DOI:10.1145/2996864

DianNao Family: Energy-Efficient Hardware Accelerators for Machine Learning

By Yunji Chen, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam

Abstract

Machine Learning (ML) tasks are becoming pervasive in a broad range of applications, and in a broad range of systems (from embedded systems to data centers). As computer architectures evolve toward heterogeneous multi-cores composed of a mix of cores and hardware accelerators, designing hardware accelerators for ML techniques can simultaneously achieve high efficiency and broad application scope.

While efficient computational primitives are important for a hardware accelerator, inefficient memory transfers can potentially void the throughput, energy, or cost advantages of accelerators, that is, an Amdahl's law effect, and thus, they should become a first-order concern, just like in processors, rather than an element factored in accelerator design on a second step. In this article, we introduce a series of hardware accelerators (i.e., the DianNao family) designed for ML (especially neural networks), with a special emphasis on the impact of memory on accelerator design, performance, and energy. We show that, on a number of representative neural network layers, it is possible to achieve a speedup of 450.65x over a GPU, and reduce the energy by 150.31x on average for a 64-chip DaDianNao system (a member of the DianNao family).

1. INTRODUCTION

As architectures evolve towards heterogeneous multi-cores composed of a mix of cores and accelerators, designing hardware accelerators which realize the best possible tradeoff between flexibility and efficiency is becoming a prominent issue. The first question is for which category of applications one should primarily design accelerators? Together with the architecture trend towards accelerators, a second simultaneous and significant trend in high-performance and embedded applications is developing: many of the emerging high-performance and embedded applications, from image/video/audio recognition to automatic translation, business analytics, and robotics rely on *machine learning techniques*. This trend in application comes together with a third trend in machine learning (ML) where a small number of techniques, based on neural networks (especially *deep learning techniques*[16, 26]), have been proved in the past few years to be state-of-the-art across a broad range of applications. As a result, there is a unique opportunity to design accelerators having significant application scope as well as high performance and efficiency.[4]

Currently, ML workloads are mostly executed on multicores using SIMD,[44] on GPUs,[7] or on FPGAs.[2] However, the aforementioned trends have already been identified by researchers who have proposed accelerators implementing, for example, Convolutional Neural Networks (CNNs)[2] or Multi-Layer Perceptrons[43]; accelerators focusing on other domains, such as image processing, also propose efficient implementations of some of the computational primitives used by machine-learning techniques, such as convolutions.[37] There are also ASIC implementations of ML techniques, such as Support Vector Machine and CNNs. However, all these works have first, and successfully, focused on efficiently implementing the computational primitives but they either voluntarily ignore memory transfers for the sake of simplicity,[37, 43] or they directly plug their computational accelerator to memory via a more or less sophisticated DMA.[2, 12, 19]

While efficient implementation of computational primitives is a first and important step with promising results, inefficient memory transfers can potentially void the throughput, energy, or cost advantages of accelerators, that is, an Amdahl's law effect, and thus, they should become a first-order concern, just like in processors, rather than an element factored in accelerator design on a second step. Unlike in processors though, one can factor in the specific nature of memory transfers in target algorithms, just like it is done for accelerating computations. This is especially important in the domain of ML where there is a clear trend towards scaling up the size of learning models in order to achieve better accuracy and more functionality.[16, 24]

In this article, we introduce a series of hardware accelerators designed for ML (especially neural networks), including DianNao, DaDianNao, ShiDianNao, and PuDianNao as listed in Table 1. We focus our study on memory usage, and we investigate the accelerator architecture to minimize memory transfers and to perform them as efficiently as possible.

2. DIANNAO: A NEURAL NETWORK ACCELERATOR

Neural network techniques have been proved in the past few years to be state-of-the-art across a broad range of applications. DianNao is the first member of the DianNao accelerator family, which accommodates state-of-the-art neural

The original version of this paper is entitled "DianNao: A Small-Footprint, High-Throughput Accelerator for Ubiquitous Machine Learning" and was published in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* 49, 4 (March 2014), ACM, New York, NY, 269–284.

Table 1. Accelerators in the DianNao family.

Name	Process (nm)	Peak performance (GOP/s)	Peak power (W)	Area (mm²)	Applications
DianNao	65	452	0.485	3.02	Neural networks
DaDianNao	28	5585	15.97	67.73	Neural networks
ShiDianNao	65	194	0.32	4.86	Convolutional neural networks
PuDianNao	65	1056	0.596	3.51	Seven representative machine learning techniques

network techniques (e.g., deep learning[a]), and inherits the broad application scope of neural networks.

2.1. Architecture

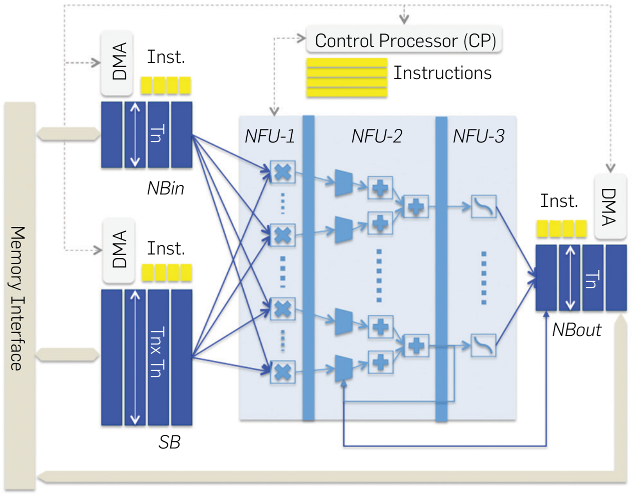
DianNao has the following components: an input buffer for input neurons (NBin), an output buffer for output neurons (NBout), and a third buffer for synaptic weights (SB), connected to a computational block (performing both synapses and neurons computations) which we call the Neural Functional Unit (NFU), and the control logic (CP), see Figure 1.

Neural Functional Unit (NFU). The NFU implements a functional block of T_i inputs/synapses and T_n output neurons, which can be time-shared by different algorithmic blocks of neurons. Depending on the layer type, computations at the NFU can be decomposed in either two or three stages. For classifier and convolutional layers: multiplication of synapses × inputs, additions of all multiplications, sigmoid. The nature of the last stage (sigmoid or another nonlinear function) can vary. For pooling layers, there is no multiplication (no synapse), and the pooling operations can be average or max. Note that the adders have multiple inputs, they are in fact *adder trees*, see Figure 1; the second stage also contains shifters and max operators for pooling layers. In the NFU, the sigmoid function (for classifier and convolutional layers) can be efficiently implemented using piecewise linear interpolation ($f(x) = a_i \times x + b_i, x \in [x_i; x_{i+1}]$) with negligible loss of accuracy (16 segments are sufficient).[22]

On-chip Storage. The on-chip storage structures of DianNao can be construed as modified buffers of scratchpads. While a cache is an excellent storage structure for a general-purpose processor, it is a sub-optimal way to exploit reuse because of the cache access overhead (tag check, associativity, line size, speculative read, etc.) and cache conflicts. The efficient alternative, scratchpad, is used in VLIW processors but it is known to be very difficult to compile for. However a scratchpad in a dedicated accelerator realizes the best of both worlds: efficient storage, and both efficient and easy exploitation of locality because only a few algorithms have to be manually adapted.

We split on-chip storage into three structures (NBin, NBout, and SB), because there are three type of data (input neurons, output neurons and synapses) with different characteristics (e.g., read width and reuse distance). The first benefit of splitting structures is to tailor the SRAMs to the appropriate read/write width, and the second benefit of splitting storage structures is to avoid conflicts, as would occur in a cache. Moreover, we implement three DMAs to exploit spatial locality of data, one for each buffer (two load DMAs for inputs, one store DMA for outputs).

Figure 1. Accelerator architecture of DianNao.



2.2. Loop tiling

DianNao leverages loop tiling to minimize memory accesses, and thus efficiently accommodates large neural networks. For the sake of brevity, here we only discuss a classifier layer[b] that has N_n output neurons, fully connected to N_i inputs. We present in Figure 2 the original code of the classifier, as well as the tiled code that maps the classifier layer to DianNao.

In the tiled code, the loops *ii* and *nn* reflects the aforementioned fact that the NFU is a functional block of T_i inputs/synapses and T_n output neurons. On the other hand, input neurons are reused for each output neuron, but since the number of input neurons can range anywhere between a few tens to hundreds of thousands, they will often not fit in the NBin size of DianNao. Therefore, we further tile loop *ii* (input neurons) with tile factor T_{ii}. A typical tradeoff of tiling is that improving one reference (here *neuron*[*i*] for input neurons) increases the reuse distance of another reference (*sum*[*n*] for partial sums of output neurons), so we need to tile for the second reference as well, hence loop *nnn* and the tile

[a] According to a recent review[25] written by LeCun, Bengio, and Hinton, *Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics.*

[b] Readers may refer to Ref.[5] for details of other layer types.

Figure 2. Pseudo-code for a classifier layer (top: original code; bottom: tiled code).

```
for (int n = 0; n < Nn; n++)
    sum[n] = 0;
for (int n = 0; n < Nn; n++) // output neurons
    for (int i = 0; i < Ni; i++) // input neurons
        sum[n] += synapse[n][i] * neuron[i];
for (int n = 0; n < Nn; n++)
    neuron[n] = sigmoid(sum[n]);
```

```
for (int nnn = 0; nnn < Nn; nnn += Tnn){ // tiling for output neurons
    for (int iii = 0; iii < Ni; iii += Tii){ // tiling for input neurons
        for (int nn = nnn; nn < nnn + Tnn; nn += Tn){
            for (int n = nn; n < nn + Tn; n++)
                sum[n] = 0;
            for (int ii = iii; ii < iii + Tii; ii += Ti)
                for (int n = nn; n < nn + Tn; n++)
                    for (int i = ii; i < ii + Ti; i++)
                        sum[n] += synapse[n][i] * neuron[i];
            for (int n = nn; n < nn + Tn; n++)
                neuron[n] = sigmoid(sum[n]);
}   }   }
```

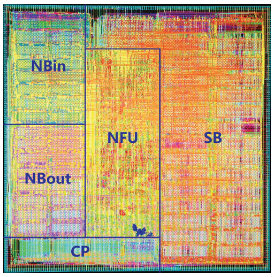
factor T_{nn} for output neurons partial sums. The layer memory behavior is now dominated by synapses. In a classifier layer, all synapses are usually unique, and thus there is no reuse within the layer. Overall, tiling drastically reduces the total memory bandwidth requirement of the classifier layer, and we observe a ∼50% reduction in our empirical study.[5]

2.3. Experimental observations

We implemented a custom cycle-accurate, bit-accurate C++ simulator of the accelerator. This simulator is also used to measure time in number of cycles. It is plugged to a main memory model allowing a bandwidth of up to 250 GB/s. We also implemented a Verilog version of the accelerator, which uses $T_n = T_i = 16$ (16 hardware neurons with 16 synapses each), so that the design contains 256 16-bit truncated multipliers (for classifier and convolutional layers), 16 adder trees of 15 adders each (for the same layers, plus pooling layer if average is used), as well as a 16-input shifter and max (for pooling layers), and 16 16-bit truncated multipliers plus 16 adders (for classifier and convolutional layers, and optionally for pooling layers). For classifier and convolutional layers, multipliers and adder trees are active every cycle, achieving $256 + 16 \times 15 = 496$ fixed-point operations every cycle; at 0.98 GHz, this amounts to 452 GOP/s (Giga fixed-point OPerations per second). We have done the synthesis and layout of the accelerator at 65 nm using Synopsys tools, see Figure 3.

We implemented an SIMD baseline using the GEM5+McPAT[27] combination. We use a 4-issue superscalar x86 core with a 128-bit (8 × 16-bit) SIMD unit (SSE/SSE2), clocked at 2 GHz. Following a default setting of GEM5, the core has a 192-entry reorder buffer and a 64-entry load/store queue. The L1 data (and instruction) cache is 32 KB and the L2 cache is 2 MB; both caches are 8-way associative and use a 64-byte line; these cache characteristics correspond to those of the Intel Core i7. In addition, we also employ NVIDIA K20M (28 nm process, 5 GB GDDR5, 3.52 TFlops peak) the GPU baseline.

Figure 3. Snapshot of DianNao's layout.



We employed several representative layer settings as benchmarks of our experiments, see Table 2. We report in Figure 4 the speedups of DianNao over SIMD and GPU. We observe that DianNao significantly outperforms SIMD, and the average speedup is **117.87**x. The main reasons are twofold. First, DianNao performs 496 16-bit operations every cycle for both classifier and convolutional layers, that is, 62x more $\left(\frac{496}{8}\right)$ than the peak performance of the SIMD baseline. Second, compared with the SIMD baseline without prefetcher, DianNao has better latency tolerance due to an appropriate combination of preloading and reuse in NBin and SB buffers.

In Figure 5, we provide the energy reductions of DianNao over SIMD and GPU. We observe that DianNao consumes 21.08x less energy than SIMD on average. This number is actually more than an order of magnitude smaller than previously reported energy ratios between processors and accelerators; for instance Hameed et al.[15] report an energy ratio of about 500x, and 974x has been reported for a small Multi-Layer Perceptron.[43] The smaller ratio is largely due to the energy spent in memory accesses, which was voluntarily not factored in the two aforementioned studies. Like in these two accelerators and others, the energy cost of computations has been considerably reduced by a combination of more efficient computational operators (especially a massive number of small

Table 2. Benchmark layers for DianNao.

Layer	N_x	N_y	K_x	K_y	N_i	N_o	Description
CONV1	500	375	9	9	32	48	Street scene parsing (CNN)[12] (e.g., identifying "building," "vehicle," etc.)
POOL1	492	367	2	2	12	–	
CLASS1	–	–	–	–	960	20	
CONV2*	200	200	18	18	8	8	Detection of faces in YouTube videos (DNN),[24] largest NN to date (Google)
CONV3	32	32	4	4	108	200	Traffic sign identification for car navigation (CNN)[39]
POOL3	32	32	4	4	100	–	
CLASS3	–	–	–	–	200	100	
CONV4	32	32	7	7	16	512	Google Street View house numbers (CNN)[38]
CONV5*	256	256	11	11	256	384	Multi-Object recognition in natural images (DNN),[16] winner 2012 ImageNet competition
POOL5	256	256	2	2	256	–	

CONV, convolutional; POOL, pooling; CLASS, classifier.
$N_x \times N_y$ is the size of an input feature map, $K_x \times K_y$ is the size of a convolutional/pooling window, and N_i and N_o are numbers of input and output feature maps, respectively.
*Indicates private kernels.

Figure 4. Speedups of DianNao over SIMD.

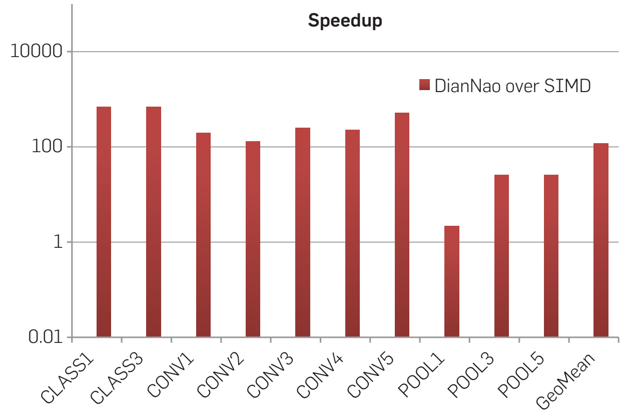
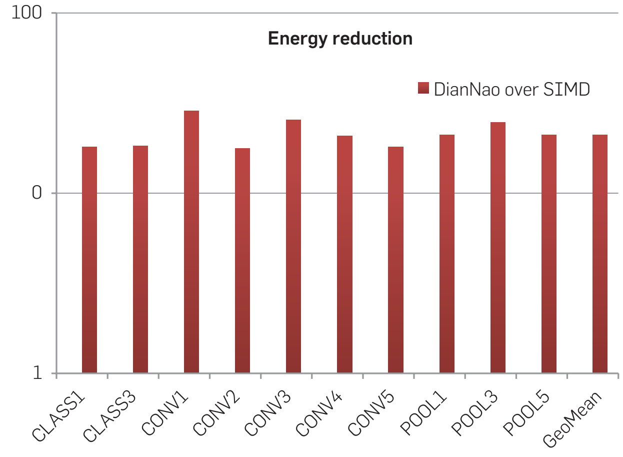


Figure 5. Energy reductions of DianNao over SIMD.



16-bit fixed-point truncated multipliers in our case), and small custom storage located close to the operators (64-entry NBin, NBout, SB, and the NFU-2 registers). As a result, there is now an Amdahl's law effect for energy, where any further improvement can only be achieved by bringing down the energy cost of main memory accesses. We tried to artificially set the energy cost of the main memory accesses in both the SIMD and accelerator to 0, and we observed that the average energy reduction of the accelerator increases by more than one order of magnitude, in line with previous results.

We have also explored different parameter settings for DianNao in our experimental study, where we altered the size of NFU as well as sizes of NBin, NBout, and SB. For example, we evaluated a design with $T_n = 8$ (i.e., the NFU only has 8 hardware neurons), and thus 64 multipliers in NFU-1. We correspondingly reduced widths of all buffers to fit the NFU. As a result, the total area of that design is 0.85 mm^2, which is 3.59x smaller than for the case of $T_n = 16$.

3. DADIANNAO: A MACHINE LEARNING SUPERCOMPUTER

In the ML community, there is a significant trend towards increasingly large neural networks. The recent work of Krizhevsky et al.[20] achieved promising accuracy on the ImageNet database[8] with "only" 60 million parameters. There are recent examples of a 1-billion parameter neural network.[24] Although DianNao can execute neural networks at different scales, it has to store the values of neurons and synapses in main memory when accommodating large neural networks. Frequent main memory accesses greatly limit the performance and energy-efficiency of DianNao.

While 1 billion parameters or more may come across as a large number from a ML 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 ML 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*.

In large neural networks, 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, and classifier layers (which are usually fully connected, and thus have lots of synapses). In DaDianNao, 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 (Swanson *et al.* adopted a similar strategy dataflow computing[41]); (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 of DaDianNao is a set of nodes, one per chip, all identical, arranged in a classic mesh topology. Each node contains significant storage, especially for synapses, and neural computational units (the classic pipeline of multipliers, adder trees and nonlinear functions implemented via linear interpolation), which we still call NFU for the sake of consistency with the DianNao accelerator.[5] Here we briefly introduce some key characteristics of the node architecture.

Tile-based organization. Putting all functional units (e.g., adders and multipliers) together in a single computational block (NFU) is an acceptable design choice when the NFU has a moderate area, which is the case of DianNao. However, if we significantly scale up the NFU, the data movement between NFU and on-chip storage will require a very high internal bandwidth (even if we split the on-chip storage), resulting in unacceptably large wiring overheads.[6] To address this issue, we adopt a tile-based organization in each node, see Figure 6. Each tile contains an NFU and four RAM banks to store synapses between neurons.

When accommodating a neural network layer, 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). 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 RAM 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 RAM. 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 RAM bank.

Storage. In some of the largest known neural networks, the storage size required by a layer typically 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,[30] while an eDRAM memory of the same size and at the same technology node requires 7.27 mm^2,[45] that is, a 2.85x higher storage density. In each DaDianNao node, we have implemented 16 tiles, all using eDRAMs as their on-chip storage. Each tile has four eDRAM banks (see Figure 6), each bank contains 1024 rows of 4096 bits, thus the total eDRAM capacity in one tile is $4 \times 1024 \times 4096 = 2$ MB. The central eDRAM in each node (see Figure 6) has a size of 4 MB. Therefore, the total node eDRAM capacity is thus $16 \times 2 + 4 = 36$ MB.

Interconnect. Because neurons are the only values transferred, and because these values are heavily reused within each node, the amount of communications, while significant, is usually not a bottleneck. 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 multinode DaDianNao system uses a simple 2D mesh topology, thus each chip must connect to four neighbors via four HT2.0 IP blocks.

We implemented a custom cycle-accurate bit-accurate C++ simulator for the performance evaluation of the DaDianNao architecture. We also implemented a Verilog version of the DaDianNao node, and have done the synthesis and layout at a 28 nm process (see Figure 7 for the node layout). The node (chip), clocked at 606 MHz, consumes an area of 67.73 mm^2, and has the peak performance 5585 GOP/s. We evaluated an architecture with up to 64 chips. On a sample of the largest existing neural network layers, we show that a single DaDianNao node achieves a speedup of **21.38**x over the NVIDIA K20M GPU and reduces energy by **330.56**x on

Figure 6. DaDianNao architecture: tile-based organization of a node (left) and tile architecture (right).

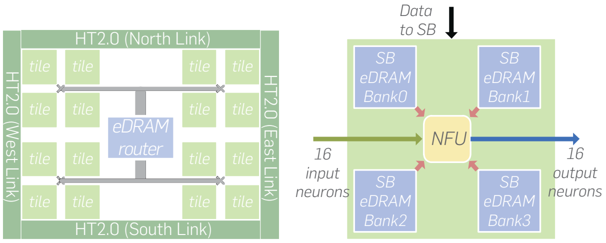
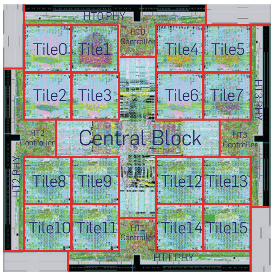


Figure 7. Snapshot of DaDianNao's node layout.



average; a 64-node system achieves a speedup of **450.65**x over the NVIDIA K20M GPU and reduces energy by **150.31**x on average.[6]

4. SHIDIANNAO: A LOW-POWER ACCELERATOR FOR CONVOLUTIONAL NEURAL NETWORK

DaDianNao targets at high-performance ML applications, and integrates eDRAMs in each node to avoid main memory accesses. In fact, the same principle is also applicable to embedded systems, where energy consumption is a critical dimension that must be taken into account. In a recent study, we focused on image applications in embedded systems, and designed a dedicated accelerator (ShiDianNao[9]) for a state-of-the-art deep learning technique called CNN.[26]

In a broad class of CNNs, it is assumed that each neuron (of a feature map) *shares* its weights with other neurons, making the total number of weights far smaller than in fully connected networks. For instance, a state-of-the-art CNN has 60 millions weights[21] versus up to 1 billion[24] or even 10 billions for state-of-the-art deep networks. This simple property can have profound implications for us: we know that the highest energy expense is related to memory behaviors, in particular main memory (DRAM) accesses, rather than computation.[40] Due to the small memory footprint of weights in CNNs, it is possible to store a whole CNN within a small on-chip SRAM next to the functional units, and as a result, there is no longer a need for DRAM memory accesses to fetch the CNN model (weights) in order to process each input.

The absence of DRAM accesses combined with a careful exploitation of the specific data access patterns within CNNs allows us to design the ShiDianNao accelerator which is 60× more energy efficient than the DianNao accelerator (c.f., Figure 8). We present a full design down to the layout at a 65 nm process, with an area of 4.86 mm^2 and a power of 320 mW, but still over 30x faster than NVIDIA K20M GPU. The detailed ShiDianNao architecture was presented at 42nd ACM/IEEE International Symposium on Computer Architecture (ISCA'15).[9]

5. PUDIANNAO: A POLYVALENT MACHINE LEARNING ACCELERATOR

While valid to many different ML tasks, DianNao, DaDianNao, and ShiDianNao can only accommodate neural networks. However, neural networks might not always be the best choice in every application scenario, even if regardless of their high computational complexity. For example, in the classification of linearly separable data, complex neural networks can easily become over-fitting, and perform worse than a linear classifier. In application domains such as financial quantitative trading, linear regression is more widely used than neural network due to the simplicity and interpretability of linear model.[3] The famous No-Free-Lunch theorem, though was developed under certain theoretical assumptions, is a good summary of the above situation: any learning technique cannot perform universally better than another learning technique.[46] In this case, it is a natural idea to further extend DianNao/DaDianNao to support a basket of diverse ML techniques, and the extended accelerator will have much broader application scope than its ancestors do.

PuDianNao is a hardware accelerator accommodating seven representative ML techniques, that is, *k*-means, *k*-NN, naive bayes, support vector machine, linear regression, classification tree, and deep neural network. PuDianNao consists of several Functional Units (FUs), three data buffers (HotBuf, ColdBuf, and OutputBuf), an instruction buffer (InstBuf), a control module, and a DMA, see Figure 9. The functional unit for machine learning (MLU) is designed to support several basic yet important computational primitives. As illustrated in Figure 10, the MLU is divided into 6 pipeline stages (Counter, Adder, Multiplier, Adder tree, Acc, and Misc), and different combinations of selected stages collaboratively compute primitives that are common in representative ML techniques, such as dot product, distance calculations, counting, sorting, nonlinear functions (e.g., sigmoid and tanh) and so on. In addition, there are some less common operations that

Figure 8. Snapshot of ShiDianNao's layout.

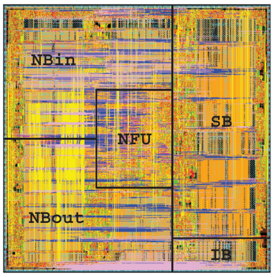


Figure 9. Accelerator architecture of PuDianNao.

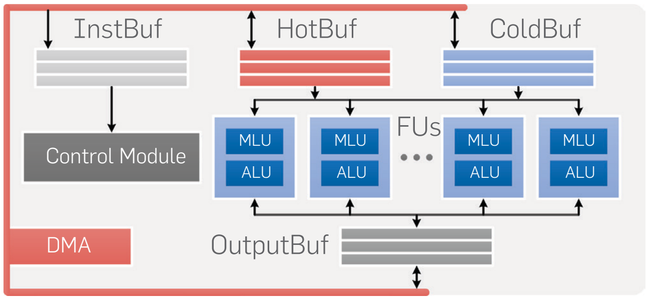
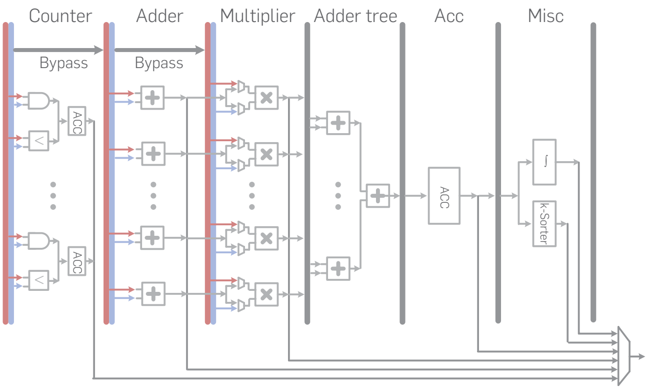


Figure 10. Implementation of Machine Learning Unit (MLU).



are not supported by the MLUs (e.g., division and conditional assignment), which will be supported by small Arithmetic Logic Units (ALUs).

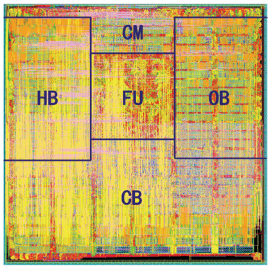
On the other hand, loop tiling can effectively exploit the data locality of ML techniques (as we have revealed in Section 2.2). For tiled versions of different ML techniques, we further observed that average reuse distances of variables often cluster into two or three classes.[28] Therefore, we put three separate on-chip data buffers in the PuDianNao accelerator: HotBuf, ColdBuf, and OutputBuf. HotBuf stores the input data which have short reuse distance, ColdBuf stores the input data with relative longer reuse distance, and OutputBuf stores output data or temporary results.

We implemented a cycle-accurate C simulator and a Verilog version of the accelerator, which integrates 16 MLUs (each has 49 adders and 17 multipliers), a 8 KB HotBuf (8 KB), a 16 KB ColdBuf and a 8 KB OutputBuf. PuDianNao can achieve a peak performance of $16 \times (49 + 17) \times 1 = 1056$ GOP/s at 1 GHz frequency, almost approaching the performance of a modern GPU. We have done the synthesis and layout of PuDianNao at a 65 nm process using Synopsys tools, see Figure 11 for the layout. On 13 critical phases of seven representative ML techniques,[28] the average speedups of PuDianNao over Intel Xeon E5-4620 SIMD CPU (32 nm process) and NVIDIA K20M GPU (28 nm process) are **21.29**x and **1.20**x, respectively. PuDianNao is significantly more energy-efficient than two general-purpose baselines, given that its power dissipation is only 0.596 W.

6. RELATED WORK

Due to the end of Dennard scaling and the notion of Dark Silicon,[10, 35] architecture customization is increasingly viewed as one of the most promising paths forward. So far, the emphasis has been especially on custom *accelerators*. There have been many successful studies, working on either approximation of program functions using ML,[11] or acceleration of ML itself. Yeh *et al*. designed a *k*-NN accelerator on FPGA.[47] Manolakos and Stamoulias designed two high-performance parallel array architectures for *k*-NN.[33, 40] There have also been dedicated accelerators for *k*-means[14, 18, 34] or support vector machine,[1, 36] due to their broad applications in industry. Majumdar *et al*. proposed an accelerator called MAPLE, which can accelerate matrix/vector and ranking operation used in five ML technique families (including neural network, support vector machine and *k*-means).[31, 32] Recent advances on deep learning[23] even triggers the rebirth of hardware neural network.[13, 29, 42] However, few previous investigations on ML accelerators can simultaneously address (a)*computational primitives* and (b)*locality properties* of (c)*diverse representative machine learning techniques*.

Figure 11. Snapshot of PuDianNao's layout.



7. CONCLUSION

In this article, we conduct an in-depth discussion on hardware accelerations of ML. Unlike previous studies that mainly focus on implementing major computational primitives of ML techniques, we also optimize memory structures of the accelerators to reduce/remove main memory accesses, which significantly improve the energy-efficiency of ML compared with systems built with general-purpose CPUs or GPUs.

We devoted DianNao, DaDianNao, and ShiDianNao to neural network (deep learning) techniques, in order to achieve the rare combination of efficiency (due to the small number of target techniques) and broad application scope. However, using large-scale neural networks might not always be a promising choice due to the well-known No-Free-Lunch Theorem,[46] as well as the distinct requirements in different application scenarios. Therefore, we also develop the PuDianNao accelerator which extends the basic DianNao architecture to support a basket of seven ML techniques.

Following the spirit of PuDianNao, we will further study a pervasive accelerator for ML in our future work, and the purpose is to energy-efficiently accommodate a very broad range of ML techniques. A comprehensive review on representative ML techniques can help to extract common computational primitives and locality properties behind the techniques. However, a straightforward hardware implementation of functional units and memory structures simply matching all extracted algorithmic characteristics could be expensive, because it integrates redundant components to resist significant diversity among ML techniques. This issue can probably be addressed by a *reconfigurable ASIC* accelerator, which supports dynamic reconfiguration of functional units and memory structures to adapt to diverse techniques. Such an accelerator only involves a moderate number of coarse-grained reconfigurable parameters, which is significantly more energy-efficient than Field Programmable Gate Array (FPGA) having millions of controlling parameters.

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 and XDB02040009), the International Collaboration Key Program of the CAS (under Grant 171111KYS-B20130002), the 10,000 talent program, a Google Faculty Research Award, and the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI).

References

1. Cadambi, S., Durdanovic, I., Jakkula, V., Sankaradass, M., Cosatto, E., Chakradhar, S., Graf, H.P. A massively parallel fpga-based coprocessor for support vector machines. In *17th IEEE Symposium on Field Programmable Custom Computing Machines, 2009. FCCM'09* (2009) IEEE, 115–122.
2. Chakradhar, S., Sankaradas, M., Jakkula, V., Cadambi, S. A dynamically configurable coprocessor for convolutional neural networks. In *International Symposium on Computer*

Architecture (Saint Malo, France, June 2010). ACM 38(3): 247–257.

3. Chan, E. *Algorithmic Trading: Winning Strategies and Their Rationale*. John Wiley & Sons, 2013.
4. Chen, T., Chen, Y., Duranton, M., Guo, Q., Hashmi, A., Lipasti, M., Nere, A., Qiu, S., Sebag, M., Temam, O. BenchNN: On the broad potential application scope of hardware neural network accelerators. In *International Symposium on Workload Characterization*, 2012.
5. Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., Temam, O. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, (March 2014). ACM 49(4): 269–284.
6. Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., Li, L., Chen, T., Xu, Z., Sun, N., Temam, O. Dadiannao: A machine-learning supercomputer. In *ACM/IEEE International Symposium on Microarchitecture (MICRO)* (December 2014). IEEE Computer Society, 609–622.
7. Coates, A., Huval, B., Wang, T., Wu, D.J., Ng, A.Y. Deep learning with cots HPC systems. In *International Conference on Machine Learning*, 2013: 1337–1345.
8. Deng, J. Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L. ImageNet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition* (CVPR) (2009). IEEE, 248–255.
9. Du, Z., Fasthuber, R., Chen, T., Ienne, P., Li, L., Luo, T., Feng, X., Chen, Y., Temam, O. Shidiannao: Shifting vision processing closer to the sensor. In *Proceedings of the 42nd ACM/IEEE International Symposium on Computer Architecture (ISCA'15)* (2015). ACM, 92–104.
10. Esmaeilzadeh, H., Blem, E., Amant, R.S., Sankaralingam, K., Burger, D. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th International Symposium on Computer Architecture (ISCA)* (June 2011). IEEE, 365–376.
11. Esmaeilzadeh, H., Sampson, A., Ceze, L., Burger, D. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture* (Dec 2012). IEEE Computer Society, 449–460.
12. Farabet, C., Martini, B., Corda, B., Akselrod, P., Culurciello, E., LeCun, Y. NeuFlow: A runtime reconfigurable dataflow processor for vision. In *CVPR Workshop* (June 2011). IEEE, 109–116.
13. Farabet, C., Martini, B., Corda, B., Akselrod, P., Culurciello, E., LeCun, Y. Neuflow: A runtime reconfigurable dataflow processor for vision. In *2011 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2011). IEEE, 109–116.
14. Frery, A., de Araujo, C., Alice, H., Cerqueira, J., Loureiro, J.A., de Lima, M.E., Oliveira, M., Horta, M., et al. Hyperspectral images clustering on reconfigurable hardware using the k-means algorithm. In *Proceedings of the 16th Symposium on Integrated Circuits and Systems Design, 2003. SBCCI 2003* (2003). IEEE, 99–104.
15. Hameed, R., Qadeer, W., Wachs, M., Azizi, O., Solomatnikov, A., Lee, B.C., Richardson, S., Kozyrakis, C., Horowitz, M. Understanding sources of inefficiency in general-purpose chips. In *International Symposium on Computer Architecture* (New York, New York, USA, 2010). ACM, 38(3): 37–47.
16. Hinton, G., Srivastava, N. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:* ..., 1–18, 2012.
17. Hussain, H.M., Benkrid, K., Seker, H., Erdogan, A.T. Fpga implementation of k-means algorithm for bioinformatics application: An accelerated approach to clustering microarray data. In *2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)* (2011). IEEE, 248–255.
18. Keckler, S. Life after Dennard and how I learned to love the Picojoule (keynote). In *International Symposium on Microarchitecture*, Keynote presentation, Sao Paolo, Dec. 2011.
19. Kim, J.Y., Kim, M., Lee, S., Oh, J., Kim, K., Yoo, H.-J.A. GOPS 496 mW real-time multi-object recognition processor with bio-inspired neural perception engine. *IEEE Journal of Solid-State Circuits 45*, 1 (Jan. 2010), 32–45.
20. Krizhevsky, A., Sutskever, I., Hinton, G. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (2012), 1–9.
21. Krizhevsky, A., Sutskever, I., Hinton, G. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (2012) 1–9.
22. Larkin, D., Kinane, A., O'Connor, N.E. Towards hardware acceleration of neuroevolution for multimedia processing applications on mobile devices. In *Neural Information Processing* (2006). Springer, Berlin Heidelberg, 1178–1188.
23. Le, Q.V. Building high-level features using large scale unsupervised learning. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2013). IEEE, 8595–8598.
24. Le, Q.V., Ranzato, M.A., Monga, R., Devin, M., Chen, K., Corrado, G.S., Dean, J., Ng, A.Y. Building high-level features using large scale unsupervised learning. In *International Conference on Machine Learning*, June 2012.
25. LeCun, Y., Bengio, Y., Hintion, G. Deep learning. *Nature* 521, 7553 (2015), 436–444.
26. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86* 11 (1998), 2278–2324.
27. Li, S., Ahn, J.H., Strong, R.D., Brockman, J.B., Tullsen, D.M., Jouppi, N.P. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42 (New York, NY, USA, 2009). ACM, 469–480.
28. Liu, D., Chen, T., Liu, S., Zhou, J., Zhou, S., Teman, O., Feng, X., Zhou, X., Chen, Y. Pudiannao: A polyvalent machine learning accelerator. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (2015). ACM, 369–381.
29. Maashri, A.A., Debole, M., Cotter, M., Chandramoorthy, N., Xiao, Y., Narayanan, V., Chakrabarti, C. Accelerating neuromorphic vision algorithms for recognition. In *Proceedings of the 49th Annual Design Automation Conference* (2012). ACM, 579–584.
30. Maeda, N., Komatsu, S., Morimoto, M., Shimazaki, Y. A 0.41µa standby leakage 32 kb embedded SRAM with low-voltage resume-standby utilizing all digital current comparator in 28 nm hkmg CMOS. In *International Symposium on VLSI Circuits (VLSIC)*, 2012.
31. Majumdar, A., Cadambi, S., Becchi, M., Chakradhar, S.T., Graf, H.P. A massively parallel, energy efficient programmable accelerator for learning and classification. *ACM Trans. Arch. Code Optim. (TACO) 9*, 1 (2012), 6.
32. Majumdar, A., Cadambi, S., Chakradhar, S.T. An energy-efficient heterogeneous system for embedded learning and classification. *Embedded Systems Letters* 3, 1 (2011), 42–45.
33. Manolakos, E.S., Stamoulias, I. IP-cores design for the KNN classifier. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)* (2010). IEEE, 4133–4136.
34. Maruyama, T. Real-time k-means clustering for color images on reconfigurable hardware. In *18th International Conference on Pattern Recognition (ICPR)* (Aug 2006). IEEE, Volume 2, 816–819.
35. Muller, M. Dark silicon and the internet. In *EE Times "Designing with ARM" Virtual Conference*, *26*, 70(2010), 285–288.
36. Papadonikolakis, M., Bouganis, C. A heterogeneous FPGA architecture for support vector machine training. In *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)* (May 2010). IEEE, 211–214.
37. Qadeer, W., Hameed, R., Shacham, O., Venkatesan, P., Kozyrakis, C., Horowitz, M.A. Convolution engine: Balancing efficiency & flexibility in specialized computing. In *International Symposium on Computer Architecture*, 2013). ACM, 41(3), 24–35.
38. Sermanet, P., Chintala, S., LeCun, Y. Convolutional neural networks applied to house numbers digit classification. In *Pattern Recognition (ICPR)*, ..., 2012.
39. Sermanet, P., LeCun, Y. Traffic sign recognition with multi-scale convolutional networks. In *International Joint Conference on Neural Networks* (July 2011). IEEE, 2809–2813.
40. Stamoulias, I., Manolakos, E.S. Parallel architectures for the KNN classifier–design of soft IP cores and FPGA implementations. *ACM Transactions on Embedded Computing Systems (TECS) 13*, 2 (2013), 22.
41. Swanson, S., Michelson, K., Schwerin, A., Oskin, M. Wavescalar. In *ACM/IEEE International Symposium on Microarchitecture (MICRO)* (Dec 2003). IEEE Computer Society, 291.
42. Temam, O. The rebirth of neural networks. In *International Symposium on Computer Architecture*, (2010).
43. Temam, O. A defect-tolerant accelerator for emerging high-performance applications. In *International Symposium on Computer Architecture* (Sep 2012). Portland, Oregon, 40(3), 356–367.
44. Vanhoucke, V., Senior, A., Mao, M.Z. Improving the speed of neural networks on CPUs. In *Deep Learning and Unsupervised Feature Learning Workshop* (NIPS) (2011). Vol. 1.
45. Wang, G., Anand, D., Butt, N., Cestero, A., Chudzik, M., Ervin, J., Fang, S., Freeman, G., Ho, H., Khan, B., Kim, B., Kong, W., Krishnan, R., Krishnan, S., Kwon, O., Liu, J., McStay, K., Nelson, E., Nummy, K., Parries, P., Sim, J., Takalkar, R., Tessier, A., Todi, R., Malik, R., Stiffler, S., Iyer, S. Scaling deep trench based EDRAM on SOI to 32 nm and beyond. In *IEEE International Electron Devices Meeting (IEDM)* (2009). IEEE, 1–4.
46. Wolpert, D.H. The lack of a priori distinctions between learning algorithms. *Neural Comput. 8*, *7* (1996), 1341–1390.
47. Yeh, Y.-J., Li, H.-Y., Hwang, W.-J., Fang, C.-Y. Fpga implementation of KNN classifier based on wavelet transform and partial distance search. In *Image Analysis* (June 2007). Springer Berlin Heidelberg, 512–521.

Yunji Chen, Tianshi Chen, Zhiwei Xu, and Ninghui Sun ({cyj, chentianshi, zxu, snh}@ict.ac.cn), SKL of Computer Architecture, ICT, CAS, China.

Olivier Temam (olivier.temam@inria.fr), Inria Saclay, France.

© 2016 ACM 0001-0782/16/11 $15.00