2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) | 978-1-6654-3946-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/ISVLSI51109.2021.00050

Analyzing the Design Space of Spatial Tensor Accelerators on FPGAs

Liancheng Jia[1], Zizhang Luo[1], Liqiang Lu[1], Yun Liang[1,2*]
[1] Center for Energy-efficient Computing and Applications, School of EECS, Peking University, Beijing, China
[2] Beijing Academy of Artificial Intelligence, Beijing,China
Email: {jlc, semiwaker, liqianglu, ericlyun}@pku.edu.cn

***Abstract*—Tensor algebra applications are widely used in various domains, and these applications can deliver high performance and low power when accelerated on FPGA-based spatial hardware accelerators. The architecture of spatial accelerator has a large design space including PE unit, workload mapping, PE interconnection, and memory subsystem. An efficient design space exploration (DSE) framework is required to find the optimal architecture for an tensor algorithm with limited FPGA resources. Previous works propose DSE frameworks for hardware design and hardware-software mapping. But the design space is limited and prior works lack of accurate performance models. In this paper, we propose a systematic approach to analyze the design space when mapping tensor applications onto the spatial accelerators on FPGAs. We also propose performance models to evaluate the performance of different architecture design choices. Our presented design space is more comprehensive than prior works and we support a wider range of tensor applications. Our performance models consider the FPGA-specific hardware specifications including DSP usage, memory capacity, bandwidth and frequency.**

I. Introduction

Tensor algebra finds application in various applications. For example, convolutions are widely used in machine learning, MTTKRP, TTM, and tensor decomposition are used in data analytic and scientific computing. Tensor algebra features different computation kernel, data dependency, data access pattern, and loop nest dimensions, which requires different hardware architectures for efficient processing. Spatial architecture has been demonstrated as an effective architecture for tensor algorithms thanks to its regular design and high data reuse. Due to its flexibility, FPGAs are widely used to implement spatial accelerators for tensor algebra[1, 2].

Spatial accelerators have a complex architecture hierarchy including algorithm, workload mapping, PE interconnection and memory subsystem. State-of-the-art spatial architectures use different dataflow and mapping schemes. The workload is partitioned differently, and the PE array uses different connection patterns. For example, [1] uses output stationary systolic array dataflow because the output tensor elements stay inside PE during execution. Similarly, [3] uses weight stationary dataflow and [4] uses row stationary dataflow where the PEs are connected both vertically and diagonally. Some previous works [5–9] study the FPGA-specific optimization for DNN accelerators.

*Corresponding Author.

To develop high performance and energy-efficient spatial accelerators, prior works proposed design space exploration and hardware generation frameworks to analyze both hardware architecture and software-hardware mapping problem [10–14]. PolySA [2], AutoSA[15], Susy [16] , Gemmini [17] and X.Wei's work [1] are able to generate the hardware RTL code as well as search for the optimal one, but they are limited to systolic array architecture. Tensorlib [18] provides an accelerator generation framework that supports various spatial architectures including systolic array, reduction tree and multicast. Timeloop [10] presents a highly flexible hardware model. It defines PE array dataflow with loop unrolling and delta, which is able to represent typical dataflow such as systolic and multicast. However, the application is limited to standard convolution algorithm, and it cannot analyze the difference of dataflows with the same PE workloads. Interstellar [11] uses similar notations to represent dataflows and it supports clustering, but the analysis on memory reuse is not accurate and didn't consider the reuse of memory for a specific tensor. MASTERO [12] creates data-centric notation to represent dataflows and it supports a larger variant of CNN algorithms. However, it can not flexibly define the connection pattern of PEs. TENET [19] uses relation-centric notation to design a more accurate model to evaluate dataflows, but it doesn't consider the memory effect. Other works such as dMazeRunner [13], MAGNet [20] and HASCO [21] only supports a fixed set of architectures.

In this paper, we propose an analysis on the design space of FPGA-based spatial accelerators for tensor applications. We form the design space in three steps. Firstly, we specify the workloads assigned to PE array and cluster level with loop factoring and reordering. Next, we use space-time transformation to detect the reuse behavior of tensor data and generate PE interconnection patterns. Finally, the remaining loop nest is factored to build the memory hierarchy. In this way, we form a comprehensive design space of spatial accelerators. Based on the design space, we present a performance model which analyzes the performance trade-off of different design choices. The performance model takes consideration of hardware-specific parameters including DSP resource, memory resource, bandwidth and frequency. In summary, we make the following contributions,

- We present a comprehensive analysis of the hardware design space of spatial accelerator for tensor algebras.

978-1-6654-3946-6/21/$31.00 ©2021 IEEE
DOI 10.1109/ISVLSI51109.2021.00050

- We analyze the impact of different hardware design choices on the performance of hardware accelerators by modeling the utilization of hardware resource and memory efficiency.
- We evaluate different accelerator designs on FPGAs and analyze the impact of both cycle-level performance and throughput of different design choices and hardware parameters.

II. The Design Space of Spatial Accelerator

In this section, we discuss the design space of the spatial tensor accelerator in four steps. The tensor algorithm is defined using a perfectly nested loop where all the assignments are inside the innermost loop. Firstly, each processing element (PE) implements the assignment function in the innermost loop. Next, a partition of the loop nest is mapped to the PE array execution. The detailed PE array dataflow and interconnection can be derived with space-time analysis. Finally, the remaining loop nest is partitioned again and data inside the inner loop is buffered with on-chip BRAM. When the tensor algorithm and PE structure are fixed, we divide the entire design space of spatial accelerator into three subspaces: workload mapping, PE dataflow and memory hierarchy.

A. The PE Algorithm

Usually, each PE performs one computation formula in each time step (one or multiple clock cycles). The computation can be vectorized and the data type of computation can also be altered according to requirement. The PE computation cell architecture is determined by the tensor algebra formula, and is often implemented with specialized IPs, such as 1-D convolution [4], vectorized multiplication-accumulation (MAC) [1, 3, 17] or some other computation kernels.

B. PE Array Workload Mapping

The first step to create the accelerator is to map the workload to PE array in a coarse-grained manner, by only assigning the loop nest executed in PE array without specifying the concrete dataflow. The typical shape of PE array is a 2D meshgrid, which can run a 3-dimension loop nest in a certain time. However, most tensor algebras contain more than three loop nests. Only a 3D subspace of loop nests can be selected to run in the PE array, and others run sequentially in different periods. Multiple PE arrays can form clusters and different partial workloads can also run concurrently in each engine, which enables more scheduling possibilities.

Figure 2 shows two workload mapping schemes for Depthwise-Conv2D algorithm (Figure 1) on a 2D PE array. Mapping #1 maps C, OY and OX loops to the PE array while #2 maps C, OX and FY. When mapping different workloads to PE array, the dimension of each tensor in the PE array becomes different, which also lead to different PE array dataflows.

The size of design space for workload mapping subspace is $S_{wm} = C_N^3 \times max\_pe\_size^2 \times max\_t$, which means selecting three loops for PE array execution. The selected loops are split into tiles to fit into the PE array's space and time limitation (*max_pe_size* and *max_t*).

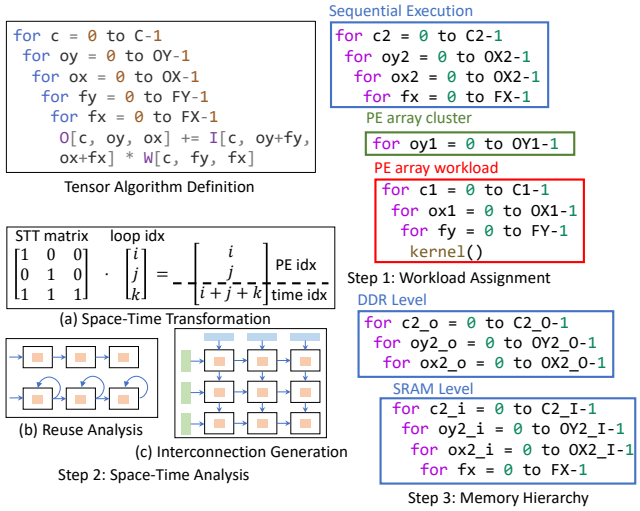
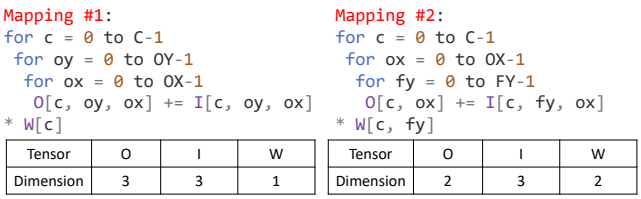


Fig. 1. Design space of spatial accelerators



Tensor	O	I	W
Dimension	3	3	1

Tensor	O	I	W
Dimension	2	3	2

Fig. 2. Workload mapping examples for Depthwise-Conv2D

C. Dataflow Analysis

After mapping the workload to the PE array, we use space-time analysis to find the dataflow and the interconnection pattern of the PE array. Prior works shown that the PE dataflow and interconnection can be generated with space-time transformation (STT) [2, 15, 18, 22]. The space-time analysis consists three steps. Given a computation loop nest and a transformation matrix, it first determines the time and position (PE index) of each computation instance by multiplying the loop index vector with the transformation matrix. For 2D PE array, the size of transformation matrix is 3×3, which maps a 3D loop indexes to 2D space and 1D time index. Next, it analyzes the reuse pattern of the same data element in the PE array. For example, the data element appears in the same PE at different cycles, or appears in different PEs. The reuse pattern is represented by reuse vector $\vec{v} = (\Delta x, \Delta y, \Delta t)$, where the tensor data is reused in (x, y, t) and $(x+\Delta x, y+\Delta y, t+\Delta t)$. The reuse pattern corresponds to the PE array dataflow and forms the PE interconnection patterns[18]. Each tensor in the computation can have different access patterns, resulting in different types of PE interconnection.

With space-time analysis, we identify six types of PE interconnection patterns as shown in Table I. In unicast dataflow, the tensor element only appears once in the execution stage without reusing. In stationary dataflow, data is reused in the same PE at different time steps. In systolic dataflow, data transfers between PEs at each time step. In multicast dataflow, data appears at different PEs at the same time. Rep-Systolic and broadcast dataflow is used when there are two reuse

TABLE I
PE INTERCONNECTION TYPES

Name	Reuse	Description
Unicast	1	No inter-PE reuse
Stationary	N	Data stays inside one PE during execution
Systolic	N	Data transfers to adjacent PE every cycle
Multicast	N	Data is sent to PEs in a row at the same cycle
Rep-Systolic	N^2	Data stays in multiple PEs during execution
Broadcast	N^2	Data is broadcasted to all PEs at the same cycle

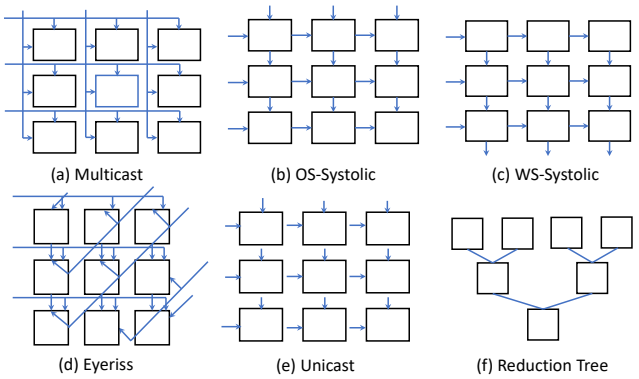


Fig. 3. Examples of PE dataflow

vectors and data is reused in two dimensions.

Figure 3 gives several examples of PE array dataflow topology by combining different patterns together. The first five designs have the same amount of PEs but different connection patterns. (a) and (d) shows two multicast connections where PEs are directly connected to the same memory port. (b) and (c) shows output-stationary and weight-stationary systolic arrays with adjacent PE connection. Unicast dataflow (e) means data is only used once in the PE array without reusing. Reduction tree structure (f) is used when the output tensor is applied with a multicast dataflow. Partial results are generated at the same time which requires a reduction tree for the final result.

In space-time transformation, each dataflow can be expressed with a 3×3 matrix. The rank of a legal STT matrix must be 3 so that the mapping between loop iterators and space-time vector is a one-to-one mapping. To reduce the design space, we do not consider the non-adjacent PE connection patterns by limiting the absolute value of STT matrix elements to 0 or 1. In summary, the size of dataflow subspace (S_{stt}) is the number of full-rank 3×3 matrix whose elements are either 0 or 1, which is 174.

D. Memory Hierarchy

Typical spatial accelerators on FPGA use a three-level memory hierarchy: off-chip DRAM, on-chip BRAM, and register file. Off-chip DRAM usually has only one port but (theoretically) infinite capacity. On-chip BRAM is divided into multiple banks. Only one data element in each bank can be accessed per cycle. The register file is individual to each PE. It has a small capacity but data can be randomly accessed. To define the memory hierarchy from the computation, the computation loop nest is also divided into three levels, and each level of loop nest is mapped to one level of the memory

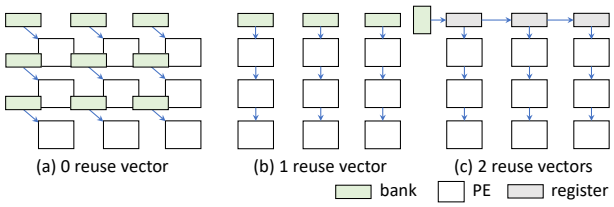


Fig. 4. The memory bank arrangement

hierarchy. The workload that is mapped to PE array uses a register file for storage, and the rest loops are further reordered and split again. The outer loops use off-chip DRAM and inner loops use on-chip BRAM for data buffering.

The hierarchy of on-chip BRAM is determined by the PE connection pattern. Figure 4 shows the connection between PE array and the memory bank. When data is not reused (unicast), every PE connects to a unique memory bank, and the number of memory bank is same as PE's number. When data is reused in systolic, stationary or multicast manner, the number of reuse vectors is one and each memory bank connects to one column or row of the PE array. Finally, when the data is reused with two dimensions (2 reuse vectors), there is only one memory bank to transfer data with all PEs.

The loop nest in each memory hierarchy can be permuted arbitrarily. The permutation can affect the memory access pattern of its lower-level memory. For example, in Figure 1 step 3, if the c2_o loop interchange with ox2_o loop in DDR level, the access pattern of BRAM level will become different. If a tensor whose index is irrelevant with c2_o, it can reuse the data in BRAM without fetching from DDR again. In certain cases, data of different tiles can overlap, and only part of the data needs to be reloaded, providing the opportunity for partial data reuse. The size of memory design space is $S_{mem} = {F_{max}}^N \times N!$, where F_{max} is the maximum value of memory-level loop factorization.

E. Putting it Together

We have discussed three sub-spaces to form the entire design space of spatial accelerator by multiplying the three sup-spaces together as shown in Equation 1.

$$S = S_{wm} \times S_{stt} \times S_{mem} \tag{1}$$

We use Depthwise-Conv2D in Figure 1 as an example to form the design space. The algorithm has five loop iterators. $S_{wm} = 20 \times max\_pe\_size^2 \times max\_t$, $S_{stt} = 148$ which is the number of full-rank 0-1 3×3 matrix. $S_{mem} = {F_{max}}^5 \times 120$. The space can be pruned to remove identical designs.

III. CYCLE-LEVEL PERFORMANCE ANALYSIS

In this section, we present a model to analyze the cycle-level performance of different designs of spatial accelerators. The hardware utilization factor P can be represented as the multiplication of three factors: resource usage, PE activeness and memory limitation, as shown in Equation 2. P reaches maximum value 1 when all PEs are actively working in every cycle. We will analyze the resource utilization and PE

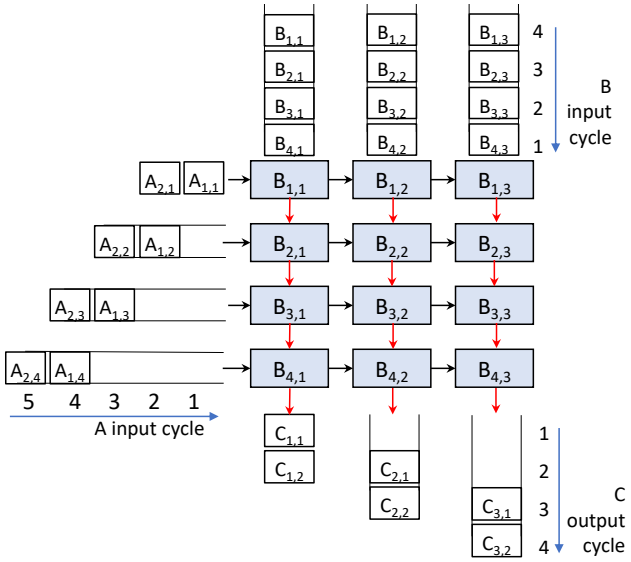


Fig. 5. PE array execution example in systolic dataflow

activeness in this section, and discuss the memory factor in the next section.

$$P = P_{res} \times P_{active} \times P_{mem} \tag{2}$$

A. Hardware Resource Usage

The hardware resource usage (P_{res}) is measured by the utilization of computation resource in FPGA (used in circuit or not). Assume that the FPGA uses DSP resources to perform computation, when the architecture of PE is determined, the DSP usage is proportional to the size of PE array. Since space-time analysis transforms every loop index to a PE index and time index, the PE size is determined by the maximum PE index generated by STT after workload assignment. The P_{res} can be represented as follow:

$$P_{res} = \frac{\#PE \times DSP\_per\_PE}{\#totalDSP} \tag{3}$$

B. PE Activeness

The PE activeness factor (P_{active}) is measured by the average active cycle (performing computation) divided by total cycles. The PE can be idle due to pipeline filling and draining, or inter-PE data communication. When STT maps a computation instance to PE array, one PE can be assigned with an illegal loop index (out of loop range, or negative) at a particular time. Under this circumstance, the PE is idle at the time and doesn't perform actual computation.

Another factor that influences the PE activeness is the communication delay of stationary data. Stationary data stays in the PE during the entire execution stage and updates after the stage finishes. For example, when the execution cycle of systolic array C is smaller than its size S, the stationary data will require S cycles to transfer, leading to $S-C$ idle cycles in every S cycle.

Figure 5 shows the execution of a systolic dataflow for matrix multiplication $C = A \times B$. It demonstrates the effect of idle cycle caused by both pipeline and communication delay. At the beginning, only the PE at upper-left corner receives valid data, and other PEs are idle because the pipeline is not filled. The pipeline delay only occurs at the beginning and end of computation when pipeline is not full. When switching computation stage (the matrix B changes), the new matrix B are sent to PEs with double buffer, so the update of stationary data can be concurrent with calculation and the computation pipeline isn't affected. However, it takes 4 cycles to pass the data to every PE. If the cycle of each computation stage is smaller, the pipeline has to stall to wait for the data update. In Figure 5, the pipeline needs to stall for two cycles.

IV. Memory Analysis

The roofline model shows that the performance of the accelerator is limited by either computation performance or memory bandwidth. The previous section discussed the factors that influence computation performance. In this section, we analyze the memory bandwidth requirement of the spatial accelerator according to PE array dataflow and on-chip memory hierarchy. Since the bandwidth between device memory and the accelerator is a fixed value, when the requirement of memory bandwidth exceeds the hardware's limitation, the computation must stall to wait for data transfer. Otherwise, the memory bandwidth is sufficient and the performance is limited by computation capacity.

To measure the memory requirement, we have to infer the amount of off-chip memory accessing during a certain time. The space-time analysis transforms the computation loop nest to time stamp, so the length of time is already determined. The same data element can be reused multiple times on-chip without being loaded again from the off-chip memory. However, the data access pattern for each tensor in the computation is different, and they show different reuse behaviors. The amount of off-chip memory access can be estimated as the total PE workload divided by the reuse volume (*#reuse*) for each tensor.

A. Data Reuse Analysis and Memory Size Requirement

Data reuse appears when the same tensor data is accessed multiple times in the computation loop. We have discussed the data reuse inside the PE array in section II.C. When data is reused inside PE array, the on-chip memory is only accessed once, and data is reused in the PE array at different space and time. Data reuse also occurs in the on-chip memory. The on-chip memory is large enough to cache data elements so that data can be accessed multiple times with on-chip memory without reloaded from device memory.

```
for i = 0 to I
  for j = 0 to J
    for k = 0 to K
      for l = 0 to L
        read A[i, k+l], B[i, j]
```

Listing 1. A 4-level loop nest of data accessing.

The above example shows a 4-level loop nest accessing two 2D tensors. The size of loop nest is $IJKL$ while the two tensor

shapes are $I(K+L)$ and IJ. The reuse volume *#reuse* is the total cycle number in the loop nest divided by the total unique tensor indexes, which is $\frac{JKL}{K+L}$ for tensor A and KL for tensor B. There are two sources of on-chip memory reuse. The first one is an irrelevant loop. Take tensor A as an example. Loop J is irrelevant for tensor A. When loop J index changes, the index of tensor A remains same so the data is reused. The second one is linear dependent loops. Loop K and L are linearly dependent for tensor A and it only accesses $(K+L)$ unique value in KL cycles.

Increasing the loop range J,K,L can improve the reuse volume, but also increase the requirement of on-chip memory size. In the above example, the required memory size is $I(K+L)+IJ$. The loop range should be carefully selected in order to improve the reuse volume while complying with the memory resource limitation.

B. Memory Bandwidth Requirement

The bandwidth requirement B_{req} can be modeled as the total amount of off-chip memory accessing for each tensor divided by execution time. The amount of off-chip memory accessing is the same as the size of on-chip memory for all memory banks(every element is from off-chip initially). However, the number of memory banks for each tensor could be different. The total access amount should be multiplied by the number of banks. According to Table I, the bank number is related to the reuse volume inside the PE array.

$$mem\_req = \sum \#bank \times mem\_size\_per\_bank \tag{4}$$

$$exec\_time = \frac{total\_cycle}{frequency} \tag{5}$$

$$B_{req} = \frac{mem\_req}{exec\_time} \tag{6}$$

With the memory analysis we can find the size of off-chip memory access during the execution period. When B_{req} is bigger than B_{res}, the memory bandwidth is not enough and the PE execution will be stalled due to insufficient data.

$$P_{mem} = min(1, \frac{B_{res}}{B_{req}}) \tag{7}$$

C. Frequency Analysis

Here, we also discuss the factors that influence the frequency performance on FPGAs, which is a critical issue of FPGA accelerator design.

Multicast dataflows require a large fan-out structure, while systolic dataflows require a more complex controller inside each PE, and the reduction tree requires unique connection patterns. The different interconnection patterns lead to both energy and frequency variation between dataflows. The upper bound of PE array size is determined by the DSP resource of FPGA. In the real situation, the DSP utilization cannot reach 100% because large DSP utilization can cause routing congestion, which severely decreases the frequency. As a result, there's a trade-off between utilization and frequency. The size of on-chip BRAM is also limited by the BRAM resource amount. Larger memory might provide more reuse opportunities, which reduces the off-chip memory access, but it may also influence frequency.

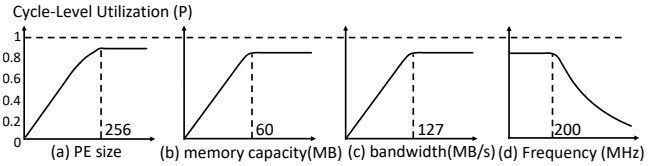


Fig. 6. Normalized performance of hardware parameters

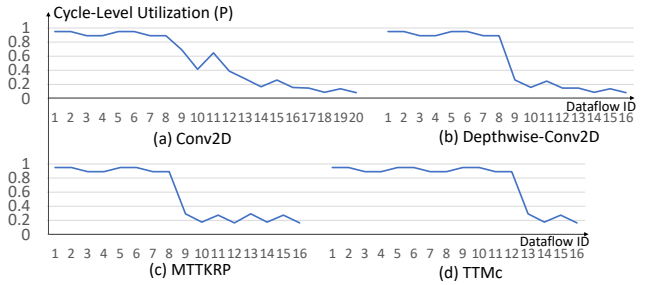


Fig. 7. Normalized performance of different dataflows for tensor applications

V. Experimental Results

A. Experiment Setup

We use Xilinx Virtex Ultrascale+ VU9P FPGA to evaluate the performance of different spatial accelerator designs. VU9P contains 2586K logic cells, 6840 DSPs and 75.9MB block RAMs. The memory bandwidth is 32GB/s. We use Chisel [23] hardware construction language to generate the hardware RTL code of different designs, and synthesize it with Vivado 2020.1 software. The performance evaluation is conducted with a performance model and validated with hardware simulation, while the frequency results are collected from on-board profiling. We evaluate typical tensor algorithms including Conv2D, Depthwise-Conv2D, MTTKRP and TTMc.

B. Cycle-level Performance Evaluation

We compute the cycle-level performance of different design choices with our performance model. The model estimates the performance with the accelerator design choice and predefined hardware parameters: PE size, on-chip memory capacity, memory bandwidth and frequency. First, we evaluate the performance of a systolic array dataflow for Conv2D with different hardware parameters. Next, we fix the hardware parameters and evaluate the performance of different dataflows for each application. We set PE size to 16×16.

Figure 6 shows the performance of the same dataflow under different hardware parameters. We fix the rest three parameters and evaluate the performance when the selected parameter grows based on the performance model. When the PE size $\#PE$ increases, $\#bank$ is proportional to $\sqrt{\#PE}$ and $\#reuse$ is inverse proportional to $\sqrt{\#PE}$, so B_{req} is proportional to $\#PE$. When exceeding bandwidth limitation, the P_{mem} neutralizes the growth of PE number. The performance impact of other factors can be inferred similarly.

Figure 7 shows the impact of workload mapping to hardware utilization measured by P. We find that when PE array size is fixed, the major factor that influences utilization is memory

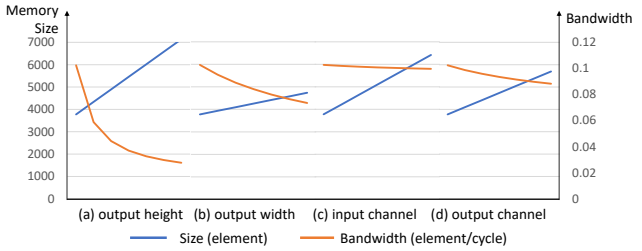


Fig. 8. Memory requirement of different memory configurations

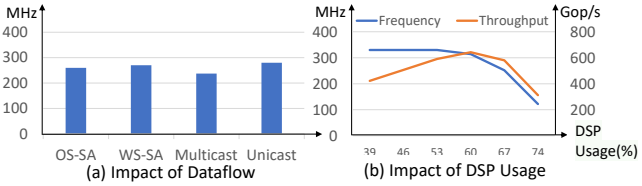


Fig. 9. Frequency and throughput result

bandwidth. Some dataflows can achieve similar high performance because the bandwidth is sufficient. Other dataflows have poor performance since the tensor cannot be reused in PE array with the particular mapping, corresponding to unicast dataflow.

C. Memory Requirement Evaluation

We use the Conv2D benchmark with kernel size 3 to evaluate different memory arrangement schemes. The memory size and layout are determined by the loop range mapped to BRAM-level. We evaluate the memory size and bandwidth requirement of each case. The loop range of input channel, output channel, output width, output height is set to 16, 16, 16, 1, respectively. We increase one of the loop range while keep others fixed. The memory size and bandwidth requirement are shown in Figure 8.

The result shows that when each of the loop range increases, the on-chip memory requirement increases and the bandwidth requirement decreases. However, the changing rate is different for each case. In this example, increasing output height has the best performance on reducing memory bandwidth since it goes from 0.1 to 0.03. However, increasing the input channel has a little improvement since the memory bandwidth only decreases to 0.09. The result guides the selection of loop range that maps to the BRAM-level memory to satisfy both memory size and bandwidth requirement.

D. Frequency and Throughput Evaluation

Figure 9 shows the impact of PE size and PE interconnection to the performance of spatial accelerator on FPGAs. The X-axis is PE dataflow and DSP utilization while Y-axis is frequency and throughput. When the size of PE is small, the accelerator only consumes a small portion of FPGA's DSP and logic resource, which delivers high frequency. When the size of PE becomes large, the routing congestion problem exists which hurts the frequency. Result shows that the PE array throughput can reach the peak performance of 643Gop/s when DSP utilization is 60%. PE array dataflow also influences the frequency. Systolic (OS-SA and WS-SA) and Unicast dataflows achieve higher frequency than multicast dataflow because multicast dataflow has a large fan-out structure from on-chip memory to PEs. However, unicast dataflow suffers from memory bandwidth problems so it's not as efficient as systolic dataflows.

VI. Conclusion

In this paper, we present a comprehensive analysis on the hardware design space of spatial accelerator for tensor algebras. The design space includes PE algorithm, workload distribution, PE array dataflow, clustering and memory hierarchy. We analyze the impact of hardware design choice as well as the hardware parameters on the performance of hardware accelerator. We find that different solutions in the design space have different design trade-offs, and hardware parameters also influence performance in a certain way.

Acknowledgement

This project is partially supported by National Key R&D Program of China (2020AAA0105200) and Key-Area Research and Development Program of Guangdong Province (No. 2019B010155002).

References

[1] X. Wei *et al.*, "Automated systolic array architecture synthesis for high throughput CNN inference on fpgas," in *DAC*, 2017.
[2] J. Cong and J. Wang, "Polysa: polyhedral-based systolic array auto-compilation," in *ICCAD 2018*.
[3] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *ISCA*, 2017.
[4] Y. Chen *et al.*, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *ISCA 2016*.
[5] H. Sharma *et al.*, "From high-level deep neural models to fpgas," in *MICRO 2016*.
[6] Q. Xiao *et al.*, "Fcnnlib: an efficient and flexible convolution algorithm library on fpgas," in *DAC*, 2020.
[7] X. Wei *et al.*, "Overcoming data transfer bottlenecks in fpga-based DNN accelerators via layer conscious memory management," in *DAC*, 2019.
[8] S. Wang *et al.*, "Flexcl: An analytical performance model for opencl workloads on flexible fpgas," in *DAC*, 2017.
[9] X. Wei *et al.*, "TGPA: tile-grained pipeline architecture for low latency CNN inference," in *ICCAD*, 2018.
[10] A. Parashar *et al.*, "Timeloop: A systematic approach to DNN accelerator evaluation," in *ISPASS*, 2019.
[11] X. Yang *et al.*, "Interstellar: Using halide's scheduling language to analyze DNN accelerators," in *ASPLOS 2020*.
[12] H. Kwon *et al.*, "Understanding reuse, performance, and hardware cost of DNN dataflow: A data-centric approach," in *MICRO 2019*.
[13] S. Dave *et al.*, "dmazerunner: Executing perfectly nested loops on dataflow accelerators," *ACM Trans. Embedded Comput. Syst.*, 2019.
[14] N. Srivastava *et al.*, "T2s-tensor: Productively generating high-performance spatial hardware for dense tensor computations," in *FCCM*, 2019.
[15] J. Wang *et al.*, "Autosa: A polyhedral compiler for high-performance systolic arrays on fpga," in *FPGA*, 2021.
[16] Y. Lai *et al.*, "Susy: A programming model for productive construction of high-performance systolic arrays on fpgas," in *ICCAD 2020*.
[17] H. Genc *et al.*, "Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures," *arXiv*, 2019.
[18] L. Jia *et al.*, "Tensorlib: A spatial accelerator generation framework for tensor algebra," in *DAC*, 2021.
[19] L. Lu *et al.*, "Tenet: A framework for modeling tensor dataflow based on relation-centric notation," in *ISCA*, 2021.
[20] R. Venkatesan *et al.*, "Magnet: A modular accelerator generator for neural networks," in *ICCAD*, 2019.
[21] Q. Xiao *et al.*, "Hasco: Towards agile hardware and software co-design for tensor computation," in *ISCA*, 2021.
[22] L. Jia *et al.*, "Generating systolic array accelerators with reusable blocks," 2020.
[23] J. Bachrach *et al.*, "Chisel: constructing hardware in a scala embedded language," in *DAC*, 2012.