Machine Learning Computers with Fractal von Neumann Architecture

Yongwei Zhao, Zhe Fan, Zidong Du, *Member, IEEE,* Tian Zhi, Ling Li, Qi Guo *Member, IEEE,* Shaoli Liu, Zhiwei Xu *Senior Member, IEEE,* Tianshi Chen and Yunji Chen *Senior Member, IEEE*

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 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. We further propose Cambricon-FR, which enhances the Cambricon-F machine learning computers to flexibly and efficiently support all the fractal operations with a reconfigurable fractal instruction set architecture. Compared to the Cambricon-F instances, Cambricon-FR machines achieve 1.96x, 2.49x better performance on average. Most importantly, Cambricon-FR computers are able to save the code length with a factor of 5.83, thus significantly improving the programming productivity.

Index Terms—Machine Learning, Architecture, Neural Networks, Programming Efficiency

1 Introduction

Machine learning techniques are pervasive tools for emerging commercial applications, including image recognition [1], [2], [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].

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 [20], 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 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

• *Yongwei Zhao, Zhe Fan, Zidong Du, Tian Zhi, Qi Guo, Shaoli Liu, Zhiwei Xu, Tianshi Chen and Yunji Chen are with State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, 100190. Yongwei Zhao, Zhe Fan, Zhiwei Xu and YunjiChen are also with University of Chinese Academy of Sciences, Beijing, 100049. Yongwei Zhao, Zhe Fan, Zidong Du, Tian Zhi, Qi Guo, Shaoli Liu, and Tianshi Chen are also with Cambricon Technologies. Ling Li is with Institute of Software, Chinese Academy of Sciences, Beijing, 100190. Yunji Chen is also with Institute of Brain-Intelligence Technology, Zhangjiang Laboratory (BIT, ZJLab), Shanghai Research Center for Brian Science and Brain-Inspired Intelligence (Shanghai Brain/AI), CAS Center for Excellence in Brain Science and Intelligence Technology (CEBSIT). Zidong Du is the corresponding author.*
Emails: {zhaoyongwei, fanzhe, duzidong, guoqi, liushaoli, zxu, chentianshi, cyj}@ict.ac.cn; liling@iscas.ac.cn.

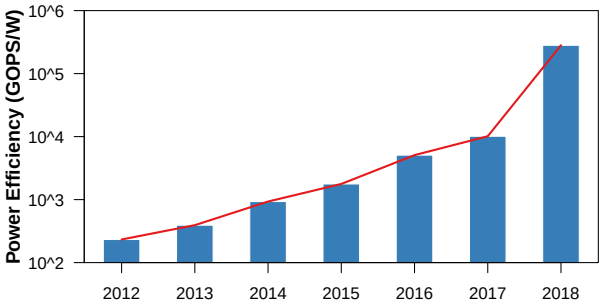

Fig. 1. Power efficiency of recent proposed machine learning accelerators [19], [20], [21], [22], [23], [24], [25].

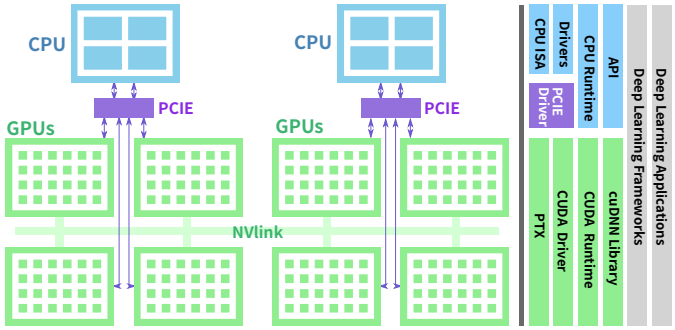

Fig. 2. A typical machine learning computer architecture.

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?

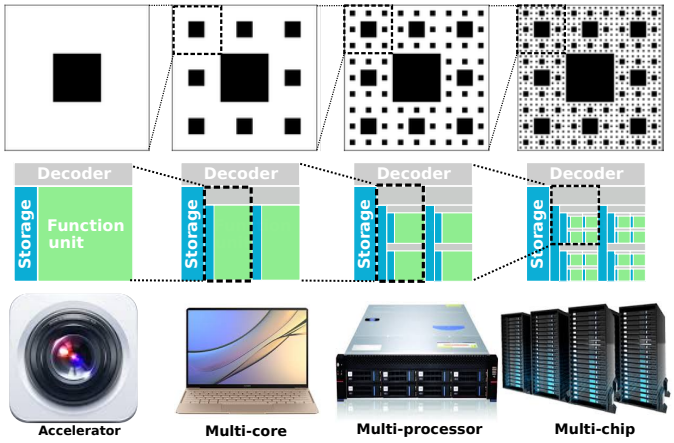

Fig. 3. *Top:* A fractal graph example: Sierpinski carpet [30]. The graph is subdividing itself into smaller copies and continuing recursively. *Botttom:* Fractal computers, analogy to Sierpinski carpet.

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 diving 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. Furthermore, we propose a reconfigurable FISA for fractal machine, which allows user-defined fractal instructions and user-specified executing procedures, to flexibly and efficiently support all the fractal operations. We further propose the Cambricon-FR machine learning computers to architectural support the reconfigurable FISA.

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.
- We analyze the ineffectiveness in FISA and propose Cambricon-FR, which leverages a reconfigurable fractal instruction set architecture to efficiently and flexibly support all fractal operations.
- We evaluate two Cambricon-FR instances, i.e., Cambricon-FR1 and Cambricon-FR100. Compared to GPU based machines, Cambricon-FR machines achieve 34.48x, 25.69x better performance on average. Most importantly, Cambricon-FR computers is able to save the code length with a factor of 5.83, thus significantly improving the programming productivity.

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], [32], [33], FPGA [34], [35], [36], and even ASIC chips [23], [37], [38], [39], [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

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	-	94.7%	0.18%	5.02%	0.12%	-	
DNN	-	-	-	99.9%	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%	-	-

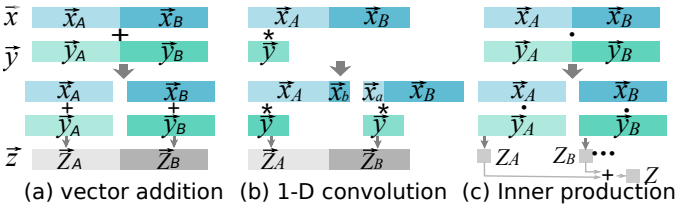

Fig. 4. Fractal operation dependency: (a) independent; (b) input dependent; (c) output dependent.

(DNNs). Others are k-means, k-NN, support vector machine (SVM), and learning vector quantization (LVQ). In line with previous works [23], [42], [43], [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.

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,

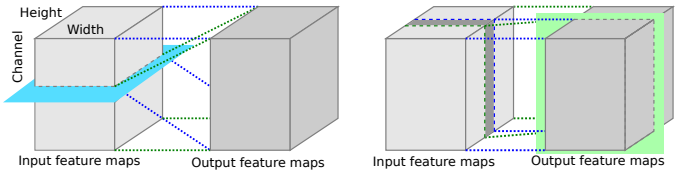

Fig. 5. *CONV* decomposition. *Left:* dividing in channel dimension. *Right:* dividing in height dimension.

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 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

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	-

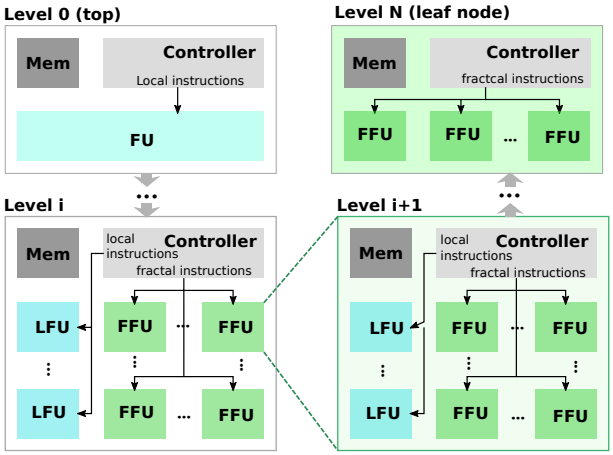

Fig. 6. A typical fractal von Neumann architecture: level 0 (top node)...level i node and its son node in level $i+1$...level N (leaf node).

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).
- **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).

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 that 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, 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 $\mathcal{G}'_1, \mathcal{G}'_2, \ldots, \mathcal{G}'_n$ ($\mathcal{G}'_i \preceq \mathcal{G}$, $\preceq$ is the partial order defined on scale indicators) that I can be achieved through computing with $I'_1(\mathcal{G}'_1), I'_2(\mathcal{G}'_2), \ldots, I'_n(\mathcal{G}'_n)$ 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 is a fractal machine, *iff* there exists at least one fractal instruction 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. 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.

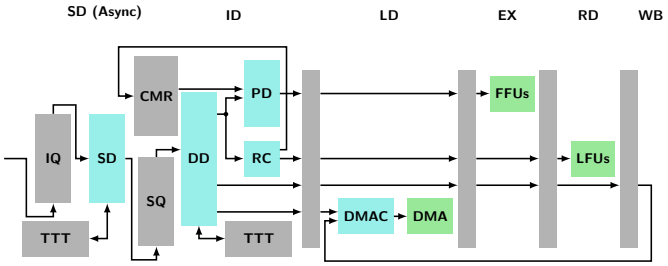

Fig. 7. Pipeline partition in an Cambricon-F node.

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

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

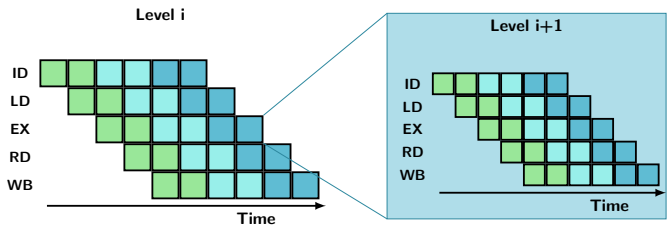

Fig. 8. Fractal Pipeline of FISA.

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 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.

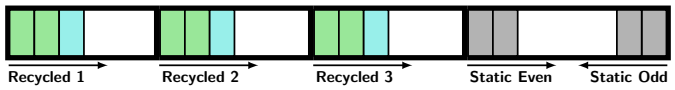

Fig. 9. Memory Management of Cambricon-F Controller. The memory space is divided into 4 segments (3 recycling and 1 static), managed as 5 stacks (2 stacks in the static segment).

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 (TTT)* to find out data that can be forwarded or reused, resulting in a similar behaviour as "pipeline forwarding" in common practices of processor pipelines.

3.6 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, Tensor Transposition Table 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 for each record in TTT. TTT is split into banks as memory spaces does, where each bank only maintains the information of its corresponding memory segment. Whenever the segment is recycled, new data are allocated on, and old data may be overwritten thus never safe to forward again, the TTT invalidates all records. Thus, with such validity mechanism, the lifetime of records won't exceed the lifetime of the referenced data. 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.

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 10, we show a typical Cambricon-F inline assembly code using a k-Nearest Neighbor algorithm as a driving example. The principle of FISA is 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

k-Nearest Neighbors

```
int K = 5, 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]
```

Fig. 10. An Cambricon-F program of k-NN.

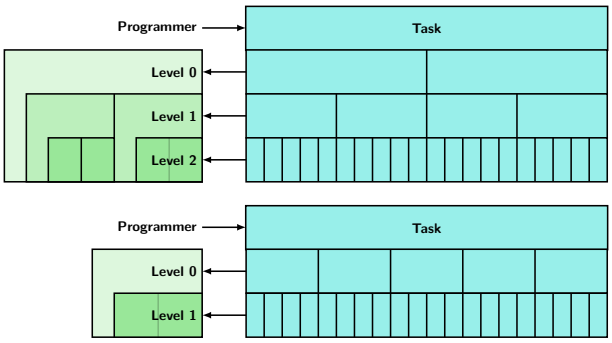

Fig. 11. STMH execution model.

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 11, 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.

5 Methodology

Benchmarks. As shown in Table 4, we use seven different benchmarks in this paper. For the importance of deep learning, we select VGG-16 [45], a 16-layer CNN with 138 M parameters in total, and ResNet-152 [46], a very deep network with 152 layers, running with ImageNet [41] dataset as representative benchmarks. We also 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 [47] and cuDNN 7 [48]), and optimize the computation graph via NVIDIA TensorRT 4 [48]. 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 5. At the top level (L0), Cambricon-F100 contains 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

TABLE 4
Benchmarks.

Benchmark	Size
VGG-16 [45]	1.38×10^8 params, 3.09×10^{10} Ops, variable batch
ResNet-152 [46]	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

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 5. 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 [49], 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 12, where we adopt the Roofline Model [50] 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 13. In Table 6, we present the detailed hardware characteristics of the chip in Cambricon-F100 and Cambricon-F1. Cambricon-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-F100, 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 large memory.

In Table 7, 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 12 (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. 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 12 (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 opera-

TABLE 5
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		Card		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

TABLE 6
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 7
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	-	-

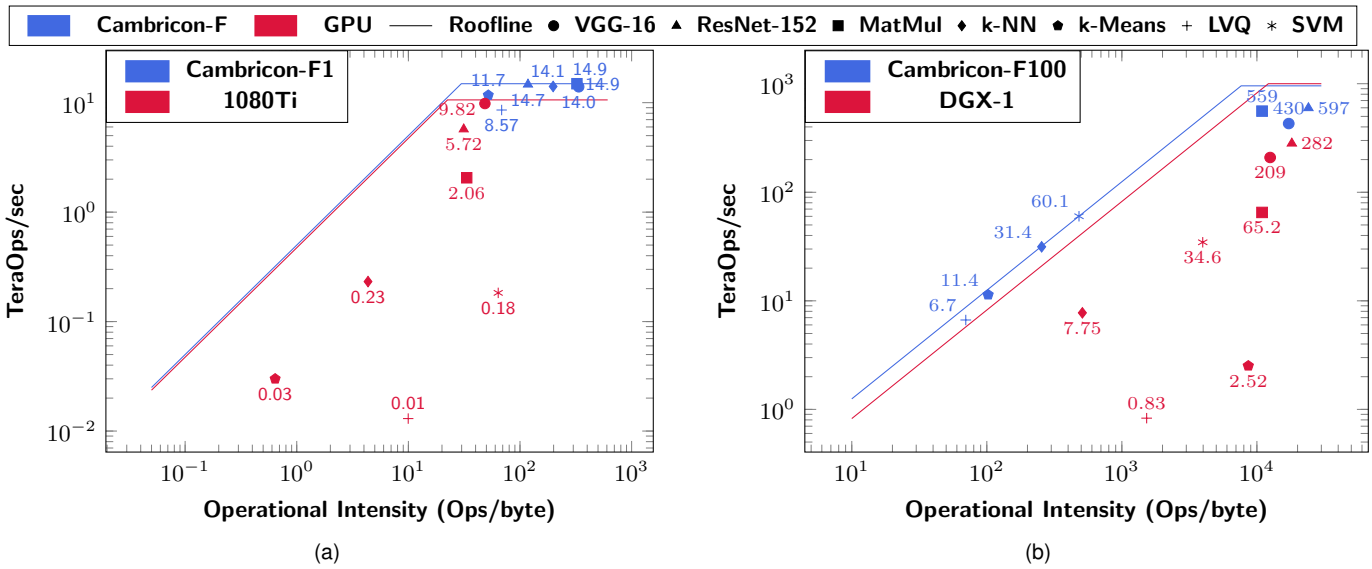

Fig. 12. Roofline Cambricon-Fs compared to GPUs. (a) Cambricon-F1 and 1080Ti. (b) Cambricon-F100 and DGX-1.

tional 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 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 run-time, 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.

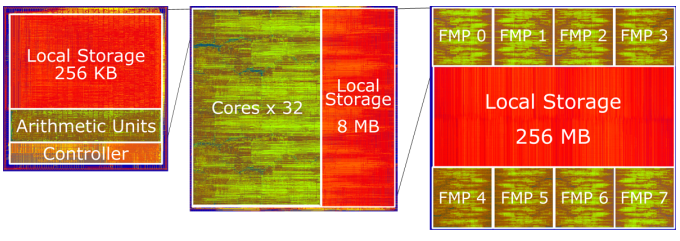

Fig. 13. Layout of Cambricon-Fs. Left: Leaf *Core*. Mid: *FMP*(Cambricon-F1 Chip). Right: Cambricon-F100 Chip.

7 Cambricon-FR: Fractal Computers with a Reconfigurable FISA

In this section, we propose *Cambricon-FR*, which enhanced the Cambricon-F machine learning computers to flexibly and efficiently support all the operations with a *reconfigurable* fractal instruction set architecture (RFISA).

7.1 Ineffectiveness in FISA

While Cambricon-F is able to achieve high programming efficiency with maintained higher performance and energy efficiency, it still suffers from *ineffectiveness* when executing new operations that are not covered in the FISA set. In the FISA set (c.f., Section 3.2), several important operations from popular machine learning techniques are contained as FISA primitives directly for fast and efficient implementation. But other operations that are not FISA primitives can only be emulated through combining low-level operations and FISA primitives, which could be very *ineffective* in realization.

From the perspective of complexity, we can define two types of *ineffectiveness* in Cambricon-F, i.e., *computation ineffectiveness* (time complexity) and *communication ineffectiveness* (communication complexity). More precisely, regarding the *computation ineffective*, for a fractal operation which is *effective* on a fractal machine M, its obtained speedup ratio r_c is irrelevant to the operation granularity $\mathcal{G}$, where the r_c is defined as the ratio of operation compute time on the leaf node of M and on the whole machine M, i.e., $r_c = T_M / T_{leaf\ node}$. For an operation is *ineffective* on a fractal machine, its achieved speedup ratio is *asymptotically* related to the granularity $\mathcal{G}$, which means its computational complexity could become worse due to the non-direct support in FISA. For example, *TopK*, an operation not in FISA, could be very *ineffective* on Cambricon-F machines. As *TopK* can be support indirectly through the combination of Sort1D and Merge1D

primitives in FISA, solving a $TopK(n,K)$ is $O(n\log n)$ complexity in time. However, running a leaf node, it becomes $O(n\log K)$ complexity in time.

Regarding the *communication ineffectiveness*, for a fractal operation which is *effective* on a fractal machine M, its data traffic ratio r_d is irrelevant to the operation granularity $\mathcal{G}$, where the r_d is defined as the ratio of data communications required to execute on the whole machine M and on the leaf node of M, i.e., $r_d = D_M/D_{leaf\ node}$ where D denotes data traffics. For example, *Conv3D*, which is a convolutional operation with a pair of additional spatial dimensions D and K_D, could be very *communication ineffective* if achieving indirectly using CV2D primitive in FISA. While the time complexity remains the same, the lower-bound of data traffic of such *communication ineffective* realization could be $O(K_D)$ times higher than direct *Conv3D* primitive, as there will be at least $K_D - 1$ partial sum of results created in the root memory as intermediate data.

The two types of *ineffectiveness* at root are caused by the indirect support in FISA. For those *ineffective* operations, their execution on Cambricon-F machines are heavily limited by combining fragmented low-level instructions, which have to be executed sequentially. Therefore, to address the *ineffectiveness* issues in fractal machines, FISA should be able to support as many as possible operations. However, directly integrated all the possible operations is unrealistic and costly. Hence the question is, how to break the constraints of FISA and enable the fractal execution of any fractal operations on a fractal machine flexibly?

Our solution. To address the *ineffectiveness* issue, we propose a *reconfigurable* FISA (RFISA) for fractal machines, which allows user-defined fractal instructions and user-specified executing procedures. Therefore, the fragmented low-level instructions are fused again in RFISA, enabling effective support for those *ineffective* operations. In order to realize a fractal machine for RFISA, several challenges must be addressed:

- **FISA.** The RFISA needs to be flexible but still keep the programming productivity.
- **Architectural Support.** The corresponding architecture should be able to support RFISA but still keep the homogeneous characteristic of control logic in each node, so as to maintain the fractal feature.
- **Programming Paradigm.** A special designed programming model and a language is required, which should unify the programming of both Sequential Decomposer and Parallel Decomposer/Reduction Controller.

7.2 Reconfigurable Fractal Instruction Set Architecture

Figure 14 compares the FISA, RFISA, and traditional ISAs (e.g., VLIW and RISC) in terms of the abstract level (which is decisive to *productivity*) and *flexibility*.

The topmost FISA can provide excellent productivity since it directly maps high-level primitives (e.g., convolution and sorting) to fractal instructions. However, other operations (which cannot be directly mapped to one single pre-defined fractal instruction) need to be programmed with multiple sequentially executed fractal instructions for flexibility. As the original operation is executed with multiple FISA instruction in sequence, all other nodes, except for the root node, are not aware of what the original operations is. This leads to the waste of computations and lots of chances for data reuse. Moreover, nowadays machine learning algorithms are evolving drastically with newly emerged high-level primitives. Once current fractal instructions in FISA cannot emulate a new operation, the only way to support that operation is to update the instruction set, design and manufacture new machines with an updated instruction set.

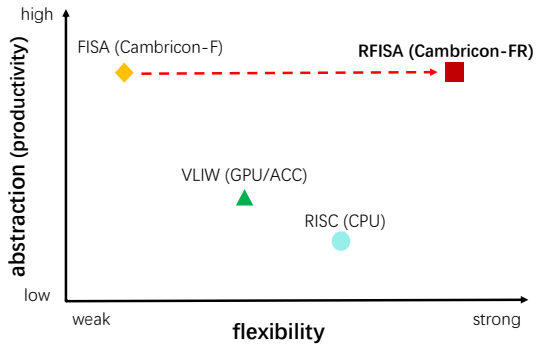

Fig. 14. The comparison of FISA, RFISA, VLIW, and RISC in terms of abstraction and flexibility. The RFISA can achieve both good productivity and flexibility.

RFISA dominates the FISA and VLIW/RISC in Figure 14, in terms of productivity and flexibility. In contrast to the FISA, RFISA does not provide native fractal instructions any more. Instead, a set of local instructions (same as in the FISA) are provided as the basic building blocks. Table 8 lists examples of local instructions. Based on such local instructions, for a new operation, programmers are able to build a corresponding new fractal instruction on demand with essential hardware supports (which will be elaborated in Section 7.3). Therefore, the desired operation is mapped to one single fractal instruction, instead of multiple sequentially executed fractal instructions in FISA, so as to improve the efficiency. In contrast to the VLIW and RISC instruction sets, which are more closer to the native computation ability of underlying hardware with low level of abstraction[1], RFISA can improve the productivity as high-level primitives can be expressed as one single fractal instruction. Moreover, the flexibility of RFISA is even better than VLIW/RISC for two aspects. First, the diversity of local instruction (i.e., four categories including data transfer, computational, logical and miscellaneous all operates with both vectors and scalars) offers the opportunity to compose various complicated operations. Second, from the programmers' perspective, a reconfigurable and customized instruction set can extend existing instruction set without updating the hardware, which is very flexible for adapting to different applications.

In practice, the original fractal instructions provided in FISA can also be pre-configured to RFISA, so that a machine with RFISA can be fully compatible with the programs of FISA.

7.3 Cambricon-FR Architecture

In this section, we present the architecture support for RFISA, i.e., the Cambricon-FR machine learning computers. Overall, Cambricon-FR maintains almost the same architecture as Cambricon-F, except the controller modules where the controller in Cambricon-FR is *reconfigurable*. In Figure 15, we show the two controllers in both Cambricon-F and Cambricon-FR. The controller in Cambricon-F contains a Sequential Decomposer (SD), a Parallel Decomposer (PD), a Demotion Decoder (DD), and a Reduction Controller (RC). Cambricon-FR replaces the SD, PD, and RC with a new module (a *reconfigurable DEC*). The *DEC* controls the execution of user-defined fractal instructions following the pre-loaded user-specified executing procedures in its storage.

1. For example, many RISC instructions are directly mapping of native functional units (e.g., ALU and FPU).

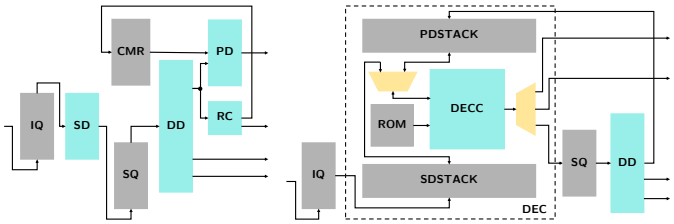

Fig. 15. Controller structure, *Left*: Cambricon-F. *Right*: Cambricon-FR.

Therefore, with the key component of *DEC*, Cambricon-FR is flexible to support any fractal operation effectively.

DEC. Figure 15 illustrates the detailed structure of DEC modules in Cambricon-FR and how it can be configured to perform the functionalities of SD and PD in Cambricon-F. The *DEC* contains a ROM, two stacks (PDSTACK and SDSTACK), and a controller (DECC). The ROM is used to store the user-specified executing procedures to be executed by DECC and the two data stacks are used to preserve the states when executing those procedures. The *DEC* deploys a double-thread executing model, where one thread performs the *sequential decomposing* (SD functionality) and the other performs the *parallel decomposing* (PD functionality). With a pre-defined thread priority, the DEC is able to perform one certain decomposition over the other. Additionally, since only the decomposing strategies are different in *sequential decomposing* and *parallel decomposing*, a user-specified executing procedure has many parts shared for the two threads, thus reducing the program size in ROM largely.

Dynamic control. One possible situation in Cambricon-FR is that some instructions need data only decided in runtime, i.e., dynamic control. For example, when decomposing sparse data into equal pieces where each piece contains roughly same number of non-zero data, POPCNT (*pop count*) instruction will be applied to find the exact size of each piece. As such sparse information can only be obtained in runtime, POPCNT instruction needs dynamic control support. Cambricon-FR supports such dynamic control by allowing a memory address field of some RFISA instructions. And the DEC applies a scoreboard mechanism to replace the fields with contents from corresponding memory addresses before every sequential decomposition. Hence the sub-instructions after sequential decomposition can be directly sent to sub-nodes in Cambricon-FR without requirement of dynamic control support.

7.4 Programming

DEFRACTALK programming model. To ease the burden of programming user-specified execution procedures for user-defined fractal instructions, we propose DEFRACTALK (*Decomposing Fractal Talk*), a domain-specific programming model for programming the Cambricon-FR DEC. The DEFRACTALK helps the programmers to define the execution flow of user-defined instructions but without bothering the programmers with too many details about the machine. In Figure 16, we show the programming flow of DEFRACTALK , where the programmers perform the left part light-weight tasks to tell the DEC how to perform the decomposing process. Specifically, the programmers declare the form of instruction with name and parameters where the DEFRACTALK provides instruction instances with the declared name of parameters. The programmers define several *decomposition options (opt)* and the DEFRACTALK tries its best to choose a decomposition option based on the hardware characteristics. The programmers define the range of *pivot value* which is used to decide the granularity of decomposed sub-instructions and the DEFRACTALK finds the best pivot value based on the hardware characteristics. The programmers provide a piece of imperative code which defines how to write out the decomposed sub-instructions based on the pivot value and instruction arguments given by the DEFRACTALK . Finally, the DEFRACTALK is able to generate sub-instructions for DEC to decompose the user-defined instructions recursively.

TABLE 8
Examples of Local Instructions

Type	Operation	Name
Data Transfer	Explicit Tensor Move	tmove
Computational	Element-wise Non-linear Transform Add (vector, scalar, mixture) Horizontal Maximum / Logical ANY Matrix Multiply	veltw vadd, sadd, vsadd hmax mmul
Logical	Logical Exclusive-OR Comparison Greater Than	vxor, sxor vgt, sgt
Miscellaneous	Generate Random Vector Population Count Merge Sorted Lists	vrng vpopcnt vmerge

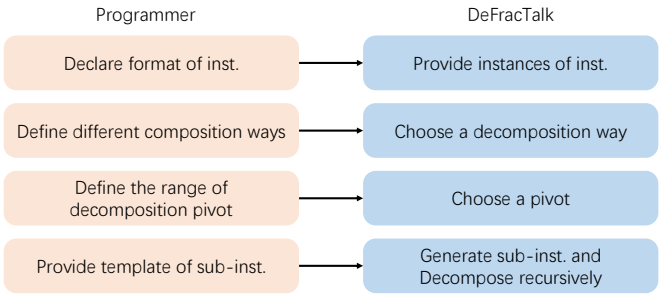

Fig. 16. DEFRACTALK programming flow.

The key feature of DEFRACTALK is that it simplifies the programming and hardware design by providing an interaction interface to separate the definition and the specific decomposing process, see Figure 16. For example, in the programming, programmers leave two key decisions, i.e., choosing the *opt* value and choosing the *pivot value*, to be made by DEFRACTALK based on the machine characteristics. In such way, DEFRACTALK achieves three the key advantages:

- **Independence of hardware configuration.** As the decomposition is largely decided by DEFRACTALK based on specific machine configuration information, programmers are able to define the same instruction but with different decomposition for different Cambricon-FR machines.
- **Independence of instruction granularity.** As the two key parameters are decided by the DEFRACTALK with regard to the machine characteristics, programmers are unaware of the detailed decomposition on internal nodes, thus providing an instruction granularity-independent programming experience.
- **Independence of SD or PD.** As the DEC is configured to perform the functionalities of SD and PD, the two types of decomposition run in a double-thread mode. Thus, programmers are unaware of which decomposing process is running.

DEFRACTALK Language. We propose a specialized programming language for DEFRACTALK programming, i.e., the DEFRACTAL *(DeFracTalk Language)*. In order to support the future emerging fractal operations, we design the DEFRACTAL strictly following the definition of fractal operation (given in

Section 2.2) and DEFRACTALK . We designed the syntax of DEFRACTAL based on the C programming language. A part of syntax rules are listed below:

⟨*translation-unit*⟩ ::= ⟨*rfisa-def*⟩ ⟨*translation-unit*⟩$_{\text{optional}}$

⟨*rfisa-def*⟩ ::= ⟨*rfisa-opcode*⟩ ⟨*rfisa-param-list*⟩ { ⟨*opt-def-list*⟩ }

⟨*pivot-decl*⟩ ::= ⟨*pivot*⟩ **:** [⟨*lower*⟩ **,** ⟨*upper*⟩]

⟨*opt-def*⟩ ::= ⟨*opt-specifier*⟩ ⟨*pivot-decl*⟩$_{\text{optional}}$ ⟨*comp-stmt*⟩

⟨*stmt*⟩ ::= ⟨*var-decl*⟩ | ⟨*comp-stmt*⟩ | ⟨*frac-stmt*⟩ | ⟨*reduce-stmt*⟩ | ⟨*sub-inst-stmt*⟩

⟨*frac-stmt*⟩ ::= **frac** ⟨*expr*⟩ **:** ⟨*expr*⟩ ⟨*stmt*⟩

⟨*reduce-stmt*⟩ ::= **reduce** ⟨*stmt*⟩

⟨*sub-inst-stmt*⟩ ::= ⟨*rfisa-opcode*⟩ ⟨*expr-list*⟩ **;**

The *rfisa-def* defines the instruction name (*rfisa-opcode*), parameters (*rfisa-param-list*), and a list of options (*opt-def*) where each *opt-def* defines a decomposition option. In practice, there are *opt*s that do not require a pivot to be defined, especially the last decomposition which generates all local instructions. *opt-def* contains a compound statement that corresponds to the imperative code piece defining the sub-instructions. There are three new statement types: *frac-stmt*, *reduce-stmt* and *sub-inst-stmt*, correspond to the target operator $f(\cdot)$, the retrieving operator $g(\cdot)$ as defined in Section 2.2, and the sub-instruction actually written out. With these correspondence, an operation can be defined in DEFRACTAL as long as it meets the formal definition of fractal operation. In addition, DEFRACTAL provided *opt-specifier* to specify whether the current defining *opt* is suitable in the context of specific decomposing phases, although DEFRACTALK does not differentiate SD and PD. This is added for performance optimizing and simplicity of compiler design.

In Figure 17, we show the programming in DEFRACTAL using the definition of convolution instruction CV2D as an example. The program declares the fractal instruction CV2D, the leaf decomposition (a local instruction *conv* in this example) and six *opt*s decomposing the instruction along the dimension of batch, channel-out, height, width and channel-in. A special case is for channel-in which is separated into two *opt*s for SD and PD, since the partial result is placed on the static segment when running in SD, but placed on the recycle segment when running in PD. In SD the local instruction *add* is serving as a sequential sub-instruction, while in PD it becomes a reduction instruction.

7.5 Evaluation

7.5.1 Methodology

Benchmarks. In Table 9, we show all benchmarks we used to evaluate Cambricon-FR instances. We select six fractal operations that are not natively supported on Cambricon-F as benchmarks, including 3D Convolution (Conv3D), Deconvolution (Deconv), Depthwise Convolution (DwiseConv), General Matrix Multiply (GEMM), Sparse Matrix Multiply Matrix (SPMM) and TopK, to evaluate the inefficiency issue. We also select five machine-learning applications which involve these operations, including C3D [51], FCN [52], Sparse AlexNet [3], MobileNet-V2 [53] and k-Nearest Neighbors, to evaluate the impact on overall applications.

GPUs. We use the same GPUs as baseline, i.e. Nvidia DGX-1 and Nvidia GeForce GTX-1080Ti, see Section 5. To evaluate both performance and programming productivity accurately on GPU systems, we write programs of benchmarks in plain CUDA C++ without calling computation libraries. On DGX-1, we write plain CUDA kernel functions utilizing TensorCore when possible. We report the maximum possible throughput as the performance

Cv2D in DeFracTaL

```
CV2D WT[CO,KY,KX,CI], IN[BT,YI,XI,CI], OUT[BT,YO,XO,CO] {
  opt(r,p) bt : [1, BT] {
    static WT;
    frac bt : BT {
      CV2D WT, IN[BT:...bt:bt,YI,XI,CI], OUT[BT:...bt:bt,YO,XO,CO];
    }
  } opt co : [1, CO] {
    static IN;
    frac co : CO {
      CV2D WT[CO:...co:co,KY,KX,CI], IN, OUT[BT,YO,XO,CO:...co:co];
    }
  } opt(r,p) yo : [KY, YO] {
    static WT;
    frac yo : YO {
      CV2D WT, IN[BT,YI:...yo:yo+KY-1,XI,CI], OUT[BT,YO:...yo:yo,XO,CO];
    }
  } opt(r,p) xo : [KX, XO] {
    static WT;
    frac xo : XO {
      CV2D WT, IN[BT,YI,XI:...xo:xo+KX-1,CI], OUT[BT,YO,XO:...xo:xo,CO];
    }
  } opt(s,r) ci : [1, CI] {
    static par[BT,YO,XO,CO], res[BT,YO,XO,CO];
    frac ci : CI {
      CV2D WT[CO,KY,KX,CI:...ci:ci], IN[BT,YI,XI,CI:...ci:ci], par;
      add res, par, res;
    }
    OUT = res;
  } opt(p) ci : [1, CI] {
    frac ci : CI {
      recycle par[BT,YO,XO,CO];
      CV2D WT[CO,KY,KX,CI:...ci:ci], IN[BT,YI,XI,CI:...ci:ci], par;
    } reduce add par..., OUT;
  } opt(1) {
    conv WT, IN, OUT, CI, CO, KX, KY, XO, YO, BT;
  }
}
```

Fig. 17. A DEFRACTAL program defining CV2D.

metric by testing on various batch size setup. *Lines of Source Code (SLoC)* are used as the quantitative metrics, which are including minimum required codes to run the computation, excluding data preprocessing, any comments, blank lines or dead codes.

Cambricon-Fs and Cambricon-FRs. We also use the same configuration for Cambricon-Fs and Cambricon-FRs as described in Section 5. Cambricon-FR1 and Cambricon-FR100 have the same configuration as Cambricon-F1 and Cambricon-F100 respectively, despite the different controller structures. We write the program in FISA/RFISA instructions, and also DEFRACTAL specifying new operations on Cambricon-FRs. Since programs can be ported without any adjustment between fractal machines, the codes on Cambricon-F1 and Cambricon-F100 are shared, so does Cambricon-FR1 and Cambricon-FR100. Therefore, we report SLoCs for Cambricon-F and Cambricon-FR, not the specific instances. We build C++ simulator to simulate these programs to obtain performance. The C++ simulator is event-driven, behavioral, modeled the execution on the hierarchical pipeline, which is sufficient to

TABLE 9
Benchmarks.

Benchmark	Dataset / Size Configuration
CONV3D	16×56×56×64, K=3
DECONV	224×224×256, K=3, S=2
DWISECONV	224×224×256, K=3
GEMM	32,768×32,768×32,768
SPMM	32,768×32,768×32,768, 60% sparse ratio (left)
TOPK	1 GB data, K = 512
C3D [51]	UCF101 [54] dataset
FCN [52]	PASCAL VOC2012 [55] dataset
SPARSE ALEXNET [3]	ImageNet [41] dataset
MOBILENET-V2 [53]	ImageNet dataset
K-NN	MNIST [56] dataset, K = 5

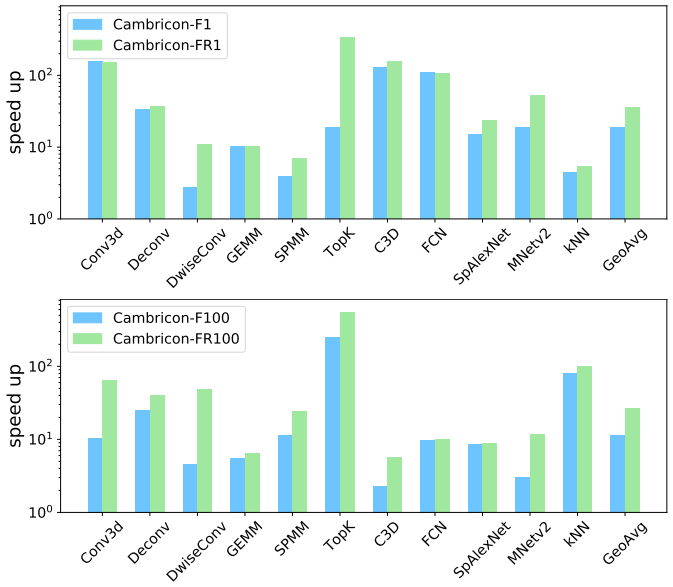

Fig. 18. Speed up fractal machines compared to GPU. Top: Cambricon-F1 versus Cambricon-FR1. Bottom: Cambricon-F100 versus Cambricon-FR100.

differentiate Cambricon-Fs and Cambricon-FRs.

7.5.2 Experimental Results

Performance. The performance comparison is shown in Figure 18. For the desktop-scale fractal machines, Cambricon-FR1 runs 1.96x faster than Cambricon-F1 on average, and for the server-scale fractal machines, Cambricon-FR100 runs 2.49x faster than Cambricon-F100 on average. The result shows that RFISA breaks the obstruction lying between split FISA instructions, and achieved remarkable performance improvement on most of the benchmarks.

Depthwise convolution achieves a 5.72x higher performance on Cambricon-FR1 versus Cambricon-F1, and a 13.63x higher performance on Cambricon-FR100 versus Cambricon-F100. The reason is that without RFISA, depthwise convolution is built upon massive element-wise multiplication, which has a poor data locality; with a user-defined depthwise convolution instruction, the data locality exploited in the convolutional window can be preserved through every memory hierarchies. Since depthwise convolution is improved greatly, MobileNet-V2 which is mainly constructed from depthwise and pointwise convolutions also benefits a lot, resulting in a 2.90x/5.30x performance gain.

Conv3D achieves a 3.60x higher performance on Cambricon-FR100 versus Cambricon-F100, but there are only negligible improvements achieved on Cambricon-FR1 versus Cambricon-F1. As analysed in Section 7.1, Conv3D is *communication ineffective* on Cambricon-F and have a K_D times higher lower-bound of communication. But on a smaller-scaled machine as Cambricon-FR1, the communication is already bounded by the memory capacity of the fractal nodes, thus the communication ineffectiveness has been hidden. Similar effects can be observed also on C3D and GEMM.

In contrast, TopK achieves an 18.88x higher performance on Cambricon-FR1 versus Cambricon-F1, but only 2.17x higher on Cambricon-FR100 versus Cambricon-F100. As analysed in Section 7.1, TopK is *computation ineffective* on Cambricon-F and have a worse time complexity. Cambricon-FR can reduce computation operations required in TopK dramatically, but for communications, the reduction is not as much. On a larger-scaled machine as Cambricon-FR100, the computation power is much higher but the bandwidth is limited, thus the effect on reducing computation operations cannot be clearly manifested.

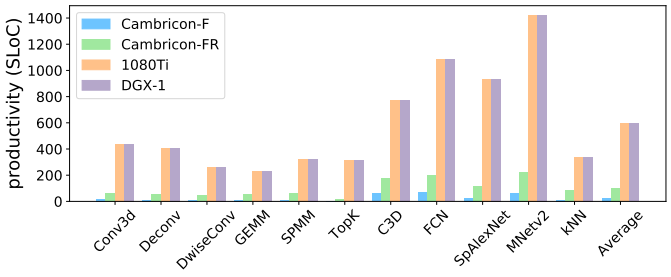

Fig. 19. Programming Productivity of Cambricon-F, Cambricon-FR and GPUs quantitated in SLoC.

Programming productivity. Programming productivity can be measured from the perspective of algorithm programmers or system software developers (including compiler). Regarding the practitioners, they can only write one program for multiple Cambricon-F/Cambricon-FR machines. The required programs are reduced from N to 1, significantly improved programming productivity. And for programming a certain program, Cambricon-F/Cambricon-FR provides a similar programming experience as the programming framework (e.g. TensorFlow, PyTorch, etc.). In Figure 19, we report the SLoC to quantitatively measure the programming productivity of algorithm programmers for one certain program. Compared to DGX-1 and 1080Ti, Cambricon-FR saves SLoC with a factor of 6.35, 6.06 on average, respectively, showing the significant improvement of programming productivity. And for N Cambrion-F/Cambricon-FR machines, the programming productivity can be further improved by N times, as non-fractal machines would have to write different programs for high efficiency.

Regarding the system software developers, they program to provide a programming environment to algorithm programmers. For Cambricon-FR machines, the DEFRACTAL code is used to define new RFISA instructions, thus it can be treated as the system software programming. Since Cambricon-FR requires additional DEFRACTAL codes, the SLoC of Cambricon-F is less than Cambricon-FR, with a factor of 4.70 on average. However, once the DEFRACTAL codes are written, they can be reused between applications. If DEFRACTAL codes are reused, they can improve the programming productivities for Cambricon-FR over Cambricon-F. The average length of DEFRACTAL codes is 81.1 lines, and RFISA codes are shorter than FISA codes with a factor of 1.30, with the help of newly defined instructions.

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. Most recent works are included in [57], [58], [59], [60], [61], [62], [63].

Yunji Chen et al. proposed the DianNao family of machine learning accelerators [22], [23], [37], [38], [39], which minimizes memory accesses to achieve both high performance and low power consumption. Yu-Hsin Chen et al. proposed Eyeriss [24] 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 [64] 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 addresses the programming productivity issue with new ISA. Venkat et al. [65] 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. Further, we propose Cambricon-FR, featured with a reconfigurable FISA, to flexibly and efficiently support all fractal operations. Our results show that the two Cambricon-FR instances achieve 1.96x, 2.49x better performance on average when comparing against Cambricon-F instances. Cambricon-FRs are also able to save the line of codes with a factor 5.83 on average compared to selected GPUs, thus significantly improving the programming productivity.

Acknowledgments

This work is partially supported by the National Key Research and Development Program of China (under Grant 2017YFA0700900, 2017YFA0700902,2017YFA0700901, 2017YFB1003101, 2018AAA0103300), 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), Strategic Priority Research Program of Chinese Academy of Science (XDB32050200, XDC01020000) and Standardization Research Project of Chinese Academy of Sciences (BZ201800001).

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. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-01264-9_48

[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. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-01270-0_1

[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. [Online]. Available: https://www.microsoft.com/en-us/research/publication/deep-learning-intelligent-video-analysis/

[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. [Online]. Available: http://arxiv.org/abs/1603.01870

[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*, Aug 2010, pp. 178–185.

[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." [Online]. Available: http://www.cambricon.com/news/index.php?c=show&id=253

[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)*. Ieee, jun 2011, pp. 109–116. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5981829

[20] 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, 2018, pp. 488–490.

[21] 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, 2013, pp. 1–12. [Online]. Available: http://dl.acm.org/citation.cfm?id=2540710

[22] 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, 2014, pp. 269–284. [Online]. Available: http://dl.acm.org/citation.cfm?id=2541967

[23] 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)*, 2015, pp. 369–381.

[24] 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)*, 2016, pp. 367–379. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7551407

[25] 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.

[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, 2016, pp. 265–283.

[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, no. 61772350, 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*, 2017, pp. 786–799. [Online]. Available: https://doi.org/10.1145/3123939.3123970

[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*, 2017, pp. 367–381. [Online]. Available: https://dl.acm.org/citation.cfm?id=3123979

[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)*, 2017, pp. 535–547. [Online]. Available: http://arxiv.org/abs/1607.00064

[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. [Online]. Available: https://www.csl.cornell.edu/{~}tchen/files/parallelxl-micro18.pdf

[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)*, 2015, pp. 609–622.

[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. [Online]. Available: 10.1145/2594446{%}5Cnhttps://ejwl.idm.oclc.org/login?url=http://search.ebscohost.com/login.aspx?direct=true{&}db=bth{&}AN=95797996{&}site=ehost-live

[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*, 2015, pp. 92–104.

[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)*, 2017, pp. 1–17.

[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)*. Ieee, 2006, pp. 2773–2776. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1693199

[45] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[46] 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.

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

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

[49] 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*, ser. DATE '15. San Jose, CA, USA: EDA Consortium, 2015, pp. 1543–1546. [Online]. Available: http://dl.acm.org/citation.cfm?id=2755753.2757168

[50] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009. [Online]. Available: http://doi.acm.org/10.1145/1498765.1498785

[51] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *The IEEE International Conference on Computer Vision (ICCV)*, 2015.

[52] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1520–1528.

[53] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018. [Online]. Available: http://arxiv.org/abs/1801.04381

[54] K. Soomro, A. R. Zamir, and M. Shah, "Ucf101: A dataset of 101 human actions classes from videos in the wild," *arXiv preprint arXiv:1212.0402*, 2012.

[55] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.
[56] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[57] 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)*, June 2018, pp. 43–56.
[58] 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)*, June 2018, pp. 340–352.
[59] 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)*, June 2018, pp. 383–396.
[60] 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)*, June 2018, pp. 674–687.
[61] 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)*, June 2018, pp. 662–673.
[62] 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)*, June 2018, pp. 688–698.
[63] 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)*, June 2018, pp. 764–775.
[64] 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)*, June 2017, pp. 320–332.
[65] 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)*, Feb 2019, pp. 42–55.

Yongwei Zhao recieved the bachelor degree in Computer Science and Technology from Huazhong University of Science and Technology, Wuhan, China, in 2015. Now he is a PhD student of Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, and the University of Chinese Academy of Science, Beijing, China.

Zhe Fan recieved the bachelor degree in Computer Science and Technology from Huazhong University of Science and Technology, Wuhan, China, in 2017. Now he is a PhD student of Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, and the University of Chinese Academy of Science, Beijing, China.

Zidong Du recieved the bachelor degree of Engineering from Department of Electronic Engineering, Tsinghua University in 2011 and PhD degree from ICT, CAS in 2016 with the guidance from Prof. Yunji Chen, Prof. Olivier Temam and Prof. Chengyong Wu. He is currently an associate professor at Institute of Computing Technology (ICT), Chinese Academy of Sciences(CAS).

Tian Zhi recieved the bachelor degree of Biomedical Engineering, Zhejiang University in 2009 and PhD degree from IE, CAS in 2014. She is currently an associate professor at Institute of Computing Technology (ICT), Chinese Academy of Sciences(CAS).

Ling Li is a professor at Institute of Software, Chinese Academy of Sciences. She joined the Godson (Loongson) project in 2009. She was the chief architect of Godson video decoding IP. Ling Li has authored or coauthored papers on journals (including IEEE TIP, IEEE TPDS, IET IP) and conferences (including DCC, SPAA, ICASSP). Her research interests include intelligent computing and video processing.

Qi Guo is a professor at Institute of Computing Technology, Chinese Academy of Sciences, China. He received the PhD degree from Institute of Computing Technology, Chinese Academy of Sciences in 2012. He recieved the B.E. degree in CS from Tongji University in 2007. From 2012 to 2014, he was a staff researcher at IBM Research, China. From 2014 to 2015, he was a postdoctoral researcher at Carnegie Mellon University. His research interests include computer architecture, machine learning, and programming models.

Shaoli Liu received the bachelor degree in mathematics from Nankai University, Tianjin, China, in 2009, and the PhD degree from the Institute of Computing Technology (ICT), Chinese Academy of Sciences, Beijing, China, in 2014. He is currently an associate professor at ICT. Hist current research interests include parallel computing, network on chip, microarchitectures, hardware verification, and computational intelligence. He is also serving in a startup called Cambricon Technologies Corporation Limited.

Zhiwei Xu received the Ph.D. degree from the University of Southern California, Los Angeles, CA, USA, in 1987. He is currently a Professor with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. His current research interests include high-performance computer architecture and distributed computing systems.

Tianshi Chen received the bachelor degree in mathematics from the Special Class for the Gifted Yong, University of Science and Technology of China (USTC), Hefei, China, in 2005, and the PhD degree in computer science from the Department of Computer Science and Technology, USTC, in 2010. He received the China Computer Federation Distinguished Doctoral Dissertation Award in 2011 and the Chinese Academy of Sciences Distinguished Doctoral Dissertation Award in 2011 for his PhD work on computational complexity analysis of evolutionary algorithms. He is currently an professor with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. He is also serving as the CEO of a startup called Cambricon Technologies Corporation Limited, whose commercial processor products are named "Cambricon".

Yunji Chen received the graduation degree from the Special Class for the Gifted Young, University of Science and Technology of China, Hefei, China in 2002. He received the PhD degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences, Beijing, China, in 2007. He is currently a professor with ICT. His current research interests include parallel computing, microarchitectures, hardware verification, and computational intelligence. He has authored or coauthored one book and more than 60 papers in these areas.