# Scalpel: Customizing DNN Pruning to the Underlying Hardware Parallelism

ISCA 17

Neural Network Quantization & Compact Network Design Study, AI Robotics KR

Constant (Sang-Soo) Park

https://constantpark.github.io/

2019-10-20

# Contents

- Introduction
- Background & Motivations
- Contributions
- Scalpel: SIMD-aware weight/Node pruning
- Evaluation
- Conclusion

# Introduction

- **Pruning deep neural networks to make them fast and small**
  - Removing connections that don't affect significantly affect the results[1]
  - Possible to reduce the number parameters, the amount of computation
  - **But will it really run faster in modern processor and dedicated processor?**

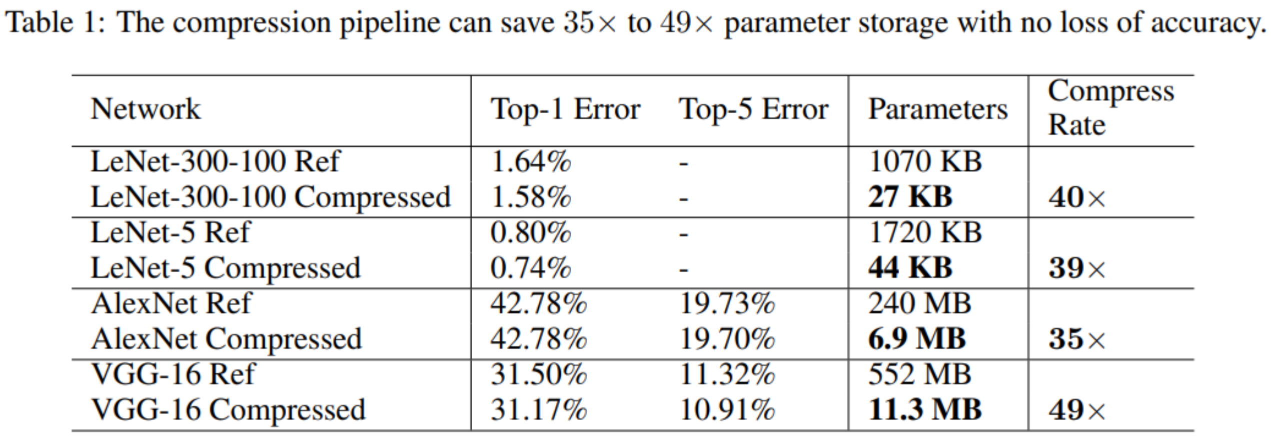
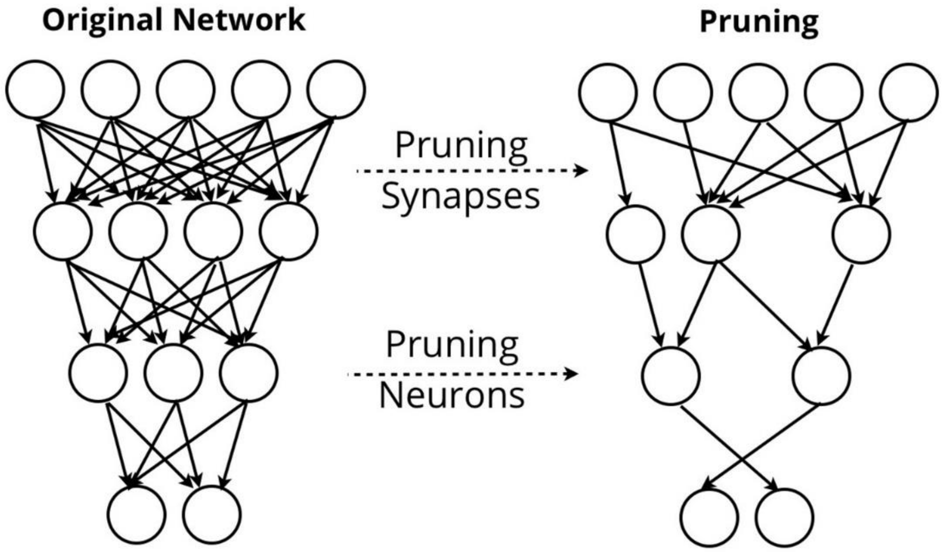


Table 1: The compression pipeline can save 35× to 49× parameter storage with no loss of accuracy.

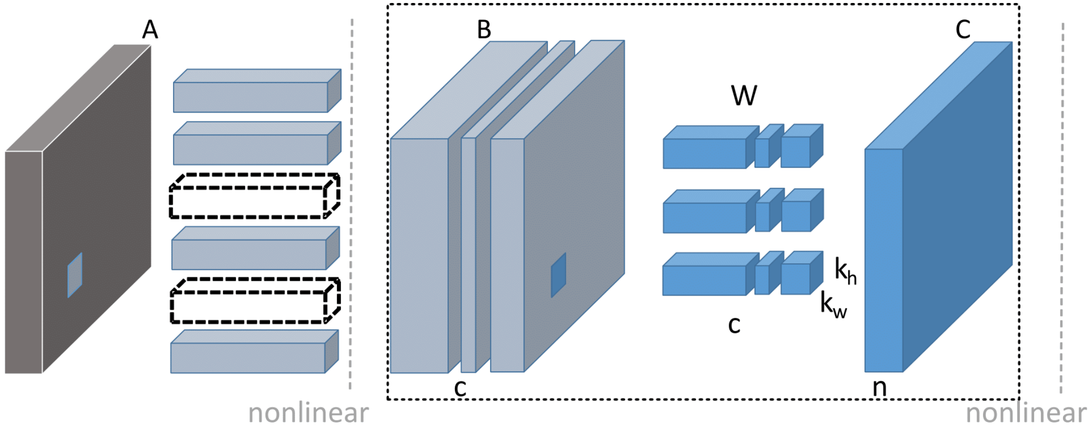
| Network | Top-1 Error | Top-5 Error | Parameters | Compress Rate |
|---|---|---|---|---|
| LeNet-300-100 Ref | 1.64% | - | 1070 KB | |
| LeNet-300-100 Compressed | 1.58% | - | **27 KB** | **40**× |
| LeNet-5 Ref | 0.80% | - | 1720 KB | |
| LeNet-5 Compressed | 0.74% | - | **44 KB** | **39**× |
| AlexNet Ref | 42.78% | 19.73% | 240 MB | |
| AlexNet Compressed | 42.78% | 19.70% | **6.9 MB** | **35**× |
| VGG-16 Ref | 31.50% | 11.32% | 552 MB | |
| VGG-16 Compressed | 31.17% | 10.91% | **11.3 MB** | **49**× |



[1] DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING, ICLR 2016.

# Background: Pruning

- **Fine-grained vs Course-grained pruning**
    - Fine-grained: Random distribution of zero values[2]
    - Course-grained: **Structural pruning** such as channel-pruning[3]



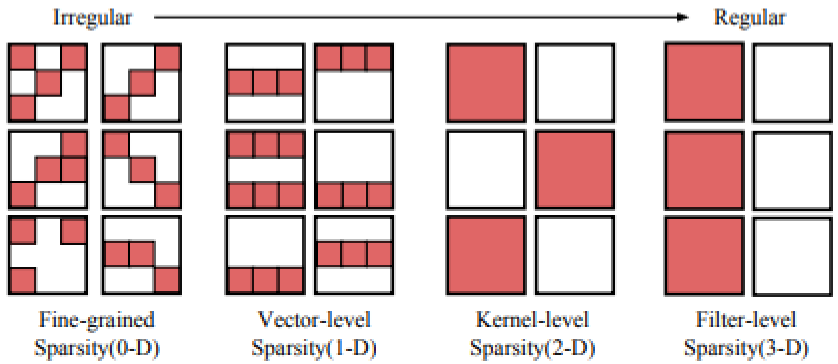
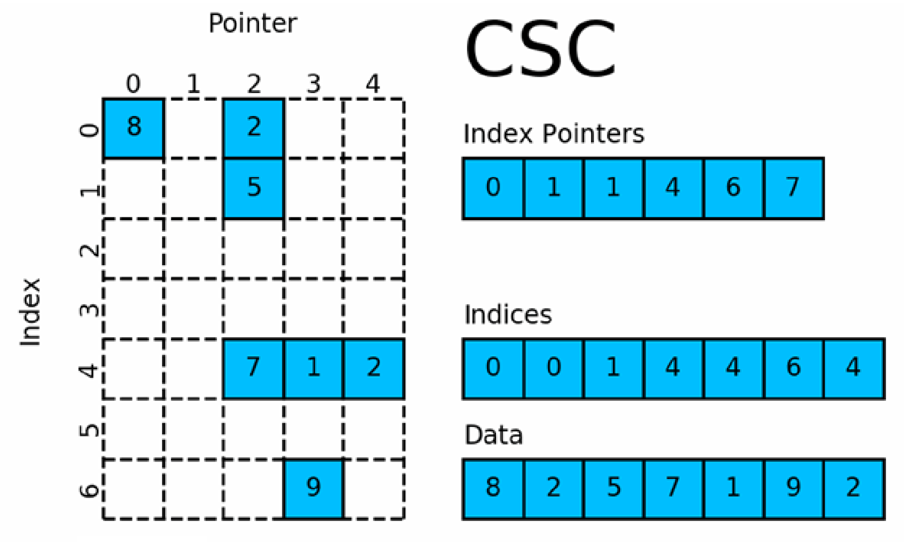

Irregular
Regular
Fine-grained Sparsity(0-D)
Vector-level Sparsity(1-D)
Kernel-level Sparsity(2-D)
Filter-level Sparsity(3-D)

Figure 1: Different sparse structure in a 4-dimensional weight tensor. Regular sparsity makes hardware acceleration easier.

[2] https://github.com/yihui-he/channel-pruning
[3] https://github.com/NervanaSystems/distiller/wiki/Frequently-Asked-Questions-(FAQ)

# Background: Sparse matrix format

- **Storage format for storing matrix**
  - Store non-zero values and **index information (where this value is stored)**
  - Many ways to store sparse matrix: COO, CSR, CRS, etc.
  - Require additional decoding computations



CSC

| Index \ Pointer | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 8 |  | 2 |  |  |
| 1 |  |  | 5 |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  | 7 | 1 | 2 |
| 5 |  |  |  |  |  |
| 6 |  |  |  | 9 |  |

Index Pointers

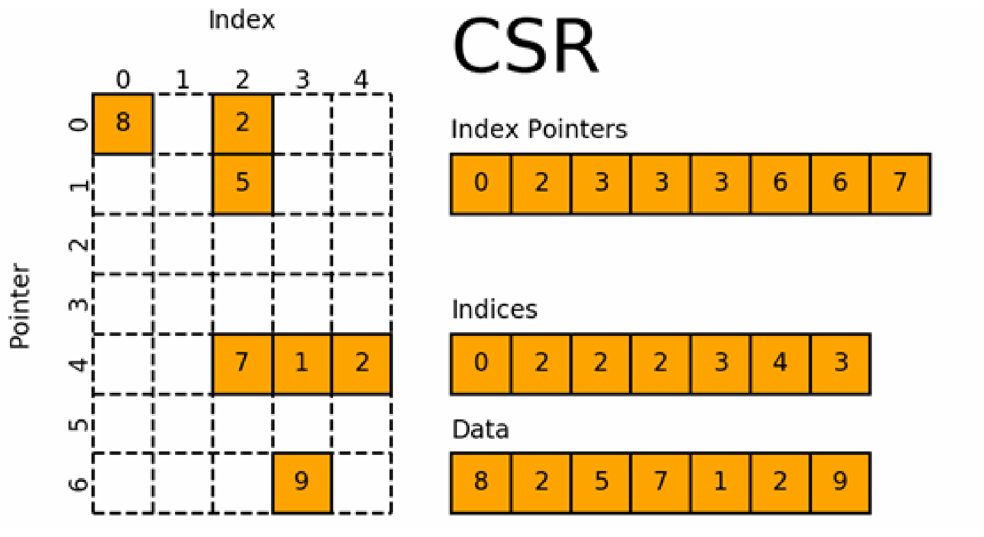
| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 1 | 4 | 6 | 7 |

Indices

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 4 | 4 | 6 | 4 |

Data

| | | | | | | |
|---|---|---|---|---|---|---|
| 8 | 2 | 5 | 7 | 1 | 9 | 2 |



CSR

| Pointer \ Index | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 8 |  | 2 |  |  |
| 1 |  |  | 5 |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  | 7 | 1 | 2 |
| 5 |  |  |  |  |  |
| 6 |  |  |  | 9 |  |

Index Pointers

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 2 | 3 | 3 | 3 | 6 | 6 | 7 |

Indices

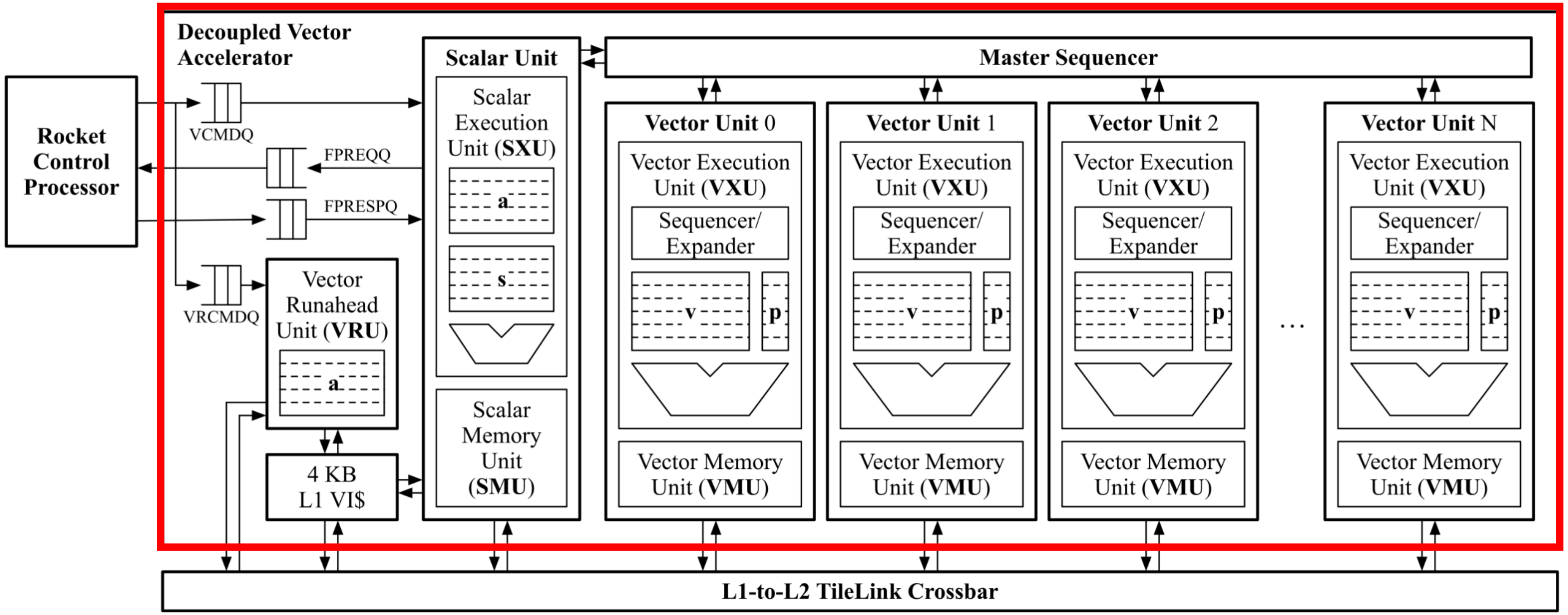
| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 2 | 2 | 2 | 3 | 4 | 3 |

Data

| | | | | | | |
|---|---|---|---|---|---|---|
| 8 | 2 | 5 | 7 | 1 | 2 | 9 |

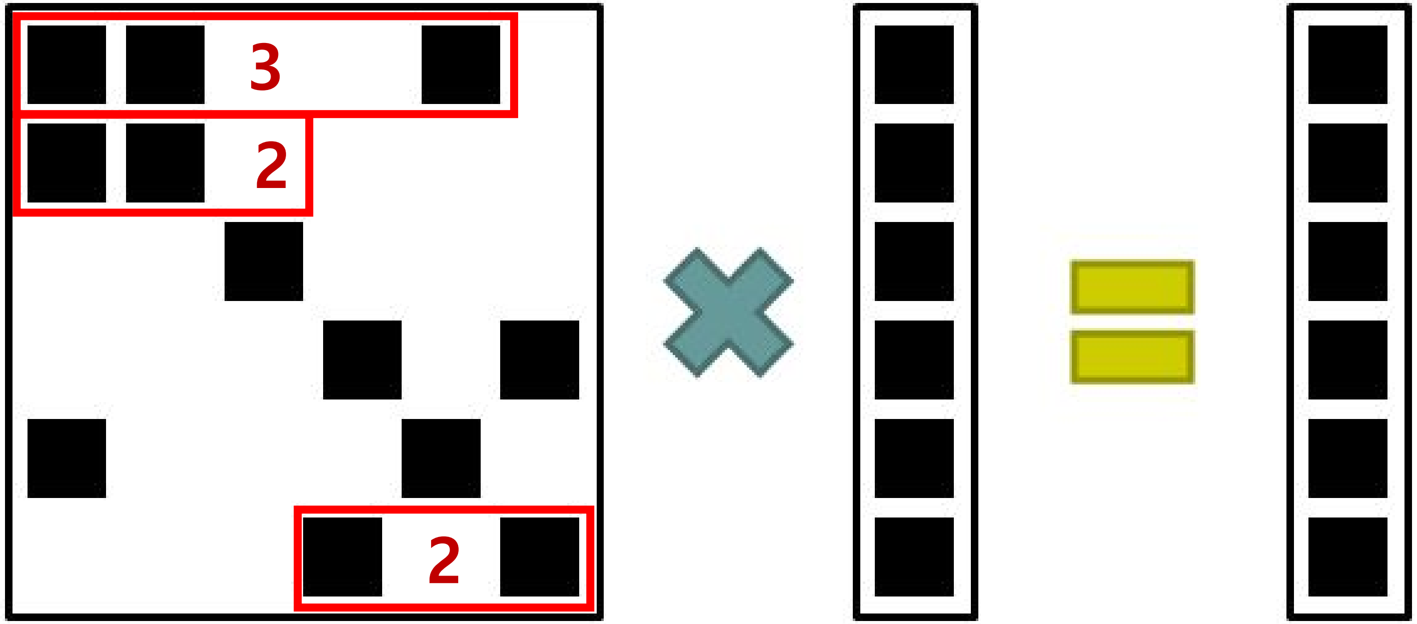
# Background: Vector processor (SIMD)

- **Single instruction multiple data (SIMD)**
  - Dedicated accelerator for vector processing inside processor
  - Compute arithmetic operations **in parallel using SIMD accelerator**
  - **# of Lane: # of arithmetic operations that can be computed in parallel**



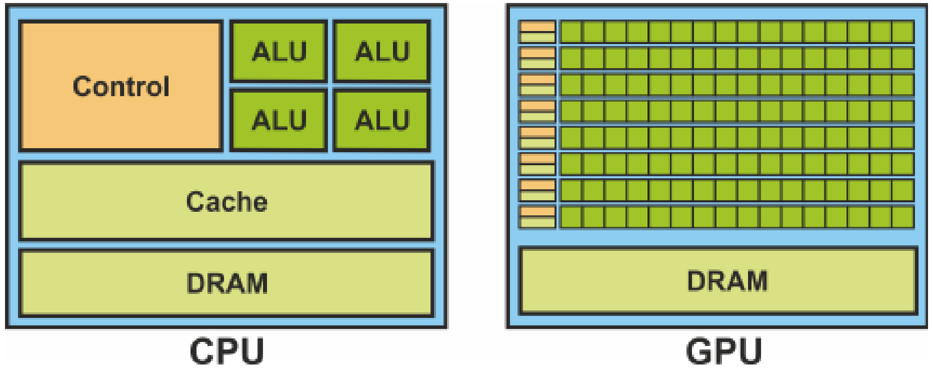
# Motivation: Vector processor (SIMD)

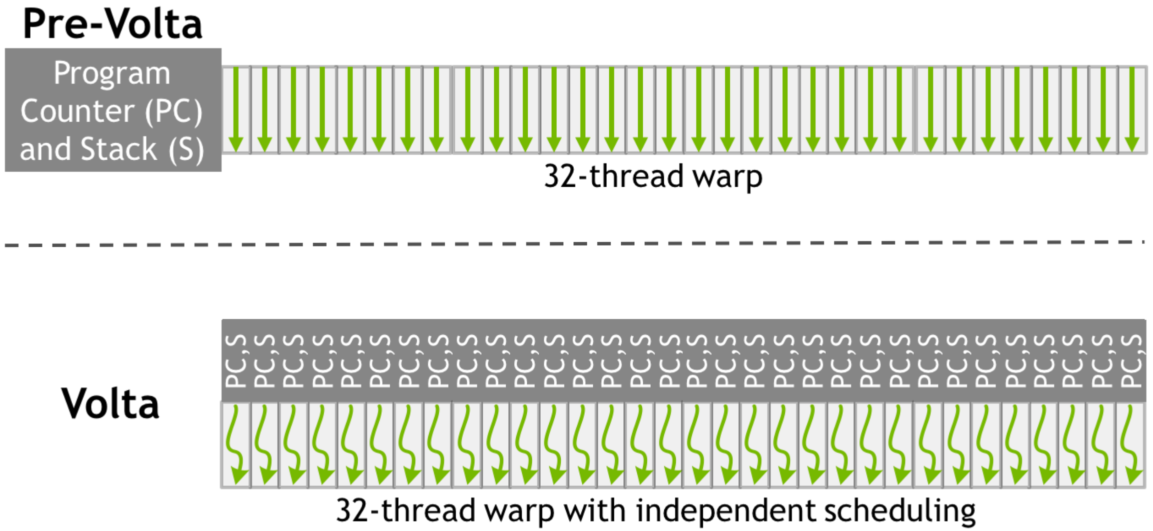
- **Problems in SIMD operations**
  - In the case of **sparse matrix-vector multiplication (SPMV)**
  - **Maximum vector length (# of lane) is 4**
    - Vector length of 3: 3 computations in parallel, but one lane is idle



# Background: Vector processor (SIMT)

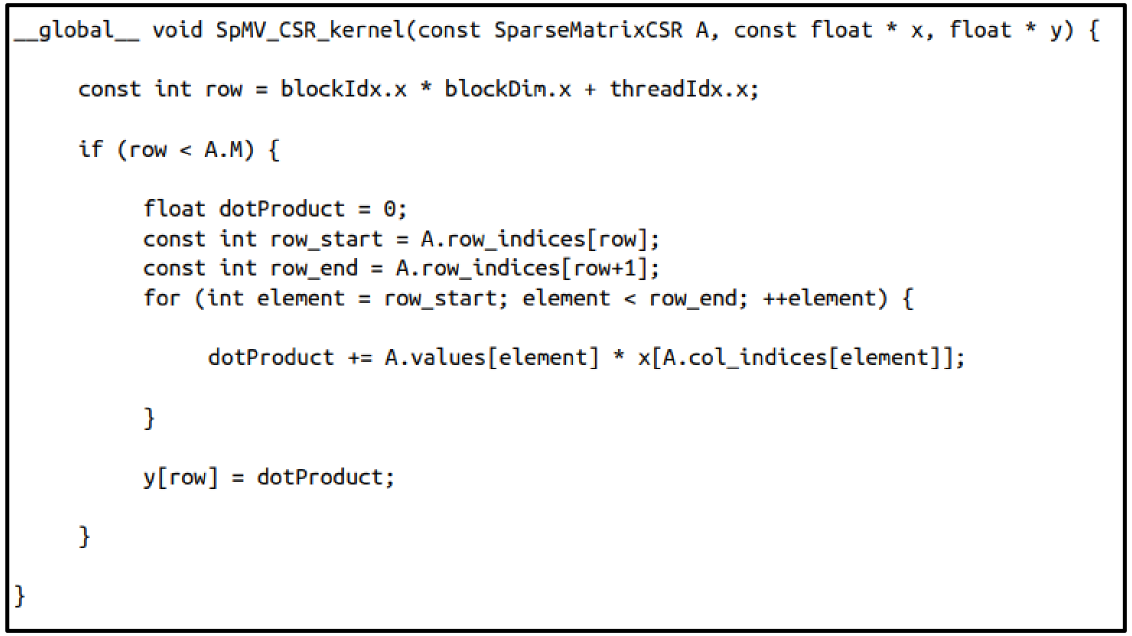
- **Single instruction multiple thread (SIMT)**
  - Graphic processing unit (GPU): **many core architecture**
  - CPU: 1~16 cores, GPU: more than 1,000 cores
  - Threads are grouped and controlled by group of thread (thread-block/work-group, warp/wave-front)



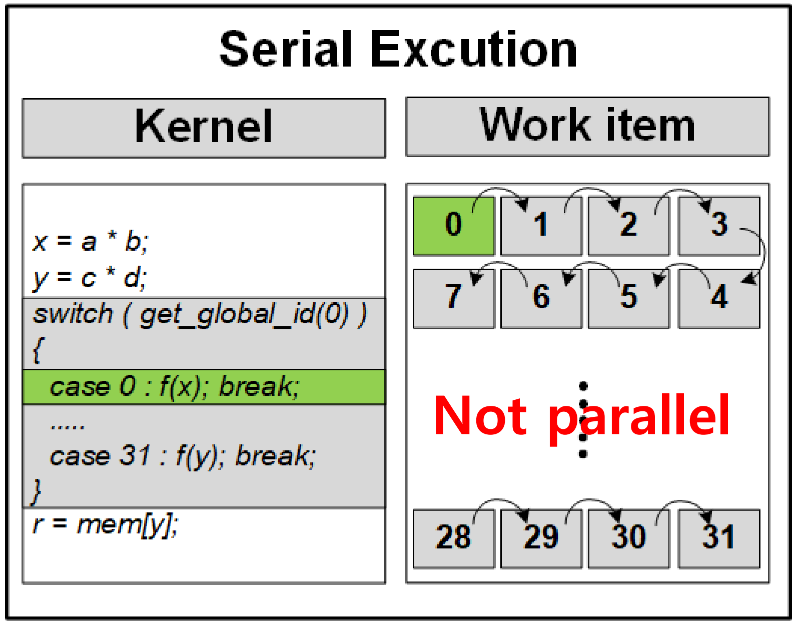

Pre-Volta
Program Counter (PC) and Stack (S)
32-thread warp
Volta
PC,S
32-thread warp with independent scheduling

# Motivation: Vector processor (SIMT)

- **Divergence branch in GPU**
  - Conditions (If-else, Case): serial execution by branch instruction, Idle state computing units (PEs)[4]
  - **Decoding makes additional condition loops**



```
__global__ void SpMV_CSR_kernel(const SparseMatrixCSR A, const float * x, float * y) {

    const int row = blockIdx.x * blockDim.x + threadIdx.x;

    if (row < A.M) {

        float dotProduct = 0;
        const int row_start = A.row_indices[row];
        const int row_end = A.row_indices[row+1];
        for (int element = row_start; element < row_end; ++element) {

            dotProduct += A.values[element] * x[A.col_indices[element]];

        }

        y[row] = dotProduct;

    }

}
```



Serial Excution

Kernel

```
x = a * b;
y = c * d;
switch ( get_global_id(0) )
{
  case 0 : f(x); break;
  .....
  case 31 : f(y); break;
}
r = mem[y];
```

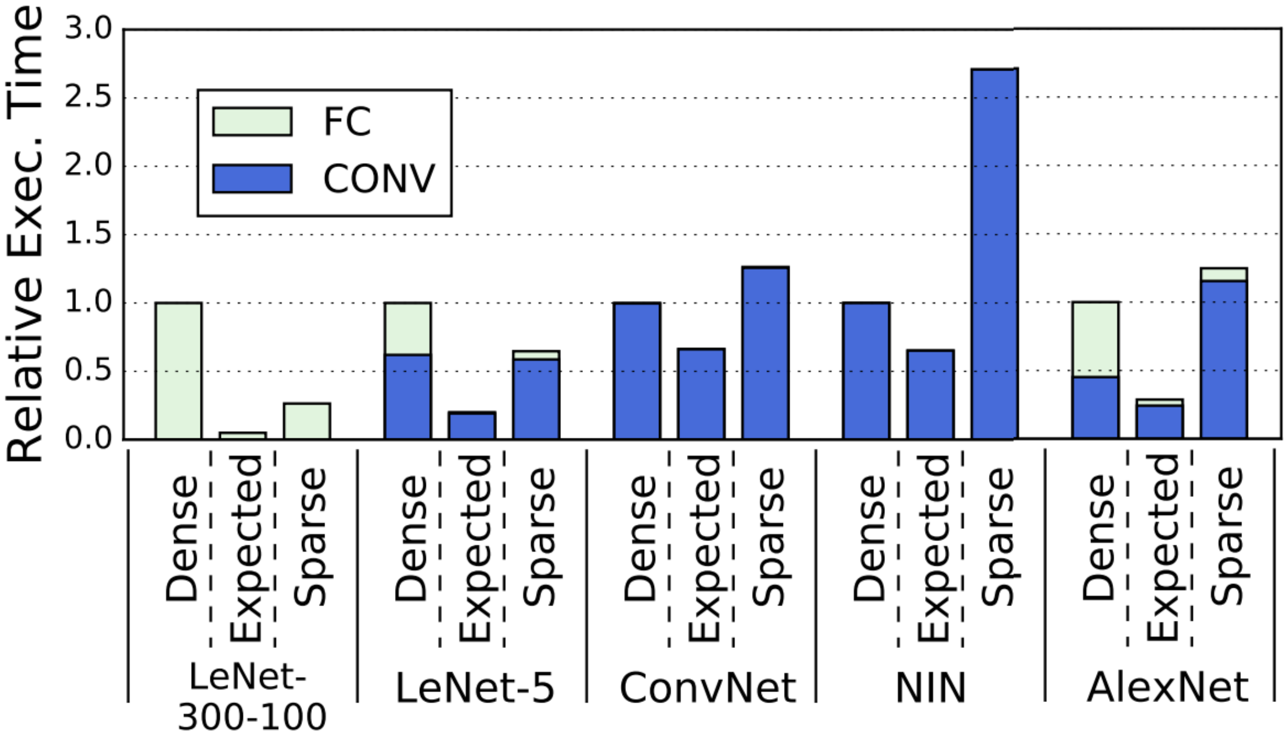
Work item

Not parallel

[4] Modified Convolution Neural Network for Highly Effective Parallel Processing, IEEE IRI 2017.

# Contributions

- **DNN weight pruning is not a panacea**
  - **Performance affected by both the structure network and data-parallel hardware**
  - Pruned network that is customized to hardware platform (CPU, GPU)
  - **Depend on hardware parallelism and layer type**, choose **SIMD-aware weight pruning** or **Node-pruning**



# Scalpel: Pruning technique

- **Customized DNN pruning for different hardware platform**
  - Two pruning techniques: SIMD-aware weight punning/Node pruning

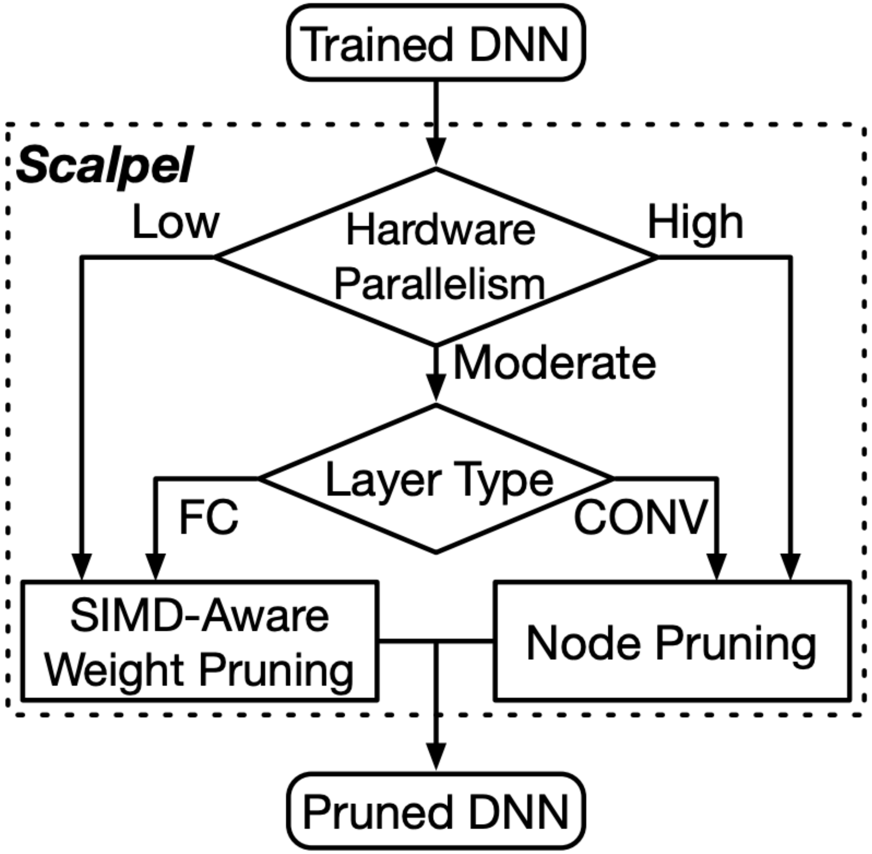
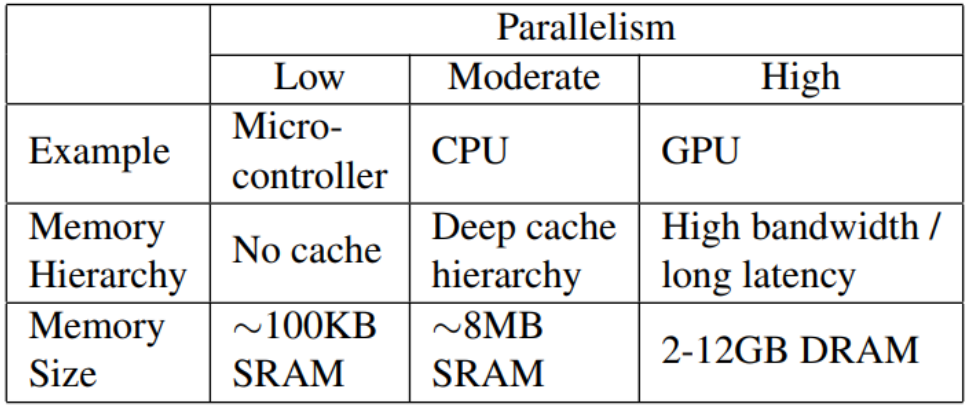


**Table 1: Hardware platforms with different parallelism.**



| | Parallelism | | |
|---|---|---|---|
| | Low | Moderate | High |
| Example | Micro-controller | CPU | GPU |
| Memory Hierarchy | No cache | Deep cache hierarchy | High bandwidth / long latency |
| Memory Size | ∼100KB SRAM | ∼8MB SRAM | 2-12GB DRAM |

# Scalpel: SIMD-aware weight pruning

- **Customized DNN pruning for different hardware platform**
  - Grouping weights by the number of SIMD lanes
  - Using root-mean-square (RMS), measure the importance of weight groups
  - Removing grouped weights under threshold and retraining
  - Layer by layer manner with dropout

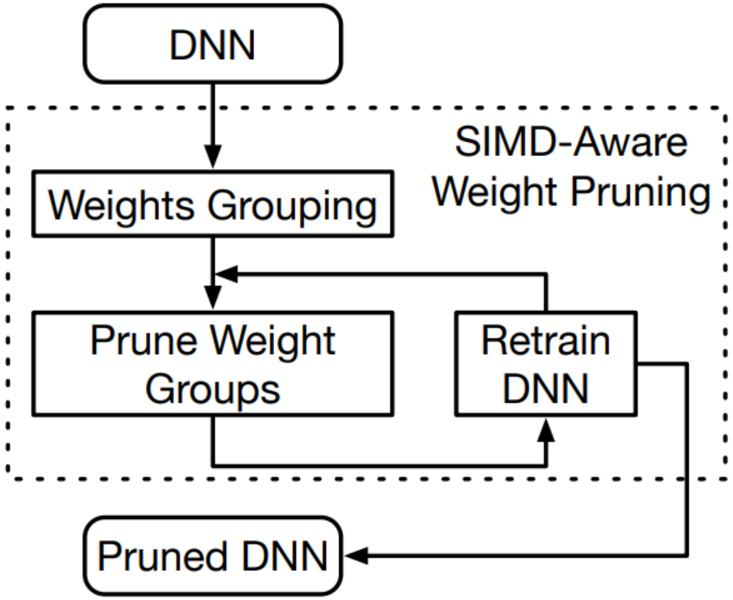
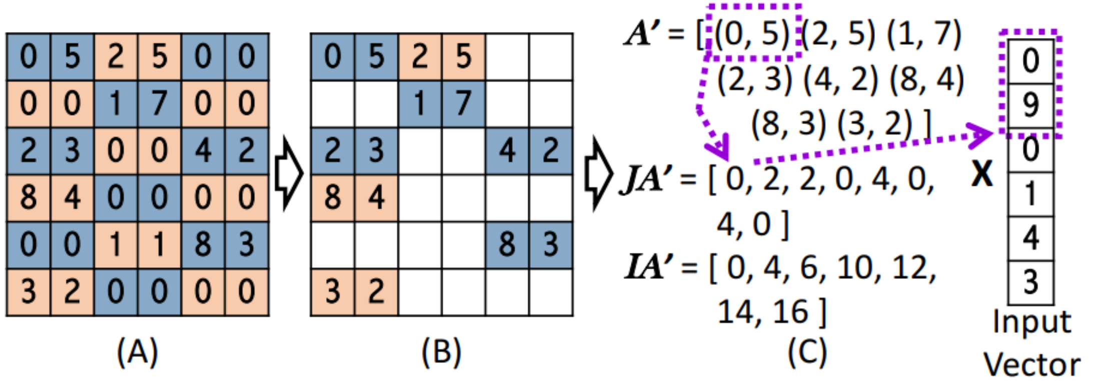


Figure 7: Main steps of SIMD-aware weight pruning.



|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 0 | 5 | 2 | 5 | 0 | 0 |
| 0 | 0 | 1 | 7 | 0 | 0 |
| 2 | 3 | 0 | 0 | 4 | 2 |
| 8 | 4 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 8 | 3 |
| 3 | 2 | 0 | 0 | 0 | 0 |

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 0 | 5 | 2 | 5 |  |  |
|  |  | 1 | 7 |  |  |
| 2 | 3 |  |  | 4 | 2 |
| 8 | 4 |  |  |  |  |
|  |  |  |  | 8 | 3 |
| 3 | 2 |  |  |  |  |

|   |
|---|
| 0 |
| 9 |
| 0 |
| 1 |
| 4 |
| 3 |

Figure 8: (A) Weights grouping; (B) Sparse weight matrix after pruning weight groups; (C) Modified CSR format for SIMD-aware weight pruning.

# Scalpel: Node pruning (=Channel pruning)

- **Hardware friendly pruning technique**
  - Sparse matrix multiplication requires more execution time compared to dense matrix
  - Using **mask layer** (Boolean ***α***): blocking one node (one neuron in FC, one feature map in convolution layer)
  - L1 regularization to control the number of nodes got removed
  - If the node is not important, ***β*** will be decreased to be lower that threshold ***T***

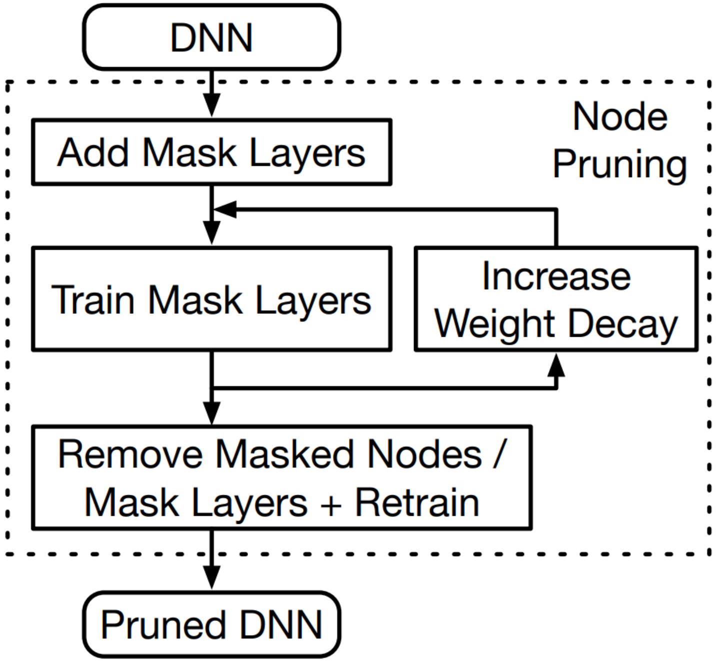


**Figure 11: Main steps of node pruning.**

In training iteration $k \geqslant 1$, $\alpha_i$ is calculated as

$$y'_i = \alpha_i \cdot y_i$$

$$R_{i,\,L1} = \lambda|\beta_i| = \lambda\beta_i$$

$$\alpha_i|_k = \begin{cases} 1, & T+\varepsilon \leqslant \beta_i|_k \\ \alpha_i|_{k-1}, & T \leqslant \beta_i|_k < T+\varepsilon \\ 0, & \beta_i|_k < T \end{cases}$$

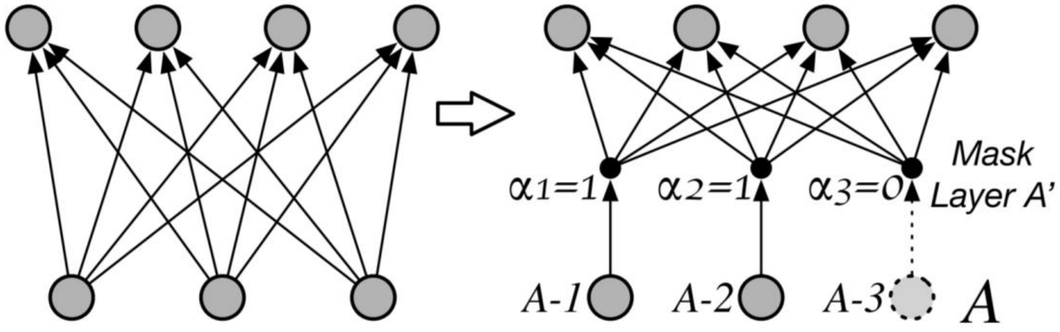


**Figure 12: Mask layers. Node *A-3* with $\alpha_3 = 0$ can be removed. The whole mask layer $A'$ will be removed after pruning all redundant nodes.**

# Evaluation: Hardware platform

- **Micro-controller unit (MCU): low parallelism**
  - ARM Cortex-M4: 3-stage in-order pipeline
  - **2-way SIMD unit** for 16b fixed-point
  - ARM's sparse matrix-vector BLAS
- **CPU: moderate parallelism**
  - Intel Core i7-6700 Skylake
  - 8-way SIMD ISA for 32b float
  - Math kernel library (MKL) CSRMV/CSRMM
- **GPU: high parallelism**
  - NVIDIA Titan X
  - cuSPARSAE CSRMV/CSRMM

# Evaluation: Speedup in CPU

- **Speedup in i7-6700 CPU running convolution and fully-connected**
  - SIMD-Sparse (SIMD-aware), Simple-Sparse (Sparse matrix), Intel's BLAS (MKL-Dense/MKL-Sparse)
  - **Conventional sparse methods** (MKL/Simple-Sparse) require more execution time compared to dense
  - **To achieve performance speedup with pruning, at least 79% of weights need to be removed**
  - But conv layers have much less redundancy compared fully-connected layer

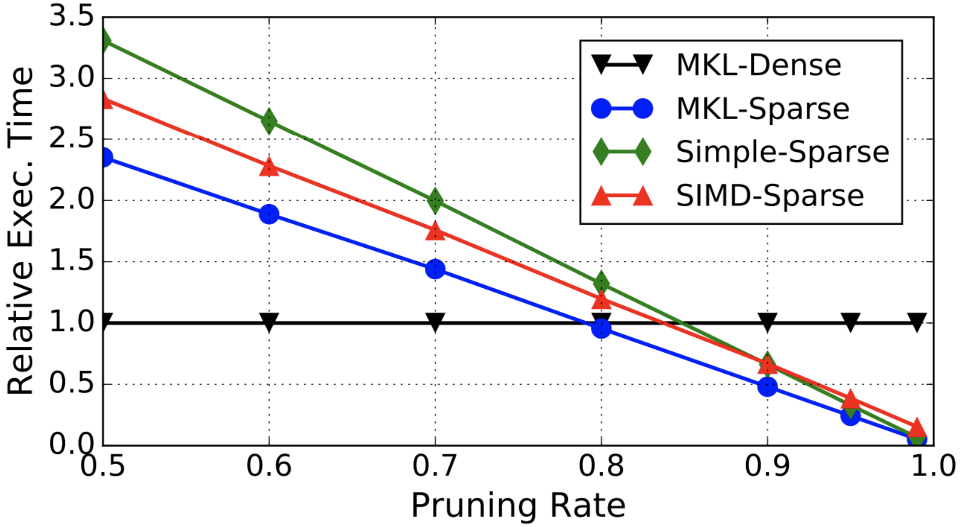


**Figure 14: Relative execution time for sparse matrix-matrix multiplication (CONV layers) on Intel Core i7-6700. The weight matrix and input matrix have the size of 128 x 1200 and 1200 x 729, respectively.**

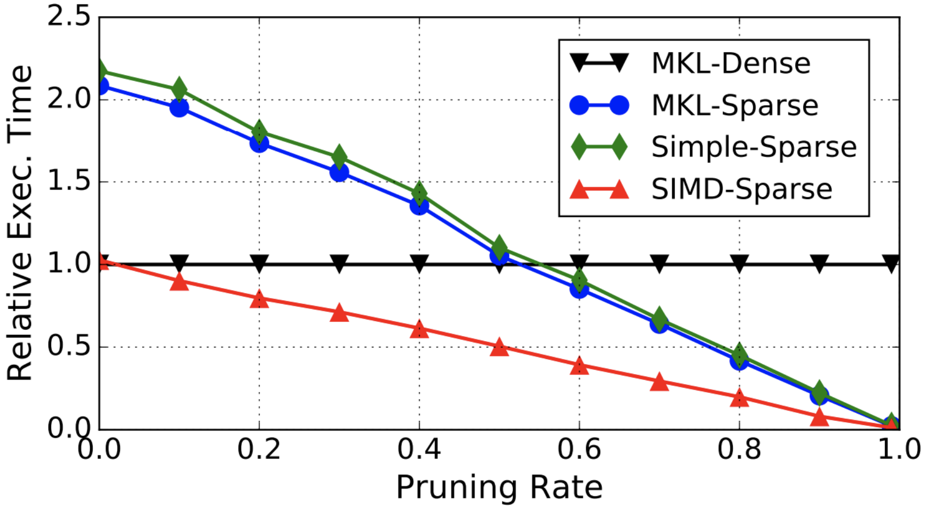


**Figure 13: Relative execution time for sparse matrix-vector multiplication (FC layers) on Intel Core i7-6700. The matrix size is 4096 x 4096 and the vector size is 4096. MKL-Dense/Sparse show the results of dense and sparse weight matrix with the Intel MKL library.**

# Evaluation: Model size in CPU

- **Improvement of speedup and model size**
  - Speedup can be up to **6.86x**, relative size is reduced by up to **94.8%**
  - **Non of the layer in NIN can have more than 50% of weights**
  - **Converting to sparse matrix is expected to require additional overhead.**

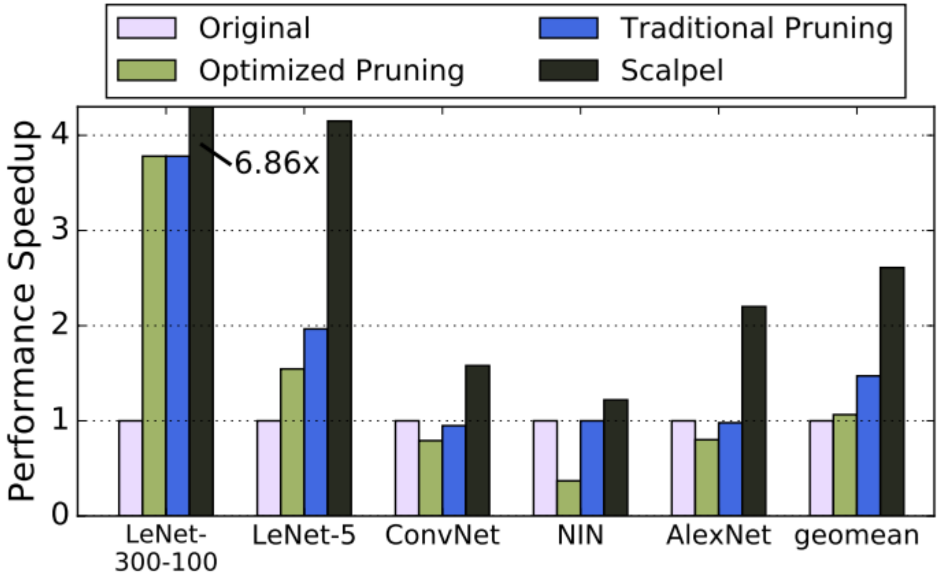


**Figure 18: Relative performance speedups of the original models, traditional pruning, optimized pruning and Scalpel on Intel Core i7-6700 CPU.**

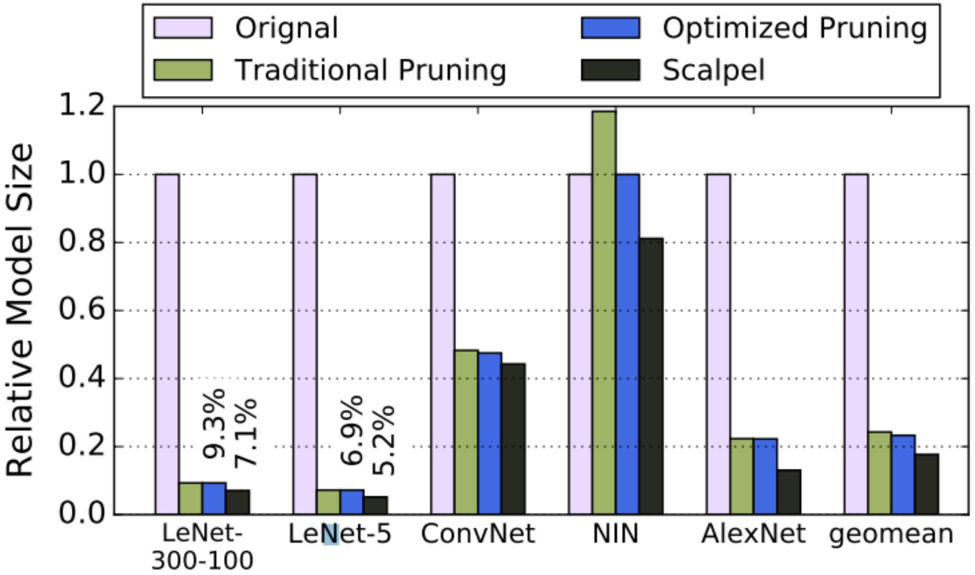
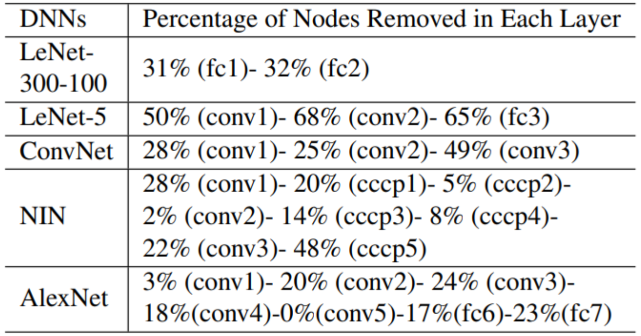


**Figure 19: Relative model sizes of the original models, traditional pruning, optimized pruning and Scalpel for Intel Core i7-6700 CPU.**

# Evaluation: High parallelism in GPU



| DNNs | Percentage of Nodes Removed in Each Layer |
|---|---|
| LeNet-300-100 | 31% (fc1)- 32% (fc2) |
| LeNet-5 | 50% (conv1)- 68% (conv2)- 65% (fc3) |
| ConvNet | 28% (conv1)- 25% (conv2)- 49% (conv3) |
| NIN | 28% (conv1)- 20% (cccp1)- 5% (cccp2)- 2% (conv2)- 14% (cccp3)- 8% (cccp4)- 22% (conv3)- 48% (cccp5) |
| AlexNet | 3% (conv1)- 20% (conv2)- 24% (conv3)- 18%(conv4)-0%(conv5)-17%(fc6)-23%(fc7) |

- **Improvement of speedup and model size**
  - LeNet-300-100 is very **tiny model**
  - **Removing DNN redundancy at a granularity of nodes to keep regular structure**

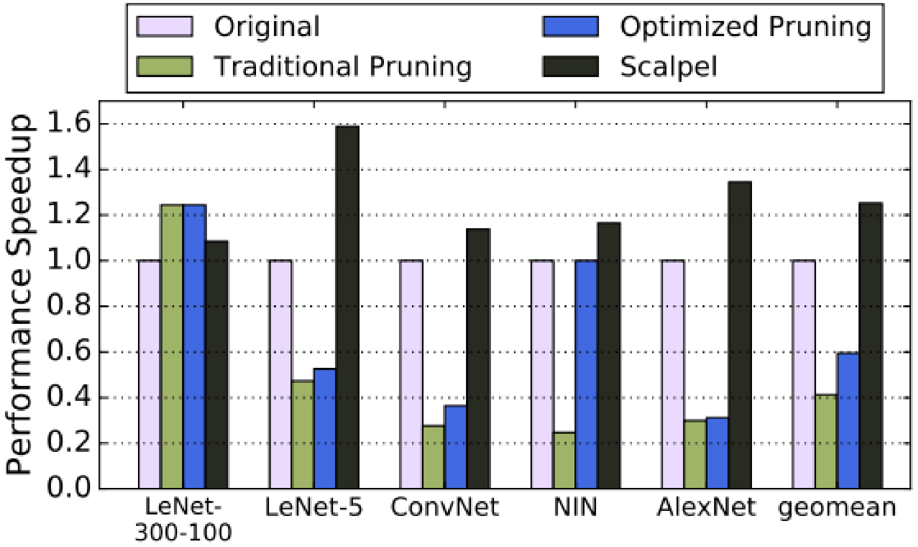


**Figure 20: Relative performance speedups of the original models, traditional pruning, optimized pruning and Scalpel on NVIDIA GTX Titan X GPU.**

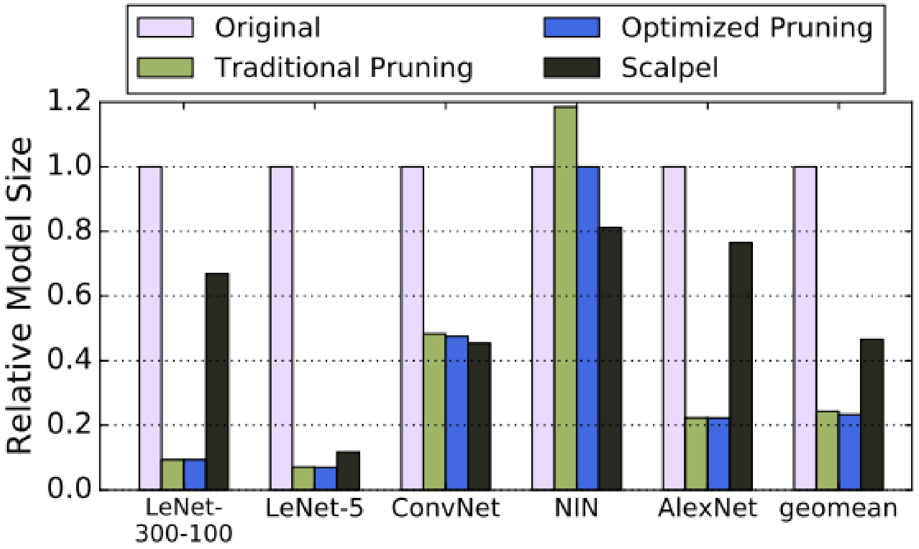


**Figure 21: Relative model sizes of the original models, traditional pruning, optimized pruning and Scalpel for NVIDIA GTX Titan X GPU.**

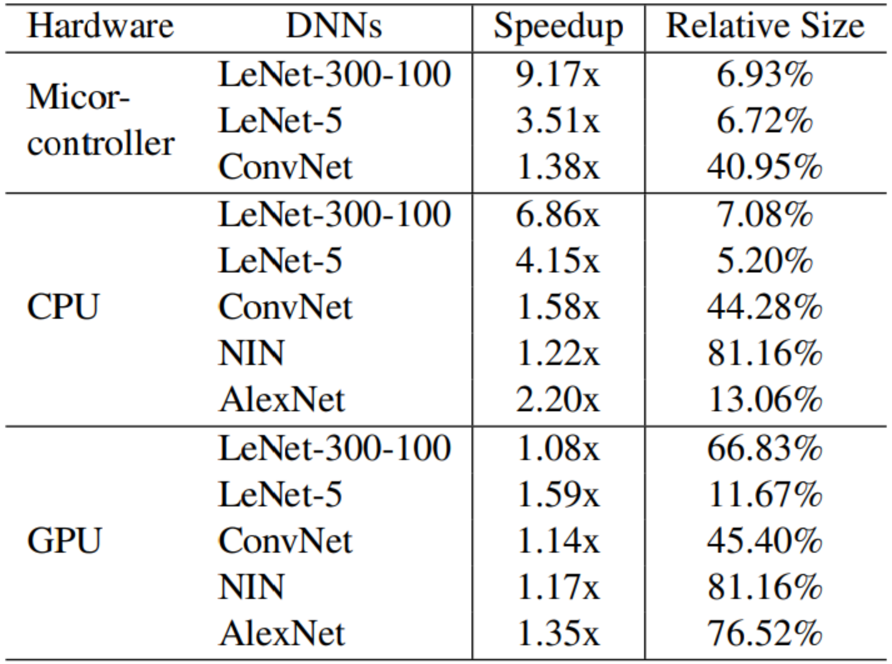
# Evaluation: Compared to STOA

- **Performance close to Deep Compression[1]**
  - **Deep Compression**: Pruning + Quantization + Lookup table
  - **Scalpel: 32b** floating-point + Pruning
  - In the case of AlexNet
    - Model size: Deep Compression (13.06%), Scalpel (11%)
    - Speedup: Deep Compression (3x), Scalpel (2.2x)

In Deep Compression

| DNNs | Relative Size |
|---|---|
| LeNet-300-100 | 8% |
| LeNet-5 | 8% |
| AlexNet | 11% |
| VGG-16 | 7.50% |

Table 3: Results overview.



| Hardware | DNNs | Speedup | Relative Size |
|---|---|---|---|
| Micor-controller | LeNet-300-100 | 9.17x | 6.93% |
| | LeNet-5 | 3.51x | 6.72% |
| | ConvNet | 1.38x | 40.95% |
| CPU | LeNet-300-100 | 6.86x | 7.08% |
| | LeNet-5 | 4.15x | 5.20% |
| | ConvNet | 1.58x | 44.28% |
| | NIN | 1.22x | 81.16% |
| | AlexNet | 2.20x | 13.06% |
| GPU | LeNet-300-100 | 1.08x | 66.83% |
| | LeNet-5 | 1.59x | 11.67% |
| | ConvNet | 1.14x | 45.40% |
| | NIN | 1.17x | 81.16% |
| | AlexNet | 1.35x | 76.52% |

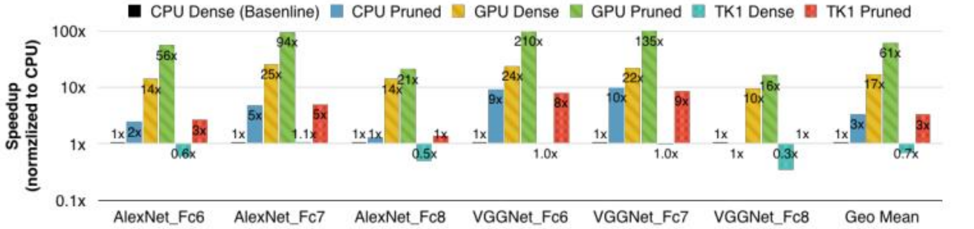


Figure 9: Compared with the original network, pruned network layer achieved 3× speedup on CPU, 3.5× on GPU and 4.2× on mobile GPU on average. Batch size = 1 targeting real time processing. Performance number normalized to CPU.

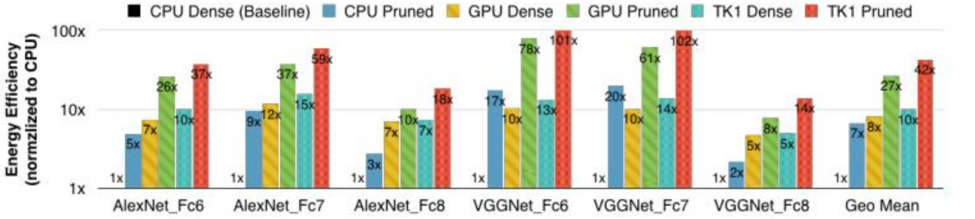


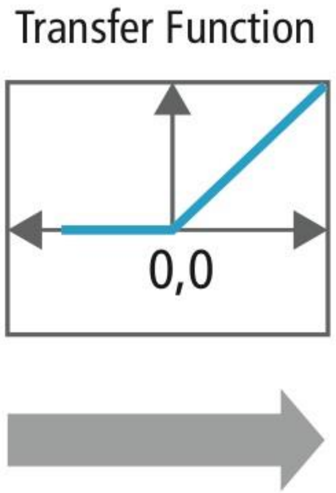
Figure 10: Compared with the original network, pruned network layer takes 7× less energy on CPU, 3.3× less on GPU and 4.2× less on mobile GPU on average. Batch size = 1 targeting real time processing. Energy number normalized to CPU.

# Conclusion

- **Scalpel to customize DNN pruning for different hardware platform**
  - SIMD-aware weight pruning, Node pruning
  - On MCU, CPU, and GPU
    - 3.54x, 2.61x, and 1.25x of speedup
    - 88%, 82%, 53% of reduced model size

- **Potential problem**
  - Activation pruning when **zero** value

|  |  |  |  |
|---|---|---|---|
| 15 | 20 | -10 | 35 |
| 18 | -110 | 25 | 100 |
| 20 | -15 | 25 | -10 |
| 101 | 75 | 18 | 23 |



|  |  |  |  |
|---|---|---|---|
| 15 | 20 | 0 | 35 |
| 18 | 0 | 25 | 100 |
| 20 | 0 | 25 | 0 |
| 101 | 75 | 18 | 23 |