Performance-Portable Autotuning of OpenCL Kernels for Convolutional Layers of Deep Neural Networks

Yaohung M. Tsai, Piotr Luszczek

Jakub Kurzak, Jack Dongarra

Nov. 14, 2016

MLHPC Workshop @ SC16

INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

BLAS: The Core of Numerical Algorithms

BLAS: Basic Linear Algebra Subprograms

Developers are always trying to map the computation part of their algorithms into matrix operations to take advantage of highly optimized BLAS libraries:

- ATLAS: Automatically Tuned Linear Algebra Software
- GotoBLAS, OpenBLAS
- Intel Math Kernel Library
- AMD Core Math Library, clBLAS
- Nvidia cuBLAS

They usually obtains 90% of theoretical peak performance with sufficient matrix size.

What about Deep Neural Network?

- Training can easily take days or weeks.

- Nvidia cuDNN
- Intel MKL 2017, Intel MKL-DNN (Open source).

- So many frameworks.
- Fast growing / evolving area.
- New models / networks using the same or similar components / layers.

Why don't we do auto tuning?

The Hardware Side

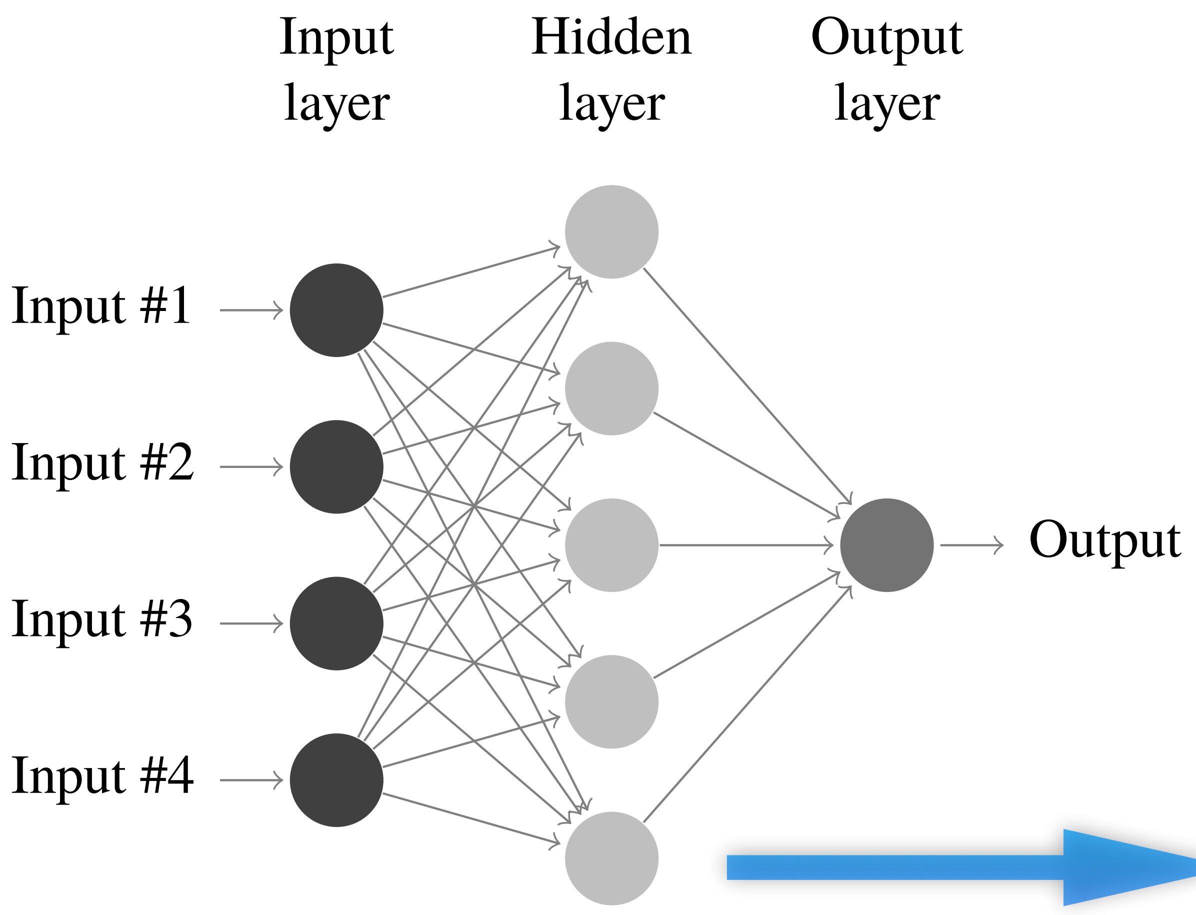
GPU Card Name	Peak Performance (GFlop/s)	Price (USD)	Performance/Price (GFlops/s/USD)	Memory Size (GB)	Memory Bandwidth (GB/s)
Nvidia Titan X (Maxwell)	6144	$900	6.83	12	336
AMD Fury X	8601	$390	22.05	4	512 (HBM)
Nvidia Titan X (Pascal)	10157	$1200	8.46	12	480
Nvidia GTX1080 (Pascal)	7967	$620	12.85	8	320

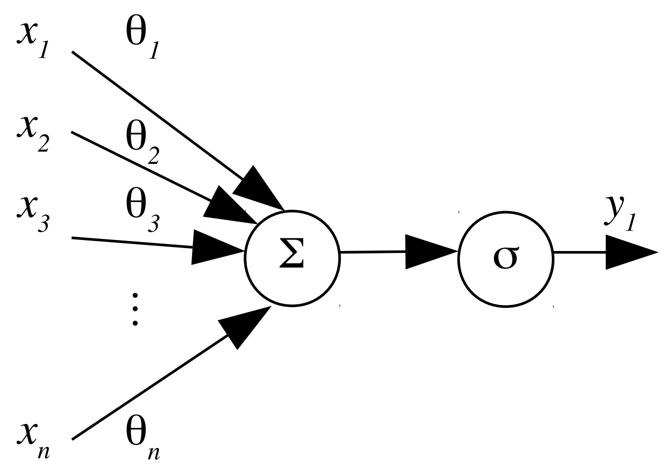
Single precision performance.

Price is the market price in Nov. 2017.

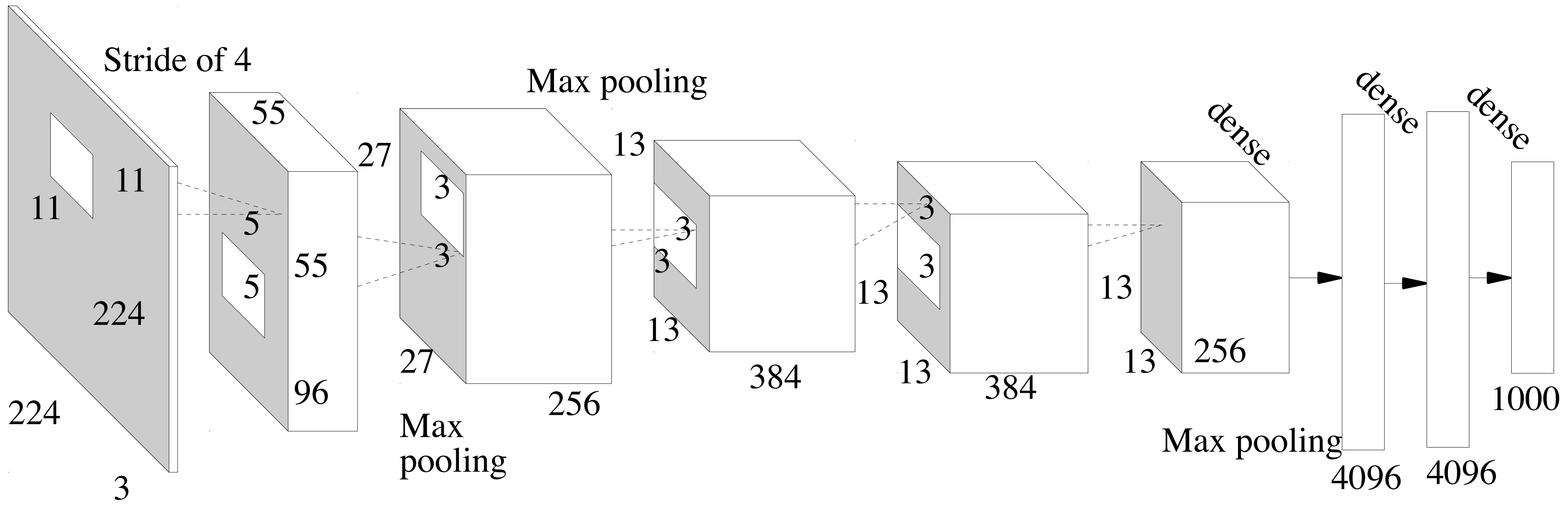
No Nvidia Tesla P100, P4, P40 price, yet.

Background: Neural Network





Background: DNN, Training, Inference.



Alexnet for image classification.

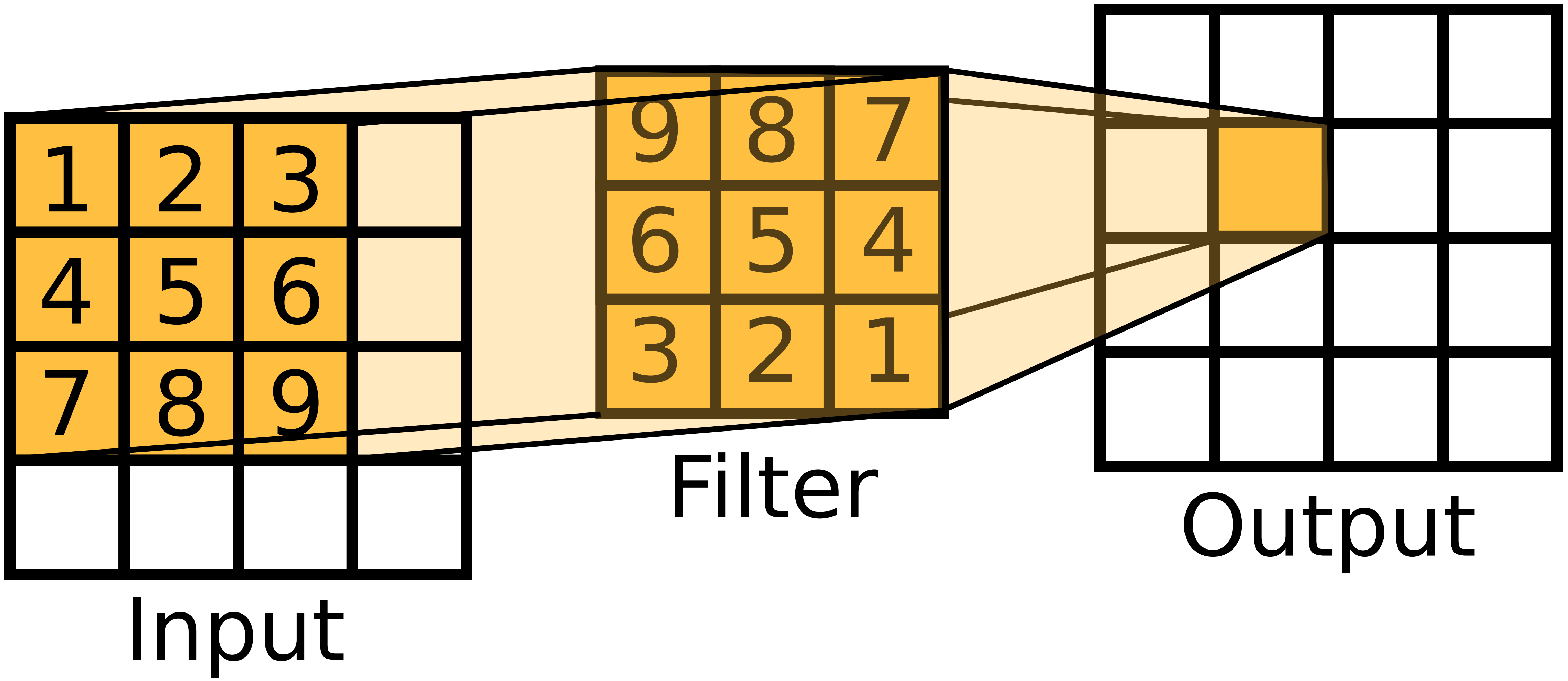
Inference: Feed image into network and get the result likelihood vector.

Training: Back propagate the error and update the weights within layers.

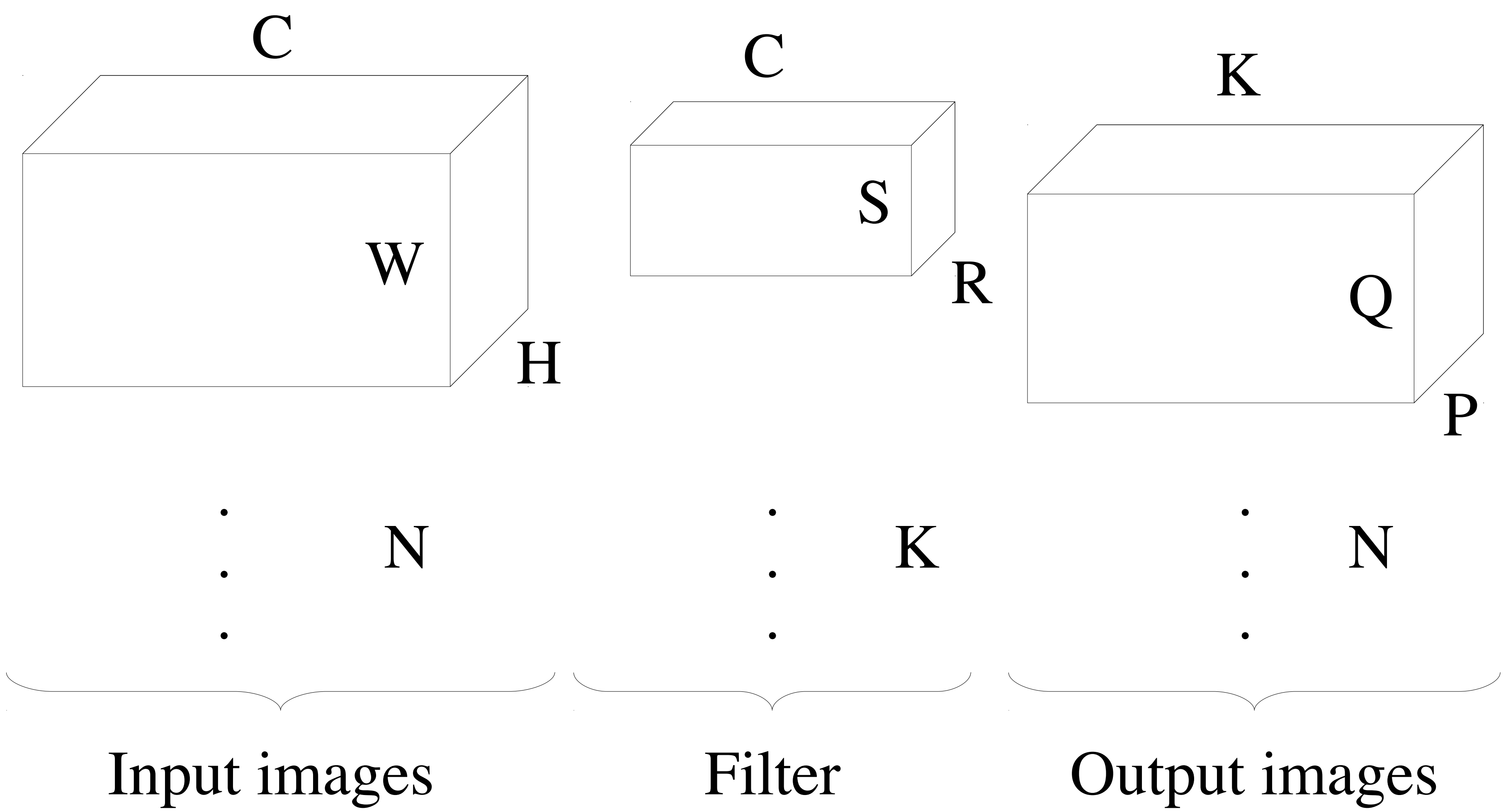
Convolutional layers and fully connected layers are the compute intensive parts.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

Discrete 2D Convolution

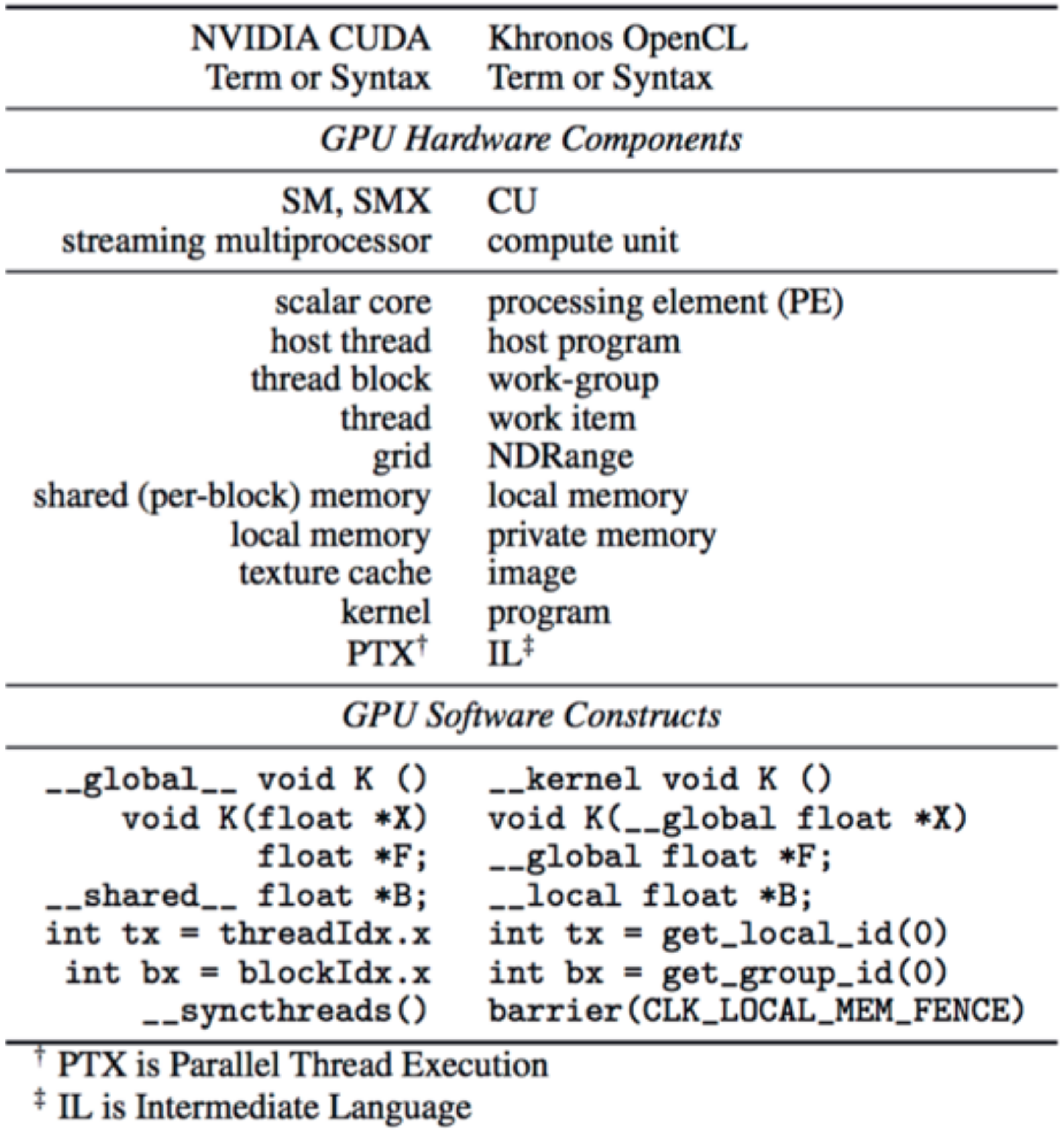


Dimensions of Convolutional Layer



$$O^{(n)}(k) = \Sigma_c \mathtt{conv2}\left(I^{(n)}(c), F^{(c)}(k)\right)$$

CUDA v.s. OpenCL



NVIDIA CUDA Term or Syntax	Khronos OpenCL Term or Syntax
GPU Hardware Components	
SM, SMX streaming multiprocessor	CU compute unit
scalar core	processing element (PE)
host thread	host program
thread block	work-group
thread	work item
grid	NDRange
shared (per-block) memory	local memory
local memory	private memory
texture cache	image
kernel	program
PTX[†]	IL[‡]
GPU Software Constructs	
`__global__ void K ()`	`__kernel void K ()`
`void K(float *X)`	`void K(__global float *X)`
`float *F;`	`__global float *F;`
`__shared__ float *B;`	`__local float *B;`
`int tx = threadIdx.x`	`int tx = get_local_id(0)`
`int bx = blockIdx.x`	`int bx = get_group_id(0)`
`__syncthreads()`	`barrier(CLK_LOCAL_MEM_FENCE)`

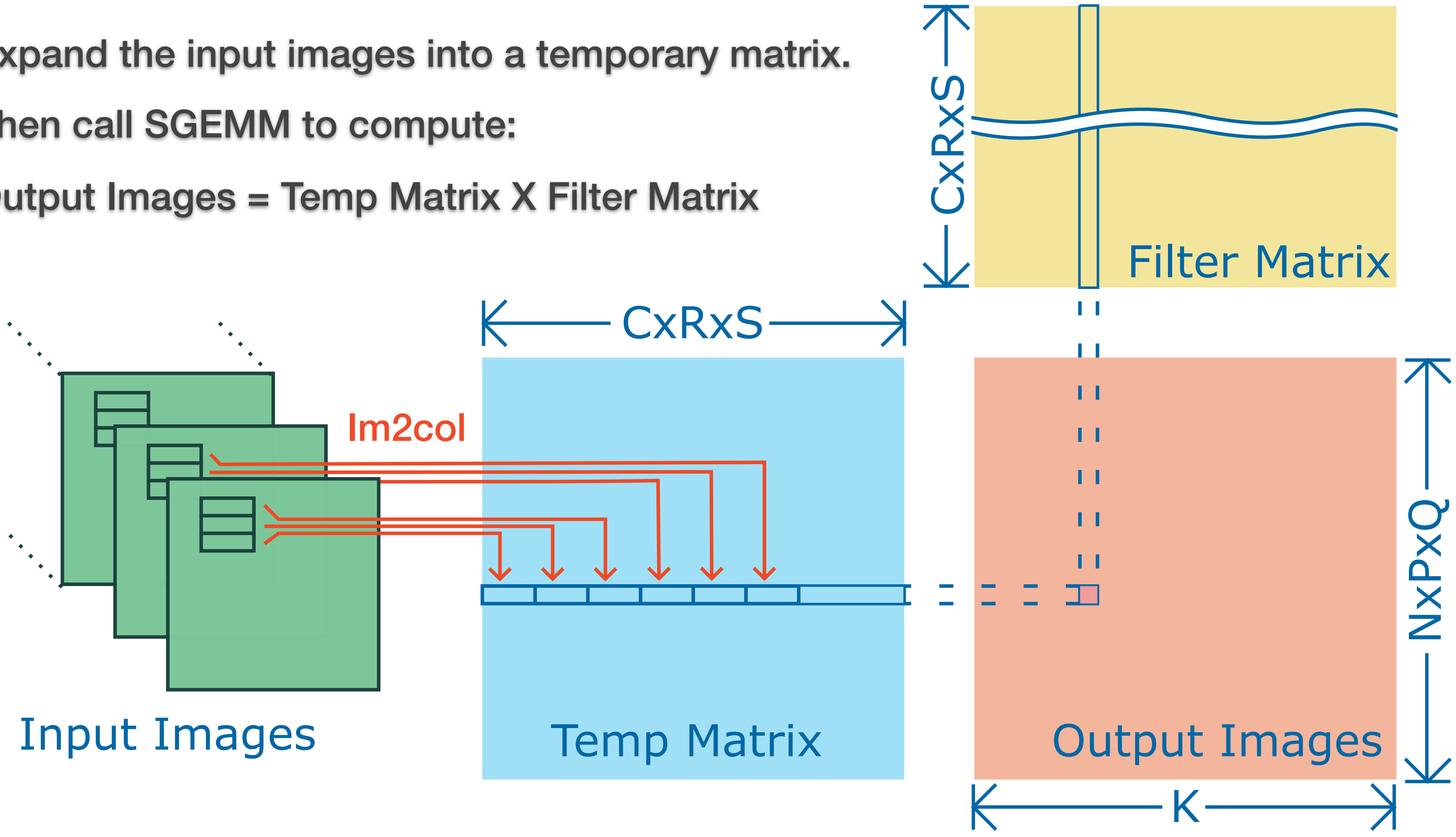
[†] PTX is Parallel Thread Execution
[‡] IL is Intermediate Language

Typical Im2col approach with SGEMM

Expand the input images into a temporary matrix.

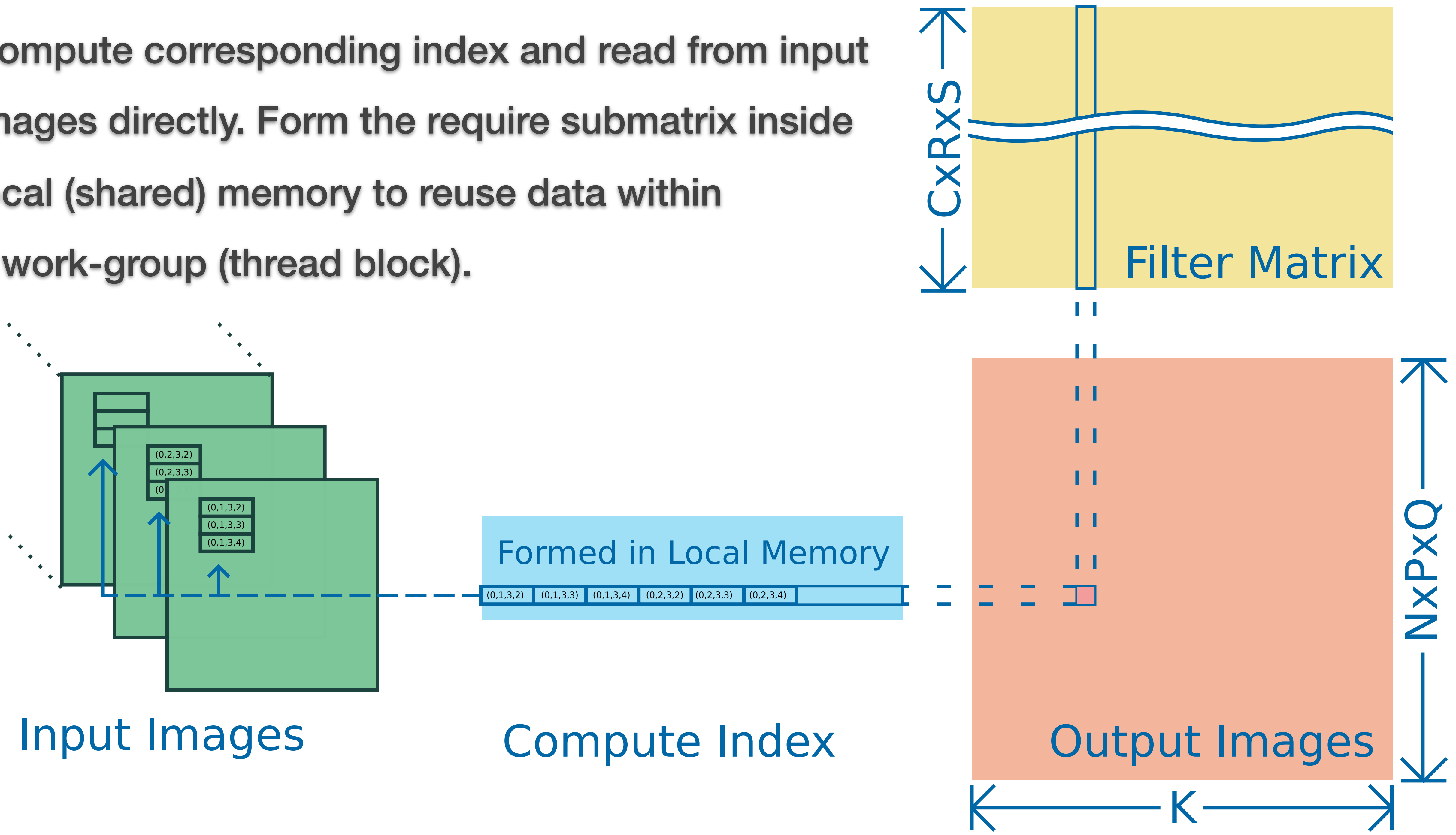
Then call SGEMM to compute:

Output Images = Temp Matrix X Filter Matrix



Proposed Local Memory Convolution Kernel

Compute corresponding index and read from input images directly. Form the require submatrix inside local (shared) memory to reuse data within a work-group (thread block).



Comparing SGEMM and Convolutional Layer

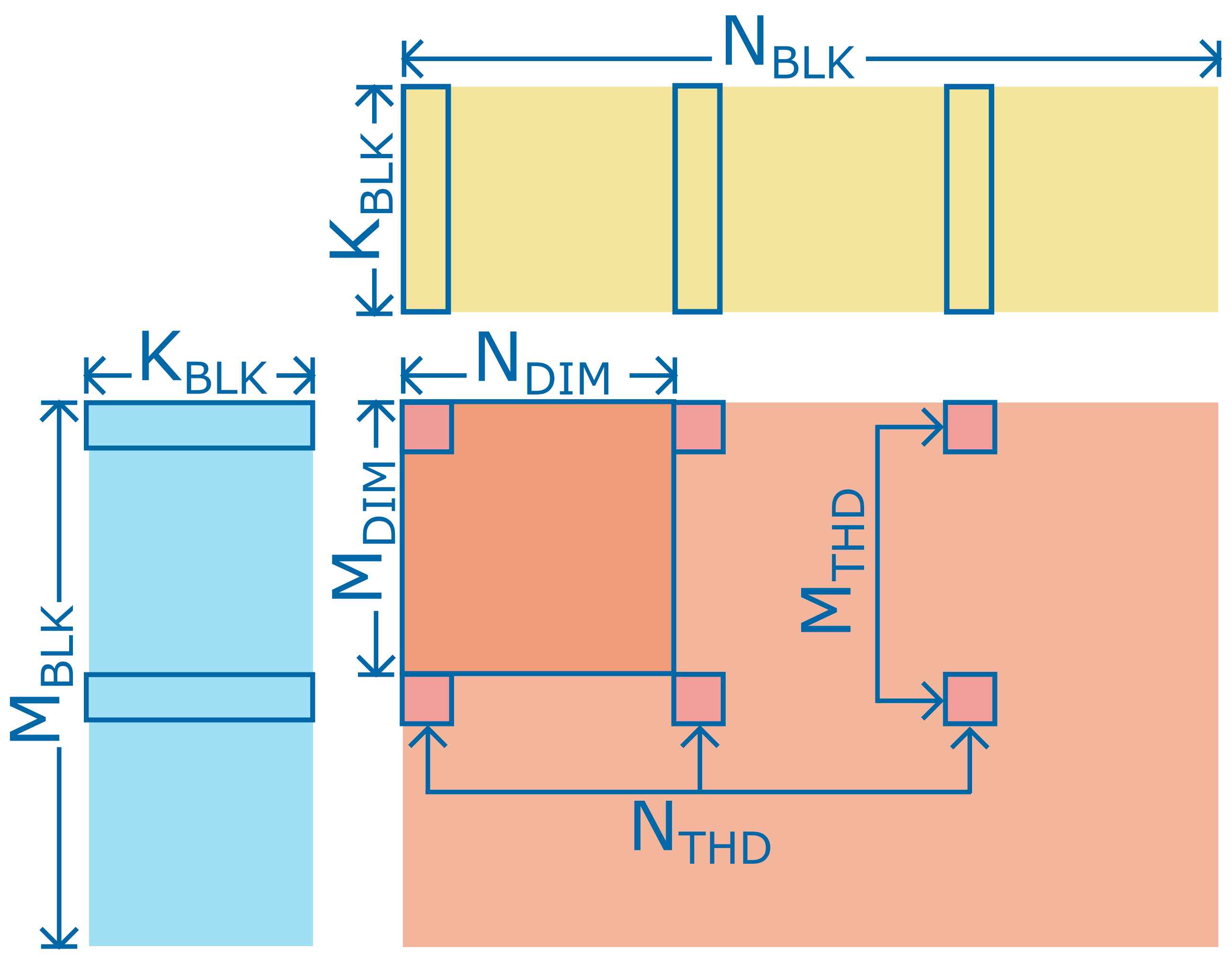
clBLAS SGEMM

```
for(int k=0; k<K; k+=16) {
  __local float* plA = lA + idx*97+idy;
  __local float* plB = lB + idx*97+idy;


  barrier(CLK_LOCAL_MEM_FENCE);

  //Load next submatrix into local memory







  for(int i=0; i<96; i+=16)
    plA[i] = A[i];
  for(int i=0; i<96; i+=16)
    plB[i] = B[i*ldb];




  barrier(CLK_LOCAL_MEM_FENCE);

  //Inner computation loop
  ...

  //Move to next submatrix
  A += 16*lda;
  B += 16;
}
```

Our Kernel

```
for(int k=0; k<K; k+=16) {
  __local float* plA = lA + idx*97+idy;
  __local float* plB = lB + idx*97+idy;
  int x = p * stride_u + r;
  int y = q * stride_v + s;
  barrier(CLK_LOCAL_MEM_FENCE);

  //Load next submatrix into local memory
  //Check if it's in the padding region
  if( x < pad_h || x >= H+pad_h ||
      y < pad_w || y >= W+pad_w )
    for(int i=0; i<96; i+=16)
      plA[i] = 0;
  else
    for(int i=0; i<96; i+=16)
      plA[i]=A[index+c*N*H*W+r*N*W+s*N+i];
  for(int i=0; i<96; i+=16)
    plB[i] = B[i*ldb];

  //Update indices for next submatrix
  s += 16; r += s/S; s = s%S;
  c += r/R; r = r%R;
  barrier(CLK_LOCAL_MEM_FENCE);

  //Inner computation loop
  ...

  //Move to next submatrix
  A += 16*lda;
  B += 16;
}
```

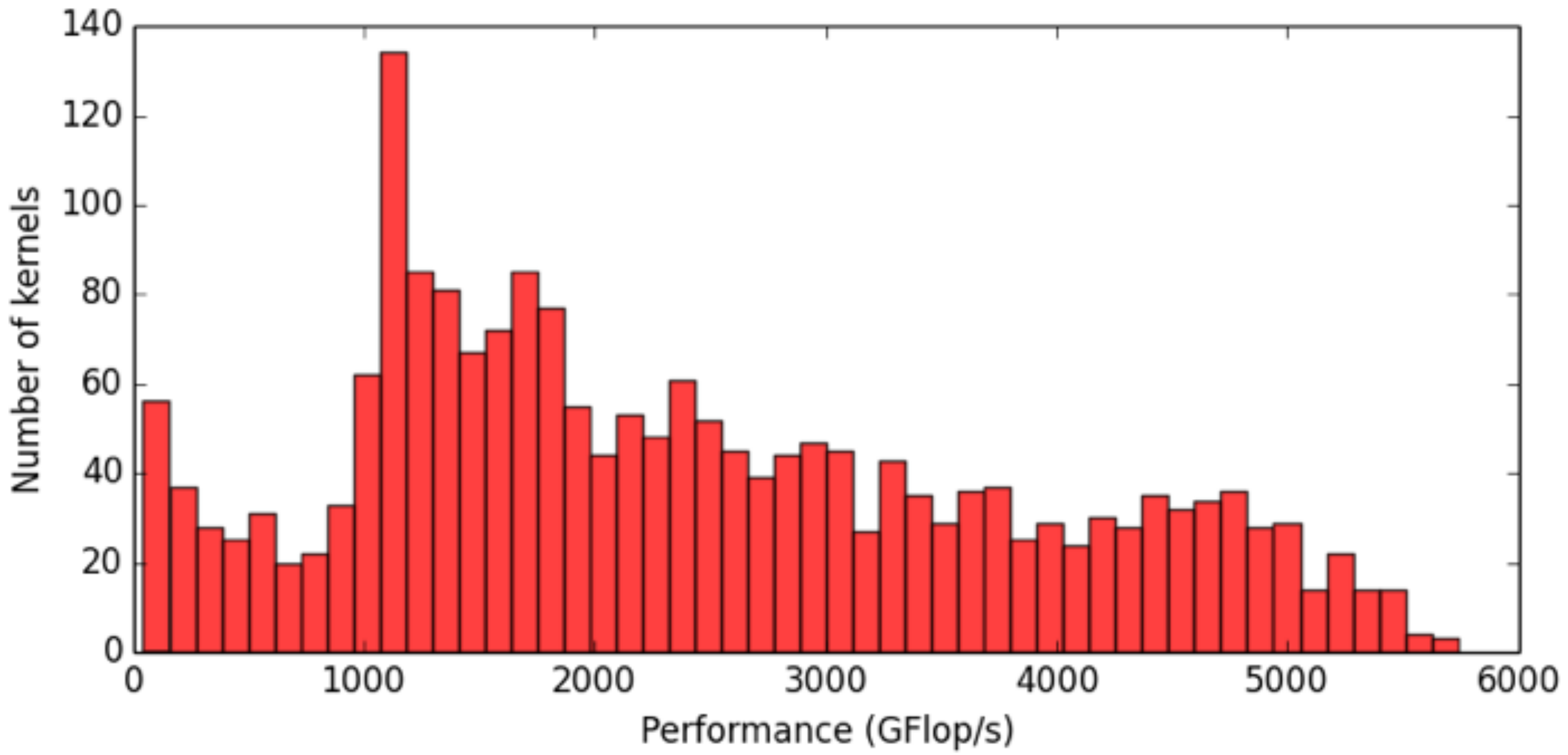
Auto-tuning Parameters and Constraints



- Algorithm level: Constraints from the kernel to remove boundary check.
- Software level: Problem sizes are divisible by tile sizes to remove imbalance workload.
- Hardware level: Ensure there are enough registers and local memory spaces with minimum occupancy.

Performance Results

The tested GPU was the AMD Fury X with peak single-precision (FP32) performance of 8602 Gflop/s and core frequency of 1050 MHz.



The histogram of performances from 2056 kernels on Alexnet L2.

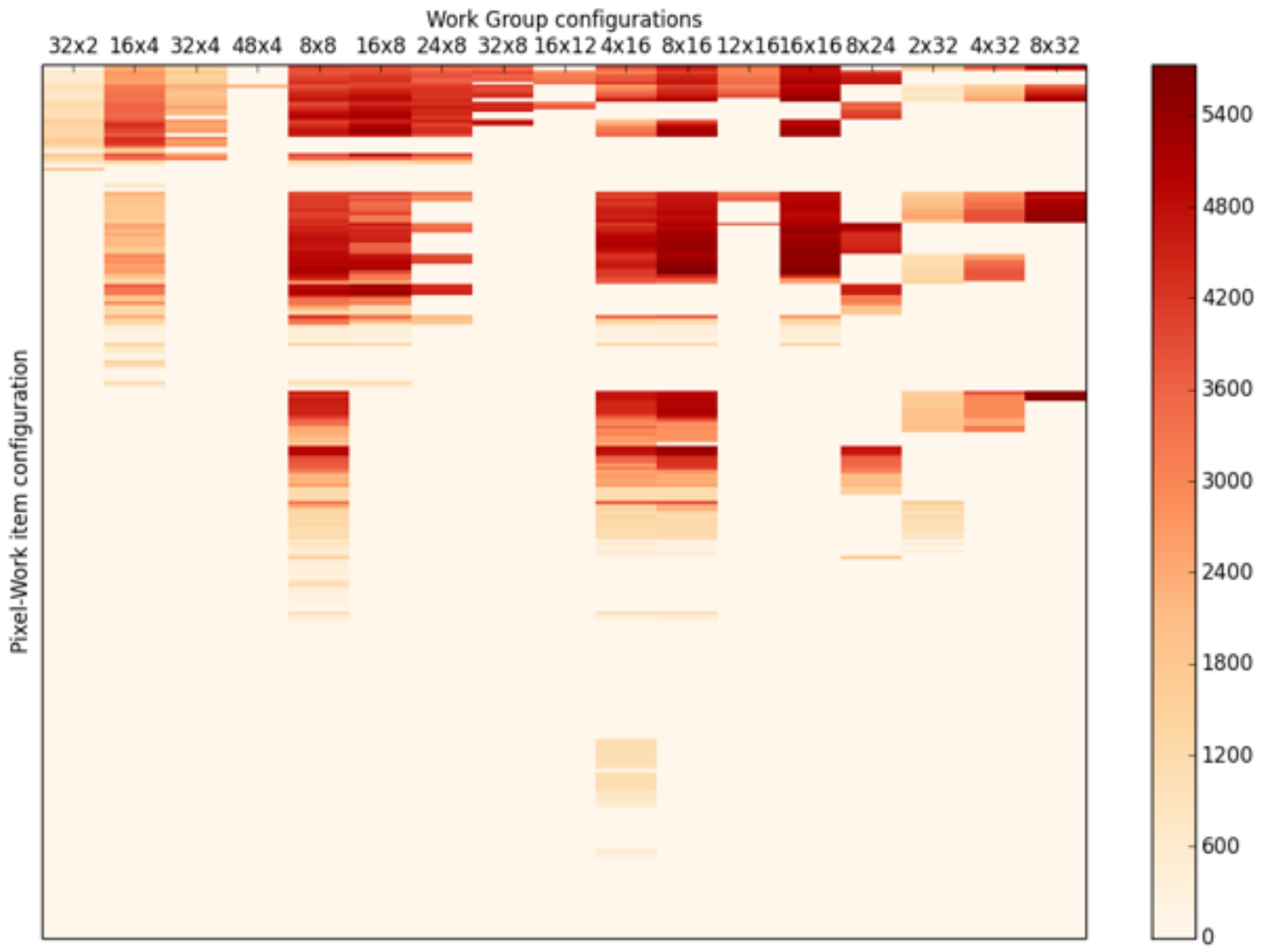
The best kernel achieved 66.7% of peak performance.

The tuning result on Alexnet L2

The darker red indicate the better performance in this heat map.

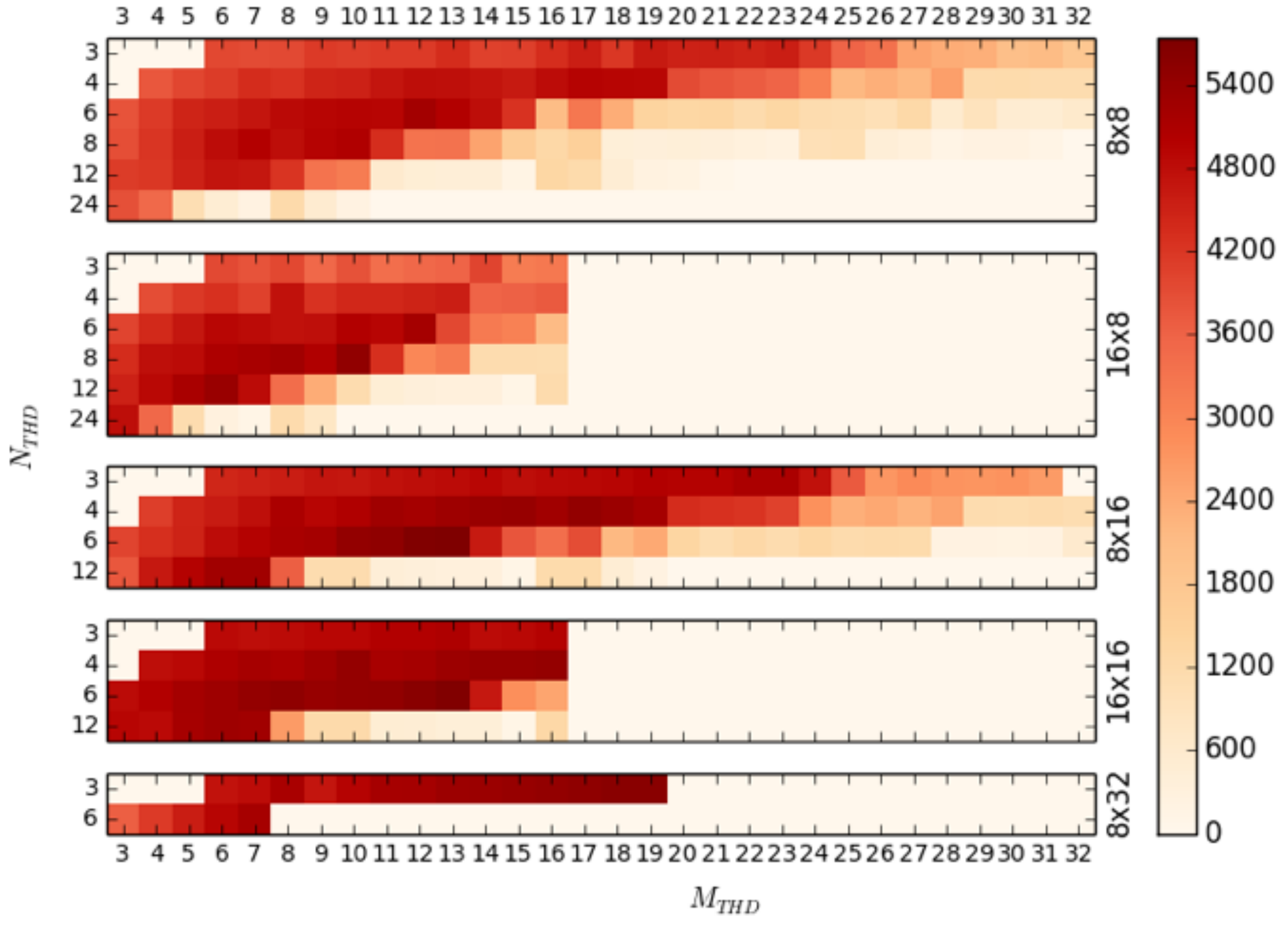
76.7% (3314) of the kernels were invalid and pruned before code generation.

3.4% (145) of the kernels failed at runtime.



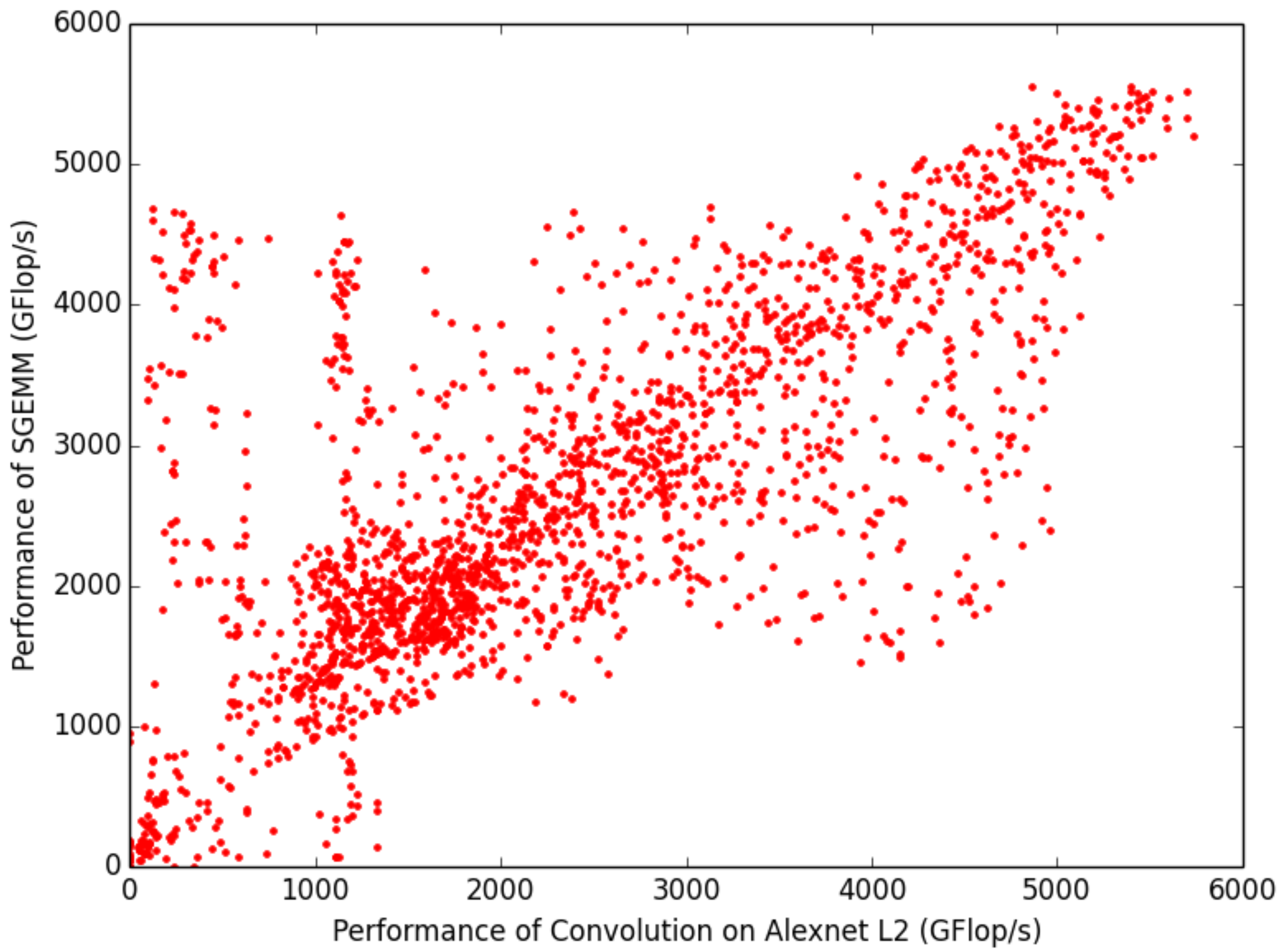
Result of 5 work group configurations on AlexNet L2

Each subfigure is extracted from one column of previous heat map.

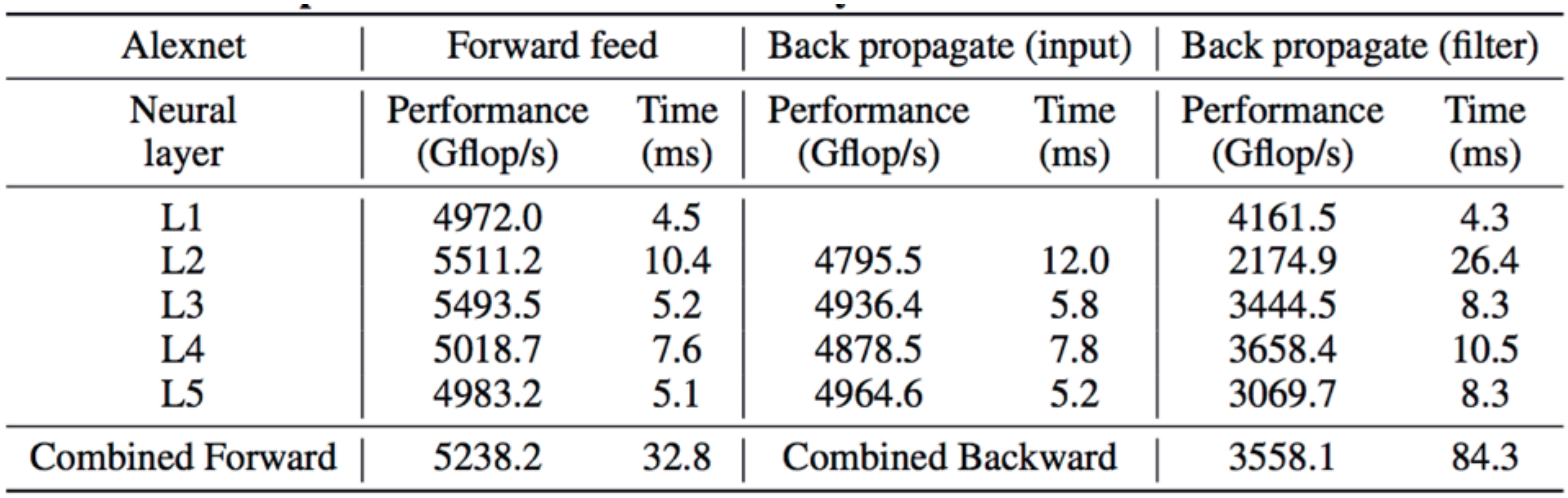


ICL THE UNIVERSITY OF TENNESSEE KNOXVILLE

Comparison between Convolution and SGEMM



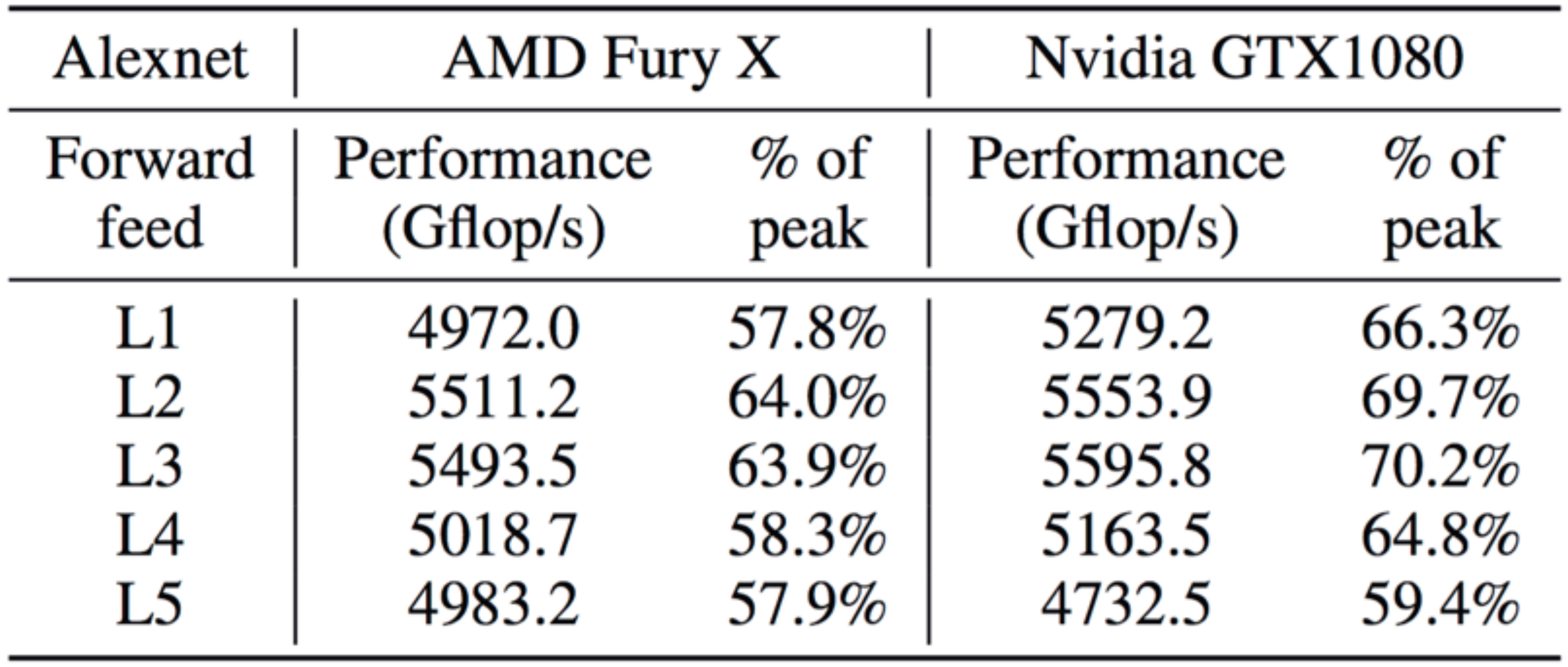
Performance on Alexnet with batch size N=128



Alexnet	Forward feed		Back propagate (input)		Back propagate (filter)	
Neural layer	Performance (Gflop/s)	Time (ms)	Performance (Gflop/s)	Time (ms)	Performance (Gflop/s)	Time (ms)
L1	4972.0	4.5			4161.5	4.3
L2	5511.2	10.4	4795.5	12.0	2174.9	26.4
L3	5493.5	5.2	4936.4	5.8	3444.5	8.3
L4	5018.7	7.6	4878.5	7.8	3658.4	10.5
L5	4983.2	5.1	4964.6	5.2	3069.7	8.3
Combined Forward	5238.2	32.8	Combined Backward		3558.1	84.3

The last column represents updating the filters, whose size K X C X R X S usually is much smaller than either the input or the output images. Hence, there is a trade-off between occupancy and data reuse in local memory as the kernels have to pass the data between each other.

The Portable Performance



Alexnet	AMD Fury X		Nvidia GTX1080	
Forward feed	Performance (Gflop/s)	% of peak	Performance (Gflop/s)	% of peak
L1	4972.0	57.8%	5279.2	66.3%
L2	5511.2	64.0%	5553.9	69.7%
L3	5493.5	63.9%	5595.8	70.2%
L4	5018.7	58.3%	5163.5	64.8%
L5	4983.2	57.9%	4732.5	59.4%

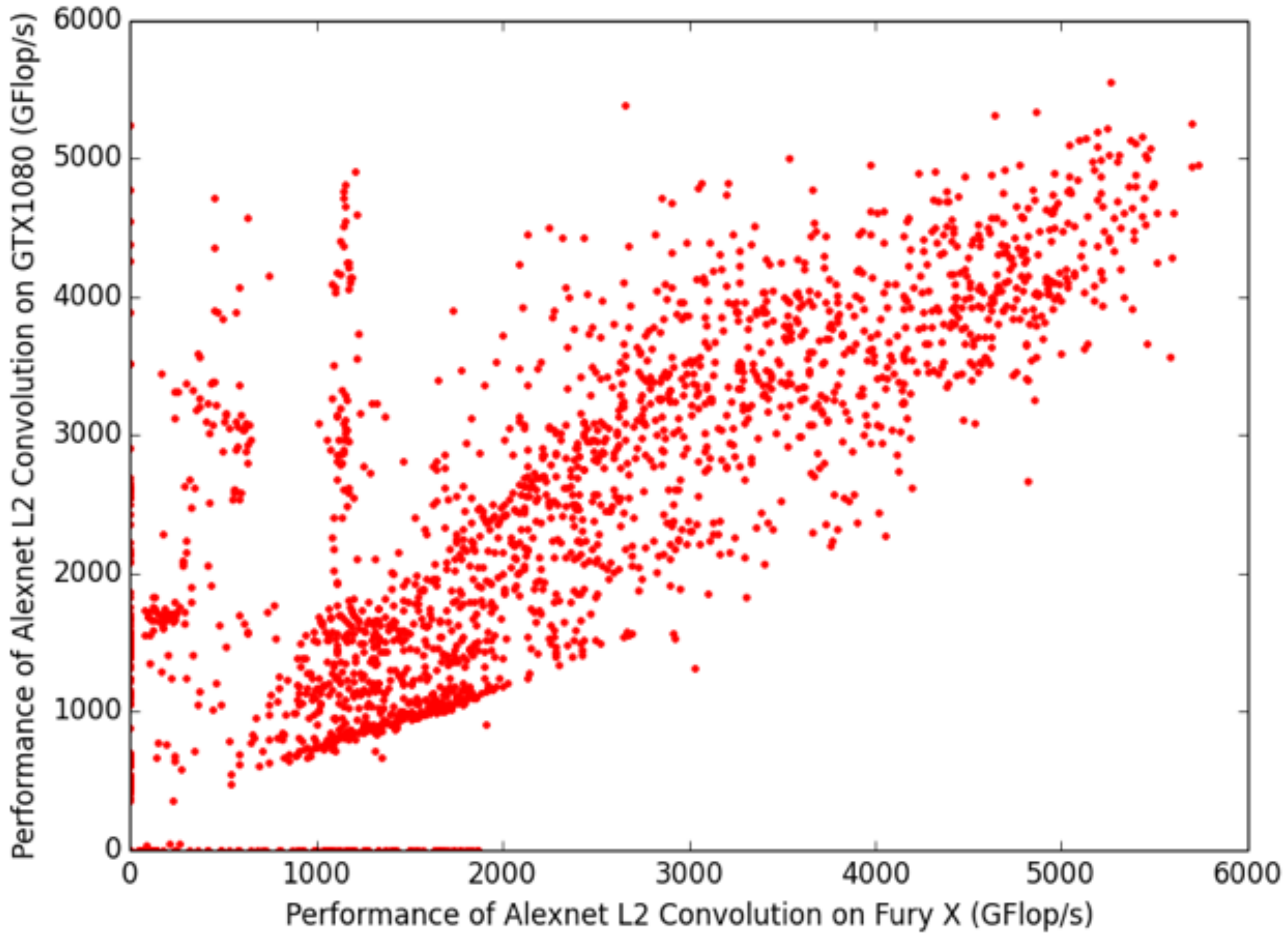
Theoretical peak performance of AMD Fury X : 8602 Gflop/s.

Nvidia GTX1080 : 7967 Gflop/s.

The Portable Performance

Theoretical peak performance of AMD Fury X : 8602 Gflop/s.

Nvidia GTX1080 : 7967 Gflop/s.



Conclusions and Future Work

We proposed:

- An efficient implementation of convolutional layers which does not require extra memory space, which leads to larger batch size N and faster training process.
- Auto tuning approach to achieve high performance without digging into architecture detail.
- Portable performance cross different GPU vendors.

Future directions:

- Integrate our kernel generator and autotuner directly into a deep learning framework.
- Merging other layers like ReLU into single kernel.
- Fast and specialized algorithms like Winograd for 3-by-3 filters.

ICL
INNOVATIVE
COMPUTING LABORATORY

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE