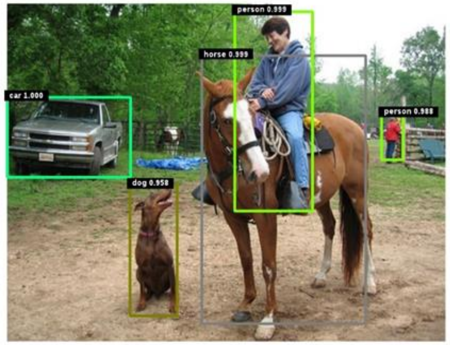
SPEED-UP OF OBJECT DETECTION NEURAL NETWORK WITH GPU

Takuya Fukagai[1], Kyosuke Maeda[2], Satoshi Tanabe[1], Koichi Shirahata[1], Yasumoto Tomita[1], Atsushi Ike[1], Akira Nakagawa[1]
1 FUJITSU LABORATORIES LTD.
2 FUJITSU SOFTWARE TECHNOLOGIES LTD.

Background

- Object detection is one of the most useful and basic applications of deep neural networks
  - NN-based methods achieved the highest scores in the competitions such as ILSVRC and COCO
  - Various detection networks have been proposed
    - Faster R-CNN, R-FCN, YOLO, SSD etc.
  - High computational complexity



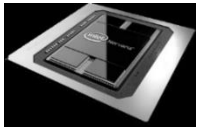
- Accelerators for fast neural network processing
  - Highly efficient processing
    - Domain-specific architecture
    - Many cores, specialized cores for NN
    - High memory bandwidth

GPU (NVIDIA)

DLU (FUJITSU)

TPU (Google)

Nervana NNP (Intel)



Fast object detection network processing with NN accelerators

Related Work

FUJITSU

- Many algorithms have been proposed to speed-up the calculation of convolution and fully-connected layers in CNN
  - Fast convolution algorithms such as Winograd [2016 Lavin et al.], FFT [2014 Mathieu et al.], summed area table [2017 Kasagi et al.]
  - NN compression algorithms such as Column weight pruning [2017 Wang et al.]
- Lightweight object detection networks (PVANet [2016 Kim et al.])
  - Speed up by redesigning CNN architecture feature extraction part
    - Less channels with more layers, adoption of concatenated ReLU, Inception, HyperNet [Kong et al. 2016], batch normalization, residual connections

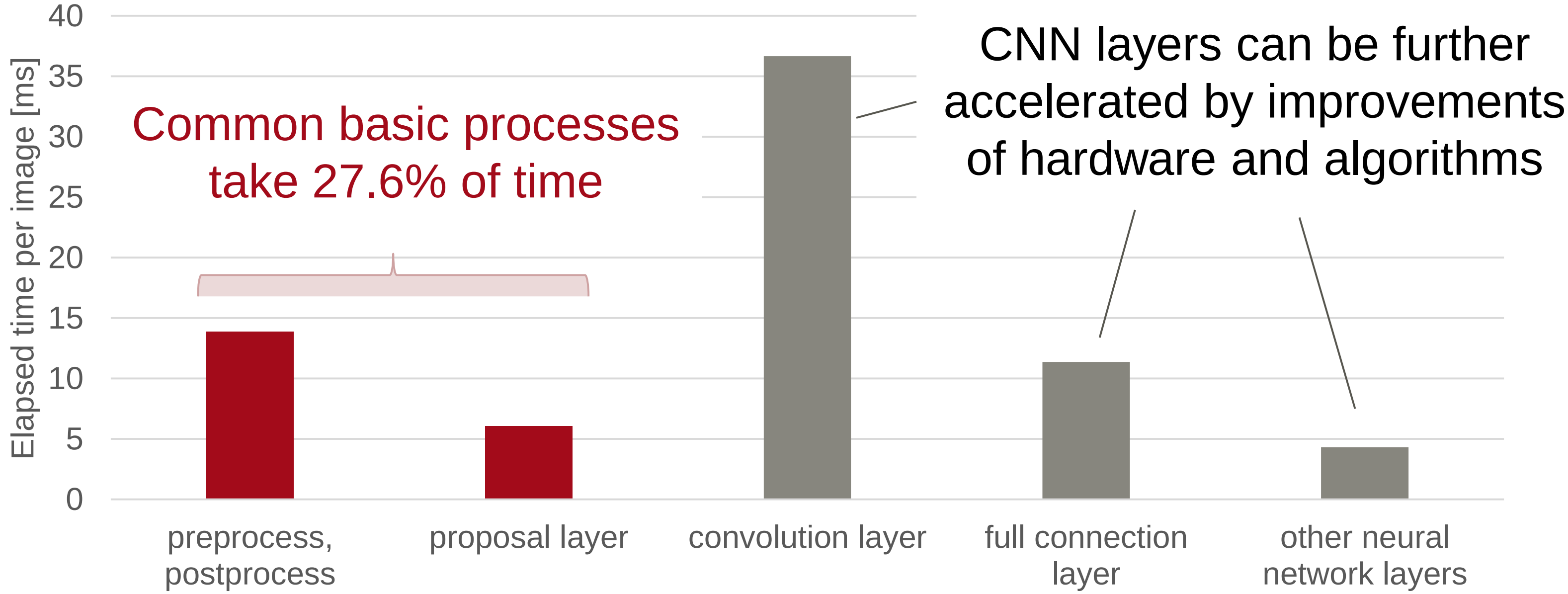
→ CNN feature extraction part in object detection networks has been accelerated

Existing works focus on fast computation of CNN layers

Problem

FUJITSU

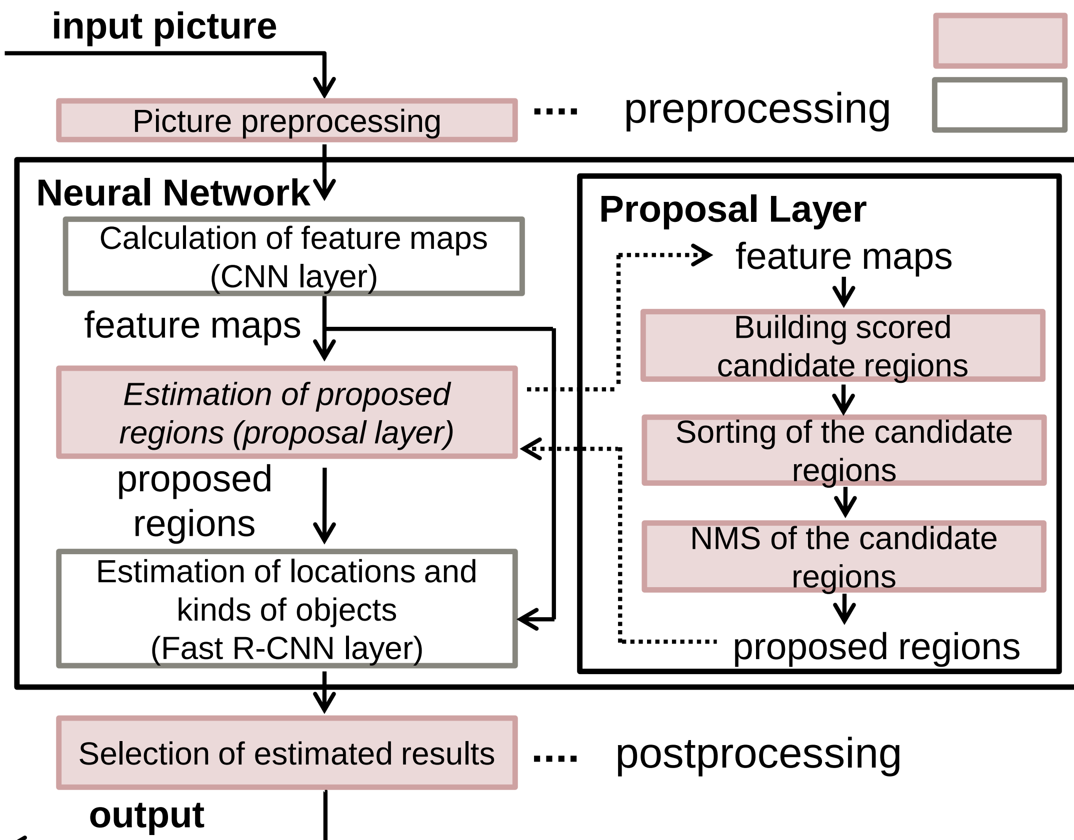
- In detection networks, not only convolution and fully-connected layers but also the other processes require fair amount of time
  - Our evaluation with existing Faster R-CNN implementation (py-faster-rcnn) shows 27.6% of time is used for outside CNN feature extraction
  - These are the common basic processes of detection networks such as Faster R-CNN, R-FCN, YOLO, and SSD



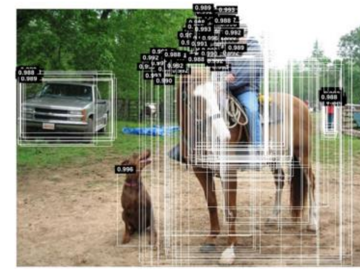
Speed-up of common basic processes becomes more important

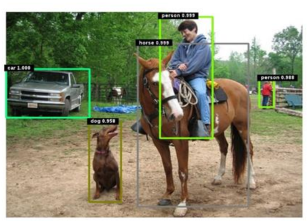
Faster R-CNN Architecture

- The common basic processes are executed on CPU
  - preprocessing, proposal layer, and postprocessing



Executed with CPU
Executed with GPU





We speed up the common basic processes with GPU

Proposal

- We propose speed up methods for the common basic processes of the detection networks with GPU
  - **We implement the common basic processes with GPU and assign a thread for each element to utilize many cores of GPU**
    - Fuse multiple GPU functions (CUDA kernels) to improve memory locality
    - Avoid CPU-GPU data transfer during the common basic processes
  - **We design and implement a high speed parallel sorting and a Non-Maximum-Suppression (NMS) with GPU**
    - We design an efficient sort algorithm for sorting candidate regions
    - Improve existing GPU-based NMS by skipping unnecessary calculation

- Result
  - Our GPU-accelerated Faster R-CNN processed in 55.2ms per image
  - 25.5% speed-up compared to py-faster-rcnn in whole time

Preprocessing with GPU

- Resize input pictures and subtract average RGB values
  - A thread is assigned for each output pixel
  - We process them in a single GPU function (CUDA kernel)

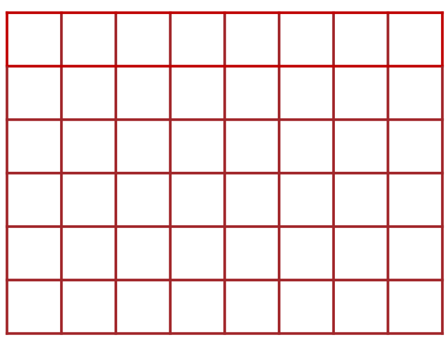
Example Input Image
500 x 375 pixel

Resize input pictures

600 x 800 pixel

Subtract average RGB values

Preprocessed Image
600 x 800 pixel



A thread for each output pixel

600 x 800 threads

Postprocessing with GPU

- Build scored candidates of detected results from network outputs, and applies NMS
  - A thread is assigned for each candidate region
  - We process in a single CUDA kernel for each part

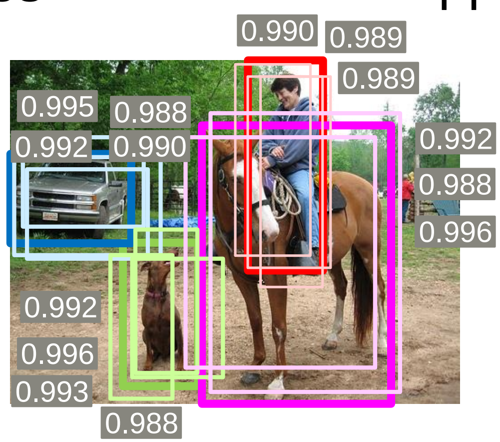
Building scored candidates

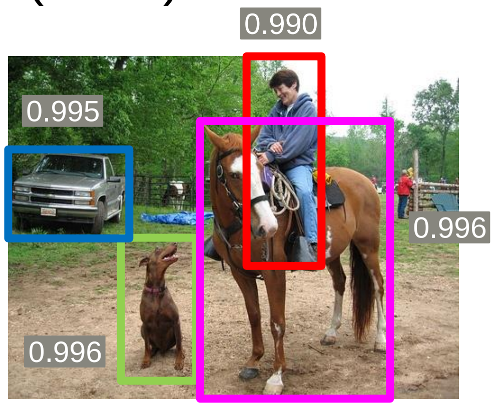
Non-Maximum-Suppression (NMS)

Proposed regions

Differences to be added to proposed regions

Scores for regions





Process in a single CUDA kernel

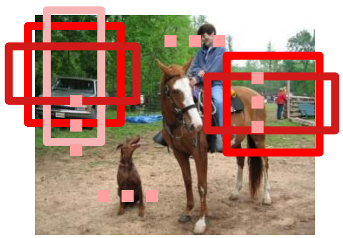
Process in a single CUDA kernel

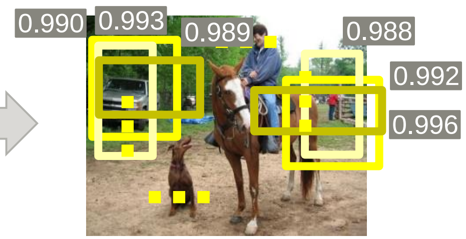
Proposal Layer with GPU

- Propose rectangular regions where objects are likely to exist
  - A thread is assigned for each element (anchor or candidate region)
  - We process each part in one or two kernels

Building scored candidate regions

Fixed anchor regions





Candidate regions

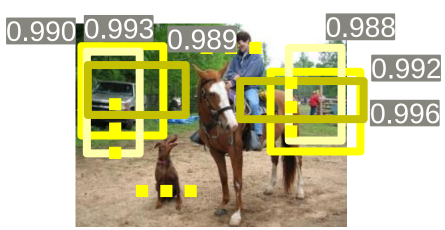
Sorting of candidate regions

Sort scored candidate regions and get top 1,000 - 6,000 regions

NMS of candidate regions

Select high score regions and suppress overlapping regions

Candidate regions



Proposed regions

Proposal Layer with GPU

- Propose rectangular regions where objects are likely to exist
  - A thread is assigned for each element (anchor or candidate region)
  - We process each part in one or two kernels

Building scored candidate regions

We design and implement a high speed parallel sorting and a Non-Maximum-Suppression (NMS) with GPU

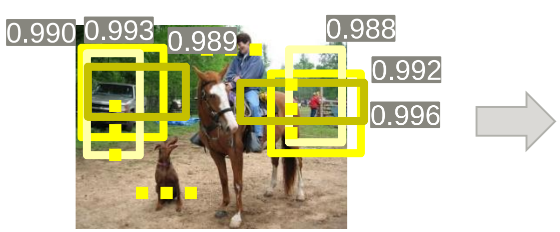
Sorting of candidate regions

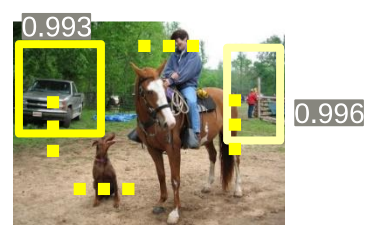
Sort scored candidate regions and get top 1,000 - 6,000 regions

NMS of candidate regions

Select high score regions and suppress overlapping regions

Candidate regions

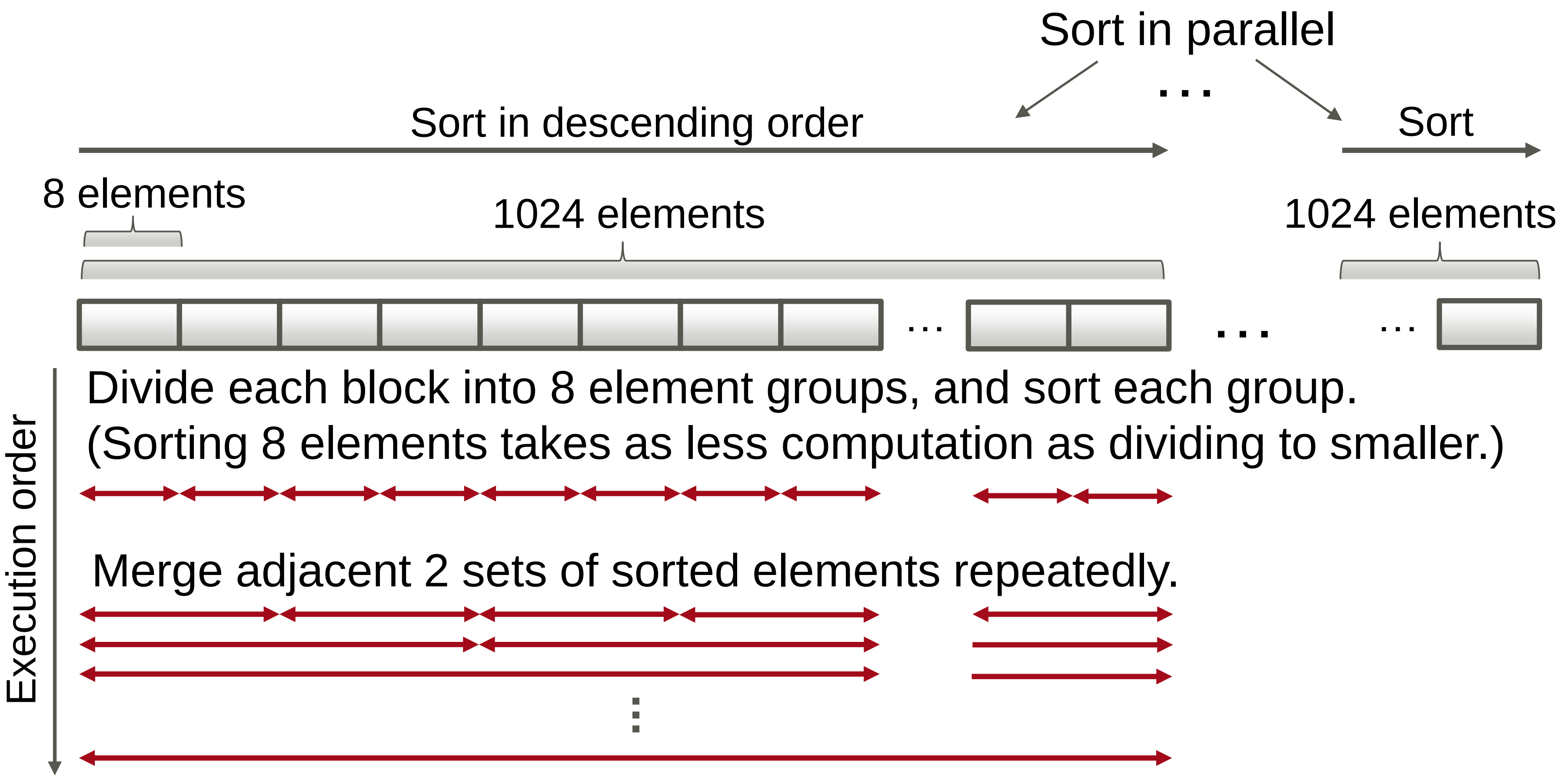




Proposed regions

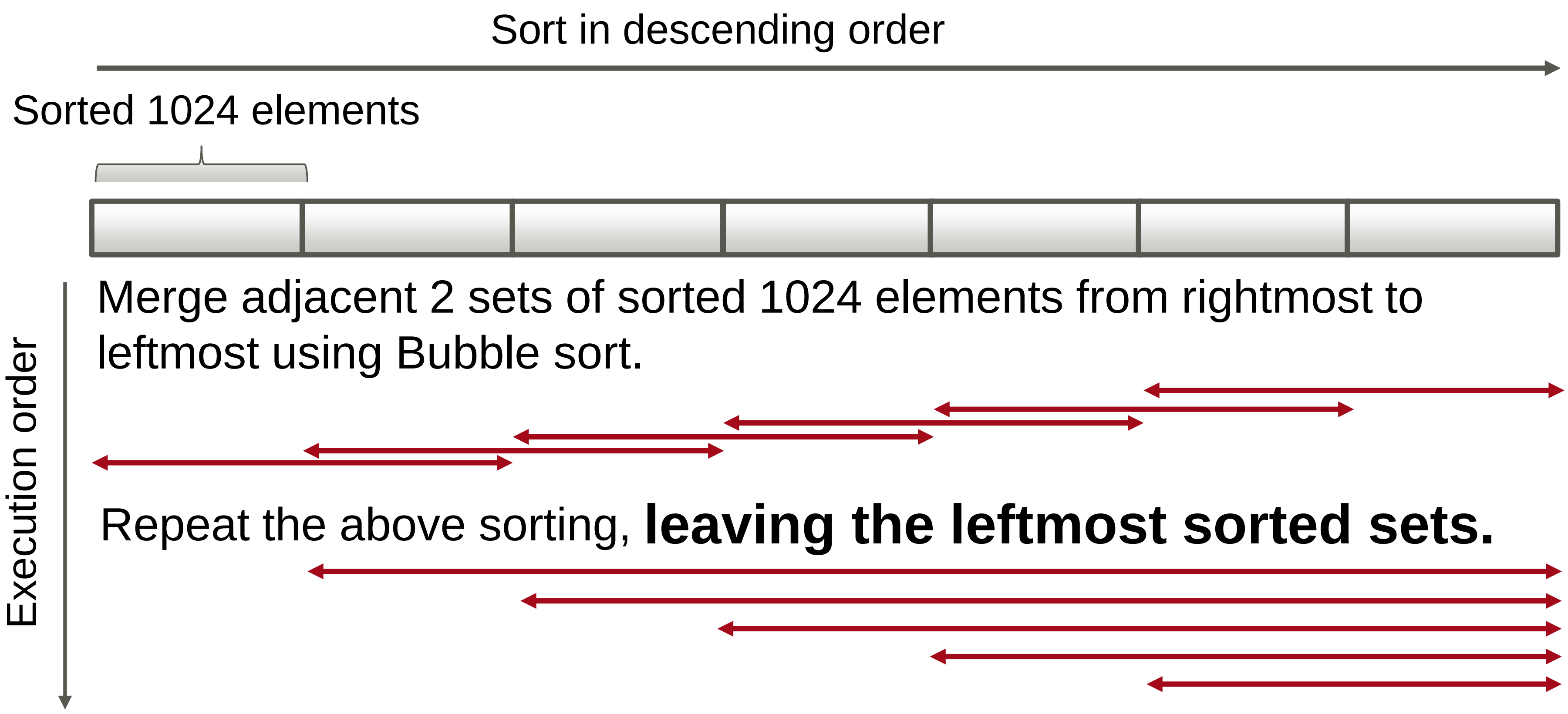
Our GPU Sorting of Candidate Regions

- **Step 1**：Make sorted blocks of 1024 elements
  - The maximum number of threads in a thread block is 1024.
  - Multiple blocks are computed in parallel with multiple thread blocks



Our GPU Sorting of Candidate Regions

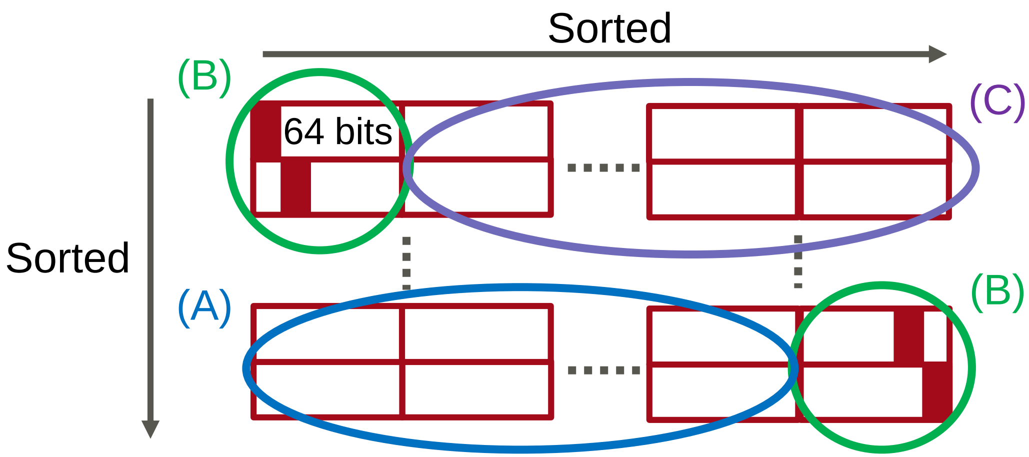
- **Step 1**：Make sorted blocks of 1024 elements
- **Step 2**：Gather top sorted elements

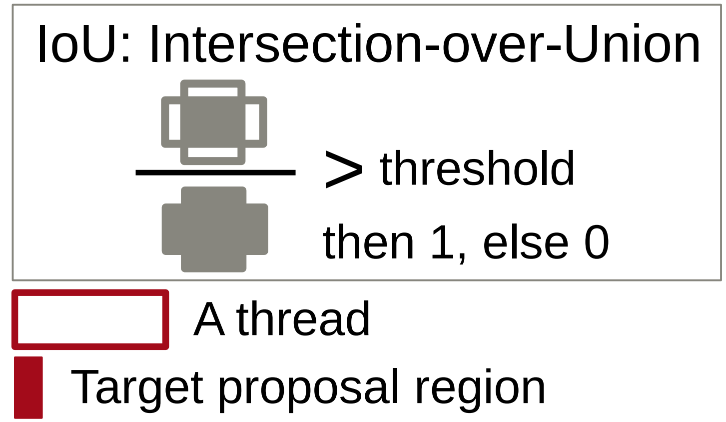


Reduce calculation by sorting only top elements

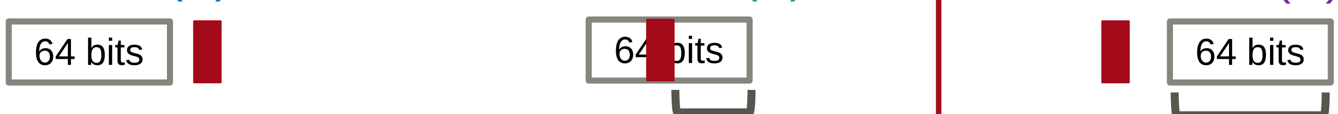
Our GPU Non-Maximum-Suppression

- Evaluate IoU in order to remove overlapping regions
  - We assign a thread for each 64 bit mask (64 bit unsigned integer type).
  - We categorize the threads into 3 patterns, and evaluate IoU if needed.





Pattern (A) | Pattern (B) | Pattern (C)



Pattern (A): All bits are set to 0 without evaluation

Pattern (B): Regions with lower scores are evaluated

Pattern (C): All areas are evaluated

Skip unnecessary calculations

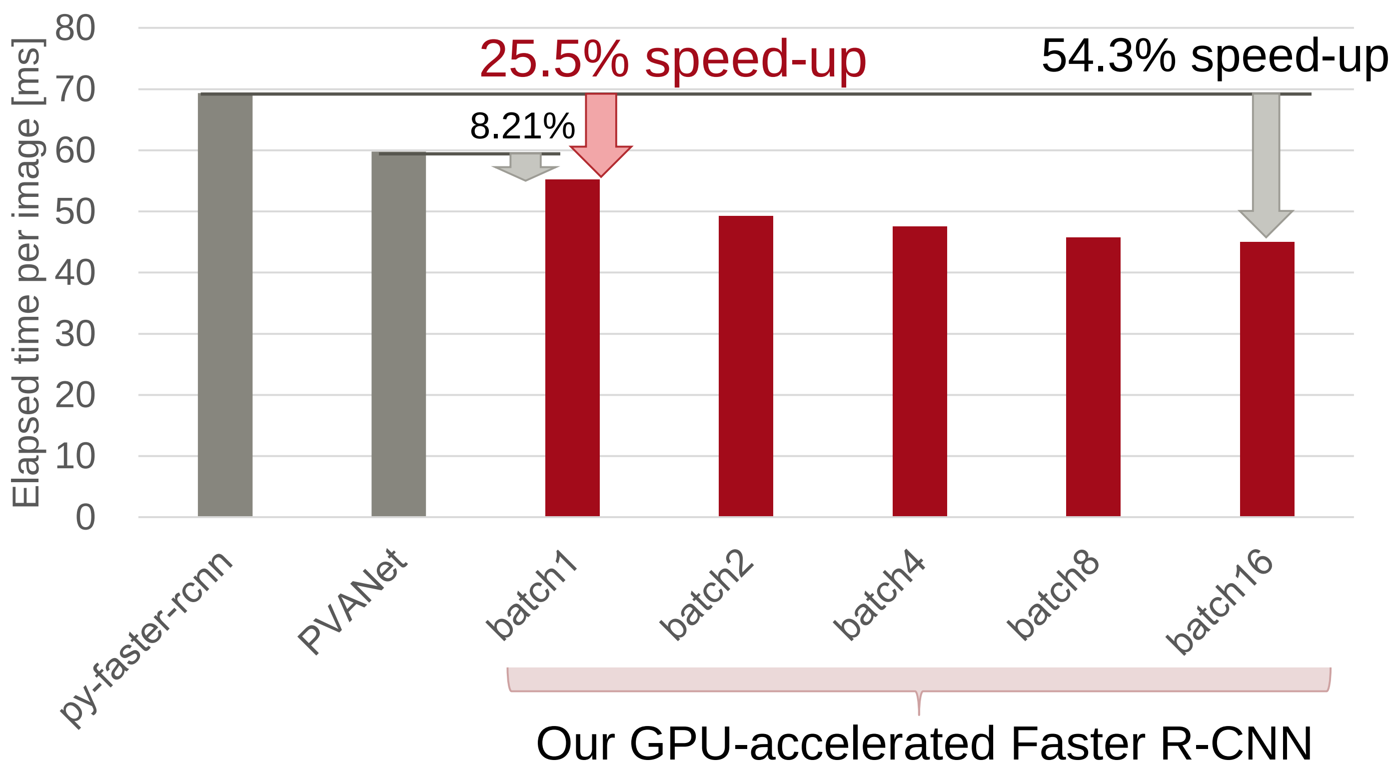
Experiment

- We measure whole cycle time between py-faster-rcnn, PVANet, and our GPU-accelerated Faster R-CNN
  - We implement the inference phase of Faster R-CNN in CUDA
  - Select 4096 images of 500 x 375 pixels from PASCAL VOC 2007
  - Use VGG16 as base CNN for all the implementations
- Measurement method
  - Since there was a difference in configurations between py-faster-rcnn and the others in our paper, we adjusted the configuration and measured elapsed time of the implementations again with the same configuration
  - We measured elapsed time 5 times and show results of the worst values
  - We calculate speed-up ratio by 100 x (ET of original) / (ET of proposal)
    - ET: elapsed time

CPU	2x Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz
GPU	1x Tesla P100-PCIE-16GB
OS	Ubuntu 14.04.5 LTS (GNU/Linux 4.2.0-42-generic x86 64)
Libraries	MKL (v20170003), CUDA 8.0, cuDNN v5.1

Results: Whole Cycle Time

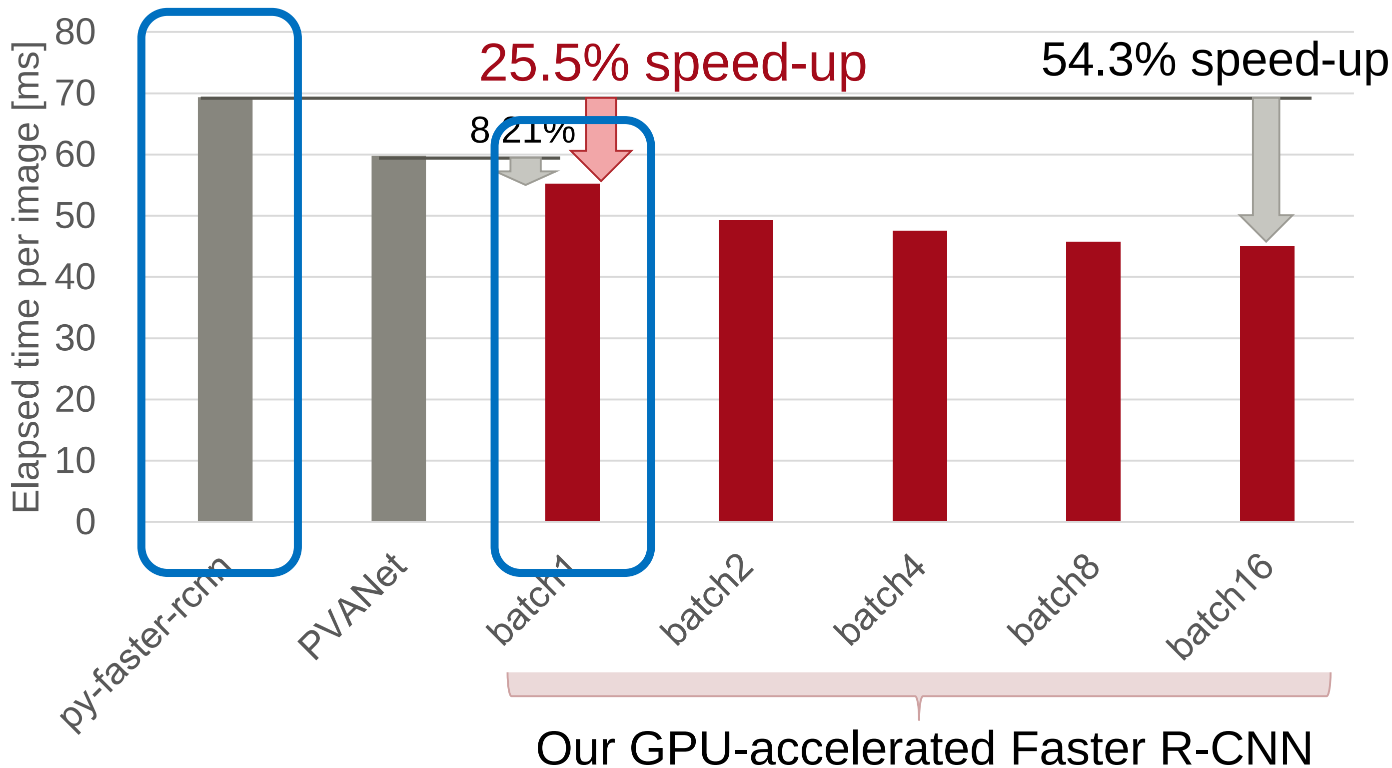
- Our GPU-accelerated Faster R-CNN processed in 55.2ms per image (25.5% speed-up with batch size 1)
  - 8.21% faster compared to PVANet with VGG16
  - Further speed-up is obtained by increasing batch size: 54.3% speed-up with batch size 16



Results: Whole Cycle Time

- Our GPU-accelerated Faster R-CNN processed in 55.2ms per image (25.5% speed-up with batch size 1)
  - 8.21% faster compared to PVANet with VGG16
  - -up

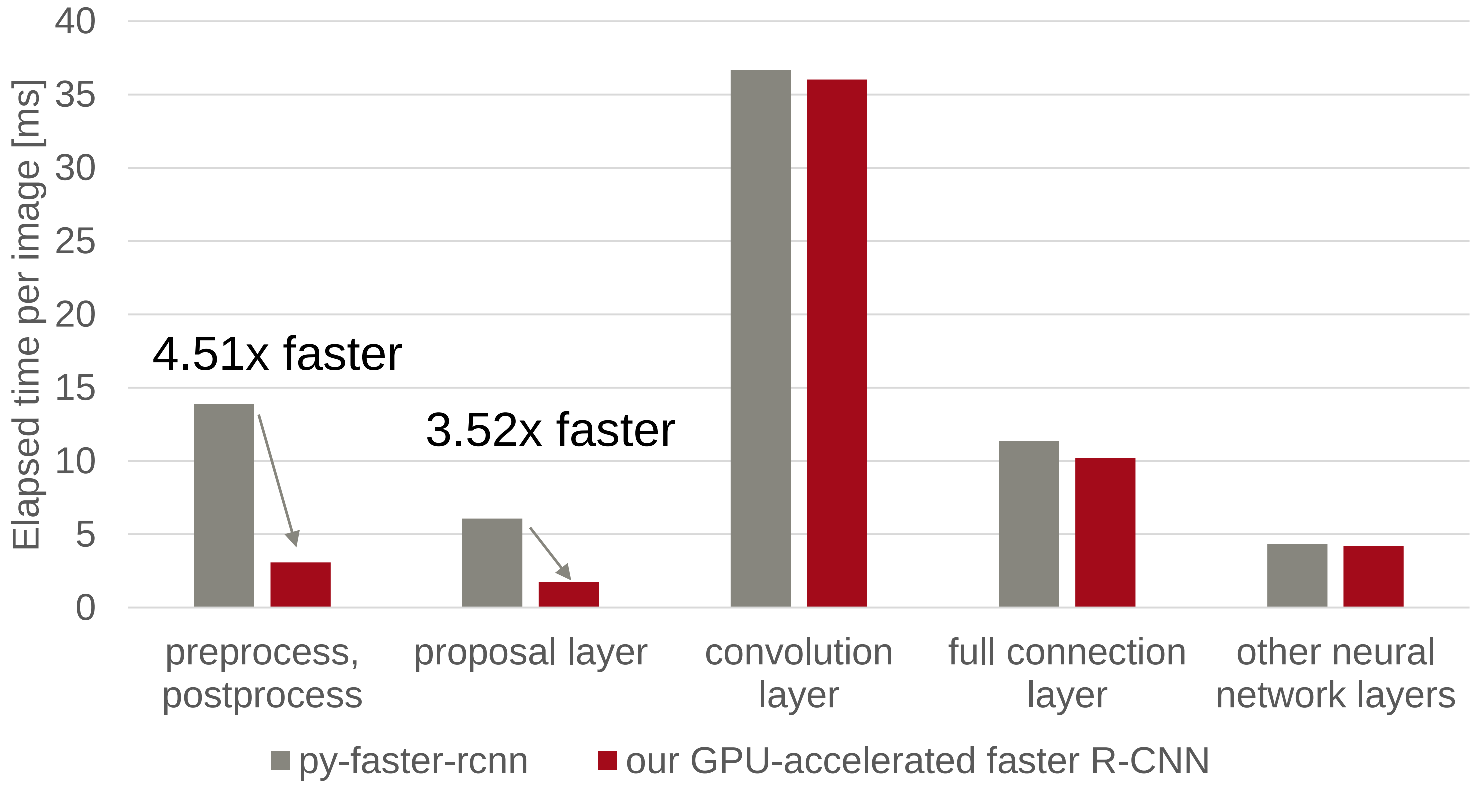
We show breakdown of py-faster-rcnn and Our GPU-accelerated Faster R-CNN with batch size 1



Results: Breakdown

FUJITSU

- Our GPU-accelerated Faster R-CNN outperformed py-faster-rcnn by 4.51x in preprocess plus postprocess, and 3.52x in proposal layer



We confirm speed-up of the common basic processes

Conclusions

- We propose speed-up methods for Faster R-CNN with GPU
  - We realized a speed-up of the common basic processes in object detection networks
  - Our speed-up methods are applicable to other detection networks such as R-FCN, YOLO, and SSD
- We evaluate the speed-up of Faster R-CNN by comparing with py-faster-rcnn
  - Our GPU-accelerated Faster R-CNN processed in 55.2ms per image: 25.5% speed-up compared to py-faster-rcnn
  - We expect to observe more significant speed-up when we apply our methods to the network with less convolution and fully-connected layers
- Future work
  - Apply our GPU-based parallel processing methods to other object detection neural networks such as R-FCN, SSD, YOLO etc. and evaluate their effectiveness

shaping tomorrow with you