

# GTA: a new General Tensor Accelerator with Better Area Efficiency and Data Reuse

Chenyang Ai  
Peking University  
Beijing, China  
chenyang\_ai@stu.pku.edu.cn

Lechuan Zhao  
Peking University  
Beijing, China  
lczhao@stu.pku.edu.cn

Zhijie Huang  
Peking University  
Beijing, China  
zhijiehuang@stu.pku.edu.cn

Cangyuan Li  
Chinese Academy of Sciences  
Beijing, China  
licangyuan20g@ict.ac.cn

Xinan Wang  
Peking University  
Beijing, China  
anxinwang@pku.edu.cn

Ying Wang  
Chinese Academy of Sciences  
Beijing, China  
wangying2009@ict.ac.cn

## ABSTRACT

Recently, tensor algebra have witnessed significant applications across various domains. Each operator in tensor algebra features different computational workload and precision. However, current general accelerators, such as VPU, GPGPU, and CGRA, support tensor operators with low energy and area efficiency. This paper conducts an in-depth exploration of general accelerator for tensor processing.

First, we find the similarity between matrix multiplication and precision multiplication, and create a method classifying tensor operators using GEMM and vector operations. Then, we implement two discoveries and introduce the systolic architecture into general-purpose accelerator. Therefore, we propose a new General Tensor Accelerator (GTA), which has a better area efficiency and data reuse. Furthermore, we create a general tensor scheduling optimization strategies based on dataflow, precision and array resize. Our evaluation results demonstrate that GTA is able to achieves  $7.76\times$ ,  $5.35\times$ ,  $8.76\times$  memory efficiency and  $6.45\times$ ,  $3.39\times$ ,  $25.83\times$  speedup over of VPU, GPGPU and CGRA.

## KEYWORDS

Tensor Operator, General Accelerator, Systolic Architecture, Matrix Multiplication, Scheduling Space

## 1 INTRODUCTION

In recent years, computing kernels in various domains, such as machine learning, data analysis, signal processing, and scientific computing, have been categorized as operators in tensor algebra (like GEMM, CONV, GEMV, MTTKRP, TTMc etc.). These tensor operators exhibit different computational workload, including inherent variation in tensor dimension, computation, and accumulation. [16, 25]. Furthermore, these operators involve different precision. For instance, applications like audio and image signal processing often demand filter operators with INT8 or INT16 precision [34, 37]. Scientific computing, encryption algorithms, and other zero-error algorithms necessitate NTT(Number Theoretic Transform) with INT32 or even INT64 precision [1, 21, 40]. Even in one domain, there can be diverse precision requirements. In machine learning, for example, quantization inference (INT8) [14], training weights (FP16, BP16) [12], and high-precision AI networks (FP32, FP64) [2] encompass a wide spectrum of precision.

Previous works focus on dedicated ASIC accelerators for specific tensor operator [6, 7, 16]. But, the general purpose hardware still have many application scenarios, and they are typically employed to handle various tensor applications. With overwhelming computation, CPU can't bear the burden of tensor operators. Existing general accelerators adopted by academia and industry refer to Vector Processing Unit (VPU) and Cuda Core in GPGPU.

However, these vector architectures support the computational workload of operators with low energy efficiency [3, 28]. One of the most important reasons is that vectorization is indeed the most general method for tensor operators, but the computing unit cannot exploit data reuse in tensor operators, resulting in a lot of access to storage.

Besides, the emergence of various precision of tensor operator requires hardware support. For example, contemporary vector instruction sets need to implement operations for eight kinds of precision, including INT8, INT16, INT32, INT64, BP16, FP16, FP32, and FP64 [22, 30, 36]. These precision units are often established independently and occupy the majority of the overall area [4]. But, the specific workload always only utilizes single precision unit of all, which leads to a low area efficiency.

To summarize, current general accelerators suffer from low area and energy efficiency simultaneously. Actually, it is difficult to achieve both generality and efficiency in hardware design. To our knowledge, previous works only improves the energy efficiency of GEMM [10, 18, 27] for general accelerators. There are also solutions to introduce CGRA [25] into general tensor processing. However, their energy efficiency and generality can be further improved, and they overlook opportunity to co-optimize with computational precision.

This work takes an ingenious perspective, dissecting the computational workload and precision from the perspective of architecture and computing paradigms. Based on traditional VPU, we proposes a new General Tensor Accelerator (GTA) aiming for processing tensor applications more efficiently. In summary, this paper makes the following contributions:

- We find the similarity between matrix multiplication and precision multiplication, and create a classification of tensor operators. Then we implement two discoveries on systolic array for better area and memory efficiency.
- We design a Multi-Precision Reconfigurable Array (MPRA) and implement MPRA in vector architecture to compose

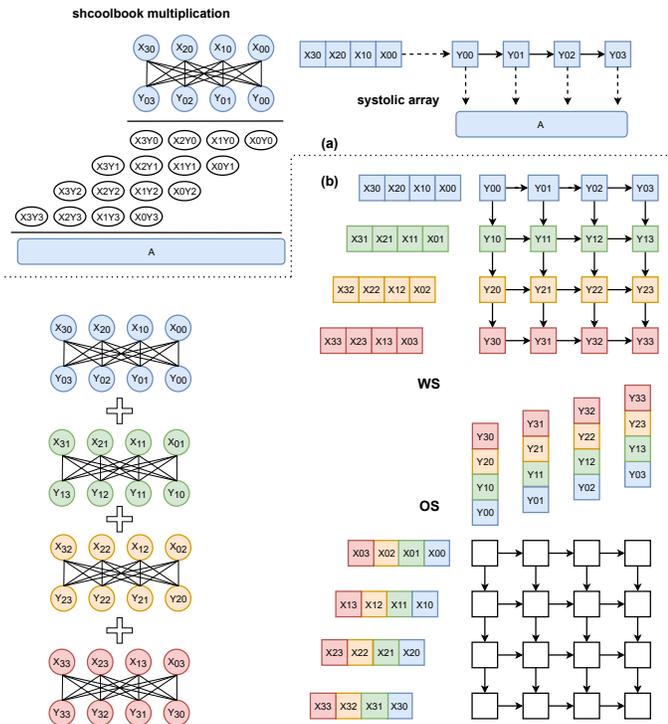
GTA, which can compute the tensor operators with arbitrary computational workload and precision.

- We implement general tensor scheduling optimization strategies based on dataflow, precision and array resize and make an analysis of scheduling space.

According to the evaluation, GTA is able to achieve  $7.76\times$ ,  $5.35\times$ ,  $8.76\times$  memory efficiency and  $6.45\times$ ,  $3.39\times$ ,  $25.83\times$  speedup over VPU (Ara), GPGPU (NVIDIA H100) and CGRA (hycube).

## 2 BACKGROUND

In this section, We provide here an introduction to the specialized accelerators about tensor operators compared with general accelerators. The accelerators are mainly spatial architecture, and systolic array is a typical example. Then we provide a brief overview of previous work about reconfigurable architecture and reveal the distance to a general tensor accelerator.



**Figure 1: The diagram of multi-precision matrix multiplication implement on the systolic array.**

### 2.1 Specialized Accelerators and Systolic Array

To tackle the problem of tensor algebra acceleration, various spatial architectures are adopted such as Tensor Core [7], eyeriss [6], etc. These spatial accelerators are often composed of PE arrays and their interconnections, which can exploit different types of data reuse. Usually, the majority of spatial architectures is naturally suitable for single kind of tensor operator with specific computational workload because of their microarchitecture. For example, the systolic array is

well suited to accelerating GEMM, while eyeriss is for CONV. In most cases, they are only designed for one precision.

Among them, the systolic array has emerged as a great choice due to its high power efficiency and peak throughput. Adopting simple PE unlike that with large memory in other dataflow architectures [6] and implementing simple interconnection instead of complicated and poor timing one in Tensor Core [7] makes it easy to scale the array, which achieves a higher degree of data reuse [31, 39]. Furthermore, there are several works on improving the utilization, which is the primary drawback of the systolic array. RASA [15] utilizes systolic array as an additional pipeline matrix unit of CPU, also overlooking the potential for reusing the original components. Mirroring [23] and Redas [11] work on flexible multi-workloads to improve utilization. All the above works have not jointly optimized dataflow and precision. Meanwhile, previous efforts have integrated systolic array with CPU [18], VPU [27], GPU [10]. But they cannot fully exploit original control and communication hardware resources, which could have been used to improve the utilization of systolic array. They also ignore the precision unit of general accelerators.

### 2.2 Specialized Accelerators Become More and More General

Recently, while general-purpose accelerators are enhancing their performance for GEMM in neural networks [10, 18, 27], specialized accelerators are trying to accept more operators in robustness. But we hold the view that previous works only explore reconfigurable architecture and functional versatility, which involves limited range of computational workload and precision.

For example, there are a few studies exploring the functional versatility of TPU [13, 24]. These works lack hardware modifications, resulting in limited improvement. Furthermore, they don't take an intrinsic view into the generality of tensor operators. Morphing [25] addresses how to transform CGRA to accommodate dense and sparse tensor operators. We think that CGRAs often require large-area interconnect, memory and control units. Therefore the hardware often comprises small array, like  $8 \times 8$  and  $4 \times 4$ , with inadequate ability of data reuse and accelerating. Besides, critical precision aspect is overlooked. Tensorlib [16] focuses on the generation of hardware towards tensor operators. But it ignores the generality and leads to under-utilization of the array.

Above works all ignore the chance to co-optimize with the precision, which is the significant factor about implementation. Bitfusion [32] and GPNPU [33] consider the precision and computation at the same time, but are stuck in the limited range of computational precision and workload domains of operators.

## 3 INSIGHTS INTO MULTI-PRECISION TENSOR OPERATORS

In this section, we analyze the inherent characteristics of the computational workload and precision of tensor operators. We find the similarity between Matrix Multiplication and precision multiplication and a method classifying tensor operators. At last, we implement two discoveries on systolic array in each subsection respectively.

### 3.1 Similarity between Matrix Multiplication and Precision Multiplication

It is known that a long multiplication could be decomposed into several basic multiplications and additions. As shown in Figure 1(a), operands (i.e.  $X$  and  $Y$  are 32 bits) are decomposed into 8-bit limbs respectively (like  $X_{00}, Y_{00}$ , etc). It is obvious that all 8-bit limbs of the  $X$  and  $Y$  need to be cross-multiplied, and then the product of limbs should be added to get the result, which is similar to the computing pattern in matrix multiplication (by the way, the carry-bits among the product of limbs will be processed in the accumulator).

The reuse pattern of systolic array is in conformity with matrix multiplication, which can be migrated to multiplication detailed above. It could assume that the precision of a single PE is 8 bits. As can be seen from Figure 1(a), in Weight-Stationary (WS) mode, we can place the limbs in consecutive positions as the weight used in WS mode and the other one is taken as input streaming into the array from the leftmost PE. And 32-bit multiplication is achieved within 4 PEs.

As shown in Figure 1(b) of WS mode, in the column direction, the partial product of this multiplication flows downward to next row. Therefore the corresponding partial products produced at the same position are added, which is essentially equivalent to the addition of two multiplication results. Also, the input could be reused by row direction. After extending the row and column, it is equivalent to a multi-precision matrix multiplication.

The three types of dataflow supported by the systolic array in this work are WS, Input-Stationary(IS), and Output-Stationary(OS). The dataflow of IS is the same as that of WS, and the operands occupying the array are inputs. As shown in Figure 1(b) of OS mode, the difference between multi-precision in OS and WS is analogous to the distinctions in matrix multiplication. However, in OS mode, due to the fact that both input and weight are mapped onto the array, the size of the workload mapped on the array expands with multiple in both the column and row directions. In contrast, when working in WS mode, it only affects the row direction. Leveraging the array’s scalability, it could enable the realization of matrix multiplication with arbitrary multiples of PE’s precision.

### 3.2 Classification of Tensor Operators using p-GEMM and vector operators

It is worth reconsidering these tensor operators from a hardware perspective. The computational workload of tensor operators can be decomposed into two dimensions: algorithmic parallelism and arithmetic intensity. Arithmetic intensity reflects the data reuse opportunity of operators, indicating how many computations are performed after fetching data from memory. Algorithmic parallelism represents the parallel operations that could be extracted under certain circumstances, like the extent of vectorization achievable in a kernel. For instance, not absolutely, image processing algorithms always exhibit higher algorithmic parallelism compared to audio processing algorithms. These two dimensions serve as axes for partitioning existing tensor operators, yielding the following approximate classification results in Figure 2.

Along the arithmetic intensity axis, tensor operators with no intensity could only be compiled into vector operations without data reuse opportunity, while those with higher intensity could be transformed

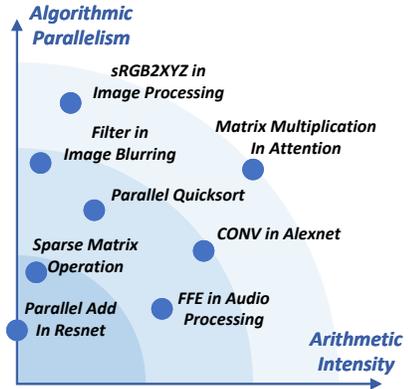


Figure 2: The indicative algorithmic parallelism and arithmetic intensity of some tensor operators.

into GEMM, like solving tensor contraction problems. Tensor contractions can be rewritten equivalently as the form of Transpose-Transpose-GEMM-Transpose sequences [5, 35]. Also, it is easy to manually convert to a GEMM kernel based on the data reuse characteristics. Based on the results of the above works, these GEMM come with different sizes, whose transformation results depend on both the arithmetic intensity and algorithmic parallelism, encompassing operations of varying size hierarchies such as matrix multiplication, matrix-vector multiplication, and vector inner product. In general, operators with high algorithmic parallelism and arithmetic intensity are transformed into matrix multiplication.

Therefore, we can define them as p-GEMM (p represents pseudo) including operators of arbitrary size. Although systolic dataflow architecture is ideal for handling matrix multiplication, it may lead to under-utilization problems when meeting various operators in p-GEMM unsuitable for the size of array, let alone the vector operations. Vector hardware has flexible interconnection and control units. So we could combine the vector and systolic architecture to improve the utilization of systolic array by reusing these fine-grained control modules. We will discuss this in detail in the next section.

## 4 HARDWARE ARCHITECTURE

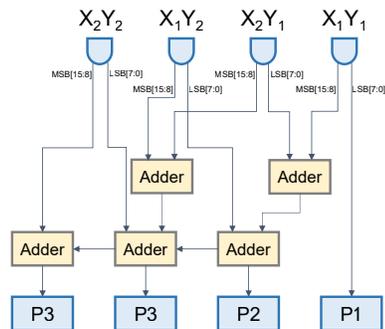
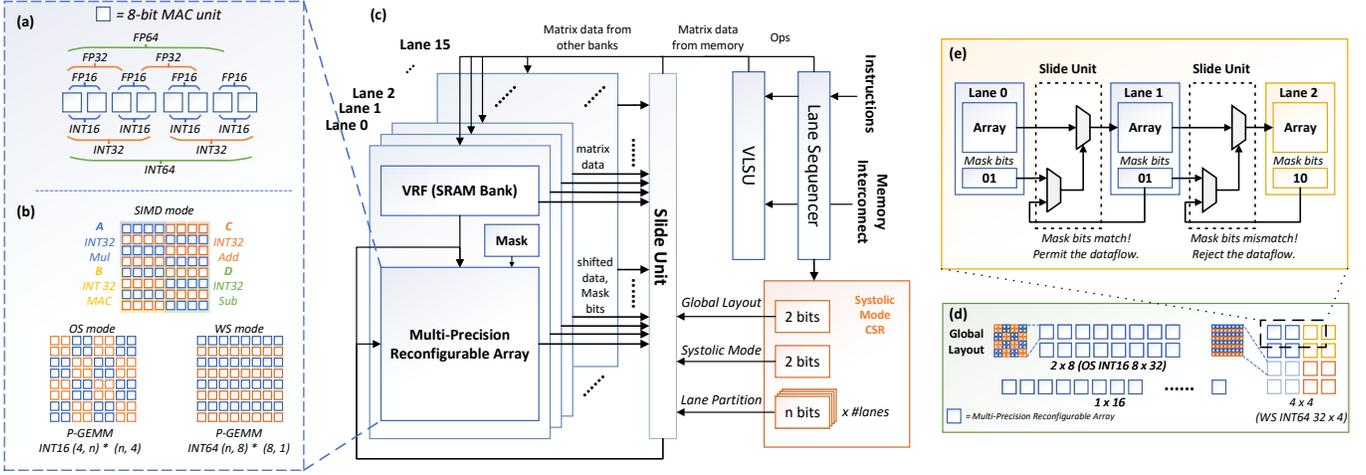


Figure 3: Implementation of a 16-bit multi-precision accumulator.



**Figure 4: Architecture overview: example of 16 lanes. (a) One row of MPRA cover various kinds of precision in WS mode. (b)  $8 \times 8$  MPRA operate multi-precision p-GEMM and vector operation. (c) The overview of GTA architecture (d) The MPRA combine the whole array with reconfigurable shape (e) The modified slide unit use mask bits to arrange the dataflow of array.**

We choose the VPU as the combining vector hardware. In this section, based on above discoveries, the GTA architecture will be explained, detailing the implementation of Multi-Precision Reconfigurable Array (MPRA) and reuse of fine-grained control and inter-connection logic in VPU.

#### 4.1 Multi-Precision Reconfigurable Array

We can call it Multi-Precision Reconfigurable Array (MPRA) which could implement systolic dataflow to operate multi-precision matrix multiplication as described in Section 3.1. As shown in Figure 4(a), the precision that a single PE supports is 8-bit and the multiplication is performed in a linear pattern. The number of column is set to 8 so that one row of PEs supports the  $8 \times n$  bits ( $n = 1, 2, 4, 8$ ) multiplication in WS or IS mode by utilizing corresponding partition. A MPRA set  $64(8 \times 8)$  PEs to support 64-bit in OS mode.

The multi-precision accumulator is composed of basic accumulator units to support accumulation in different bit width. We take a 16-bit accumulator as an example. Figure 3 shows the microarchitecture of a basic 16-bit accumulator unit. As shown, a 16-bit accumulator unit takes as input four 16-bit operands –  $X_1Y_1$ ,  $X_2Y_1$ ,  $X_1Y_2$  and  $X_2Y_2$ , which corresponds to four partial products of 16-bit multiplications generated by systolic array. Based on the mathematical property, the 16-bits accumulator unit uses shift-add operations to easily generate the results of 16-bit multiplications.

MPRA can be reconfigured to perform mantissa multiplication in different width, coordinated with other functional units to execute the FP operation. In addition to mantissa computation, the FPadd and FPmul require alignment, normalization, overflow processing, rounding and other steps. And the dominant area and energy consumption comes with the multiplier of the mantissa. Specifically, the mantissa multiplication for BP16, FP16, FP32, and FP64 can be equivalently represented as the multiplication of INT8, 12, 24, and 53, respectively.

#### 4.2 GTA Overall Architecture

As can be seen in Figure 4(c), the VPU could be configured with various lanes and a lane scheduler responsible for inter-lane communication and allocation of computational resources. Within each lane of original VPU, Multiply Accumulate (MAC) units in various precision are set up to meet the requirements of the vector extension instruction set, occupying a substantial area. We introduce one MPRA into each lane to replace these MAC units. Due to the configurability of GTA architecture, we can set arbitrary number of lanes. We take an example of 16 lanes below.

Besides the MAC unit, the PE in MPRA is equipped with three operand registers, systolic mode register, operation units (the same as lane's), and a centrally controlled finite state machine. The systolic mode register is synchronized with the global configuration in CSR, which controls the data transfer of single PE.

We introduce the Systolic Control and Status Register (SysCSR) as shown in Figure 4(c), which achieves a three-level array interconnect configuration. These levels correspond to the Global Layout, Systolic Mode, and Mask Group fields, respectively. The Global Layout field encodes the logical layout of lanes, and upon decoding, programs direction of the data shuffling (source lane and destination lane) in the Slide Unit as shown in Figure 4(d). We can define it as array arrangement. This influences the interconnection of the lane, thus inducing different shapes of the array. The Systolic Mode specifies the content when data movement is performed between lanes. For example, in the GEMM-OS mode, the movement with three sets of operands between lanes is required, while in the GEMM-WS(IS) mode, a set of input data and partial sum results need to be transferred. The situations in IS mode is similar to that of WS mode.

The Mask Sets consist of a group of mask bit sets in the number of lanes. After operator scheduling is completed, the hardware library generates mask bit sets based on shape information. Following the execution of SysCSR write instructions, the Lane Scheduler

- (1) Determine the absolute size of the systolic array.
- (2) Determine the absolute size of the load according to the accuracy and load size.

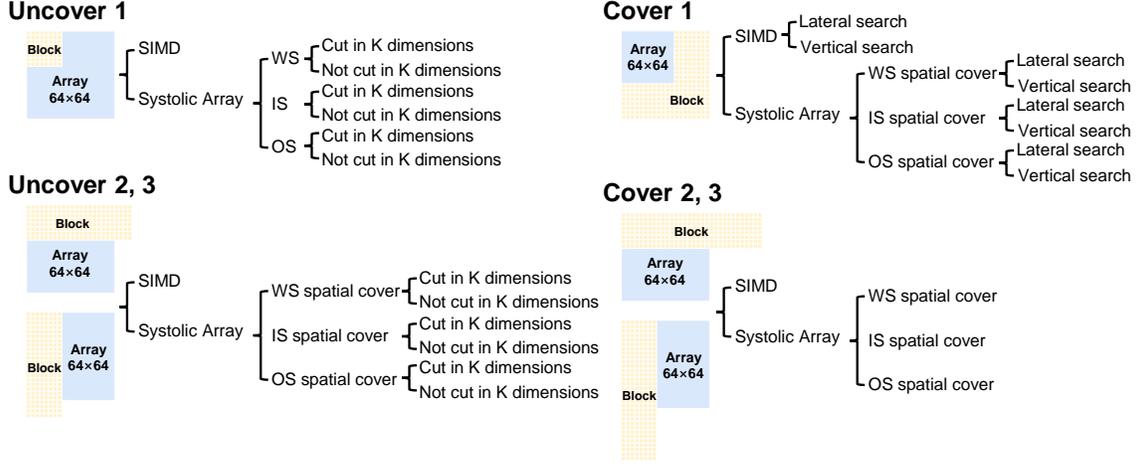


Figure 5: Dataflow pattern matching: 64 lanes and  $64 \times 64$  size array example.

loads these sets into the mask registers within each lane to control the data transfer. As shown in Figure 4(e), we introduce the Mask Match Mechanism to limit the data communication between lanes to logically divide lanes into different sub-regions, each of which contains lanes possessing a same set of mask bits permitting the data transfer. And the width of mask bits determines how many partitions are achievable in the architecture. Therefore GTA could combine its all MPRA as a whole array with several array rearrangements and freely schedule matrix operation of arbitrary size in high array utilization.

In conclusion, by reconfiguring the interconnection between PEs, MPRA can perform in either vector operator supported by original VPU or p-GEMM operator in multi-precision as shown in Figure 4(a) and (b). In other words, the previous works [10, 18, 27] only equip accelerator with GEMM and vector operation, while our work evolve into p-GEMM. The proposed architecture is applicable to a wide range of scenarios according to the number of configuring lanes, spanning from high-performance general computing to embedding accelerator. Compared with popular products in academia and industry, "VPU+Systolic Array" system [9, 17, 38], GTA improve area efficiency in both aspects of computational precision and combining vector and matrix unit.

## 5 SCHEDULING SPACE EXPLORING FOR P-GEMM OPERATOR

The vector operators are executed by GTA as usual VPU. However, for a p-GEMM operator, the scheduling approach is influenced by three factors, including the array resize, computational precision, dataflow. According to section 3.1, the size mapped on the GTA depends on the computational workload of the tensor operator transformed into p-GEMM and its corresponding precision. Also, the mapping size is influenced by selected dataflow. We employ three systolic dataflows (WS, IS, OS) and SIMD(vector) mode (some

p-GEMM operators may get better result from vectorization) to address this challenge. Additionally, the shape of whole array depends on array resize with numerous lanes. Different p-GEMM operators benefit from different array shape.

In the context of matrix multiplication in three systolic dataflows, typically characterized by three dimensions:  $M$ ,  $N$ , and  $K$ , where  $M$  and  $N$  can be assumed as two dimensions mapped onto the array spatially, and  $K$  represents the temporal dimension of the operation.

To enhance array utilization, the left computational tasks are mapped to the available part of array. In such cases, we can segment the  $K$  dimension to increase the array utilization to reduce computation time. Moreover, according to the scale-sim [31], larger continuous computation areas mapped on the systolic array result in a higher degree of data reuse. Conversely, if the segmentation is thinner, memory access counts increase. Therefore, in these situations, the theoretical conflict between improving array utilization (associated with computation speed) and data reuse (associated with the number of memory access) arises.

Additionally, when the load mapping size significantly exceeds the array, tiling in both row and column directions towards the load mapping often introduces idle array part in the edge column or row of every tiling loop. To mitigate this, tasks from the next column or row can be brought in prematurely to fill the idle array to get a spatial cover. Therefore, the optimal one can be selected from two directions. As illustrated in Figure 5, we can categorize these situations into four distinct cases:

- **Uncover 1:** The workload falls short of covering the array in two directions.
- **Uncover 2, 3:** The workload exceeds the array size in either the row or column direction, but the total size does not fully cover the entire array.
- **Cover 2, 3:** The workload exceeds the array size in either the row or column direction, achieving full coverage of the entire array.

**Table 1: Evaluated platforms' information.**

|            | GTA   | VPU-Ara   | GPGPU-NVIDIA H100                              | CGRA-hycube                                       |
|------------|---|---|--|---|
| Technology | 14nm  | 14nm  | 4nm  | 28nm  |
| Frequency  | 1GHz  | 250MHz  | 1755MHz  | 704MHz  |
| Area       | 0.35mm <sup>2</sup>                               | 0.33mm <sup>2</sup>                               | 814.00mm <sup>2</sup>                          | 7.82mm <sup>2</sup>                               |
| Cores      | 4   | 4   | 528-tensor core                                | 4x4   |
| Precision  | INT8, INT16, INT32, INT64, BP16, FP16, FP32, FP64 | INT8, INT16, INT32, INT64, BP16, FP16, FP32, FP64 | FP64, TF32, FP32, INT32, BP16, FP16, FP8, INT8 | INT8, INT16, INT32, INT64, BP16, FP16, FP32, FP64 |

- **Cover 1:** The workload exceeds the array size in both directions. The tiling placement could be in direction of Lateral or Vertical.

We define this approach as dataflow pattern matching, resulting in a dataflow scheduling space. Because of multitude of scheduling schemes, we could prioritize choices based on a comprehensive priority strategy, wherein diverse outcomes are normalized, and the preference is given to the one with the least sum of squares.

## 6 METHODOLOGY

### 6.1 Implementation

Our architecture is based on the open source vector processor Ara [4]. The RTL design is synthesized using Synopsys Design Compiler and the SAED 14nm library to evaluate the chip area and energy of our architecture. At the same time, we also synthesize Ara under the same conditions for comparison. The maximum clock frequency of reference design is only approximately 250 MHz under our technology library. After replacing the original computing units with MPRA, the design can be synthesized at 1 GHz. Obviously, it is very advantageous in terms of timing under the design of basic PE in 8-bit. The lane with  $8 \times 8$  MPRA can be implemented using only 60.76% of the original lane area and cover all precision. Adding additional processing units for floating-point numbers, the overall area is about the same as that of the original lane. Due to the reuse of existing structures, the control and other logic have only 6.06% area overhead over original Ara's setting 4 lanes.

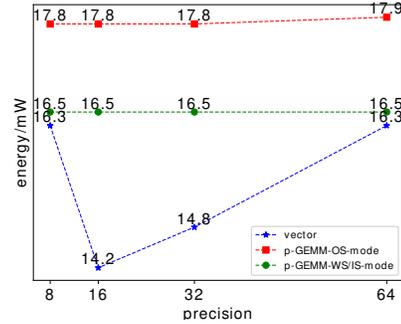
We measure the power consumption of MPRA under different precision and operation modes as shown in Figure 6. The abscissa refers to the corresponding integer and floating point precision. The clock constraint is set at 250 MHz. It can be observed that their energy consumption is approximately the same. Although MPRA's average energy consumption is a little higher than original lane's computation unit, it can significantly reduce the energy efficiency of memory access.

### 6.2 Workload

As listed in Table 2, we select important tensor applications in various precision that are prevalent in various domains, and decompose them into p-GEMM and vector operators for execution.

### 6.3 Baseline

GTA can be configured to operate in various scenarios, which vary in terms of memory configurations and array sizes. Therefore, our focus is specifically on two most important aspects, computing cycle

**Figure 6: MPRA's energy when executing different mode.****Table 2: Size and precision of p-GEMM operators in various workloads.**

| Workload | Description   | Precision |
|----------|---|-----------|
| BNM      | Big Numbers Multiplication in Scientific Computing and Encryption Field | INT64     |
| RGB      | SRGB2XYZ in Image Processing  | INT8      |
| FFE      | FFE in Audio Processing   | INT16     |
| MD       | Matrix Decomposition in Mathematical Analysis                           | INT32     |
| PCA      | PCA in Data Analysis  | FP64      |
| ALT      | Alexnet Training in ML  | FP32      |
| FFL      | GPT3-Feed-Forward Layers in ML  | BP16      |
| ALI      | Alexnet Inference in ML   | INT8      |
| Nerf     | Nerf in ML  | FP32      |

and memory access, for core computing architecture. We assume the same clock frequency and configure different number of MPRA to match the same area according to technology library. The baseline is listed in Table 1:

- **VPU:** The vector units are parallel precision units essentially. Therefore we take Ara as an example and set the lane with all kinds of precision units.
- **GPGPU:** We choose the NVIDIA's newest Hopper [7]. It involves Tensor Core and CUDA Core. For precision that Tensor Core cannot support, we use the closely higher precision.
- **CGRA:** We choose CGRA with all kinds of precision to compare performance with GTA. We choose the hycube architecture [19].

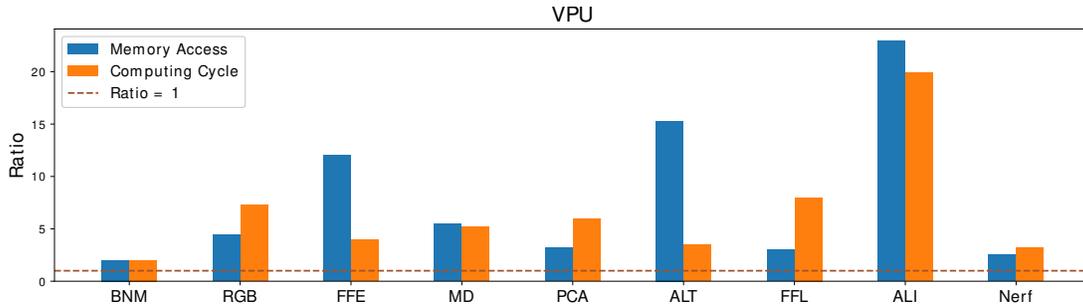


Figure 7: Comparisons with original VPU.

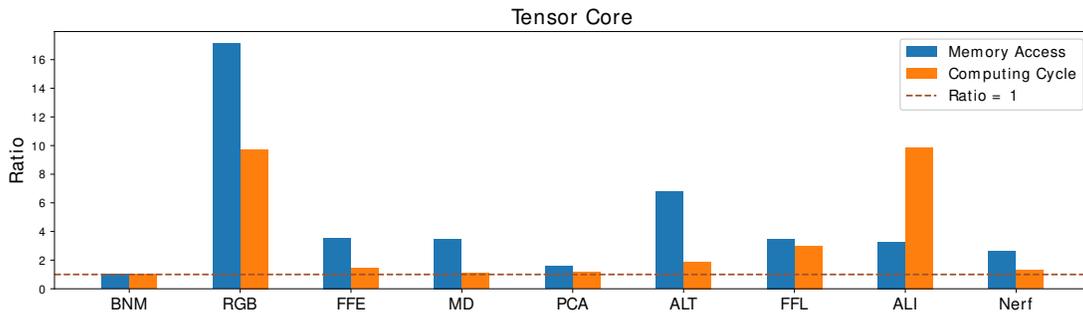


Figure 8: Comparisons with original GPGPU.

We develop cycle-accurate simulators, based on scale-sim [31], CGRA simulator morpher [8], VPU simulator [29] and GPU simulator [20, 26], of our architecture and other baselines to get the performance statistics. We verify the GTA’s simulator against our verilog implementation.

## 7 EVALUATION

### 7.1 Schedule Analysis

For the first time, we explore the mixed scheduling of precision and dataflow as shown in Figure 9. The values on both axes are the ratio to the minimum value gotten from all the available configurations. We choose one conv layer in Alexnet and set three kinds of precision used in case of real-world case. Due to the flexibility of our architecture and the exploration of scheduling space, we can display the statistical distribution of the scheduling cases. It is noteworthy that, unlike placing the precision units independently in other accelerators, different precision results in nonlinear distributions for the same operator.

### 7.2 Compared with VPU

The vector operators commonly encountered in every application. As shown in Table 3, an increase in throughput is observed as precision decreases when compared to the original VPU. The bitwidth of mantissa in floating-point numbers is not multiples of 8 bits, so the gain is not an integer. Leveraging the reconfigurability of 8 bit-width computing units, the utilization is higher than the original design.

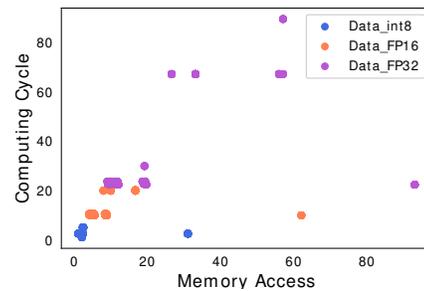


Figure 9: The scatter diagram of the scheduling cases with computing cycles and memory access index.

Table 3: SIMD gains for all data types.

| Data Type | Throughput | Data Type | Throughput |
|-----------|------------|-----------|------------|
| INT8      | 8×         | BP16      | 16×        |
| INT16     | 4×         | FP16      | 4×         |
| INT32     | 2×         | FP32      | 3.56×      |
| INT64     | 1×         | FP64      | 1.3×       |

Compared to systolic array architecture, the chaining technique in VPU exhibits weaker data reuse capability. Additionally, the number of computation units decreases with higher precision. Moreover,

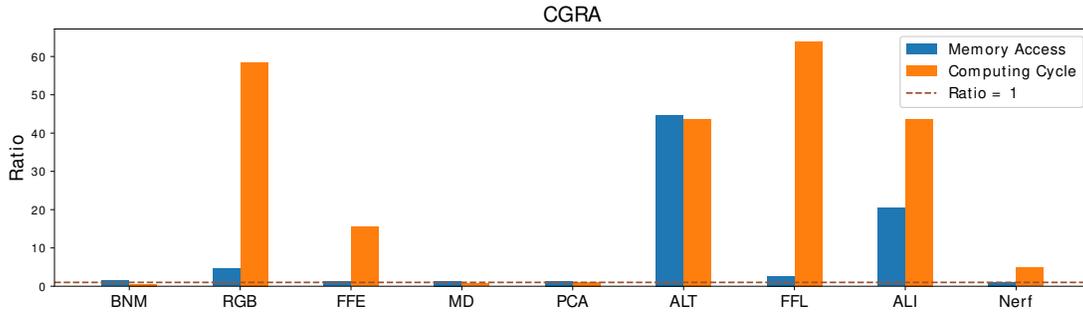


Figure 10: Comparisons with original CGRA in p-GEMM operators.

factors at the microarchitectural level, such as maximum vector length, also impose limitations on computational speed. As shown in Figure 7, due to the area advantages brought by the additional computing units and the memory access advantages brought by systolic architecture, we can notice the savings in computational time and memory access.

The average memory access saving is  $7.76\times$  and the average computational speedup is  $6.45\times$ . The results demonstrates strong overall performance. The comparison with VPU demonstrates the effectiveness of our structure in achieving notable generality in handling tensor operators in various domains.

### 7.3 Compared with GPGPU

The GPGPU consists of Tensor Core and CUDA Core. Tensor Core is only for accelerating GEMM. This work can be also regarded as the merger of Tensor Core and CUDA Core. Therefore, GTA have better area efficiency besides of the factor of precision. Furthermore, aiming to maximize the throughput, Tensor Core is consisted of small cube computing matrix multiplication, which requires large numbers of memory operations and high on-chip memory bandwidth. To get a fair comparison, we give the decomposed vector operator to cuda core and the p-gemm operator to tensor core.

As shown in Figure 8, there are still superiority of computing speed and the memory access. Due to the high throughput in high precision of Tensor Core, some performance remain modest, but there is a significant improvement in memory access. The average memory access saving is  $5.35\times$  and the average computational speedup is  $3.39\times$ .

### 7.4 Compared with CGRA

CGRA realizes the flexibility for tensor operators, which use wordlevel reconfigurability and contain larger logic blocks and datapath-oriented interconnections. Therefore, CGRA is consisted of small arrays in physical implementation. As a result, they are relatively weak in acceleration and data reuse. It can be seen from the experimental data that high-precision computing units such as FP64 have a larger number of settings and can be on par with GTA in performance. But there are many PE in the idle state in the mapping. As shown in Figure 10, overall it still performs well. GTA is able to achieves  $8.76\times$  memory efficiency and  $25.83\times$  speedup over of CGRA.

## 8 CONCLUSION

Based on the similarity between matrix multiplication and precision multiplication and classification of tensor operators using p-GEMM and vector, We propose GTA, a general tensor accelerator combining of systolic array and VPU. GTA covers tensor operators of arbitrary precision. Besides, we explore the schedule space consisting of dataflow, precision and array resize. GTA achieves significant speedup and memory benefits compared to other general accelerators.

## REFERENCES

- [1] David H Bailey, Roberto Barrio, and Jonathan M Borwein. 2012. High-precision computation: Mathematical physics and dynamics. *Appl. Math. Comput.* 218, 20 (2012), 10106–10121. doi: [10.1016/j.amc.2012.03.087](https://doi.org/10.1016/j.amc.2012.03.087).
- [2] Kaifeng Bi, Lingxi Xie, Hengheng Zhang, Xin Chen, Xiaotao Gu, and Qi Tian. 2023. Accurate medium-range global weather forecasting with 3D neural networks. *Nature* 619, 7970 (2023), 533–538. doi: [10.1038/s41586-023-06185-3](https://doi.org/10.1038/s41586-023-06185-3).
- [3] Jared Casper and Kunle Olukotun. 2014. Hardware acceleration of database operations. In *Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays*. 151–160.
- [4] Matheus Cavalcante, Fabian Schuiki, Florian Zaruba, Michael Schaffner, and Luca Benini. 2020. Ara: A 1-GHz+ Scalable and Energy-Efficient RISC-V Vector Processor With Multiprecision Floating-Point Support in 22-nm FD-SOI. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28, 2 (2020), 530–543. <https://doi.org/10.1109/TVLSI.2019.2950087> doi: [10.1109/TVLSI.2019.2950087](https://doi.org/10.1109/TVLSI.2019.2950087).
- [5] Lorenzo Chelini, Andi Drebes, Oleksandr Zinenko, Albert Cohen, Nicolas Vasilache, Tobias Grosser, and Henk Corporaal. 2021. Progressive Raising in Multi-level IR. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. 15–26. <https://doi.org/10.1109/CGO51591.2021.9370332> doi: [10.1109/CGO51591.2021.9370332](https://doi.org/10.1109/CGO51591.2021.9370332).
- [6] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2016. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits* 52, 1 (2016), 127–138.
- [7] Jack Choquette. 2023. Nvidia hopper h100 gpu: Scaling performance. *IEEE Micro* (2023).
- [8] Wijerathne Dhananjaya, Li Zhaoying, Karunaratne Manupa, Peh Li-Shiuan, and Mitra Tulika. 2022. Morphier: An Open-Source Integrated Compilation and Simulation Framework for CGRA. *Fifth Workshop on Open-Source EDA Technology (WOSET)* (November 2022).
- [9] Alexander Fell, Daniel J Mazure, Teresa C Garcia, Borja Perez, Xavier Teruel, Pete Wilson, and John D Davis. 2021. The marenstrum experimental exascale platform (meep). *Supercomputing Frontiers and Innovations* 8, 1 (2021), 62–81. doi: [10.14529/jsfi210105](https://doi.org/10.14529/jsfi210105).
- [10] Cong Guo, Yangjie Zhou, Jingwen Leng, Yuhao Zhu, Zidong Du, Quan Chen, Chao Li, Bin Yao, and Minyi Guo. 2020. Balancing Efficiency and Flexibility for DNN Acceleration via Temporal GPU-Systolic Array Integration. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6. <https://doi.org/10.1109/DAC18072.2020.9218732> doi: [10.1109/DAC18072.2020.9218732](https://doi.org/10.1109/DAC18072.2020.9218732).
- [11] Meng Han, Liang Wang, Limin Xiao, Tianhao Cai, Zeyu Wang, Xiangrong Xu, and Chenhao Zhang. 2023. ReDas: Supporting Fine-Grained Reshaping and Multiple Dataflows on Systolic Array. arXiv:2302.07520 [cs.AR] doi: [10.48550/arXiv.2302.07520](https://doi.org/10.48550/arXiv.2302.07520).

- [12] Greg Henry, Ping Tak Peter Tang, and Alexander Heinecke. 2019. Leveraging the float16 artificial intelligence datatype for higher-precision computations. In *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*. IEEE, 69–76. doi: [10.1109/ARITH.2019.00019](https://doi.org/10.1109/ARITH.2019.00019).
- [13] Kuan-Chieh Hsu and Hung-Wei Tseng. 2021. Accelerating applications using edge tensor processing units. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14. doi: [10.1145/3458817.3476177](https://doi.org/10.1145/3458817.3476177).
- [14] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2704–2713. doi: [10.48550/arXiv.1712.05877](https://doi.org/10.48550/arXiv.1712.05877).
- [15] Geonhwa Jeong, Eric Qin, Ananda Samajdar, Christopher J Hughes, Sreenivas Subramoney, Hyesoon Kim, and Tushar Krishna. 2021. Rasa: Efficient register-aware systolic array matrix engine for cpu. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 253–258. doi: [10.1109/DAC18074.2021.9586257](https://doi.org/10.1109/DAC18074.2021.9586257).
- [16] Liancheng Jia, Zizhang Luo, Liqiang Lu, and Yun Liang. 2022. Automatic generation of spatial accelerator for tensor algebra. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022).
- [17] Norman P Jouppe, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, et al. 2021. Ten lessons from three generations shaped google's tpuv4i: Industrial product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1–14. doi: [10.1109/ISCA52012.2021.00010](https://doi.org/10.1109/ISCA52012.2021.00010).
- [18] Yuhao Ju and Jie Gu. 2022. A 65nm systolic neural CPU processor for combined deep learning and general-purpose computing with 95% PE utilization, high data locality and enhanced end-to-end performance. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. IEEE, 1–3. doi: [10.1109/ISSCC42614.2022.9731757](https://doi.org/10.1109/ISSCC42614.2022.9731757).
- [19] Manupa Karunaratne, Aditi Kulkarni Mohite, Tulika Mitra, and Li-Shiuan Peh. 2017. HyCUBE: A CGRA with reconfigurable single-cycle multi-hop interconnect. In *Proceedings of the 54th Annual Design Automation Conference 2017*. 1–6.
- [20] Mahmoud Khairy, Zhesheng Shen, Tor M Aamodt, and Timothy G Rogers. 2020. Accel-Sim: An extensible simulation framework for validated GPU modeling. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 473–486.
- [21] Donald E Knuth. 2014. *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional. doi: [10.1137/1012065](https://doi.org/10.1137/1012065).
- [22] Allen Baum Krste Asanovic, Alon Amid. 2021. RISC-V "V" Vector Extension Version 1.0. <https://github.com/riscv/riscv-v-spec/tree/v1.0> <https://github.com/riscv/riscv-v-spec/tree/v1.0>
- [23] Joungwoo Lee, Jinwoo Choi, Jaeyeon Kim, Jinho Lee, and Youngsok Kim. 2021. Dataflow mirroring: Architectural support for highly efficient fine-grained spatial multitasking on systolic-array npus. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 247–252. doi: [10.1109/DAC18074.2021.9586312](https://doi.org/10.1109/DAC18074.2021.9586312).
- [24] Cangyuan Li, Ying Wang, Huawei Li, and Yinhe Han. 2023. APPEND: Rethinking ASIP Synthesis in the Era of AI. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6. doi: [10.1109/DAC56929.2023.10247872](https://doi.org/10.1109/DAC56929.2023.10247872).
- [25] Liqiang Lu and Yun Liang. 2021. Morphling: A reconfigurable architecture for tensor computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 11 (2021), 4733–4746.
- [26] Weile Luo, Ruibo Fan, Zeyu Li, Dayou Du, Qiang Wang, and Xiaowen Chu. 2024. Benchmarking and Dissecting the Nvidia Hopper GPU Architecture. *arXiv preprint arXiv:2402.13499* (2024).
- [27] Mateo Vázquez Maceiras, Muhammad Waqar Azhar, and Pedro Trancoso. 2022. VSA: A Hybrid Vector-Systolic Architecture. In *2022 IEEE 40th International Conference on Computer Design (ICCD)*. 368–376. <https://doi.org/10.1109/ICCD56317.2022.00061> doi: [10.1109/ICCD56317.2022.00061](https://doi.org/10.1109/ICCD56317.2022.00061).
- [28] Ardavan Pedram, Robert A Van De Geijn, and Andreas Gerstlauer. 2012. Codesign tradeoffs for high-performance, low-power linear algebra architectures. *IEEE Trans. Comput.* 61, 12 (2012), 1724–1736.
- [29] Cristóbal Ramírez, César Alejandro Hernández, Oscar Palomar, Osman Unsal, Marco Antonio Ramírez, and Adrián Cristal. 2020. A risc-v simulator and benchmark suite for designing and evaluating vector architectures. *ACM Transactions on Architecture and Code Optimization (TACO)* 17, 4 (2020), 1–30.
- [30] James R Reinders. June 2017. Intel AVX-512 Instructions. <https://software.intel.com/en-us/blogs/2013/avx-512-instructions> <https://software.intel.com/en-us/blogs/2013/avx-512-instructions>
- [31] Ananda Samajdar, Jan Moritz Joseph, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2020. A systematic methodology for characterizing scalability of DNN accelerators using SCALE-sim. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 58–68. doi: [10.1109/ISPASS48437.2020.00016](https://doi.org/10.1109/ISPASS48437.2020.00016).
- [32] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Joon Kyung Kim, Vikas Chandra, and Hadi Esmaeilzadeh. 2018. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 764–775. doi: [10.1109/ISCA.2018.00069](https://doi.org/10.1109/ISCA.2018.00069).
- [33] Zhuoran Song, Jianfei Wang, Tianjian Li, Li Jiang, Jing Ke, Xiaoyao Liang, and Naifeng Jing. 2020. Gpnp: Enabling efficient hardware-based direct convolution with multi-precision support in gpu tensor cores. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.
- [34] Andreas Spanias, Ted Painter, and Venkatraman Atti. 2006. *Audio signal processing and coding*. John Wiley & Sons. doi: [10.1038/s42003-021-01817-8](https://doi.org/10.1038/s42003-021-01817-8).
- [35] Paul Springer and Paolo Bientinesi. 2018. Design of a High-Performance GEMM-like Tensor-Tensor Multiplication. *ACM Trans. Math. Softw.* 44, 3, Article 28 (jan 2018), 29 pages. <https://doi.org/10.1145/3157733> doi: [10.1145/3157733](https://doi.org/10.1145/3157733).
- [36] Nigel Stephens, Stuart Biles, Matthias Boettcher, Jacob Eapen, Mbou Eyole, Giacomo Gabrielli, Matt Horsnell, Grigorios Magklis, Alejandro Martinez, Nathanael Premillieu, Alastair Reid, Alejandro Rico, and Paul Walker. 2017. The ARM Scalable Vector Extension. *IEEE Micro* 37, 2 (2017), 26–39. <https://doi.org/10.1109/MM.2017.35> doi: [10.1109/MM.2017.35](https://doi.org/10.1109/MM.2017.35).
- [37] Bernard Widrow and István Kollár. 2008. *Quantization noise: roundoff error in digital computation, signal processing, control, and communications*. Cambridge University Press. doi: [10.1017/CBO9780511754661](https://doi.org/10.1017/CBO9780511754661).
- [38] Yuqi Xue, Yiqi Liu, Lifeng Nai, and Jian Huang. 2023. V10: Hardware-Assisted NPU Multi-tenancy for Improved Resource Utilization and Fairness. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–15. doi: [10.1145/3579371.3589059](https://doi.org/10.1145/3579371.3589059).
- [39] Ahmet Caner Yüzügüler, Canberk Sönmez, Mario Drumond, Yunho Oh, Babak Falsafi, and Pascal Frossard. 2023. Scale-out Systolic Arrays. *ACM Transactions on Architecture and Code Optimization* 20, 2 (2023), 1–25. doi: [10.1145/3572917](https://doi.org/10.1145/3572917).
- [40] Yujia Zhai, Mohannad Ibrahim, Yiqin Qiu, Fabian Boemer, Zizhong Chen, Alexey Titov, and Alexander Lyashevsky. 2022. Accelerating encrypted computing on intel gpus. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 705–716. doi: [10.1109/IPDPS53621.2022.00074](https://doi.org/10.1109/IPDPS53621.2022.00074).