Cambricon-F: Machine Learning Computers with Fractal von Neumann Architecture

Yongwei Zhao[1,2,3], Zidong Du[1,3], Qi Guo[1,3], Shaoli Liu[1,3], Ling Li[4], Zhiwei Xu[1,2], Tianshi Chen[1,3], and Yunji Chen ✉[1,2]

[1]SKL of Computer Architecture, Institute of Computing Technology, CAS

[2]University of Chinese Academy of Sciences [3]Cambricon Tech. Ltd [4]Institute of Software, CAS

{zhaoyongwei, duzidong, guoqi, liushaoli, zxu, chentianshi, cyj}@ict.ac.cn; liling@iscas.ac.cn

ABSTRACT

Machine learning techniques are pervasive tools for emerging commercial applications and many dedicated machine learning computers on different scales have been deployed in embedded devices, servers, and data centers. Currently, most machine learning computer architectures still focus on optimizing performance and energy efficiency instead of programming productivity. However, with the fast development in silicon technology, programming productivity, including programming itself and software stack development, becomes the vital reason instead of performance and power efficiency that hinders the application of machine learning computers.

In this paper, we propose *Cambricon-F*, which is a series of homogeneous, sequential, multi-layer, layer-similar, machine learning computers with the same ISA. A Cambricon-F machine has a fractal von Neumann architecture to iteratively manage its components: it is with von Neumann architecture and its processing components (sub-nodes) are still Cambricon-F machines with von Neumann architecture and the same ISA. Since different Cambricon-F instances with different scales can share the same software stack on their common ISA, Cambricon-Fs can significantly improve the programming productivity. Moreover, we address four major challenges in Cambricon-F architecture design, which allow Cambricon-F to achieve a high efficiency. We implement two Cambricon-F instances at different scales, i.e., Cambricon-F100 and Cambricon-F1. Compared to GPU based machines (DGX-1 and 1080Ti), Cambricon-F instances achieve 2.82x, 5.14x better performance, 8.37x, 11.39x better efficiency on average, with 74.5%, 93.8% smaller area costs, respectively.

1. INTRODUCTION

Machine learning techniques are pervasive tools for emerging commercial applications, including image recognition [1–3], speech recognition [4, 5], face cognition [6, 7], video analysis [8, 9], advertisement recommendation [10], and games [11, 12]. In recent years, many dedicated machine learning computers on different scales have been deployed in embedded devices, servers, and data centers. For example, Huawei Mate10 and P20 cellphones integrated Cambricon-1A machine learning processor core [13]. Apple iPhone X cellphones also integrated a machine learning subsystem to identify faces of users [14]. NVIDIA produced DGX-1 and DGX-2 machine learning computers based on NVIDIA GPU [15, 16]. Google announced a machine learning computer with 100 Petaflops peak performance based on TPU-3 chips [17]. Recently, IBM announced Summit, which is a machine learning supercomputer with 9216 POWER9 CPUs and 27648 NVIDIA V100 GPUs [18].

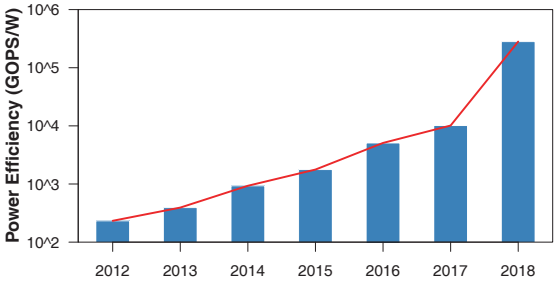


Figure 1: Power efficiency of recent proposed machine learning accelerators [19–25].

Currently, most machine learning computer architectures still focus on optimizing performance and energy efficiency instead of programming productivity. In Figure 1, we try our best effort to summarize the power efficiencies of the most efficient machine learning accelerators proposed in the very year from 2012 to 2018. Obviously, the power efficiency keeps increasing at a dramatic speed, i.e., 3.2x each year. Neuflow achieves 230GOPS/W with IBM 45 nm technology in 2012 [19]. DianNao, a deep neural network accelerator proposed in 2014, improves the power efficiency by a factor of 4.05x. And in 2018, Conv-RAM achieves 28.1TOPS/W [25], i.e., 1213x improvement compared with those in 2012.

While energy efficiency of machine learning computers keeps increasing rapidly, programming productivity—including programming itself and software stack development—becomes the vital reason that hinders the deployment of machine learning techniques. Even if a machine learning computer has a high peak performance-energy efficiency, high-quality program and software stack are still essential to fulfill the actual performance and energy consumption

✉ Corresponding Author

ISCA '19, June 22–26, 2019, Phoenix, AZ, USA

DOI: https://doi.org/10.1145/3307650.3322226

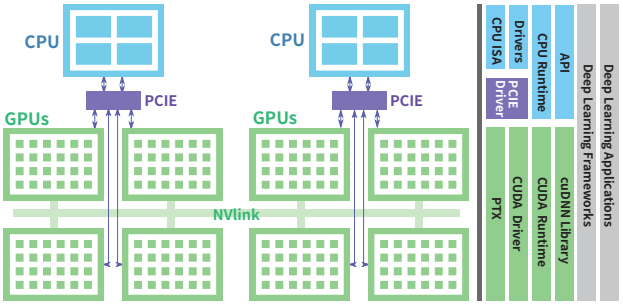


Figure 2: A typical machine learning computer architecture.

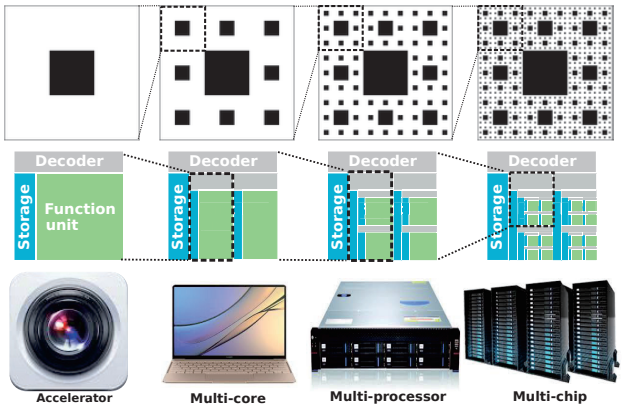


Figure 3: *Top:* A fractal graph example: Sierpinski carpet [30]. The graph is subdividing itself into smaller copies and continuing recursively. *Botttom:* Fractal computers, analogy to Sierpinski carpet.

requirements of machine learning applications.

Programming productivity is further compromised by different programming interfaces in a single machine learning computer. As illustrated in Figure 2, a traditional machine learning computer often has many heterogeneous parallel components organized in a hierarchical way. While programming heterogeneous systems and parallel systems are already notoriously difficult, each layer in a traditional hierarchical machine learning computer may have a different programming interface, which further exacerbates the programming challenge. For example, a GPU-based machine learning computer, such as NVIDIA DGX-2 [16], contains heterogeneous chips, i.e., 2 CPUs (24 cores per CPU) and 16 V100 GPUs. Except that programming multiple GPUs requires manual work based on MPI or NCCL, programming a single GPU chip needs to use the CUDA language to manipulate thousands of GPU threads; programming CPUs needs to write C/C++ with parallel API support for tens of CPU threads. Moreover, even the software stack inside a single GPU is also quite complicated, which includes CUDA PTX for programming grids/blocks/threads in the GPU, and microcode for programming a stream processor [26]. Considering there have been so many different machine learning computers, the industry needs to put huge efforts on porting system software (including but not limited to libraries, algorithm primitives, programming frameworks, assemblers, and compiler backends) to machine learning computers. For instance, just in the Tensorflow alone, there are thousands of operators [27], and optimizing an operator (e.g., convolution) on a certain GPU can cost several months for a skilled developer. Porting an operator to a multi-GPU computer could be even more time-consuming. HuaWei and Cambricon have put hundreds of software developers to port programming frameworks to the machine learning subsystem in Mate10 cellphone [28].

In a nutshell, the programming productivity is greatly reduced by the heterogeneous, parallel, and layer-different nature of machine learning computer. Hence, we claim that an ideal computer for programmer should be homogeneous, sequential, and layer-similar, which allows simple sequential programming for machine learning system software and applications. Moreover, if all machine learning computers (even with extremely different scales) have the same ISA, then the burden of programmers can be further alleviated, since they do not need to implement and port machine learning system software again and again. Here the question is: Is it possible to develop a series of homogeneous, sequential, layer-similar, machine learning computers with the same ISA, which still have high efficiency?

To answer this question, we propose *Cambricon-F*, which can achieve easy-programming and high-efficiency for machine learning simultaneously. The key insight of Cambricon-F is to organize the components of a computer in a fractal way. Originally, the word "fractal" in math is used to describe complicated objects which exhibit similar patterns at different scales, known as expanding symmetry or evolving symmetry [29]. Without dive into the controversy in math, we borrow the *concept* of fractal for iterative decomposition with self-similar patterns to any scale, see Figure 3 *Top*. Extended to computer domain, Cambricon-F is a series of homogeneous, sequential, multi-layer, layer-similar, machine learning computers with the same ISA. A Cambricon-F machine has a fractal von Neumann architecture to iteratively manage its components: it is with von Neumann architecture and its processing components (sub-nodes) are still Cambricon-F machines with von Neumann architecture and the same ISA. It features the fractal computing that iteratively decomposes an instruction on it into several instructions on low-layer sub-nodes. Hence, Cambricon-Fs with different scales can be used for different scenarios from embedded systems, desktops, data centers to supercomputers. As shown Figure 3, a single-core accelerator, multi-core chip, multi-chip server, and multi-server system can be architected in a fractal way with the same ISA, for different scenarios in different scales. Thus, programmers only need to consider one sequential ISA to run the same code on any of such devices.

In this paper, we made the following major contributions.

- We thoroughly find that common machine learning primitives can be considered as fractal operations, which can be decomposed into several smaller self-similar operations iteratively.
- We proposed Cambricon-F, which is a series of homogeneous, sequential, multi-layer, layer-similar, machine learning computers with fractal von Neumann architecture and same ISA. By providing a sequential view to programmers, Cambricon-F can achieve easy-programming and high-efficiency simultaneously.
- We summarize the four challenges in mapping different types of fractal operations onto Cambricon-F, including reduction operation mapping, fractal data management, communication congestion, and inter-instruction optimization. We propose a series of techniques to address the four major challenges.
- We design and implement two Cambricon-F instances at different scale down to layout level and evaluate these Cambricon-F instances with quantitative experimental results. Compared

to GPU based machines, with higher programming productivity (due to the same sequential ISA), Cambricon-F instances are also able to achieve better performance and efficiency.

2. FRACTAL OPERATION AND MACHINE LEARNING

In this section, we first analyze common machine learning techniques by decomposing them into computing primitives. Then we define the fractal operation, analyze three types of fractal operation with different computing dependencies, and demonstrate that all common machine learning computing primitives fall into the three types of fractal operation. We finally present the challenges in designing a fractal architecture that can effectively process all three types of fractal operations.

2.1 Machine Learning

Machine Learning Techniques. Machine learning techniques are usually computation&memory intensive and diverse in many aspects, such as processing flow, learning style, and training methodology Fortunately, they are highly paralleled at different levels, and thus can be accelerated with heterogeneous machine learning computers, which equip dedicated devices, including GPU [31–33], FPGA [34–36], and even AISC chips [22, 37–40]. Here, we first decompose these techniques into computing primitives, then illustrate the mapping to fraction computing form.

Computing primitives. We select six representative techniques and decompose the CPU execution time with typical dataset into their common primitives, see Table 1. Specifically, for the popularity of deep learning, we select the famous AlexNet [3] running with ImageNet [41] to represent convolutional neural networks (CNNs), a 3-layer multi-layer perceptron (MLP) to deep neural networks (DNNs). Others are k-means, k-NN, support vector machine (SVM), and learning vector quantization (LVQ). In line with previous works [22, 42–44], we decompose machine learning techniques into matrix and vector based operations. We aggregate operations such as vector multiplying matrix and matrix multiplying vector into matrix multiplying matrix, operations such as matrix adding/subtracting matrix, matrix multiplying scalar, and vector elementary arithmetics into element-wise operation. Hence we get seven major computing primitives after decomposition, including convolution (CONV), pooling (POOL), matrix multiplying matrix (MMM), element-wise operation (ELTW), sorting (SORT), and counting (COUNT). We still have *CONV*, *POOL* primitives instead of only using *MMM* for the convenience of analyzing and mapping emerging important deep learning algorithms. Note that *IP* is actually vector-multiplying-vector, which can also represent the fully connected layer in deep networks. It can be observed that these seven computing primitives characterize machine learning techniques mainly.

Table 1: Decomposing execution times of typical machine learning techniques into common primitives (IP: inner production; CONV: convolution; POOL: pooling; MMM: matrix multiplying matrix; ELTW: element-wise operation; SORT: sorting; COUNT: counting).

ML	Primitives						
	IP	CONV	POOL	MMM	ELTW	SORT	COUNT
CNN	-	94.7x%	0.18%	5.02%	0.12%	-	
DNN	-	-	-	99.9x%	0.11%	-	-
k-Means	90.8%	-	0.116%	-	9.08%	0.178%	0.012%
k-NN	99.6%	-	-	-	-	0.432%	-
SVM	99.3%	-	0.190%	-	0.507%	-	-
LVQ	39.9%	-	0.254%	-	59.8%	-	-

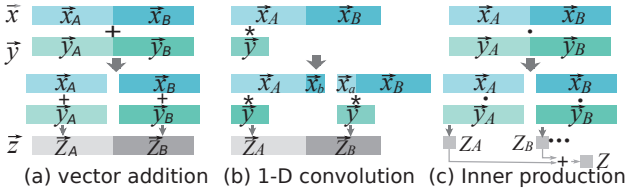


Figure 4: Fractal operation dependency: (a) independent; (b) input dependent; (c) output dependent.

2.2 Fractal operation

Fractal operation. We say that an operation $f(\cdot)$ with an input tensor $\mathbf{X}$ is a fractal operation if there exists an operation $g(\cdot)$ allowing

$$f(\mathbf{X}) = g(f(\mathbf{X_A}), f(\mathbf{X_B}), ...) \quad (1)$$

where $f(\cdot)$ is the target operator, $g(\cdot)$ is the retrieving operator, $\mathbf{X}$ represents all operands of $f(\cdot)$, $\mathbf{X_A}, \mathbf{X_B}, ...$ are the subsets of $\mathbf{X}$. Based on the relationship among $\mathbf{X_A}, \mathbf{X_B}...$ and $\mathbf{X}$, we can divide the fractal operations into three categories: *independent*, *input dependent*, and *output dependent*.

- If $\mathbf{X_A}, \mathbf{X_B}...$ are independent, non-overlapped to each other, each subset is independent that they can be computed locally, i.e., *independent*. In Figure 4 (a), we use a vector adding operation as an example to present *independent* fractal operation. For clear illustration, we split $\mathbf{X}$ into two operands, i.e., $\vec{x}, \vec{y}$—two input vectors for adding. As $\vec{x}$ and $\vec{y}$ can be divided into two independent pieces ($\vec{x_A}, \vec{x_B}$ and $\vec{y_A}, \vec{y_B}$), two vector adding operations can be achieved independently, i.e., $\vec{z_A} = \vec{x_A} + \vec{y_A}$ and $\vec{z_B} = \vec{x_B} + \vec{y_B}$. Each piece is working on independent part of the inputs and the final outputs just need assemble with no additional operation, i.e., $\vec{z} = [\vec{z_A}, \vec{z_B}]$. Thus, $g(\cdot)$ is linear function $g(x) = x$.

- If $\mathbf{X_A}, \mathbf{X_B}...$ are overlapped, each subset requires extra copies of some inputs that leads input redundancy in the fractal operation, i.e., *input dependent*. For example, a one-dimensional convolution as shown in Figure 4 (b). Similarly, we use $\vec{x}, \vec{y}$ to represent two operands and $\vec{x} = [\vec{x_A}, \vec{x_B}]$. We still divide the operation into two pieces, where each piece is working on independent part of outputs, i.e., $\vec{z} = [\vec{z_A}, \vec{z_B}] = \vec{x} \star \vec{y} = [\vec{x_A}, \vec{x_B}] \star \vec{y}$. However, these two operations have overlapped inputs, where parts of $\vec{x_A}$ and $\vec{x_B}$ ($\vec{x_a}, \vec{x_b}$, respectively) are required additionally, i.e., $\vec{z_A} = [\vec{x_A}, \vec{x_b}] \star \vec{y}$ and $\vec{z_B} = [\vec{x_a}, \vec{x_B}] \star \vec{y}$. But, there is still no additional operation for final outputs, i.e., $g(x) = x$.

- In some cases, $g(\cdot)$ is introduced to reduce the results of pieces into the final results, i.e., *output dependent*. For example, as shown in Figure 4 (c), an inner production operation ($z = \vec{x} \cdot \vec{y}$) can be divided into smaller pieces where each piece still performs an inner production operation ($z_A = \vec{x_A} \cdot \vec{y_A}$ and $z_B = \vec{x_B} \cdot \vec{y_B}$); but to get the final results, the results of those pieces will be summed up, i.e., $z = z_A + z_B$. Thus, $g(\cdot)$ is the sum operation, $g(\cdot) = sum(\cdot)$. Note that a fractal operation can be both *output dependent* and *input dependent*.

2.3 Fractal computing for machine learning

We present how machine learning computing primitives can be accomplished in a fractal form (i.e., fractal computing) and analyze

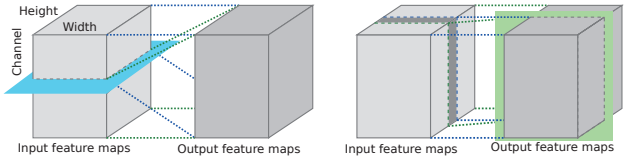


Figure 5: *CONV* decomposition. *Left:* dividing in channel dimension. *Right:* dividing in height dimension.

the challenges for designing corresponding architecture. Based on the above analysis, we can classify all machine learning primitives into three categories, see Table 2. Note that different decomposition can lead to different dependence. For example, *CONV* can divide the input features maps in channel dimension, where the final outputs rely on results from each divided pieces (thus *output dependent*), as shown in Figure 5 *(left)*; *CONV* can divide the input feature maps in height or width dimension, where each part of the output results only need inputs with some overlaps (thus *input dependent*), as shown in Figure 5 *(right)*.

More importantly, to effectively process fractal operations, fractal architecture should be built hierarchically with a tree-like topology where several son nodes compose a father node iteratively, see example Cambricon-F architecture shown in Figure 3. Obviously, *independent* operations are easily mapped to such fractal architecture and computed fractally. Also, *input dependent* can be transformed to *independent* with input redundancy. For the 1D convolution operation in Figure 4 (b), each part only needs some more inputs from **X** then the fractal operation is *independent*. In Table 2, we present the analysis of decomposition of computing primitives in a fractal form. Additionally, we present the data redundancy if using *independent* decomposition instead of *input dependent*. For the *output dependency* operations, $g(\cdot)$ is inevitable no matter whether inputs are dependent or independent. Thus, it is totally feasible to perform machine learning computations in a fractal form. But for designing fractal architecture, we must solve the following challenges related to extra data redundancy and reduction operation $g(\cdot)$:

- **Reduction Operation.** Reduction operation $g(\cdot)$ in *output dependent* operations are not naturally fitted in fractal operation as *independent* and *input dependent* operations. Thus, for efficiently processing $g(\cdot)$, we introduce lightweight computing unit (i.e., LFU) in each node locally. By aggregating data in son FFUs into a father LFU iteratively, such operations can be processed efficiently in father LFUs in Cambricon-F. We introduce that in detail in later Section 3.1, 3.2, 3.3.

- **Data Redundancy.** In fractal operation computing, *input dependent* operations can be computed as *independent* operations but with data redundancy. For that, the memory is hierarchically organized and the memory allocation leveraging the separable time order (Section 3.5). Also, we implement tensor transposition table with pipeline concatenating and data broadcasting to reduce the data movements, so as to improve the efficiency of Cambricon-F (Section 3.6).

Table 2: Computing primitives analysis.

Primitives	Decomposition	Dependency	$g(\cdot)$	Data Redundancy
IP	Length-Wise	Output	Add	-
CONV	Feature-Wise	Output	Add	-
CONV	Batch-Wise	Input	-	Weight
CONV	Spatial	Input	-	Weight, Overlapped
POOL	Feature-Wise	Independent	-	-
POOL	Spatial	Input	-	Overlapped
MMM	Left,Vertical	Output	Add	-
MMM	Right,Vertical	Input	-	Left Matrix
ELTW	Any	Independent	-	-
SORT	Any	Output	Merge	-
COUNT	Any	Output	Add	-

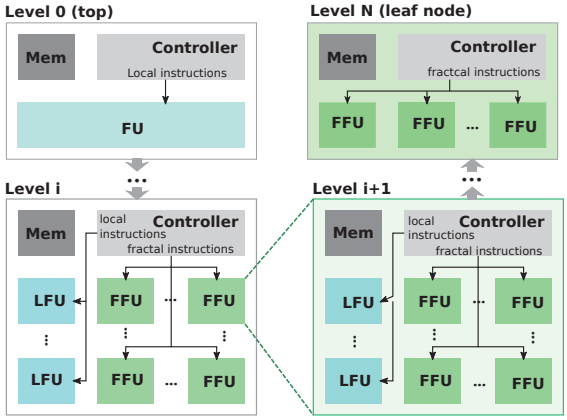


Figure 6: A typical fractal von Neumann architecture: level 0 (top node)...level i node and its son node in level $i+1$...level N (leaf node).

- **Communication.** Communication among different nodes would lead to enormous wire connections and consequently to be costly in terms of area, latency, and energy. For that, from our analysis, even the *output dependent* operations only require data movements from leaf to root node for reduction operations. Thus, it is unnecessary to have communication between any pair of nodes. In Cambricon-F, we organize the machine learning computations iteratively in a fractal form and limit the connections to father-son nodes only, thus reducing the wire congestion (Section 3.3, 3.4).

- **Optimization.** Optimization among instructions would be critical to obtain high efficiency. Individual operations are easy to be implemented the entire processing flow from data fetching to results writing back, but it may ignore the potential benefits from data reuse among instructions. For that, we implement pipeline to leverage the parallelism among instructions (Section 3.4); we also introduce pipeline concatenating and data broadcasting in Cambricon-F to maximally utilize the data already allocated in FFUs (Section 3.6).

In summary, after addressing the above concerns, the fractal architecture would be able to achieve at least comparable efficiency with traditional architecture for machine learning applications.

3. CAMBRICON-F COMPUTERS

In this section, we present the Cambricon-F computers from the architects' perspective, including overall architecture, instruction set architecture, decoder, pipeline, memory hierarchy, and implementation details.

3.1 Fractal von Neumann Architecture

A Cambricon-F machine has a fractal von Neumann architecture, which is hierarchical architecture built iteratively, as illustrated in Figure 6. At the top level (root node), programmers should only learn a simple von Neumann architecture that contains a memory

component (Mem), a functional unit (FU), and a controller with a decoder inside to decode instructions. In the middle levels, each node is still with von Neumann architecture, containing a controller (it can be either hardware or software), a memory component (Mem), several processing units including local functional units (LFU) and several fractal functional units (FFU). Each FFU is a son node (*Level i+1*) of the current node (*Level i*) and has the same ISA and similar architecture. At the bottom level, each leaf node is an accelerator than finishes the most part of the computation. Therefore, a Cambricon-F machine is built with a fractal von Neumann architecture to iteratively manage its components.

The ISA of Cambricon-F is Fractal Instruction Set Architecture (FISA), where each fractal operation can be performed with one or more FISA instructions. FISA includes two different kinds of instructions: local instructions and fractal instructions. For a local instruction, the controller can directly issue it to an LFU, and the LFU will complete the local instruction. For a fractal instruction, the controller will translate into several instruction segments, where each instruction segment is solved by an FFU. Hence, programming Cambricon-F only needs to consider a single sequential ISA, while the heterogeneity can be implicitly solved through the collaboration between LFUs and FFUs, and the parallelism can be implicitly solved through the parallelism between FFUs. Since an Cambricon-F computer and its all descendant Cambricon-Fs/FFUs have the same ISA, a programmer does not need to consider the difference between different layers of a machine learning computer. Moreover, different Cambricon-F computers with different scales (either a machine learning supercomputer or a small machine learning subsystem in a cellphone) can use the same ISA, which allows a same binary code to run on platforms from cloud to end.

To efficiently process fractal operations, Cambricon-F adopts a hierarchical memory system. Cambricon-F manages the storage in two types: global memory and local memory. At the top level, Cambricon-F contains a larger memory for buffering input data, i.e., the global memory, which is also visible to programmers. Each node in Cambricon-F contains a local storage to buffer the data, which will become a "global memory" shared among its son nodes. In such a manner, we manage all the memory in Cambricon-F hierarchically.

3.2 Instruction Set Architecture

Cambricon-F leverages a special instruction set architecture to achieve the fractal computing, i.e., Fractal Instruction Set Architecture (FISA). Formally, we give the definitions of FISA instruction and FISA:

- *FISA instruction.* A FISA instruction, I, is a 3-tuple $\langle O, \mathcal{P}, \mathcal{G} \rangle$, where O is an operation, $\mathcal{P}$ is a finite set of operands, $\mathcal{G}$ is granularity indicator.
- *Fractal instruction.* A FISA instruction, $I\langle O, \mathcal{P}, \mathcal{G} \rangle$, is a fractal instruction, *iff* there exists a set of scale indicators $S'_1, S'_2, \ldots, S'_n$ ($S'_i \preceq S$, $\preceq$ is the partial order defined on scale indicators) that I can be achieved through computing with $s'_1, s'_2, \ldots, s'_n$ and other FISA instructions iteratively.
- An ISA set is a FISA set, *iff* it contains at least one fractal FISA instruction.
- A machine M running FISA set R is a fractal machine, *iff* there exists at least one fractal instruction I that is fractal-executed on M.

The FISA design for Cambricon-F stays at a relatively higher level so as to improve the programming productivity with same sequential code, as in Table 3 where we show a subset of FISA.

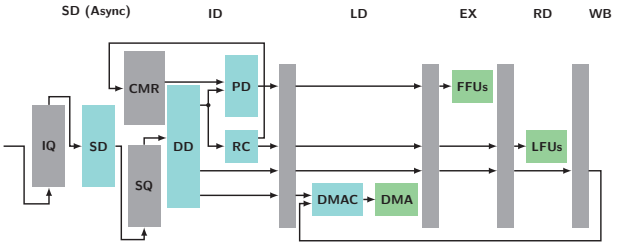


Figure 7: Pipeline partition in an Cambricon-F node.

Primitives such as convolution and sorting can be directly expressed with FISA instructions. Operations of low operation intensity (e.g. Element-Wise Operations) are also supported in FISA for better programming versatility. Such instructions will be considered as a reduction operation by Cambricon-F and tend to execute on LFUs.

3.3 Controller

The controller exists in each node in a Cambricon-F, serving to manage its son nodes working in a fractal manner. From a functionality perspective, the controller consists of three phases: a sequential decomposition phase, a demotion phase, and a parallel decomposition phase. Several specific modules are thusly designed in pipeline stages to accomplish the transformation procedure from input instructions to sub-level nodes, including FFUs and LFUs, see Figure 7. Briefly, in sequential decomposition phase, input instructions are loaded into *Inst Queue* (IQ), which is later fetched by *Sequential decomposer* (SD). SD decomposes into a sequential executed instruction list regarding the hardware limitation. In demotion phase, reformed instructions in the list are decoded by *Demotion Decoder* (DD) into sub-level instructions. In parallel decomposition phase, sub-level instructions for fractal computing will be passed to FFUs through a *Parallel decomposer* (PD), for local computations to LFUs through a *Reduction Controller* (RC), for data movements to *DMA Controller* (DMAC) to access memory.

Particularly, ***Demotion Decoder***, the key component in controller, decode input upper instructions to sub-level instructions to be fractally computed. For each *sub-level instruction SQ*, *DD* checks operand dependencies to instructions running in the pipeline. *DD* will stall the pipeline if a *read-after-write* (RAW) dependency exists. *DD* also checks the storage requirements of operands, allocates memory space locally, and generates DMA instructions. DMA instructions will be sent to *DMAC* for data exchange between local memory and "shared memory" in upper level, e.g., loading sources or writing back results. *DD* then binds the new *local* addresses to

Table 3: Examples of Cambricon-F Instructions

Type	Operation	Name
Deep Learning	Convolution Pooling LRN [3]	CV2D, CV3D MAX2D, MIN2D, AVG2D LRN
Linear Algebra	Matrix Multiplication Euclidian Distance	MATMUL EUCLIDIAN1D
Sort	Merge Sort	SORT1D
Count	Count	COUNT1D
Reduction	Binary Element-wise Unary Element-wise Horizontal Merge	ADD1D, SUB1D, MUL1D ACT1D HSUM1D, HPROD1D MERGE1D

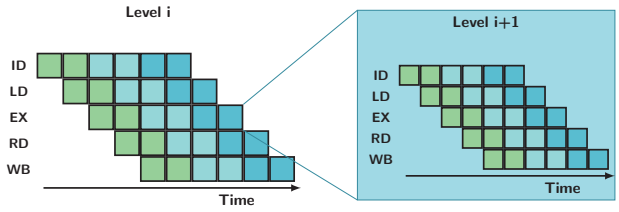


Figure 8: Fractal Pipeline of FISA.

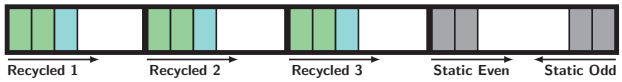


Figure 9: Memory Management of Cambricon-F Controller. The memory space is divided into 4 segments (3 recycling and 1 static), managed as 5 stacks (2 stacks in the static segment).

operands in *sub-level instructions* which later sent to *PD* for fractal computing, *RC* for reduction operations.

Parallel decomposer subdivides sub-level instructions into multiple FISA instructions, i.e., fractal instructions, that are assigned FFUs. FFUs process fractal FISA instructions during the EX pipeline stage in parallel.

Reduction Controller aims to perform the reduction operation in *output dependent* fractal operations normally. However, reduction operation can be assigned to FFUs instead of LFUs for high efficiency, when *RC* predicts significantly reduced execution time on FFUs or founds LFUs unavailable. In cases, *Reduction Controller* send a *commission* to *PD* by writing the operation into *Commission Register (CMR)*. *PD* will check if there is a commission in the register at the start of each FISA cycle, and append the commissioned operation.

3.4 Pipelining

Normally, Cambricon-F performs each instruction recursively. Top level (level 0) node decodes and sends fractal instructions to its FFUs where each FFU repeats the decoding/sending procedure until the leaf nodes for execution. Leaf nodes return computed results to their father nodes and that repeats until the top node. During such execution process, FFUs in leaf nodes, where the heavy computation tasks are performed, are idle when its upper-level nodes are decoding instructions recursively.

Thus, in order to increase the throughput of Cambricon-F, we pipeline the FISA instruction execution into *five* stages: *Instruction Decoding (*ID*), Loading (*LD*), Execution (*EX*), Reduction (*RD*)* and *Writing Back (*WB*)*, see Figure 7. Similar to pipeline in CPUs, an issued instruction will be decoded local instructions, fractal instructions, and DMA operations in the Controller (ID). Data is loaded from memory to local storage for FFUs and LFUs computation (EX stage) with DMA operations at LD stage. LFUs will start the reduction operations at RD stage or be bypassed if no reduction operation needed. The final or partial results will be written to memory from local storage at WB stage. Note that SD is executed asynchronically where *SD* keeps decompositing instructions from *IQ* to *SQ*.

As each node in Cambricon-F executes its 5-stage pipeline for its instructions, Cambricon-F will execute FISA with a recursive pipeline. In Figure 8, we show the pipeline for a two-level Cambricon-F. In the EX stage of level 0, FFUs run their own pipeline, i.e., level 1 pipeline. As a result, the recursive pipeline of FISA has utilized every component of every hierarchy at almost any time, except the pipeline startup and emptying.

3.5 Memory Management

As each phase in the *Controller* may require memory allocation, memory management is challenging and more critical to the overall efficiency. Fortunately, we observe that memory allocation can be well separated in term of time, owing to functionality separated modules in *Controller*, i.e., *SD*, *DD* and *PD*. Specifically, memory blocks allocated by *PD* are always allocated later, but released sooner than blocks simultaneous allocated by *DD*, and blocks allocated by *Demote Decoder* are always allocated later but released sooner than blocks simultaneously allocated by *SD*. Furthermore, blocks for parallel decomposition only live in EX and sometimes RD pipeline stage, and blocks for demotion live in the whole FISA cycle. But memory blocks for sequential decomposition may live across multiple FISA cycles since there could be multiple sub-level instructions decomposed from the FISA instruction.

Thus, we design the memory controller with leveraging the timing order observation. As four stages except for ID of FISA pipeline involved in memory movements, there are only four instructions maximally that may need memory access at the same time. Thus, we divide the local storage into four dependent memory spaces, and three are assigned to these four instructions in the pipeline as a new instruction reaching LD stage can reuse the memory space of the instruction at the WB stage. For each instruction, memory allocation requests will be queued in a list and processed by DMA controller sequentially. As shown in Figure 9, memory space is always allocated in the list order, which is consistent with the time order that *Controller* requests. For the memory allocation alive for multiple FISA instructions, i.e. those for sequential decomposition, we use the fourth memory space (*static segment*) which is shared by every pipeline stage for their different lifecycles. Memory allocation to *static segment* are double-ended for parity of instructions to avoid overlapped memory lifecycles of adjacent instructions. The design of the allocation list significantly reduces the complexity of memory management, and keep the space utilization efficiency.

Here, we do not manually release the allocated memory space, where new instruction will directly refill with new data. The reason is two-fold. First, results of instruction will be written back and the remain inputs or intermediate results are usually useless for later computations, as our FISA instructions work at a relatively higher level. Second, for seldom cases that following instructions may share the inputs/outputs, we implement a Tensor Transposition Table to find out data that can be forwarded or reused, resulting in a similar behavior as "pipeline forwarding" (see Section 3.6).

3.6 Implementation details

Tensor Transposition. As memory bandwidth is always critical to performance for large designs and heavy workloads, we propose the *Tensor Transposition Table* (TTT) to optimize the data reuse in two ways. First, TTT avoids the unnecessary data movements when two adjacent instructions share the same inputs. For a five-level Cambricon-F with 2048 cores (each level contains 1, 4, 8, 64, 2048 nodes), we observe that it can only reach 3% of the peak performance on RESNET-152, but with a 93.36% utilization of root memory bandwidth during the execution. With TTT recording the local allocation, *Demotion Decoder* is able to learn which operand is locally available and change the loading source address to local address during address binding; thus, remote data movements are avoided. As a result, Cambricon-F achieves 62% of peak performance for the previous example, with a 20x improvement. Second, TTT enables the data reuse as *pipeline forwarding*, where the next instruction uses the previous instruction's result as input. In this

case, two instructions can be pipelined without bubbles.

Data consistency and coherence. In Cambricon-F, we apply many constraints to manage the data consistency and coherence problems. Since our system will decompose the operation into smaller non-overlapped segments for son nodes execution; thus, data may have many copies in different nodes. However, as presented in Section 3.5, our instruction generation will not allow write data to the read address space, thus ensuring data consistency naturally in most of the cases. Additionally, TTT introduces the risk of data inconsistency as it forwards data from write address space to read address space. Instead of implementing costly consistency protocols, we set up a validity period of two FISA cycles for each record in TTT. TTT is split into two banks, where each bank is used only by one instruction. Whenever an instruction is using one bank, the new instruction entering the EX stage will use the other bank to record. Records in that bank were used by the previous instruction and can be overwritten. Thus, with such validity mechanism, the lifetime of records won't exceed the lifetime of the referenced data lying in parent memory. To guarantee the data coherence, simultaneous memory writes into the same memory address are always prohibited. The destination addresses of instructions assigned to each FFU will always be different.

Pipeline concatenating. In Cambricon-F, parallel decomposed sub-instructions are assigned to FFUs when EX stage starts; the EX stage ends once sub-instructions assigned to all FFUs are retired. It may cause pipeline emptying and child nodes refilling repeatedly at the boundaries of every parent FISA cycle. To concatenate pipelines of child nodes across the boundaries, we pre-assign sub-instructions in next EX stage to FFUs, advancing one FISA cycle. The pre-assigned instructions can be issued to pipelines of FFUs once DMA operations of parent node are finished. Note that there are some instructions which can not be pre-assigned because of the possible data dependency violations. In our experiments, 93.11% of instructions in RESNET-152 are able to be pre-assigned, which could benefit from pipeline concatenating, resulted in a 13.0% overall performance gain.

Data broadcasting. To further reduce the traffic at local memory, we enable the data broadcasting mechanism in each node. As parallel decomposed sub-instructions often have shared data among all FFUs, broadcasting data will require the local memory to send data only one time, thus reducing the data traffic. Each FFU will receive the other operand through DMA between itself and the local memory. While processing each FISA instruction, DMA will start processing data broadcasting after DMA requests in WB and LD stages are finished. Our experimental results show that data broadcasting improves the performance by 19.0%, and reduce the local memory traffic by 24.2%, while executing RESNET-152 on a Cambricon-F computer.

Memory size. Here, we study the relationship between memory size and performance under certain bandwidth constraint. As Cambricon-F adopts a fractal architecture where local memory in each node is shared by all the son nodes, we can view each node as a parallel computing model. Thus, the problem in each node turns to be a *memory bound scalable problem* in parallel computing model. Inspired from [45–47] which studies the relationship between memory capacity and workload size and operation intensity and bandwidth, respectively, we propose the Memory Bounded Operational Intensity (MBOI) problem studying the relationship between operation intensity and memory size, i.e., operational intensity $I = \mathrm{MBOI}(M)$. Thus, we are able to decide memory size at each node based on MBOI.

The MBOI problem is tightly related to each algorithm and workload. For many applications, $\mathrm{MBOI}(M)$ rises monotonically with

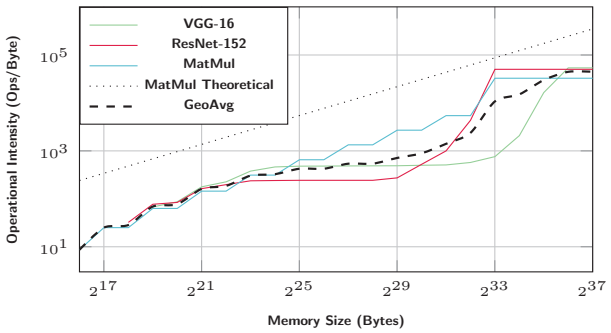


Figure 10: Measured Memory Bounded Operational Intensity (MBOI) on an Cambricon-F node.

M. In Figure 10, we show the measured and theoretical MBOI numbers in a Cambricon-F node (i.e., parallel computing model) of three representative algorithms. For Cambricon-F with different algorithms and workloads, we use the average MBOI (MBOI_{Ref}) to determine memory size of each node. To overlap computation and communication and avoid one of them being bottleneck, the bandwidth and peak performance of each Cambricon-F node should be balanced at a ratio around operational intensity, i.e.

$$\text{Peak Performance}/\text{Bandwidth} \approx \mathrm{MBOI}_{\mathrm{Ref}}(M)$$

Thus, we have

$$M \approx \mathrm{MBOI}^{-1}_{\mathrm{Ref}}(\text{Peak Performance}/\text{Bandwidth})$$

With above determination method, we are able to decide memory size in different Cambricon-F instance designs, which vary in hierarchy depth, node number at each level, and bandwidth constraints. More importantly, we are able to compare different Cambricon-F designs. In Table 4, we report simulated area and energy results of different Cambricon-F designs with the same capability, i.e., 512 Cores $\times$ 0.46 Tops per Core $=$ 238 TFlops. Note the local storage are eDRAMs simulated with DESTINY [48] with sizes up to 256 MB. Attainable performance in the table is defined as the geometric average of estimated performance of VGG-16, RESNET-152 and MATMUL running on a server composed of four chips. It can be observed that designs with fewer hierarchies tend to attain higher performance, but the desired memory space to support such a dense hierarchy is impractically large.

4. PROGRAMMING AND EXECUTION

Programming. With all the effort to provide programmers with sequential programming experiences, Cambricon-F are able to run the same piece of code without any other work. In Figure 11, we show a typical Cambricon-F inline assembly code using a k-Nearest Neighbor algorithm as a driving example. The principle of FISA is

Table 4: Estimated Power and Performance of Different Designs

Hierarchy	Power Watt	Performance TOPs/sec	Efficency TOPs/J	Area mm^2
1-512	1035.02	140.92	0.14	5662.72
1-16-512	176.49	**150.05**	0.85	783.53
1-2-16-512	**55.66**	113.34	**2.04**	**184.91**
1-4-16-512	57.52	107.12	1.86	263.64
1-4-16-64-512	68.83	104.94	1.52	208.72

k-Nearest Neighbors

```
int K = 3, N = 262144
tensor C[1,N], X[512,N]
tensor D[N,N], C2[N,N]
C2[:] = C[0]
# calculate distance for each pair of samples
fisa euclidian1d X, X, D
tensor C3[K,N], P[K,N]
# sort to find k-nearest neighbors' category
fisa sort1d D, dc, C2, C3[K,N][N,N][0,0]
# population count in k categories
fisa count1d C3, P
# sort to find the most popular category
fisa sort1d P, dc, C3, C[1,N][K,N][K-1,0]
```

Figure 11: An Cambricon-F program of k-NN.

that the nodes perform their *own* duties and *Do Not Interfere* with how the child nodes work. The programmer of Cambricon-F, which acts as the "controller" beyond the top level node, also follows the principle. The programming of Cambricon-F has the following characteristics:

- **High level, arbitrary granularity.** Each FISA instruction is corresponding to a *complete* machine learning primitive. The programmer does not interfere with how the operation is decomposed. High-level instructions bring higher operational intensity and help decrease data movements.
- **Implicit data movement.** Contrary to RISC, Cambricon-F does not provide explicit load-store instruction to the programmer. FISA hides the internal storage from the programmer by forcing all operands to be external. The programmer does not interfere with how the internal storage is used, so the program does not need to adapt to different internal storage sizes when applied to different Cambricon-F instances or nodes.
- **Hardware transparency.** Note that there is no hardware information appeared in the code. The programmer of Cambricon-F only dedicates on defining the computation task, and do not interfere with the internal hardware behaviors.

For the next level nodes, the controller of the parent node acts as a programmer. The Do-Not-Interfere principle reduced the complexity of the programming, meanwhile, it also reduced the complexity of the controller.

Execution on Different Cambricon-F Instances. The execution model of Cambricon-F can be summarized as *Single Task, Multiple Heritors (STMH)*. As shown in Figure 12a, a task is executed simultaneously on every hierarchies of Cambricon-F, where each hierarchy see a part of the task with different granularity. STMH defines how two adjacent hierarchies cooperate reducing the granularity to inherit the task from the higher hierarchy to the lower hierarchy. More specifically, the cooperating mechanics can be decoupled to two relations: the relation with parent node, and the relation between sibling nodes. Here, we define the paternity relation via *Sequential decomposer*, and the sibling relation via *Parallel decomposer*. Given the paternity and sibling relations, and under the assumption that leaf nodes can solve the assigned tasks directly, the execution of whole machine is clearly defined, regardless what configuration does the Cambricon-F instance have.

Figure 12 illustrates both relations. The sibling relation (Figure 12b) is to distribute the task to achieve concurrent execution. The paternity relation (Figure 12c) is to provide a strong guarantee

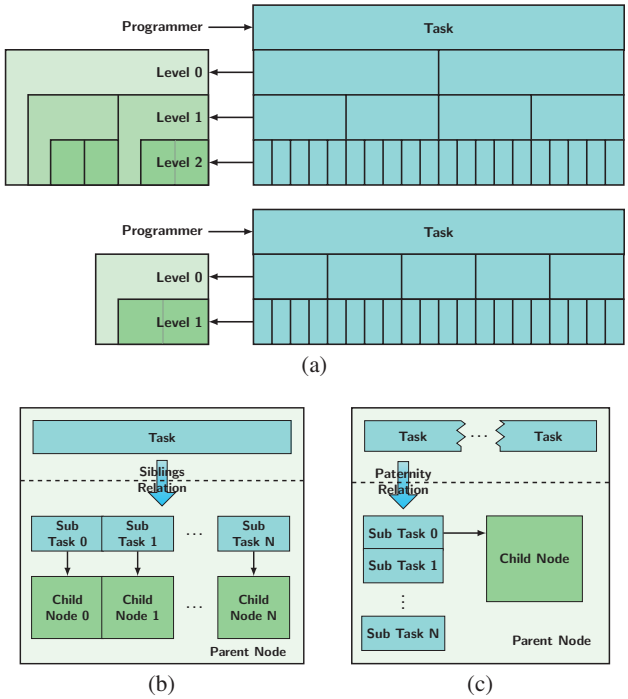


Figure 12: STMH execution model. (a) Execution of same task on different Cambricon-F instances. (b) Sibling Relation. (c) Paternity Relation.

that each Cambricon-F node accepts tasks of any granularity not subjected to hardware constraints so that the parent node (and programmers) would not need to have knowledge of any implementation details of the node to assign tasks. With this guaranty provided, the interface of Cambricon-F is unified among any hierarchies of any Cambricon-F instances. The program of Cambricon-F could be migrated without any modification since there is no hardware-dependent information involved in the program. Figure 12a illustrates the execution of the same task on two Cambricon-F instances of different configuration. Note that even two instances have different executions for the task, the task given by the programmer is identical.

In Figure 13, we give a concrete example to illustrate the execution process of sample code from Figure 11 on different Cambricon-F instances. The identical sample code is executed on two different instances, which are also the designs used for evaluation in this paper, i.e., Cambricon-F1 and Cambricon-F100 (see Section 5 for details). The execution on Cambricon-F1 which has a smaller configuration is heavily decomposed, while the execution on Cambricon-F100 has remained a relatively large granularity of sub-instructions. The last part of execution on Cambricon-F1 where has been communication-dominated is corresponding to the sorting and counting operations from the code. The execution time of Cambricon-F100 is dominated by the communications of the top-level hierarchy, which could be easily found out in Figure 13 (c).

5. METHODOLOGY

Benchmarks. As shown in Table 5, we use seven different benchmarks in this paper. For the importance of deep learning, we select VGG-16 [49], a 16-layer CNN with 138 M parameters in total, and ResNet-152 [50], a very deep network with 152 layers, running with ImageNet [41] dataset as representative benchmarks. We also

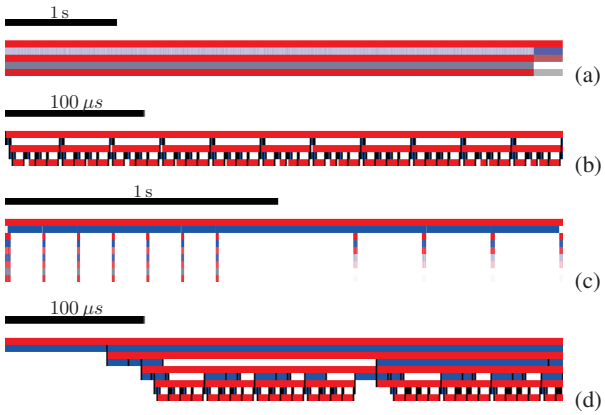


Figure 13: Execution timeline of sample code (Figure 11) on two different Cambricon-F instances: (a) Full execution on Cambricon-F1, (b) Figure (a) zoomed in between 0ms and 0.4ms. (c) Full execution on Cambricon-F100. (d) Figure (c) zoomed in between 1.4ms and 1.8ms. (Blue blocks: DMA execution; red blocks: FFUs and LFUs execution. Each figure shows the execution time of Cambricon-F hierarchies from top to bottom.)

select four popular machine learning techniques, including K-NN, K-MEANS, LVQ, and SVM, as representative benchmarks. For these four machine learning techniques, we use a randomly generated data set, which contains 262 thousand 512-dimension samples within 128 categories, to emulate a computation-heavy scenario. Additionally, as MATMUL is the most important operation in the machine learning domain, we also include MATMUL running with randomly generated 32768-order square matrices as our benchmark.

GPUs. In this paper, we select two GPUs as our baseline, i.e., Nvidia DGX-1 [15] and Nvidia GeForce GTX-1080Ti. DGX-1 is a supercomputer with eight NVIDIA Tesla V100-SXM2 GPUs, where each has a 125TeraOps/sec peak performance. The bandwidth from the host to devices is measured as 84.24GB/s in total. 1080Ti is a high-end graphics card with 10.6TeraOps/sec peak performance and 484GB/s memory bandwidth. For DGX-1, we program the benchmarks under the framework TensorFlow 1.9 [27] with GPU support (CUDA 9.0 [51] and cuDNN 7 [52]), and optimize the computation graph via NVIDIA TensorRT 4 [52]. We use `nvprof` and `nvidia-smi` to measure its power and memory bandwidth usage.

Cambricon-F. We build two different size Cambricon-F instances that have similar characteristics as GPUs, i.e., Cambricon-F100 and Cambricon-F1, for a fair comparison to GPUs. Cambricon-F100 is a fractal machine learning supercomputer with a peak performance of 956 Top/s, similar to DGX-1 (125*8=1000 Tops/s). Cambricon-F100 is a five-level architecture of Server, Card, Chip, Fractal Multiprocessor (FMP), and Core in each level from top to bottom, see Table 6. At the top level (L0), Cambricon-F100 contains

Table 5: Benchmarks.

Benchmark	Size
VGG-16 [49]	1.38×10^8 params, 3.09×10^{10} Ops, variable batch
ResNet-152 [50]	6.03×10^7 params, 2.26×10^{10} Ops, variable batch
K-NN	262,144 samples, 512 dimensions, 128 categories
K-Means	262,144 samples, 512 dimensions, 128 categories
LVQ	262,144 samples, 512 dimensions, 128 categories
SVM	262,144 samples, 512 dimensions, 128 categories
MATMUL	32,768 orders, square matrix

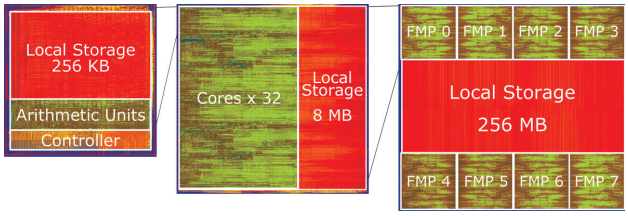


Figure 14: Layout of Cambricon-Fs. Left: Leaf *Core*. Mid: *FMP*(Cambricon-F1 Chip). Right: Cambricon-F100 Chip.

four Cambricon-F100 Computing Cards connected through PCI-E 3.0, a host CPU (Intel Xeon E5-4640 v4) serving as high-level controller and LFU, and 1TB host memory. The leaf node (L4) is a Cambricon-F accelerator serving as a computing Core, which has 256 KB eDRAM local storage, 16×16 MAC matrix running at 1 GHz, reaching peak performance of 477 GOPs/s. Cambricon-F1 is a Cambricon-F accelerating card at desktop scale with a peak performance of 14.9 Top/s, similar characteristics to 1080Ti (10.6 Top/s). Cambricon-F1 has a three-level architecture of Card, FMP, and Core in each level from top to bottom, see Table 6. Cambricon-F1 has one FMP on-chip and that has 32 cores inside.

To obtain the hardware characteristics, we implemented the Cambricon-F designs (up to chip level) in RTL and synthesize, place, and route using Synopsys toolchain under TSMC 45 nm technology. Fortunately, Cambricon-F is a fractal architecture built iteratively, we are able to estimate the hundreds millimeter square design using smaller pieces following bottom-up design philosophy. Due to the extreme long hardware emulation time and large design, we carefully build a simulator in C++ to get the performance. For energy costs, we dump data movements from our simulator and estimate memory costs with DESTINY [48], other parts are estimated based on our layout characteristics.

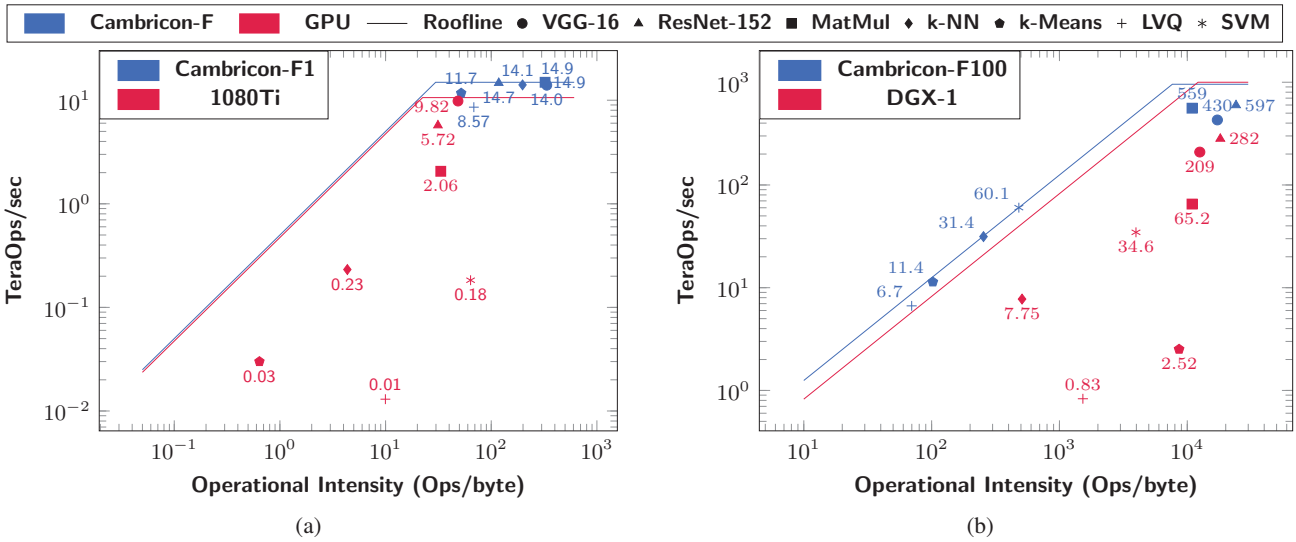
6. EXPERIMENTAL RESULTS

We first present the main characteristics of Cambricon-F instances, then present the performance and energy results when comparing against GPUs and accelerators. The experimental results are shown in Figure 15, where we adopt the Roofline Model [46] to illustrate the efficiency and bottleneck of the systems.

Hardware Characteristics. The layout of a *Core*, a *FMP* (same as a Cambricon-F1 Chip) and a Cambricon-F100 Chip are shown in Figure 14. In Table 7, we present the detailed hardware characteristics of the chip in Cambricon-F100 and Cambricon-F1. Cambricon-

Table 6: Specification of Cambricon-F instances.

Cambricon-F100	L0	L1	L2	L3	L4
Name	Server	Card	Chip	FMP	Core
# FFUs	4	2	8	32	-
# LFUs	1	0	16	16	-
Local Storage	1 TB	32 GB	256 MB	8 MB	256 KB
Bandwidth (GB/s)	128	512	512	512	80
Peak Perf.(TOPs/sec)	956	238	119	14.9	0.46
Cambricon-F1		**L0**		**L1**	**L2**
Name		Chip		FMP	Core
# FFUs		1		32	-
# LFUs		0		16	-
Local Storage		32 GB		8 MB	256 KB
Bandwidth (GB/s)		512		512	80
Peak Perf.(TOPs/sec)		14.9		14.9	0.46



(a) (b)

Figure 15: Roofline Cambricon-Fs compared to GPUs. (a) Cambricon-F1 and 1080Ti. (b) Cambricon-F100 and DGX-1.

F1 occupies 29.21mm^2 area, consuming a power of 4.94W, where each core has an area cost of 0.43mm^2, a power of 75.18mW at 45nm. Cambricon-F1, which is a 8-chip server having 2048 cores in total, has an area of 415mm^2 in total, consuming a power of 42.87W at 45nm. It can be observed that Cambricon-F favors of large memory.

In Table 8, we also compare Cambricon-F chips with GPUs and accelerators. It can be observed that Cambricon-F1 chip has the highest power efficiency and area efficiency, 3.02 Tops/W and 0.51 Tops/mm^2. Cambricon-F100 chip achieves the comparable area efficiency, but slightly lower power efficiency when compared against Google TPU [40]. While considering the entire card where 32 GB DRAM is included in each Cambricon-F Computing Card, Cambricon-F1 has a 40.57% more peak performance, but with 45.11% power cost of 1080Ti GPU card and Cambricon-F100 Computing Card has a 1.90x more peak performance with 67.34% power cost of a V100-SXM2 GPU card.

Cambricon-F1 vs. 1080Ti. As shown in Figure 15 (a), Cambricon-F1 has attained a 5.14x performance and 87.3% lower traffic on average when compared to 1080Ti. An Cambricon-F1 Computing Card consumes an average of an 83.1 Watt power for all benchmarks, and 1080Ti consumes an average of 199.9 Watt. The attained performance of Cambricon-F1 is from 1.42x to 659x higher than 1080Ti. Note that Cambricon-F1 has a 40.6% higher peak performance and a 5.8% higher root bandwidth relatively to 1080Ti.

The main reason for that is because of the large on-chip storage. As described in Section 3.6, large intermediate storage enables significantly greater *Memory Bounded Operational Intensity*. While in 1080Ti, the programmable nodes under the root memory, i.e., CUDA cores, have very limited local storage space (96KB shared memory vs. 8MB L1 local storage); thus, the operational intensity is bounded. The operational intensity of all seven benchmarks on Cambricon-F1 has reached the ridge point of the roofline, indicating that the root bandwidth will not be the performance bottleneck of Cambricon-F1. Thus, Cambricon-F1 has attained 57.4%-99.8%, 88.9% on average of peak performance on all benchmarks.

Cambricon-F100 vs. DGX-1. As shown in Figure 15 (b), Cambricon-F100 has a 51.9% higher root memory bandwidth compared to DGX-1, while the peak performance of Cambricon-F100 is 4.4% lower than DGX-1. For power consumption, four Cambricon-F100 Computing Cards consume an average of 614.5 Watt at the total, and eight V100-SXM2 GPU cards consume an average of 1986.5 Watt. Overall, Cambricon-F100 have attained 1.74x-8.58x performance, 2.82x on average, compared to DGX-1.

On deep learning tasks, Cambricon-F100 improved the operational intensity by 37% and 33% for VGG-16 and ResNet-152, respectively, when compared to DGX-1. The operational intensity benefits from greater sub-problem scale, i.e. from larger batch size

Table 7: Cambricon-F layout characteristics.

Component	Area(μm^2)	(%)	Power(mW)	(%)
Core	426,348		75.18	
Memory	201,588	(47.28%)	16.15	(21.48%)
Combinational	176,228	(41.33%)	23.74	(31.58%)
Registers	42,248	(9.91%)	27.38	(36.42%)
Others	6,284	(1.47%)	8.38	(11.14%)
Chip				
Cambricon-F1	29,206,289		4,935.32	
Cambricon-F100	415,109,951		42,873.06	

Table 8: Hardware characteristics comparison.

Chip	Cam-F1	Cam-F100	1080Ti	V100	DaDN [37]	TPU [40]
ISA type	FISA	FISA	SIMD	SIMD	VLIW	CISC
Technology	45nm	45nm	16nm	12nm	28nm	28nm
Type	Cam-F	Cam-F	GPU	GPU	ASIC	ASIC
Memory type	eDRAM	eDRAM	SRAM	SRAM	eDRAM	SRAM
Memory Size	16 MB	448 MB	12.8 MB	33.5 MB	36 MB	28 MB
Peak Perf. (Tops)	14.9	119	10.6	125	5.58	92
Area (mm^2)	29	415	471	815	67	($\leqslant$ 331)
Power (W)	4.94	42.87	-	-	15.97	40
Power efficiency (Tops/W)	3.02	2.78	-	-	0.35	2.3
Area efficiency (Tops/mm^2)	0.51	0.29	0.02	0.15	0.08	0.28
Card	Cam-F1	Cam-F100	1080Ti	V100	DaDN	TPU
Dies	1	2	1	1	-	1
DRAM size	32 GB	32 GB	11 GB	16 GB	-	8 GB
Peak Perf. (Tops)	14.9	238	10.6	125	-	92
Power (W)	90.19	167.22	199.90	248.32	-	-

used. GPU performance does not always increase with batch size, which caused the best batch size choosing on GPU is smaller than on Cambricon-F. The broadcasting optimization of Cambricon-F improved operational intensity even further.

On machine learning tasks, DGX-1 achieves up to 85x higher operation intensity when compared Cambricon-F100. This difference is caused by the implicit management of intermediate memory in Cambricon-F. In Cambricon-F, programmers do not manipulate on memories except the main memory explicitly, Cambricon-F will write the intermediate result after each instruction back to the root once the tensor transposition mechanics failed to forward the data, which caused the traffic on root raised. For control intensive workloads as ML tasks in the benchmark, control flow always breaks the FISA pipeline and data forwarding, forcing the intermediate results written back to the root. K-NN and SVM have a relatively complete essential computation block. For K-NN, calculating distances between each pair of samples constituted $\geqslant 95\%$ of the total runtime, and for SVM, the kernel between each pair of samples, which is sufficiently operation-intensive, is calculated in each iteration. Thus, their operational intensity on Cambricon-F is less affected. K-MEANS and LVQ are also iterative algorithms as SVM is, but they do not have an operation-intensive computation block in each iteration, thus their operational intensity is more affected, which heavily limited the performance attainable. Moreover, the significantly smaller granularities of operations on these two benchmarks may be insufficient to hide the control latency of Cambricon-F nodes, resulting in an even worse performance on Cambricon-F100 compared to Cambricon-F1. With such a better operational intensity, DGX-1 has still shown a significant gap between attained performance and the roofline, since the bottleneck of GPU system is between graphic memories and chips. For K-MEANS and LVQ, GPU suffers from the control flow either and showing an even worse performance.

7. DISCUSSION

GPU. GPUs have been the most commonly used accelerators for machine learning workloads for their high performance. However, the slowly increasing bandwidth started to limit the performance of GPUs from five years ago. Figure 16 shows the sustained growth of memory bandwidth and the number of cores for NVIDIA GPUs. While the growth rate of memory bandwidth still maintains about 15% annually for last 10 years, the growth rate of the number of cores has dramatically reduced—67.6% per year during 2009 to 2013 and 8.8% per year for last 5 years. GPUs can not simply benefit from increasing peak performance (i.e., the number of cores) for memory intensive workloads due to the bandwidth limitation. In a GPU, all cores are designed to retrieve data directly from graphics memory. In the scenario that two neighboring cores require the same data, GPU will transfer that data twice on the graphics memory bus inefficiently. To help reduce the duplicated memory accesses, architects have put caches and memory compression mechanics on GPU [53, 54]. But it did not solve the problem fundamentally.

Unlike GPU, Cambricon-F organized cores in a fractal way. The intermediate nodes in Cambricon-F will serve as agents of leaf cores, and combine memory requests from leaf cores as a greater request, eliminating duplicated data traffics. By analysis of memory bounded operational intensity, Cambricon-F reduces 73.4%∼98.8% of the memory traffic between DRAM and chips when compared to graphics memory traffic in GPU. In other words, Cambricon-F has better scalability.

Comparing with ASIC accelerators. TPU adopted a systolic architecture for deep learning workloads and thus decomposed the communications in a systolic manner, which evenly distributed among PEs. With a systolic array, one can also build a fractal

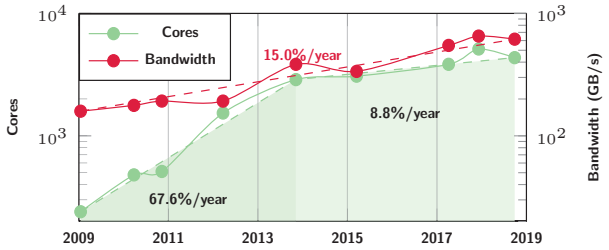


Figure 16: Growth in cores and bandwidth of NVIDIA GPUs since 2009.

computer. However, the versatility is the major consideration that we did not choose systolic fractal-computer in this paper. Despite various machine learning techniques, Cambricon-F is able to support any operation as long as it can be resolved as fractal operations.

DaDianNao placed all data involved in the computation in local memory where data is transmitted onto the chip only once. A similar effect happens in Cambricon-F when data fits in local memory, where duplicated global memory accesses are replaced by local memory accesses by tensor transposition. But unlike DaDianNao that its workload size is strictly limited by its memory size, Cambricon-F is still able to support workload in any size with fractal decomposition.

8. RELATED WORK

Machine learning accelerators. Due to the end of Moore's Law and Dennard Scaling, domain-specific accelerators designed for machine learning, especially DNNs, have become hot topics of computer architecture community in recent years. Many machine learning workloads have high intrinsic parallelism to be exploited by specific architecture. 15% of all papers on ISCA'16 are relevant to DNN accelerators and increased further to 26% on ISCA'18. Most recent works are included in [55–61].

Yunji Chen et al. proposed the DianNao family of machine learning accelerators [21, 22, 37–39], which minimizes memory accesses to achieve both high performance and low power consumption. Yu-Hsin Chen et al. proposed Eyeriss [23] accelerator for deep CNNs which adopts a reconfigurable data path and running-length compression to skip zeros in the data, both to minimize memory access. Google's TPU [40] adopts a systolic array of PEs as its computing component to eliminate the requirement of local memory on PEs. Many previous works have shown that minimizing memory accesses is essential for machine learning accelerators, but have not quantized the effects of their efforts to reduce memory accesses.

Machine learning computers. Akhil Arunkumar et al. proposed MCM-GPU [62] to continue the scalability of monolithic GPU. By designing memory system and integration, MCM-GPU proposed a multi-chip module of GPUs with interconnections and caches showing that the performance of a multilayered GPU system can be comparable to a similarly sized, monolithic GPU. Both MCM-GPU and Cambricon-F provided a user-transparent extension to system scalability. Compared to Cambricon-F100 which also has a similar module—a computing card composing two chips, the control of MCM-GPU is fine-grained and heterogeneous while Cambricon-F100 remained homogeneous.

As the state-of-the-art GPU system, DGX-1 [15] was originally launched by NVIDIA in 2016 featuring eight NVIDIA Tesla P100 GPUs, then refreshed with new NVIDIA Tesla V100 GPUs which are particularly designed for deep learning acceleration. Compared

to Cambricon-F100, the eight GPUs in DGX-1 are connected in a hybrid cube mesh network by NVLink, while the interconnection of Cambricon-F100 nodes is limited within parent-to-children paths, forming an H-tree topology. Building interconnection among sibling nodes for Cambricon-F may further improve performance, we left this exploration for future works.

ISA for heterogeneous systems. Recent research also address the programming productivity issue with new ISA. Venkat et al. [63] proposed Composite-ISA which constitutes a composite ISA superset with multi-ISA for heterogeneous multicores, while Cambricon-F uses a unified ISA for multi systems with different scales.

9. CONCLUSION

In this paper, we propose Cambricon-F, machine learning computers with fractal von Neumann architecture and the same ISA, aiming to address the emerged critical issue that hinders the deployment of machine learning computers, i.e., programming productivity, including both programming itself and software stack development. We thoroughly analyze machine learning techniques for fractal computation and solve the three different types of fractal operation in our Cambricon-F architecture design. Cambricon-F features the fractal computing that iteratively decomposes an instruction on it into several instructions on low-layer sub-nodes. Thus, achieving easy-programming and high-efficiency simultaneously. Our results show that Cambricon-F achieves 5.14x, 2.82x better performance, 11.39x, 8.37x better efficiency on average, with 93.8%, 74.5% smaller area costs when comparing against 1080Ti and V100 GPU, respectively. With the unified ISA and code for high programming productivity, Cambricon-F is also able to achieve better performance and efficiency.

10. ACKNOWLEDGEMENT

This work is partially supported by the National Key Research and Development Program of China (under Grant 2017YFA0700900, 2017YFA0700902, 2017YFA0700901, 2017YFB1003101), the NSF of China (under Grants 61432016, 61532016, 61672491, 61602441, 61602446, 61732002, 61702478, 61732007 and 61732020), Beijing Natural Science Foundation (JQ18013), the 973 Program of China (under Grant 2015CB358800), National Science and Technology Major Project (2018ZX01031102), the Transformation and Transfer of Scientific and Technological Achievements of Chinese Academy of Sciences (KFJ-HGZX-013), Key Research Projects in Frontier Science of Chinese Academy of Sciences (QYZDB-SSW-JSC001) and Strategic Priority Research Program of Chinese Academy of Science (XDB32050200, XDC01020000).

11. REFERENCES

[1] Google Inc., "Cloud vision: Derive insight from your images with our powerful pretrained API models or easily train custom vision models with AutoML Vision." https://www.ibm.com/thought-leadership/summit-supercomputer/.

[2] B. Jiang, R. Luo, J. Mao, T. Xiao, and Y. Jiang, "Acquisition of localization confidence for accurate object detection," *Lecture Notes in Computer Science*, p. 816–832, 2018.

[3] A. Krizhevsky, G. E. Hinton, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," tech. rep., 2012.

[4] Google Inc., "Cloud speech-to-text: Speech-to-text conversion powered by machine learning and available for short-form or long-form audio." https://cloud.google.com/speech-to-text/.

[5] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[6] Amazon, "Easily recognize famous individuals and celebrities using Amazon Rekognition." https://console.aws.amazon.com/rekognition/home.

[7] E. Zhou, Z. Cao, and J. Sun, "Gridface: Face rectification via learning local homography transformations," *Lecture Notes in Computer Science*, p. 3–20, 2018.

[8] Google Inc., "CLOUD VIDEO INTELLIGENCE: Search and discover your media content with Cloud Video Intelligence." https://cloud.google.com/video-intelligence/.

[9] T. Mei and C. Zhang, "Deep learning for intelligent video analysis," October 2017.

[10] S. Chaudhuri, G. Theocharous, and M. Ghavamzadeh, "Personalized advertisement recommendation: A ranking approach to address the ubiquitous click sparsity problem," *CoRR*, vol. abs/1603.01870, 2016.

[11] T. Mahlmann, A. Drachen, J. Togelius, A. Canossa, and G. N. Yannakakis, "Predicting player behavior in Tomb Raider: Underworld," in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pp. 178–185, Aug 2010.

[12] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, F. Hui, L. Sifre, G. V. D. Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, 2017.

[13] Cambricon, "Cambricon 1H provides strong AI computing in Huawei Kirin 980."

[14] Apple Inc., "Get Ready for Core ML 2." https://developer.apple.com/machine-learning/.

[15] NVIDIA Corporation, "NVIDIA Tesla V100 GPU Architecture," 2018. https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.

[16] NVIDIA Corporation, "NVIDIA DGX-2H," 2018. https://www.nvidia.com/content/dam/en-zz/es_em/Solutions/Data-Center/dgx-2/dgx-2h-datasheet-us-nvidia-841283-r6-web.pdf.

[17] Google Inc., "What makes TPUs fine-tuned for deep learning?," 2018. https://cloud.google.com/blog/products/ai-machine-learning/what-makes-tpus-fine-tuned-for-deep-learning.

[18] IBM, "The most powerful computers on the planet." https://www.ibm.com/thought-leadership/summit-supercomputer/.

[19] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "NeuFlow: A runtime reconfigurable dataflow processor for vision," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 109–116, Ieee, jun 2011.

[20] S. Venkataramani and V. Chippa, "Quality programmable vector processors for approximate computing," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, no. i, pp. 1–12, 2013.

[21] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems (ASPLOS)*, (Salt Lake City, UT, USA), pp. 269–284, 2014.

[22] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. Chen, "PuDianNao : A Polyvalent Machine Learning Accelerator," in *Proceedings of the 20th international conference on Architectural support for programming languages and operating systems (ASPLOS)*, pp. 369–381, 2015.

[23] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 367–379, 2016.

[24] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "Envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable Convolutional Neural Network processor in 28nm FDSOI," *IEEE International Solid-State Circuits Conference*, vol. 60, pp. 246–247, 2017.

[25] A. Biswas and A. P. Chandrakasan, "Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications," in *IEEE International Solid-State Circuits Conference*, vol. 61, pp. 488–490, 2018.

[26] NVIDIA Corporation, "Parallel Thread Execution ISA Version 6.2," 2018. https://docs.nvidia.com/cuda/parallel-thread-execution/index.html.

[27] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: a system for large-scale machine learning.," in *OSDI*, vol. 16, pp. 265–283, 2016.

[28] Huawei, "Huawei Launches HiAI 2.0, Commits to Creating the Ultimate AI App Experience." https://www.huawei.com/en/press-events/news/2018/11/huawei-hiai-2-ultimate-ai-app-experience.

[29] W. Sierpiński, "Sur une courbe cantorienne qui contient une image biunivoque et continue de toute courbe donnée," 1916.

[30] M. T. Barlow and R. F. Bass, "The construction of brownian motion on the sierpinski carpet," *Ann. Inst. H. Poincaré*, vol. 25, no. 1989, pp. 225–257, 1989.

[31] W. Wzr, V. Surfhvv, L. V. Ghsor, H. G. Rq, K. Hqg, D. Rq, P. D. Q. Fkdoohqjlqj, P. Ohduqlqj, H. J. L. W. Wdnhv, W. Zhhnv, W. R. Wudlq, R. Q. Irxu, and K. Hqg, "Towards Pervasive and User Staisfactory CNN across GPU Microarchitecture," in *Proceedings of The 23rd IEEE Symposium on High Performance Computer Architecture (HPCA)*, 2017.

[32] X. Zhang, C. Xie, J. Wang, W. Zhang, and X. Fu, "Towards Memory Friendly Long-Short Term Memory Networks (LSTMs) on Mobile GPUs," in *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, vol. 1537085, 2018.

[33] P. Hill, A. Jain, M. Hill, B. Zamirai, C.-H. Hsu, M. A. Laurenzano, and S. Mahlke, "DeftNN: Addressing Bottlenecks for DNN Execution on GPUs via Synapse Vector Elimination and Near-compute Data Fission," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 786–799, 2017.

[34] J. Park, H. Sharma, D. Mahajan, J. K. Kim, P. Olds, and H. Esmaeilzadeh, "Scale-out acceleration for machine learning," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 367–381, 2017.

[35] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN Accelerator Efficiency Through Resource Partitioning," in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA'17)*, pp. 535–547, 2017.

[36] T. Chen, S. Srinath, C. Batten, and G. E. Suh, "An Architectural Framework for Accelerating Dynamic Parallel Algorithms on Reconfigurable Hardware," in *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, no. 2, 2018.

[37] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A Machine-Learning Supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-47)*, pp. 609–622, 2015.

[38] Y. Chen, T. Chen, X. Zhiwei, and O. Temam, "DianNao Family: Energy-Efficient Hardware Accelerators for Machine Learning," *Communications of the ACM*, vol. 57, no. 5, p. 109, 2014.

[39] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting Vision Processing Closer to the Sensor," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pp. 92–104, 2015.

[40] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-L. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. Mackean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA'17)*, pp. 1–17, 2017.

[41] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[42] S. Liu, Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, and T. Chen, "Cambricon : An Instruction Set Architecture for Neural Networks," 2016.

[43] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," in *Deep Learning and Unsupervised Feature Learning Workshop, Neural Information Processing Systems Conference (NIPS)*, 2011.

[44] H. Esmaeilzadeh, P. Saeedi, B. Araabi, C. Lucas, and S. Fakhraie, "Neural Network Stream Processing Core (NnSP) for Embedded Systems," in *2006 IEEE International Symposium on Circuits and Systems (ISCS)*, pp. 2773–2776, Ieee, 2006.

[45] X. Sun and L. Ni, "Scalable problems and memory-bounded speedup," *Journal of Parallel and Distributed Computing*, vol. 19, no. 1, pp. 27 – 37, 1993.

[46] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, pp. 65–76, Apr. 2009.

[47] P. Flajolet and M. Golin, "Exact asymptotics of divide-and-conquer recurrences," in *Automata, Languages and Programming* (A. Lingas, R. Karlsson, and S. Carlsson, eds.), (Berlin, Heidelberg), pp. 137–149, Springer Berlin Heidelberg, 1993.

[48] M. Poremba, S. Mittal, D. Li, J. S. Vetter, and Y. Xie, "DESTINY: A tool for modeling emerging 3D NVM and eDRAM caches," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference*, DATE '15, (San Jose, CA, USA), pp. 1543–1546, EDA Consortium, 2015.

[49] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[50] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[51] NVIDIA Corporation, "CUDA Toolkit Documentation v9.0.176," 2018. https://docs.nvidia.com/cuda/archive/9.0/.

[52] NVIDIA Corporation, "NVIDIA Deep Learning SDK," 2018. https://docs.nvidia.com/deeplearning/sdk/index.html.

[53] V. Sathish, M. J. Schulte, and N. S. Kim, "Lossless and lossy memory I/O link compression for improving performance of GPGPU workloads," in *Parallel Architectures and Compilation Techniques (PACT), 2012 21st International Conference on*, pp. 325–334, IEEE, 2012.

[54] NVIDIA Corporation, "NVIDIA Tesla P100," 2017. https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf.

[55] P. Srivastava, M. Kang, S. K. Gonugondla, S. Lim, J. Choi, V. Adve, N. S. Kim, and N. Shanbhag, "PROMISE: An end-to-end design of a programmable mixed-signal accelerator for machine-learning algorithms," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 43–56, June 2018.

[56] F. Tu, W. Wu, S. Yin, L. Liu, and S. Wei, "RANA: Towards efficient neural acceleration with refresh-optimized embedded DRAM," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 340–352, June 2018.

[57] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 383–396, June 2018.

[58] K. Hegde, J. Yu, R. Agrawal, M. Yan, M. Pellauer, and C. Fletcher, "UCNN: Exploiting computational reuse in deep neural networks via weight repetition," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 674–687, June 2018.

[59] V. Akhlaghi, A. Yazdanbakhsh, K. Samadi, R. K. Gupta, and H. Esmaeilzadeh, "SnaPEA: Predictive early activation for reducing computation in deep convolutional neural networks," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 662–673, June 2018.

[60] E. Park, D. Kim, and S. Yoo, "Energy-efficient neural network accelerator based on outlier-aware low-precision computation," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 688–698, June 2018.

[61] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 764–775, June 2018.

[62] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C. Wu, and D. Nellans, "MCM-GPU: Multi-chip-module GPUs for continued performance scalability," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 320–332, June 2017.

[63] A. Venkat, H. Basavaraj, and D. M. Tullsen, "Composite-ISA Cores: Enabling Multi-ISA Heterogeneity Using a Single ISA," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 42–55, Feb 2019.