Delayed Weight Update for Faster Convergence in Data-parallel Deep Learning

Tetsuya Youkawa, Haruki Mori, Yuki Miyauchi,
Kazuki Yamada, Shintaro Izumi,
Masahiko Yoshimoto, and Hiroshi Kawaguchi

Kobe University, Kobe, Japan

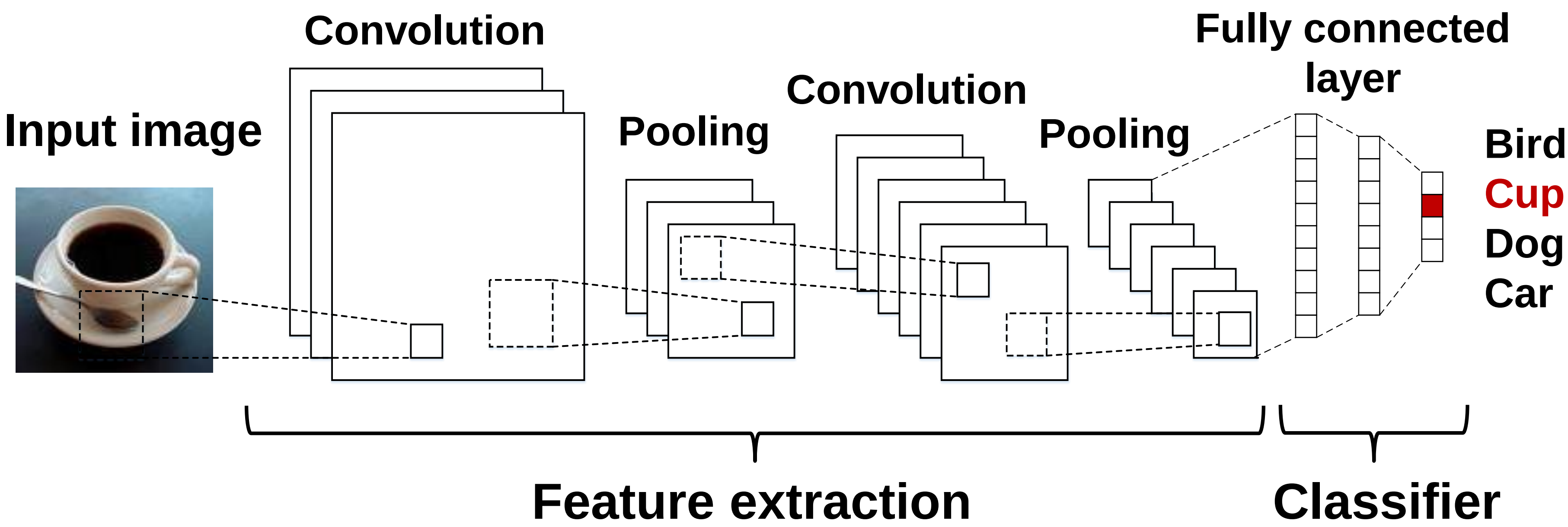
Outline

1. Introduction
   a. Deep learning
   b. Parallelization for deep learning
2. Proposed data-parallelism
3. Experimental result
4. Conclusion

Background: Deep learning

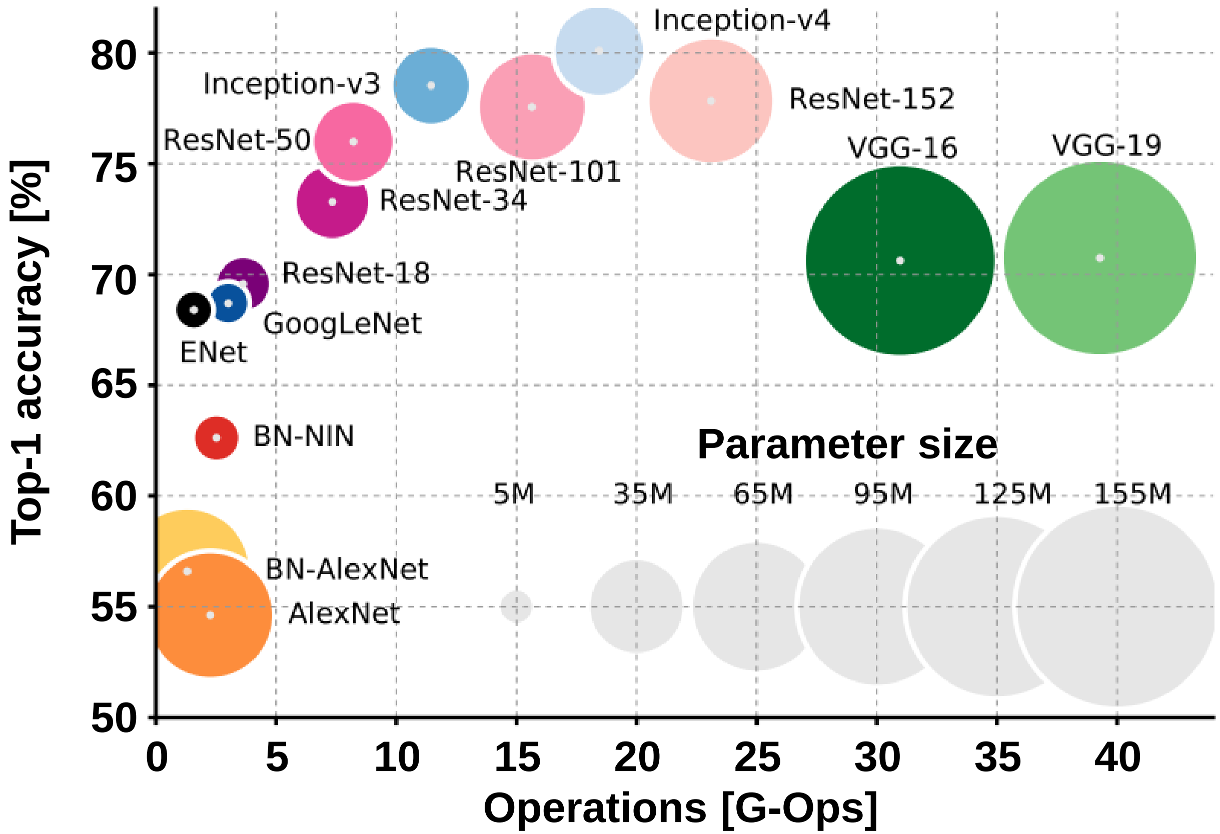
- Deep learning has succeeded in various tasks with different kinds of data by changing the form
  - Convolutional Neural Network for image data
  - Recurrent Neural Network for time series data
  - Deep Q Network for playing video game
  - etc…

- Convolutional Neural Network (CNN)



Background: CNN

- **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)**

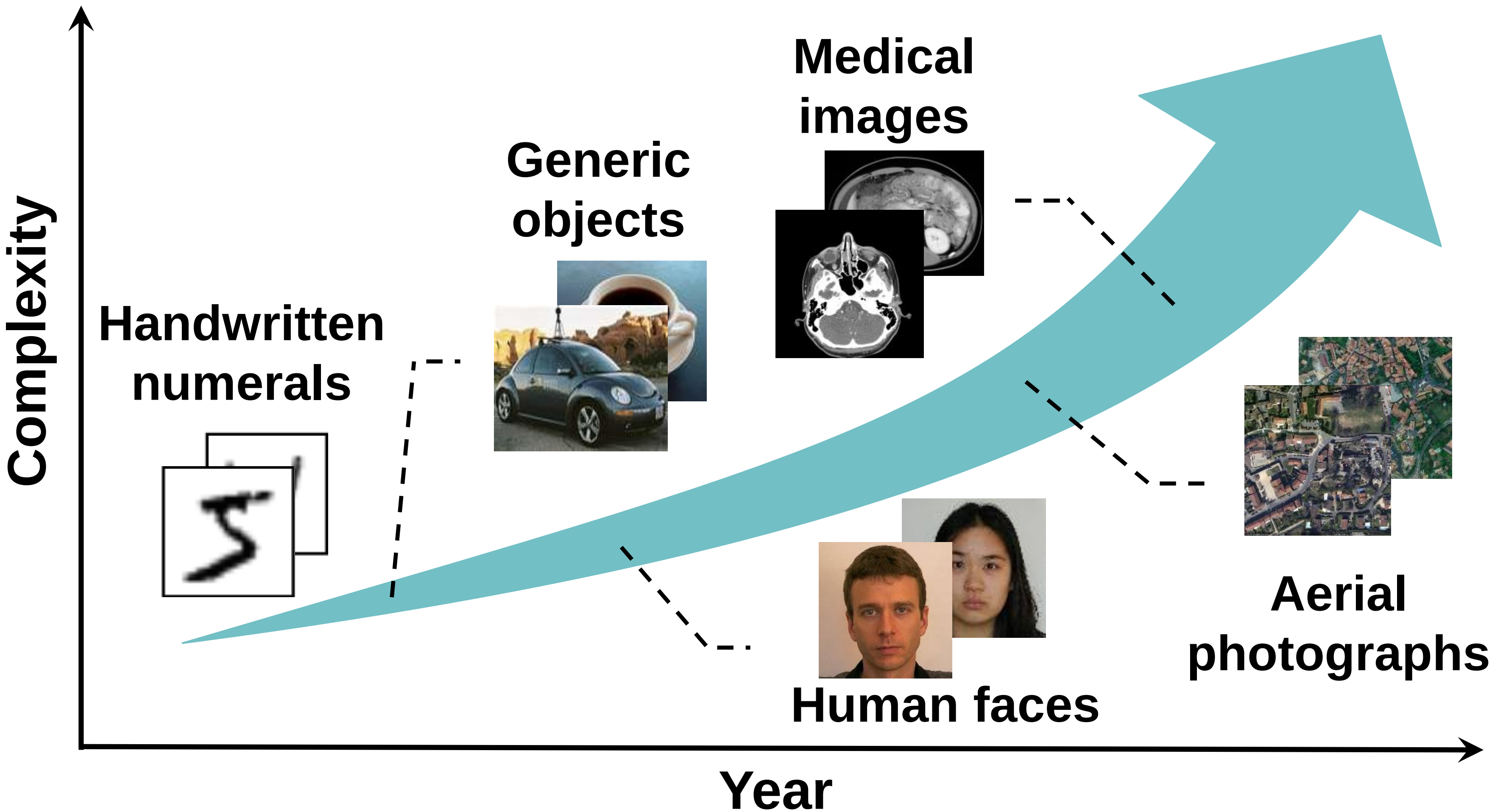


The evolution of the algorithm made it possible to deepen the model.

- **Accuracy improved dramatically, but the amount of computation and parameter exploded in accordance with the number of layers.**

Alfredo Canziani, et al., arXiv:1605.07678, Apr., 2017.

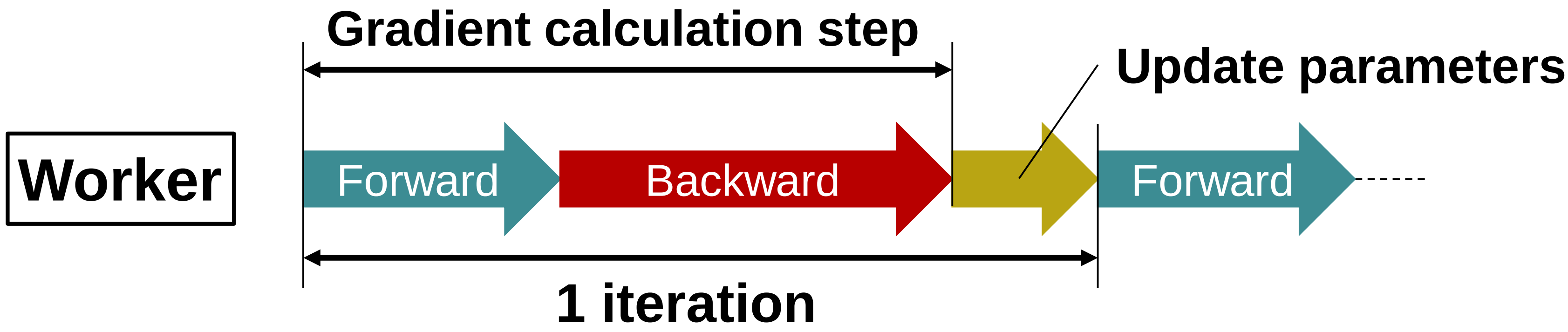
Background: CNN's target



Along with the development, target data gets more complicated. Then the CNN model gets deeper, and so computational complexity will get larger.

Background: CNN's challenge

- **Learning flow w/ single worker**



- **Due to the large amount of computation, devices specialized for matrix computation like GPUs are used.**

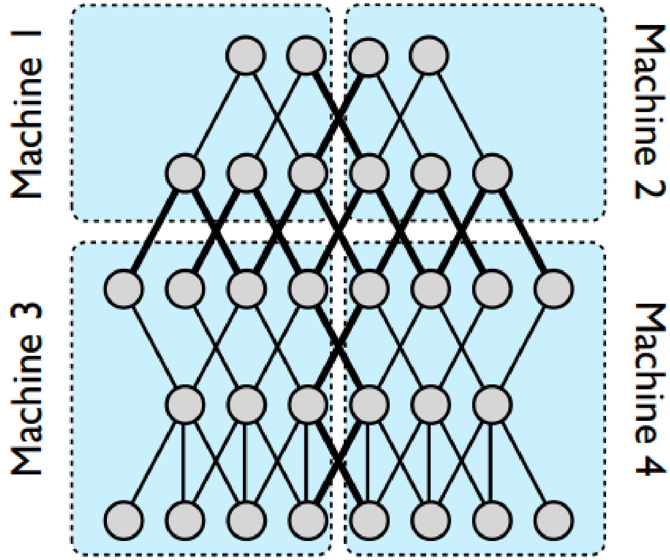
- **However, since it is necessary to calculate millions of iterations, it is difficult to learn in realistic time even with such a device.**
  - **Distributed deep learning has received a remarkable amount of attention.**

Outline

1. Introduction
   a. Deep learning
   b. Parallelization for deep learning
2. Proposed data-parallelism
3. Experimental result
4. Conclusion

Model parallelism

- Model parallelism divides **dimensions of model**.
  - Each worker calculates the different part of the model.

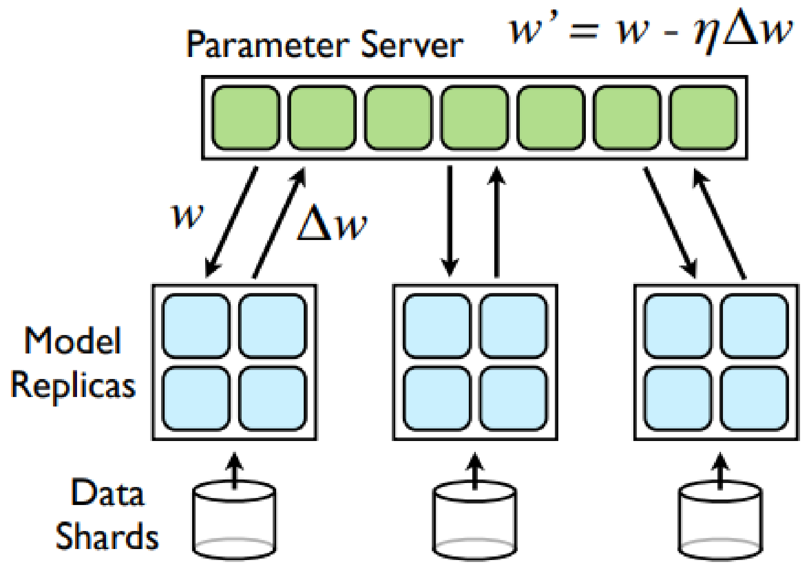


✓ No need for parameter integration
✓ Possibility of a huge model implementation even when the model cannot fit into GPU
× Model-dependent
× Low versatility

Dean, Jeffrey et al. "Large Scale Distributed Deep Networks"

Data parallelism

- Data parallelism divides dimensions of data.
  - Each worker calculates a different batch data.



✓ Simple implementation for homogeneous workers

✓ Dominant method for high versatility

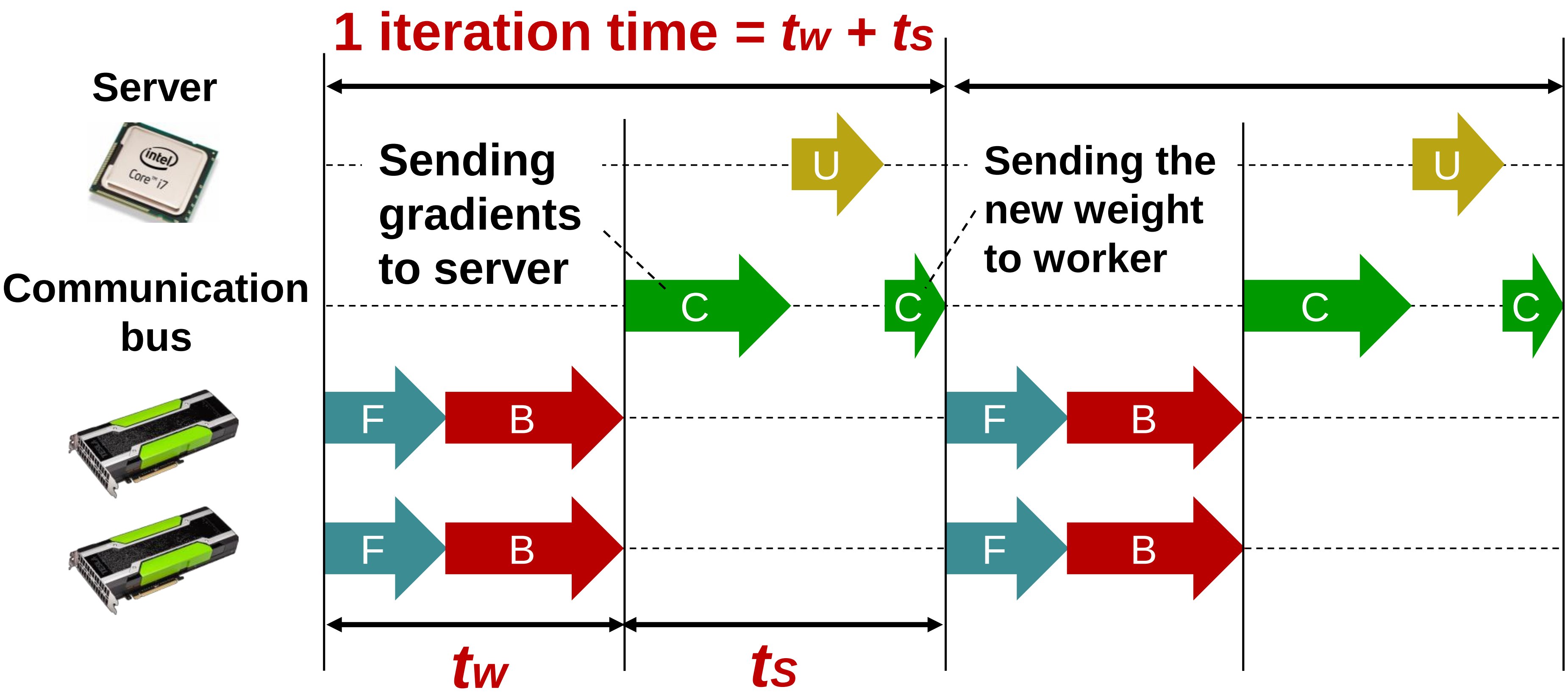
× Need for parameter integration

- For VGG-F network, it is necessary to communicate 60M parameter per thread.

Dean, Jeffrey et al. "Large Scale Distributed Deep Networks"

Data parallelism challenge

- **Data parallel learning flow w/ two GPUs**



For worker, *ts* is the latency for next step. Particulary when the size of gradient is large, communication latency becomes bottleneck.

➢ **Our target is to <u>eliminate this communication latency</u>.**

Outline

1. Introduction
    a. Deep learning
    b. Parallelization for deep learning
2. Proposed data-parallelism
3. Experimental result
4. Conclusion

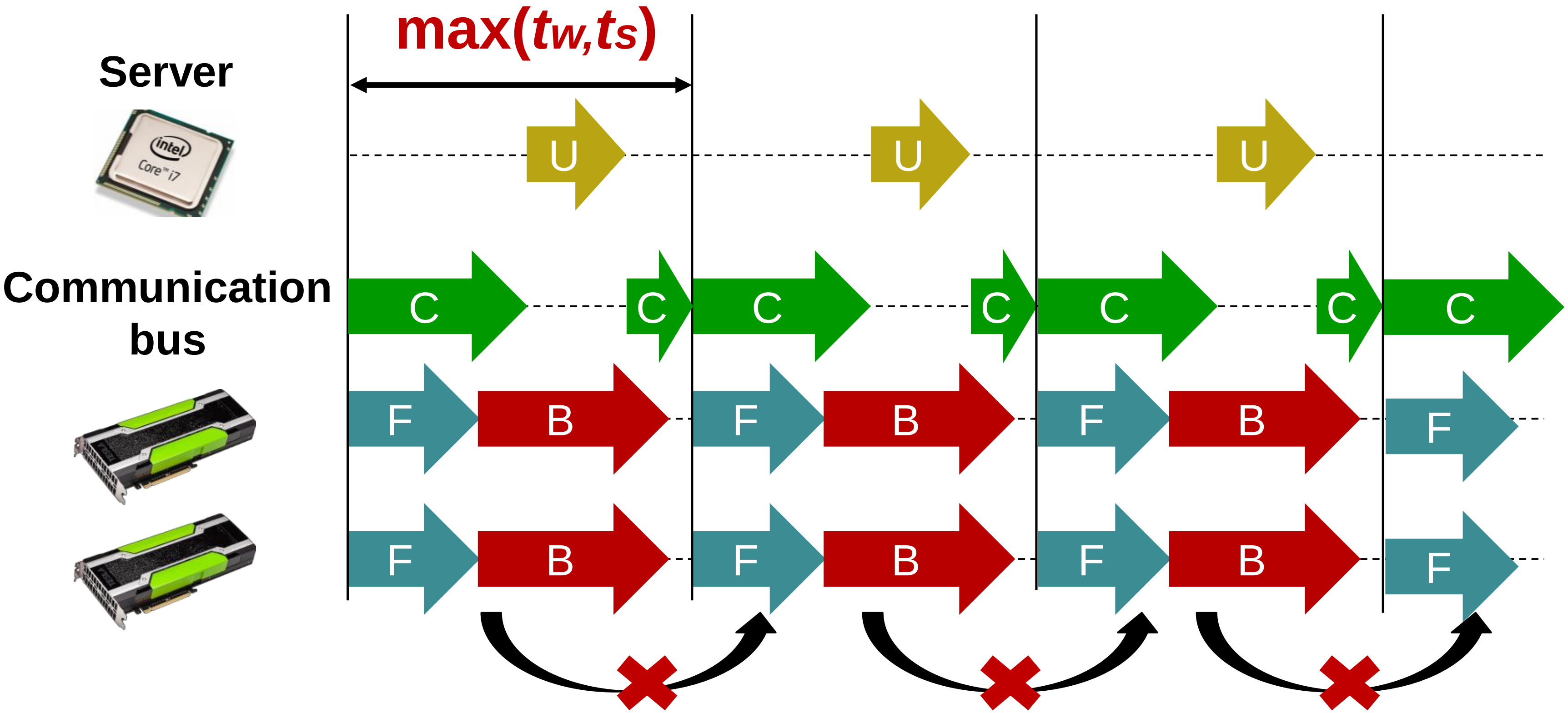
Proposed method

- In the conventional data parallel method, workers need to wait until they receive the new weight.
  - When the weight size is large, communication latency hinders speeding up by parallelism.

- Then, we proposed a method
  - to calculate the gradient using weights delayed by one step.
  - to overlap the communication with the gradient calculation.

Proposed method

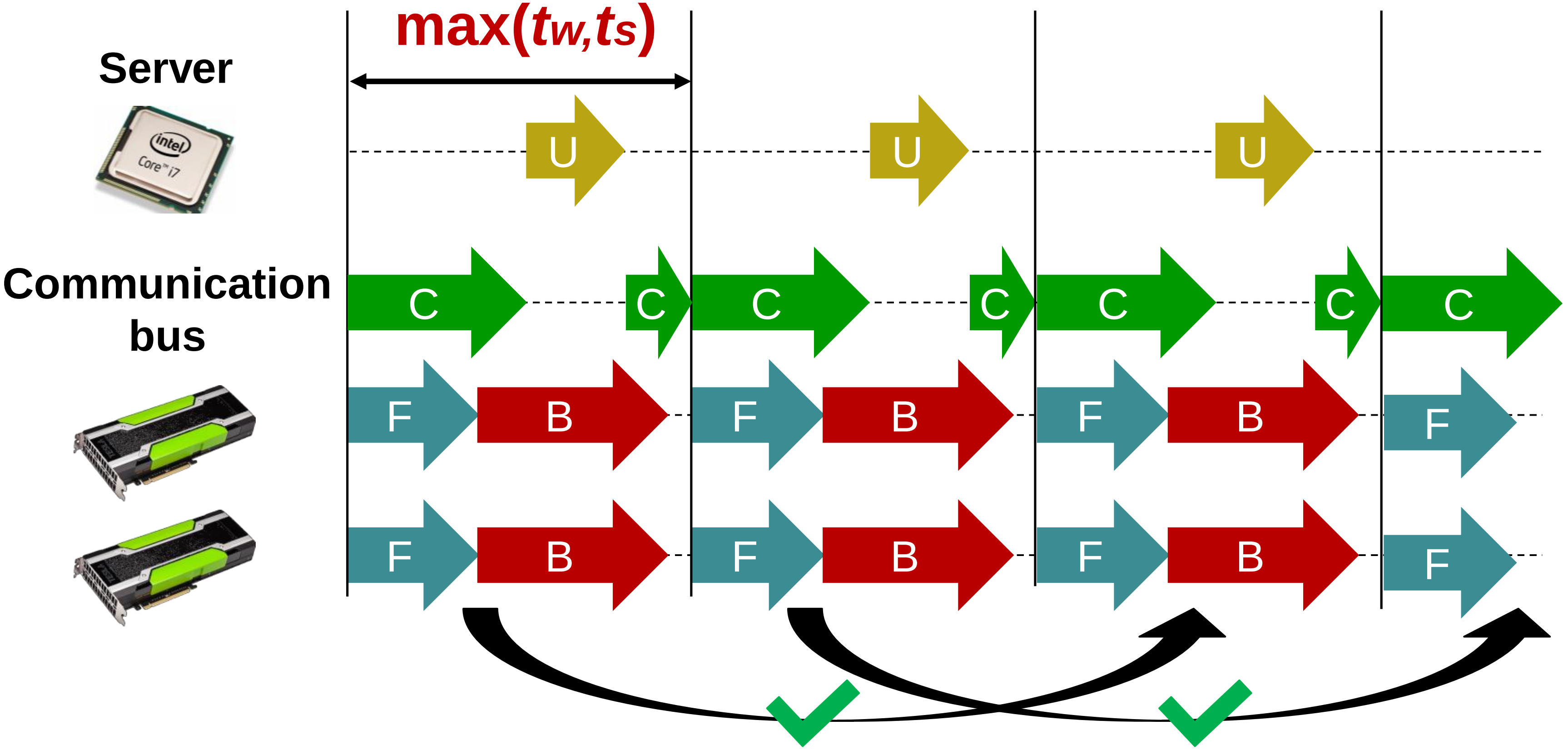
- **Proposed data parallel learning flow w/ two GPUs**



With proposed method, gradient calculation doesn't reflect the last batch update, and so we can overlap the parameter integration with the gradient calculation.

Proposed method

- **Proposed data parallel learning flow w/ two GPUs**



Gradient calculation reflects 2 steps earlier batch. The proposed method can eliminate the communication bottleneck.

- **However, it is not naïve SGD, and so we must verify the accuracy.**

Acceleration ratio

- **Conventional method takes ($t_w + t_s$) to process 1 iteration.**
- **Proposed method takes the longer one of t_w and t_s**

- **Compared to conventional data parallelism, the proposed method accelerates learning by r_{ws}.**

- **r_{ws} can be expressed as follows:**

$$r_{ws} = \frac{t_w + t_s}{max(t_w, t_s)}$$

$$t_w = \text{gradient calculation time}$$

$$t_s = \text{communication and update time}$$

Outline

1. Introduction
   a. Deep learning
   b. Parallelization for deep learning
2. Proposed data-parallelism
3. Experimental result
4. Conclusion

Experimental environment

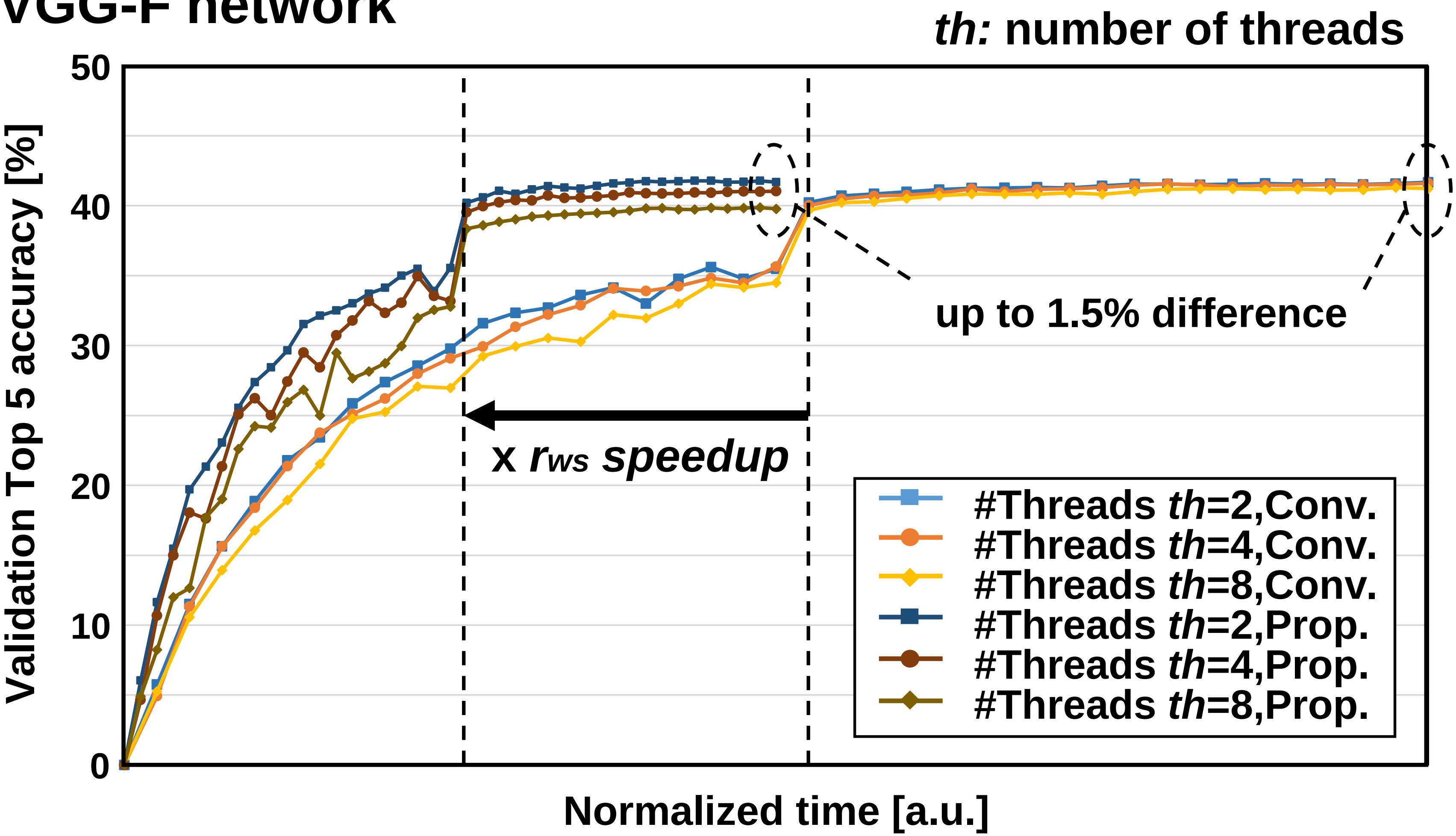
CPU	Core i7 6700K
GPU	Nvidia Geforce GTX 1080
Framework	Matconvnet
Communication interface (CPU-GPU)	PCI express Gen3 x16
Network	VGG-F network ResNet-50
Optimizer	MomentumSGD
Learning rate (based on linear scaling rule)	VGG-F: 0.001 * *th* ResNet: 0.025 * *th* (divided by 10 every 30 epoch)
Dataset	ImageNet-1k (50k images for train)

th: number of threads

A. Krizhevsky, "One weird trick for parallelizing convolutional neural network"

Training convergences evaluation

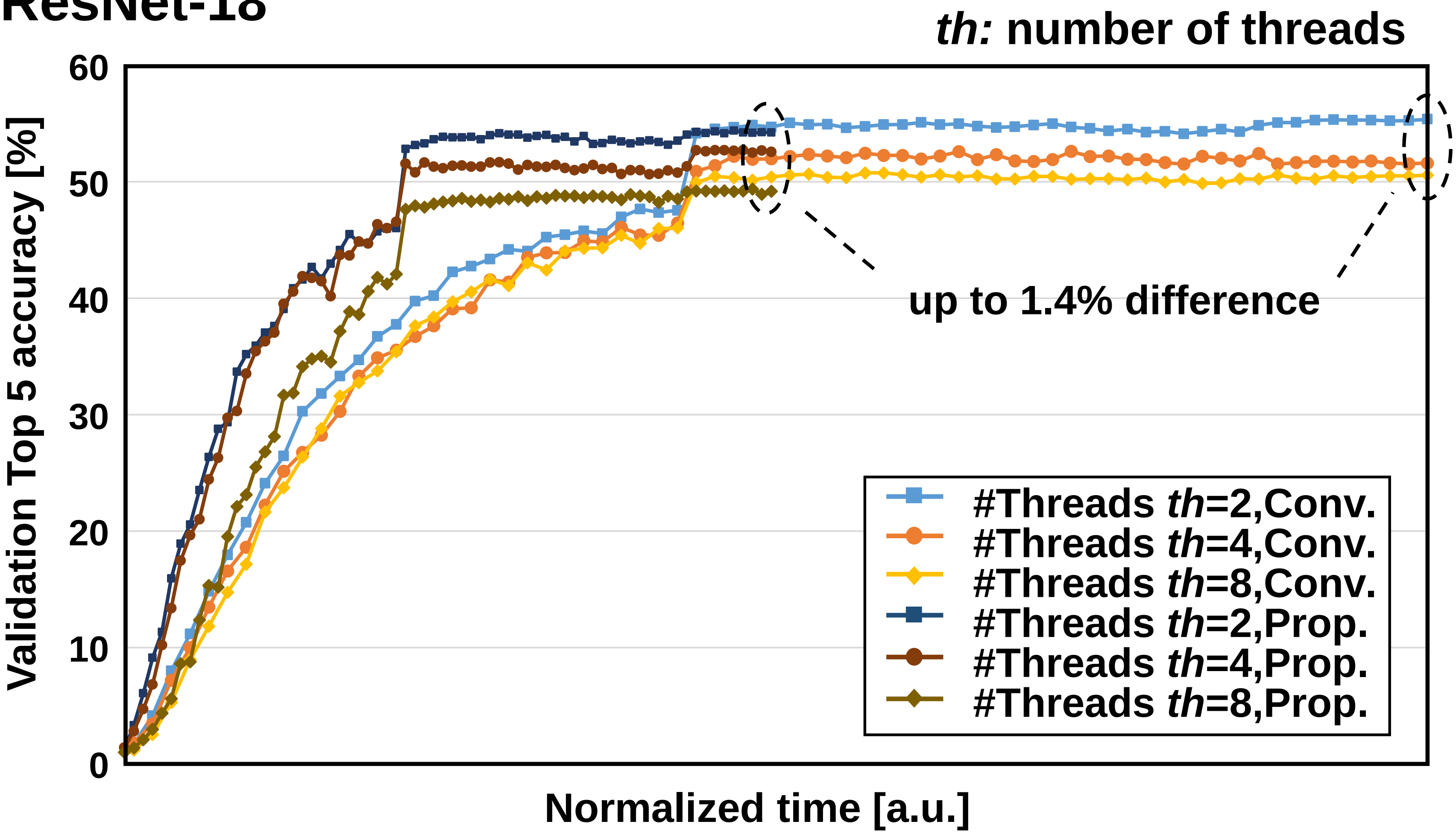
- VGG-F network



The final accuracy degradation is 1.5% at the maximum.There is a possibility that this accuracy degradation can be suppressed by changing optimizer or tuning parameters well.

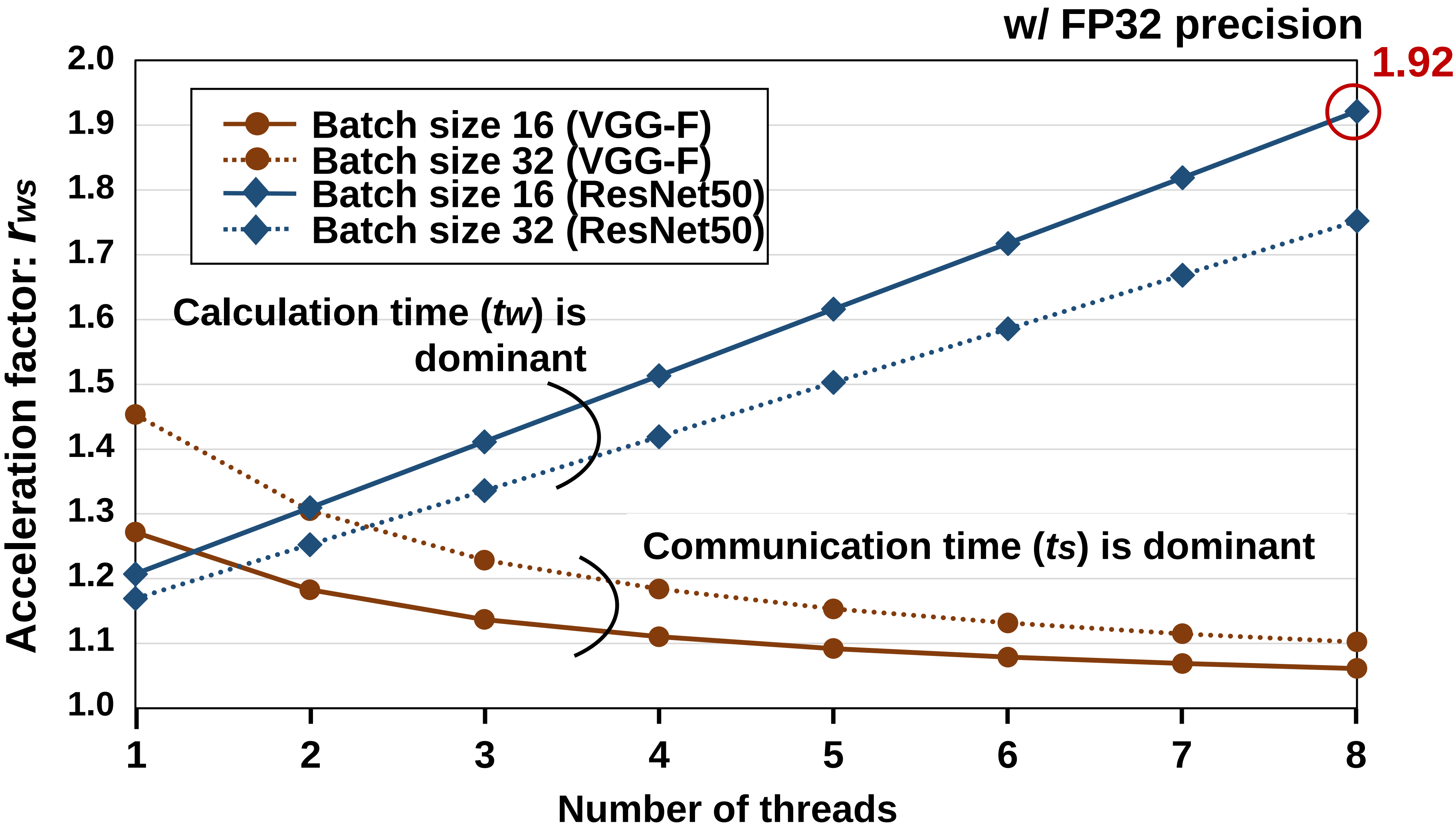
Training convergences evaluation

- **ResNet-18**



Resnet-18's result exhibits a similar tendency with VGG-f's result.
The final accuracy degradation is 1.4% at the maximum.

Acceleration ratio evaluation



We calculated the ratio *rws* by measuring communication and calculation time. In ResNet50, *rws* reaches 1.92 when the number of threads is eight and the batch size is 16.

Conclusion

- **In conventional data parallelism, when the gradient data size is large, communication latency can be bottleneck.**
- **Using one step delayed weights for gradient calculation is proposed.**
  - **Nevertheless proposed method is not naïve SGD, there is up to 1.5% difference between the accuracy of proposed method and that of conventional one.**
  - **We confirmed that the proposed method actually accelerates learning with general equipment.**
  - **Particularly in ResNet50, *rws* reaches 1.92 when the number of threads is eight and the batch size is 16.**