High Performance and Power Efficient Accelerator for Cloud Inference

Jianguo Yao
SJTU and Enflame-Tech Inc.
Shanghai, China
jianguo.yao@sjtu.edu.cn
jianguo.yao@enflame-tech.com

Hao Zhou, Yalin Zhang, Ying Li, Chuang Feng, Shi Chen,
Jiaoyan Chen, Yongdong Wang, Qiaojuan Hu
Enflame-Tech Inc.
Shanghai, China
{vincent.zhou, arthur.zhang, vivian.li, chuang.feng, shi.chen, joyan.chen, jarod.wang, qiaojuan.hu}@enflame-tech.com

2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 978-1-6654-7652-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/HPCA56546.2023.10070941

Abstract—**Facing the growing complexity of Deep Neural Networks (DNNs), high-performance and power-efficient AI accelerators are desired to provide effective and affordable cloud inference services. We introduce our flagship product, i.e., the Cloudblazer i20 accelerator, which integrates the innovated Deep Thinking Unit (DTU 2.0). The design is driven by requests drawn from various AI inference applications and insights learned from our previous products. With careful trade-offs in hardware-software co-design, Cloudblazer i20 delivers impressive performance and energy efficiency while maintaining acceptable hardware costs and software complexity/flexibility. To tackle computation- and data-intensive workloads, DTU 2.0 integrates powerful vector/matrix engines and a large-capacity multi-level memory hierarchy with high bandwidth. It supports comprehensive data flow and synchronization patterns to fully exploit parallelism in computation/memory access within or among concurrent tasks. Moreover, it enables sparse data compression/decompression, data broadcasting, repeated data transfer, and kernel code prefetching to optimize bandwidth utilization and reduce data access overheads. To utilize the underlying hardware and simplify the development of customized DNNs/operators, the software stack enables automatic optimizations (such as operator fusion and data flow tuning) and provides diverse programming interfaces for developers. Lastly, the energy consumption is optimized through dynamic power integrity and efficiency management, eliminating integrity risks and energy wastes. Based on the performance requirement, developers also can assign their workloads with the entire or partial hardware resources accordingly. Evaluated with 10 representative DNN models widely adopted in various domains, Cloudblazer i20 outperforms Nvidia T4 and A10 GPUs with a geometric mean of 2.22x and 1.16x in performance and 1.04x and 1.17x in energy efficiency, respectively. The improvements demonstrate the effectiveness of Cloudblazer i20's design that emphasizes performance, efficiency, and flexibility.**

I. Introduction

Applications powered by Artificial Intelligence (AI) are now ubiquitous. Over the past few years, we have witnessed the remarkable success of deploying Deep Neural Networks (DNNs) in various real-life related domains, such as object detection [27], [29], [72], image classification [41], [78], [80], video enhancement [56], natural language processing (NLP) [28], and speech recognition [38].

Meanwhile, from Convolution Neural Network (CNN) [54], [55] to Transformer [70], [85], the evolution of DNN models has never stopped [36], [43], [52], [81], [91]. However, the continuously improved inference accuracy, comes at the cost of increased computational intensity and architectural complexity [40]. We have seen many applications struggling to be deployed on low-cost embedded edge devices [45], [73]. Cloud-hosted DNN inference, on the other hand, also encounters serious challenges [20], [22], [37], [40], [48], [68].

TABLE I
Technical specifications of the Cloudblazer i20 accelerator.

Performance		Features	
FP32	32 teraFLOPS	Memory	16GB
TF32	128 teraFLOPS	Bandwidth	819GB/s
FP16	128 teraFLOPS	Board TDP	150W
BF16	128 teraFLOPS	Interconnect	PCIe Gen4 64GB/s
INT8	256 TOPS	Software	Enflame Customized

Providing a comprehensive AI cloud inference service requires multi-objective optimization. A list of metrics needs to be synthetically considered, such as inference latency, throughput, accuracy, price-performance ratio, and the capabilities of adapting various DNNs/frameworks. Many of these metrics are inter-correlated. How to balance and incorporate them into design considerations has been the main focus investigated by both cloud service providers and hardware vendors.

Industry giants Amazon and Microsoft adopted powerful Nvidia GPUs as DNN inference accelerators into their cloud service platforms AWS [3] and Azure [4]. Alternatively, Google developed its own commercial Domain Specific Architectures (DSAs) for DNNs, i.e., Tensor Processing Units (TPUs), as the substrate for data centers and cloud [49], [50], [64]. Following that, many customized DSAs [24], [30], [39], [47], [48], [57], [58], [89] have been designed worldwide in recent years, aiming at higher performance, efficiency, and flexibility for DNN deployment [33], [68].

In this paper, we introduce the Cloudblazer i20 accelerator, the flagship product from Enflame, for AI inference in cloud and data centers. Table I lists its technical specifications. The accelerator equips with the 2nd generation Deep Thinking Unit (DTU 2.0). It is a 12nm FinFET chip, innovated to reinforce AI cloud inference services. The enhancements made in DTU 2.0 lies in the observations of the characteristics/trends of AI inference applications, along with lessons learned from the DTU 1.0 released in 2019 [58] (see Section II for details). Both hardware innovations and software adaptions contribute to our solution of designing the Cloudblazer i20 accelerator and DTU 2.0. We made careful trade-offs to boost the performance and efficiency while maintaining acceptable hardware costs and software complexity/flexibility.

For instance, to satisfy the growing demands from computation-/data-intensive workloads, DTU 2.0 integrates specialized yet flexible vector/matrix engines and adopts a multi-level memory hierarchy with large-capacity local registers/buffers and distributed multi-ports shared memory. Additionally, high-bandwidth HBM2E chips are utilized to speed up the data exchange. The increased computational power, data storage, and bandwidth make it more beneficial to apply effective kernel fusion by the software and thus optimize data reuse and computation efficiency, meanwhile eliminating unnecessary data movements. DTU 2.0 delivers efficient data transfer by taking advantage of data sparsity. Moreover, it enables instruction cache and supports user-controlled kernel code prefetching, alleviating the impact of slow loading of the kernel code, especially for oversized kernels generated through kernel fusion. Furthermore, to fully exploit parallelism in computation and memory access within or among concurrent threads/tasks, DTU 2.0 supports comprehensive data flow and synchronization patterns. In particular, it enables data broadcasting to facilitate data-sharing and optimizes repeated data transfer with reduced configuration overheads. Auto-tuning performed by the software helps developers to find efficient data access patterns. Lastly, DTU 2.0's energy consumption is optimized through dynamic power integrity and efficiency management, which prevents integrity risks and energy wastes. On the other hand, developers can appropriately map workloads to the entire or partial underlying hardware based on their performance requirements, benefiting from the resource abstraction provided by DTU 2.0.

The key contributions in this paper are:

- We analyze the characteristics of DNN models and summarize crucial indicators for cloud inference services. Based on lessons learned from our previous products, we discuss critical trade-offs in system design to ensure performance, efficiency, and flexibility.
- We introduce a novel architecture to strengthen cloud inference services. With the innovative compute engine, memory hierarchy, data flow/synchronization engine, resource abstraction, and power management, it provides sufficient headroom to adapt to emerging DNNs and delivers satisfying outcomes.
- We present the software stack that helps developers to efficiently deploy their workloads. By offering comprehensive programming interfaces and enabling automatic optimizations, it exploits the full potential of the underlying hardware.
- We evaluate 10 representative DNNs widely used in diverse domains. Compared to Nvidia's T4 and A10 GPUs, on average, Cloudblazer i20 delivers 2.22x and 1.16x performance improvements and increases power efficiency by 4% and 17%, respectively.

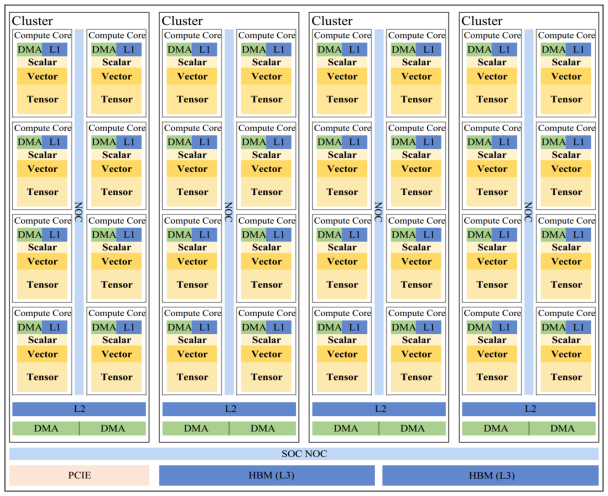


Fig. 1. The SoC architecture of DTU 1.0.

The rest of the paper is organized as follows. We look back into DTU 1.0 and discuss metrics for evaluating cloud inference services in Section II. In Section III, we highlight the motivations and outline the hardware/software enhancements made in DTU 2.0. Then, we introduce the design of DTU 2.0 and Cloudblazer i20 in Sections IV and V, respectively. The evaluation is presented in Section VI and followed by related work in Section VII. Finally, Section VIII concludes.

II. Background

A. The Predecessor DTU 1.0

DTU 1.0 [58] is Enflame's first DSA design for DNN applications. It adopts the 12nm FinFET technology and is constructed with 14.1 billion transistors. Fig. 1 illustrates the architecture of its system-on-chip (SoC).

Inside DTU 1.0, 32 compute cores based on the Very Long Instruction Word (VLIW) architecture [31] are equally distributed into 4 clusters, performing efficient scalar, vector, and tensor computations. Its vector engine processes on 1024-bit vectors, which functions similarly to the Single Instruction Multiple Data (SIMD) unit widely adopted in mainstream CPUs. The tensor engine supports general matrix-matrix multiplication (GEMM) on specific tensor shapes and data types. Overall, DTU 1.0 delivers performance up to 20, 80, and 80 teraFLOPS for FP32, FP16, and BF16. As to INT32, INT16, and INT8, the peak performances are 20, 80, and 80 TOPS.

To accelerate memory access and exploit data locality, DTU 1.0 adopts a 3-level memory hierarchy: (L1) the 256KB local data buffer resides in each compute core, (L2) the 4MB DRAM shared by 8 compute cores in each cluster, and (L3) the two 8GB HBM2 chips with the maximum bandwidth up to 512GB/s. DTU 1.0 utilizes specialized direct memory access (DMA) engines for managing data movements within the memory hierarchy. Each compute core is paired with one dedicated DMA engine to exchange data between the L1 and L2 memory. Meanwhile, two additional DMA engines in each cluster are in charge of transporting data between the L2 and L3 memory. The PCIe4x16 interface on DTU 1.0 is used for host-device communications, of which the maximal data transfer rate is 64GB/s.

B. Metrics for Cloud Inference Services

We believe solid product design comes from thorough requirements analysis. Considering different aspects and emphases of AI cloud inference services, we categorize some important indicators into the following three scopes:

- **Performance**: Peak performance reveals how many *operations per second* (on floating-point or integer data) the underlying hardware can accomplish in theory. To measure the performance of a specific DNN model, more accurate indicators include the *latency* incurred to process one sample and the *throughput* of handling batched inputs. Furthermore, for precision-sensitive applications, *accuracy* is another crucial factor that affects the overall performance. These major indicators are *Quality of Service* (QoS) related.
- **Efficiency**: Cloud users normally assess inference efficiency based on the expense spent on finishing specific tasks, i.e., the *price-performance ratio*. It is associated with the *Total Cost of Ownership* (TCO) of the AI cloud [37], for which both the capital expense and the operation expense are measured. Therefore, *resource utilization* and *energy efficiency* of the underlying hardware are keys to building affordable and efficient cloud inference.
- **Flexibility**: To keep up with the pace of the emerging DNNs and frameworks, AI cloud infrastructure has to offer adequate *adaptivity* to cater to rapidly evolving workloads. Not only does it support the integration with the existing best-practice models, but also the *scalability* of deploying user-customized ones. It requires the underlying hardware to be generic, and thus be able to effectively process various data types, operations, and data access/communication patterns.

III. Motivations to Innovate DTU

Driven by the demands of constructing comprehensive cloud inference infrastructure, we redesigned the DTU and aimed to provide sufficient performance, efficiency, and flexibility. We analyzed the characteristics and trends of DNN inference applications and shaped DTU 2.0 accordingly. Meanwhile, lessons were learned from DTU 1.0 to avoid potential pitfalls and offer headroom for adapting future DNN advancements. In this section, we discuss important insights that motivate DTU 2.0's design.

Capability v.s. Quantity: Given the constraints of chip area and power consumption, balancing the capability and quantity of processing cores is crucial in delivering satisfying performance and efficiency. Unlike Nvidias GPU, which consists of thousands of simplified processing elements, DTU is constructed with less yet more powerful compute cores. Computation-bound AI inference workloads would benefit from wider vectors and matrix multiplication/convolution engines with high computational density [40]. By integrating sophisticated vector/matrix/tensor engines in each compute core, we have observed remarkable improvements in accelerating DNN inference applications with DTU 1.0. It is promising to further strengthen these compute cores with more effective support for typical DNN operators. The matrices processed in *Fully Connected* (FC) and *Convolutional Layers* usually have square shapes. However, common primitive operators often involve tall and skinny matrices, such as group/depth-wise convolutions in computer vision workloads, as discussed in [68]. Matrix engines that only support limited matrix shapes would be inefficient for handling them. Therefore, instead of supporting coarse-grained GEMM in DTU 2.0, we redesigned the matrix engine to accelerate fine-grained vector-matrix multiplication (VMM). Meanwhile, hardware-implemented VMM also benefits other tasks such as Top-K calculation [82].

Memory v.s. ALUs: To fully display the computational strength of the compute core, powerful ALUs need to pair with fast data access. As the DNN model's input size grows rapidly, exploiting locality becomes more critical [87]. For example, matrix convolution is well known as the key operator in CNNs, for which the weight reuse is crucial to improve arithmetic intensity [40]. DTU integrates large register files and a 3-level memory hierarchy to ensure more required data retain at registers or the memory level close to ALUs. To benefit from data reuse and software optimizations such as operator fusion, we believe increasing the capacity of register files, L1, and L2 memory would further boost DTU's overall performance.

Bandwidth v.s. Capacity: It is inevitable to transfer data between memory and compute core on systems based on the Von Neumann architecture. Computations with low arithmetic intensities, such as sparse embedding, can be bottle-necked due to memory bandwidth when large-capacity on-chip memory fails to accommodate the entire data required [40], [61], [68]. Effective memory bandwidth needs to go up with the memory capacity to maintain sufficient compute utilization and deliver desirable performance for current/future DNN models [32]. We consider several feasible solutions, such as equipping memory with parallel read/write ports and higher bandwidth and optimizing bandwidth for data with high sparsity.

Data flow v.s. Computation: Time-consuming tensor layout transformation operations are common in DNNs [68]. DTU utilizes DMA engines to accomplish tensor manipulation while data transfer. It alleviates the burden of the compute core and enables parallel execution of computation and data flow. Assisted with multiple-buffering, the latency incurred due to data movements can be appropriately hidden [32]. Meanwhile, we observed regular data access patterns involved in load-compute-store operation sequences in many DNN operators (e.g., element-wise tensor operations). For these patterns, we try to optimize the DMA configuration overheads.

Kernel code loading matters: Kernel code loading drags DNN execution, because the compute core starts running only until the required kernel code has been transferred to the instruction buffer. The performance impact caused by kernel code loading aggravates as the kernel code size increases, especially after operator fusion. To improve the efficiency of kernel code loading, we draw inspiration from instruction

TABLE II
HARDWARE INNOVATION AND SOFTWARE ADAPTION MADE FOR DTU 2.0 TO CONFRONT THE DEMANDS OF CLOUD INFERENCE WORKLOADS AND SERVICES (MORE DETAILS OF HARDWARE/SOFTWARE ENHANCEMENTS ARE INTRODUCED IN SECTIONS IV AND V).

Characteristics/Trends of Workloads	Hardware Innovation for DTU 2.0	Software Adaption for DTU 2.0	Enhancements over DTU 1.0	Service Indicator
Matrix multiplication and convolution are dominant computations in DNNs. Matrix shape can be tall and skinny.	Hardware implemented vector-matrix multiplication (VMM) that support various matrix shapes and data types.	High performance libraries utilizing VMM instructions; Enabled auto-tensorization.	Adopt fine-grained VMM implementation over GEMM; More than 40 VMM patterns supported.	Performance Flexibility
Element-wise computation with transcendental functions is required for calculating diverse activation functions in DNNs.	Enhanced special function unit (SFU) for processing transcendental functions.	High performance libraries utilizing SFU-related instructions; Enabled auto-vectorization.	Around 10 transcendental functions are accelerated. The throughput of the SFU is improved.	Performance
Efficient Top-K recommendation relies on the highly optimized implementation of data sorting.	VMM-assisted data sorting through generating the relationship matrix and the transformation matrix.	High performance libraries utilizing sorting-related instruction.	New feature introduced to accelerate Top-K selection.	Performance
In DNN's forward pass, operators usually use their immediate predecessor's results as inputs.	Larger register file and L1/L2 memory to accommodate intermediate values.	Enable auto-fusion to eliminate unnecessary data materialization and scan.	4x/6x larger capacities of the L1/L2 memory per compute core/cluster.	Performance
Kernel code loading affects overall DNN performance.	Enable the cache mode of the instruction buffer.	Support user-controlled kernel code prefetch.	New feature introduced to fasten kernel loading.	Performance
The growing data leveraged by DNNs often exceed the on-chip memory's capacity.	Increased memory bandwidth; Multiple parallel read/write ports for shared memory.	Enable affinity-aware memory allocation to optimize data access latency.	1.6x higher bandwidth; 4 parallel read/write ports for the L2 memory.	Performance
Data with high sparsity is often observed in DNN's inputs and intermediate values.	Enable sparse data decompression during data transfer with DMA.	Support user-controlled data compression/decompression.	New feature introduced to alleviate the growing bandwidth pressure.	Performance
Batch processing with concurrent tasks often shares common inputs.	Enable data broadcasting to multiple L2 destinations.	Support user-controlled data broadcasting.	New feature introduced to optimize data sharing.	Performance
Time-consuming tensor layout transformations required in DNNs may become the performance bottleneck.	Enhanced DMA engine that supports various tensor layout transformations.	Support data-flow auto-tuning to exploit parallelism in computation and data access.	Enable data movements in any direction; More comprehensive support on tensor manipulation.	Performance Flexibility
Regular patterns of load-compute-store operations are found in many DNN operators.	Enable efficient DMA configurations for repetitive data transfer patterns.	Support user-controlled repeat-DMA configuration.	New feature introduced to reduce DMA configuration overheads.	Performance
Exploiting thread-level parallelism requires flexible means of thread synchronization.	Enable 1-to-1, 1-to-N, N-to-1, N-to-M synchronization.	Support user-controlled thread synchronization.	New feature introduced to benefit thread-level parallelism.	Performance Flexibility
Multi-task/tenancy helps balance overall throughput and latency.	Hardware abstraction with isolated hardware resources.	Support user-controlled resource mapping between hardware and workloads.	Up to 6 groups of isolated hardware resources serve independently.	Performance Efficiency
Different workloads result in various power utilization of the underlying hardware.	Power integrity/efficiency managements based on on-demand adjustments.	N/A	Improved system energy efficiency.	Efficiency
DNNs become more dynamic.	N/A	Support dynamic tensors and shape inference.	New feature supported in the software stack.	Flexibility
DNNs keep on evolving. Users may customize their own DNNs/operators.	Support diverse data types, VMM shapes, DMA options, synchronization patterns.	Support development on invented/customized DNNs/operators;	More general and flexible design in both hardware and software.	Flexibility

cache and data prefetching widely adopted in CPU designs.

Multi-task/tenancy matters: Exploiting task-level parallelism is a promising approach to trade-off throughput and latency [48], [74]. Through multi-tenancy, abundant hardware resources get better utilized, and the relative cost for performing each task gets lower. We believe hardware resource abstraction and isolation would benefit multi-task/tenancy through straightforward resource assignment and management. More importantly, as isolated hardware resources prevent interference among each other, system throughput is increased without compromising inference latency, improving the overall QoS. Furthermore, it helps protect user data safety.

Power management matters: Different layers in DNNs (e.g., convolution, activation, and embedding) often show diverse execution patterns/behaviors in computation and memory access [40], [68]. By carefully monitoring DTU's runtime power consumption, we witnessed how the energy utilization of function engines (such as the compute core and DMA engine) varies according to the workloads. To address both power efficiency and integrity issues, we desire flexible and dynamic power management to distribute sufficient power over system engines and thus alleviate performance bottlenecks.

Software stack matters: As the interface between users and the underlying hardware, the software stack is responsible for providing means of application adaption and development [18], [25], [69], [84]. Fine-tuned libraries paired with

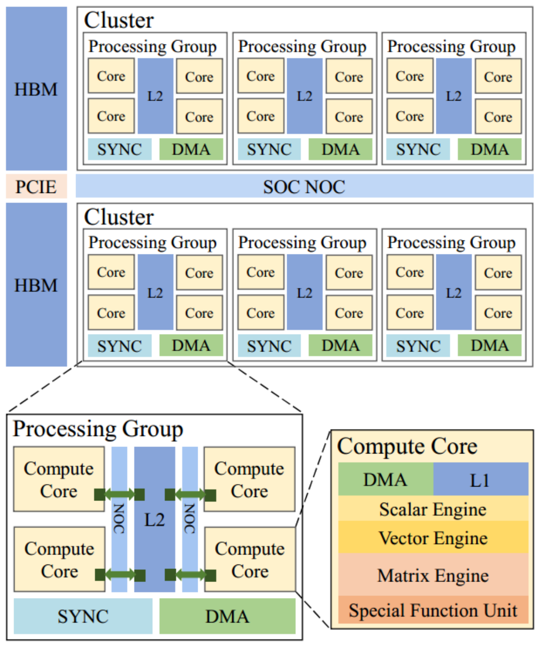


Fig. 2. The SoC architecture of DTU 2.0.

effective compiler optimizations are necessary for developers to conveniently utilize DTU at full capacity. Moreover, the programming interfaces provided to developers should be general and flexible enough to meet their different needs.

Based on the design considerations discussed above, we innovate DTU with several hardware/software enhancements, as outlined in Table II. We introduce more details in the following sections.

IV. The Design of DTU 2.0

The architecture of DTU 2.0's SoC is illustrated in Fig. 2, which shows a major micro-architectural update compared with Fig. 1. The chip consists of 2 clusters, each containing 12 compute cores. Compared to DTU 1.0, although the total number of compute cores reduced from 32 to 24, DTU 2.0 offers 1.6x peak performance on FP32/FP16/BF16/INT32/INT16 data types and 3.2x peak performance on the INT8 data type (in terms of teraFLOPS/TOPS as shown in Table I). The performance improvements come from increased computational density and MACs. We describe some key features of the compute core in Section IV-A.

DTU 2.0 continually adopts a 3-level memory hierarchy, similar to DTU 1.0. However, the L2 memory is distributed evenly into three parts, each associated with 4 compute cores. Compared to DTU 1.0, the overall capacities of L1 and L2 memory are increased by 3x. Due to the decrease in compute cores, the L1/L2 memory per core becomes 4x/6x larger. The capacity of L3 memory remains the same. Its bandwidth is 1.6x larger, owing to the replacement of HBM2 chips with more powerful HBM2E chips.

Besides 1/3 of the L2 memory, every 4 compute cores in each cluster are bundled with 1 DMA engine and 1 synchronization engine. In this way, each cluster is abstracted as 3 identical and independent processing groups. The hardware

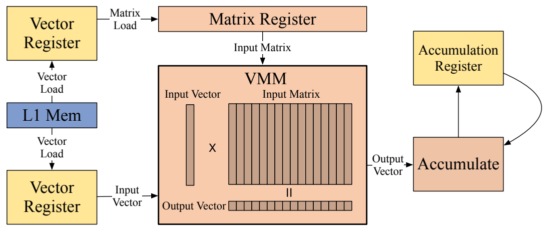


Fig. 3. The vector matrix multiplication (VMM) facility.

isolation makes it easier for the software to assign parallel tasks to adequate resources accordingly. We discuss more details about resource abstraction in Section IV-E.

A. Compute core

The compute core takes charge of executing the kernel code of DNN operators. Same as DTU 1.0, it adopts the VLIW architecture and supports a full range of widely used data types, i.e., from 8-bit up to 32-bit integer and floating-point types. Each core integrates dedicated ALUs for processing scalar, vector, matrix, and special functions. Among its enhancements in many aspects, we focus on introducing two components, i.e., the matrix engine and the special function unit (SPU).

1) Matrix engine: To accelerate typical linear algebra operations commonly seen in DNN workloads (e.g., matrix multiplication and convolution), the matrix engine leverages 2 matrix registers (32x512-bit), 32 vector registers (512-bit), and 1024 accumulation registers (512-bit) to perform efficient vector-matrix multiplication (VMM). Taking the FP32 data type as an example, the matrix shapes supported include 16x16, 8x16, and 4x16, and the corresponding vector lengths for VMM are 16, 8, and 4, respectively.

Fig. 3 illustrates how VMM is computed. At the data preparation stage, required data are loaded from the L1 memory and filled into corresponding matrix rows/columns via vector registers. Then, through a series of out-product operations, the input vector is operated with each row/column of the input matrix. The final output is kept in an accumulation register. By improving the computational throughput and utilizing sufficient accumulation registers (to maximize data reuses and minimize data movements), the matrix engine effectively optimizes VMM's performance and power efficiency.

Additionally, the matrix engine facilitates the efficient implementation of data sorting for Top-K queries [82], [92]. As shown in Fig.4, the matrix engine generates the relationship matrix of the data contained in the input vector, by comparing vector elements against each other (Step ①). Identical elements in the input vector are appropriately handled according to their original indices. Then, the relative order of vector elements (stored in order vector) is obtained through calculating the sum of each column of the relationship matrix (Step ②). It is used to generate the transformation matrix (i.e., the permutation matrix [10]), of which the column index of the unique non-zero value "1" in i-th row (the bottom row is Row 0, and the rightmost column is Column 0) equals the i-th element in the order vector (the rightmost element is Element 0) (Step ③). Finally, by applying VMM computation with

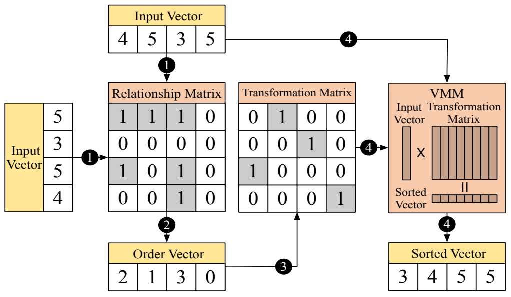


Fig. 4. The data sorting facility.

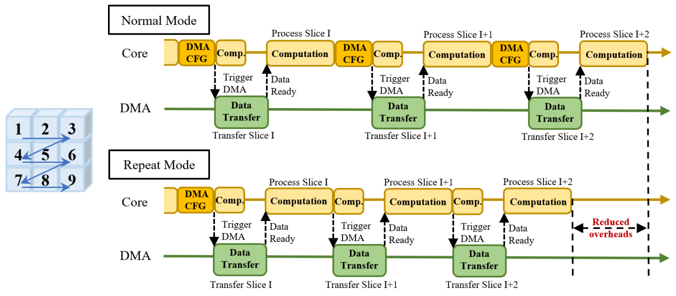


Fig. 6. A comparison of DMA engine's normal mode and repeat mode in DTU 2.0, considering data slicing out of a large tensor.

the input vector and the transformation matrix, we obtain the sorted vector (Step ④).

2) Special Function Unit (SPU): Activation functions are widely used operators in DNNs, which require transcendental function computation. Cooperated with the vector engine, the SPU executes efficient calculations on transcendental functions by computing the quadratic Taylor polynomial, according to the derivative values found in the Lookup Table. It supports activation functions such as *Softplus*, *Tanh*, *Sigmoid*, *Gelu*, *Swish*, *Softmax*, etc.

B. Memory Hierarchy

As illustrated in Fig. 2 and introduced earlier in Section IV, DTU 2.0 has a 3-level memory hierarchy, in which the L2 memory is divided equally into 3 processing groups. Fig. 5 gives a more straightforward view of the memory hierarchy.

To increase the bandwidth of shared memory in each processing group, the L2 memory equips with 4 parallel read/write ports. Therefore, 4 compute cores in the processing group can access L2 memory without interference.

Considering the performance impact caused by loading kernel code directly from the L3 memory into the instruction buffer of each compute core, DTU 2.0 enables instruction cache and provides specific instructions to the programmers for controlling kernel code prefetch. The design is inspired by similar techniques adopted in CPU designs. By inserting the prefetch instructions, the kernel code of the upcoming operator is loaded in advance to avoid performance penalties. Besides, it solves the problem of loading extremely large kernels that exceed the capacity of the instruction buffer. On cache misses, the instruction buffer triggers kernel code loading automatically.

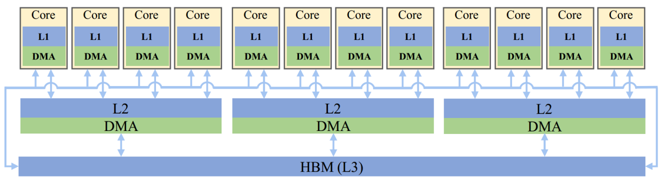


Fig. 5. The memory hierarchy and data movements in DTU 2.0 (only one cluster is shown).

C. DMA Engine

Data movements among memory levels in the memory hierarchy are accomplished by DMA engines, as shown in Fig. 5. The primary motivation we integrate DMA engines in DTU is to relieve compute core's burden of manipulating tensor layout and make it focus more on computation. During data transfer, DMA engines can perform tensor layout transformations on the fly according to the configuration, such as padding, slicing, transposing, and concatenation on specified tensor dimensions. Particularly, to optimize bandwidth for transferring sparse data, DMA engines in DTU 2.0 supports automatic data decompression. Given the data compressed in hardware-defined formats, DMA engines decompress the data while storing them at the destination memory locations.

Different from DTU 1.0 for which the data in L1 memory must be loaded from or stored in the L2 memory, DTU 2.0's DMA engines can directly transfer data between the L1 and L3 memory. Therefore, the valuable L2 memory bandwidth and capacity are saved for data shared among multiple compute cores. Data movements between different locations at the same memory level are also supported. Besides the support for 1-on-1 data transfer, in each cluster, DMA engines can perform data broadcasting in L2 memory across 3 processing groups. According to user-configured destination locations, 3 identical data copies are written all at once. It maximizes bandwidth utilization and accelerates inter-group data sharing.

DMA engines help parallelize data movements and computations. However, each time, the compute core needs to configure DMA engines to perform data transfer as desired. To reduce the configuration overheads, in DTU 2.0, we introduce the *repeat* mode of the DMA engine. It triggers multiple DMA transactions that follow a repetitive and regular pattern with one single DMA configuration. The repeat mode benefits many data transfer patterns observed in DNN operators, and Fig. 6 gives a typical example. Here, the large tensor is consumed in small slices (labeled from 1 to 9) with fixed strides. Without the repeat mode, N DMA transactions/configurations are required. Enabling repeat mode eliminates $(N\text{-}1)/N$ of the DMA configuration overheads, resulting in higher performance.

D. Synchronization Engine

DTU 2.0 offers flexible facilities for synchronizing among compute cores and DMA engines. As shown in Fig. 2, each processing group integrates a dedicated synchronization

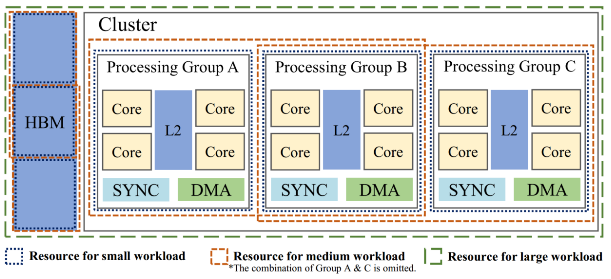

HBM
Cluster
Processing Group A
Processing Group B
Processing Group C
Core
L2
SYNC
DMA
Resource for small workload
Resource for medium workload
*The combination of Group A & C is omitted.
Resource for large workload

Fig. 7. Resource abstraction and assignment for multi-task/tenancy with different performance requirements (for one cluster only).

engine. It supports 1-to-1, 1-to-N, N-to-1, and N-to-M synchronization patterns, inside or across processing groups. The synchronization engine builds the foundation of exploiting parallelism at various levels, e.g., among computations and data transfers of one single task (i.e., intra-task), and among workloads running on different compute cores or processing groups (i.e., inter-tasks).

E. Resource Abstraction for Multi-task/tenancy

For cloud inference, the ability to efficiently serve multiple user requests is crucial to improve throughput and hardware utilization [93]. By isolating hardware resources and abstracting them as independent processing groups, DTU 2.0 provides a more natural and convenient way of mapping user requests with the underlying hardware. Besides dedicated functional engines residing in processing groups, other hardware resources (e.g., the L3 memory) are shared appropriately.

To achieve the desired performance, users can utilize the hardware adequately based on carefully evaluating resource requirements for their workloads. Fig. 7 demonstrates different ways of assigning hardware resources, considering the processing group as the minimal unit for workload deployment. For large workloads, all the resources in the cluster (i.e., 3 processing groups) are assigned; for small workloads, one single processing group is sufficient; otherwise, 2 processing groups combined would be a better fit. Specific workload assignment can be performed by the compiler (automatically) or the developer (manually). Diver and firmware eventually map the tasks to underlying hardware during runtime. Mapping workloads to adequate hardware resources also minimize unnecessary energy wastes, optimizing system power efficiency.

F. Power Management Engine

The goal of power management in DTU 2.0 is to ensure power integrity and improve energy efficiency. As shown in Fig. 8, DTU 2.0 adopts a hybrid power management architecture consisting of central and local power management engines. The central power management engine (CPME) dynamically assigns power budgets to function units and controls their voltage/frequency scaling, according to the overall power limit and real-time performance demands. On the other hand, the local power management engines (LPME), located at various function units such as computer cores and DMA engines, monitor representative power and performance indicators and deliver real-time power regulation accordingly.

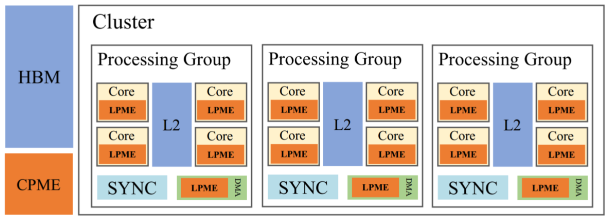


Fig. 8. The power management architecture with central and local power management engines, i.e., CPME & LPMEs (for one cluster only).

Interactions between CPME and LPMEs permit global power status awareness and on-demand power adjustment, balancing the performance and energy efficiency.

1) Power Integrity Management: Based on the overall power limit, power integrity management [59], [86] aims to prevent any integrity risks. On system booting, CPME conservatively assigns a baseline power budget to every function unit (i.e., the minimal power budget the function unit requires) and reserves the remaining budgets for runtime distribution. LPME ensures that real-time power consumption is always under the given power budget. By monitoring local events and status, LPME projects the required power consumption to process current workloads. If it tends to exceed the assigned power budget, LPME inserts stalls/bubbles to the workload pipeline based on a negative feedback loop.

LPME can apply for additional budgets from CPME reserved, to obtain better performance. Fig. 9 illustrates the dynamic process of power budget reallocation. LPME keeps tracking the ratio of pipeline stalls/bubbles in several continuous observation windows. When the ratio in the current observation window exceeds the predefined budget-borrow threshold, LPME determines whether the function unit has become a system performance bottleneck by evaluating the historical ratio records. If frequent pipeline stalls are observed in M out of N recent observation windows, LPME sends a request to CPME, asking for more power budgets. CPME processes LPME's request based on its power management model, assuring the overall power integrity is risk-free. In addition, when the total assigned power budget exceeds the function unit's actual need, LPME reserves the adequate power budget and returns the remaining ones to CPME.

2) Energy Efficiency Management: DTU 2.0 applies a customized dynamic voltage and frequency scaling (DVFS) strategy [23], [51], [77] to optimize power efficiency, empha-

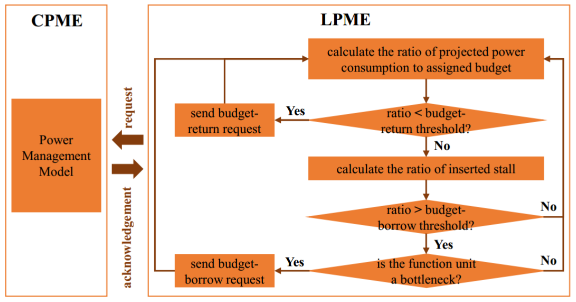

CPME
Power Management Model
request
acknowledgement
LPME
calculate the ratio of projected power consumption to assigned budget
ratio < budget-return threshold?
Yes
send budget-return request
No
calculate the ratio of inserted stall
ratio > budget-borrow threshold?
No
Yes
is the function unit a bottleneck?
Yes
send budget-borrow request
No

Fig. 9. Power integrity strategy in DTU 2.0.

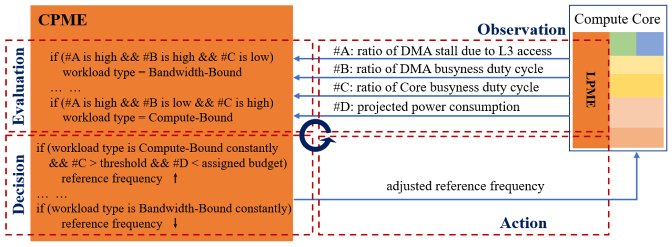

CPME
Evaluation
if (#A is high && #B is high && #C is low)
workload type = Bandwidth-Bound
… …
if (#A is high && #B is low && #C is high)
workload type = Compute-Bound
Decision
if (workload type is Compute-Bound constantly
&& #C > threshold && #D < assigned budget)
reference frequency ↑
… …
if (workload type is Bandwidth-Bound constantly)
reference frequency ↓
Observation
#A: ratio of DMA stall due to L3 access
#B: ratio of DMA busyness duty cycle
#C: ratio of Core busyness duty cycle
#D: projected power consumption
Compute Core
LPME
adjusted reference frequency
Action

Fig. 10. Power efficiency strategy in DTU 2.0.

sizing correlations between computation and memory access of the workload.

Fig. 10 illustrates the whole process. At the *Observation* stage, LPME collects the real-time status of the compute core and its paired DMA engine in each observation window. It sends crucial information to the CPME, such as the projected power consumption, the ratio of busyness duty cycles, and the ratio of DMA stalls caused by L3 memory access. At the *Evaluation* stage, CPME assesses the type of current workload (i.e., compute-bound, bandwidth-bound, or balanced) according to the observed ratios. Then, at the *Decision* stage, it determines whether to change the frequency of the compute core by looking back at the historical status in the last few observation windows. For example, if the workload is considered compute-bound in continuous observation windows, meanwhile the ratio of busyness duty cycles exceeds a specified threshold, CPME increases the compute core's frequency accordingly at the *Action* stage. With the 4-stage closed-loop control of frequency scaling, DTU 2.0's performance and power consumption dynamically adjust for different workloads.

V. The Implementation of Cloudblazer i20

A. Accelerator Card

The Cloudblazer i20 accelerator card integrates an onboard DTU 2.0 chip and a PCIe Gen4.0x16 interface to collaborate with the CPU host. An 8-pin 12V PCIe power connector is adopted for power supplement, and the board TDP is 150W. The accelerator card is air-cooled with a dual-direction cooling air flow path (left-to-right and right-to-left), enabling its convenient deployment in the AI cloud. It can be integrated into many server vendors' popular products, such as the NF5468M6 server from Inspur and the UniServer R5300 G5 server from H3C.

B. Software Support

We developed a dedicated software stack to support convenient and efficient utilization of the DTU and the Cloudblazer accelerator. Fig. 11 gives its structure. The essential components include the graph compiler (i.e., TopsInference), the operator compiler (i.e., TopsEngine), highly optimized libraries (e.g., TopsDNN and TopsRuntime), driver, debugger, and profiler.

The graph compiler *TopsInference* leverages ONNX [9] to import/convert DNN models developed with various frameworks, such as Pytorch [69] and TensorFlow [18]. The generated computation graph is optimized through automatic operator fusion [46], [60], [63], [76], to eliminate unnecessary materialization and scan of intermediate values and benefit from the increased register/memory capacity. Currently, the strategy of operator fusion is designed with expert knowledge. We consider enabling search-based automatic operator fusion soon as a supplementary approach to discovering more beneficial solutions.

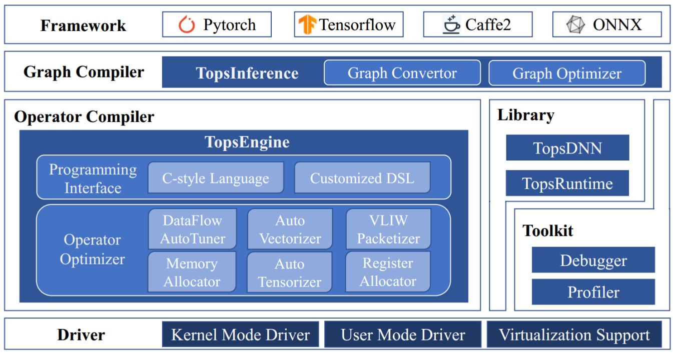


Fig. 11. The software stack for DTU and Cloudblazer accelerator.

The operator compiler *TopsEngine* provides two sets of programming interfaces to the operator developers, i.e., a user-friendly C-style language with sufficient built-in libraries and a customized domain-specific language (DSL) exposing the architecture design details. Similar to CUDA [6], the developer needs to allocate device memory and launch the kernel to interact with accelerator from the host CPU. The purpose of the C-style programming interfaces is to ease the development, especially for programmers already familiar with C/C++ programming languages. On the other hand, the focus of the DSL is to allow programmers to access the underlying hardware directly. Compared to the C-style language, programming with DSL tends to obtain higher performance because developers can utilize and fine-tune hardware-related features as they desire. Developers shall choose the preferred programming interfaces according to their expertise in DTU's hardware design and the development target/cycle.

TopsEngine assists developers in generating efficient data flow and kernel code with the following features:

- Auto-tuning on data flows [25], [84], [94], [96] searches for efficient data tiling solutions that benefit most from DTU's memory hierarchy and bandwidth. The generated data flows are mapped to specific DMA transactions, performing data layout transformations on the fly. By pipelining the computation and data flow, DTU's computational power is effectively utilized.
- Shared memory allocation is optimized with considerations of: (1) the L2 memory affinity to compute cores in the processing group and (2) the L3 memory affinity to clusters. As illustrated in Fig. 2, L2 memory's 4 read/write ports are bonded to 4 computer cores in each processing group. The latency of accessing different memory locations varies for compute cores through their dedicated memory ports. Therefore, TopsEngine allocates shared L2 memory wisely to take advantage of the

TABLE III
DNN BENCHMARKS FOR EVALUATION.

Category	DNNs	Source	Input Size
Object Detection	Yolo v3 [72]	Pytorch [17]	3x608x608
	CenterNet [29]	Pytorch [2]	3x512x512
	Retinaface [27]	Pytorch [12]	3x640x640
Image Classification	VGG16 [78]	Pytorch [11]	3x224x224
	Resnet50 v1.5 [41]	Pytorch [11]	3x224x224
	Inception v4 [80]	Tensorflow [14]	3x299x299
Segmentation	Unet [75]	Tensorflow [16]	3x512x512
Super Resolution	SRResnet [56]	Tensorflow [13]	224x224x3
NLP	Bert large [28]	Tensorflow [1]	384
Speech Recognition	Conformer [38]	Pytorch [5]	80x401

memory affinity and improve data access efficiency.

- Auto-vectorization aims to exploit the SIMD parallelism at the loop [65], [98] and super-word [53], [97] levels. The generated code utilizes vector instructions that operate on the vector engine. In particular, TopsEngine ensures transcendental functions the DTU supports are properly vectorized to improve overall performance.
- Auto-tensorization [90], [95] is developed to harness DTU's matrix engine. It targets special computation patterns, such as matrix multiplication and convolution. Loop transformations, e.g., loop tiling and loop switching, are applied to help identify VMM computations according to various vector/matrix shapes the matrix engine supports.
- Register allocator tries to avoid register bank conflicts that lead to pipeline stalls [35], [88]. By preventing register bank conflicts during compilation, the VLIW pipeline can access required instruction operands without incurring hardware/software overheads [19], [21], [71].
- VLIW packetizer is enhanced along with the instruction scheduler. We made enhancements on alias analysis [79] to reduce ambiguous dependencies. Independent instructions are discovered and packed into one instruction packet, then issued all at once. Besides the improvements in runtime performance, kernel code size is optimized.

Moreover, the software stack offers several high-performance libraries. *TopsDNN* includes expert-tuned DNN operators developed with DTU's assembly or C intrinsic to utilize powerful SFU and matrix/vector engines. Dynamic tensor and shape inference have been supported. *TopsRuntime* is a library for DTU runtime management. It triggers resource allocation and task execution, which is critical for efficient deployment of heterogeneous systems.

VI. EVALUATION

To demonstrate the effectiveness of Cloudblazer i20's hardware and software, we validated its performance, efficiency, and flexibility, respectively.

TABLE IV
AI INFERENCE ACCELERATORS ADOPTED FOR EVALUATION.

	Cloudblazer i10	Nvidia T4 [8]	Nvidia A10 [7]
FP32 Perf (TFLOPS)	20	8.1	31.2
FP16 Perf (TFLOPS)	80	65	125
INT8 Perf (TOPS)	80	130	250
Memory (GB)	16	16	24
Bandwidth (GB/s)	512	320	600
Board TDP (Watt)	150	70	150
Chip Technology (nm)	12	12	7
Interconnect	PCIe4	PCIe3	PCIe4

A. Experimental Setup

The benchmarks we evaluated are popular DNNs adopted in various domains, as listed in Table III. We use CPU's DNN inference results as the reference, and the differences in inference precision of the tests run on CPU and accelerators are configured as 0.01% for all tested DNNs except for *Bert Large*, which is 0.05%.

We compared the Cloudblazer i20 with its predecessor Cloudblazer i10 (i.e., the AI inference accelerator with 1 DTU 1.0 onboard). Meanwhile, we evaluated Nvidia's T4 and A10 GPUs that both target data centers and clouds. Table IV gives the specifications of these accelerators.

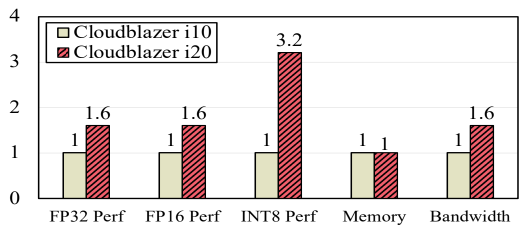
The host machine we used for evaluation equips an Intel Xeon Gold 6240 CPU (2.60GHz) running with Ubuntu 18.04. To test Nvidia GPUs, we utilized CUDA 11.4 and TensorRT 8.0.1.6. Performance measurements are obtained with the *trtexec* tool [15].

B. Performance

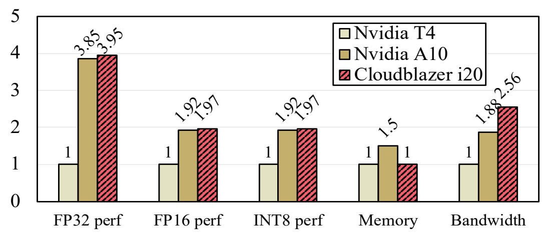
We firstly compare some key performance features of different platforms. As depicted in Fig. 12, Cloudblazer i20 is the most powerful accelerator in terms of the peak performance on FP32, FP16, and INT8 data types. Its memory bandwidth is 1.6x, 2.56x, and 1.36x higher than Cloudblazer i10, Nvidia T4, and A10, respectively. Nvidia A10 has the largest memory capacity (1.5x larger than others).

Fig. 13 compares the latency of DNN models running with different accelerators. We omit the results of Cloudblazer i10, which performs worse than Cloudblazer i20 for all tested DNNs. On average, Cloudblazer i20 outperforms Nvidia T4 and A10 GPUs with the *GeoMean* speedups of 2.22x and 1.16x, respectively. The highest speedup it introduces is observed with the *SRResnet* model, which is 4.34x (2.37x) over Nvidia T4 (A10). The significant performance improvements achieved by Cloudblazer i20 rely on effective enhancement and utilization of the matrix engine, memory capacity, and bandwidth, as well as optimizations applied by the software stack (such as operator fusion and high performance libraries).

Nvidia A10 consistently performs better than T4 for the evaluated benchmarks. It outperforms Cloudblazer i20 on 3 out of 10 evaluated DNNs. Through further tuning, we believe Cloudblazer i20 could deliver better performance on these DNNs too.



(a) Cloudblazer i20 v.s. i10 (normalized with i10).



(b) Cloudblazer i20 v.s. Nvidia T4/A10 (normalized with T4).

Fig. 12. Comparisons of peak performance, memory capacity, and bandwidth across different platforms.

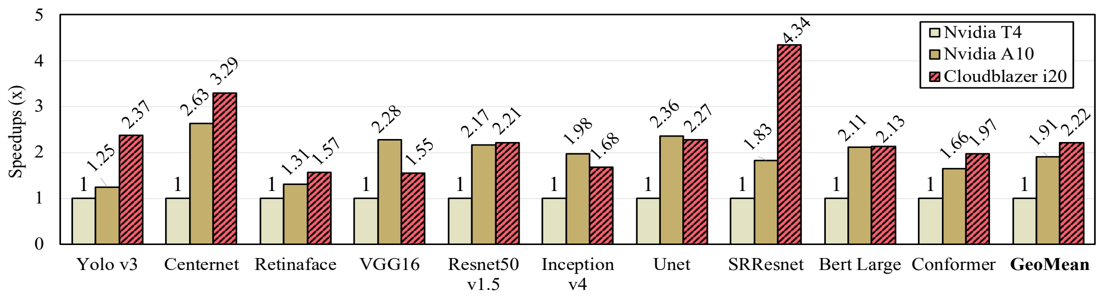


Fig. 13. Comparisons of DNN latency across different platforms (normalized with T4 and omitted i10's results, all DNNs use FP16).

C. Efficiency

In Fig. 14, we compare Cloudblazer i20's TDP and power efficiency (i.e., peak performance/TDP) with the other 3 platforms. Nvidia T4 has the lowest TDP, around 47% of the others. Its power efficiency on FP16 (INT8) is 1.11x (1.11x), 1.74x (3.48x), and 1.09x (1.09x) higher than Nvidia A10, Cloudblazer i10, and i20. However, for FP32, Cloudblazer i20's power efficiency is the best, which is 1.6x, 1.84x, and 1.03x higher than Cloudblazer i10, Nvidia T4, and A10.

The power efficiencies of running DNNs on different platforms are compared in Fig. 15. Similarly, we omit Cloudblazer i10's results. Overall, Cloudblazer i20's power efficiency is better than Nvidia T4 and A10 with 4% and 17% improvements on average. The *SRResnet* model shows its largest improvement on power efficiency, which is 2.03x and 2.39x higher than Nvidia T4 and A10.

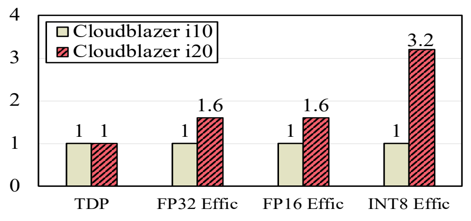
Cloudblazer i20s TDP is more than doubled compared to Nvidia T4. However, its power efficiency is better than Nvidia T4 for half of the tested DNNs. Recall that Nvidia T4 has the best power efficiency on FP16 in theory, as shown in Fig. 14(b). By obtaining adequate DNNs performance, Cloudblazer i20 can achieve higher power efficiency for more DNNs. It is also worth noting that, compared to Nvidia A10 made with the 7nm technology, Cloudblazer i20 delivers better results in terms of performance and power efficiency on average.

D. Discussion

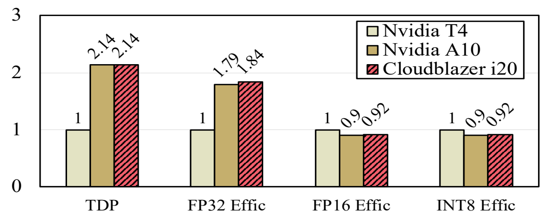
Object detection v.s. Image classification: For each of these two important domains in computer vision, we evaluated 3 representative DNNs. According to the performance results shown in Fig. 13, Cloudblazer i20 performs the best for all 3 DNNs of object detection (i.e., *Yolo v3*, *Centernet*, and *Retinaface*), but it is less impressive in image classification (e.g., *VGG16* and *Inception v4*) than Nvidia A10 GPU. Object detection is well known as a more complex and time-consuming workload than image classification [68]. Based on our profiling statistics, the average percentage of operators with high computational density (i.e., matrix convolution and multiplication) in object detection DNNs is less than image classification DNNs (around 81%). However, their input sizes are more than 2x larger, leading to more computation and bandwidth costs. With the powerful matrix/vector engine and improved bandwidth, Cloudblazer i20 is able to deliver satisfying performance. On the other hand, Nvidia A10 GPU achieves better results in image classification, especially for *VGG16*, owing to its strengthened computational power and the kernel libraries well-optimized for typical CNN operators.

Latency v.s. Throughput: Unlike DNN training, which is primarily throughput-bound, DNN inference is much more latency-sensitive to satisfy service-level agreements [33]. Therefore, we focused on evaluating DNN's latency in Section VI-B. However, increasing computation intensity generally improves hardware efficiency. Within DNN's latency limits, it is desirable to run DNNs with larger batch sizes [40]. The GPU architecture is famous for its optimized throughput. CNN models that exhibit high levels of parallelism and data reuse are suitable for accelerating with GPUs [33]. The Cloudblazer i20 improves its throughput by supporting multi-task/tenancy with parallel and isolated processing groups. We tested the *VGG16* model for image classification that has less strict latency requirements [68], using batch sizes equaling 8 and 16. Cloudblazer i20 is able to perform better than Nvidia's A10 with improvements of 1.11x and 1.17x, respectively. The results reveal the potential of improving task throughput with multi-batches on Cloudblazer i20.

Power management ON v.s. OFF: To evaluate the benefits introduced by the power management strategy presented in Section IV-F, we evaluated the *Resnet50 v1.5* and *Bert Large* models, representative in computer vision and text processing. We compared two configurations: (1) with power management



(a) Cloudblazer i20 v.s. i10 (normalized with i10).



(b) Cloudblazer i20 v.s. Nvidia T4/A10 (normalized with T4).

Fig. 14. Comparisons of power and energy efficiency (Perf/TDP) across different platforms.

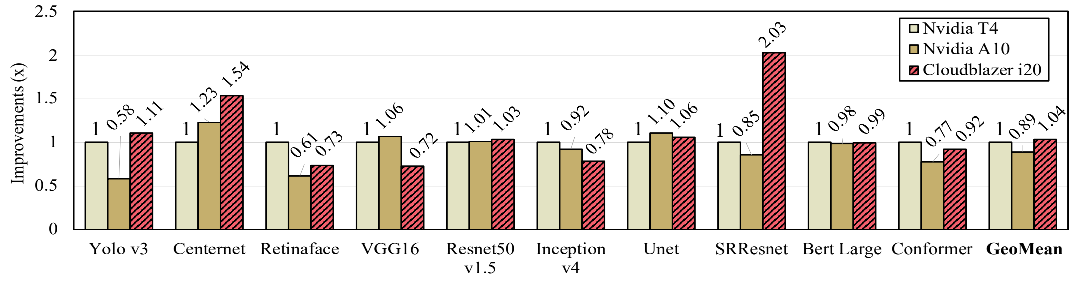


Fig. 15. Comparisons of DNN energy efficiency (Perf/TDP) across different platforms (normalized with T4 and omitted i10's results, all DNNs use FP16).

switched on, for which the clock frequency dynamically adjusts from 1.0 GHz to 1.4 GHz, and (2) with power management switched off, for which the clock frequency is fixed as 1.4 GHz to get the maximal performance. Under these two settings, we observed comparable performance with only 0.85% and 3.2% performance drop when power management is turned on. However, in terms of energy efficiency, we saw 13% improvements for both DNNs, which demonstrated the effectiveness of our customized on-demand frequency scaling.

Flexibility v.s. Diversity: A significant challenge faced by the DSA designers is the fast pace that the DNN workload evolves. Models with new data types or varying tensor shapes keep emerging [68]. AI accelerators must be sufficiently generic and programmable to provide sustained performance across diverse DNNs. In this paper, we have seen 10 typical DNN models in 6 diverse domains successfully deployed on the Cloudblazer i20, with promising performance and energy efficiency improvements. The flexibility that Cloudblazer i20 provides to adapt to various workloads is multifaceted: (1) the compute core supports different vector/matrix shapes for VMM computation patterns; (2) commonly seen tensor layout transformations are facilitated by the DMA engine; (3) the synchronization engine enables comprehensive synchronization patterns; (4) hardware resources abstraction provides feasible workload mapping and deployment; moreover, (5) the software stack provides abundant programming interfaces for developers to harness the underlying hardware, making their DNN development customizable and extendable. All these hardware and software design decisions contribute to the flexibility that Cloudblazer i20 offers.

VII. Related Work

Many industrial processors/accelerators have been developed in recent years to benefit DNN applications.

Nvidia T4 GPU [66] is a low-power inference accelerator that mainly targets the cloud inference market. Nvidia A10 GPU [67] is a recent upgrade with increased performance and TDP for servers and data centers. They adopt the Single Instruction Multiple Thread (SIMT) architecture and process hundreds of threads in parallel with the CUDA and Tensor cores. Dynamic warp switching is applied to hide the memory access latency, exploiting the data parallelism from abundant threads. We made comparison with them in Section VI.

Google has released its 4th generation Deep Learning DSA (i.e., TPUv4i) for AI inference [48]. It is a 7nm chip with a peak performance up to 138 TFLPOS (for INT8). Its TDP is 175 Watts. Intel's NNP-I 1000 (Spring Hill) [89] is a VLIW-based low-power chip (up to 50 Watts) consisting of 12 inference compute engines. It adopts a LPDDR4 memory and delivers peak performance up to 92 TOPS (for INT8). Habana's Goya [39] is also a VLIW-based chip with a 200 Watts TDP. It only supports FP32 and integers, offering 46% of Nvidia T4's power efficiency [48]. Huawei's Ascend 910 is designed for AI training servers. Its peak performance is 256 TFLOPS, and the TDP is 300 Watts [57]. Compared to them, our Cloudbalzer i20, built upon the novel DSA, delivers competitive performance and/or power efficiency.

VIII. Conclusion

As Dennard scaling and Moore's Law come to an end, we believe hardware-software co-design is a promising solution for constructing effective DSAs that handle rapidly-evolving DNNs [26], [34], [42], [44], [62], [68], [83]. We introduce our latest product, i.e., the Cloudblazer i20 accelerator, equipped with our innovative DTU 2.0. Based on thorough studies on DNN's features and requirement analysis on cloud inference services, we made enhancements to its compute engine, memory hierarchy, data flow/synchronization engine, resource assignment and power management. The software stack assists developers in flexibly exploiting the powerful hardware with diverse programming interfaces and automatic optimizations. Evaluated with 10 representative DNNs, Cloudblazer i20 outperforms Nvidia's T4 (A10) GPU by 2.22x (1.16x) in performance and 1.04x (1.17x) in power efficiency on average.

References

[1] "Bert models from google research," https://github.com/google-research/bert, accessed: 2022-07-08.

[2] "CenterNet models," https://github.com/xingyizhou/CenterNet, accessed: 2022-07-08.

[3] "Cloud computing services - amazon web services (aws)," https://aws.amazon.com/, accessed: 2022-07-08.

[4] "Cloud computing services - microsoft azure," https://azure.microsoft.com/en-us/, accessed: 2022-07-08.

[5] "Conformer models," https://github.com/NVIDIA/NeMo/tree/main/examples/asr, accessed: 2022-07-08.

[6] "Cuda programming guide," https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html, accessed: 2022-07-08.

[7] "Nvidia A10 GPU," https://www.nvidia.com/en-us/data-center/products/a10-gpu/, accessed: 2022-07-08.

[8] "Nvidia T4 GPU," https://www.nvidia.com/en-us/data-center/tesla-t4/, accessed: 2022-07-08.

[9] "Open neural network exchange," https://www.onnx.ai, accessed: 2022-07-08.

[10] "Permutation matrix," https://mathworld.wolfram.com/PermutationMatrix.html, accessed: 2022-07-08.

[11] "Pytorch image classification models," https://github.com/pytorch/vision/tree/main/torchvision/models, accessed: 2022-07-08.

[12] "Retinaface models," https://https://github.com/biubug6/Pytorch_Retinaface, accessed: 2022-07-08.

[13] "SRResnet models," https://github.com/tensorlayer/srgan/, accessed: 2022-07-08.

[14] "Tensorflow models," https://github.com/tensorflow/models/tree/master/research/slim/nets, accessed: 2022-07-08.

[15] "Tensorrt command-line wrapper: trtexec," https://www.ccoderun.ca/programming/doxygen/tensorrt/md_TensorRT_samples_opensource_trtexec_README.html, accessed: 2022-07-08.

[16] "Unet models," https://catalog.ngc.nvidia.com/orgs/nvidia/resources/unet_industrial_for_tensorflow, accessed: 2022-07-08.

[17] "Yolo v3 models," https://github.com/ultralytics/yolov3, accessed: 2022-07-08.

[18] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16. USA: USENIX Association, 2016, p. 265283.

[19] M. Abdel-Majeed, A. Shafaei, H. Jeon, M. Pedram, and M. Annavaram, "Pilot register file: Energy efficient partitioned register file for gpus," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 589–600.

[20] M. Anderson, B. Chen, S. Chen, S. Deng, J. Fix, M. Gschwind, A. Kalaiah, C. Kim, J. Lee, J. Liang, H. Liu, Y. Lu, J. Montgomery, A. Moorthy, S. Nadathur, S. Naghshineh, A. Nayak, J. Park, C. Petersen, M. Schatz, N. Sundaram, B. Tang, P. Tang, A. Yang, J. Yu, H. Yuen, Y. Zhang, A. Anbudurai, V. Balan, H. Bojja, J. Boyd, M. Breitbach, C. Caldato, A. Calvo, G. Catron, S. Chandwani, P. Christeas, B. Cottel, B. Coutinho, A. Dalli, A. Dhanotia, O. Duncan, R. Dzhabarov, S. Elmir, C. Fu, W. Fu, M. Fulthorp, A. Gangidi, N. Gibson, S. Gordon, B. P. Hernandez, D. Ho, Y.-C. Huang, O. Johansson, S. Juluri, S. Kanaujia, M. Kesarkar, J. Killinger, B. Kim, R. Kulkarni, M. Lele, H. Li, H. Li, Y. Li, C. Liu, J. Liu, B. Maher, C. Mallipedi, S. Mangla, K. K. Matam, J. Mehta, S. Mehta, C. Mitchell, B. Muthiah, N. Nagarkatte, A. Narasimha, B. Nguyen, T. Ortiz, S. Padmanabha, D. Pan, A. Poojary, Ye, Qi, O. Raginel, D. Rajagopal, T. Rice, C. Ross, N. Rotem, S. Russ, K. Shah, B. Shan, H. Shen, P. Shetty, K. Skandakumaran, K. Srinivasan, R. Sumbaly, M. Tauberg, M. Tzur, S. Verma, H. Wang, M. Wang, B. Wei, A. Xia, C. Xu, M. Yang, K. Zhang, R. Zhang, M. Zhao, W. Zhao, R. Zhu, A. Mathews, L. Qiao, M. Smelyanskiy, B. Jia, and V. Rao, "First-generation inference accelerator deployment at facebook," 2021. [Online]. Available: https://arxiv.org/abs/2107.04140

[21] H. Asghari Esfeden, F. Khorasani, H. Jeon, D. Wong, and N. Abu-Ghazaleh, "Corf: Coalescing operand register file for gpus," ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 701714. [Online]. Available: https://doi.org/10.1145/3297858.3304026

[22] M. D. Avgerinou, P. Bertoldi, and L. Castellazzi, "Trends in data centre energy consumption under the european code of conduct for data centre energy efficiency," *Energies*, vol. 10, pp. 1–18, 2017.

[23] K. R. Basireddy, A. K. Singh, B. M. Al-Hashimi, and G. V. Merrett, "Adamd: Adaptive mapping and dvfs for energy-efficient heterogeneous multicores," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2206–2217, 2020.

[24] S. Bavikadi, A. Dhavlle, A. Ganguly, A. Haridass, H. Hendy, C. Merkel, V. J. Reddi, P. R. Sutradhar, A. Joseph, and S. M. Pudukotai Dinakarrao, "A survey on machine learning accelerators and evolutionary hardware platforms," *IEEE Design & Test*, vol. 39, no. 3, pp. 91–116, 2022.

[25] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Tvm: An automated end-to-end optimizing compiler for deep learning," ser. OSDI'18. USA: USENIX Association, 2018, p. 579594.

[26] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, M. Abeydeera, L. Adams, H. Angepat, C. Boehn, D. Chiou, O. Firestein, A. Forin, K. S. Gatlin, M. Ghandi, S. Heil, K. Holohan, A. El Husseini, T. Juhasz, K. Kagi, R. K. Kovvuri, S. Lanka, F. van Megen, D. Mukhortov, P. Patel, B. Perez, A. Rapsang, S. Reinhardt, B. Rouhani, A. Sapek, R. Seera, S. Shekar, B. Sridharan, G. Weisz, L. Woods, P. Yi Xiao, D. Zhang, R. Zhao, and D. Burger, "Serving dnns in real time at datacenter scale with project brainwave," *IEEE Micro*, vol. 38, no. 2, pp. 8–20, 2018.

[27] J. Deng, J. Guo, Y. Zhou, J. Yu, I. Kotsia, and S. Zafeiriou, "RetinaFace: Single-stage dense face localisation in the wild," May 2019, arXiv:1905.00641.

[28] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," Oct. 2018, arXiv:1810.04805.

[29] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "CenterNet: Keypoint triplets for object detection," Apr. 2019, arXiv:1904.08189.

[30] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, D. Sylvester, D. Blaauw, R. Das, and R. Iyer, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," *IEEE Micro*, vol. 39, no. 3, pp. 11–19, 2019.

[31] J. A. Fisher, "Very long instruction word architectures and the eli-512," *SIGARCH Comput. Archit. News*, vol. 11, no. 3, p. 140150, jun 1983. [Online]. Available: https://doi.org/10.1145/1067651.801649

[32] B. Fleischer, S. Shukla, M. Ziegler, J. Silberman, J. Oh, V. Srinivasan, J. Choi, S. Mueller, A. Agrawal, T. Babinsky, N. Cao, C.-Y. Chen, P. Chuang, T. Fox, G. Gristede, M. Guillorn, H. Haynie, M. Klaiber, D. Lee, S.-H. Lo, G. Maier, M. Scheuermann, S. Venkataramani, C. Vezyrtzis, N. Wang, F. Yee, C. Zhou, P.-F. Lu, B. Curran, L. Chang, and K. Gopalakrishnan, "A scalable multi- teraops deep learning processor core for ai training and inference," in *2018 IEEE Symposium on VLSI Circuits*, 2018, pp. 35–36.

[33] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, "A configurable cloud-scale dnn processor for real-time ai," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 1–14.

[34] A. Fuchs and D. Wentzlaff, "The accelerator wall: Limits of chip specialization," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 1–14.

[35] M. Gebhart, S. W. Keckler, and W. J. Dally, "A compile-time managed multi-level register file hierarchy," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: Association for Computing Machinery, 2011, p. 465476. [Online]. Available: https://doi.org/10.1145/2155620.2155675

[36] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[37] B. Grot, D. Hardy, P. Lotfi-Kamran, C. Nicopoulos, Y. Sazeides, and B. Falsafi, "Optimizing data-center tco with scale-out processors," *IEEE Micro*, vol. 32, no. 5, p. 163, sep 2012. [Online]. Available: https://doi.org/10.1109/MM.2012.71

[38] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang, "Conformer: Convolution-augmented transformer for speech recognition," 2020. [Online]. Available: https://arxiv.org/abs/2005.08100

[39] Habana, *Goya Inference Platform White Paper*. Habana, 2020.

[40] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang, "Applied machine learning at facebook: A datacenter infrastructure perspective," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 620–629.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[42] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Commun. ACM*, vol. 62, no. 2, p. 4860, jan 2019. [Online]. Available: https://doi.org/10.1145/3282307

[43] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269.

[44] J. Huang, P. Majumder, S. Kim, A. Muzahid, K. H. Yum, and E. J. Kim, "Communication algorithm-architecture co-design for distributed deep learning," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 181–194.

[45] Z. Jelicová and M. Verhelst, "Delta keyword transformer: Bringing transformers to the edge through dynamically pruned multi-head self-attention," 2022. [Online]. Available: https://arxiv.org/abs/2204.03479

[46] Z. Jia, O. Padon, J. Thomas, T. Warszawski, M. Zaharia, and A. Aiken, "Taso: Optimizing deep learning computation with automatic generation of graph substitutions," ser. SOSP '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 4762. [Online]. Available: https://doi.org/10.1145/3341301.3359630

[47] Y. Jiao, L. Han, and X. Long, "Hanguang 800 npu the ultimate ai inference solution for data centers," in *2020 IEEE Hot Chips 32 Symposium (HCS)*, 2020, pp. 1–29.

[48] N. P. Jouppi, D. Hyun Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma, T. Norrie, N. Patil, S. Prasad, C. Young, Z. Zhou, and D. Patterson, "Ten lessons from three generations shaped googles tpuv4i : Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 1–14.

[49] N. P. Jouppi, C. Young, N. Patil, and D. Patterson, "A domain-specific architecture for deep neural networks," *Commun. ACM*, vol. 61, no. 9, p. 5059, aug 2018. [Online]. Available: https://doi.org/10.1145/3154484

[50] 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, D. Killebrew, 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 *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12.

[51] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core dvfs using on-chip switching regulators," in *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, 2008, pp. 123–134.

[52] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 8490, may 2017. [Online]. Available: https://doi.org/10.1145/3065386

[53] S. Larsen and S. Amarasinghe, "Exploiting superword level parallelism with multimedia instruction sets," ser. PLDI '00. New York, NY, USA: Association for Computing Machinery, 2000, p. 145156. [Online]. Available: https://doi.org/10.1145/349299.349320

[54] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[55] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[56] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-realistic single image super-resolution using a generative adversarial network," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 105–114.

[57] H. Liao, J. Tu, J. Xia, H. Liu, X. Zhou, H. Yuan, and Y. Hu, "Ascend: a scalable and unified architecture for ubiquitous deep neural network computing : Industry track paper," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 789–801.

[58] R. Liu and C. Feng, "AI compute chip from enflame," in *2021 IEEE Hot Chips 33 Symposium (HCS)*, 2021, pp. 1–27.

[59] Y. Lu, F. Yang, F. Chen, and P. K. T. Mok, "A distributed power delivery grid based on analog-assisted digital ldos with cooperative regulation and ir-drop reduction," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 8, pp. 2859–2871, 2020.

[60] L. Ma, Z. Xie, Z. Yang, J. Xue, Y. Miao, W. Cui, W. Hu, F. Yang, L. Zhang, and L. Zhou, "Rammer: Enabling holistic deep learning compiler optimizations with rtasks," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 881–897. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/ma

[61] A. Mishra, J. A. Latorre, J. Pool, D. Stosic, D. Stosic, G. Venkatesh, C. Yu, and P. Micikevicius, "Accelerating sparse deep neural networks," 2021. [Online]. Available: https://arxiv.org/abs/2104.08378

[62] D. Mudigere, Y. Hao, J. Huang, Z. Jia, A. Tulloch, S. Sridharan, X. Liu, M. Ozdal, J. Nie, J. Park, L. Luo, J. A. Yang, L. Gao, D. Ivchenko, A. Basant, Y. Hu, J. Yang, E. K. Ardestani, X. Wang, R. Komuravelli, C.-H. Chu, S. Yilmaz, H. Li, J. Qian, Z. Feng, Y. Ma, J. Yang, E. Wen, H. Li, L. Yang, C. Sun, W. Zhao, D. Melts, K. Dhulipala, K. Kishore, T. Graf, A. Eisenman, K. K. Matam, A. Gangidi, G. J. Chen, M. Krishnan, A. Nayak, K. Nair, B. Muthiah, M. khorashadi, P. Bhattacharya, P. Lapukhov, M. Naumov, A. Mathews, L. Qiao, M. Smelyanskiy, B. Jia, and V. Rao, "Software-hardware co-design for fast and scalable training of deep learning recommendation models," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 9931011. [Online]. Available: https://doi.org/10.1145/3470496.3533727

[63] W. Niu, J. Guan, Y. Wang, G. Agrawal, and B. Ren, "Dnnfusion: Accelerating deep neural networks execution with advanced operator fusion," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, ser. PLDI 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 883898. [Online]. Available: https://doi.org/10.1145/3453483.3454083

[64] T. Norrie, N. Patil, D. H. Yoon, G. Kurian, S. Li, J. Laudon, C. Young, N. Jouppi, and D. Patterson, "The design process for google's training chips: Tpuv2 and tpuv3," *IEEE Micro*, vol. 41, no. 2, pp. 56–63, 2021.

[65] D. Nuzman, I. Rosen, and A. Zaks, "Auto-vectorization of interleaved data for simd," in *Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 132143. [Online]. Available: https://doi.org/10.1145/1133981.1133997

[66] NVIDIA, *NVIDIA T4 70W Low Profile PCIe GPU Accelerator (PB-09256-001_v05)*. NVIDIA, 2020.

[67] NVIDIA, *NVIDIA A10 GPU Accelerator (PB-10415-001_v04)*. NVIDIA, 2022.

[68] J. Park, M. Naumov, P. Basu, S. Deng, A. Kalaiah, D. Khudia, J. Law, P. Malani, A. Malevich, S. Nadathur, J. Pino, M. Schatz, A. Sidorov, V. Sivakumar, A. Tulloch, X. Wang, Y. Wu, H. Yuen, U. Diril, D. Dzhulgakov, K. Hazelwood, B. Jia, Y. Jia, L. Qiao, V. Rao, N. Rotem, S. Yoo, and M. Smelyanskiy, "Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications," 2018.

[69] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[70] Z. Qu, L. Liu, F. Tu, Z. Chen, Y. Ding, and Y. Xie, "Dota: Detect and omit weak attentions for scalable transformer acceleration,"

in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 1426. [Online]. Available: https://doi.org/10.1145/3503222.3507738

[71] A. M. Radaideh and P. V. Gratz, "Cmrc: Comprehensive microarchitectural register coalescing for gpgpus," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 1803–1808.

[72] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," Apr. 2018, arXiv:1804.02767.

[73] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "Hybrid method for minimizing service delay in edge cloud computing through vm migration and transmission power control," *IEEE Trans. Comput.*, vol. 66, no. 5, p. 810819, may 2017. [Online]. Available: https://doi.org/10.1109/TC.2016.2620469

[74] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "Infaas: A model-less and managed inference serving system," 2019. [Online]. Available: https://arxiv.org/abs/1905.13348

[75] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 234–241.

[76] N. Rotem, J. Fix, S. Abdulrasool, G. Catron, S. Deng, R. Dzhabarov, N. Gibson, J. Hegeman, M. Lele, R. Levenstein, J. Montgomery, B. Maher, S. Nadathur, J. Olesen, J. Park, A. Rakhov, M. Smelyanskiy, and M. Wang, "Glow: Graph lowering compiler techniques for neural networks," 2018. [Online]. Available: https://arxiv.org/abs/1805.00907

[77] A. Sethia and S. Mahlke, "Equalizer: Dynamic tuning of gpu resources for efficient execution," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 647–658.

[78] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," Sep. 2014, arXiv:1409.1556.

[79] Y. Sui, X. Fan, H. Zhou, and J. Xue, "Loop-oriented array- and field-sensitive pointer analysis for automatic simd vectorization," *SIGPLAN Not.*, vol. 51, no. 5, p. 4151, jun 2016. [Online]. Available: https://doi.org/10.1145/2980930.2907957

[80] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI'17. AAAI Press, 2017, p. 42784284.

[81] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[82] M. Theobald, G. Weikum, and R. Schenkel, "Top-k query evaluation with probabilistic guarantees," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, ser. VLDB '04. VLDB Endowment, 2004, p. 648659.

[83] B. W. Thompto, D. Q. Nguyen, J. E. Moreira, R. Bertran, H. Jacobson, R. J. Eickemeyer, R. M. Rao, M. Goulet, M. Byers, C. J. Gonzalez, K. Swaminathan, N. R. Dhanwada, S. M. Muller, A. Wagner, S. K. Sadasivam, R. K. Montoye, W. J. Starke, C. G. Zoellin, M. S. Floyd, J. Stuecheli, N. Chandramoorthy, J.-D. Wellman, A. Buyuktosunoglu, M. Pflanz, B. Sinharoy, and P. Bose, "Energy efficiency boost in the ai-infused power10 processor," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 29–42.

[84] N. Vasilache, O. Zinenko, T. Theodoridis, P. Goyal, Z. DeVito, W. S. Moses, S. Verdoolaege, A. Adams, and A. Cohen, "Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions," 2018. [Online]. Available: https://arxiv.org/abs/1802.04730

[85] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 60006010.

[86] A. Vijayakumar, V. C. Patil, and S. Kundu, "An efficient method for clock skew scheduling to reduce peak current," in *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*, 2016, pp. 505–510.

[87] N. Vijaykumar, E. Ebrahimi, K. Hsieh, P. B. Gibbons, and O. Mutlu, "The locality descriptor: A holistic cross-layer abstraction to express data locality in gpus," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 829–842.

[88] D. Voitsechov, A. Zulfiqar, M. Stephenson, M. Gebhart, and S. W. Keckler, "Software-directed techniques for improved gpu register file utilization," *ACM Trans. Archit. Code Optim.*, vol. 15, no. 3, sep 2018. [Online]. Available: https://doi.org/10.1145/3243905

[89] O. Wechsler, M. Behar, and B. Daga, "Spring hill (nnp-i 1000) intels data center inference chip," in *2019 IEEE Hot Chips 31 Symposium (HCS)*, 2019, pp. 1–12.

[90] J. Weng, A. Jain, J. Wang, L. Wang, Y. Wang, and T. Nowatzki, "Unit: Unifying tensorized instruction compilation," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021, pp. 77–89.

[91] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 818–833.

[92] S. Zhang, C. Sun, and Z. He, "Listmerge: Accelerating top-k aggregation queries over large number of lists," in *Proceedings, Part II, of the 21st International Conference on Database Systems for Advanced Applications - Volume 9643*, ser. DASFAA 2016. Berlin, Heidelberg: Springer-Verlag, 2016, p. 6781. [Online]. Available: https://doi.org/10.1007/978-3-319-32049-6_5

[93] X. Zhao, M. Jahre, and L. Eeckhout, *HSM: A Hybrid Slowdown Model for Multitasking GPUs*. New York, NY, USA: Association for Computing Machinery, 2020, p. 13711385. [Online]. Available: https://doi.org/10.1145/3373376.3378457

[94] L. Zheng, C. Jia, M. Sun, Z. Wu, C. H. Yu, A. Haj-Ali, Y. Wang, J. Yang, D. Zhuo, K. Sen, J. E. Gonzalez, and I. Stoica, "Ansor: Generating High-Performance tensor programs for deep learning," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 863–879. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/zheng

[95] S. Zheng, R. Chen, A. Wei, Y. Jin, Q. Han, L. Lu, B. Wu, X. Li, S. Yan, and Y. Liang, "Amos: Enabling automatic mapping for tensor computations on spatial accelerators with hardware abstraction," ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 874887. [Online]. Available: https://doi.org/10.1145/3470496.3527440

[96] S. Zheng, Y. Liang, S. Wang, R. Chen, and K. Sheng, "Flextensor: An automatic schedule exploration and optimization framework for tensor computation on heterogeneous system," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: Association for Computing Machinery, 2020, p. 859873. [Online]. Available: https://doi.org/10.1145/3373376.3378508

[97] H. Zhou and J. Xue, "A compiler approach for exploiting partial simd parallelism," *ACM Trans. Archit. Code Optim.*, vol. 13, no. 1, mar 2016. [Online]. Available: https://doi.org/10.1145/2886101

[98] H. Zhou and J. Xue, "Exploiting mixed simd parallelism by reducing data reorganization overhead," in *2016 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2016, pp. 59–69.