FlexTensor: An Automatic Schedule Exploration and Optimization Framework for Tensor Computation on Heterogeneous System

Size Zheng, Yun Liang, Shuo Wang, Renze Chen, Kaiwen Sheng

25th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2020

Presenter: Seok-Joong Kim

https://sites.google.com/view/seokjoongkim

Jun 9, 2020

Neural Network Acceleration Study Season #2

Contents of presentation

- **Introduction**
- **Key contribution**
  - Use reinforcement learning to generate schedule
- **Experiment**
- **Conclusion**

Introduction

- **Tensor Computation & Schedules**

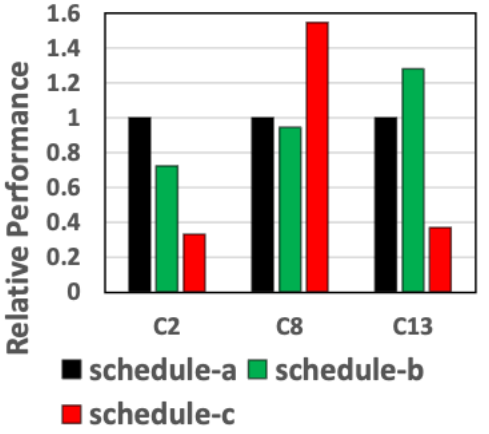
Table 1. Definitions of different tensor computations

Operator	Definition	Description
GEMV	$O_i = A_{i,k} \circ B_k$	Matrix-vector multiply
GEMM	$O_{i,j} = A_{i,k} \circ B_{k,j}$	Matrix-matrix multiply
Bilinear	$O_{i,j} = A_{i,k} \circ B_{j,k,l} \circ C_{i,l}$	Bilinear transformation
1D convolution	$O_{b,k,i} = I_{b,rc,i+rx} \circ W_{k,rc,rx}$	1D sliding window convolution
Transposed 1D convolution	$O_{b,k,i} = I_{b,rc,i+rx} \circ W_{rc,k,L-rx-1}$	Transposed convolution for 1D array
2D convolution	$O_{b,k,i,j} = I_{b,rc,i+rx,j+ry} \circ W_{k,rc,rx,ry}$	2D sliding window convolution
Transposed 2D convolution	$O_{b,k,i,j} = I_{b,rc,i+rx,j+ry} \circ W_{rc,k,X-rx-1,Y-ry-1}$	Transposed convolution for 2D matrix
3D convolution	$O_{b,k,d,i,j} = I_{b,rc,d+rd,i+rx,j+ry} \circ W_{k,rc,rd,rx,ry}$	3D sliding window convolution
Transposed 3D convolution	$O_{b,k,d,i,j} = I_{b,rc,d+rd,i+rx,j+ry} \circ W_{rc,k,D-rd-1,X-rx-1,Y-ry-1}$	Transposed convolution for 3D cube
Group convolution	$O^g_{b,k,i,j} = I^g_{b,rc,i+rx,j+ry} \circ W^g_{k,rc,rx,ry}$	2D convolution separated into groups
Depthwise convolution	$O_{b,k,i,j} = I_{b,c,i+rx,j+ry} \circ W^c_{k,rx,ry}$	2D convolution separated by channels
Dilated convolution	$O_{b,k,i,j} = I_{b,rc,i+rx\times dx,j+ry\times dy} \circ W_{k,rc,rx,ry}$	2D convolution with kernel dilation

[1] https://www.synopsys.com/designware-ip/technical-bulletin/building-efficient-deep-learning-dwtb_q318.html

Introduction

- **Motivation**



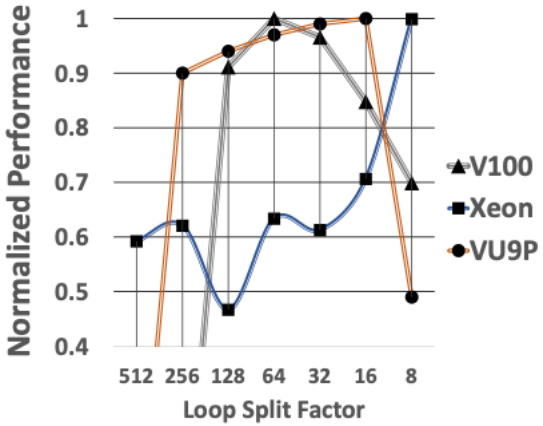


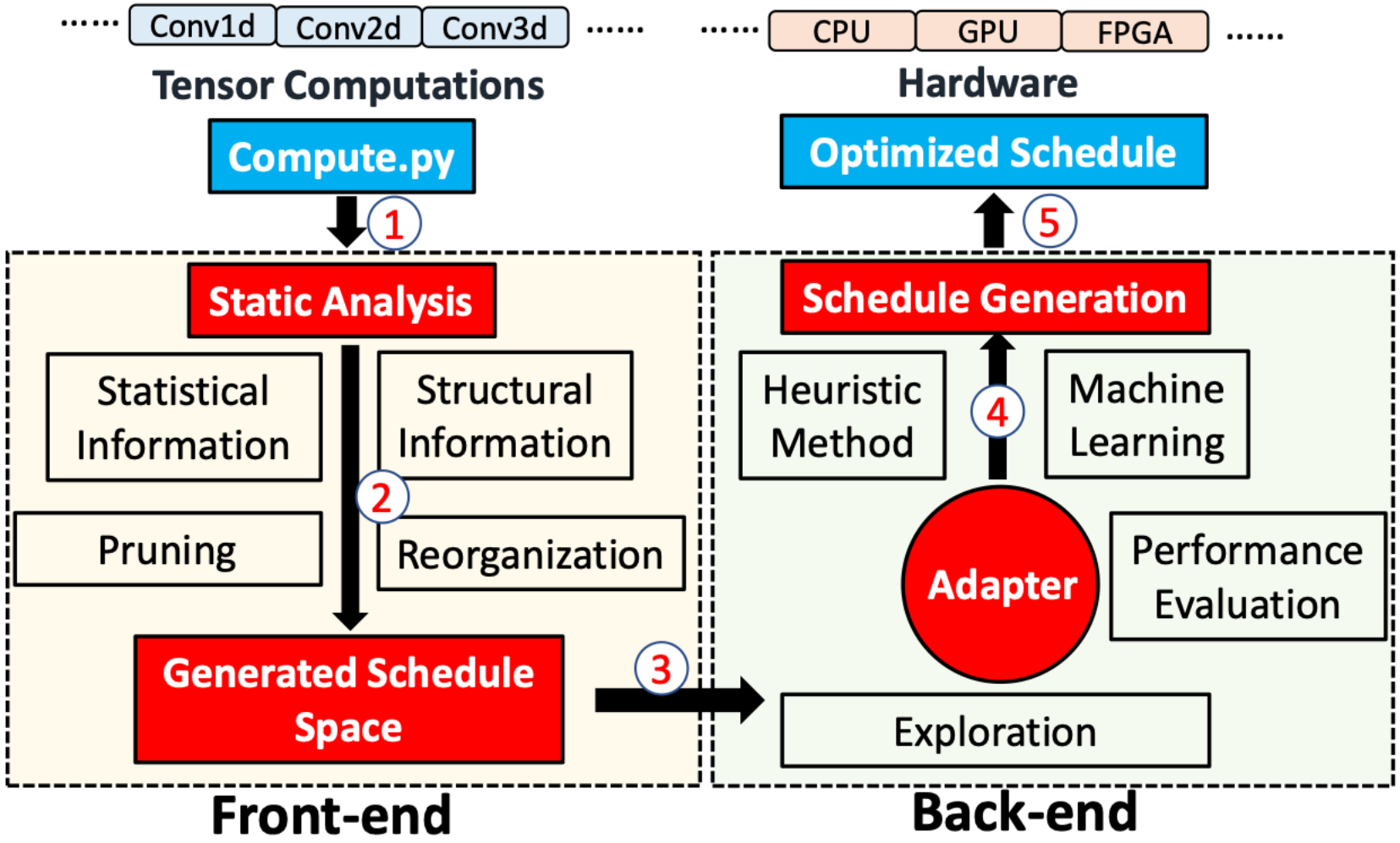
Table 4. Configurations of 15 distinctive convolution layers in YOLO v1.

Name	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
C	3	64	192	128	256	256	512	256	512	512	1024	512	1024	1024	1024
K	64	192	128	256	256	512	256	512	512	1024	512	1024	1024	1024	1024
H/W	448	112	56	56	56	56	28	28	28	28	14	14	14	14	7
k, st	7,2	3,1	1,1	3,1	1,1	3,1	1,1	3,1	1,1	3,1	1,1	3,1	3,1	3,2	3,1

[1] https://www.synopsys.com/designware-ip/technical-bulletin/building-efficient-deep-learning-dwtb_q318.html

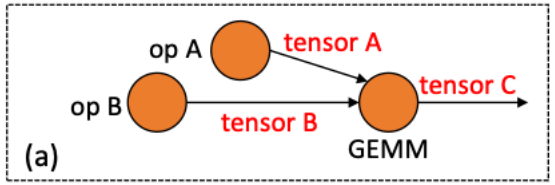
Introduction

- **FlexTensor**



Front-end of FlexTensor

- **Static Analysis**
  - **Structural Information**
    : Node in computation graph
    >> trip counts, loop order, # of spatial/reduce loop
  - **Statistical Information**
    : Edge in computation graph
    >> # of edge in mini-graph, # of input/output tensor in node, # of consumer nodes




```
for i  in range ( 1024 ):
   for j  in range ( 1024 ):
      C [ i, j ] = 0  # initialize C
      for k  in range ( 1024 ):
         C [ i, j ] += A [ i , k ] * B [ k, j ]
```

(b)

statistical

# sl	2
# rl	1
stc	[1024, 1024]
rtc	[1024]
order	[i, j, k]

structural

# node	3
# in	2
# out	1
# cs	0

(c)

Front-end of FlexTensor

- **Generating Schedule Space**
  - **Enumerate combination of schedule primitives and parameters >> Too large schedule space without constraint**
  - **Pruning schedule space**
    1. **Limit the number of primitive combinations**
    2. **Limit the split factors to divisible split choices for each loop**
    3. **Pre-determine certain decisions for different hardware**
  - **Reorder space to use structural similarity after pruning**
  - **Change original 1-dimensional list to high-dimensional space**

"split": $i(1024) \rightarrow [i_1(4), i_2(4), i_3(8), i_4(8)]$
"split": $j(1024) \rightarrow [j_1(4), j_2(4), j_3(8), j_4(8)]$
"split": $k(1024) \rightarrow [k_1(8), k_2(4), k_3(8), k_4(4)]$
"reorder": $i_1, j_1, i_2, j_2, k_1, i_3, k_2, j_3, k_3, i_4, k_4, j_4$
"fuse": $(i_1, j_1, i_2, j_2) \rightarrow outer$
"parallel": $outer$
"vectorize": j_4
"unroll": i_4

(d)

```
[
split  [4, 4, 8, 8], [4, 4, 8, 8], [8, 4, 8, 4],
reorder  [1,5,2,6,9,3,10,7,11,4,12,8],
fuse  [1, 5, 6, 7, 8, 9, 10, 11, 12],
unroll  [0, 0, 0, 0, 0, 1, 0, 0]
]
```

(e)

Back-end of FlexTensor

- **Exploring Schedule Space**
  - **Heuristic Method (Simulated Annealing)**
    E_p**: Performance of** p
    E^***: Highest performance value**
    $exp(-\gamma \frac{(E^* - E_p)}{E^*})$**: Probability that** p **is chosen as the starting point**

Back-end of FlexTensor

- **Exploring Schedule Space**
  - **Machine Learning (DQN)**
    state: point p
    action: direction d
    reward: $r(p, d) = E_e - E_p$ where e is obtained if we move from p along direction d
  - **Objective: find a series of good direction ds to finally reach the optimal point**
    objective function: $\sum_{t}^{T} r(s_t, a_t)$
  - **Evaluate E_p**
    Use analytical model with workload, #PE, data write/read time, computation time

Back-end of FlexTensor

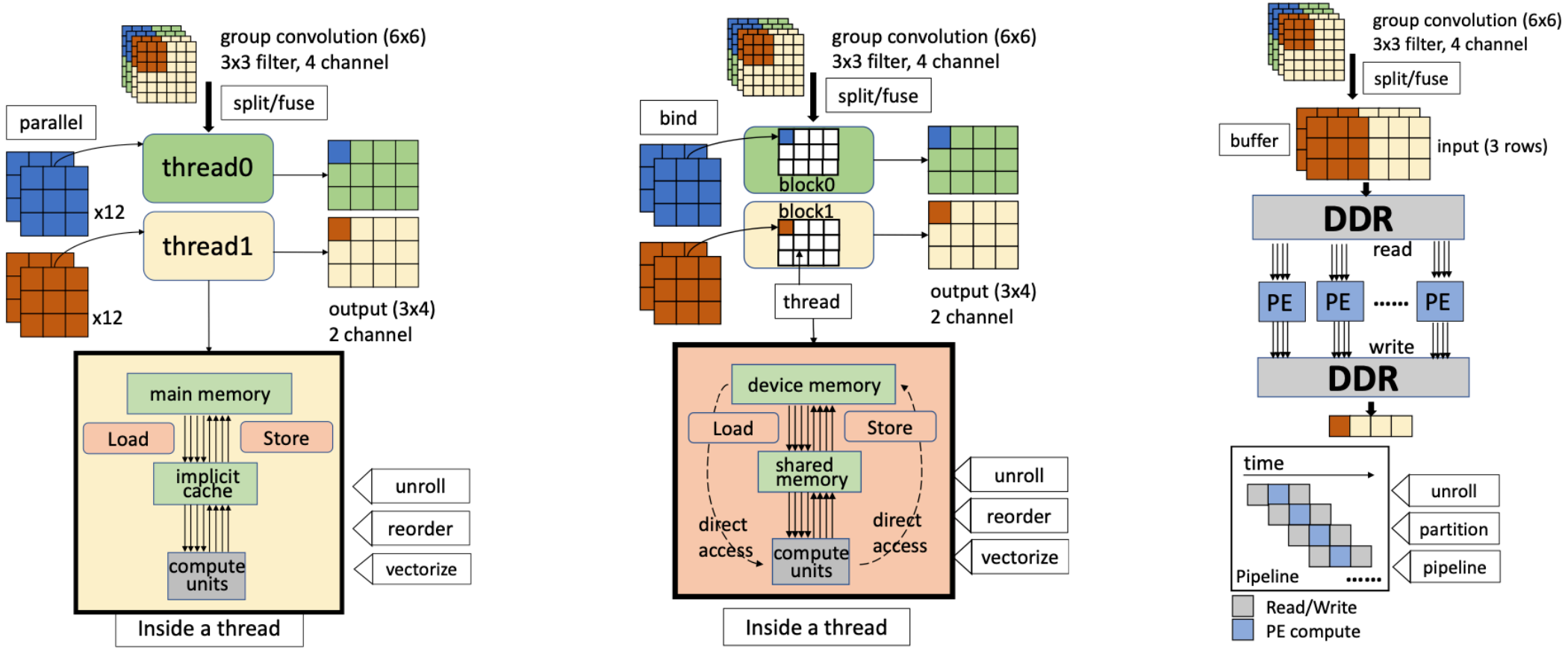


Figure 4. FlexTensor's schedule generation for CPU, GPU and FPGA. Use group convolution as example for illustration.

Experiment

Table 3. Benchmark specifications.

Tensor Computations		Analysis Results		Library Support		FLOPs	Precision	Test Cases
Operator	Abbr.	#sl/rl	#node	CPU	GPU			
GEMV	GMV	1/1	1	MKL	cuBlas	16K-1M	float32	6
GEMM	GMM	2/1	1	MKL	cuBlas	32K-8.6G	float32	7
Bilinear	BIL	2/2	1	MKL	cuBlas	1G	float32	5
1D convolution	C1D	6/2	2	MKL-DNN	cuDNN	50M-200M	float32	7
Transposed 1D convolution	T1D	9/2	3	PyTorch	cuDNN	50M-200M	float32	7
2D convolution	C2D	8/3	2	MKL-DNN	cuDNN	77M-3.7G	float32	15
Transposed 2D convolution	T2D	12/3	3	PyTorch	cuDNN	77M-3.7G	float32	15
3D convolution	C3D	10/4	2	PyTorch	cuDNN	77M-6.6G	float32	8
Transposed 3D convolution	T3D	15/4	3	PyTorch	cuDNN	77M-6.6G	float32	8
Group convolution	GRP	4/3	2	MKL-DNN	cuDNN	20M-900M	float32	14
Depthwise convolution	DEP	4/3	2	MKL-DNN	cuDNN	250K-3.6M	float32	7
Dilated convolution	DIL	4/3	2	MKL-DNN	cuDNN	100M-1.2G	float32	11

Experiment

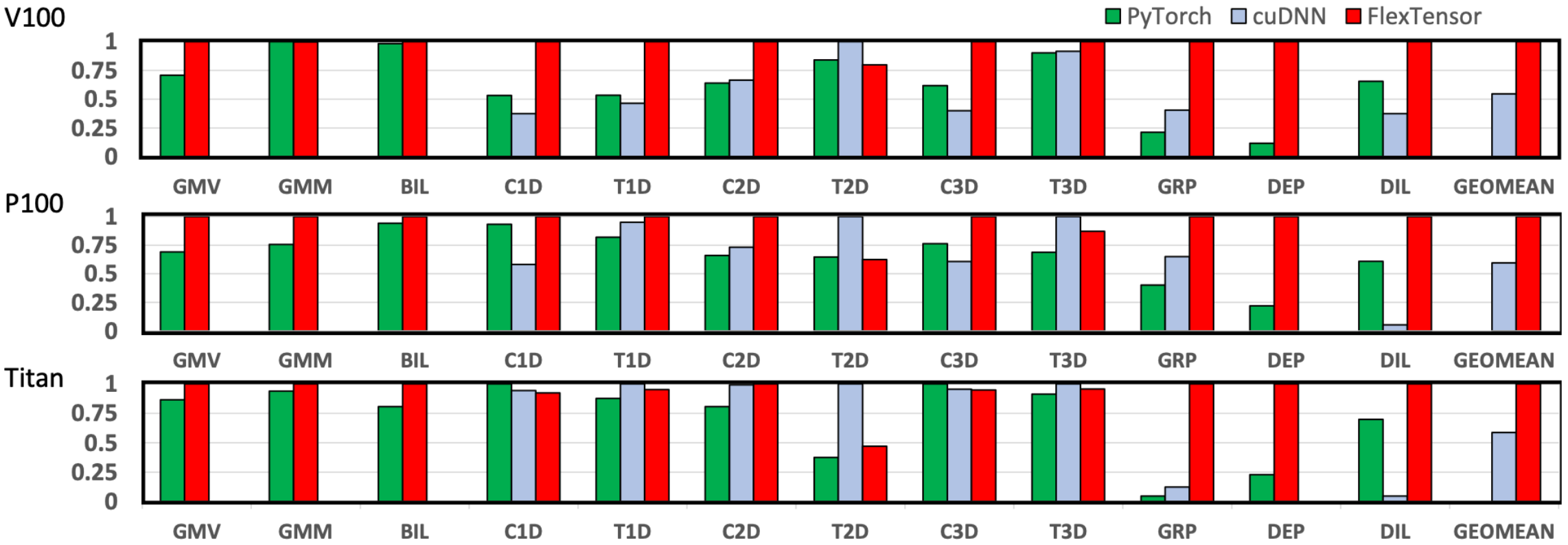
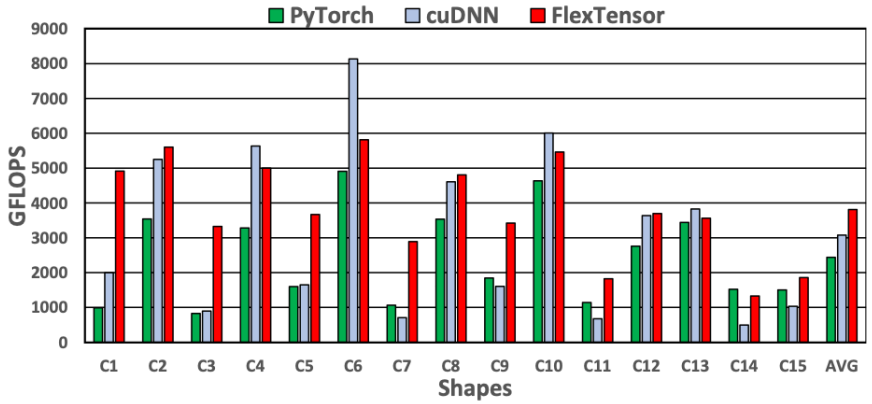
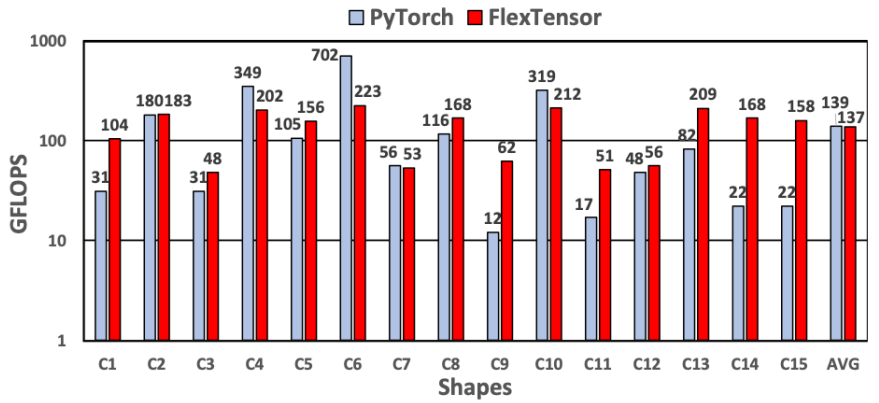


Figure 5. Normalized performance of native PyTorch, cuDNN and FlexTensor on different GPUs.

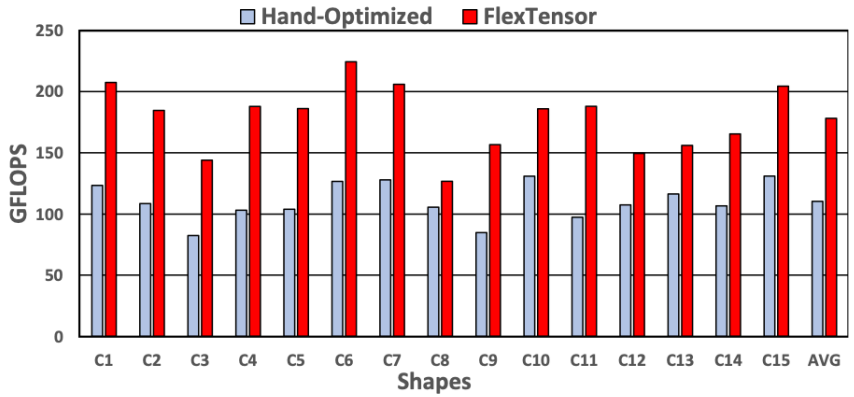
Experiment



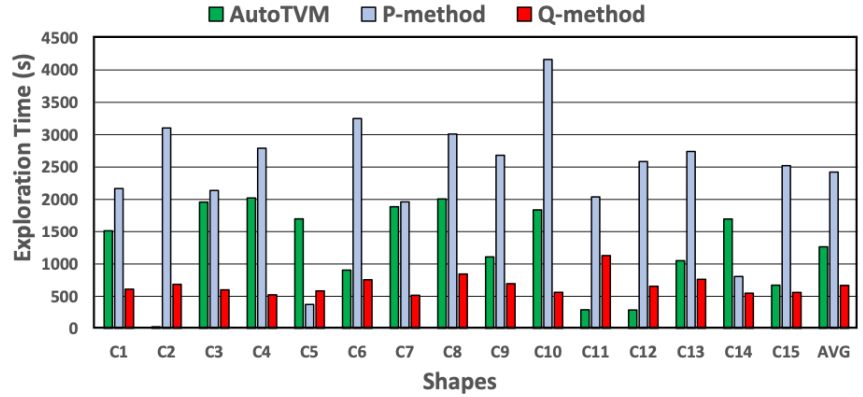
(a) Results of FlexTensor compared to PyTorch (without cuDNN) and cuDNN library on NVIDIA V100 GPU for 2D convolutions.



(b) Results of FlexTensor compared to PyTorch on Intel Xeon E5-2699 v4 for 2D convolutions.



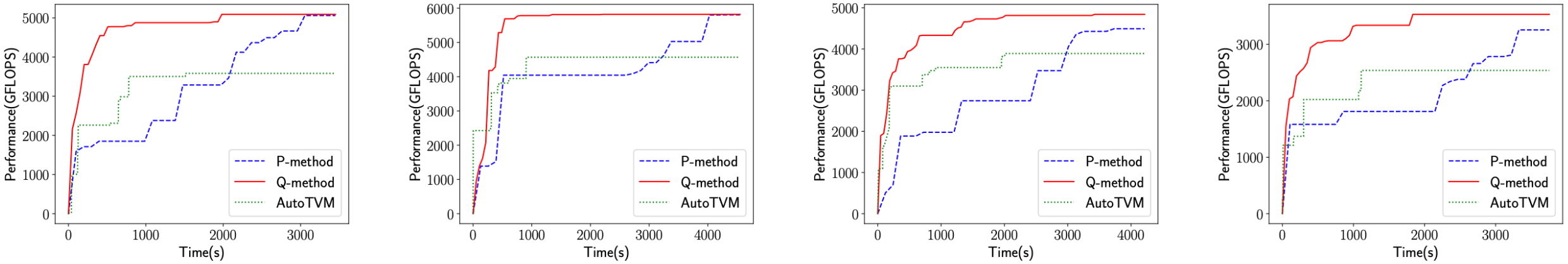
(c) Results of FlexTensor comapred to hand-optimized OpenCL baselines on Xilinx VU9P FPGA for 2D convolutions.



(d) Exploration time comparison of AutoTVM, P-method, and Q-method.

Figure 6. Performance for 2D convolution on heterogeneous hardware.

Experiment



(a) C1 **(b)** C6 **(c)** C8 **(d)** C9

Figure 7. Performance vs. Exploration time

Thank you