EMS: Efficient Memory Subsystem Synthesis for Spatial Accelerators

Liancheng Jia¹, Yuyue Wang², Jingwen Leng³, Yun Liang¹†
¹Peking University, ²University of California, Los Angeles ³Shanghai Jiao Tong University
Email: ¹{jlc.ericlyun}@pku.edu.cn ²yuyue@cs.ucla.edu ³leng-jw@stju.edu.cn

ABSTRACT
Spatial accelerators provide massive parallelism with an array of homogeneous PEs, and enable efficient data reuse with PE array dataflow and on-chip memory. Many previous works have studied the dataflow architecture of spatial accelerators, including performance analysis and automatic generation. However, existing accelerator generators fail to exploit the entire memory-level reuse opportunities, and generate suboptimal designs with data duplication and inefficient interconnection.

In this paper, we propose EMS, an efficient memory subsystem synthesis and optimization framework for spatial accelerators. We first use space-time transformation (STT) to analyze both PE-level and memory-level data reuse. Based on the reuse analysis, we develop an algorithm to automatically generate data layout of the multi-banked scratchpad memory, data mapping, and access controller for the memory. Our generated memory subsystem supports multiple PE-memory interconnection topologies including direct, multicast, and rotated connection. The memory and interconnection generation approach can efficiently utilize the memory-level reuse to avoid duplicated data storage with low hardware cost. EMS can automatically synthesize tensor algebra to hardware designed in Chisel. Experiments show that our proposed memory generator reduces the on-chip memory size by an average of 28% than the state-of-the-art, and achieves comparable hardware performance.

KEYWORDS
spatial accelerators, memory, space-time transformation

1 INTRODUCTION
A wide range of computer applications are increasingly relying on spatial accelerators for hardware specialization on different platforms including ASIC, FPGA, CGRA and other embedded devices [2, 7]. Most spatial accelerators are mainly composed of an array of homogeneous PEs connected with memory and network-on-chips (NoC). The spatial accelerators can provide huge parallelism with a massive number of PEs (processing elements) and high frequency. A large number of spatial accelerators have been proposed, and they can employ various hardware dataflows including systolic array [7, 16, 17], multicast network [5, 18], and tree-based structures [8].

The on-chip memory is a fundamental component of spatial accelerators, which is usually implemented with multi-banked scratchpad memory (SPM) and connected with the PE array with PE-memory interconnections. SPM design has a large design space, including layout, data mapping, bank partitioning, and the interconnection network between other modules. For example, Eyeriss [2] and ShiDianNao [5] use multicast interconnection topology, [14] uses rotation buffers between memory banks and PE array, and MAERI [8] uses tree-based interconnection. The design of memory subsystem is complex, and can significantly impact the accelerator performance and power/area consumption.

Many previous works [3, 16, 17] use the polyhedral model to analyze tensor dataflows and high-level synthesis (HLS) to generate hardware architectures. [6, 10, 11, 15] create domain-specific languages (DSL) and compilers to generate hardware with high-level language. However, they mainly focus on the analysis of the architecture of PE array and their internal interconnection, and provide limited analysis and optimization of SPM architecture and the memory-related interconnection. For example, Tensorlib [6] uses space-time transformation analysis to generate PE array without the complete memory architecture. AutoSA [16] and Susy [11] support automatic generation of the memory hierarchy, but cannot fully explore SPM-level reuse and result in unnecessary data duplication. Analysis frameworks such as [4, 9, 12, 13] only discuss memory-level data reuse based on storage hierarchy without actual hardware synthesis and implementation.

To overcome these limitations, we propose EMS, a memory subsystem synthesis and optimization framework for spatial accelerators. We first propose data reuse analysis for on-chip memory with the consideration of multiple memory banks. Space-time transformation (STT) was initially used for PE array dataflow analysis [6, 11]. We use STT to analyze both PE-level and memory-level data reuse. For the multi-bank SPM architecture, the memory-level data reuse is further classified into intra-bank and inter-bank reuse. This analysis provides useful guidance to the implementation of memory layout and PE-memory interconnection.

Based on the STT reuse analysis, EMS can automatically generate two key components of the memory subsystem: the multi-bank SPM module, and the PE-memory interconnection network. For the SPM module, we develop an algorithm to build the SPM layout, the data mapping, and the data access controller. The algorithm generates...
an efficient memory layout by merging intra-bank reusable data. For PE-memory interconnection, unlike previous works [11, 15, 16] which only support direct and multicast interconnection, EMS supports three types of PE-memory interconnection network including direct, multicast, and rotation networks. The rotation network can support inter-bank data reuse with low hardware costs. EMS uses the tensor algebra definition and STT dataflow mapping as inputs to generate the memory subsystem designed in Chisel [1] hardware language, and can be integrated with PE array architectures to build complete spatial accelerators.

In summary, the contribution of this paper are as follows:

• We propose EMS, an efficient memory subsystem synthesis and optimization framework for spatial accelerators.
• We propose space-time transformation based analysis to identify different types of memory reuse patterns.
• We propose an SPM architecture and PE-memory interconnection generation algorithm to efficiently implement the data reuse at memory level.

Experiment shows that our proposed memory subsystem synthesis can generate efficient on-chip memory and network for different tensor algorithms. It uses 28% less memory space on average for various 2D convolution workloads compared with the state-of-the-art AutoSA generator [16], and achieves comparable hardware performance with other accelerator generators.

2 BACKGROUND

Figure 1 (a) shows a typical hierarchy of spatial accelerators. It contains three components: memory banks, PE-memory interconnection network, and PE array. Data from upper-level hardware (usually off-chip memory) is distributed to each memory bank. Data communication is implemented with PE-memory networks and PE array networks [2, 13, 19].

Space-time transformation (STT) based techniques have been used to model spatial hardware dataflows [6, 11, 16]. It applies linear transformation to map computation loop indexes to both spatial and temporal indexes of the spatial accelerator. Given the

![Figure 1: Dataflow analysis with space-time transformation](image)

Figure 2: The Overview of EMS Framework

loop iteration index $\mathbf{x} = [i, j, \ldots]^T$ and a transformation matrix $T$, the space and time index $\mathbf{y}$ can be calculated as follows,

$$\mathbf{y} = T \mathbf{x}$$  \hspace{1cm} (1)

The space-time index $\mathbf{y}$ can be represented as $[\mathbf{y}]$. The space part $\mathbf{y}$ means the PE array coordinates and time part $t$ is the time stamp of execution, which can be multi-dimensional. Figure 1(b) shows an example of STT for matrix multiplication $C[i, j] = A[i, k] \times B[k, j]$. The iteration index $[0, 2, 1]^T$ is transformed to space-time index $[0, 2, 3]^T$ by multiplying with the STT matrix $T$, which indicates that the instance occurs at $PE[0, 2]$ and cycle 3.

The tensor data index $I$ accessed at iteration $\mathbf{x}$ can be represented with affine address reference $I = X \mathbf{x}$, where $X$ is the tensor access matrix. As shown in Figure 1 (c), we can build the relationship between the space-time vector and the tensor index to model the PE array dataflow based on STT and the tensor access matrix. By replacing the $\mathbf{x}$ with $T^{-1}\mathbf{y}$ using Equation (1), we have:

$$AT^{-1}\mathbf{y} = I$$  \hspace{1cm} (2)

With space-time transformation, we can find all the space-time vectors $\mathbf{y}$ that reuse the same tensor index by solving equation (2). For example, in Figure 1 (d), the same tensor element $A[0, 1]$ appears in three space-time indexes (marked as yellow blocks). In the PE array hardware, every PE sends its data to its neighbor PE after delaying one cycle, which forms a systolic dataflow. Similarly, the dataflow of other tensors can be inferred with different tensor access matrixes and the same STT matrix. In practice, the value of elements in STT matrix is restricted to 1, 0 and -1 and the matrix must be full-ranked for an valid and efficient mapping.

Prior studies [3, 6] have developed analysis tools for dataflows of PE arrays, but lack the ability to model the reuse opportunities in the memory level, especially when considering various design alternatives of the memory subsystem. In this paper, we develop an automatic memory subsystem synthesis tool for spatial accelerators to exploit memory-level data reuse.

3 OVERVIEW OF EMS

Figure 2 shows the overall working flow of EMS framework. The input of EMS framework is the definition of the tensor algebra specified with perfectly nested loops, and the hardware mapping definition specified with space-time transformation (STT) matrix.
EMS first extracts the access matrix of each tensor with the algorithm definition. Next, EMS performs the data reuse analysis based on space-time transformation. EMS identifies three types of reuse: dataflow-based reuse, intra-bank reuse, and inter-bank reuse. They depict different hardware-level implementation methods. Details about the analysis algorithm are shown in Section 4.

Next, EMS builds the memory subsystem architecture based on the type of data reuse vectors. The dataflow-based reuse can be handled by existing works [6, 11, 16], so EMS mainly deals with the latter two. To implement the intra-bank reuse, EMS develops an algorithm to generate the layout and data mapping of SPM, and also builds the address generator for the PE array to access the SPM. EMS supports three types of PE-memory interconnection including direct, multicast, and rotation. For inter-bank reuse, EMS can build a rotation buffer between PE array and memory to support data sharing across different banks. Details are in section 5.

Finally, EMS framework combines the generated memory subsystem with PE array modules generated by existing works [6] to build the complete architecture of spatial accelerators. EMS framework is written in Chisel hardware construction language, and the RTL code can be generated with Chisel compiler for synthesis.

4 DATA REUSE ANALYSIS

To perform reuse analysis, the loop nest of the original program needs to be tiled and transformed to the space-time space with multi-dimensional time. Based on the space and time direction of the data reuse, we classify the reuse patterns into three types. EMS identifies each type of data reuse with STT analysis.

We define the reuse vector as a vector in the space-time space. When space-time index \( \vec{s} \) shifts on the direction of reuse vector \( \vec{x} \), the data index to access doesn’t change, so reuse vector \( \vec{x} \) satisfy \( AT^{-1}\vec{s} = \vec{0} \), as shown in Equation (2). There can be multiple linearly independent reuse vectors, and each reuse vector is a basis vector of \( AT^{-1}\)'s kernel space. The tensor algebra involves multiple operands and they have different tensor access matrices, so their reuse space and reuse vectors are also different.

The reuse vector is represented as \( \vec{p} / \vec{t} \), where \( \vec{p} \) has N PE index dimensions, and \( \vec{t} \) has multiple time dimensions. Each reuse vector can be classified into three types based on whether their spatial part \( \vec{p} \) and discontinuous temporal part \( t_2...t_n \) are zero (Z) or non-zero (NZ), as shown in Table 1. The innermost time \( t_1 \) can be either Z or NZ, but the entire vector cannot be a zero vector. The example column shows a 5D vector with 2D space and 3D time.

(1) **Dataflow reuse vector.** The non-zero value is restricted in the PE index and the innermost time dimension. The discontinuous time dimensions \( t_2...t_n \) are zeroes. This kind of data reuse can be exploited by the PE array dataflow.

(2) **Intra-bank reuse vector.** The non-zero value is restricted in the time dimensions but not PE index dimension. Each PE repeatedly accesses the same data element at different time.

(3) **Inter-bank reuse vector.** The reuse vector contains both non-zero spatial part and discontinuous temporal part.

The three types of reuse vectors imply different hardware costs. The dataflow reuse vector has the least hardware cost, since the memory is only accessed once, and communication is implemented with interconnection between PEs. The intra-bank reuse has greater hardware cost with multiple accesses of the on-chip memory. The inter-bank reuse has the highest hardware cost since data from different SPM banks cannot communicate directly. Algorithm 1 shows the reuse vector generation algorithm. We first calculate the entire data reuse subspace \( \text{Ker}(AT^{-1}) \) (\( \text{Ker} \) refers to kernel space). When extracting basis vectors from the reuse subspace, we try to select the basis vector with smaller hardware cost first, and switch to the next level when no new reuse vector can be found at the current level. Afterwards we extract the basis vectors from each of the three subspaces. The values of each reuse vector should be relatively prime.

5 GENERATING MEMORY ARCHITECTURE

In this section, we show how EMS builds the SPM architecture with intra-bank reuse vectors and generate different PE-memory interconnection networks with dataflow and inter-bank reuse.

5.1 SPM Architecture Generation

The entire SPM is composed of multiple SPM banks. We assume that the SPM bank is a two-port SRAM with one read port and one write port. Each SPM bank can communicate data with one PE at every cycle, and the other port is used to communicate with low-level storage. Each bank directly connects to one PE in the PE array, but only some of the PEs directly connect to the memory banks, while other PEs transfer data with PE array interconnection determined by dataflow reuse vectors.

We use the intra-bank reuse vectors to build the SPM architecture by reorganizing the memory layout. Algorithm 2 shows the SPM layout and address controller generation process for each SPM bank. The layout can be described with a multi-dimensional tensor which stores a partition of the original tensor accessed by a particular PE across all dimensions. The access controller is represented with a matrix that maps the time indexes to memory indexes.

Firstly, the shape of SPM is initialized to the same shape as time dimensions. This initial layout doesn’t consider data reuse and thus data can be duplicated at different memory addresses. Next, we use the intra-bank reuse vectors to eliminate duplicated entries in each bank. For a reuse vector \( V \) and any timestamp \( S \), we have:

\[
\forall l, \text{Addr}(S) = \text{Addr}(S + \lambda V)
\]

Table 1: Type of Reuse Vectors.

<table>
<thead>
<tr>
<th>Type</th>
<th>&amp;p</th>
<th>( t_1 )</th>
<th>( t_2...t_n )</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataflow</td>
<td>Z or NZ</td>
<td>Z or NZ</td>
<td>Z</td>
<td>[0, 1, 1, 0, 0]</td>
</tr>
<tr>
<td>Intra-bank</td>
<td>Z</td>
<td>Z or NZ</td>
<td>NZ</td>
<td>[0, 0, 1, 0, 1]</td>
</tr>
<tr>
<td>Inter-bank</td>
<td>NZ</td>
<td>Z or NZ</td>
<td>NZ</td>
<td>[1, 0, 0, 1, 0]</td>
</tr>
</tbody>
</table>

Algorithm 1: Reuse Vector Generation Algorithm

**input**: STT Matrix \( T \), Access Matrix \( A \)

**output**: Dataflow, Intra-bank, Inter-bank Reuse Vectors \( V_D, V_M, V_C \)

1. Generate the entire reuse subspace \( S = \text{Ker}(AT^{-1}) \);
2. Generate the dataflow subspace \( S_D = S \cap \{ vi \geq 2, t_i = 0 \} \)
3. Generate the intra-bank subspace \( S_M = (S - S_D) \cap \{ \vec{p} = 0 \} \)
4. Generate the inter-bank subspace \( S_C = S - S_D - S_M \)
5. **return** The basis vectors of each subspace \( V_D = \text{Basis}(S_D), V_M = \text{Basis}(S_M), V_C = \text{Basis}(S_C) \)
Algorithm 2: SPM generation algorithm

input : Range of each time dimensions \( R_T \)
input : Intra-bank Reuse Vectors \( V \)
output: Range of each memory dimensions \( R_M \)
output: Access matrix of SPM \( A \)
1: initialize the access matrix as identity matrix \( A = I_n \)
2: initialize the memory dimension \( R_M = R_T \)
for each intra-bank reuse vector \( V \) do
4: find the index \( p \) in \( V \) where \( V_p \) is 1 or -1.
5: for each time dimension \( i \) other than \( p \) do
6: update the access matrix \( A[i, p] = A[i, p] - \frac{V_p}{V_p} \)
7: update the i-th memory range \( R_M[i] \)
8: end for
9: remove the p-th row of matrix \( A \).
10: end for
11: return Access matrix \( A \), Memory range of each dimension \( R_M \)

The address accessed at time \( S \) can be replaced with \( S + \lambda V \) for any \( \lambda \). We find an index \( p \) from \( V \) where \( V_p \) is 1 or -1. For any \( S \), we can find a \( \lambda \) so that the address index at \( p \)-th dimension is zero at time \( S \). Equation 4 gives the value of \( \lambda \).

\[
Addr(S + \lambda V)_p = 0, \lambda = \frac{S_p}{V_p}
\]  

(4)

For any timestamp \( S \), it can find an entry in the SPM whose index at \( p \)-th dimension is always zero. In this way, the \( p \)-th memory dimension can be removed from the SPM layout, and the SPM address accessed at time \( S \) becomes \( Addr(S - \frac{S_p}{V_p} V) \). For each reuse vector, one of the dimensions is removed, and the final memory layout is generated when all memory reuse vectors have been processed. The range of each remaining dimension is also updated with the \( p \)-th dimension range.

EMS can also support common memory optimization techniques like tiling and double buffering. To enable loop tiling, the user should manually split the tiled loop into two sub-loops and reschedule the computation with space-time mapping. The user can specify the boundary of memory layout, and EMS will generate the memory layout from the innermost time dimension to the boundary dimension. For double buffering, the user can simply tile a loop with a factor of 2, and set the loop of length 2 as the boundary dimension. EMS will generate the SPM with the outermost dimension of length 2, and repeatedly access the ping-pong buffers.

5.2 PE-Memory Interconnection

Figure 3 (a) and (b) show the existing PE-memory interconnection used in previous works. (a) is the direct interconnection where each bank only communicates with one PE. (b) shows the multicast interconnection for dataflow-based reuse. The memory communicates data directly with one PE, and the PE transfers data to other PEs with PE array dataflow such as systolic or multicast. However, existing solutions cannot support inter-bank reuse and require duplicated storage in different banks, since data is loaded to different PEs at incontinuous time stamps. To efficiently implement the inter-bank reuse, EMS proposes the rotation buffer structure for PE-memory interconnection, which enables cross-bank data communication with a fixed direction, as shown in Figure 3 (c).

Figure 3: The PE-memory interconnection types

(a) Direct interconnection
(b) Multicast interconnection
(c) Rotation buffer

Figure 4: Comparison of different interconnections

To use the rotation buffer, the inter-bank reuse vector is first split into spatial part \( \vec{p} \) and temporal part \( t \). We apply the algorithm in section 5 on \( t \) to generate the memory layout. Afterwards, the duplication is eliminated but the required data may reside in another bank. The distance between PE index and the bank index storing its data is the multiply of \( \vec{p} \). The i-th PE gets its required data from the same bank at index \( (i + R) \mod N_{bank} \). The shift distance \( R \) is the same for every PE, so the PE can get its required data by rotating the bank output for length \( R \). In the rotation buffer, the data from each bank are packed together, and can be rotated for an arbitrary length before sent to PEs. In Figure 3 (c), the data in each bank is \((1,2,3,4,5)\). It right-rotates for \( R = 2 \) words and sends \((4,5,1,2,3)\) to the PEs. The rotation distance can be different at each time stamp.

The rotation buffer can also be used in dataflow reuse vector whose spatial and temporal parts are both non-zero. Figure 4 shows a 1-D convolution case. Rotation buffer can achieve the best performance and PE utilization among the three interconnections. Multicast interconnect (a) sends data from one memory bank to multiple PEs in a systolic way. It takes a long time to fill and drain the pipeline, and only starts computation when the farthest PE receives data, which may cause great performance degradation. Direct interconnection (b) stores duplicated data in multiple banks, wasting a large number of memory resources. For example, data \([0,2]\) is stored in all three banks. By contrast, rotation buffer can have much better performance and PE utilization. Figure 4 (c) gives an example, at cycle \([0,1]\), data \([0,3]\), \([0,1]\), \([0,2]\) are read from multiple banks in parallel, then simultaneously left-rotate for one word length and enter PE array. It removes data duplication and brings high utilization at the cost of memory bank number increment and more complex interconnection modules.
5.3 Case Study: 2D Convolution

We explain the memory synthesis using 2D convolution as an example. 2D convolution is defined as a 6-level loop nest in Figure 5 (a). We only discuss the generation of tensor $A$ (input feature map). The loop nest is mapped to a 2D PE array and 4-dimensional time stamps. We apply two different STT and generate two PE-memory architectures in (b) and (c), corresponding to the weight-stationary (WS) and output-stationary (OS) dataflows.

The dataflow reuse vector $[1, 0, 1, 0, 0, 0]$ is same for the two designs. It shows a vertical systolic dataflow. Each memory bank connects to one PE in the first row. For the WS design, the shape of PE array is $K \times C$. The two intra-bank reuse vectors remove two memory dimensions $P, Q$ from the original memory layout, and the range of the remaining two memory dimensions become $X + Q$ and $Y + P$. At time step $(x, y, q, p)$, memory index $[y+q, x+q]$ is accessed. For the OS design, the shape of PE array is $K \times X$ and there is only one intra-bank reuse vector. The temporal part of inter-bank reuse vector $[0, 1, 0, 0]$ is first treated as intra-bank reuse, and generates the memory layout of $(Y + P) \times X$. The data from each bank are rotated in the rotation buffer before entering PEs. Since the size of $X$ dimension is larger than PE size, the final layout further multiplies the number of tiles in $X$ dimension $tile_X$.

6 EXPERIMENTAL EVALUATION

6.1 Experiment Setup

Given a tensor algebra formula and the dataflow specified by STT matrix, EMS generates the memory bank architecture and the interconnection with Chisel [1] hardware implementation. We use Tensorlib [6] to generate the PE array, and integrate with EMS to build a complete spatial accelerator architecture. The generated Chisel code is compiled to Verilog and then synthesized on both ASIC and FPGA platforms. For ASIC evaluation, we use Synopsys DC compiler with 55nm UMC 1P8F technology. For FPGA evaluation, we use Xilinx VU9P with 6840 DSPs and 2160 BRAMs and Xilinx Vivado 2020.1 software.

We first evaluate the size of our generated memory module for different tensor workloads and STT mappings, and compare with the state-of-the-art. Then, we compare the hardware power and area for different PE-memory interconnection types. Finally, we compare the overall performance on FPGA with prior works.

6.2 Memory Size and Performance Comparison

We evaluate the on-chip memory size generated by EMS as well as peak PE utilization with benchmarks in Table 2 including different Conv2D variants, MTTKRP, and Jacobii2D, and compare the results with the state-of-the-art generator AutoSA [16] on Conv2D benchmarks. The PE size is set to $32 \times 32$.

For Conv2D, we evaluate early and late layers as well as strided and depthwise layers. EMS generates three mappings labeled by the loops mapped to the two dimensions of PE array. KC mapping is applied in TPU [7]. KX mapping is applied in [17], and XY mapping is applied in ShiDiannao [5]. The memory size is normalized with the optimal size without any duplication.

Figure 6 shows the size of on-chip memory generated by EMS and PE utilization result for different benchmarks and different space-time mappings. Based on the utilization, we find that each Conv2D layer favors different mappings. (a, c, d) favor KX-mapping with small channel size and large feature map size. (b) is the opposite and favors KC-mapping. In typical convolution networks, earlier layers have larger feature map size but smaller channel size, so KX and XY-mapping performs better. Later layers have larger channel size but smaller feature map size, so KC-mapping performs better. For KX and XY mapping, rotation buffers can be applied to optimize the memory size since inter-bank reuse is available for input data. The overall memory size can be reduced by 15% to 64%. KC mapping and MTTKRP benchmark don’t involve inter-bank reuse. The XY mappings of Conv2D and Jacobii2D have both vertical and horizontal inter-bank reuse, but the rotation buffer can only work in one dimension, so EMS still generates suboptimal memory layout with duplication.

We compare with the state-of-the-art systolic array generator AutoSA [16] for four Conv2D benchmarks in Figure 7 labeled as (a) to (d) and show the average result. AutoSA and EMS can generate multiple solutions and we select the mapping with the best PE utilization for comparison. EMS outperforms AutoSA in three layers with 15% to 44% memory size reduction and generates the same result as AutoSA for the (b) layer. The average size reduction is 28%. Since benchmark (a, c, d) favor KX mapping, rotation buffer can be used in EMS to eliminate inter-bank data duplication. For the layer that favors KC mapping without inter-bank reuse, EMS...
and AutoSA have the same performance. Furthermore, EMS is able to generate non-systolic designs such as Jacobi-2D, which is not supported by AutoSA.

### 6.3 Hardware Cost Comparison

Next, we study the hardware cost of rotation buffer design. We use Conv2D-KX as benchmark which uses rotation buffer for input data interconnection, and compare the power and area of different memory subsystem designs. EMS can generate the memory with rotation buffer and remove duplication (Rotated w/o Dup.), or without rotation buffer but requires duplication (Direct w/ Dup.). The two designs are compared with the optimal case (Direct w/o Dup.). With the rotation buffer, the SRAM size decreases from 168 KB to 128 KB. Figure 8 shows the power and area of SRAM, interconnection and access controller modules. The rotation buffer introduces 1.6X overhead in the interconnection module, but the memory size reduction compensates for the overhead, which brings 13% and 28% optimization on power and area.

### 6.4 Overall FPGA Performance Evaluation

We evaluate the FPGA performance of the complete accelerator and compare with other works on the Conv2D workload. The result is shown in Table 3. Compared with [6], the LUT and DSP resources of EMS-WS only slightly increases, which means that the memory accessing controller of EMS has very little hardware overhead. After adding the rotation buffer in EMS-OS, the LUT resource increases from 76% to 83%. We apply the similar frequency boosting technique applied in [16], and achieve 301 MHz for WS and 295 MHz for OS.

### 7 CONCLUSION

In this paper, we propose EMS, an efficient memory subsystem synthesis framework for spatial accelerators. We first use space-time transformation (STT) to analyze both dataflow and memory-level data reuse. Based on STT, we extract different level of data reuse inside memory, and develop an algorithm to generate the SPM layout and PE-memory interconnections. Our proposed generator can be integrated to existing dataflow-based accelerators. Experiment shows that our proposed generator requires 28% smaller memory size than previous works to process the same tensor algebra workload on average, and achieves comparable hardware performance.

### ACKNOWLEDGMENTS

This work was supported by Key-Area Research and Development Program of Guangdong Province (No. 2019B010155002).

### REFERENCES