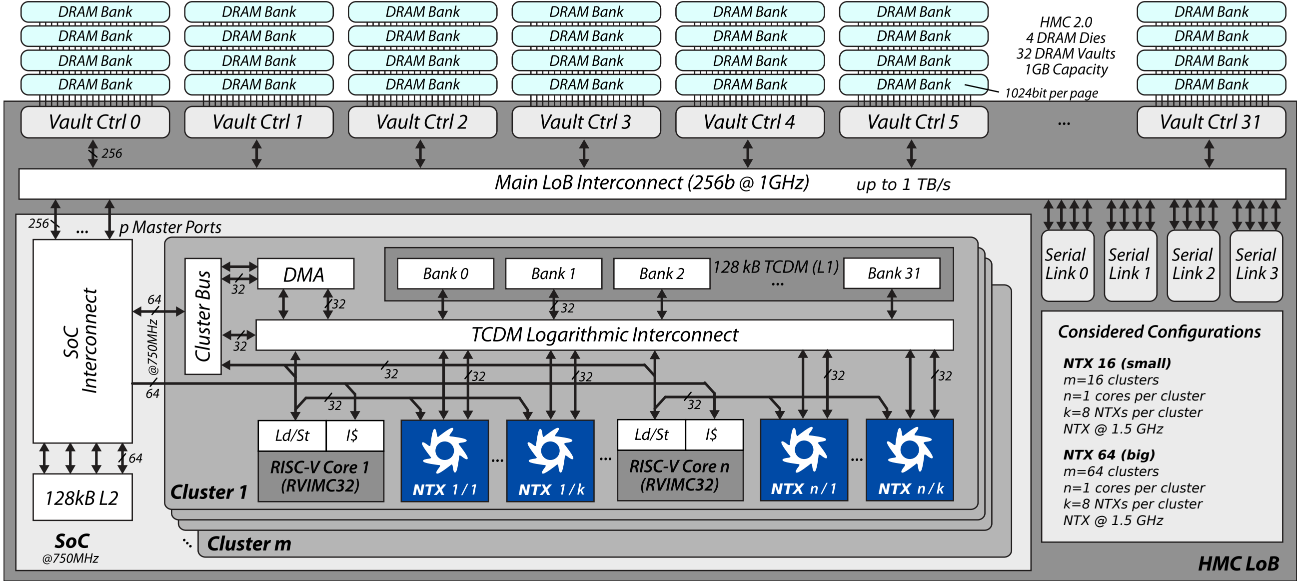

Figure 2. Top-level block diagram of one HMC enhanced with m processing clusters. The LoB contains the vault controllers, main interconnect, and the four serial links that lead off-cube. The proposed processing clusters attach directly to the main interconnect and gain full access to the HMC's memory space and the serial links. Each cluster consists of a DMA unit, a TCDM, and one or more RISC-V processor cores augmented with NTX streaming co-processors. We designed the NTX to operate at $1.5\,\mathrm{GHz}$, while the remaining additions to the system operate at $750\,\mathrm{MHz}$.

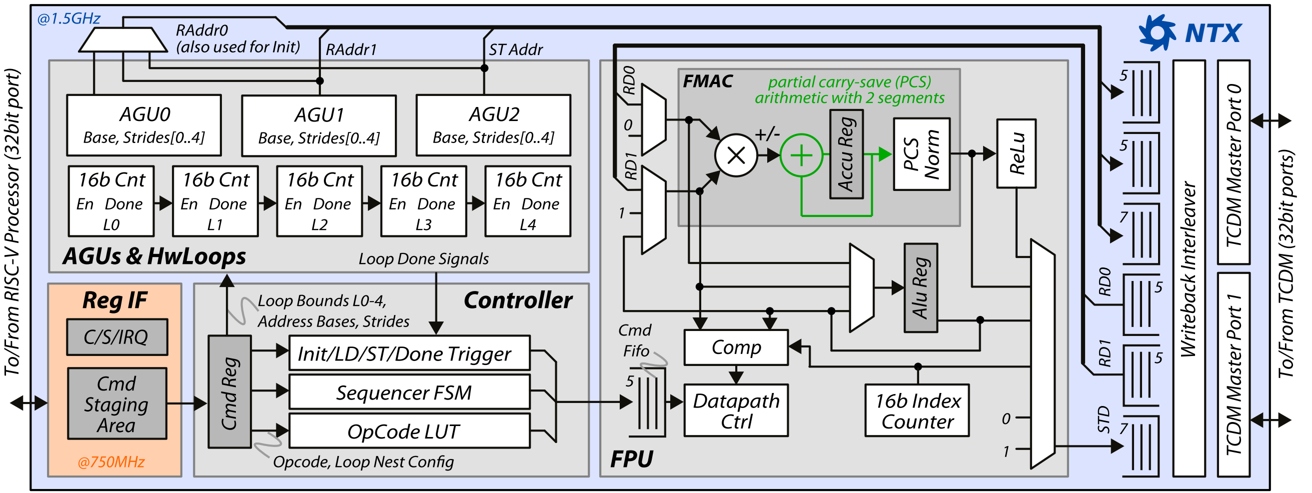

Figure 3. Block diagram of the NTX accelerator. It contains 5 hardware loops; 3 address generator units; a double-buffered command staging area in the register interface; a main controller; and a FPU with a comparator, index counter (for argmax calculations) and a fast FMAC unit. The employed depths for all FIFOs are indicated and have been determined in simulations for a TCDM read-latency of 1 cycle.

2.4 Hardware Loops and Address Generation

At the core of address generation in NTX are the five Hardware Loops (HWLs). Each loop is managed by a 16 bit counter that has a programmable maximum count register N_i. Additionally, the counter can be explicitly enabled or disabled, and it has a signal indicating whether the counter has reached its maximum value and is about to reset to zero. To support nesting, each counter is enabled by the previous counter's "done" signal. The first counter (L0) is only disabled upon a pipeline stall. The "done" signal of the last counter (L4) indicates that all loop iterations have been performed. The enable signals of all counters are concatenated into a 5 bit output.

Three Address Generation Units (AGUs) allow NTX to keep track of three pointers into memory. Each unit consists of a 32 bit register holding the address and an adder. The address is incremented by one of five programmable step sizes p_i, each of which corresponds to one of the hardware loops. The enabled counter with the highest index dictates the chosen stride. This allows addresses of the form

$$A = A_{\text{base}} + i_0 s_0 + i_1 s_1 + i_2 s_2 + i_3 s_3 + i_4 s_4 \tag{1}$$

to be calculated, but using only one addition per cycle. The conversion from strides s_i to step sizes p_i is trivial:

$$p_0 = s_0 \tag{2}$$

$$p_i = s_i - (N_{i-1} - 1) \cdot p_{i-1} \tag{3}$$

where N_i is the iteration limit of an HWL. This conversion can be performed by the controlling CPU core when

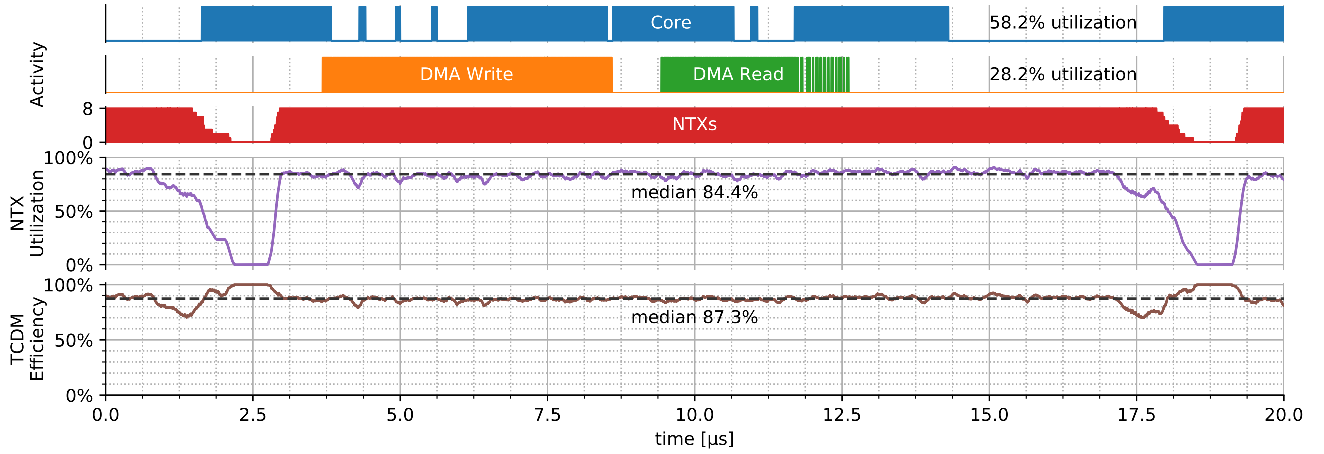

Figure 4. A 3×3 convolution running on one cluster. The periods of activity of the RISC-V processor and the DMA unit are shown as blocks, the activity of the co-processors is indicated as the number of active NTXs. The processor and DMA are busy during 58.2% and 28.2% of the computation, respectively. The utilization of the co-processors is given as percent of maximum throughput. The efficiency of the TCDM is given as the percentage of memory requests serviced per cycle; the remaining requests stall due to conflicts. The system has a banking factor of 1.8.

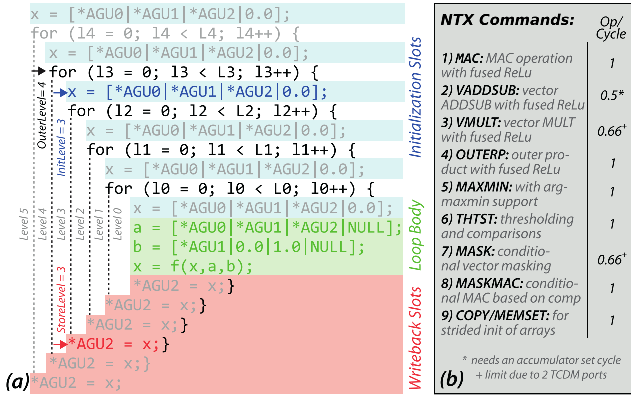

Figure 5. The structure of nested loops in C code that can be directly offloaded to NTX (a), and an overview of the supported commands and their throughput (b).

programming a command, for example as part of a driver library.

Figure 5a shows the pseudo code structure of nested loops that NTX can natively perform. The number of loops (*outer level*), position of the accumulator initialization (*init level*), and position of the accumulator write back (*store level*) are fully programmable. The AGUs provide addresses for the memory reads and writes depicted. The operation performed by the FPU always occurs in the innermost loop body and can be configured to be one of the commands listed in Figure 5b.

2.5 Offloading Support

Offloading to NTX has been enhanced with respect to the earlier NeuroStream (NS) [4] inference engine to significantly improve efficiency in training workloads. The three improvements in offloading are: (i) a command staging area, (ii) an increased number of hardware loops, and (iii) a third address generator. The following paragraphs briefly explains the impact of each.

(i) In NS, configuration of addresses, strides, loops, and the initialization of the accumulator are performed via a

Table 2
Comparison of the number of offloads necessary and execution time of an offloaded command for NTX and NeuroStream (NS) [4], for different convolution layers of GoogLeNet [1]. NS requires one offload per output pixel, whereas the increased number of hardware loops and the third address generation unit of NTX allow it to compute many output pixels per offloaded command.

Kernel	Output	Offloads		Busy Cycles per Offload	
		NS	NTX	NS	NTX
7x7x3	112x112x64	802 816	64	147	1 843 968
3x3x64	56x56x192	602 112	192	576	1 806 336
1x1x256	28x28x64	50 176	64	256	200 704
1x1x512	14x14x192	37 632	192	512	100 352

command register. This register is mapped into the controlling CPU's memory space and the written commands are pushed into a FIFO. This allows the CPU to enqueue configuration updates while a computation is still ongoing. The computation itself is also a command which is popped off the FIFO only upon completion. This also implies that the FIFO needs to be deep enough to hold all commands necessary to configure the next computation, lest the CPU has to stall. NS used depth 8 for these FIFOs, causing the CPU to stall frequently.

NTX improves on this by exposing the configuration registers as a memory-mapped "staging area". As such the CPU can directly address and modify the registers. A computation is launched by writing to the command register, which is special in the sense that it causes the entire configuration to be copied to an internal "shadow" register. This allows the CPU to immediately go ahead and configure the next operation without disturbing the current one. Furthermore, parts of the configuration that do not change between commands need not be written again since the staging area is persistent. It is worthwhile noting that the size of the staging area and its shadow copy in NTX is roughly the same as the command FIFO and the corresponding registers in NS, but the former offer significantly higher ease of use. All NTX controlled by a core are also accessible via a broadcast address, which further reduces offloading

time for configuring common parameters.

(ii) We observe that a convolution as it appears in DNNs has six nested loops: three that iterate over each output pixel, and three to perform the per-pixel reduction of the input dimensions (3D input and output, 4D weights). NS offers three hardware loops, which allows the 3D per-pixel reduction to be expressed in one command. To compute the first convolution of GoogLeNet [1] which has a $7 \times 7 \times 3$ kernel and yields a $112 \times 112 \times 64$ output, 802816 offloads need to be issued by the CPU each of which ideally takes only 147 cycles. This leaves only few cycles to coordinate DMA transfers and configure the next command, thus limiting the number of NeuroStreams that can be controlled by one CPU.

NTX improves on this by increasing the number of hardware loops to five. This allows multiple output pixels to be calculated with one offload. For the aforementioned convolution, this translates to only 64 offloads that need to be issued, each of which ideally takes 1843968 cycles. In practice the size of the offloaded computation is now bounded by the tile size that fits into the TCDM, thus the CPU only needs to issue one offload per NTX per tile. This reduces the control overhead of NTX to almost zero. The CPU is now free to do more elaborate data transfers, for example issuing multiple small DMA transfers to copy slices of a tensor and performing zero-padding, thus not requiring that the data be laid out in memory in a zigzag tiling fashion as described in [4]. This is an important improvement, since such a tiling cannot be maintained during training without significant data reshuffling between layers, which would severely reduce the energy efficiency and inflate bandwidth. Table 2 shows this effect for select convolutions in GoogLeNet [1].

(iii) To allow NTX to calculate multiple pixels in the output image with one offload, we added a third address generator to maintain a pointer for autonomously writing back multiple results to memory. This in contrast to NS [4] which requires an explicit command from the CPU to store the accumulated value.

3 NTX Execution

The combination of RISC-V processors and dedicated FP streaming co-processors makes our architecture very flexible. It is a many-core platform with explicitly managed scratchpad memories, where data copies are performed by a DMA engine and bulk computations by NTX co-processors in parallel to a running program. The proposed architecture allows for entire DNN training batches to be performed completely in memory, without intervention from a host outside the HMC, as follows. Starting from a reference implementation of a training step in C or C++, nested loops of the form described in Section 2 are amenable to acceleration on NTX. This includes the bulk operation of all DNN layers. These loops are replaced by an offload sequence consisting of writes to the eight staging areas. Furthermore, the input and output data of each loop nest must be tiled and data movement appropriately scheduled, as described below.

Figure 4 shows the execution of one tile of a 3×3 convolution on NTX. The RISC-V core first configures and launches the main computation on NTX, then controls the DMA to write back the output of the previous tile and read the input of the next one. Additional tasks such as zero padding and address computation are performed in the background. The short period of NTX idleness in between tiles is due to the core using the NTX to initialize the next tile, which is a quick command that terminates faster than the core can configure the next big computation.

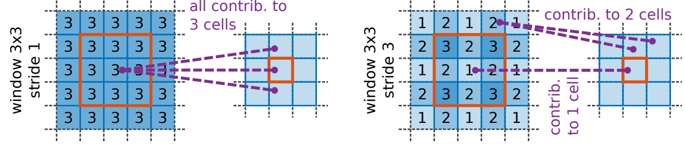

Figure 6. Irregularity introduced by stride in stencil operations such as convolution and max pooling. With a stride of one (left), all input cells contribute to the same number of output cells. With a stride of three (right), input cells contribute to one, two, or three output cells.

3.1 Memory and Tiling

As described in Section 2 and [4], [5], the core and NTX operate directly on a scratchpad memory inside the cluster. A DMA unit in conjunction with a lean MMU is used to copy data from DRAM into cluster memory, where the accelerators operate on them. This mechanism is similar to how caching works on CPUs/GPUs, but is explicitly managed by the programmer.

The scratchpad memory in the cluster is limited in size. To evaluate an entire convolution layer for example, the input and output data are tiled to fit into memory. The tiles need to overlap in the convolution case. The DMA unit can run in parallel to the computation and is used to write back previous results and read next inputs while a computation is ongoing (double buffering), as can be seen in Figure 4. In [4] the authors made use of 4D tiling, which requires that the data is already laid out in such a tiled fashion in DRAM, including replication of the overlapping areas. This allows the DMA to copy a tile in a single consecutive transfer, requiring little control by the core. For training this scheme is infeasible, since forward and backward passes require different tile sizes, and the data would need to be retiled after several subsampling layers to maintain a sufficient tile size. This retiling translates into no-op movement of data, which wastes bandwidth and energy.

The improved offloading scheme described in Section 2.5 and increased independence of NTX compared to [4] frees up significant RISC-V core resources. This allows us to now store tensors in memory as dense chunks of FP values, without any replication or tiling pre-applied. To transfer tiles, we task the core with issuing multiple DMA transfers, each of which copies one consecutive stripe of data. Zero padding can also be performed by the core in this way. Hence we can drop the requirement of the data being laid out in memory in a pre-tiled fashion.

3.2 Strided Stencil Operations

A stride greater than one in stencil operations such as convolution and pooling causes an irregularity during training. For example, a strided convolution can be thought of as a regular convolution where a subset of the output

pixels are discarded. The backward pass correspondingly can be represented roughly as a sparse convolution where the discarded pixels are 0. For efficiency reasons we would like to skip multiplications with 0, effectively leveraging the sparsity of the problem. However NTX cannot change the number of summands within the course of one operation, so we must perform convolutions that have a constant number of operations required per pixel. This does not hold for strided convolutions in the backward pass, where the input derivative contains contributions from a varying number of output pixels. See figure Figure 6. We observe that we can subdivide the pixels of the input derivative into different categories: Each pixel subset can be computed as a regular convolution with a subset of the filter weights, and the overall result can be found by interleaving the subset results. This scheme allows us to decompose a sparse convolution (as found in the derivative of strided convolutions) into multiple dense convolutions each contributing a subset of the result pixels.

3.3 Special Functions (exp, log, div, sqrt)

There is no dedicated hardware to evaluate special functions such as division, exp, log, square roots, or arbitrary powers. These are needed for the softmax layer or various forms of normalization. As the number of such operations is typically very low (in the order of a few thousands per training step), it is feasible to implement them using iterative algorithms on the NTX, calculating multiple results in parallel. We found that for tens to hundreds of inputs, pipeline latency can be hidden and the evaluation takes on the order of 30 to 100 cycles per element.

3.4 Communication across HMCs

The serial links in the HMC are accessible to the processor cores and DMA units in each cluster. This allows a mesh of HMCs to be programmed in a similar way as a two-tiered network of compute nodes. Within the HMC, clusters may exchange data via the DRAM and L2. Across HMCs, the processor cores may cooperate to perform complex systolic operations via the serial links. Section 4.9 provides an example of this.

4 Experimental Results and Analysis

In this section we evaluate the silicon and energy efficiency of our proposed architecture and compare it against NeuroStream, the most closely related other accelerator [4]. Furthermore we investigate the effects of voltage and frequency scaling and the impact of multiple logic dies per memory cube. We conclude by comparing different NTX configurations against existing accelerators and evaluate the data center scale impact of our architecture.

4.1 Methodology

4.1.1 DRAM Power

We model the power consumption of the vault controllers, DRAM dies, and HMC interconnect as the following relationship:

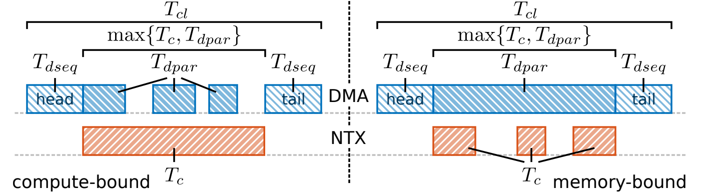

Figure 7. Execution time of a kernel running on a cluster. See Section 4.1.2 for a details. T_{dseq} corresponds to memory transfers that need to happen before and after the main computation, e.g. first data fetch and last data store. T_{dpar} corresponds to transfers that can happen in parallel to the computation T_{c}. Shown are a compute-bound case where T_{c} dominates, and a memory-bound case where T_{dpar} dominates.

$$P_{\text{dram}}(B) = 7.9\,\text{W} + B \cdot 21.5\,\text{mW}\,\text{s}/\text{GB},$$

where B is the requested bandwidth. We call this the "DRAM" power. This model is based on the observation in [4] that $7.9\,\text{W}$ are consumed in a $1\,\text{GB}$ cube under no traffic into DRAM ($50\,\text{nm}$, see Section 4.1.6). Under an average traffic of $51.2\,\text{GB/s}$ caused by their investigated workloads, this increases to the reported $9.0\,\text{W}$, a bandwidth-dependent power increase of $21.5\,\text{mW}\,\text{s}/\text{GB}$. These estimates are conservative and do not consider further power-saving measures, such as Voltage and Frequency Scaling (VFS) or power gating of HMC components.

4.1.2 Cluster Power

We have synthesized our design for a $28\,\text{nm}$ Fully Depleted Silicon On Insulator (FD-SOI) technology using Synopsys Design Compiler, which we also use to estimate power based on simulation traces. The Register Transfer Level (RTL) model was back-annotated with timing information obtained from the synthesized design at $125\,^{\circ}\text{C}/1.0\,\text{V}$ (slow-slow corner). We execute the worst-case kernel in an RTL simulation of the cluster, which gives us a cycle-accurate picture of the computation. This is depicted in Figure 4, and furthermore gives us an estimate of the cluster's power consumption, $165\,\text{pJ}$ per clock cycle in this case. The worst-case kernel is a convolution which makes full use of the FPUs in the NTXs and has a high utilization of the DMA. Memory-bound kernels consume less power, since the FPU utilization is lower, reducing its power contribution. This simulation also gives us realistic utilization efficiencies of $\eta_{\text{c}} = 84\%$ and $\eta_{\text{d}} = 87\%$ for NTX and TCDM, respectively.

4.1.3 Network Layer Energy

To evaluate applications, we model the execution of individual network layers. The computation and data movement performed by a cluster is very predictable. For each network layer we therefore compute the number of FP operations necessary, as well as the amount of data that needs to be transferred. The latter we further split into data that must be moved before computation can start (*head*), data that can be moved in parallel to the computation, and data that must be moved once the computation completes (*tail*). This closely models the double buffering possible by overlapping operation of the DMA and NTX within the cluster. For each

kernel we determine the execution time of the computation (T_c) and DMA transfers (T_{dpar}, T_{dseq}) as:

$$T_c = N_c \,/\, \eta_c \, r_c \, f \quad [s] \quad (4)$$

$$T_{dpar} = (D_{dma} - D_{head} - D_{tail}) \,/\, \eta_d \, r_d \, f \quad [s] \quad (5)$$

$$T_{dseq} = (D_{head} + D_{tail}) \,/\, \eta_d \, r_d \, f \quad [s] \quad (6)$$

where T_{dpar} represents DMA transfers that can run in parallel with computation and T_{dseq} those that need to happen before and after. In more detail, N_c and D_{dma} are the total number of compute operations performed and bytes transferred by the kernel; r_c are the peak compute operations per cycle of the cluster; and r_d is the peak bandwidth of the DMA per cycle. For the architecture with 8 NTXs presented in Section 2, $r_c = 8\,\mathrm{op}$ and $r_d = 4\,\mathrm{B}$. η_c and η_d account for inefficiencies such as interconnect contentions and are determined empirically from simulations. We then formulate the execution time, requested bandwidth, and power consumption of the kernel as:

$$T_{cl} = \max\{T_c, T_{dpar}\} + T_{dseq} \quad [s] \quad (7)$$

$$B_{cl} = D_{dma} \,/\, T_{cl} \quad [B/s] \quad (8)$$

$$P_{cl} = 165\,\mathrm{pJ} \cdot f \quad [W] \quad (9)$$

See Figure 7 for a visual explanation. Note that we issue DMA transfers in chunks of multiple kB and the engine is capable of having multiple simultaneous transfers in flight. This allows us to hide the latency into DRAM which we estimate to be on the order of 40 core cycles. It is crucial that we fully saturate the precious bandwidth into DRAM when performing strided memory accesses, e.g. when transferring a tile of a tensor. There the length of the tile's innermost dimension is critical, as it determines the length of one burst accesses. Since we have full control over the tiling, we can ensure that a tile has at least 8 elements along its shortest dimension. This yields consecutive accesses of at least 32 B, which is the minimum block size in an HMC [6].

4.1.4 Cube Power

Based on the above, we model the requested bandwidth, power consumption, and energy efficiency of a kernel parallelized on a HMC with K clusters as

$$B = K \cdot B_{cl} \quad [B/s] \quad (10)$$

$$T = T_{cl} \,/\, K \quad [s] \quad (11)$$

$$P = P_{dram}(B) + K \cdot P_{cl} \quad [W] \quad (12)$$

$$\eta = 2\,N_c \,/\, P\,T \quad [\mathrm{flop/s\,W}] \quad (13)$$

The parallelization is achieved by distributing the tiles of computation described in Section 3 across the clusters of a cube.

4.1.5 Network Training Energy

We then model different layers of DNNs as the amount of computation and data transfers necessary. This also gives us a per-layer estimate of the number of parameters and intermediate activations. We proceed to model each of the investigated networks as a sequence of these layers, giving us a realistic estimate for the execution time of the inference and training steps of one image on the proposed architecture, together with the associated bandwidth requirement. We then further use the execution time and the cluster and bandwidth-dependent DRAM/LoB power determined above to estimate the overall energy required to process one image.

4.1.6 Technology Scaling

We use internal comparisons and publicly available information to estimate the effect of scaling down the technology node of the LoB from the 28 nm FD-SOI process investigated by us to a more modern 14 nm FinFET node [8], [9]. For this change we observed across several designs an increase of $1.4\times$ in speed, a decrease of $0.4\times$ in area, and $0.7\times$ in dynamic power dissipation.

To our knowledge there is no publicly available information on the DRAM characteristics of HMCs. "SMCSim" [5] assumes them to be similar to the MT41J512M8 device by Micron, which is based on a 50 nm process. Given the manufacturer and [10], the device seems to be a reasonable reference for early HMCs. We estimate the DRAM technology scaling factor for power consumption to be 0.87, by comparing the supply currents and voltages of this device to the newer 30 nm MT40A512M8.

4.1.7 GPU Efficiency Estimation

We estimate GPU efficiency based on the training time per image measured by [11], [12]. For each network we compute the amount of flop necessary per image based on our model of the network. This yields an estimate of the actual throughput in flop/s achieved. Assuming the GPU can reach its TDP under such highly optimized workloads (e.g. cuDNN), we determine the energy efficiency as the ratio between that throughput and the TDP. We do not assume optimizations such as Winograd to be performed on the GPU, and as such overestimate the number of FP operations performed, making the estimated energy efficiency optimistic. Furthermore, this excludes the power consumed by the CPU to constantly push training data into GPU memory.

4.2 Precision, Sparsity, Compression

Training a DNN with reduced FP precision or even fixed-point arithmetic is much harder than doing the same for inference. The intuition here is that the SGD algorithm performs smaller and smaller changes to the parameters as training progresses. If these changes fall beneath the numeric precision, the algorithm effectively stops converging. There is no a priori obvious range of magnitudes within which parameters fall, thus the arithmetic must support a significant dynamic range without additional prior analysis. NTX employs 32 bit FP arithmetic which is commonly used in deep learning frameworks and CPUs/GPUs, rendering such analysis unnecessary. Note that there is evidence that training is possible in fixed-point arithmetic with little accuracy loss in some cases [13]. However, results tend to be limited to specific networks and other work suggests that reducing precision may not be feasible at all without incurring significant accuracy loss [14].

Recent work on network compression and pruning techniques has shown promising results in terms of reducing computational overhead [15]. The general purpose nature of the RISC-V processors in our architecture allows some of

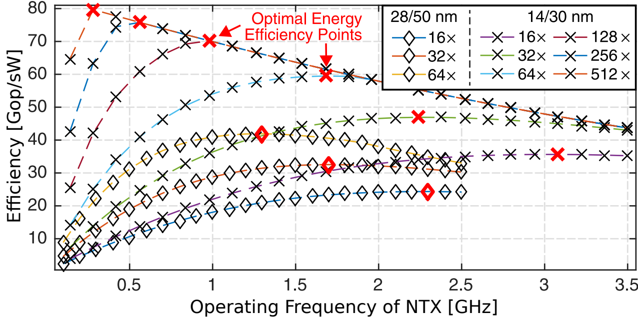

Figure 8. Energy efficiency versus operating frequency of different numbers of clusters and different technology nodes: $28\,\mathrm{nm}$ logic / $50\,\mathrm{nm}$ DRAM, $14\,\mathrm{nm}$ logic / $30\,\mathrm{nm}$ DRAM. The clusters perform a 3×3 convolution. The voltage is varied between $0.6\,\mathrm{V}$ and $1.2\,\mathrm{V}$ in proportion to the frequency. Points of highest efficiency of each configuration are marked in bold red. The internal bandwidth of the HMC puts an upper bound on the achievable energy efficiency, visible in the upper half of the graph.

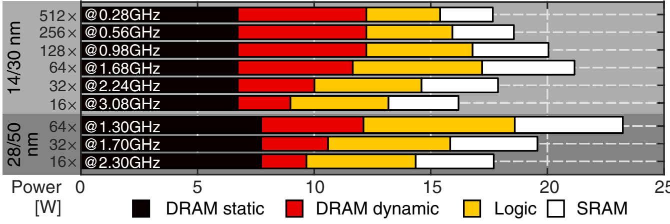

Figure 9. Power dissipation of different configurations, evaluated at their most-efficient operating point in Figure 8. Note that even the massively parallel configurations with more than 64 clusters are below a TDP of $25\,\mathrm{W}$.

these schemes to be implemented. For example entire convolutions may be skipped or certain forms of decompression and re-compression may be performed on the processor cores. The NTX has not been optimized for sparse tensor operations however, and we leave their detailed analysis for future work.

4.3 Voltage and Frequency Scaling (VFS)

In this section we assess the efficiency of NTX at different operating points. We vary the supply voltage between $0.6\,\mathrm{V}$ and $1.2\,\mathrm{V}$; and the operating frequency between $0.1\,\mathrm{GHz}$ and $2.5\,\mathrm{GHz}$ for the $28\,\mathrm{nm}$ process and $0.14\,\mathrm{GHz}$

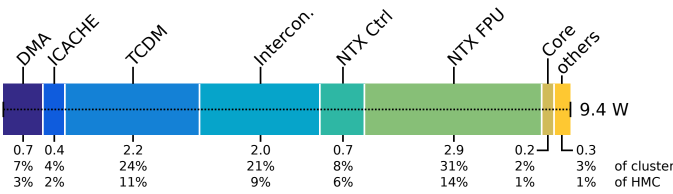

Figure 10. Power breakdown of the processing clusters in an NTX 64 in 14 nm performing the 3×3 convolution described in Figure 4. 47% of the HMC's power are consumed by the clusters. More precisely, 76% of the cluster power are dedicated to computation (NTX FPUs, TCDM, and TCDM interconnect) while 21% are consumed by control logic (DMA, ICACHE, NTX Controller, RISC-V Core). 3% are consumed by cluster peripherals. The DRAM, memory controllers, and interconnect in the HMC consume another $11.6\,\mathrm{W}$, 53% of the total cube power. Notably 14% of the total HMC power are spent in the FPUs.

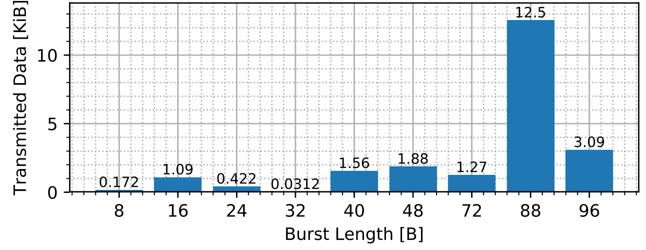

Figure 11. Histogram of data burst lengths issued by the DMA when calculating a 3×3 convolution tile.

and $3.5\,\mathrm{GHz}$ for the $14\,\mathrm{nm}$ process. The voltage is assumed to scale linearly with frequency [16] and is thus varied in proportion to the frequency. Figure 8 plots the energy efficiency of HMCs with different NTX configurations against the operating frequency. Two counteracting effects lead to a tradeoff between efficiency and frequency: On one hand DRAM consumes significant static power, making it beneficial to operate at a higher frequency to decrease the time to solution. On the other hand the NTX power consumption increases quadratically with voltage and thus frequency. For larger configurations, the internal bandwidth limit of the HMC is reached at a certain frequency, visible as a dent in the efficiency. The points of highest efficiency are listed in Table 5. Figure 9 shows a breakdown of the power consumption at these operating points. Figure 10 provides a more detailed power breakdown of the NTX 64 configuration in 14 nm. All configurations remain within a power budget of 25 W, which according to [17] is feasible for a HMC with active cooling and keeps DRAM temperature within nominal refresh limits. If the static power of the DRAM decreases, e.g. by switching to a different memory technology, these optimal operating points will change.

4.4 Multiple Logic Layers

Table 5 shows the area occupied by different NTX configurations. The unoccupied area on the LoB is not precisely known, and estimates range from $10\,\mathrm{mm}^2$ [4] to $50\,\mathrm{mm}^2$ [18]. In the following we assume that the LoB has an area of $50\,\mathrm{mm}^2$, of which $25\,\mathrm{mm}^2$ are unused and thus available to custom logic. This allows configurations of up to 64 clusters per HMC. For larger configurations, we propose the use of multiple stacked logic dies such as the 3D Logic in Memory (LiM) proposed in [19]. While the use of additional layers increases the complexity of the die stack, they allow for a significant increase in parallelism and efficiency. Furthermore, the use of LiM layers for custom accelerator logic has the additional benefit of decoupling the LoB manufacturing process from the accelerator, thus allowing modular assembly of "Application Specific Memory Cubes (ASMCs)". We expect this concept to be relevant for High Bandwidth Memory (HBM) as well.

4.5 Memory

Table 3 summarizes our estimates for the memory occupied by the parameters and the intermediate activations of the networks investigated in the paper. We derive these from the network structure outlined in the corresponding papers.

Table 3
Memory footprint of the parameters and intermediate activations of select DNNs. The two last columns show the total memory requirement for training with batch size 1 and higher. [MB]

Network	Param.	Interm. Act.	BS=1†	BS>1†
AlexNet [20]	232.5	6.0	238.5	471.0
GoogLeNet [1]	26.7	46.5	73.2	99.8
Inception v3 [21]	90.8	99.2	190.0	280.8
ResNet-34 [22]	176.2	28.3	204.5	380.6
ResNet-50 [22]	174.6	67.1	241.7	416.3
ResNet-152 [22]	306.4	154.4	460.7	767.1

† Batch Size

For training with a batch size of 1 the footprint amounts to 239 MB, 73.2 MB, and 461 MB for AlexNet, GoogLeNet, and ResNet-152, respectively. For batch sizes greater than 1, where the gradient of each image is computed separately and added to a weighted average, another set of of parameters is needed to hold the accumulated gradient. This amounts to 471 MB, 99.8 MB, and 767 MB, respectively. Note that the memory footprint then remains constant for all batch sizes.[1] This leaves 0.5 GB to 7 GB for training data depending on network and HMC size, around 3550 to 48600 sample images ($227 \times 277 \times 3$), equating to 31 s to 247 s of independent training operation on NTX 64.

To fully utilize the bandwidth into DRAM, it is paramount that the accesses emitted by the DMA occur in sufficiently long bursts and have high locality with respect to DRAM pages to reduce overhead. In the case of 4D tiling [4], this is given by the fact that the pre-tiled data lies in DRAM as a dense consecutive sequence. In the case of on-the-fly tiling the DMA has to issue more and smaller bursts since the required data does not lie in DRAM consecutively. The tile dimensions however offer multiple degrees of freedom to adjust the access patterns generated by the clusters. For example, HMCs [6] use an internal bus width of 32 B, and a maximum DRAM page size in the range of 32 B to 256 B. In the aforementioned 3×3 convolution most data transfers occur as bursts of 72 B, 88 B, or 96 B, and 92% of all data is transferred in bursts above 32 B. Figure 11 shows a histogram of the burst lengths issued by the DMA into the DRAM. The few small bursts are due to convolution weight transfers, which can be cached to improve burst length further. We thus conclude that our architecture is capable of fully utilizing DRAM bandwidth by emitting sufficiently large accesses.

4.6 Comparison with NeuroStream

NeuroStream (NS) [4] was aimed primarily at efficient inference and requires data to be very carefully laid out in memory (4D tiling). This constraint on data layout makes training very inefficient, since intermediate activations after each layer need to be re-tiled when storing them back to DRAM. This puts a significant workload on the RISC-V processor cores and causes additional traffic into memory. The processors are under high load to keep the NS saturated with FP operations, such that spending compute cycles on re-tiling also means stalling the NS co-processors. Our architecture does not depend on such a tiling.

1. Images in a batch may still be processed in sequence before a weight update is performed, keeping memory need constant.

Table 4
Architecture comparison in 28 nm FD-SOI between NTX (this work) and the inference architecture NS [4].

Figure of Merit	NS [4]	NTX “small”	NTX “big”
Clusters/Cores/Accelerators	16/4/2	16/1/8	64/1/8
Cluster/Accelerator Freq. [GHz]	1.0/1.0	0.75/1.5	0.75/1.5
Peak Performance [Gop/s]	256	384	1536
Core Efficiency [Gop/s W]	116	97	97
Area [mm^2]			
Clusters Logic	4.48	5.38	21.5
Clusters Memory	4.8	5.1	19.5
Total	9.3	10.5	41.0
Power [W]			
Clusters Logic	1.10	2.31	9.24
Clusters Memory	1.10	1.65	6.60
HMC without Clusters	9.00	9.14	12.9
Total	11.2	13.1	28.7
Inference (GoogLeNet, 1 image)			
Execution Time [ms]	14.0	11.3	2.83
Avg./Peak Bandwidth [GB/s]	14.4/51.2	17.8/57.6	71.0/230
Efficiency [Gop/s W]	**20.3**	**21.4**	**39.1**
Training (GoogLeNet, 1 image)			
Execution Time [ms]	56.8	34.8	8.69
Avg./Peak Bandwidth [GB/s]	11.3/51.2	18.5/57.6	74.0/231
Efficiency [Gop/s W]	**15.0**	**21.0**	**38.3**

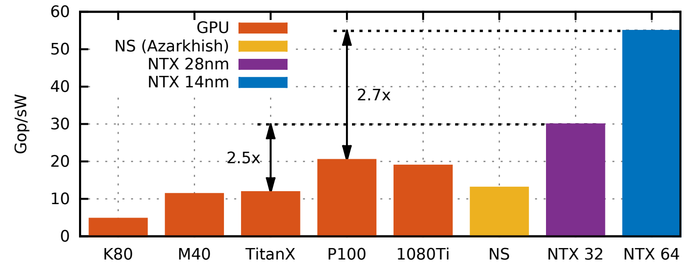

Figure 12. Comparison of energy efficiency when training the networks listed in Table 5 (geometric mean), with GPUs, NS [4], and the largest NTX configurations that do not require additional LiMs. NTX 32 in 28 nm achieves a $2.5\times$ increase, and NTX 64 in 14 nm a $2.7\times$ increase in efficiency over GPUs in similar technology nodes.

In Table 4 we compare NTX to NS, both implemented in 28 nm. The much improved offloading scheme allows us to increase the ratio of co-processors to control cores from 2:1 to 8:1. The fast FMAC allows us to operate the NTX at twice the frequency of the rest of the cluster, leading to an increase of peak performance from 256 Gop/s to 384 Gop/s for the 16 cluster version. The increased number of hardware loops and operations supported by NTX, together with the improved performance, allow us to increase the energy efficiency of a training step from 15 Gop/s W to 21 Gop/s W.

The 16 cluster configuration requests a peak bandwidth of 57.6 GB/s, which does not saturate the internal bandwidth of up to 320 GB available inside the HMC. We can improve the energy efficiency to 38.3 Gop/s W by increasing the number of clusters to 64.

4.7 Comparison with other Accelerators

To compare against other accelerators, we use one training step of AlexNet [20], GoogLeNet [1], Inception v3 [21], three variants of ResNet [22], and a Long Short-Term Memory (LSTM) with 512 inputs and hidden states as workload.

Table 5
Comparison between different configurations of the architecture proposed in this work, related custom accelerators, and GPUs. The energy efficiencies reported are with respect to training different DNNs and an LSTM.

Platform	**Characteristics**							**Energy Efficiency** [Gop/s W]							
	Logic [nm]	DRAM [nm]	Area [mm^2]	LiM	Freq. [GHz]	Peak Top/s	Arithmetic ††	AlexNet [20]	GoogLeNet [1]	Incep. v3 [21]	ResNet 34 [22]	ResNet 50 [22]	ResNet 152 [22]	**Geom. Mean**	LSTM§
This Work															
NTX (16×)	28	50	**10.5**	0	2.30	0.589	(a)	19.7	23.6	24.1	21.6	21.3	23.5	**22.3**	29.9
NTX (32×)	28	50	**20.7**	0	1.70	0.870	(a)	26.3	31.6	32.3	28.9	28.5	31.4	**29.9**	40.0
NTX (64×)	28	50	**41.0**	1	1.30	1.331	(a)	34.0	40.8	41.7	37.3	36.8	40.6	**38.6**	51.6
NTX (16×)	14	30	**4.2**	0	3.08	0.788	(a)	28.8	34.6	35.4	31.6	31.2	34.4	**32.8**	43.8
NTX (32×)	14	30	**8.3**	0	2.24	1.219	(a)	38.0	45.6	46.7	41.7	41.2	45.4	**43.2**	61.3
NTX (64×)	14	30	**16.4**	0	1.68	1.720	(a)	48.3	58.0	59.3	53.0	52.3	57.7	**54.9**	73.1
NTX (128×)	14	30	**32.8**	1	0.98	2.007	(a)	57.9	69.5	71.0	63.4	62.6	69.1	**65.8**	90.1
NTX (256×)	14	30	**65.6**	2	0.56	2.294	(a)	65.5	78.6	80.4	71.8	70.9	78.2	**74.4**	111.2
NTX (512×)	14	30	**131.2**	3	0.28	2.294	(a)	69.1	82.9	84.8	75.7	74.8	82.5	**78.5**	116.9
Custom Accelerators															
NS (16×) [4]	28	50	**9.3**	—	1.0	0.256	(a)	10.2	15.1	14.6	13.1	12.9	14.2	**13.0**	—
DaDianNao [13]	28	28	**67.7**	—	0.6	2.09	(b)	—	—	—	—	—	—	**65.8***	—
ScaleDeep [23]	14	—	—	—	0.6	680	(c)	87.7*	83.0*	—	139.2*	—	—	**100.8***	—
GPUs															
Tesla K80 †	28	40×	**561**	—	0.59	8.74	(a)	—	4.5	3.5	—	3.7	8.8	**4.7**	—
Tesla M40 †	28	30×	**601**	—	1.11	7.00	(a)	—	11.3	—	—	—	—	**11.3**	15.6
Titan X ‡	28	30×	**601**	—	1.08	7.00	(a)	12.8	9.9	—	17.6	8.5	12.2	**11.8**	—
Tesla P100 †	16	21°	**610**	—	1.3	10.6	(a)	—	19.8	19.5	—	18.6	24.18	**20.4**	—
GTX 1080 Ti ‡	16	20×	**471**	—	1.58	11.3	(a)	20.1	16.6	—	27.6	13.4	19.56	**18.9**	—

† Inception/ResNet: batch size 64 with TensorFlow/cuDNN 5.1 [11]; GoogLeNet: batch size 128 with Torch/cuDNN 5.1 [12]
‡ All nets: batch size 16 Torch/cuDNN 5.1 [24] § 512 inputs and hidden states, batch size 32 for NTX and 64 for GPU [25]
× GDDR5 and GDDRX5, process node estimated based on GPU release year ° HBM2 * Estimated system efficiency including DRAM, see Section 5.2 †† (a) floating-point 32 bit, (b) fixed-point 16/32 bit, (c) floating-point 16/32 bit

Table 5 and Figure 12 provides an overview of the compared architectures.

To our knowledge there are two other custom accelerators besides NeuroStream [4] that claim support for training at precisions similar to ours: DaDianNao [13] and ScaleDeep [23]. Both provide much less memory relative to their computational power than GPUs, NeuroStream, and NTX. To compare on a system level, we estimate the efficiency of these accelerator including additional DRAM to hold training data, as described in Section 5.2. In this case, DaDianNao has an efficiency of 65.8 Gop/s W with fixed-point arithmetic, which is identical to the computationally equivalent NTX 128. ScaleDeep has an efficiency of 100.8 Gflop/s W which is 1.3× higher than NTX 512, the largest configuration considered by us.

GPUs are currently the accelerator of choice to train DNNs. Our architecture can achieve significantly higher energy efficiency than a GPU at a comparable technology node (see Figure 12). Considering the largest NTX configurations that do not require additional LiMs, we achieve an efficiency increase of 2.5× from 11.8 Gop/s W to 29.9 Gop/s W in 28 nm, and an increase of 2.7× from 20.4 Gop/s W to 54.9 Gop/s W in 14 nm. Compared to the GPU power analysis and model published in [26], NTX spends a larger fraction of power in the FPUs, namely 14% versus 4.8%. Assuming an FMA requires the same energy per item in similar technology nodes, this increase corresponds to the observed efficiency increase and gives an intuition of why NTX outperforms GPUs. This is in part due to the absence of caches in NTX and the GPU's significant idle power. See Figure 10.

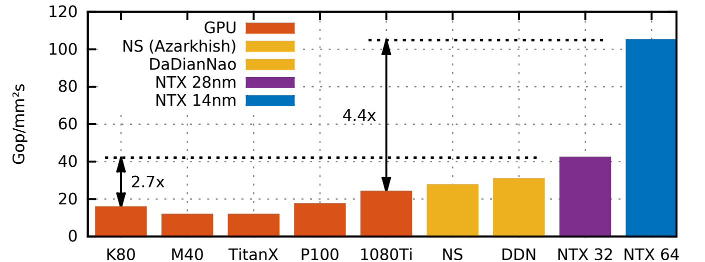

Figure 13. Comparison of the Gop/s of compute performance per deployed area of silicon, for GPUs, NS [4], and the largest NTX configurations that do not require additional LiMs. NTX 32 in 28 nm achieves a 2.7× increase, and NTX 64 in 14 nm a 4.4× increase in area efficiency over GPUs in similar technology nodes.

4.8 Deployed Silicon

One unique key benefit of our architecture is that it leverages existing unused silicon area. This incurs almost no additional costs, since we assume the HMCs to be already present in the system as main memory of the CPU, and manufacturing costs of the spare silicon area is the same regardless of whether it is being used. This allows us to deploy up to 32 processing clusters in 28 nm and 64 processing clusters in 14 nm with no additional silicon needed.

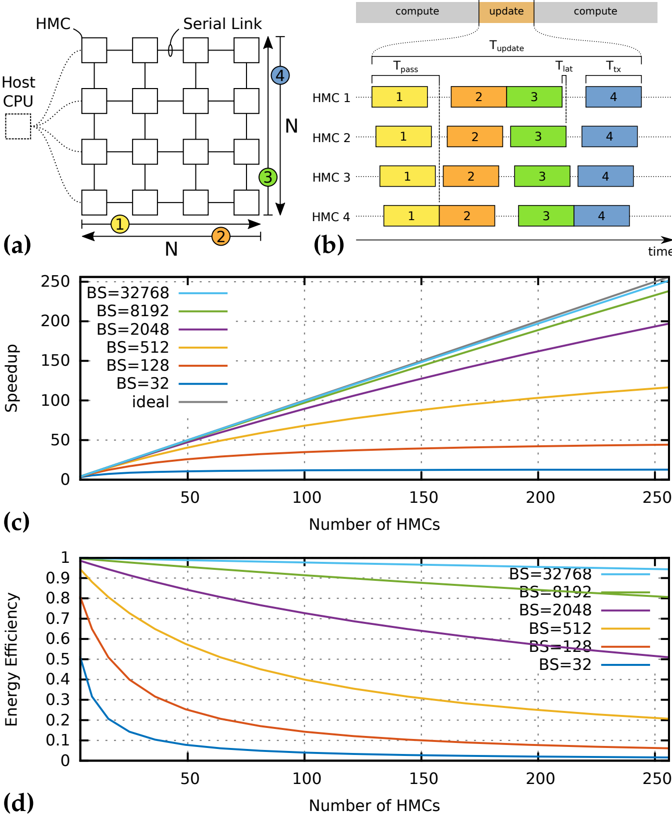

Figure 14. Scaling behavior of data-parallel training on a mesh of HMCs. (a) depicts the square mesh with N^2 cubes. The arrows indicate the four phases and wave direction of the global weight update. (b) shows the time line corresponding to the mesh for the compute and, in more detail, update phases. (c) outlines the speedup for different mesh and total batch sizes. (d) shows the corresponding energy efficiencies. Larger batch sizes help amortize the cost of the global weight update.

Figure 13 compares the $\mathrm{Gop/s}$ of compute performance per deployed amount of silicon for NTX and GPUs. Our solution requires $4.4\times$ less area to achieve the same compute performance as a GPU. Even more when one considers that the chosen 32 and 64 cluster configurations can fit into the aforementioned unused silicon, their cost is virtually zero. This sets our solution apart from ScaleDeep, DaDianNao, and other GPUs, which require significant silicon overhead.

4.9 Scaling to multiple HMCs

We investigate the scaling behavior of NTX 64 and organize the HMCs in a square mesh of different side lengths N as depicted in Figure 14a. Each link operates at $60\,\mathrm{GB/s}$ [6]. We leverage data parallel training to distribute computation across the HMCs in the mesh, which is also commonly done on GPUs [27]. Each HMC computes its local weight update first. The global update is then performed in four waves as a horizontal followed by a vertical systolic average which can be performed in a streaming fashion.

We assume the weight update to be $300\,\mathrm{MB}$, which takes $T_{\text{tx}} = 4.88\,\mathrm{ms}$ to transmit. Each cube takes $104\,\mu\mathrm{s}$ to compute the average, which is negligible compared to T_{tx}. Furthermore the internal bandwidth of the HMC is much larger than the $120\,\mathrm{GB}$ required by the two serial links active in parallel during streaming operation. We assume a latency of $T_{\text{lat}} = 20\,\mu\mathrm{s}$ inside the cube, which is a very conservative estimate. The time taken for one of the four waves described above is then

$$T_{\text{pass}} = T_{\text{tx}} + N \cdot T_{\text{lat}} \tag{14}$$

Since T_{lat} is small relative to T_{tx}, the number of cubes in the mesh has only little influence on this time. For a very large mesh of $N = 16$ (256 HMCs) $T_{\text{pass}} = 5.20\,\mathrm{ms}$. Since four such passes are necessary, the total time to perform the weight update across the mesh is

$$T_{\text{update}} = 4 \cdot T_{\text{pass}} = 20.8\,\mathrm{ms} \tag{15}$$

A time diagram of such an update is depicted in Figure 14b. The time required by the mesh to calculate the local weight update is

$$T_{\text{step}} = 8.69\,\mathrm{ms} \cdot L_B \,/\, N^2 \tag{16}$$

where L_B is the total batch size across all cubes. This yields a total execution time for one batch across the mesh of $T_{\text{total}} = T_{\text{update}} + T_{\text{step}}$. A single HMC would perform the same computation in $T_{\text{single}} = 8.69\,\mathrm{ms} \cdot L_B$. Figure 14c depicts the speedup. At a batch size of 8192, a system with 64 HMCs achieves almost perfect speedup of $62.8\times$ (98% parallel efficiency), and 144 HMCs achieve $138\times$ (95.8% parallel efficiency).

Regarding energy efficiency we consider two operating modes of the cubes: During the global mesh update the serial links and clusters are active. We assume the four serial links to consume $P_{\text{link}} = 8\,\mathrm{W}$ [4]. The energies to compute a wave pass and to power-cycle the serial links [6] are

$$E_{\text{pass}} = T_{\text{pass}} \cdot (21\,\mathrm{W} + P_{\text{link}}) = 150.9\,\mathrm{mJ} \tag{17}$$
$$E_{\text{pwrud}} = 2 \cdot P_{\text{link}} \cdot 50\,\mathrm{ms} = 800\,\mathrm{mJ} \tag{18}$$

The energies spent per HMC for the global and local weight updates are

$$E_{\text{update}} = 4 \cdot E_{\text{pass}} + E_{\text{pwrud}} = 1.403\,\mathrm{J} \tag{19}$$
$$E_{\text{step}} = T_{\text{step}} \cdot 21\,\mathrm{W} \cdot N^2 \tag{20}$$

and the overall energy for one batch across the mesh requires

$$E_{\text{total}} = (E_{\text{update}} + E_{\text{step}}) \cdot N^2 \tag{21}$$

A single HMC would require $E_{\text{single}} = T_{\text{single}} \cdot 21\,\mathrm{W}$ for the same task. At a batch size of 8192 as above, 64 HMCs achieve an energy efficiency of 94.3%, 144 HMCs achieve 88.1%.

4.10 Savings at Data Center Scale

Computing at a data center scale incurs a significant energy and cost overhead over the raw hardware's power consumption. This is among other factors due to the required air conditioning and cooling. A standard measure for this overhead is the Power Usage Effectiveness (PUE) [28], the ratio of the power consumed by a data center to the power consumed solely by its compute units:

$$\eta_{\text{pue}} = \frac{P_{\text{total}}}{P_{\text{compute}}}$$

Data centers are reported to have $\eta_{\text{pue}} = 1.12$ [29]. The figure depends heavily on the local climate and usually only

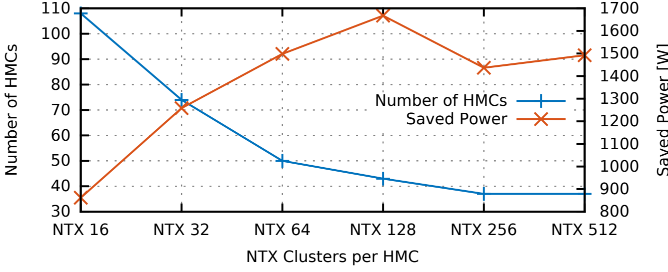

Figure 15. Number of HMCs required to meet a compute of $84.8\,\mathrm{Tflop/s}$ for different numbers of NTX clusters per HMC, and the corresponding power savings over 8 GPUs achieving the same compute capability.

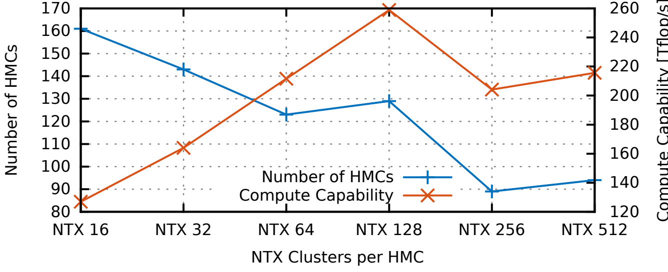

Figure 16. Number of HMCs that can be deployed with a power budget of $2.4\,\mathrm{kW}$ for different numbers of NTX clusters per HMC, and the corresponding compute capability.

the winter months' numbers are published. We assume an average $\eta_{\text{pue}} = 1.2$. We consider a NVIDIA DGX-1 server with two Intel Xeon CPUs and eight Tesla P100 cards. One such unit consumes $3.2\,\mathrm{kW}$ of power, $2.4\,\mathrm{kW}$ of which are due to the GPUs. We assume DDR4 DRAM to consume $6\,\mathrm{W}$ per $16\,\mathrm{GB}$ of storage under full load [30]. We investigate two different approaches of replacing the GPUs of the system with NTX-augmented HMCs. Consider that the $512\,\mathrm{GB}$ of system memory requires 256 chips distributed across the DIMM modules if built from $16\,\mathrm{Gbit}$ DRAM chips. An 8 GB HMC is roughly equivalent to 4 such chips, so the same system built from HMCs would comprise 64 memory cubes.

4.10.1 *Same Peak Compute*

The P100 cards achieve a combined peak compute of $84.8\,\mathrm{Tflop/s}$. Figure 15 shows the number of HMCs required to match this performance, and the achievable energy savings, with different NTX configurations per cube. The 43 HMCs with NTX 128 required to achieve the same compute power consume only $860\,\mathrm{W}$, saving $2.4\,\mathrm{kW}$ of GPU power and an additional $128\,\mathrm{W}$ of DRAM power, for an overall reduction of $2.1\times$. With a PUE of 1.2 this translates to $1868\,\mathrm{kW}$ of saved power, which at an energy price of $0.1104\,\$/\mathrm{kWh}$ [31] is $1808\,\$$ per year and server.

4.10.2 *Same Thermal Design Power*

Figure 16 shows the number of HMCs that can be deployed within the $2.4\,\mathrm{kW}$ GPU power budget of the DGX-1. 129 HMCs with NTX 128 are capable of achieving $258.9\,\mathrm{Tflop/s}$ in total, a $3.1\times$ improvement over the P100.

5 Related Work

Acceleration of DNNs, in particular the forward pass, is a well researched field with a rich literature. Goodfellow, et al. [3] provide a good coverage of the mathematical background of Deep Learning. An overview of techniques for efficient DNN inference and the involved challenges can be found in [2].

5.1 Accelerators for Inference

Architectures to accelerate the inference process of Convolutional Neural Network (CNN) have been studied extensively in literature. FPGA-based accelerators report energy efficiencies on the order of $10\,\mathrm{Gop/s\,W}$ and usually rely on fixed-point arithmetic and less than 32 bit precision [32]. ASIC-based accelerators porivde efficiencies on the order of $1000\,\mathrm{Gop/s\,W}$ at reduced precisions, for example Google's TPU [33] which uses 8 bit arithmetic, or [34] with 1 bit. Near-memory inference architectures embedded in the logic die of an HMC have also been investigated, for example the 2D accelerator array presented in [18] which uses 16 bit fixed-point arithmetic and achieves up to $450\,\mathrm{Gop/s\,W}$, or the clustered many-core architecture [4] which is based on 32 bit FP co-processors and achieves up to $22.5\,\mathrm{Gflop/s\,W}$. Certain architectures such as [35] employ a distributed memory model, where the entire network's parameter are stored on chip. This becomes increasingly difficult with modern networks that require hundreds of MB [1], [20], [21], [22], and the network to be trained is tightly bound to the number of chips that can be interconnected. We furthermore observe that due to the vast difference in energy spent for computation and data transfer, it is only meaningful to compare architectures that use the same arithmetic precision and bit width.

5.2 Accelerators for Training

We observe that much fewer architectures have been proposed to cover the training aspect of DNNs. Many of the aforementioned architectures are not suitable for this since they lack the ability or memory capacity to store intermediate activations, e.g. due to optimizations in the data path, or the precision and dynamic range for the training to converge.

The NeuroCube [36] is based on 16 bit fixed-point MAC units which are capable of performing the necessary computations, but it is unclear if training of modern deep networks converges at this precision and dynamic range. NTX surpasses NeuroCube's efficiency of $7.63\,\mathrm{Gop/s\,W}$ because we focus on maximizing the energy spent in the FPU, e.g. by doubling its clock frequency. We furthermore use VFS to increase efficiency.

DaDianNao [13] uses 64 chips to perform training at 32 bit fixed-point precision with $2.3\,\mathrm{GB}$ of distributed memory. A single chip with $2.1\,\mathrm{Top/s}$ and $36\,\mathrm{MB}$ is roughly equivalent to NTX 128 with $2\,\mathrm{Tflop/s}$ and $16\,\mathrm{MB}$ but without the HMC. In this setting NTX achieves a $1.9\times$ better core efficiency of $250.9\,\mathrm{Gflop/s\,W}$ despite its FP arithmetic. Considering an estimated $15.8\,\mathrm{GB}$ of DRAM that is needed to match the memory-to-compute density of the DGX-1 puts DaDianNao's system energy efficiency at $65.8\,\mathrm{Gflop/s\,W}$,

assuming that the overall cost of the memory (DRAM modules, memory controller, interconnects) is comparable to that of an 8 GB HMC (1 W/GB).

ScaleDeep [23] supports training in 32 bit FP precision at 14 nm. We estimate its total die area to be $2800\,\mathrm{mm}^2$ based on its TDP and the power density of a GPU, which yields $243\,\mathrm{Gflop\,s/mm}^2$, around $2.3\times$ more than NTX 64. An entire node achieves 680 Tflop/s but has only 1.17 GB of distributed memory. As such it excludes any form of system DRAM and puts its core efficiency of 420.9 Gflop/s W on par with the 417.0 Gflop/s W achieved by NTX 512. The estimated 5.13 TB of DRAM that is needed to match the memory-to-compute density of the DGX-1 puts ScaleDeep's system power consumption at 6.75 kW under the same assumptions as DaDianNao, with a system energy efficiency of 100.8 Gflop/s W. It is unclear how much additional energy would be consumed by the processing power required to feed these accelerators with data.

5.3 GPUs

GPUs can be seen as the main workhorse of Deep Learning and are commonly used for both inference and training due to their flexibility. Recent implementations on the GTX 780 and GTX Titan (both featuring a Kepler microarchitecture) reach 1650 Gflop/s at 250 W and 999 Gflop/s at 240 W, which corresponds to 6.6 and 4.2 Gflop/s W, respectively [4], [37]. Embedded GPUs like the Tegra K1 have lower absolute throughput, but reach a similar energy efficiency of around 7 Gflop/s W [37]. The Pascal generation of GPUs offer several features beneficial to DNNs, such as HBM and 16 bit FP support. Compared to previous generations, the P100 achieves a $2\times$ higher peak throughput of 10.6 Tflop/s and a significantly higher energy efficiency around 20 Gflop/s W [11], [24]. The recently introduced Volta generation offers *tensor cores*, a new compute element able to perform 4×4 matrix Fused Multiply and Adds (FMAs) in 16 bit FP, with 16 or 32 bit outputs in one cycle. These cores promise a $5\times$ increase in Deep Learning performance compared to previous GPU generations [38]. Furthermore, GPUs have been shown to be amenable to near-memory processing as well [39].

6 Future Work

Our architecture is inherently scalable since the HMC standard allows for memory cubes to be interconnected via the serial links [6]. Mesh arrangements of HMCs offer many opportunities and different parallelization techniques [40] for training DNN should be explored. Moving to HBM promises further energy efficiency gains and brings new challenges and design constraints in using the bottom memory controller die. Improvements to the DMA engine in the compute clusters would allow for even more efficient offloading and further ease the load on the RISC-V processor core. Applicability of transprecision and compression techniques offer other interesting angles to be investigated for further gains.

7 Conclusion

We have presented the streaming FP co-processor NTX with a decisive focus on training DNNs. Its data path is built around a fast fused accumulator with full 32 bit precision, which gives it a key advantage over architectures that are based on fixed-point arithmetic or lower FP precision. The co-processor is capable of generating three independent address streams from five nested hardware loops, allowing it to traverse structures with up to five dimensions in memory independently. A rich set of arithmetic and logic commands allows it to perform the reductions and matrix/vector operations commonly found in the forward pass, but also the threshold, mask, and scatter operations encountered during the backward pass. We combine eight such co-processors with memory, a control processor, and a DMA unit into a cluster. An efficient offloading scheme frees up resources on the control processor to exert fine-grained control over data movement. The data does therefore not need to be put into memory in a specific, pre-tiled pattern, but can be operated on directly in its canonical and dense form. Integrated into the LoB of an HMC, multiple clusters can exploit the high bandwidth and low accesses latency into DRAM in this near-memory setting. Configurations which fit into the unused area on the LoB incur virtually zero additional manufacturing costs. NTX scales well to large meshes of HMCs and can provide the same compute capability at less power, or more compute capability at the same power.

Acknowledgments

The authors would like to thank Lukas Cavigelli, Derek Chou, and Erfan Azarkhish for the inspiring discussions and insights.

This work has been supported by Microsoft Research under the project "Enabling Practical, Efficient and Large-Scale Computation Near Data to Improve the Performance and Efficiency of Data Center and Consumer Systems" with MRL contract number 2017-044.

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732631, project "OPRECOMP".

References

[1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.

[2] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *arXiv:1703.09039*, 2017.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[4] E. Azarkhish, D. Rossi, I. Loi, and L. Benini, "Neurostream: Scalable and Energy Efficient Deep Learning with Smart Memory Cubes," *IEEE TPDS*, vol. PP, no. 99, 2017.

[5] ——, "Design and evaluation of a processing-in-memory architecture for the smart memory cube," in *ARCS*. Springer, 2016, pp. 19–31.

[6] "Hybrid Memory Cube Specification 2.1," http://www.hybridmemorycube.org, 2015, acc.: Sept 2017.

[7] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices," *TVLSI*, 2017.

[8] R. Tewell, "FD-SOI – Harnessing the Power & A Little Spelunking into PPA," Presented at DAC'53, USA, 2016.

[9] S. Davis, "IEDM 2013 Preview," http://electroiq.com/chipworks_real_chips_blog/2013/12/06/iedm_2013_preview/, 2013, acc.: Sept 2017.

[10] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *Hot Chips*, 2011.
[11] "Tensorflow Benchmarks," https://www.tensorflow.org/performance/benchmarks, August 2017, acc.: September 2017.
[12] J. Murphy, "Deep Learning Benchmarks of NVIDIA Tesla P100 PCIe, Tesla K80, and Tesla M40 GPUs," https://www.microway.com/hpc-tech-tips/deep-learning-benchmarks-nvidia-tesla-p100-16gb-pcie-tesla-k80-tesla-m40-gpus/, Jan 2017, acc.: Jul 2017.
[13] T. Luo, S. Liu, L. Li, Y. Wang, S. Zhang, T. Chen, Z. Xu, O. Temam, and Y. Chen, "DaDianNao: a neural network supercomputer," *TOC*, vol. 66, no. 1, pp. 73–88, 2017.
[14] U. Köster, T. Webb, X. Wang, M. Nassar, A. K. Bansal, W. Constable, O. Elibol, S. Hall, L. Hornof, A. Khosrowshahi *et al.*, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 1740–1750.
[15] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *NIPS*, 2016, pp. 2074–2082.
[16] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. De Micheli, "Dynamic voltage scaling and power management for portable systems," in *Proceedings of the 38th annual Design Automation Conference*. ACM, 2001, pp. 524–529.
[17] Y. Eckert, N. Jayasena, and G. H. Loh, "Thermal feasibility of die-stacked processing in memory," in *WoNDP*, 2014.
[18] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory," in *ASPLOS*, 2017.
[19] A. Rush, "Memory Technology and Applications," Presentation at HotChips, https://www.hotchips.org/wp-content/uploads/hc_archives/hc28/HC28.21-Tutorial-Epub/HC28.21.1-Next-Gen-Memory-Epub/HC28.21.150-Mem-Tech-Allen.Rush-AMD.v3-t1-6.pdf, August 2016, acc.: July 2017.
[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097–1105.
[21] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *CVPR*, 2016, pp. 2818–2826.
[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.
[23] S. Venkataramani, A. Ranjan, S. Banerjee, D. Das, S. Avancha, A. Jagannathan, A. Durg, D. Nagaraj, B. Kaul, P. Dubey *et al.*, "ScaleDeep: A Scalable Compute Architecture for Learning and Evaluating Deep Networks," in *ISCA*, 2017, pp. 13–26.
[24] J. Johnson, "cnn-benchmarks," https://github.com/jcjohnson/cnn-benchmarks, acc.: September 2017, Commit hash 83d441f.
[25] J. Appleyard, T. Kociský, and P. Blunsom, "Optimizing performance of recurrent neural networks on gpus," *arXiv:1604.01946*, 2016.
[26] S. Hong and H. Kim, "An integrated gpu power and performance model," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 280–289.
[27] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun, "Deep image: Scaling up image recognition," *arXiv preprint arXiv:1501.02876*, 2015.
[28] "Information technology – Data centres – Key performance indicators – Part 2: Power usage effectiveness (PUE)," International Organization for Standardization, Geneva, CH, Standard, April 2016.
[29] J. Gao and R. Jamidar, "Machine learning applications for data center optimization," *Google White Paper*, 2014.
[30] C. Angelini, "Measuring ddr4 power consumption," http://www.tomshardware.com/reviews/intel-core-i7-5960x-haswell-e-cpu,3918-13.html, August 2014, accessed Oct 2017.
[31] U.S. Energy Information Administration, "Average price of electricity to ultimate customers by end-use sector," https://www.eia.gov/electricity/monthly/epm_table_grapher.php?t=epmt_5_6_a, August 2017, accessed Oct 2017.
[32] V. Gokhale, A. Zaidy, A. X. M. Chang, and E. Culurciello, "Snowflake: an Efficient Hardware Accelerator for Convolutional Neural Networks," in *ISCAS*, 2017.
[33] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *ISCA*, 2017, pp. 1–12.
[34] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An Architecture for Ultra-Low Power Binary-Weight CNN Acceleration," *TCAD*, 2017.
[35] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *ACM Sigplan Notices*, vol. 49, no. 4, 2014, pp. 269–284.
[36] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory," in *ISCA*, 2016, pp. 380–392.
[37] L. Cavigelli, M. Magno, and L. Benini, "Accelerating real-time embedded scene labeling with convolutional networks," in *DAC*, 2015, pp. 1–6.
[38] NVidia, "Artificial Intelligence Architecture | NVIDIA Volta," https://www.nvidia.com/en-us/data-center/volta-gpu-architecture/, 2017, acc.: July 2017.
[39] A. Pattnaik, X. Tang, A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir, O. Mutlu, and C. R. Das, "Scheduling techniques for gpu architectures with processing-in-memory capabilities," in *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*. ACM, 2016, pp. 31–44.
[40] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, "Communication-efficient learning of deep networks from decentralized data," *arXiv:1602.05629*, 2016.

Fabian Schuiki received the B.Sc. and M.Sc. degree in electrical engineering from ETH Zürich, in 2014 and 2016, respectively. He is currently pursuing a Ph.D. degree with the Digital Circuits and Systems group of Luca Benini. His research interests include transprecision computing as well as near- and in-memory processing.

Michael Schaffner received his MSc and PhD degrees from ETH Zurich, Switzerland, in 2012 and 2017. He has been a research assistant at the Integrated Systems Laboratory and Disney Research from 2012 to 2017, and he is currently working as a postdoctoral researcher at the Integrated Systems Laboratory. His research interests include digital signal and video processing, and the design of VLSI circuits and systems. Michael Schaffner received the ETH Medal for his Diploma thesis in 2013.

Frank K. Gürkaynak received the B.Sc. and M.Sc. degrees in electrical engineering from Istanbul Technical University, and the Ph.D. degree in electrical engineering from ETH Zürich, in 2006. He is currently a Senior Researcher with the Integrated Systems Laboratory at ETH Zürich, where his research interests include digital low-power design and cryptographic hardware.

Luca Benini received the Ph.D. degree in electrical engineering from Stanford University, CA, USA, in 1997. He has held visiting and consulting researcher positions at EPFL, IMEC, Hewlett-Packard Laboratories, and Stanford University. His research interests are in energy efficient system design and multi-core SoC design. He is a member of the Academia Europaea, a Full Professor at the University of Bologna, and Chair of the Digital Circuits and Systems group at ETH Zürich.