DaDianNao: A Neural Network Supercomputer

Tao Luo, Shaoli Liu, Ling Li, Yuqing Wang, Shijin Zhang,
Tianshi Chen, Zhiwei Xu, Olivier Temam, and Yunji Chen

Abstract—Many companies are deploying services largely based on machine-learning algorithms for sophisticated processing of large amounts of data, either for consumers or industry. The state-of-the-art and most popular such machine-learning algorithms are Convolutional and Deep Neural Networks (CNNs and DNNs), which are known to be computationally and memory intensive. A number of neural network accelerators have been recently proposed which can offer high computational capacity/area ratio, but which remain hampered by memory accesses. However, unlike the memory wall faced by processors on general-purpose workloads, the CNNs and DNNs memory footprint, while large, is not beyond the capability of the on-chip storage of a multi-chip system. This property, combined with the CNN/DNN algorithmic characteristics, can lead to high internal bandwidth and low external communications, which can in turn enable high-degree parallelism at a reasonable area cost. In this article, we introduce a custom multi-chip machine-learning architecture along those lines, and evaluate performance by integrating electrical and optical inter-chip interconnects separately. We show that, on a subset of the largest known neural network layers, it is possible to achieve a speedup of $656.63\times$ over a GPU, and reduce the energy by $184.05\times$ on average for a 64-chip system. We implement the node down to the place and route at 28 nm, containing a combination of custom storage and computational units, with electrical inter-chip interconnects.

Index Terms—Machine learning, neuron network, supercomputer, multi-chip, interconnect, CNN, DNN

1 Introduction

Machine-Learning algorithms have become ubiquitous in a very broad range of applications and cloud services; examples include speech recognition, e.g., Siri or Google Now, click-through prediction for placing ads [27], face identification in Apple iPhoto or Google Picasa, robotics [20], pharmaceutical research [9] and so on. It is probably not exaggerated to say that machine-learning applications are in the process of displacing scientific computing as the major driver for high-performance computing. Early symptoms of this transformation are Intel calling for a refocus on Recognition, Mining and Synthesis applications in 2005 [14] (which later led to the PARSEC benchmark suite [3]), with Recognition and Mining largely corresponding to machine-learning tasks, or IBM developing the Watson supercomputer, illustrated with the Jeopardy game in 2011 [19].

Remarkably enough, at the same time this profound shift in applications is occurring, two simultaneous, albeit apparently unrelated, transformations are occurring in the machine-learning and in the hardware domains. Our community is well aware of the trend towards heterogeneous computing where architecture specialization is seen as a promising path to achieve high performance at low energy [21], provided we can find ways to reconcile architecture specialization and flexibility. At the same time, the machine-learning domain has profoundly evolved since 2006, where a category of algorithms, called Deep Learning (Convolutional and Deep Neural Networks), has emerged as state-of-the-art across a broad range of applications [28], [32], [33], [34]. In other words, at the time where architects need to find a good tradeoff between flexibility and efficiency, it turns out that just one category of algorithms can be used to implement a broad range of applications. In other words, there is a fairly unique opportunity to design highly specialized and highly efficient hardware which will benefit many of these emerging high-performance applications.

A few research groups have started to take advantage of this special context to design accelerators meant to be integrated into heterogeneous multi-cores. Temam [47] proposed a neural network accelerator for multi-layer perceptrons, though it is not a deep learning neural network, Esmaeilzadeh et al. [16] propose to use a hardware neural network called NPU for approximating any program function, though not specifically for machine-learning applications, Chen et al. [5] proposed an accelerator for Deep Learning (CNNs and DNNs). However, all these accelerators have significant neural network *size* limitations: either small neural networks of a few tens of neurons can be executed, or the neurons and synapses (i.e., weights of connections between neurons) intermediate values have to be stored in main memory. These two limitations from a machine-learning or a hardware perspective respectively, are severe.

From a machine-learning perspective, there is a significant trend towards increasingly large neural networks. The recent work of Krizhevsky et al. [32] achieved state-of-the-art accuracy on the ImageNet database [13] with "only" 60 million parameters. There are recent examples of a 1-billion parameter neural network [34], and some of the same authors even investigated a 10-billion neural network the following year [8]. However, these networks are for now

- *T. Luo, S. Liu, Y. Wang, S. Zhang, T. Chen, Z. Xu, and Y. Chen are with State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China. E-mail: {luotao, liushaoli, zhangshijin, chentianshi, zxu, cyj}@ict.ac.cn.*
- *L. Li is with Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China. E-mail: liling@ict.ac.cn.*
- *O. Temam is with Inria Scalay, Palaiseau 91120, France. E-mail: olivier.temam@inria.fr.*

Manuscript received 18 Nov. 2015; revised 3 May 2016; accepted 19 May 2016. Date of publication 29 May 2016; date of current version 19 Dec. 2016. Recommended for acceptance by Y. Xie.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2016.2574353

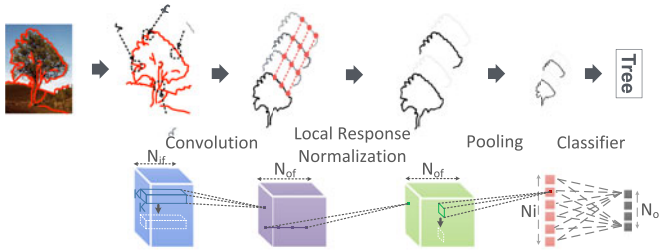

Fig. 1. The four layer types found in CNNs and DNNs.

considered extreme experiments in unsupervised learning (the first one on 16,000 CPUs, the second one on 64 GPUs), and they are outperformed by smaller but more classic neural networks such as the one by Krizhevsky et al. [32]. Still, while the neural network size progression is unlikely to be monotonic, there is a definite trend towards larger neural networks. Moreover, increasingly large inputs (e.g., HD instead of SD images) will further inflate the neural networks sizes. From a hardware perspective, the aforementioned accelerators are limited because if most synaptic weights have to reside in main memory, and if neurons intermediate values have to be frequently written back and read from memory, the memory accesses become the performance bottleneck, just like in processors, partly voiding the benefit of using custom architectures. Chen et al. [5] acknowledge this issue by observing that their neural network accelerator loses at least an order of magnitude in performance due to memory accesses.

However, while 1 billion parameters or more may come across as a large number from a machine-learning perspective, it is important to realize that, in fact, it is not from a hardware perspective: if each parameter requires 64 bits, that only corresponds to 8 GB (and there are clear indications that fewer bits are sufficient). While 8 GB is still too large for a single chip, it is possible to imagine a dedicated machine-learning computer composed of multiple chips, each chip containing specialized logic together with enough RAM that the sum of the RAM of all chips can contain the *whole* neural network, requiring *no main memory*. By tightly interconnecting these different chips through a dedicated mesh, one could implement the largest existing DNNs, achieve high performance at a fraction of the energy and area of the many CPUs or GPUs used so far. Due to its low energy and area costs, such a machine, a kind of compact machine-learning supercomputer, could help spread the use of high-accuracy machine-learning applications, or conversely to use even larger DNNs/CNNs by simply scaling up RAM storage at each node and/or the number of nodes.

In this article, we present such an architecture, composed of electrically interconnected nodes, each containing computational logic, eDRAM, and the router fabric; the node is implemented down to the place and route at 28 nm, and we evaluate an architecture with up to 64 nodes. On a sample of the largest existing neural network layers, we show that it is possible to achieve a speedup of $656.63\times$ over a GPU and to reduce energy by $184.05\times$ on average.

We also evaluate optical interconnects based on silicon photonics (SiPh) [54], and observe that the 64-node system with optical interconnects achieves a speedup of $2.20\times$ on classifier layers, an average speedup of $1.13\times$ and an average energy reduction of $1.16\times$ on the largest representative neural network layers, over the 64-node system with electrical interconnects.

In Section 2, we introduce CNNs, DNNs and silicon pohotonics techniques, in Section 3, we evaluate such NNs on GPU, in Section 4 we compare GPU and a recently proposed accelerator for CNNs and DNNs, in Section 5, we introduce the machine-learning supercomputer, we present the methodology in Section 6, the experimental results in Section 7 and the related work in Section 8.

2 Background

In this section, we firstly introduce CNNs and DNNs in current machine-learning techniques, and then introduce the silicon photonics techniques.

2.1 Machine-Learning Techniques

The state-of-the-art and most popular machine-learning algorithms are Convolutional Neural Networks (CNNs) [35] and Deep Neural Networks (DNNs) [9]. Beyond early differences in training, the two types of networks are also distinguished by their implementation of convolutional layers detailed thereafter. CNNs are particularly efficient for image applications and any application which can benefit from the implicit translation invariance properties of their convolutional layers. DNNs are more complex neural networks but they have an even broader application span such as speech recognition [9], web search [27], etc.

2.1.1 Main Layer Types

A CNN or a DNN is a sequence of multiple instances of four types of layers: pooling layers (POOL), convolutional layers (CONV), classifier layers (CLASS), and local response normalization layers (LRN), see Fig. 1. Usually, groups of convolutional, local response normalization and pooling layers alternate, while classifier layers are found at the end of the sequence, i.e., at the top of the neural network hierarchy. We present a simple hierarchy in Fig. 1; we illustrate the intuitive task performed at the top, and we provide the formal computations performed by the layer at the bottom.

Convolutional Layers. Intuitively, a convolutional layer implements a set of filters to identify characteristic elements of the input data, e.g., an image, see Fig. 1. For visual data, a filter is defined by $K_x \times K_y$ coefficients forming a *kernel*; these kernel coefficients are learned and form the layer synaptic weights. Each convolutional layer slides N_{of} such filters through the whole input layer (by steps of s_x and s_y), resulting in as many (N_{of}) output feature maps.

The concrete formula for calculating an output neuron $out(x,y)^{f_o}$ at position (x,y) of output feature map f_o is

$$out(x,y)^{f_o} = \sum_{f_i=0}^{N_{if}} \sum_{k_x=0}^{K_x} \sum_{k_y=0}^{K_y} w_{f_i,f_o}(k_x,k_y) * in(x+k_x, y+k_y)^{f_i},$$

where $in(x,y)^f$ (resp. $out()$) represents the input (resp. output) neuron activity at position (x,y) in feature map f, and $w_{f_i,f_o}(k_x,k_y)$ is the synaptic weight at kernel position (k_x,k_y) in input feature map f_i for filter (output feature map) f_o. Since the input layer itself may contain multiple feature

TABLE 1
Some of the Largest Known CNN or DNN Layers (CONVx* Indicates Convolutional Layers with Private Kernels)

Layer	N_x	N_y	K_x	K_y	N_i or N_{if}	N_o or N_{of}	Synapses	Description
CLASS1	-	-	-	-	2,560	2,560	12.5 MB	Object recognition and speech recognition tasks (DNN) [11].
CLASS2	-	-	-	-	4,096	4,096	32 MB	Multi-Object recognition in natural images (DNN), winner 2012 ImageNet competition [32].
CONV1	256	256	11	11	256	384	22.69 MB	
POOL2	256	256	2	2	256	256	-	
LRN1	55	55	-	-	96	96	-	
LRN2	27	27	-	-	256	256	-	
CONV2	500	375	9	9	32	48	0.24 MB	Street scene parsing (CNN) (e.g., identifying building, vehicle, etc.) [18].
POOL1	492	367	2	2	12	12	-	
CONV3*	200	200	18	18	8	8	1.29 GB	Face Detection in YouTube videos (DNN), (Google) [34].
CONV4*	200	200	20	20	3	18	1.32 GB	YouTube video object recognition, largest NN to date [8].

maps (N_{if} input feature maps with dimensions $N_x \times N_y$), the kernel is usually three-dimensional, i.e., $K_x \times K_y \times N_{if}$.

In DNNs, the kernels usually have different synaptic values for each output neuron (at each (x, y) position), while in CNNs, the kernels are *shared* across all neurons of the same output feature map. Convolutional layers with private (non-shared) kernels have *drastically more* synaptic weights (i.e., parameters) than the ones with shared kernels ($K \times K \times N_{if} \times N_{of} \times N_x \times N_y$ versus $K \times K \times N_{if} \times N_{of}$).

Pooling Layers. A pooling layer computes the max or average over a number of neighbor points, e.g.,

$$out(x, y)^f = \max_{0 \le k_x \le K_x, 0 \le k_y \le K_y} in(x + k_x, y + k_y)^f.$$

Its effect is to reduce the input layer dimensionality, which allows coarse-grain (larger scale) features to emerge, see Fig. 1, and be later identified by filters in the next convolutional layers. Unlike a convolutional or a classifier layer, a pooling layer has no learned parameter (no synaptic weight).

Local Response Normalization Layers. Local response normalization implements competition between neurons at the same location, but in different (neighbor) feature maps. Krizhevsky et al. [32] postulate that their effect is similar to the lateral inhibition found in biological neurons. The computations are as follows

$$out(x, y)^f = in(x, y)^f / \left(c + \alpha \sum_{g=\max(0, f-k/2)}^{\min(N_f - 1, f+k/2)} (in(x, y)^g)^2 \right)^{\beta},$$

where k determines the number of adjacent feature maps considered, and c, α and β are constants.

Classifier Layers. The result of the sequence of CONV, POOL and LRN layers is then fed to one or multiple classifier layers. This layer is typically fully connected to its N_i inputs (and it has N_o outputs), see Fig. 1, and each connection carries a learned synaptic weight. While the number of inputs may be much lower than for other layers (due to the dimensionality reduction of pooling layers), they can account for a large share of all synaptic weights in the neural network due to their full connectivity. Multi-Layer perceptrons are frequently used as classifier layers, though other types of classifiers are used as well (e.g., multinomial logistic regression). The goal of these layers is naturally to correlate the different features extracted from the filtering, normalization and pooling steps and the output categories,

$$out(j) = t\left(\sum_{i=0}^{N_i} w_{ij} * in(i) \right),$$

where $t()$ is a transfer function, e.g., $\frac{1}{1+e^{-x}}$, $tanh(x)$, $max(0, x)$ for ReLU [32], etc.

2.1.2 Benchmarks

Throughout this article, we use as benchmarks a sample of 10 of the largest known layers of each type, described in Table 1, as well as a full neural network (full NN), winner of the ImageNet 2012 competition [32]. The full NN benchmark contains the following 12 layers (the format is N_x, N_y, K_x, K_y, N_i or N_{if}, N_o or N_{of} as in the table): CONV (224,224,11,11,3,96), LRN (55,55,-,-,96,96), POOL (55,55,3,3,96,96), CONV (27,27,5,5,96,256), LRN (27,27,-,-,256,256), POOL (27,27,3,3,256,256), CONV (13,13,3,3,256,384), CONV (13,13,3,3,384,384), CONV (13,13,3,3,384,256), CLASS (-,-,-,-,9216,4096), CLASS (-,-,-,-,4096,4096), CLASS (-,-,-,-,4096,1000). For all convolutional layers, the sliding window strides s_x, s_y are 1, except for the first convolutional layer of the full NN, where they are 4. For all pooling layers, their sliding window strides equal to their kernel dimension, i.e., $s_x = K_x$, $s_y = K_y$. Note also that for LRN layers, $k = 5$. Finally, since we consider both inference and training for each layer, see Section 2.1.3, we have also considered the most popular *pre-training* method, i.e., the method used to initialize the synaptic weights, which is often time-consuming. This method is based on Restricted Boltzmann Machines (RBM) [45], and we applied it to CLASS1 and CLASS2 layers, leading to the RBM1 (2560×2560) and RBM2 (4096×4096) benchmarks.

2.1.3 Inference versus Training

A frequent and important misconception about neural networks is that *on-line* learning (a.k.a. training or backward phase) is necessary for many applications. On the contrary,

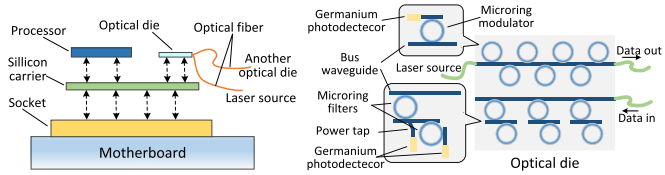

Fig. 2. Chip and optical die on a silicon carrier (left); (right) microring-based SiPh link.

for many industrial applications *off-line* learning is sufficient, where the neural network is first trained on a set of data, and then only used in inference (a.k.a. testing or feed-forward phase) mode by the end user. Note that even machine-learning researchers acknowledge this choice, as one of the few examples of hardware designs coming from that community is dedicated to inference [18]. While we put more emphasis in design and experiments on the much broader market of *users* of machine-learning algorithms, we have also designed the architecture to support the most common learning algorithms in order to also serve as an accelerator for machine-learning *researchers* and we also present experiments for that usage.

2.2 Silicon Photonics

In multi-chip architectures, the performance scalability is becoming increasingly limited by power dissipation, chip packaging and off-chip communications [55]. In particular, the off-chip communication must be compatible with the computational capacity of the chip so that it will not become a performance bottleneck but helps to achieve better energy efficiency.

Among existing interconnect technologies, the silicon photonics (SiPh) is a promising one that enables a transition to a new generation of scalable high-performance computing systems [58]. Technically, wavelength-division multiplexing (WDM) may enable high aggregate transceiver bandwidth without increasing the number of optical fibers used in the link [56], [57]. The Mach-Zehnder interferometers and rings based optical modulators are suitable devices for WDM due to their compact footprint and low power.

Recently there have been a number of successful studies in this area. Cignoli et al. [53] presented a 25 Gb/s Mach-Zehnder-based transmitter with 275 mW power consumption and $0.6\ \mathrm{mm}^2$ core area in 65 nm bulk CMOS. Chen et al. [51] developed a 25 Gb/s hybrid integrated transceiver in 28 nm CMOS and SOI with 123 mW power consumption. Rakowski et al. [56] proposed a 4×20 gb/s ring-based hybrid CMOS silicon photonics transceiver with area of 3.3×2.4 mm for 40 nm LP CMOS transceiver chip and 6×4.5 mm for 130 nm SOI silicon photonic transceiver chip. Ophir et al. [54] proposed a 225 GB/s SiPh microring links occupying approximately $1.5\ \mathrm{mm}^2$ net silicon and 4.5 W power consumption.

In this paper, we use the SiPh microring links [54] for optical interconnects. This link system integrates separate optical dies within a package. The optical dies and the processor are connected via a silicon carrier or shared substrate, as illustrated in Fig. 2. Each die contains two basic modules in Fig. 2. One is the transmit module based on a multiple microring modulators array, which is coupled with a shared-bus waveguide, and the other one is a demux and receive module based on a two-stage multiple microring demultiplexing array, which is coupled to a shared-bus waveguide. The radius of the rings in the transmit side and receive side arrays are gradually increasing so that a uniform frequency comb could be created from the combined rings resonances. Germanium photodetectors in microring filters provide detection information for thermal stabilization and high-speed signal. Two optical dies communicate over fiber, and off-chip separate laser sources are used to provide multi-wavelength laser.

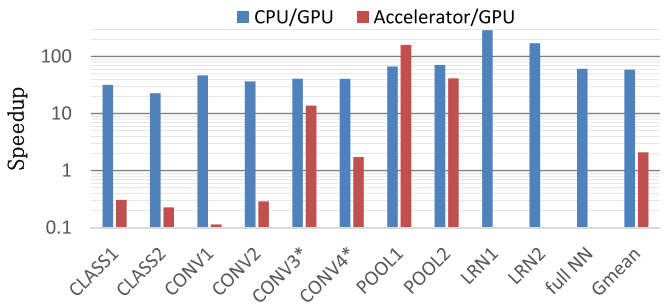

Fig. 3. Speedup of GPU over CPU (SIMD) and DianNao accelerator [5].

Fabrication variations and temperature variations are important issues for silicon photonics, which cause resonant shift in a static and dynamic way respectively. Fabrication variation in microrings results from the silicon-on-insulator (SOI) thickness unevenness and fabrication-process variability. However, as SOI thickness and fabrication process are fairly uniform on the single-die scale, it is feasible to create a fully array on a single die if a design fully spans the spectral range with equidistant resonance [65], [66]. To address temperature variation, the feedback control systems [63], [64] are used to thermally tune microring modulator.

3 The GPU Option

Currently, the most favored approach for implementing CNNs and DNNs are GPUs [6] due to the fairly regular nature of these algorithms. We have implemented in CUDA the different layer types of Table 1. We have also implemented a C++ version in order to obtain a CPU (SIMD) baseline. We have evaluated these versions on respectively a modern GPU card (NVIDIA K20M, 5 GB GDDR5, 208 GB/s memory bandwidth, 3.52 TFlops peak, 28 nm technology), and a 256-bit SIMD CPU (Intel Xeon E5-4620 Sandy Bridge-EP, 2.2 GHz, 1TB memory); we report the speedups of GPU over CPU (for inference) in Fig. 3. The GPU can provide a speedup of $58.82\times$ over a SIMD on average. This is in line with state-of-the-art results, for instance reported by Ciresan et al. [7], where speedups of $10\times$ for the smallest layers to $60\times$ for the largest layers are reported for an NVIDIA GTX480/GTX580 over an Intel Core-i7 920 on CNNs. One can also observe that the GPU is particularly efficient on LRN layers because of the presence of a dedicated exponential instruction, a computation which accounts for most the LRN execution time on SIMD.

While these speedups are high, GPUs have a number of limitations. First, their (area) cost is high because of both the number of hardware operators *and* the need to remain reasonably general-purpose (memory hierarchy, all PEs are connected to some elements of the memory hierarchy, etc).

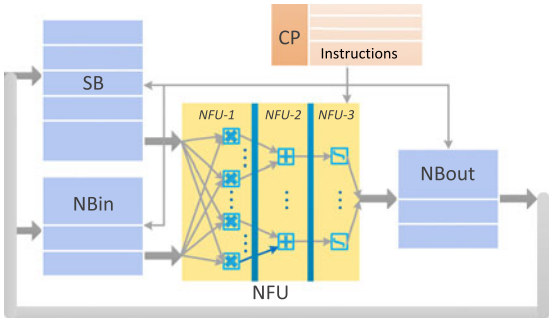

Fig. 4. Block diagram of the DianNao accelerator [5].

Second, the total execution time remains large (up to 18.03 seconds for the largest layer CLASS1); this may not be compatible with the milliseconds response time required by web services or other industrial applications. Third, the GPU energy efficiency is moderate, with an average power of over 74.93 W for the NVIDIA K20M GPU. That figure is actually optimistic because the NVIDIA K20M only contains 1.5 MB of on-chip RAM, forcing frequent high-energy accesses to the off-chip GDDR5 memory leading to a thermal design power of 225 W for the entire GPU board [43].

4 The Accelerator Option

Recently, Chen et al. [5] have proposed the DianNao accelerator for the fast and low-energy execution of the inference of large CNNs and DNNs in a small form factor (3 mm^2 at 65 nm, 0.98 GHz). We reproduce the block diagram of DianNao in Fig. 4. The architecture contains buffers for caching input/output neurons and synapses, and a Neural Functional Unit (NFU) which is largely a pipelined version of the typical computations required to evaluate a neuron output: the multiplication of synaptic values by input neurons values in the first stage, additions of all these products in the second stage (adder trees), and application of a transfer function in the third stage (realized through linear interpolation). Depending on the layer type (classifier, convolution, pooling), different computational operators are invoked in each stage.

In order to compare their architecture against GPU, we reimplement a cycle-level bit-level version of DianNao, and we use the memory latency parameters mentioned in their article. For the sake of comparison, we use at least some (four) of the same layers (CONV2, CONV4*, POOL1 and POOL2 respectively correspond to their CONV1, CONV5*, POOL1, POOL3; the layer numbers are different but the notations are the same), but we introduced even larger classifier layers (CLASS1 and CLASS2); CONV1 and CONV3* are large convolutional layers with respectively shared and private kernels, more closely matching the ones used in the references cited in Table 1. Since DianNao did not yet support LRN layers [5], we omit them from this comparison. In Fig. 3, we report the speedup of our GPU implementation (NVIDIA K20M) over DianNao. We can observe that DianNao can achieve about 47.91 percent of the GPU performance on average, in 0.53 percent of the area (the K20M is 561 mm^2 at 28 nm), which is a testimony to the potential efficiency of custom architectures.

However, the main limitation, acknowledged by the authors, is the memory bandwidth requirements of two important layer types: convolutional layers with private kernels (used in DNNs) and classifier layers used in both CNNs and DNNs. For these types of layers, the total number of required synapses can be massive, in the millions of parameters, or even tens or hundreds thereof. For an NFU processing 16 inputs of 16 output neurons (i.e., 256 synapses) per cycle, at 0.98 GHz a peak bandwidth of 467.30 GB/s would be necessary. As a reference, the NVIDIA K20M GPU has 320-bit memory interfaces at 2.6 GHz which can operate on every half-clock, for a total of 208 GB/s. Chen et al. [5] also report that off-chip memory accesses increase the total energy cost by a factor of approximately $10\times$.

In the next section, we propose a custom node and multi-chip architecture to overcome this limitation.

5 A Machine-Learning Supercomputer

We call the proposed architecture a "supercomputer" because its goal is to achieve high sustained machine-learning performance, significantly beyond single-GPU performance, and because this capability is achieved using a multi-chip system. Still, each node is significantly cheaper than a typical GPU while exhibiting a comparable or higher compute density (number of operations per second divided by the area).

We design the architecture around the central property, specific to DNNs and CNNs, that the total memory footprint of their parameters, while large (up to tens of GB), can be fully mapped to on-chip storage in a multi-chip system with a reasonable number of chips.

5.1 Overview

As explained in Section 4, the fundamental issue is the memory storage (for reuse) or bandwidth requirements (for fetching) of the synapses of two types of layers: convolutional layers with private kernels (the most frequent case in DNNs), and classifier layers (which are usually fully connected, and thus have lots of synapses). We tackle this issue by adopting the following design principles: (1) we create an architecture where synapses are always stored close to the neurons which will use them, minimizing data movement, saving both time and energy; the architecture is fully distributed, there is no main memory; (2) we create an asymmetric architecture where each node footprint is massively biased towards storage rather than computations; (3) we transfer neurons values rather than synapses values because the former are orders of magnitude fewer than the latter in the aforementioned layers, requiring comparatively little external (across chips) bandwidth; (4) we enable high internal bandwidth by breaking down the local storage into many tiles.

The general architecture is a set of nodes, one per chip, all identical, arranged in a classic mesh or torus topology. Each node contains significant storage, especially for synapses, and neural computational units (the classic pipeline of multipliers, adder trees and non-linear transfer functions implemented via linear interpolation), which we also call NFU for the sake of consistency with prior art, though our NFU is significantly more complex than the one proposed by Chen et al. [5] because its pipelined can be reconfigured for each layer and inference/training, see Section 5.2.3.

In the next sections, we detail each component and we explain the rationale for the design choices.

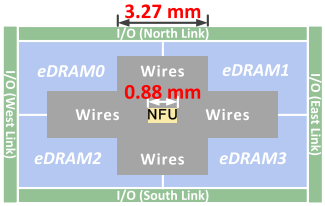

Fig. 5. Simplified floorplan with a single central NFU showing wire congestion.

Driving Example. We use the classifier layer as a driving example because it is both challenging due to its large number of synapses, but also structurally simple, and thus adequate as a driving example; note that for the sake of completeness, we explain in Section 5.2.3 how all layers are implemented on the architecture. As explained in Section 2, in a classifier layer, the N_o outputs are typically connected to all the N_i inputs, with one synaptic weight per connection. In terms of locality, it means that each input is reused N_o times, and that the synaptic weights are not reused within one classifier layer execution.

5.2 Node

In this section, we present the architecture node and explain the rationale for its design.

5.2.1 Synapses Close to Neurons

One of the fundamental design characteristic of the proposed architecture is to locate the storage for synapses close to neurons and to make it massive. This design choice is motivated by the decision to move only neurons and to keep synapses in a fixed storage location. This serves two purposes.

First, the architecture is targeted for both inference and training. In inference, the neurons of the previous layer are the inputs of the computation; in training, the neurons are forward-propagated (so neurons of the previous layer are the inputs) and then backward-propagated (so neurons of the *next* layer are now the inputs). As a result, depending on how data (neurons and synapses) are allocated to nodes, they need to be moved between the forward and backward phases. Since there are many more synapses than neurons (e.g., $O(N^2)$ versus $O(N)$ for classifier layers, $K \times K \times N_{if} \times N_{of} \times N_x \times N_y$ versus $N_{if} \times N_x \times N_y$ for convolutional layers with private kernels, see Section 2), it is only logical to move neuron outputs instead of synapses.

Second, having all synapses (most of the computation inputs) next to computational operators provides low-energy/low-latency data (synapses) transfers and high internal bandwidth.

As shown in Table 1, layer sizes can range from less than 1 MB to about 1 GB, most of them ranging in the tens of MB. While SRAMs are appropriate for caching purposes, they are not dense enough for such large-scale storage. However, eDRAMs are known to have a higher storage density. For instance, a 10 MB SRAM memory requires 20.73 mm^2 at 28 nm [36], while an eDRAM memory of the same size and at the same technology node requires 7.27 mm^2 [50], i.e., a $2.85\times$ higher storage density.

Moreover, providing sufficient eDRAM capacity to hold all synapses on the combined eDRAM of all chips will save

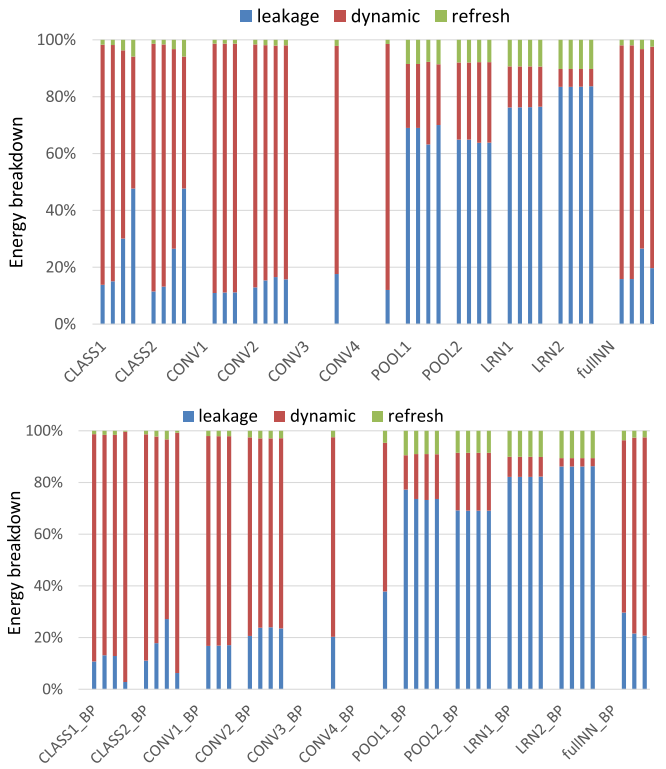

Fig. 6. The refresh overhead of eDRAM in $500\,\mu$s. Inference (up). Training (down).

on off-chip DRAM accesses, which are particularly costly energy-wise. For instance, a read access to a 256-bit wide eDRAM array at 28 nm consumes 0.0192nJ (50 μA, 0.9V, 606 MHz) [25], while a 256-bit read access to a Micron DDR3 DRAM consumes 6.18 nJ at 28 nm [40], i.e., an energy ratio of $321\times$. The ratio is largely due to the memory controller, the DDR3 physical-level interface, on-chip bus access, page activation, etc.

If the NFU is no longer limited by the memory bandwidth, it is possible to scale up its size in order to process more output neurons (N_o) and more inputs per output neuron (N_i) simultaneously, and thus, to improve the overall node throughput. For instance, to scale up by $16\times$ the number of operations performed every cycle compared to the accelerator mentioned in Section 4, we need to have $N_i = 64$ (instead of 16) and $N_o = 64$ (instead of 16). In order to achieve maximal throughput, we must fetch $N_i \times N_o$ 16-bit values from the eDRAM to the NFU every cycle, i.e., $64 \times 64 \times 16 = 65{,}536$ bits in this case.

However eDRAM has three well-known drawbacks: higher latency than SRAM, destructive reads and periodic refresh [38], as in traditional DRAMs. In order to compensate for the eDRAM drawbacks and still feed the NFU every cycle, we split the eDRAM into four banks (65,536-bit wide in the above example), and we interleave the synapses rows among the four banks. The eDRAM is split into four banks according to the low-order two bits of memory address, since the simultaneous accesses to different banks decomposed with the low-order bits of memory address are sufficient to support CNN/DNN techniques.

The refresh overhead of eDRAM in $500\,\mu$s is evaluated in Fig. 6. With $500\,\mu$s refreshing period, the refresh overhead averagely accounts for 5.23 percent of the whole energy

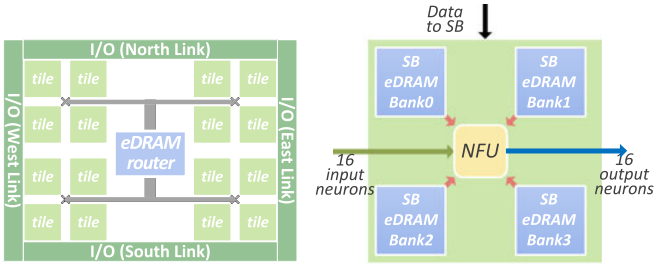

Fig. 7. Tile-based organization of a node (left) and tile architecture (right). A node contains 16 tiles, two central eDRAM banks and fat tree interconnect; a tile has an NFU, four eDRAM banks and input/output interfaces to/from the central eDRAM banks.

overhead of eDRAM for inference, and 5.53 percent for training. The percentage of refresh overhead for POOL layers and LRN layers is much higher than the average percentage, which mainly because during the the POOL layers and LRN layers, the weight eDRAMs are not accessed and the dynamic overhead of this part of eDRAMs is low. Furthermore, as proposed in paper [62], when employ the Error-Correcting-Code (ECC) mechanism, the refreshing period of eDRAM can be further reduced without any error on the data.

We placed and routed this design at 28 nm (ST technology, LP), and we obtained the floorplan of Fig. 5. The NFU footprint is very small at 0.78 mm^2 ($0.88\ mm \times 0.88\ mm$), but the process imposes an average spacing of $0.2\,\mu m$ between wires, and provides only four horizontal metal layers. As a result, the 65,536 wires connecting the NFU to the eDRAM require a width of $\frac{65{,}536 \times 0.2}{4} = 3.2768$ mm, see Fig. 5. Consequently, wires occupy $4 \times 3.2768 \times 3.2768 - 0.88 \times 0.88 = 42.18\ mm^2$, which is almost equal to the combined area of all eDRAM banks, all NFUs and the I/O.

5.2.2 High Internal Bandwidth

In order to avoid this congestion, we adopt a tile-based design, as shown in Fig. 7. The output neurons are spread out in the different tiles, so that each NFU can simultaneously process 16 input neurons of 16 output neurons (256 parallel operations), see Fig. 8. As a result, the NFU in each tile is significantly smaller, and only $16 \times 16 \times 16 = 4{,}096$ bits must be extracted each cycle from the eDRAM. We keep the four-bank (4,096-bit wide banks) organization to compensate for the aforementioned eDRAM weaknesses, and we obtain the tile design of Fig. 7. We placed and routed one such tile, and obtained an area of 1.89 mm^2, so that 16 such tiles account for 30.16 mm^2, i.e., a

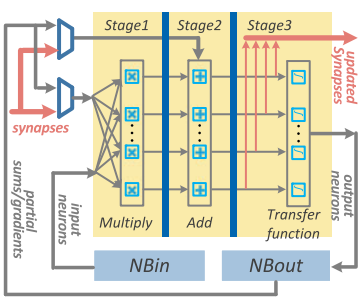

Fig. 8. The different (parallel) operators of an NFU: multipliers, adders, max, transfer function.

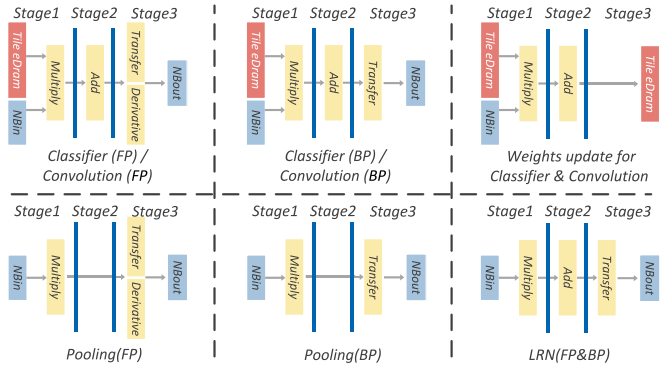

Fig. 9. Different pipeline configurations for CONV, LRN, POOL and CLASS layers.

28.5 percent area reduction over the previous design, because the routing network now only accounts for 8.97 percent of the overall area.

All the tiles are connected through a fat tree which serves to broadcast the input neurons values to each tile, and to collect the output neurons values from each tile. At the center of the chip, there are two special eDRAM banks, one for input neurons, the other for output neurons.

It is important to understand that, even with a large number of tiles and chips, the total number of *hardware* output neurons of all NFUs, can still be small compared to the actual number of neurons found in large layers. As a result, for each set of input neurons broadcasted to all tiles, multiple different output neurons are being computed on the same hardware neuron.

The intermediate values of these neurons are saved back locally in the tile eDRAM. When the computation of an output neuron is finished (all input neurons have been factored in), the value is sent through the fat tree to the center of the chip to the corresponding (output neurons) central eDRAM bank.

5.2.3 Configurability (Layers, Inference versus Training)

We can adapt the tile, and the NFU pipeline in particular, to the different layers and the execution mode (inference or training). The NFU is decomposed into a number of hardware blocks: adder block (which can be configured either as a 256-input, 16-output adder tree, or 256 parallel adders), multiplier block (256 parallel multipliers), max block (16 parallel max operations), and transfer block (two independent sub-blocks performing 16 piecewise linear interpolations; the a, b linear interpolation coefficients, i.e., $y = a \times x + b$, for each block are stored in two 16-entry SRAMs and can be configured to implement any transfer function and its derivative). In Fig. 9, we show the different pipeline configurations for CONV, LRN, POOL and CLASS layers in the forward and backward phases.

Each hardware block is designed to allow the aggregation of 16-bit operators (adders, multipliers, max, and the adders/multipliers used for linear interpolation) into fewer 32-bit operators (two 16-bit adders into one 32-bit adder, four 16-bit multipliers into 32-bit multiplier, two 16-bit max into one 32-bit max); the overhead cost of aggregable operators is very low [26], [35]. As introduced in previous works [5], 16-bit operators are largely sufficient for the inference

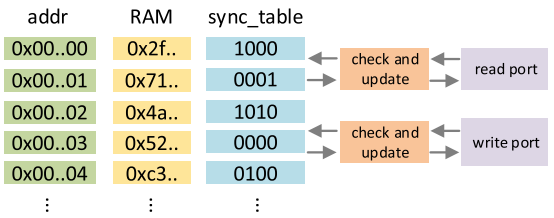

Fig. 10. Simplified architecture of synchronization table.

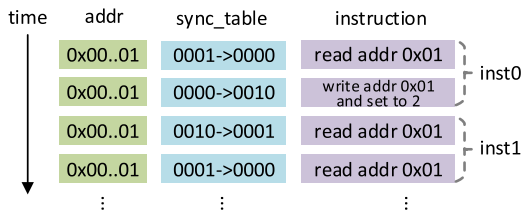

Fig. 11. Operation process of synchronization table.

usage, and 32-bit operators are used to keep the accuracy and the convergence of training. A recent work [61] demonstrates that 16-bit representation when using stochastic rounding and variable fractional lengths, incur little to no degradation in training on MNIST dataset [35] and CIFAR10 dataset [32]. Thus, 32-bit and 16-bit operators are both supported with negligible hardware overhead of the 16-bit to 32-bit reconfiguration (less than 2 percent of the whole chip area).

Beyond pipeline and block configurations, the tile must be configured for different data movement cases. For instance, a classifier layer input can come from the node central eDRAM (possibly after transfer from another node), or it can come from the two SRAM storages (16 KB) which are used to buffer input and output neuron values, or even temporary values (such as neurons partial sums to enable reuse of input neurons values) as proposed by Chen et al. [5]. In the backward phase, the NFU must also write to the tile eDRAM after the weights update step, see Fig. 9. During the gradient computations step, the input and output gradients use the data paths of input and output neurons in the forward phase, see Fig. 9 again.

5.2.4 Synchronization Table

In an instruction pipeline, data hazards occur when multiple instructions with data dependency modify data access to the same data address, including read after write (RAW) and write after read (WAR). Traditionally, this issue is addressed with reservation stack, in which an instruction will be blocked until the previous instruction with data dependency has been completely processed. Reservation stack may hinder parallel executions of multiple instructions.

We design a synchronization table (sync table for short) to avoid data hazards, as illustrated in Fig. 10. A sync table provides a counter array for certain memory space, and each counter is reserved for a unique memory address. Each instruction that writes the address must specify how many times the written value will be read by follow-up instructions (i.e., set the corresponding counter be that number in the sync table), as soon as it has performed the write. This is feasible to neural network applications as their data dependency is deterministic and can be determined by the compiler. When a later instruction attempts to read the address, the read port must make sure that the corresponding counter is not 0, and then reduces the counter by 1 after reading. Similarly, a later instruction can write the address only when the counter is 0; As soon as the instruction has written the address, it sets the corresponding counter to protect the written value from being incorrectly modified by future instructions (see Fig. 11). Read and write ports can also access the memory directly by ignoring the sync table. This design can synchronize data reading and writing in the granularity size of a memory address, and effectively handle data dependency between instructions.

5.3 Interconnects

5.3.1 Electrical Interconnects

Because neurons are the only values transferred, and because these values are heavily reused within each node, the amount of communications while significant, is not a bottleneck except for a few layers and many-node systems, as later discussed in Section 7. As a result, we did not develop a custom high-speed interconnect for our purpose, we turned to commercially available high-performance interfaces, and we used a HyperTransport (HT)2.0 IP block. The HT2.0 physical layer interface (PHY) we used for the 28 nm process is a long thin strip of 5.635 mm $\times$ 0.5575 mm (with a protrusion) due to its usual location at the periphery of the die.

We use a simple 2D mesh/torus topology; that choice may be later revisited in favor of a more efficient 3D mesh topology though. Because of the mesh/torus topology of the architecture, each chip must connect to four neighbors via four HT2.0 IP blocks (see Fig. 15), each with $16\times$ HT links, i.e., 16 pairs of differential outgoing signals, and 16 pairs of differential incoming signals, at a frequency of 1.6 GHz (we connect the HT to the central eDRAM through a 128-bit, four-entry, asynchronous FIFO). Each HT block provides a bandwidth of 6.4 GB/s in each direction. The HT2.0 latency between two neighbor nodes is about 80 ns.

5.3.2 Optical Interconnects

Inter-node communication is a performance bottleneck for classifier layers in many-node neural network accelerators, since neuron values are heavily and frequently transferred. To achieve lower power and higher speed, we evaluate optical interconnects with silicon photonic (SiPh) microring links, and integrate it to our system. The SiPh microring links used has 255 GB/s bandwidth and net silicon area of approximate $1.5\ mm^2$ [54]. We implement a torus topology, where each chip is connected to four neighbors via four SiPh microring link blocks (see Fig. 15). Each link block is with 36 wavelength channels for 12.5-Gbps modulation rate, and each SiPh microring link provides a bandwidth of 56.25 GB/s in each direction. The SiPh link latency between two neighbor nodes is 0.08 ns.

5.3.3 Router

Next to the central block of the tile, we implement the router, see Fig. 7. We use wormhole routing, the router has five input/output ports (four directions and injection/ejection port). Each input port contains eight virtual channels

TABLE 2
Architecture Characteristics

Parameters	Settings	Parameters	Settings
Frequency	606 MHz	central eDRAM size	4 MB
# of tiles	16	central eDRAM latency	~10 cycles
# of 16-bit multipliers/tile	256+32	HT link bandwidth	4×6.4 GB/s
# of 16-bit adders/tile	256+32	HT link latency	80 ns
tile eDRAM size/tile	2 MB	SiPh link bandwidth	225 GB/s
tile eDRAM latency	~3 cycles	SiPh link latency	0.08 ns

(five flit slots per VC). A 5×5 crossbar is equipped to connect all input/output ports. The router has four pipeline stages: routing computation (RC), VC allocation (VA), switch allocation (SA) and switch traversal (ST).

5.4 Overall Characteristics

The architecture characteristics are summarized in Table 2. We have implemented 16 tiles per node. In each tile, each of the four eDRAM banks contains 1,024 rows of 4,096 bits. The total eDRAM capacity in one tile is thus $4 \times 1{,}024 \times 4{,}096 = 2$ MB. The central eDRAM in each node has a size of 4 MB. The total node eDRAM capacity is thus $16 \times 2 + 4 = 36$ MB.

In order to avoid the circuit and time overhead of asynchronous transfers, we decided to clock the NFU at the same frequency as the eDRAM available in the 28 nm technology we used, i.e., 606 MHz. Note that the NFU implemented by Chen et al. [5] was clocked at 0.98 GHz at 65 nm, so our decision is very conservative considering we use a 28 nm technology. We leave the implementation of a faster NFU and asynchronous communications with eDRAM for future work. Nonetheless, a node still has a peak performance of $16 \times (288 + 288) \times 606 = 5.58$ TeraOps/s for 16-bit operation ($256 + 32 = 288$ 16-bit adders/tile and $256 + 32 = 288$ 16-bit multipliers/tile). For 32-bit operation, the peak performance of a node is $16 \times (144 + 72) \times 606 = 2.09$ TeraOps/s due to operator aggregation ($2{,}882 = 144$ 32-bit adders/tile and $2{,}884 = 72$ 32-bit multipliers/tile), see Section 5.2.3.

5.5 Programming, Code Generation and Multi-Node Mapping

5.5.1 Programming, Control and Code Generation

This architecture can be viewed as a system ASIC, so the programming requirements are low, the architecture essentially has to be configured and the input data is fed in. The input data (values of the input layer) is initially partitioned across nodes and stored in a central eDRAM bank. The neural network configuration is implemented in the form of a sequence of *node* instructions, one sequence per node, produced by a code generator (c.f., Table 3). An example output of the code generator for the inference phase of the CLASS2 layer is shown in Table 4.

In this example, output neurons are partitioned into multiple 256-bit data blocks, where each block contains $256/16 = 16$ neurons. Each node is allocated $4{,}096/16/4 = 64$ output data blocks (and it stores a quarter of all input neurons, i.e., $4{,}096/4 = 1{,}024$), and each tile is allocated $64/16 = 4$ output data blocks, resulting in four instructions per node. An instruction will load 128 input data blocks from the central eDRAM to the tiles. In the first three instructions, all the tiles will get the same input neurons, and read synaptic weights from their local (tile) eDRAM, then write back the partial sums (of output neurons) to their local NBout SRAM. In the last instruction, the NFU in each tile will finalize the sums, apply the transfer function, and store the output values back to the central eDRAM.

TABLE 3
Node Instruction Format

CP	central eDRAM								SB				NBin				NBout				NFU				
Inst Name	READ OP	WRITE OP	READ ADDR	WRITE ADDR	READ STRIDE	WRITE STRIDE	READ ITER	WRITE ITER	READ OP	WRITE OP	ADDR	STRIDE	READ OP	WRITE OP	ADDR	STRIDE	READ OP	WRITE OP	ADDR	STRIDE	NFU-1 OP	NFU-2 OP	NFU-3 OP	NFU-2-IN	NFU-2-OUT

These node instructions themselves drive the control of each tile; the control circuit of each node generates tile instructions and sends them to each tile. The spirit of a node or tile instruction is to perform the same layer computations (e.g., multiply-add-transf for classifier layers) on a set of *contiguous* input data (input neurons in the forward phase, output neurons, gradients or synapses in the backward phase). The fact the data of one instruction is contiguous allows to characterize it with only three operands: start address, step and number of iterations.

The control provides two modes of operations: processing one row at a time or *batch learning* [48], where multiple rows are processed at the same time, i.e., multiple instances of the same layer are evaluated simultaneously, albeit for different input data. This method is commonly used in machine-learning for a more stable gradient descent, and it also has the benefit of improving synapses reuse, at the cost of slower convergence and a larger memory capacity (since multiple instances of inputs/outputs must be stored).

5.5.2 Multi-Node Mapping

At the end of a layer, each node contains a set of output neurons values, which have been stored back in the central eDRAM, see Fig. 7. These output neurons form the input neurons of the next layer; so, implicitly, at the beginning of a layer, the input neurons are distributed across all nodes, in the form of 3D rectangles corresponding to all feature maps of a subset of a layer, see Fig. 12. These input neurons will be first distributed to all node tiles through the (fat tree)

TABLE 4
An Example of Classifier Code ($N_i = 4{,}096$, $N_o = 4{,}096$, four nodes)

CP	central eDRAM								SB				NBin				NBout				NFU				
Class	LOAD	LOAD	WRITE	0	256	0	4	64	4	READ	NULL	0	1	READ	NULL	0	1	NULL	WRITE	0	1	MUL	ADD	NULL			
Class	LOAD	WRITE	64	256	0	4	64	4	READ	NULL	256	1	READ	NULL	0	1	READ	WRITE	0	1	MUL	ADD	NULL			
Class	LOAD	WRITE	128	256	0	4	64	4	READ	NULL	512	1	READ	NULL	0	1	READ	WRITE	0	1	MUL	ADD	NULL			
Class	LOAD	STORE	192	256	0	4	64	4	READ	NULL	768	1	READ	NULL	0	1	READ	WRITE	0	1	MUL	ADD	SIGMOID			

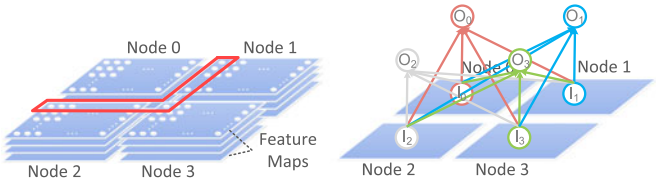

Fig. 12. Mapping of (left) a convolutional (or pooling) layer with four feature maps; the red section indicates the input neurons used by node 0; (right) a classifier layer.

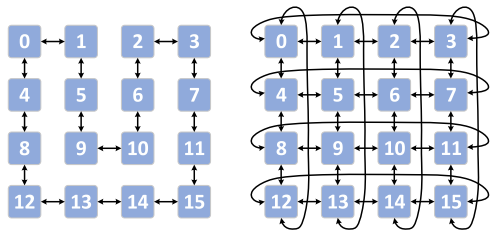

Fig. 13. Ring (left) and 2D torus (right).

internal network, see Fig. 7. Simultaneously, the node control starts to send the block of input neurons to the rest of the nodes through the mesh.

With respect to communications, there are three main layer cases to consider. First, convolutional and pooling layers are characterized by local connectivity defined by the small window (convolutional or pooling kernel) used to sample the input neurons. Due to the local connectivity, the amount of inter-node communications is very low (most communications are intra-node), mostly occurring at the border of the layer rectangle mapped to each node, see Fig. 12.

For local response normalization layers, since all feature maps at a given location are always mapped to the same node, there is no inter-node communication.

Finally, communications can be high for classifier layers because each output neuron uses all input neurons, see Fig. 12. At the same time, the communication pattern is simple, equivalent to a broadcast. Since each node performs roughly the same amount of computations at the same speed, and since each node must simultaneously broadcast its set of input neurons to all other nodes, in our previous work [52] we adopt a computing-and-forwarding communication scheme [24], which is equivalent to arranging the nodes communications according to a regular ring pattern. A node can start processing the newly arrived block of input neurons as soon as it has finished its own computations, and has sent the previous block of input neurons; so the decision is made *locally*, there is no *global synchronization* or *barrier*. Denote the numbers of nodes and output data blocks to be n and k, respectively. In the ring topology, the mean hop counts for each transfer is $\lfloor n/2 \rfloor$, and the total hop counts are $k \times \lfloor n/2 \rfloor$. Hence, given a larger number of nodes, it will take longer time to send each block of inputs to all nodes.

To reduce the broadcasting time, we use the 2D torus topology for the classifier layer (see Fig. 13). Each node in the same column shares the same block of input neurons, while all nodes in the same row together compute one block of output neurons. The torus topology has a two-stage dataflow in Fig. 14. In the first stage, the nodes in the same column share the same block of input neurons, e.g., the node 0, 3, and 6 share the same input neuron block. The nodes in the same row separately compute the partial sums of the same output neuron blocks with different blocks of input neurons. Then, they transmit the partial sums to the diagonal nodes (i.e., node 0, 4, 8), and the partial sums will be accumulated on the path to the diagonal nodes. In the second stage, each diagonal node broadcasts the output block to the rest nodes within the same column, once each diagonal node computes the final block output neuron block. In this way, nodes in the same row together compute one block of output neurons, e.g., the node 0, 1, 2 together compute the output neurons.

In other words, in the 2D torus topology, $n^{\frac{1}{2}}$ blocks of output neurons can be simultaneously computed with nodes in different rows. Each node accumulates the newly arrived block of output neurons and sends the output to a node in the same row, as soon as it has processed the block of input neurons. The final block of output neurons is computed by the node at the intersection of the current row and the diagonal of the torus topology, will be input neurons of the next layer, and are sent to the rest nodes in the same column. In the torus topology, the mean hop counts for each transfer is $n^{\frac{1}{2}}$, and $n^{\frac{1}{2}}$ blocks of output data can be sent together, whereas in the ring topology each time only one block can be sent. Thus, the total hop counts is k (i.e., $(k/n^{\frac{1}{2}}) \times n^{\frac{1}{2}}$), which is independent of the number of nodes. In short, the torus topology achieves more efficient inter-node communications and better scalability compared to the ring topology.

6 Methodology

6.1 Measurements

Our experiments use the following three tools.

CAD Tools. We implemented a Verilog version of the node, then synthesized it, and did the layout. The area, energy and critical path delays are obtained after layout using the ST 28 nm Low Power (LP) technology (0.9 V). We used the Synopsys Design Compiler for the synthesis, ICC Compiler for the layout, and the power consumption was estimated using Synopsys PrimeTime PX.

Time, eDRAM and Inter-Node Measurements. We use VCS to simulate the node RTL, an eDRAM model which includes destructive reads, and periodic refresh of a banked eDRAM running at 606 MHz (the eDRAM energy was collected using CACTI5.3 [1] after integrating the 1T1C cell characteristics at 28 nm [25]), and inter-node communications were simulated using the cycle-level Booksim2.0 interconnection network simulator [10] (Orion2.0 [29] for the network energy model).

GPU. We use the NVIDIA K20M GPU of Section 3 as a baseline. The GPU can also report its power usage. We use

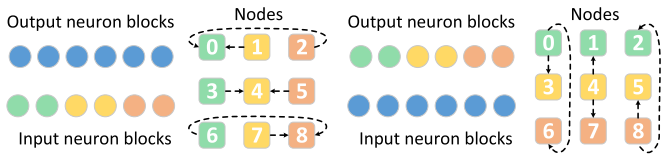

Fig. 14. The dataflow of 2D torus. In the first stage, the nodes in the same column share the same block of input neurons (left). In the second stage, each diagonal node broadcasts the output block to the rest nodes within the same column (right).

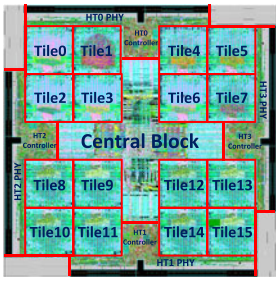

Fig. 15. Snapshot of the node layout.

CUDA SDK 5.5 to compile the CUDA version of neural network codes.

6.2 Baseline

In order to maximize the quality of our baseline, we extracted the CUDA versions from a tuned open-source version, cuda-convnet [31]. In order to assess the quality of this baseline, we have compared it against the C++ version run on the Intel SIMD CPU, see Section 3. For the C++ version, we have first compared the SIMD version against a non-SIMD version (SIMD compilation deactivated), and we have observed an average speedup of the SIMD version of $4.07\times$, confirming that the compiler was effectively taking advantage of the SIMD unit. As mentioned in Section 3, the CUDA/GPU over the C++/CPU (SIMD) speedups reported in Fig. 3 are in line with some of the best reported results so far, by Ciresan et al. [7] ($10\times$ to $60\times$).

7 Experimental Results

We first present the main characteristics of the node layout, then present the performance and energy results of the multi-chip system.

7.1 Main Characteristics

The cell-based layout of the chip with electrical interconnects is shown in Fig. 15, and the area breakdown in Table 5. 44.53 percent of the chip area is used by the 16 tiles, 26.02 percent by the four HT IPs, 11.66 percent by the central block (including 4 MB eDRAM, router and control logic). The wires between the central block and the tiles occupy 8.97 percent of the area. Overall, about a half (47.55 percent) of the chip is consumed by memory cells (mostly eDRAM). The combinational logic and register only account for 5.88 and 4.94 percent of the area respectively.

TABLE 5
Node Layout Characteristics for Chip with HT-Ring

Component/Block	Area (μm^2)	(%)	Power (W)	(%)
Whole chip	67,732,900		15.97	
Central Block	7,898,081	(11.66%)	1.80	(11.27%)
Tiles	30,161,968	(44.53%)	6.15	(38.53%)
HTs	17,620,440	(26.02%)	8.01	(50.14%)
Wires	6,078,608	(8.97%)	0.01	(0.06%)
Other	5,973,803	(8.82%)		
Combinational	3,979,345	(5.88%)	6.06	(37.97%)
Memory	32,207,390	(47.55%)	6.12	(38.30%)
Registers	3,348,677	(4.94%)	3.07	(19.25%)
Clock network	586323	(0.87%)	0.71	(4.48%)
Filler cell	27,611,165	(40.76%)		

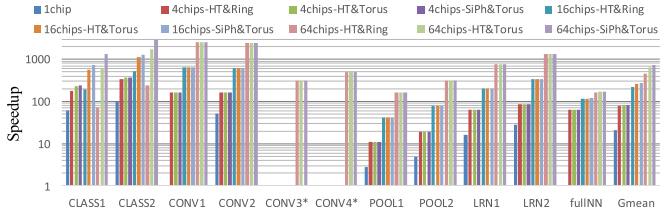

Fig. 16. Speedup w.r.t. the GPU baseline (inference). Note that CONV1 and the full NN need a four-node system, while CONV3* and CONV4* even need a 36-node system.

We used Synopsys PrimePower to estimate the power consumption of the chip with electrical interconnects. The peak power consumption is 15.97 W (at a pessimistic 100 percent toggle rate), i.e., roughly 5-10 percent of a state-of-the-art GPU card. The architecture block breakdown shows that the tiles consume more than one third (38.53 percent) of the power, and the four HT IPs consume about one half (50.14 percent). The component breakdown shows that, overall, memory cells (tile eDRAMs + central eDRAM) account for 38.30 percent of the total power, combinational logic and registers (mostly NFUs and HT protocol analyzers) consume 37.97 and 19.25 percent respectively.

For optical interconnects, we directly use area and power of SiPh links reported in [54] to replace area and power of HT IPs, because I/O blocks are independent of eDRAMs, NFUs, and wires in the floorplan (see Fig. 5). Compared to the chip with HT IPs, the chip with optical interconnects saves 17.16 percent area, and SiPh links only occupy 10.69 percent of the chip area. The peak power consumption of the chip is 12.46 W, which is 21.98 percent lower than of the chip with HT IPs.

7.2 Performance

In Fig. 16, we compare the performance of our architecture against the GPU baseline described in Section 6. Because of its large memory footprint (numbers of neurons and synapses), CONV1 needs a four-node system. Even though CONV1 is a shared-kernel convolutional layer, it contains 256 input feature maps, 384 output feature maps and 11×11 kernels, so that the total number of synapses is $256 \times 384 \times 11 \times 11 = 11,894,784$, i.e., 22.69 MB (16-bit data). We must also store all layer inputs and outputs, i.e., respectively $256 \times 256 \times 256 \times 2 = 32$ MB, $246 \times 246 \times 384 \times 2 = 44.32$ MB (fewer output neurons due to a border effect since the kernel is 11×11). So, overall, 99.01 MB must be stored, which exceeds the node capacity of 36 MB. The convolutional layers with private kernels, i.e., CONV3* and CONV4*, need a 36-node system because their size is respectively 1.29 and 1.32 GB.

The full NN contains 59.48 M synapses, i.e., 118.96 MB (16-bit data), requiring at least four nodes.

On average, the 1-node, 4-node, 16-node and 64-node architectures integrating HT and the ring topology (called HT-Ring) are respectively $21.38\times$, $79.81\times$, $216.72\times$, and $450.65\times$, faster than the GPU baseline.[1] The first reason for

1. Considering that the area of K20M GPU is about 550 mm^2, and our node with HT-Ring is only 67.7 mm^2, our design also has a high area-normalized speedup with respect to GPU ($21.38 \times 550/67.7 = 173.69\times$ for 1-node and $450.65 \times 550/(64 \times 67.7) = 57.20\times$ for 64-node with HT-Ring).

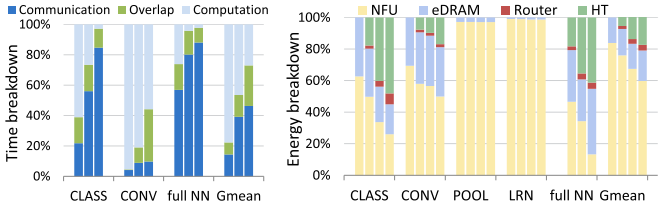

Fig. 17. Time breakdown (left) for 4, 16, 64 nodes with HT-Ring, (right) energy breakdown for 1, 4, 16, 64 nodes with HT-Ring; CLASS, CONV, POOL, LRN stand for the geometric means of all layers of the corresponding type, Gmean for the global geometric mean.

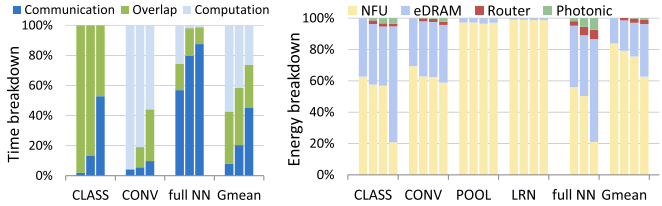

Fig. 18. Time breakdown (left) for 4, 16, 64 nodes with SiPh-Torus, (right) energy breakdown for 1, 4, 16, 64 nodes with SiPh-Torus; CLASS, CONV, POOL, LRN stand for the geometric means of all layers of the corresponding type, Gmean for the global geometric mean.

the higher performance is the large number of operators: in each node, there are 9,216 operators (mostly multipliers and adders), compared to the 2,496 MACs of the GPU. The second reason is that the on-chip eDRAM provides the necessary bandwidth and low-latency access to feed these many operators.

Nevertheless, the scalability of the different layers varies a lot. LRN layers scale the best (no inter-node communication) with a speedup of up to $1340.77\times$ for 64 nodes (LRN2), CONV and POOL layers scale almost as well because they only have inter-node communications on border elements, e.g., CONV1 achieves a speedup of $2595.23\times$ for 64 nodes, but the actual speedup of LRN and POOL layers is lower than CONV layers because they are less computationally intensive. On the other hand, CLASS layers scale less well because of the high amount of inter-node communications, since each output neuron uses all input neurons from different nodes, see Section 5.5.2, e.g., CLASS1 has a speedup of $72.96\times$ for 64 nodes. This is further illustrated in the time breakdown of Fig. 17. Note that each bar is normalized to the total execution time. This communication issue is mostly due to the relatively simple ring topology where the larger the number of nodes, the longer the time required to send each block of inputs to all nodes. It is likely that a more sophisticated multi-dimensional torus topology [4] can largely reduce the total broadcast time as the number of nodes increases, and we analyze the performance of this optimization later.

We note that the full NN scales similarly to CLASS layers ($63.35\times$, $116.85\times$, $164.80\times$ for 4-node, 16-node, 64-node with HT-Ring respectively). While it may suggest that CLASS layers dominate the full NN execution time, a breakdown by layer type, see Table 6, shows that it is not the case, on the contrary, CONV layers largely dominate. The reason is simply that this full NN contains layers which are a bit small for a large 64-node machine. For instance, there are three CONV layers with a dimension of 13×13 (though 256 to 384 feature maps), so, even though all feature maps are mapped to the same node, we need to attribute an $X \times Y$ area of size 2×2 or 3×3 at most per node ($\frac{13\times13}{64} = 2.69$) which means that either we do not use all nodes, or inter-node communications are required for every kernel computation.

TABLE 6
Full NN Execution Time Breakdown per Layer Type

	CONV	LRN	POOL	CLASS
4-node	96.63%	0.60%	0.47%	2.31%
16-node	96.87%	0.28%	0.22%	2.63%
64-node	92.25%	0.10%	0.08%	7.57%

The architectures with HT and 2D torus topology (called HT-Torus) have better scalability than architectures with HT-Ring, for layers with intensive inter-node communications, e.g., the 64-node architecture is $8.49\times$ faster than the HT-Ring for CLASS1. This improvement is mostly due to that the simple ring topology takes significantly longer time to broadcast data than the 2D torus topology (see Fig. 19). On average, the 4-node, 16-node and 64-node architectures with HT-Torus are respectively $1.04\times$, $1.23\times$, and $1.46\times$ faster than the HT-Ring. Compared with the GPU baseline, the 4-node, 16-node and 64-node architectures achieve remarkable speedup on average ($82.81\times$, $266.27\times$ and $656.63\times$), as illustrated in Fig. 16. The speedup of full NN is slight ($1.00\times$, $1.01\times$, $1.02\times$ for 4-node, 16-node, 64-node respectively over HT-Ring), since CONV layers largely dominate the full NN execution time.

Finally, architectures with SiPh links further improve the performance of CLASS layers due to the high transfer speed. With the same ring topology, the 64-node architecture with SiPh links is $1.26\times$ faster than that with HT links, as in Fig. 19. With the same torus topology, CLASS1 has a speedup of $2.20\times$ for 64 nodes over HT-Torus. The effective reduction of inter-node communications are also illustrated in the time breakdown of Figs. 17 and 18. On average, 4-node, 16-node and 64-node architectures with SiPh-Torus are respectively $1.04\times$, $1.28\times$, and $1.65\times$, faster than the HT-Ring. Meanwhile, on average, 4-node, 16-node and 64-node architectures with SiPh-Torus are respectively $1.01\times$, $1.04\times$, and $1.13\times$, faster than the HT-Torus. The speedup of full NN over the HT-Ring is slight ($1.01\times$, $1.02\times$, $1.04\times$ for 4-node, 16-node, 64-node respectively) .

Training and Initialization. We carry out similar experiments for back propagation, and the weights pre-training phase (RBM) using 32-bit fixed point operators (while inference uses 16-bit fixed-point operators), see Section 5.2.3. On average, the 1-node, 4-node, 16-node and 64-node architectures with HT-Ring are respectively $12.62\times$, $43.23\times$, $126.66\times$ and $300.04\times$ faster than the GPU baseline; the

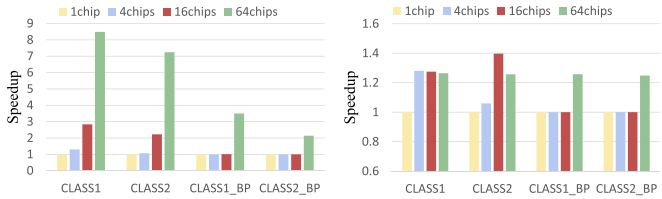

Fig. 19. Speedup for classifier layers. (left) With HT links, torus versus ring topology; (right) with ring topology, SiPh versus HT.

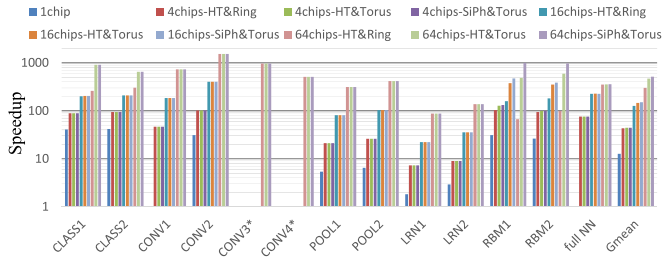

Fig. 20. Speedup w.r.t. the GPU baseline (training).

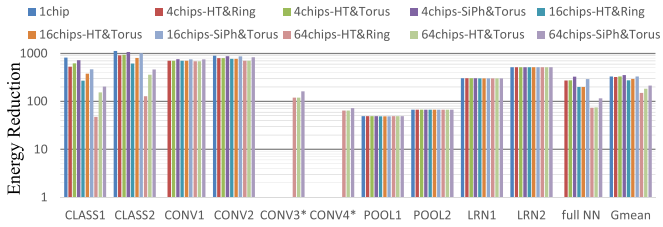

Fig. 21. Energy reduction w.r.t. the GPU baseline (inference).

speedups are high though lower than for inference essentially because of operators aggregation. As shown in Fig. 20, for CLASS layers, the scalability of the training phase is better than that of the inference phase, mainly due to CLASS layers that almost double the amount of computations of inference but only require the same amount of communications. The scalability of RBM initializations is fairly similar to that of CLASS layers in the inference phase.

By improving the topology of the architecture, on average, the 4-node, 16-node, and 64-node architectures with HT-Torus are respectively $1.03\times$, $1.15\times$ and $1.56\times$ faster than the HT-Ring. Since RBM layers has similar high amount of inter-node communications to that of CLASS layers in the inference phase, RBM1 has a speedup of $7.24\times$ for 64 nodes.

The 4-node, 16-node, and 64-node architectures with SiPh-Torus have similar speedup with HT-Torus, and are respectively $1.03\times$, $1.19\times$ and $1.74\times$ faster than the HT-Ring. Due to high inter-node communications, RBM layers achieves significant speedup for 64 nodes, e.g., RBM1 has a speedup of $14.90\times$ over HT-Ring, and has a speedup of $2.06\times$ over HT-Torus.

7.3 Energy Consumption

As shown in Fig. 21, the 1-node, 4-node, 16-node and 64-node architectures with HT-Ring can reduce the energy by $330.56\times$, $323.74\times$, $276.04\times$, and $150.31\times$ respectively compared to the GPU baseline. The minimum energy improvement is $47.66\times$ for CLASS1 with 64 nodes, while the best energy improvement is $896.58\times$, achieved with CONV2 on a single node. For convolutional, pooling, and LRN layers, we observe that the energy benefit remains relatively stable as the number of nodes is scaled up, though it degrades for classifier layers. The latter is expected as the average communication (and thus overall execution) time increases; again, a multi-dimensional torus should help reduce this energy loss.

In the energy breakdown of Fig. 17, we can observe that, for the one-node architecture, about 83.89 percent of the energy is consumed by the NFU. As we scale up the number of nodes, the trend largely corroborates previous observations: the ratio of energy spent in HT progressively increases to 29.32 percent on average for a 64-node system, especially due to the larger number of communications in classifier layers (48.11 percent).

The architectures with 2D torus topology (HT-Torus) significantly reduce the overall inter-node communication time, and reduce the energy by $3.24\times$ for CLASS1 with 64 nodes of HT-Ring. Specifically, the 4-node, 16-node and 64-node architectures can reduce the energy by $1.02\times$, $1.07\times$ and $1.22\times$ respectively compared to the HT-Ring, and achieve energy improvement by $330.68\times$, $294.92\times$ and $184.05\times$ respectively over the GPU baseline.

Since SiPh links have lower power consumption compared to HT IPs, the architectures with SiPh-Torus achieve better energy improvement as shown in Fig. 21. The 4-node, 16-node and 64-node architectures reduce the energy by $1.09\times$, $1.20\times$ and $1.42\times$ respectively compared to the HT-Ring, and reduce the energy by $1.07\times$, $1.12\times$ and $1.16\times$ respectively over the HT-Torus. The best energy improvement is $4.28\times$, achieved with CLASS1 on 64 nodes connected with HT-Ring. For convolutional layers, the energy benefit increases slightly with the number of nodes, e.g., CONV2 achieves energy benefits of $1.09\times$, $1.12\times$, and $1.18\times$ for 4-node, 16-node, and 64-node respectively.

In the energy breakdown of Fig. 18, the energy consumed by I/O interface is greatly reduced in architectures with SiPh-Torus, due to the ultra low power of SiPh links and the efficient torus topology. The ratio of energy spent in SiPh links is low, and slightly increases to 3.27 percent on average for a 64-node system.

Training and Initialization. For training and initialization, the energy reduction of our architecture with HT-Ring with respect to the GPU baseline on training is also high: $172.39\times$, $180.42\times$, $142.59\times$, and $66.94\times$ for the 1-node, 4-node, 16-node and 64-node architectures, see Fig. 22. The scalability behavior is similar to that of the inference phase.

The energy benefit achieved by topology is not as significant as that by SiPh links. The energy reduction of architectures with HT-Torus on training is $1.01\times$, $1.02\times$, and $1.06\times$ for the 4-node, 16-node and 64-node architectures compared to HT-Ring. The 4-node, 16-node, and 64-node architectures with SiPh-Torus reduce energy by $1.10\times$, $1.13\times$, and $1.23\times$ respectively over HT-Ring.

8 Related Work

Machine-Learning. Convolutional and Deep Neural Networks have become popular algorithms in data center based services: they are used for web search [27], image analysis [41], speech recognition [9], etc. Such services are

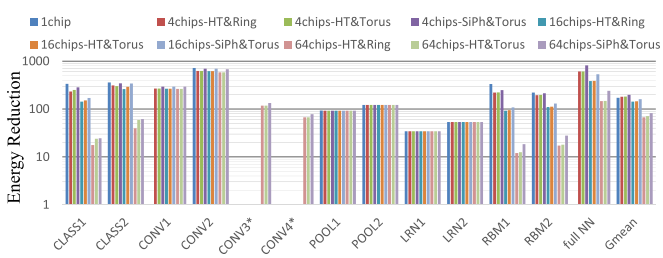

Fig. 22. Energy reduction w.r.t. the GPU baseline (training).

computationally intensive, and considering the energy and operating costs of data centers, custom architectures could help from both a performance and energy perspective. But web services are only the most visible applications. CNNs are being used for handwritten digits recognition [28] with applications in banking and post-offices, Dahl et al. [9] have recently won a pharmaceutical contest using DNN algorithms. More generally, such algorithms might take a central role in the so-called big data application domain. While CNNs and DNNs keep evolving, we note for instance that the first CNN design [35] still achieves very good results compared to the latest instantiations on benchmarks such as the MNIST handwritten digits [35], with a recognition error rate of 1.7 percent (1998) versus 0.23 percent for the best algorithm by Ciresan et al. [6] (2012).

So, even though there is an inherent risk in freezing algorithms in hardware, (1) hardware can rapidly evolve with machine-learning progress, much like it currently (and rapidly) evolves with technology progress, (2) what machine-learning researchers rightfully view as significant accuracy progress from their perspective (e.g., improving accuracy by 1 or 2 percent can be very difficult) may not be so significant from an end-user perspective, so that hardware needs not implement and follow each and every evolution, (3) end users are already accustomed to the notion of software libraries, and they can always choose between a rigid but very fast "hardware library" and a slow but more flexible CPU/GPU implementation.

Custom Accelerators. Due to the end of Dennard scaling and the notion of Dark Silicon [15], [42], architecture customization is increasingly viewed as one of the most promising paths forward. So far, the emphasis has been especially on custom *accelerators*, i.e., custom tiles within a heterogeneous multi-cores. Accelerators for video compression [21], image convolutions [44], libraries or specific workloads [17], [49] have been proposed.

Closer to the target algorithms of this paper, a number of studies have recently advocated the notion of neural network accelerators, either to approximate any function of a program [16], for signal-processing tasks [2] or specifically for machine-learning tasks [5], [22], [23], [47].

Large-Scale Custom Architectures. There are few examples of custom architectures targeting large-scale neural networks. The closest examples are the following. Schemmel et al. [46] propose a wafer-scale design capable of implementing thousands of neurons and millions of synapses. Khan et al. [30] propose the SpiNNaker system, which is a multi-chip supercomputer where each node contains 20+ ARM9 cores linked by an asynchronous network; the planned target is a million-core machine capable of modeling a billion neurons. [39] Finally, the IBM Cognitive Chip [59] is a functional chip capable of implementing 1M neurons and 256M synapses in 4.3 cm^2 at 28 nm. However, the common point between these different architectures is that their goal is the emulation of *biological neurons*, not machine-learning tasks, even though some of them have demonstrated machine-learning capabilities on simple tasks. Moreover, the neurons they implement are inspired from biology, i.e., spiking neurons, they do not implement the CNNs and DNNs which are the focus of our architecture. Majumdar et al. [37] investigate a parallel architecture for various machine-learning algorithms, including, but not only, neural networks; unlike our architecture, they have an off-chip banked memory, and they introduce memory banks close to PEs (similar to those found in GPUs) for caching purposes. Finally, beyond neural networks and machine-learning tasks, other large-scale custom architectures have been proposed, such as the recently proposed Anton 2 [60], for molecular dynamics simulation.

9 Conclusions and Future Work

In this article, we investigate a custom multi-chip architecture for state-of-the-art machine-learning algorithms (CNNs and DNNs), motivated by the increasingly central role of such algorithms in large-scale deployments of sophisticated services for consumers and industry. On both GPUs and recently proposed accelerators, such algorithms exhibit good speedups and area savings respectively, but they remain largely bandwidth-limited. We show that it is possible to design a multi-chip architecture which can outperform a single GPU by up to $656.63\times$ and reduce energy by up to $184.05\times$ using 64 nodes with electrical interconnects. Each node has an area of 67.73 mm^2 at 28 nm. By replacing electrical interconnects with optical interconnects, the 64-node architecture can achieve additional speedup by up to $2.20\times$ for classifier layers and $1.13\times$ on average, and reduce energy by $1.16\times$ on average.

We plan to improve this architecture along several directions: increasing the clock frequency of the NFU, investigating more flexible control in the form of a simple VLIW core per node and the associated toolchain. Moreover, we plan to investigate replacing eDRAM with high-bandwidth memory. However, it is not a straightforward task. To feed data to NFU, a DaDianNao node can read out 65,536 bit from eDRAM per cycle. Such a high bandwidth exceeds the capability of the current high-bandwidth memory technologies as HMC and HBM. This gap may be bridged with a novel architecture.

Acknowledgments

This work is partially supported by the NSF of China (under Grants 61133004, 61303158, 61432016, 61472396, 61473275, 61522211, 61532016, 61521092), the 973 Program of China (under Grant 2015CB358800), the Strategic Priority Research Program of the CAS (under Grants XDA06010403, XDB02040009), the International Collaboration Key Program of the CAS (under Grant 171111KYSB20130002), and the 10,000 talent program. Yunji Chen is the corresponding author.

References

[1] Cacti 5.3. [Online]. Available: http://quid.hpl.hp.com:9081/cacti/

[2] B. Belhadj, A. Joubert, Z. Li, R. Heliot, and O. Temam, "Continuous real-world inputs can open up alternative accelerator designs," in *Proc. 40th Annu. Int. Symp. Comput. Archit.*, 2013, pp. 1–12.

[3] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proc. 17th Int. Conf. Parallel Archit. Compilation Techn.*, 2008, pp. 72–81.

[4] D. Chen, et al., "The IBM Blue Gene/Q interconnection fabric," in *IEEE Micro*, vol. 32, no. 1, pp. 32–43, Jan./Feb. 2012.

[5] T. Chen, et al., "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proc. 19th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2014, pp. 269–284.
[6] D. Cirean, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proc. Int. Conf. Pattern Recognition*, 2012, pp. 3642–3649.
[7] D. Ciresan, U. Meier, and J. Masci, "Flexible, high performance convolutional neural networks for image classification," in *Proc. Int. Joint Conf. Artif. Intell.*, 2011, pp. 1237–1242.
[8] A. Coates, B. Huval, T. Wang, D. J. Wu, and A. Y. Ng, "Deep learning with COTS HPC systems," in *Proc. 30th Int. Conf. Mach. Learn.*, 2013, pp. 1337–1345.
[9] G. Dahl, T. Sainath, and G. Hinton, "Improving deep neural networks for lvcsr using rectified linear units and dropout," in *IEEE Int. Conf. Acoust. Speech Signal Process.*, 2013, pp. 8609–8613.
[10] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Mateo, CA, USA: Morgan Kaufmann 2003.
[11] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, et al. , "Large scale distributed deep networks,"" in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2012.
[12] M. M. Deneroff, D. E. Shaw, R. O. Dror, J. S. Kuskin, R. H. Larson, J. K. Salmon, and C. Young, "A specialized ASIC for molecular dynamics," in *Proc. Hot Chips*, 2008.
[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *IEEE Conf. Comput. Vision Pattern Recog.*, 2009, pp. 248–255.
[14] P. Dubey, "Recognition, mining and synthesis moves computers to the era of tera," *Technology@Intel Magazine*, vol. 9, no. 2, pp. 1–10, 2005.
[15] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. 38th Int. Symp. Comput. Archit.*, 2011, pp. 365–376.
[16] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purposed approximate programs," in *Proc. Int. Symp. Microarchitecture*, no. 3, pp. 1–6, 2012.
[17] K. Fan, M. Kudlur, G. S. Dasika, and S. A. Mahlke, "Bridging the computation gap between programmable processors and hardwired accelerators," in *Proc. IEEE 15th Int. Symp. High Perform. Comput. Archit.*, 2009, pp. 313–322.
[18] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "NeuFlow: A runtime reconfigurable dataflow processor for vision," in *Proc. CVPR Workshop*, pp. 109–116, 2011.
[19] D. A. Ferrucci, "Introduction to this is Watson," *IBM J. Res. Develop.*, vol. 56, pp. 1:1–1:15, 2012.
[20] R. Hadsell, et al., "Learning long-range vision for autonomous off-road driving," *J. Field Robotics*, vol. 26, pp. 120–144, 2009.
[21] R. Hameed, et al., "Understanding sources of inefficiency in general-purpose chips," in *Proc. 37th Annu. Int. Symp. Comput. Archit.*, 2010, pp. 37–47.
[22] A. Hashmi, H. Berry, O. Temam, and M. Lipasti, "Automatic abstraction and fault tolerance in cortical microarchitectures," in *Proc. 38th Annu. Int. Symp. Comput. Archit.*, 2011, pp. 1–10.
[23] A. Hashmi, A. Nere, J. J. Thomas, and M. Lipasti, "A case for neuromorphic ISAs," in *Proc. 16th Int. Conf. Archit. Support Program. Languages Oper. Syst.*, 2011, pp. 145–158.
[24] S.-N. Hong and G. Caire, "Compute-and-forward strategies for cooperative distributed antenna systems," in *IEEE Trans. Inf. Theory*, vol. 59, no. 9, pp. 5227–5243, Sep. 2013.
[25] K. Huang, et al., "A high-performance, high-density 28nm eDRAM technology with high-K/metal-gate," in *Proc. IEEE Int. Electron Devices Meeting*, 2011, pp. 24.7.1–24.7.4.
[26] L. Huang, S. Ma, L. Shen, Z. Wang, and N. Xiao, "Low-cost binary128 floating-point FMA unit design with SIMD support," *IEEE Trans. Comput.*, vol. 61, no. 5, pp. 745–751, May 2012.
[27] P. Huang, X. He, J. Gao and L. Deng, "Learning deep structured semantic models for web search using clickthrough data," in *Proc. Int. Conf. Inf. Knowl. Manag.*, 2013.
[28] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *Proc. 12th IEEE Int. Conf. Comput. Vis.*, 2009, pp. 2146–2153.
[29] A. Kahng, B. Li, L.-S. Peh, and K. Samadi, "Orion 2.0: A power-area simulator for interconnection networks," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 20, no. 1, pp. 191–196, Jan. 2012.
[30] M. M. Khan, et al., "SpiNNaker: Mapping neural networks onto a massively-parallel chip multiprocessor," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, 2008, pp. 2849–2856.
[31] A. Krizhevsky. [Online]. Available: https://code.google.com/p/cuda-convnet/
[32] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Advances Neural Inf. Process. Syst.*, 2012, pp. 1–9.
[33] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Int. Conf. Mach. Learn.*, 2007, pp. 473–480.
[34] Q. V. Le, et al., "Building high-level features using large scale unsupervised learning," in *Int. Conf. Mach. Learn.*, 2012, pp. 81–88.
[35] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[36] N. Maeda, S. Komatsu, M. Morimoto, and Y. Shimazaki, "A 0.41 A standby leakage 32 Kb embedded SRAM with low-voltage resume-standby utilizing all digital current comparator in 28 nm HKMG CMOS," in *Proc. Int. Symp. VLSI Circuits*, 2012.
[37] A. Majumdar, S. Cadambi, M. Becchi, S. T. Chakradhar, and H. P. Graf, "A massively parallel, energy efficient programmable accelerator for learning and classification," *ACM Trans. Archit. Code Optimization*, vol. 9, no. 1, pp. 1–30, Mar. 2012.
[38] R. E. Matick and S. E. Schuster, "Logic-based eDRAM: Origins and rationale for use," *IBM J. Res. Develop.*, vol. 49, no. 1, pp. 145–165, Jan. 2005.
[39] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45 pJ per spike in 45 nm," in *Proc. IEEE Custom Integr. Circuits Conf.*, 2011, pp. 1–4.
[40] Micron. Ddr3 sdram rdimm datasheet. [Online]. Available: http://www.micron.com/~/media/documents/products/data%20sheet/modules/parity_rdimm/jsf18c1 gx72pdz.pdf
[41] V. Mnih and G. Hinton, "Learning to label aerial images from noisy data," in *Proc. 29th Int. Conf. Mach. Learn.*, 2012, pp. 567–574.
[42] N. Goulding-Hotta, J. Sampson, G. Venkatesh, S. Garcia, J. Auricchio, P.-C. Huang, M. Arora, S. Nath, V. Bhatt, J. Babb, S. Swanson, and M. B. Taylor, "The GreenDroid Mobile Application Processor: An Architecture for Silicon's Dark Future," *IEEE Micro*, vol. 31, no. 2, pp. 86–95, 2011
[43] NVIDIA, "Tesla K20X GPU Accelerator Board Specification," *NVIDIA*, Santa Clara, CA, USA, Tech. Rep., Nov., 2012.
[44] W. Qadeer, R. Hameed, O. Shacham, P. Venkatesan, C. Kozyrakis, and M. A. Horowitz, "Convolution engine: Balancing efficiency & flexibility in specialized computing," in *Proc. 40th Annu. Int. Symp. Comput. Archit.*, 2013, pp. 24–35.
[45] R. Salakhutdinov and G. Hinton, "An efficient learning procedure for deep Boltzmann machines," *Neural Comput.*, vol. 24, no. 8. pp. 1967–2006, 2012.
[46] J. Schemmel, J. Fieres, and K. Meier, "Wafer-scale integration of analog neural networks," in *Proc. Int. Joint Conf. Neural Netw.*, pp. 431–438, 2008.
[47] O. Temam, "A defect-tolerant accelerator for emerging high-performance applications," in *Proc. Int. Symp. Comput. Archit.*, 2012, pp. 356–367.
[48] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," in *Deep Learn. Unsupervised Feature Learn. Workshop*, 2011.
[49] G. Venkatesh, J. Sampson, N. Goulding-hotta, S. K. Venkata, M. B. Taylor, and S. Swanson, "QsCORES : Trading dark silicon for scalable energy efficiency with quasi-specific cores categories and subject descriptors," in *Proc. Int. Symp. Microarchitecture*, 2011.
[50] G. Wang, "Scaling deep trench based eDRAM on SOI to 32nm and beyond," in *Proc. IEEE Int. Electron Devices Meet.*, 2009, pp. 1–4.
[51] Y. Chen, M. Kibune, A. Toda, A. Hayakawa, T. Akiyama, S. Sekiguchi, et al., "A 25 Gb/s hybrid integrated silicon photonic transceiver in 28 nm CMOS and SOI," in *IEEE Int. Solid-State Circuits Conf.*, 2015.
[52] Y. Chen, et al., "Dadiannao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2014, pp. 609–622.
[53] M. Cignoli, "A 1310 nm 3D-integrated silicon photonics Mach-Zehnder-based transmitter with 275 mW multistage CMOS driver achieving 6 dB extinction ratio at 25 Gb/s," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2015, pp. 1–3.
[54] N. Ophir, C. Mineo, D. Mountain, and K. Bergman, "Silicon photonic microring links for high-bandwidth-density, low-power chip I/O," *IEEE Micro*, vol. 33, no. 1. pp. 54–67, Jan./Feb. 2013.

[55] S. Rumley, D. Nikolova, R. Hendry, Q. Li, D. Calhoun, and K. Bergman, "Silicon photonics for exascale systems," *J. Lightw. Technol.*, vol. 33, no. 3. pp. 547–562, 2015.
[56] M. Rakowski, et al., "A 420Gb/s WDM ring-based hybrid CMOS silicon photonics transceiver," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2015 pp. 408–409.
[57] Y. A. Vlasov, "Silicon CMOS-integrated nano-photonics for computer and data communications beyond 100G," *IEEE Commun. Mag.*, vol. 50, no. 2, pp. s67–s72, 2012.
[58] I. A. Young, et al., "Optical I/O technology for tera-scale computing," *IEEE J. Solid-State Circuits*, vol. 45, no. 1. pp. 235–248, Jan. 2010.
[59] F. Akopyan, et al., "TrueNorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Trans. Comput.-Aided Des. of Integr. Circuits Syst.*, vol. 34, no. 10, pp. 1537–1557, Oct. 2015.
[60] D. E. Shaw, "Anton 2: Raising the bar for performance and programmability in a special-purpose molecular dynamics supercomputer," in *Proc. Int. Conf. High Perform. Comput., Netw. Storage Anal.*, 2014, pp. 41–53.
[61] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. 32nd Int. Conf. on Mach. Learning*, 2015, pp. 1–10.
[62] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-L. Lu, "Reducing cache power with low-cost, multi-bit error-correcting codes," in *Proc. 37th Annu. Int. Symp. Comput. Archit.*, 2010, pp. 83–93.
[63] K. Padmaraju, J. Chan, L. Chen, M. Lipson, and K. Bergman, "Thermal stabilization of a microring modulator using feedback control," *Optics Express*, vol. 20, no. 27. pp. 27999–28008, 2012.
[64] K. Padmaraju, D. F. Logan, X.-L. Zhu, J. J. Ackert, A. P. Knights, and K. Bergman, "Integrated thermal stabilization of a microring modulator," in *Proc. Opt. Fiber Commun. Conf. Expo. Nat. Fiber Optic Engineers Conf.*, 2013, pp. 14342–14350.
[65] W. A. Zortman, D. Trotter, and M. R. Watts, "Silicon photonics manufacturing," *Opt. Exp.*, vol. 18, no. 23, pp. 23598–23607, 2010.
[66] A. V. Krishnamoorthy, et al., "Exploiting CMOS manufacturing to reduce tuning requirements for resonant optical devices," *IEEE Photon. J.*, vol. 3 no. 3, pp. 567–579, Jun. 2011.

Tao Luo is an assistant professor of the Institute of Computing Technology, Chinese Academy of Sciences.

Shaoli Liu is an assistant professor of the Institute of Computing Technology, Chinese Academy of Sciences.

Ling Li is an associate professor of the Institute of Automation, Chinese Academy of Sciences.

Yuqing Wang is a PhD student of the University of Science and Technology of China.

Shijin Zhang is a PhD student of the Institute of Computing Technology, Chinese Academy of Sciences.

Tianshi Chen is an associate professor of the Institute of Computing Technology, Chinese Academy of Sciences.

Zhiwei Xu is a full professor of the Institute of Computing Technology, Chinese Academy of Sciences.

Olivier Temam is a full professor of Inria, France.

Yunji Chen is a full professor of the Institute of Computing Technology, Chinese Academy of Sciences.