2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 978-1-6654-2235-2/20/$31.00 ©2021 IEEE | DOI: 10.1109/HPCA51647.2021.00071

Ascend: a Scalable and Unified Architecture for Ubiquitous Deep Neural Network Computing

Industry Track Paper

Heng Liao*
Huawei
China
liao.heng@hisilicon.com

Jiajin Tu
Huawei
China
tujiajin@huawei.com

Jing Xia
Huawei
China
dio.xia@hisilicon.com

Hu Liu
Huawei
China
liu_hu@hisilicon.com

Xiping Zhou
Huawei
China
zhouxiping@hisilicon.com

Honghui Yuan
Huawei
China
yuanhonghui@huawei.com

Yuxing Hu
Huawei
China
huyuxing1@huawei.com

ABSTRACT

Deep neural networks (DNNs) have been successfully applied to a great variety of applications, ranging from small IoT devices to large scale services in a data center. In order to improve the efficiency of processing these DNN models, dedicated hardware accelerators are required for all these scenarios. Theoretically, there exists an optimized acceleration architecture for each application. However, considering the cost of chip design and corresponding tool-chain development, researchers need to trade off between efficiency and generality. In this work, we demonstrate that it is practical to use a unified architecture, called Ascend, to support those applications, ranging from IoT devices to data-center services. We provide a lot of design details to explain that the success of Ascend relies on contributions from different levels. First, heterogeneous computing units are employed to support various DNN models. And the datapath is adapted according to the requirement of computing and data access. Second, when scaling the Ascend architecture from a single core to a cluster containing thousands of cores, it involves design efforts, such as memory hierarchy and system level integration. Third, a multi-tier compiler, which provides flexible choices for developers, is the last critical piece. Experimental results show that using accelerators based on the Ascend architecture can achieve comparable or even better performance in different applications. In addition, various chips based on the Ascend architecture have been successfully commercialized. More than 100 million chips have been used in real products.

1. INTRODUCTION AND CHALLENGES

Deep neural networks (DNNs) have been widely applied to various applications, such as natural language processing [26, 37], autonomous driving [24, 39], robots [25, 27], smart phones [31, 42], intelligent IoT devices [28, 41], etc. However, the successes of these DNN models come with the cost of high computation intensity and storage capacity, which cannot be satisfied by the traditional computing architecture. To overcome this problem, a lot of accelerators have been proposed to improve both performance and energy-efficiency for either training or inference in the fields mentioned above.

Many companies have proposed their customized accelerators for DNN training. Google's TPU [33] and NVIDIA's V100 [1] are two representatives of accelerators on DNN training tasks. For processing DNN on mobile devices, SnapDragon 865 [17] and MediaTek Dimensity 1000 [10] show great performance and energy efficiency. In the field of automotive SoCs, NVIDIA's Xavier [14] and Mobileye's EyeQ [12] are proposed to accelerate DNN computing. Furthermore, many specialized architectures have been proposed for typical deep neural networks in academia. Han et al. [29] proposes an accelerator for LSTM-NNs in NLP. Yamada et al. [40] integrate DNN accelerators for autonomous driving. To accelerate DNN in robots, Amaravati et al. [21] implement a neuromorphic accelerator. Niu et al. [35] accelerate DNN processing on smart phones, and Whatmough et al. [38] optimize DNN accelerators on IoTs.

On the one hand, the aforementioned accelerators follow similar design principles. First, according to the common operations in the DNNs, multiple or massive dedicated processing elements (PEs) are introduced to improve computing efficiency. Second, to match computing throughput, the memory hierarchy is optimized for data reuse exploration. On the other hand, these accelerators were designed for different goals under different application constraints. These design goals can be evaluated with different metrics, such as computing throughput ($GFLOPS$), energy efficiency ($GFLOPS/Watt$), area efficiency($GFLOPS/mm^2$), cost efficiency ($GFLOPS/\$$), etc.

Although an optimized accelerator can be proposed for each application/field, it is impractical to implement each of them, mainly due to the cost consideration. First, as the

* Corresponding author.

978-1-6654-2235-2/21/$31.00 ©2021 IEEE
DOI 10.1109/HPCA51647.2021.00071

Table 1: Ascend Cores, Typical Applications and Networks

Ascend Core	Inf. / Tra.	Applications	Typical Algorithm / Network
Ascend-Tiny	Inference	IoT and smart sensors	Swing Face Detection, Gesture Detection
Ascend-Lite	Inference	IP cameras and smartphones	Mobilenet, Novel Neural Network for Image Signal Processor (ISP)
Ascend-Mini	Inference	Drones, robots, and industrial-level embedded AI	Resnet, VGG
Ascend	Inference & Training	Autonomous driving, smart city and intelligent surveillance, cloud AI inference, model training	MaskRCNN Series, Siamese Tracking, Pointsnet Series
Ascend-Max	Training & Inference	High-performance AI applications, & cloud AI training	BERT, Resnet, Wide and Deep

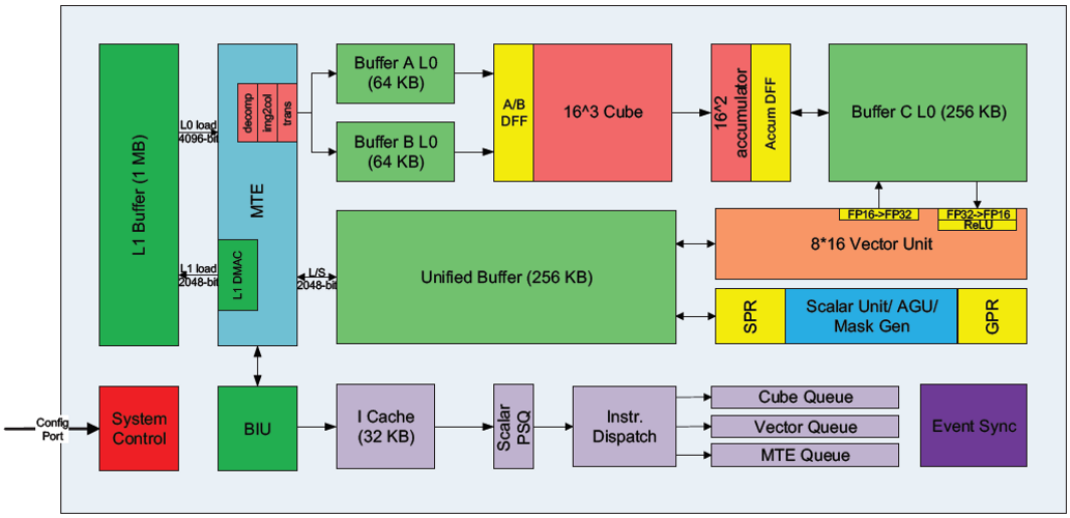

Figure 1: Ascend core (Ascend-Max Configuration)

IC technology scales down, the chip fabrication expense increases exponentially. In addition, most DNN algorithms are developed with several well-known frameworks, such as PyTorch [16], TensorFlow [20], etc. And the deployment of these DNN models on a hardware also highly relies on intermediate software tool-chains, such as compilers and computing libraries [6, 19]. However, the development of these tools can be more time-consuming and cost-consuming than the hardware implementation itself.

Consequently, it is always a critical choice to trade off between the efficiency and the generality before implementing a DNN accelerator. Our company keeps focus on the vision of enabling ubiquitous AI in an all connected intelligent world. As listed in Table 1, the DNN accelerator should drive applications ranging from smart watches, smart phones, smart cars to smart cloud. It is no doubt that a unified scalable architecture, which can be easily "tailored" to fit in different scenarios, is preferred. Here we explicitly define "unified" as one normalized architecture with basic instruction set, and we refer "scalable" as efficient extensions from the core to SOC to server to cluster. However, a lot of design issues should be handled for such a unified approach.

In this work, we propose a unified architecture, called Ascend, to support those applications, ranging from IoT devices to data-center services. We provide a lot of design details to explain that the success of Ascend relies on contributions from different levels. First, heterogeneous computing units are employed to support various DNN models. And the datapath is adapted according to the requirement of computing and data access. Second, when scaling the Ascend architecture from a single core to a cluster containing thousands of cores, it involves design efforts, such as memory hierarchy and system level integration. Third, to support efficient software development with a unified programming model, a multi-tier compiler is also critical to provide flexible choices for developers.

The rest of this paper is organized as follows. In Section 2, the components inside the Ascend core are introduced, including computing units, on-chip memory, control units, etc. In addition, the architecture design principles behind these components are also discussed. In Section 3, various SoCs designs equipped with different Ascend cores are proposed for different applications. Especially, the case of using Ascend-Max for DNN training is presented in details. In Section 4, we discuss several design issues about scaling to the board and cluster level. In Section 5, the software support for the Ascend core is described. This is critical to enable such a unified architecture for various applications. The evaluation results are presented in Section 6, followed by a conclusion.

2. ASCEND CORE DESIGN

The overall architecture of an Ascend core is illustrated

in Figure 1. It is composed of three types of computing units, multiple levels of on-chip memory and corresponding load/store units, instruction management unit, etc.

Figure 1: Ascend core (Ascend-Max Configuration)

2.1 Computing Units

To support different DNN models, accelerators always apply heterogeneous architecture that combines several types of computing units, as shown in Figure 2.

The first type is the scalar unit, which is similar to the classical RISC core.

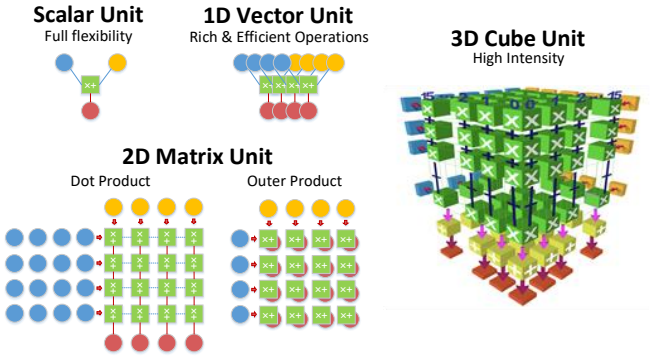

Figure 2: Different Dimension's computing units

Figure 2: Different Dimension's computing units

The second type is the 1D (vector) computing unit. A vector unit is similar to the SIMD unit in a traditional CPU or GPU, which has been widely deployed in the fields of High Performance Computing (HPC) and Computer Vision (CV). It can perform most computing operations in inference and training tasks. However, using a vector unit cannot fully exploit data reuse in the DNN models. Thus, it results in a bandwidth-bound datapath between high density ALU and local memory, which is limited by physical place & route (P&R). As a result, the 1D vector unit is insufficient for the DNN applications that keep increasing requirement for computing power.

To overcome the limitation of the vector unit, 2D computing unit have been utilized for general matrix-matrix multiplication (GEMM). Mathematically, in a 2D unit, there are two methods to accelerate the GEMM. One is dot product based method, which performs GEMM as multiple general matrix-vector multiplication (GEMV). The other one is outer product based method, it transfers GEMM to multiple vector outer product's accumulation.

To further take the highest intensity, a 3D (cube) computing unit is introduced to provide the bulk of computing power. The cube unit is dedicated for DNN applications to achieve a higher data reuse. It can help alleviate the mismatch between computing throughput and limited memory bandwidth for DNN accelerators.

Ascend Core is a heterogeneous architecture that combines the scalar, vector, and cube computing units shown in Figure 2. We list typical operations that are supported by these computing units in Table 2. We provide more details and quantitative analysis based on the real implementation.

The typical dimension of the matrices in the 3D computing is $16 \times 16 \times 16$. Thus, the whole cube is equipped with 4096 multipliers and 4096 accumulators. Besides, each operand in matrix computing is reused for 16 times. Thus, the energy consumption for loading/feeding operands into the cube computing unit is reduced to 1/16, compared to that of a vector unit. Table 3 compares three types of computing units, in respect of performance per unit area and energy per unit area. These results are measured from real chips under the 7*nm* technology. It is easy to tell that a cube computing unit can improve both metrics by one order, compared with a vector computing unit. In the actual design, we flat the 3D cube's layout to 2D to arrange it on the silicon die.

Table 2: Operations for each unit.

Unit Name	Typical Operations
Scalar	Control, Scalar Computation
Vector	Normalize, Activation, Format Transfer, CV Operators (RPN, etc.)
Cube	Convolution, FC, MatMul

Table 3: Comparison among computing units

Unit Name	**Scalar**	**Vector**	**Cube**
Performance (FLOPS)	2G	256G	8T
Power (W)	/	0.46	3.13
Area (mm^2, 7nm)	0.04	0.70	2.57
Perf./Power (TFLOPS/W)	/	0.56	2.56
Perf./Area (TFLOPS/mm^2)	0.05	0.36	3.11

In Table 4, we demonstrate the impact of matrix dimension on performance of the cube computing unit. We compare a $16 \times 16 \times 16$ cube unit (typical size in Ascend core, as source format is fp16 with destination format fp32, proven to be feasible in CNN and NLP tasks [34]; can extend to $16 \times 32 \times 16$ with int8 precision) with a $4 \times 4 \times 4$'s 8 cores GPU Stream Multiprocessor design [1], both are measured under the 12*nm* technology. The computing throughput and area of the two designs are listed for comparison. The results show that, after increasing the dimension to 16, computing throughput increases by 4.7X, while the area is only increased by 2.5X. Thus, a higher dimension of cube unit is preferred in Ascend core. Nevertheless, if the cube gets larger, such as $32 \times 32 \times 32$, it becomes inefficient due to lower MAC utilization in several neural networks.

Table 4: Area/density benefits of the cube units.

Cube Dimension	$4 \times 4 \times 4$	$16 \times 16 \times 16$
Cube Quantity	8	1
Core Area (mm^2, 12nm)	5.2* [1]	13.2
FP16 Performance (FLOPS)	1.7 T	8 T
Perf./Area (GFLOPS/mm^2)	330	600

2.2 Datapath and Control Units

As shown in Figure 1, there are multiple levels of buffers. L0 buffers are dedicated to the cube computing unit. There are three individual L0 buffers, which are Buffer A L0, Buffer B L0, and Buffer C L0. They are used to hold input feature maps, weights, and output features, respectively.

*Because these details are not released for these designs, we try our best to provide the reasonable estimation.

Data in Buffer A L0 and Buffer B L0 are loaded from the L1 buffer. The communication between the L0 buffers and the L1 buffer is managed by the component called the memory transfer engine (MTE). There are several function modules in the MTE. The *decomp* module decompresses the data for sparse network, with the help of Zero-Value Compression like algorithms [22]. The *img2col* module is applied to transfer convolution to matrix multiplication. And the *trans* module transposes the matrix.

The output results in Buffer C L0 can be processed by the Vector Unit (e.g. normalization or activation). The output results from the Vector Unit are allocated in the Unified Buffer, which is shared with the Scalar Unit. Apart from the operations listed in Table 2, the Vector Unit is also responsible for data precision conversion, such as quantization and dequantization operations among int32, fp16, and int8. We can also apply the Vector Unit to help fp32 operations.

The bus interface unit (BIU) transfers data/instructions between Ascend core and external components. The data are stored in L1 Buffer. The fetched instructions are stored in an instruction cache. These instructions are first sequenced by the Program Sequence Queue (PSQ). Then, they are dispatched separately into three following queues, which are cube queue, vector queue, and MTE queue. Then, the instructions are processed by corresponding units, respectively.

Since three computing units and the MTE work in parallel, explicit synchronization is needed to enforce data dependency across different execution units. An example is shown in Figure 3. The scalar PSQ keeps dispatching instructions, which can be processed in parallel, to different units until an explicit synchronization signal (barrier) is encountered. The barrier is generated by the compiler or operator programmers.

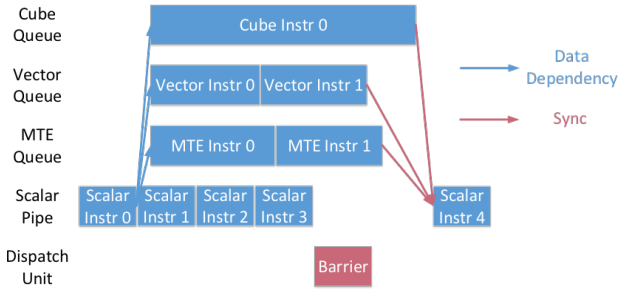

Figure 3: An example of synchronization.

2.3 Micro-architecture Exploration

Due to the rapid advancement of deep learning algorithms, modern DNN models become more and more diverse. It is impossible to design a fixed DNN accelerator, which works well for all scenarios listed in Table 1. The behavior and resource demands differ substantially from model to model and even layer to layer.

Therefore, we propose a series of versions of Ascend cores based on the same architecture, targeting different scenarios. We need to select proper configurations for different versions. The key is to obtain an optimal resource balance among different components of the core architecture. The configuration principle is listed as follows.

- Cube Unit: The size of the cube unit is first selected to meet specific computing throughput requirements.
- Vector Unit: We need to ensure that the vector unit is not the computing bottleneck. Thus, we need to profile the typical DNN models and compare the computation workloads between the cube unit and the vector unit. Then, the size of the vector unit is selected to match the throughput of the cube unit.
- L0 Buffer: The data bus between L0 buffer and the cube unit should provide enough data bandwidth. The size of L0 buffer should also be carefully designed to ensure proper data reuse.
- L1 Buffer: Similarly, the bus width should ensure that the L0/L1 bandwidth reduction ratio is not greater than the data reuse factor across all workloads.
- Unified buffer: The unified buffer plays multiple roles as cube-vector pipeline buffer, vector buffer, and output buffer. Thus, its bandwidth and size should be paid more attention to, especially in training scenario.
- L1 to SoC: The bandwidth are always bounded by Last Level Cache (LLC) bandwidth per core.

2.4 Resource Matching of the Vector Unit

As we have addressed, it is necessary to strike a balance between the width of the vector unit and the number of MAC elements within the cube unit. It depends on the resource requirement profiling on different DNN models. In the Ascend cores, we aim to maximize the utilization of the cube unit, the most costly execution unit. This objective can be realized by sizing the vector unit for the most demanding workloads. In this way, the time spent on vector instructions can be fully overshadowed by the execution time of the cube instructions.

Figure 4 shows the execution time ratio (cube/vector) across layers for inference phase of BERT model on Cube 8192 FLOPS/Cycle and Vector 256 B Width configuration. For most layers, the ratio is much greater than 1, indicating that the execution time of the vector can be hidden by that of the cube. Note that the Ascend-Max is used for both inference and training. In Figure 5, for a training workload on the same configuration, the computing on the vector unit is higher than that for the inference. Nevertheless, the ratio is still greater than 1 in most layers.

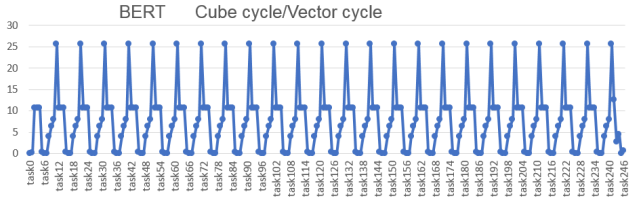

Figure 4: Execution time of cube/vector (BERT Inf.)

Figure 6 and Figure 7 show the ratio line charts for MobileNet and ResNet50 on Cube 8192 OPS/Cycle and Vector 256 B Width configuration. As shown in Figure 6, since most of the MobileNet (Ascend-Lite core's typical workload) layers' ratio are between 0 to 1, Ascend-Lite core needs a

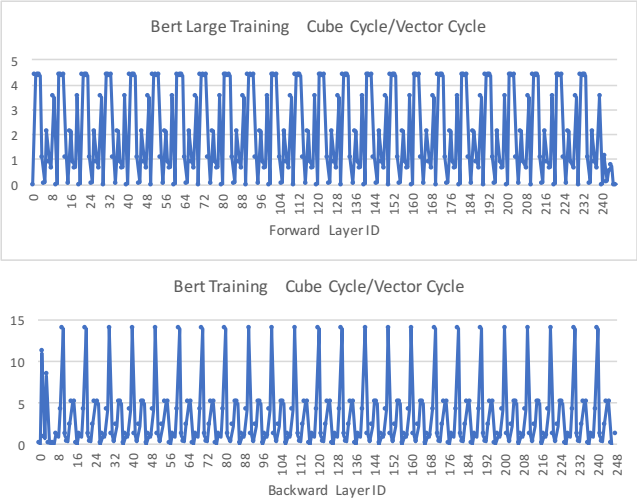

Figure 5: Execution time of cube/vector (BERT Tra.)

relative wider vector unit. So in Ascend-Lite core configuration, as cube unit been tailored from 8192 OPS/Cycle to 2048 OPS/Cycle, we shrink the vector width from 256 B to 128 B. As Figure 7 shows, in the first few layers of Resnet50 (Ascend core's and Ascend-Mini core's typical workload), the execution time ratio is close to 1, indicating that the time spent on vector instructions is almost the same as that spent on cube instructions. Thus, Ascend (-Mini) core's configuration is finally set to above settings. As a result, one could conclude that the BERT model is much more biased towards cube execution, when compared with MobileNet.

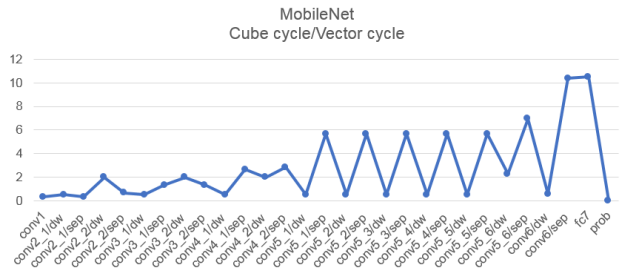

Figure 6: Execution time of cube/vector (MobileNet Inf.)

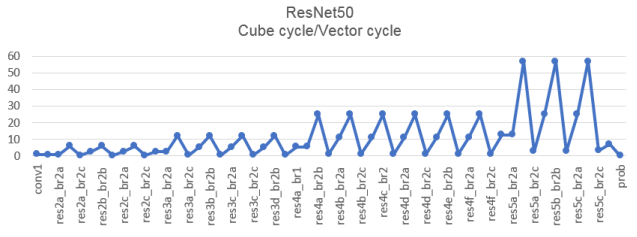

Figure 7: Execution time of cube/vector (ResNet50 Inf.)

Figure 8 shows the ratio line charts for Gesture Inference Neural Network on Cube 1024 OPS/Cycle (int8) and Vector 32B Width configuration. For all layers, the ratio is greater than 1, indicating Ascend-Tiny core's configuration can be set to above settings.

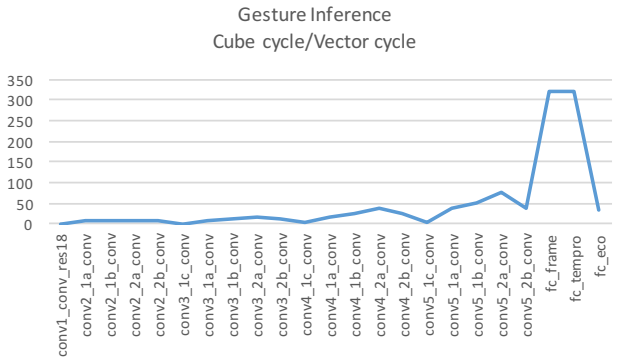

Figure 8: Execution time of cube/vector (Gesture Inf.)

2.5 Resource Matching for Bandwidth

Buffer memory bandwidth, corresponding to the bus width, is another critical resource that requires meticulous balancing. We did the profiling of L1 memory bandwidth demand, as detailed in Figure 9, on unlimited L1 memory bandwidth with Cube 8192 OPS/Cycle and Vector 256 B Width configuration, to further ensure that the pipeline is not blocked.

As Figure 9 shows, all layers' L1 memory reading bandwidth is not more than 4096 bits/cycle, with corresponding writing bandwidth less than 2048 bits/cycle. Furthermore, we compare these different networks, MobileNet (typical small network) shows more L1 memory bandwidth requirement.

With fine-grained design space exploration, Ascend carefully balances the memory hierarchy to prevent a bandwidth bottleneck from forming at key locations. Some examples are listed as follows:

- Reducing the DDR bandwidth requirement, by reusing data within L1, L0A, and L0B.
- Providing asymmetric bandwidth, based on the computation nature. For example, the bandwidth for data transmission from L1 to L0A is much higher than that for data transmission from L1 to L0B. The reason is that, in typical neural network models, the amount of data transmission from L1 to L0A is much larger than that of data transmission from L1 to L0B.

2.6 Designing Parameters

Table 5 shows key architecture parameters for several design points, including the performance of the cube unit, width of the vector unit, bus widths of the L0 and L1 buffers, and LLC bandwidth per core. These parameters were chosen based on profiling results across a wide range of DNN models. In the following subsections, we present the results of processing typical DNN models on different Ascend cores to show that the design principals are satisfied with configurations in Table 5.

3. ASCEND CORES IN VARIOUS SOCS

Ascend cores have been integrated into many SoCs used in different domains. They include DNN inference and training

Table 5: Key trade-off points of architecture designing parameter

Core Version	Cube Perf.	Vector Width	L1 Bandwidth	LLC Bandwidth/Core
Ascend-Max (1GHz) Ascend (1GHz) Ascend-Mini (1GHz)	8192 FLOPS/Cycle	256 B	A: 4 TB/s B: 2 TB/s UB: 2 TB/s	910: 94 GB/s 610: 111 GB/s 310: 96 GB/s
Ascend-Lite (0.75GHz)	2048 FLOPS/Cycle	128 B	A: 768 GB/s B: 768 GB/s UB: 768 GB/s	38.4 GB/s
Ascend-Tiny (0.75GHz)	1024 (int8) OPS/Cycle	32 B	A: 384 GB/s B: 384 GB/s UB: 192 GB/s	N/A

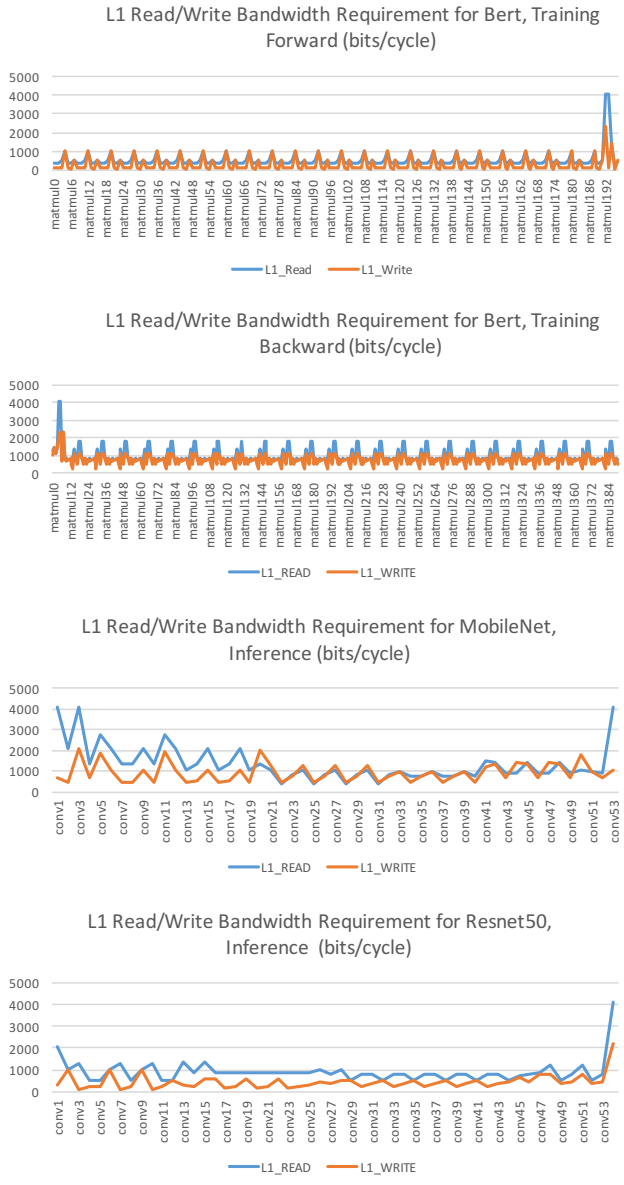

Figure 9: L1 bandwidth profiling of BERT forward and backward (Training), MobileNet and ResNet50 (Inf.)

devices in servers, mobile processors, autonomous driving systems, wireless base station processors, IoT devices, etc. In this section, we select three typical designs to explain how to tailor the unified Ascend architecture for different domains.

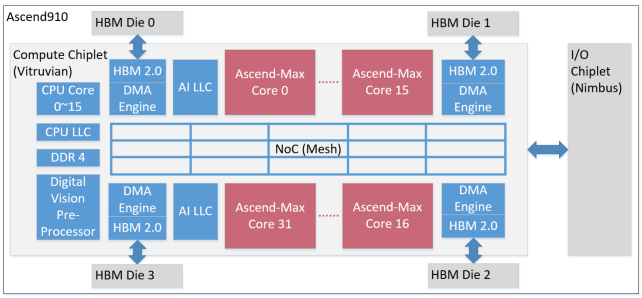

Figure 10: DNN Training SoC

3.1 SoC for DNN Training Devices

The DNN training card is normally a PCI-E interface device or Open Compute Project (OCP) socket used in a datacenter or servers. Since a large batch size is normally used in training, a $16 \times 16 \times 16$ cube unit works efficiently in most cases. However, during the backward SGD computing, the vector unit is used more frequently and more flexible. Thus, a duplex data path between unified buffer and vector is needed. Due to intensive data movement, the accesses to LLC and external memory become more frequent. This is ensured by high Load/Store bandwidth. A typical SoC design (Ascend 910) is illustrated in Figure 10.

The DNN Training SoC consists of 6 dies: a compute chiplet, an I/O chiplet, and four high bandwidth memory (HBM) dies with 1.2TB/s bandwidth in total. The compute chiplet contains 32 Ascend-Max cores with AI LLC, 16 CPU cores (Arm V8 ISA's CPU, custom made for Ascend SoC) with CPU LLC, Digital Vision Pre-Processor (decoder, etc.), and a Network-on-Chip (NoC) connects the components above-mentioned.

3.1.1 Communication Network Design

A Mesh NoC architecture is used in the SoC design to provide a unified and scalable communication network. The NoC is based on a 4×6 2D Mesh topology. The link between two adjacent nodes works at 2GHz with a 1024-bit connection width. Thus, it can achieve a high bandwidth of $256GB/S$. The bufferless architecture is used to reduce the area overhead of NoC. The components and I/O interfaces are connected on

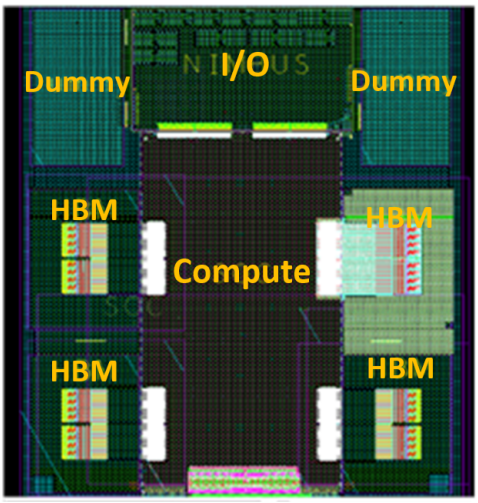

Figure 11: Package layout of Ascend 910

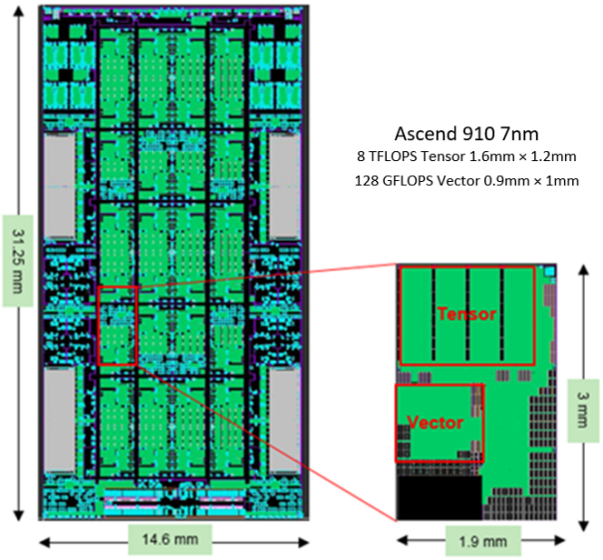

Figure 12: Ascend 910 computer chiplet's floor plan

the NoC in a symmetric style. A global scheduling policy is used to ensure communication QoS.

3.1.2 Physical Design of Ascend AI Chips

Ascend 910 delivers 256 Tera FLOPS in fp16 and 512 Tera OPS in int8. It has a 128-channel full HD video decoder, so that video data can be decoded and processed on-chip. Ascend 910 is powered by the 7+ nm EUV process, with a rated TDP of 300 W. Figure 11 shows the package layout of Ascend 910, with the compute chiplet (Vitruvian), I/O chiplet (Nimbus) and four HBM stacks integrated.

As shown in Figure 12, 32 Ascend cores are connected through an NoC. It has a mesh topology, with 6 rows and 4 columns. The total throughput to L2 memory, and to the off-chip HBM subsystem, are 4 TB/s and 1.2 TB/s, respectively.

3.2 SoC for Mobile Processors

Recently, various DNN applications have been enabled on mobile phones, which include scenario detection for picturing, photo retouching, biometric identification, etc. These applications are normally based on NN models, such as MobileNet [31] and Resnet50 [30]. The Ascend-Lite core is integrated in the SoC to support these applications.

For battery powered mobile devices, the power consumption is a critical concern. The energy efficiency of an Ascend-Lite core can achieve 4.6TOPS/W, when running in the standard mode. In addition, the dynamic voltage and frequency scaling (DVFS) mechanism is enabled in it to further improve energy efficiency. The working voltage can change dynamically according to real-time workload intensity.

Another design concern is about the task granularity on the core. For most applications on the mobile devices, the batch size of inference is only one. Thus, we prefer using a smaller cube computing unit in the Ascend-Lite core. Thus, we reduce the dimension of the cube size to $4 \times 16 \times 16$, to perform m $\times$ k $\times$ n GEMM. Taking image to column action into consideration, m = batch size $\times$ height of output $\times$ width of output. As a result, when batch size turns to 1, the smaller m dimension improves cube's MAC utilization.

In addition, mobile's wake-up function and real-time online human-computer interaction rely on AI algorithms, such like face recognition and gesture inference. To optimize the power consumption in this scenario, Ascend-Tiny core is introduced. Its cube size is $4 \times 32 \times 4$, whose source are tailored to int8 (fp16 forbidden due to power limitation, thus its typical power consumption is as low as 300mW). Ascend-Lite core and Ascend-Tiny core form the AI's Big-Little relationship in our Mobile SoC.

An example mobile SoC design (Kirin 990 5G) is illustrated in Figure 13. Two Ascend-Lite cores and one Ascend-Tiny core are connected to other components using a NoC. The instruction compression technique is used in the Ascend-Lite core to reduce the bandwidth pressure on the NoC. The core is optimized for structured sparsity in DNN models. Thus, the computing power consumption can be further reduced under (general) sparsity.

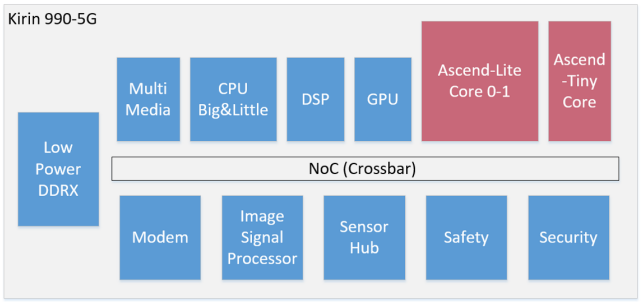

Figure 13: Mobile Processor SoC

3.3 SoC for Autonomous Driving System

The target of this SoC is an autonomous driving system that supports Level-3 and Level-4 automation with safety standard ASIL-D [9] [2]. Ascend core is used in this domain with the following dedicated optimizations.

First, the current trend in this field is to make a comprehensive decision based on outputs from multiple DNN models [18], thus the precision of inference computing for each DNN model can be reduced as a trade-off between model accuracy and calculating time / energy consumption. This indicates that low precision inference is favored. To this end,

the Ascend core supports int4 precision.

Second, autonomous driving system is a real-time design, which has a critical demand on response time. It ensures that the autonomous driving service can complete all tasks in time, including sensing, cognition, decision-making, and execution. The inference performance not only relies on highly paralleled computing and data movement inside the core, but also depends on the efficiency of Load/Store operations to external memory hierarchy. It means that the SoC should guarantee acceptable external memory's accessing latency. Thus, a series of priority mechanisms, including Memory System Resource Partitioning and Monitoring (MPAM) and Quality of Service (QoS), are adopted to both Ascend core and the SoC. QoS is mainly used to avoid starvation. MPAM manages cache capacity, NoC bandwidth, and memory bandwidth more fine-grained. These strategies can satisfy the latency requirement of the whole service.

Third, besides computing intensive DNN workloads, the system also needs to support typical SLAM tasks, such as localization and map construction. Consequently, the Vector Core (Ascend core without cube) and micro-architecture extensions, such as sorting, stereo vision, general matrix calculation (quaternion), structure data processing, clustering and linear programming specified instructions, are applied to enable efficient and flexible processing of these tasks on the vector computing unit.

Last but not the least, Ascend core supports safety standard ASIL-B, to further confirm the credible perceptual result.

An SoC architecture for autonomous driving system is illustrated in Figure 14. Besides the Ascend AI core, a Digital Vision Pre Processor (DVPP) , a dedicated ASIC accelerator, is also integrated to support image pre-processing tasks, such like resize and 360° surrounding stitch. With these components, an end-to-end acceleration for the whole service is achieved. Meanwhile, to further confirm CPU's real-time, we made a separated Ring NoC with safety standard ASIL-D.

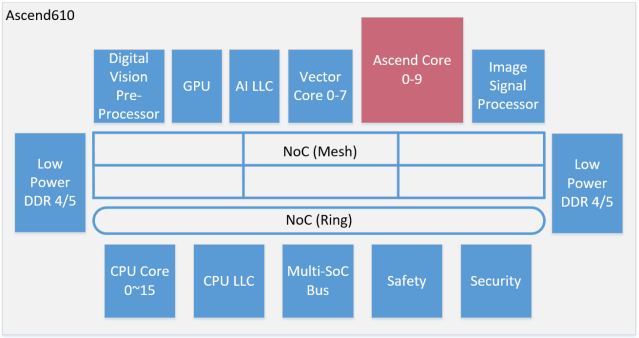

Figure 14: Autonomous Driving SoC

4. DESIGN ISSUES FOR SCALING

4.1 Memory Hierarchy Scaling

One major challenge encountered in developing Ascend chipset is the memory wall. As shown in Table 6, the core of compute units of an Ascend 910 device requires a memory bandwidth of 2048 TB/s. However, even when the most advanced HBM technology is used, this only provides a bandwidth of slightly over 1 TB/s. The required bandwidth is over 2000 times that supported by the memory subsystem. To overcome such a wide gap, we adopted a multi-layer memory hierarchy, in which L0 is used to provide the required bandwidth of the cube unit. Moving down the hierarchy, we attempted to reduce the memory bandwidth by 10 times in each lower layer, relying heavily on the prerequisite that data are reused in each layer of the memory hierarchy.

Table 6: Memory wall and I/O wall

	Bandwidth Expected	Bandwidth Ratio
Cube Engine (256TFLOPS)	2048 TB/s	1
L0 Memory	2048 TB/s	1/1
L1 Memory	200 TB/s	1/10
LLC Memory	20 TB/s	1/100
HBM Memory	1 TB/s	1/2000
Intra AI Server (8 Chips)	50 GB/s	1/40000
Inter AI Server	10 GB/s	1/200000

It is our hope that three-dimensional integrated circuit (3D-IC) technology can address challenges related to the memory wall. This can result in a substantial increase in the total amount of memory bandwidth, and reduce energy costs for memory accesses, with the binding of memory and logic. The next generation of Ascend training device utilizes the 3D-SRAM technique from TSMC [23, 36] to increase the total size of on-chip LLC. Recently, we made a coarse-grained comparison, as the capacity of LLC increases from 96MB to 720MB, the Resnet50 performance is improved by 1.71 times, and the Bert performance is improved by 1.51 times.

4.2 Ascend 910 Server and Cluster

Figure 15 (a) demonstrates Ascend 910 server and cluster's logic connection. The lower half of Figure 15 (a) shows Ascend 910 servers' design. Each Ascend 910 server contains eight Ascend 910 chips, which are organized as two groups on a board. The intra-group connection is based on a high-speed cache coherence network (HCCS) [31]. Two groups communicate with each other using a PCI-E bus. The bandwidths of intra-group connection and inter-group connection are 30GB/S and 32GB/S, respectively. Figure 15 (b) is a product form picture of Ascend 910 server.

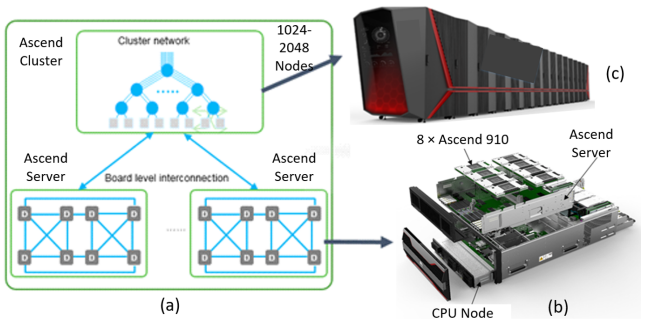

Figure 15: Ascend 910 server and cluster

To further scale out, we can organize multiple Ascend 910 servers as a cluster. The higher half of Figure 15 (a) demonstrates a 2048-node cluster, with a total computing power of 512 Peta FLOPS in fp16. It contains 256 servers,

which are connected with a Fat-Tree topology network. The link bandwidth between two servers is 100Gbps. Figure 15 (c) is a product form picture of Ascend 910 cluster cabinet.

5. SOFTWARE SUPPORT FOR ASCEND

5.1 Software Development Stack

The development stack of the Ascend core leverages a native differentiable programming paradigm and the native execution modes, which provide trade-off between development efficiency and processing efficiency. On the one hand, the users can deploy DNN models easily to various devices, which may equip different Ascend cores. They can achieve the one-time development and on-demand deployment with the help of high level programming model. On the other hand, the users, who have more architecture knowledge, can leverage the low level programming models and libraries to further improve the processing efficiency of applications. The development stack of Ascend is shown in Figure 16. We also include the counterparts in the development stack of GPU for comparison.

The DNN model development frameworks, such as PyTorch, TensorFlow, and MindSpore, are on the top of the whole stack. MindSpore is a dedicated framework for Ascend core. It can achieve better resource efficiency and make full use of the computing power of Ascend cores. As we have mentioned, the deployment of most "Apps" with DNN models also relies on these frameworks. The output from these frameworks is called "Graph", which presents the coarse grained relationship in algorithms. Then, with the help of Graph Engine, "Graph" is transferred to so called "Streams", which consist of several in-order "Tasks". The "Streams"/"Tasks" can be directly called from Operator Lib, or described by programmers with different level languages, with the help of Operator Engine.

TBE (Tensor Boost Engine) DSL (Domain Specific Language) is developed with the Level-3 programming model so that users with no hardware knowledge could write code at this level, it is also called the mathematical programming level. With the help of compiler, the instance "Tasks" can be generated automatically from the TBE DSL description.

Programmers can also develop the instance "Tasks" in the parallel/kernel level (Level 2) programming model. This level is similar to the CUDA or OpenCL for a GPU. It introduces the so-called TIK (Tensor Iterator Kernel) interfaces for parallel programming with Python. The dedicated compiler technique, called "Auto Tiling", is used to transfer big tasks into small fractals to adapt to Ascend architecture. With the help of reinforcement learning algorithms, this technology offers the best tiling and scheduling for any program by intelligently searching legitimate mapping space.

As shown in the bottom of Figure 16, the lowest level (Level 1) of programming model is the C programming interface, also known as CCE-C (Cube-based Compute Engine). In this level, all design details for each architecture are exposed to programmers. Programmers can embed the assembly-like code in the C program to leverage hardware details and trivial branch/loop descriptions. Thus, it is also called the architecture defined programming. The corresponding high-performance libraries are written by experts, who have the knowledge about computing architectures.

5.2 A Multi-level Scheduling Hierarchy

As described in last subsection, an application (APP) is transformed to and organized with different representations, i.e. streams, tasks, and blocks, in different levels of programming models. All these representations can be executed in parallel with corresponding schedulers in different levels, as shown in Figure 17. The details are introduced as follows,

- Application level: multiple apps are able to be executed in parallel on one Ascend SoC.
- Stream & Task level: graph compiler compiles each app into several streams, with several tasks in each stream. These tasks will be executed in order with the help of Ascend SoC's task scheduler.
- Block level: each task will be separated into several blocks (explicit descripted by programmers), which can be executed parallel on different Ascend cores.

Recall that each Ascend core has multiple parallel execution pipelines, which are asynchronously coupled with explicit barrier. This is shown in Figure 3. The hardware multi-threads are introduced for programmers/compiler to express programming parallelism. Thus, it is also called thread level parallel.

6. EVALUATION RESULTS

In this section, we present comparisons between different Ascend cores and state-of-the-art commercial designs in three typical AI computing scenarios. We evaluate all these designs in performance, power/energy-efficiency and area (PPA). The experimental results validate the efficiency, scalability, and flexibility of Ascend's unified architecture. All the Ascend's performance results are based on measurement.

6.1 NN Training SoC

We compare Ascend 910 with other state-of-the-art chips in NN training, and the results are illustrated in Table 7. It is obvious that Ascend 910 achieves higher performance than NVidia V100 and Google TPU. Besides, Ascend 910 outperforms Intel CPU by several orders of magnitude. Nvidia V100 combines Tensor Core and inherited GPGPU SIMT architecture to support cube-based processing on neural networks. However, the opportunities of data reuse are limited by inherent schemes and the small size of Tensor cores. Google TPU V3 plans to employ systolic array to process NN training. However, the systolic array architecture may suffer from the under-utilization of computing resources during filling pipelines and draining pipelines. Furthermore, in the NN training scenario, systolic array's pipeline is easily to be interrupt by Normalization layer. It is worth noticing that, compared with the specialized architecture designs for specific scenarios, Ascend's unified architecture design can still achieve a similar or even higher performance.

*Because these details are not released for these designs, we try our best to provide the reasonable estimation.

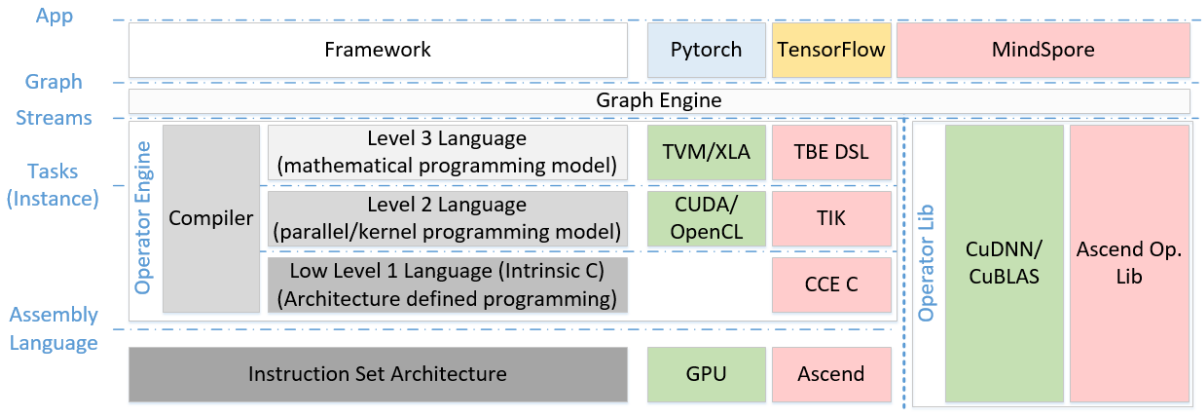

Figure 16: Illustration of Software Development Stack

Table 7: Training SoC's PPA

	NVidia V100 [1]	Google TPU V3 [5]	Intel CPU 8180 (28C, 2.5GHz) [8]	Ascend 910
Peak Performance (TFLOPS)	125	106	1.5	256
Power (W)	300	250*	205	300
Area (mm^2)	815	–	700*	456 (Comp. die) + 168 (IO die)
Process (nm)	12	16	14	7 (Comp. die) + 16 (IO die)
DRAM/HBM Bandwidth (GB/s)	900	1200	128	1200
ResNet50 V1.5 (images/s) [13]	1058	976 [3]*	-	1809
BertLarge 8p (sequences/s) [13]	822	-	-	3169

Table 8: Mobile AI Core's PPA [32] [15]

	SnapDragon 865 [17]	MTK Dimensity 1000 [10]	Exynos 9820 [32]	Apple A13 [4]	Kirin 990-5G
Peak Performance (TOPS)	8	4.5	2.1(dense)-6.9	6*	6.88
Power Efficiency (TOPS/W)	–	3.42(dense)-6.83	3.6(dense)-11.5	–	4.6
Area (mm^2)	2.4*	2.68*	5.5	2.61*	4
Process (nm)	7	7	8	7	7
AI-Score	29724	56158	23487	-	70185
MobileNet V2 (seconds per image, fp16)	15	7	15	-	5.2

Table 9: Automotive SoC's PPA

	NVidia Xavier [14]	Tesla FSD [7]	Mobileye EyeQ5 [12]	Ascend 610
Peak Performance (TOPS)	34	73	24	160
Power (W)	30	100	10	65
Area (mm^2)	350	260	–	401
Process (nm)	12	14	7	7

Table 10: Ascend Series' Cores Business Numbers

	Ascend 910	Mobile SoC with Ascend Cores	Ascend 610	Ascend 310
Release Time (Year)	2019	2019	2020	2018
Quantity (Million)	≈ 0.2	>100	/	≈ 1

6.2 Mobile AI Core

In Table 8, we show the performance comparison between Kirin990-5G and state-of-the-art AI cores in mobile SoCs. Though Kirin990-5G was released earlier than the other chips, it still achieves great performance. AI-benchmark [15] presents Kirin990-5G's excellent performance (AI-Score in Table 8) in practical applications. The mobile SoCs proposed in other work usually employ Digital Signal Processors (DSPs) to process the vector-based computation involved in AI computing. Different from these chips, Kirin990-5G employs Ascend series cores to process all kinds of computing tasks involved in AI computing. More importantly, the Ascend cores induce minor area overheads, compared with the DSPs in other chips.

6.3 Automotive SoC

We show the performance results of several Automotive SoCs and Ascend 610 in Table 9. Automotive SoCs' applications are not standard. It means there is no specific AI benchmark for this scenario. Thus, we only show the peak performance in Table 9.

We speculate Tesla FSD employs systolic arrays to process neural networks, but it suffers from the massive bubbles in pipeline during processing small-scale neural networks. Due to its low performance, Mobileye EyeQ5 has to exploit multi-

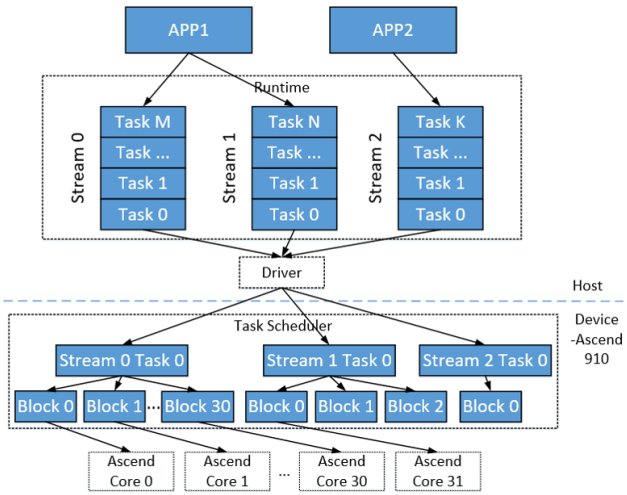

Figure 17: Schedule hierarchy

chip architecture to address L4 automotive driving scenarios, which may result in more constraints in its deployment. Compared with these chips, Ascend 610 shows better flexibility and higher energy-efficiency. Ascend 610 performs specialized optimizations in flexibility and meets requirements of different scenarios well.

7. RELATED WORK AND FUTURE WORK DISCUSSION

7.1 Related Work Discussion

There are several types of industrial AI acceleration solutions, including General Proposed GPU (GPGPU), Systolic Array based approaches, Dataflow based approaches, etc., which have different characteristics.

GPGPUs usually leverage a Single Instruction Multi Thread (SIMT) mechanism, which has an advantage on branch divergence. As a load/store architecture, SIMT is able to conceal the memory loading time by thread Warp (in terms of NVidia terminology [1]) dynamic switching. Nevertheless, DNN computation usually does not benefit from SIMT. Moreover, SIMT pays additional overhead, such as register array, which adversely impacts the performance density (TFLOPS/mm^2). In addition, in order to be compatible with the existing SIMT mechanism (such like warp, and the correspondence between register and thread), it is difficult to employ a large size of Cube (called Tensor Core in NVidia).

Systolic Array is widely adopted by commercial chips like TPU [33] and Tesla FSD (speculative) [7] for its highly efficient matrix multiplication. However, the deep pipeline of Systolic Array incurs large prologue& epilogue latency overhead when running small networks, causing low computing utilization in mobile and IoT scenarios.

Dataflow architecture is another typical paradigm for accelerator design, and can achieve high throughput for specific tasks. However, dataflow architectures are incapable of performing main stream synchronous training. Moreover, in automobile, mobile and IoT scenarios, dataflow architecture can incur low computing utilization and large output delay.

Apart from the aforementioned commercial solutions for DNN computing, many other specialized architectures have also been proposed for typical deep learnig models in academia. Han et al. [29] proposes an accelerator for LSTM-NNs, which are widely used in natural language processing. Yamada et al. [40] integrate DNN accelerators and image signal processors to process autonomous driving with high performance and low power. To accelerate deep learning computing involved in autonomous mobile robots, Amaravati et al. [21] implement a time-domain mixed-signal neuromorphic accelerator. Niu et al. [35] perform specialized optimizations to accelerate DNN processing on smartphones, and Whatmough et al. [38] exploit data sparsity and error tolerance techniques to reduce the energy consumption of deep learning accelerators on intelligent IoTs.

7.2 Future Work Discussion

We predict future HPC algorithms will incorporate more deep learning methods, therefore it is natural for next generation Ascend to take on more HPC roles. We would like to apply fp32 in the cube unit to adapt to some corner applications. And the Ascend customized special layout may limit users adding customized operators, which can be updated in next version.

Memory wall is one of the most challenging issues. Thus, we keep focus on using chiplets technology. And using advanced on-chip memory technology is also helpful. We are also considering spatial computing technology and introducing new methods of interconnection, including data flow optimization and more advanced QoS management.

With the improvement of the process and engineering technologies, we feel that using optical connection directly on the SoC may also improve scalability. And novel materials may significantly increase the memory space of a single SoC.

8. CONCLUSION

In this work, we propose an acceleration architecture called Ascend. Acceleration cores based on this unified architecture have been applied to a huge range of applications to achieve efficient processing of various DNNs. The success of Ascend shows that scaling a unified architecture to different levels of scenarios relies on several critical design efforts. Besides using heterogeneous computing units and design exploration inside the core, a holistic solution is required to overcome the memory wall problems residing inter-core, inter-die, and inter-chip. In addition, a flexible multi-level software stack is critical to provide a unified development interface in different scales. As a summary, Ascend architecture is very competitive in terms of performance, power-efficiency, and area in these scenarios. A series of Ascend core based SoCs have been successfully commercialized.

The current business results are listed in Table 10. We have shipped over hundreds of millions units.

We proudly announced Ascend 910 cluster with our full-stack optimized software, which delivered the fastest deep learning cluster performance available in cloud, trained ResNet-50 on ImageNet2012 in less than 83 seconds, using only 256 Ascend 910 chips. In the more hardware sensitive "Closed division times" MLPerf evaluation, the Ascend cluster also ranks top1 on ImageNet Training. [11]

REFERENCES

[1] "Nvidia tesla v100 gpu architecture white paper," https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf, 2017.

[2] "Iso 26262-11," https://www.iso.org/obp/ui/iso:std:iso:26262:-11:ed-1:v1:en, 2018.

[3] "Mlperf training v0.6 results," https://mlperf.org/training-results-0-6, 2019.

[4] "Apple a13," https://en.wikipedia.org/wiki/Apple_A13, 2020.

[5] "Cloud tpu," https://cloud.google.com/tpu, 2020.

[6] "Cudnn," https://developer.nvidia.com/cudnn, 2020.

[7] "Fsd chip - tesla," https://en.wikichip.org/wiki/tesla_(car_company)/fsd_chip, 2020.

[8] "Intel xeon platinum 8180 processor," https://www.intel.com/content/www/us/en/products/processors/xeon/scalable/platinum-processors/platinum-8180.html, 2020.

[9] "Iso 26262," https://en.wikipedia.org/wiki/ISO_26262, 2020.

[10] "Mediatek dimensity 1000," https://www.mediatek.com/products/smartphones/dimensity-1000, 2020.

[11] "Mlperf training v0.7 results," https://mlperf.org/training-results-0-7, 2020.

[12] "Mobileye eyeq," https://www.mobileye.com/our-technology/evolution-eyeq-chip/, 2020.

[13] "Nvidia deep learning performance," https://developer.nvidia.com/deep-learning-performance-training-inference, 2020.

[14] "Nvidia xavier," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/, 2020.

[15] "Performance ranking," http://ai-benchmark.com/ranking_processors.html, 2020.

[16] "Pytorch," https://pytorch.org/, 2020.

[17] "Snapdragon 865," https://www.qualcomm.com/products/snapdragon-865-5g-mobile-platform, 2020.

[18] "Visit mobileye at ces 2020!" https://www.mobileye.com/live/ces-2020/, 2020.

[19] "Xla," https://www.tensorflow.org/xla, 2020.

[20] 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 *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[21] A. Amaravati, S. B. Nasir, J. Ting, I. Yoon, and A. Raychowdhury, "A 55-nm, 1.0–0.4 v, 1.25-pj/mac time-domain mixed-signal neuromorphic accelerator with stochastic synapses for reinforcement learning in autonomous mobile robots," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 75–87, 2018.

[22] L. Cavigelli and L. Benini, "Extended bit-plane compression for convolutional neural network accelerators," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2019, pp. 279–283.

[23] M.-F. Chang, C.-S. Lin, W.-C. Wu, M.-P. Chen, Y.-H. Chen, Z.-H. Lin, S.-S. Sheu, T.-K. Ku, C.-H. Lin, and H. Yamauchi, "A high layer scalability tsv-based 3d-sram with semi-master-slave structure and self-timed differential-tsv for high-performance universal-memory-capacity-platforms," *IEEE journal of solid-state circuits*, vol. 48, no. 6, pp. 1521–1529, 2013.

[24] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2722–2730.

[25] N. Churamani, M. Kerzel, E. Strahl, P. Barros, and S. Wermter, "Teaching emotion expressions to a human companion robot using deep neural architectures," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 627–634.

[26] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 160–167.

[27] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3389–3396.

[28] H. HaddadPajouh, A. Dehghantanha, R. Khayami, and K.-K. R. Choo, "A deep recurrent neural network based approach for internet of things malware threat hunting," *Future Generation Computer Systems*, vol. 85, pp. 88–96, 2018.

[29] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang *et al.*, "Ese: Efficient speech recognition engine with sparse lstm on fpga," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 75–84.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[31] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[32] A. Ignatov, R. Timofte, A. Kulik, S. Yang, K. Wang, F. Baum, M. Wu, L. Xu, and L. Van Gool, "Ai benchmark: All about deep learning on smartphones in 2019," *arXiv preprint arXiv:1910.06663*, 2019.

[33] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, V. T. 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 *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*. IEEE, 2017, pp. 1–12.

[34] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, "Mixed precision training," *arXiv preprint arXiv:1710.03740*, 2017.

[35] W. Niu, X. Ma, Y. Wang, and B. Ren, "26ms inference time for resnet-50: Towards real-time execution of all dnns on smartphone," *arXiv preprint arXiv:1905.00571*, 2019.

[36] S. Srinivasa, A. K. Ramanathan, X. Li, W.-H. Chen, F.-K. Hsueh, C.-C. Yang, C.-H. Shen, J.-M. Shieh, S. Gupta, M.-F. M. Chang *et al.*, "A monolithic-3d sram design with enhanced robustness and in-memory computation support," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2018, pp. 1–6.

[37] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[38] P. N. Whatmough, S. K. Lee, H. Lee, S. Rama, D. Brooks, and G.-Y. Wei, "14.3 a 28nm soc with a 1.2 ghz 568nj/prediction sparse deep-neural-network engine with> 0.1 timing error rate tolerance for iot applications," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2017, pp. 242–243.

[39] B. Wu, F. Iandola, P. H. Jin, and K. Keutzer, "Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 129–137.

[40] Y. Yamada, T. Sano, Y. Tanabe, Y. Ishigaki, S. Hosoda, F. Hyuga, A. Moriya, R. Hada, A. Masuda, M. Uchiyama *et al.*, "A 20.5 tops multicore soc with dnn accelerator and image signal processor for automotive applications," *IEEE Journal of Solid-State Circuits*, 2019.

[41] S. Yao, Y. Zhao, A. Zhang, S. Hu, H. Shao, C. Zhang, L. Su, and T. Abdelzaher, "Deep learning for the internet of things," *Computer*, vol. 51, no. 5, pp. 32–41, 2018.

[42] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.