CNN & cuDNN

Bin ZHOU

USTC Jan.2015

Acknowledgement

- Reference:
- 1) Introducing NVIDIA® cuDNN, Sharan Chetlur, Software Engineer,
- CUDA Libraries and Algorithms Group

- 深度卷积神经网络CNNs的多GPU并行框架 及其在图像识别的应用 --
http://data.qq.com/article?id=1516

CNN

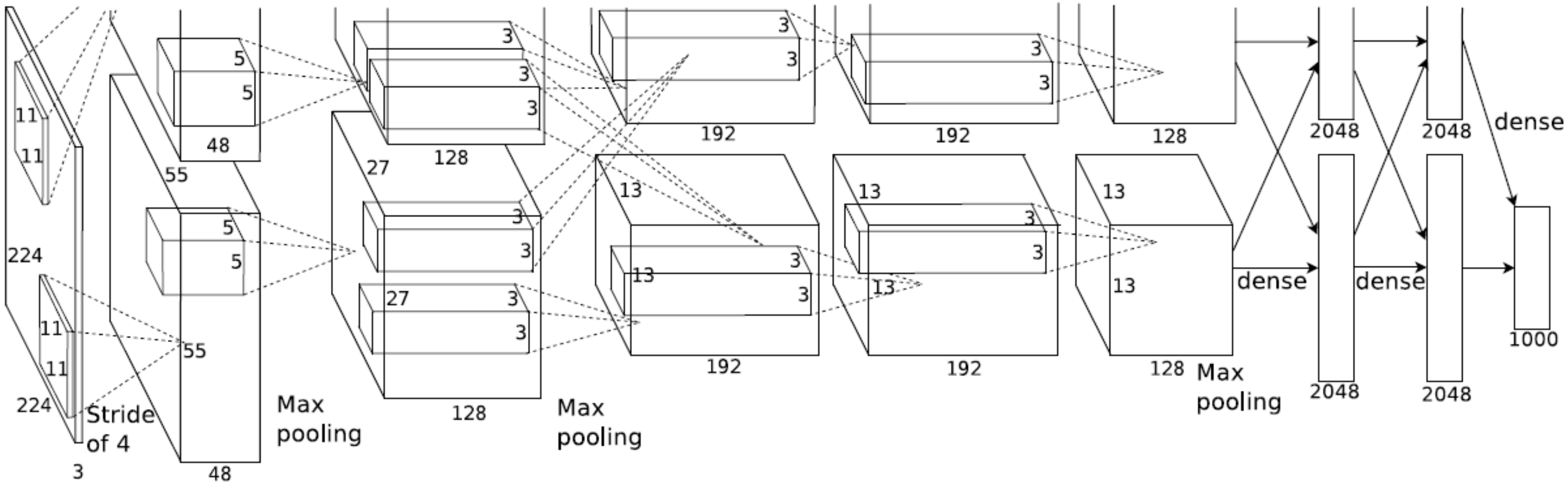
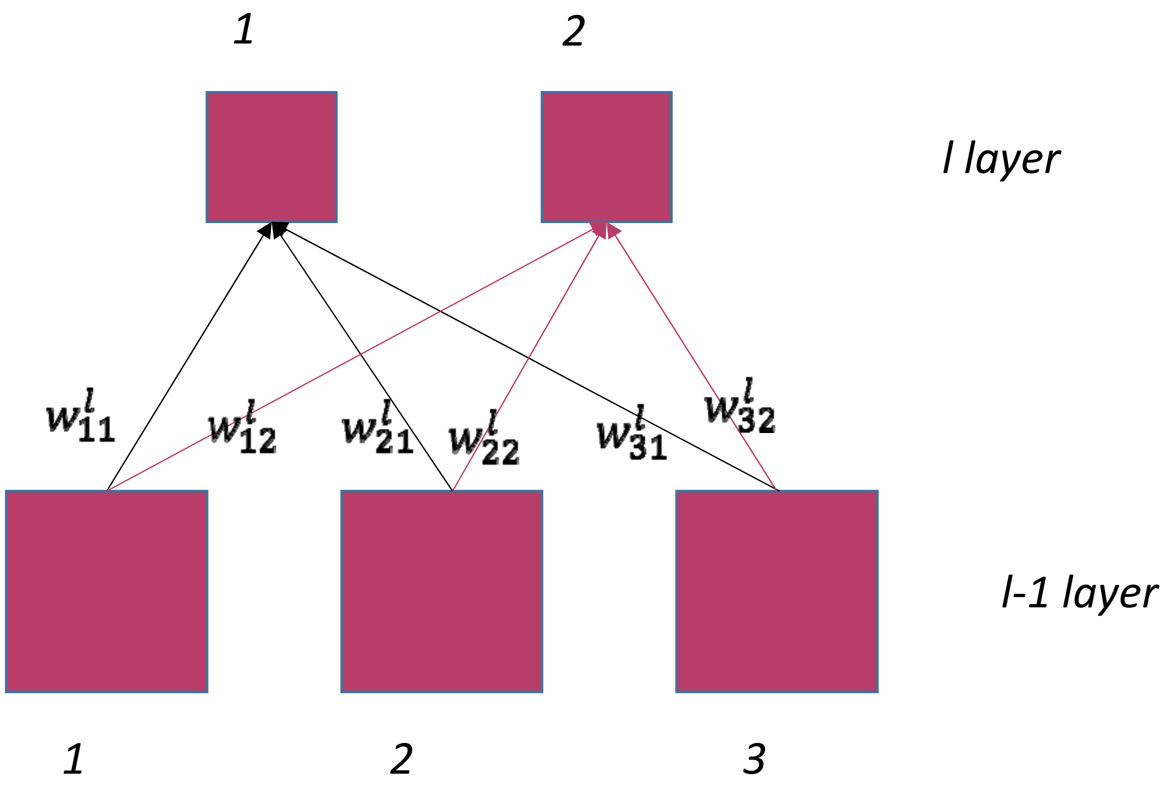


Figure1. ImageNet CNN Model

Recall BP Network



BP Brief Review

- Cost(Loss) Function   To Evaluate the output of the network

- Common Cost Function
  - MSE(Mean Squared Error)
  - Cross Function

2D Convolution

$$\begin{pmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{pmatrix} * \begin{pmatrix} 1 & 3 & 1 \\ 0 & 5 & 0 \\ 2 & 1 & 2 \end{pmatrix}$$

full

same

valid

17	75	90	35	40	53	15
23	159	165	45	105	137	16
38	198	120	165	205	197	52
56	95	160	200	245	184	35
19	117	190	255	235	106	53
20	89	160	210	75	90	6
22	47	90	65	70	13	18

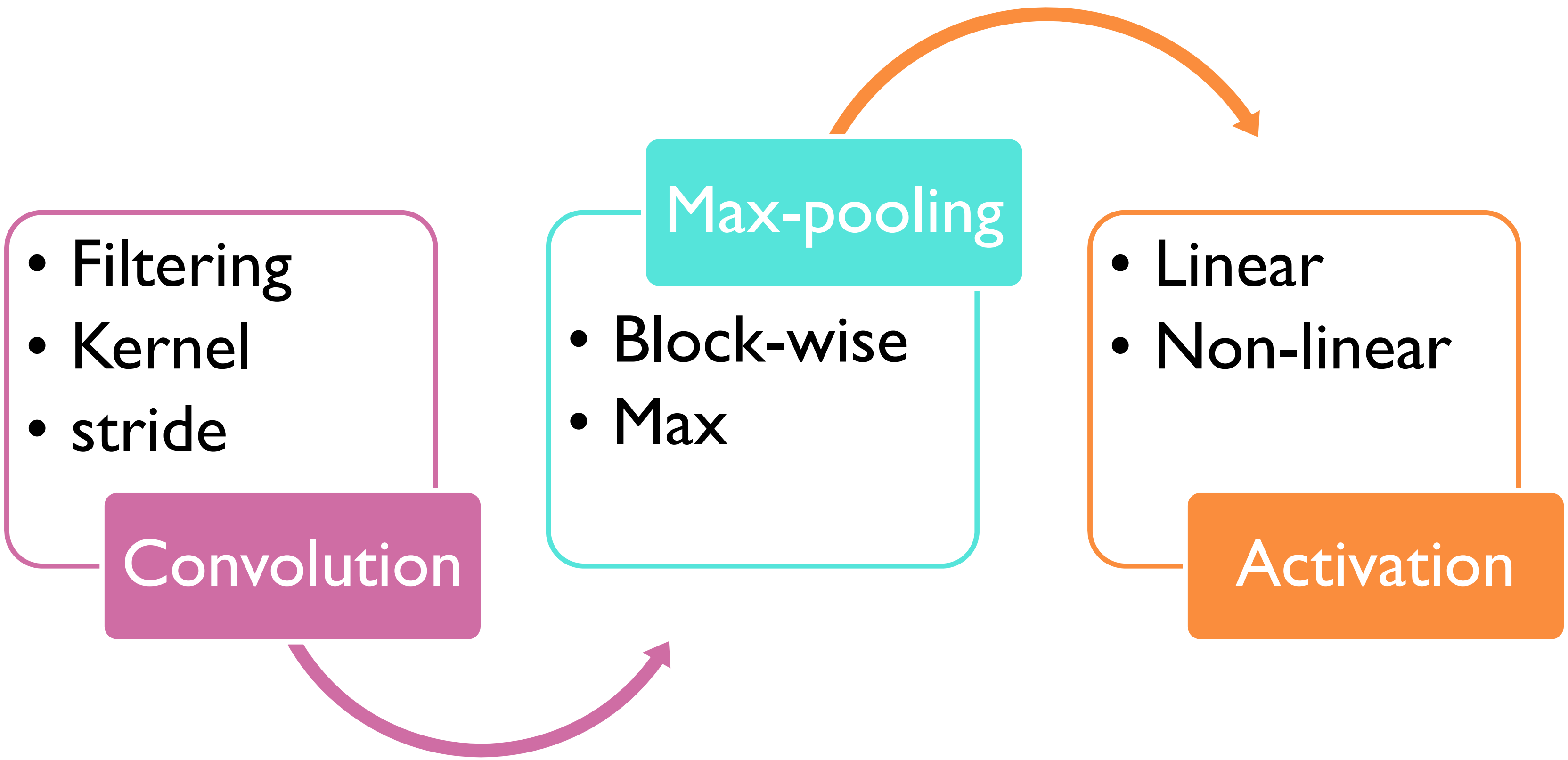
CNN Brief

- Interpret AI task as the evaluation of complex function
  - Facial Recognition: Map a bunch of pixels to a name
  - Handwriting Recognition: Image to a character
- Neural Network: Network of interconnected simple "neurons"
- Neuron typically made up of 2 stages:
  - Linear Transformation of data
  - Point-wise application of non-linear function
- In a CNN, Linear Transformation is a convolution

cuDNN

- implementations of routines
  - Convolution
  - Pooling
  - softmax
  - neuron activations, including:
    Sigmoid
    Rectified linear (ReLU)
    Hyperbolic tangent (TANH)

CNNs: Stacked Repeating Triplets



Applications ?

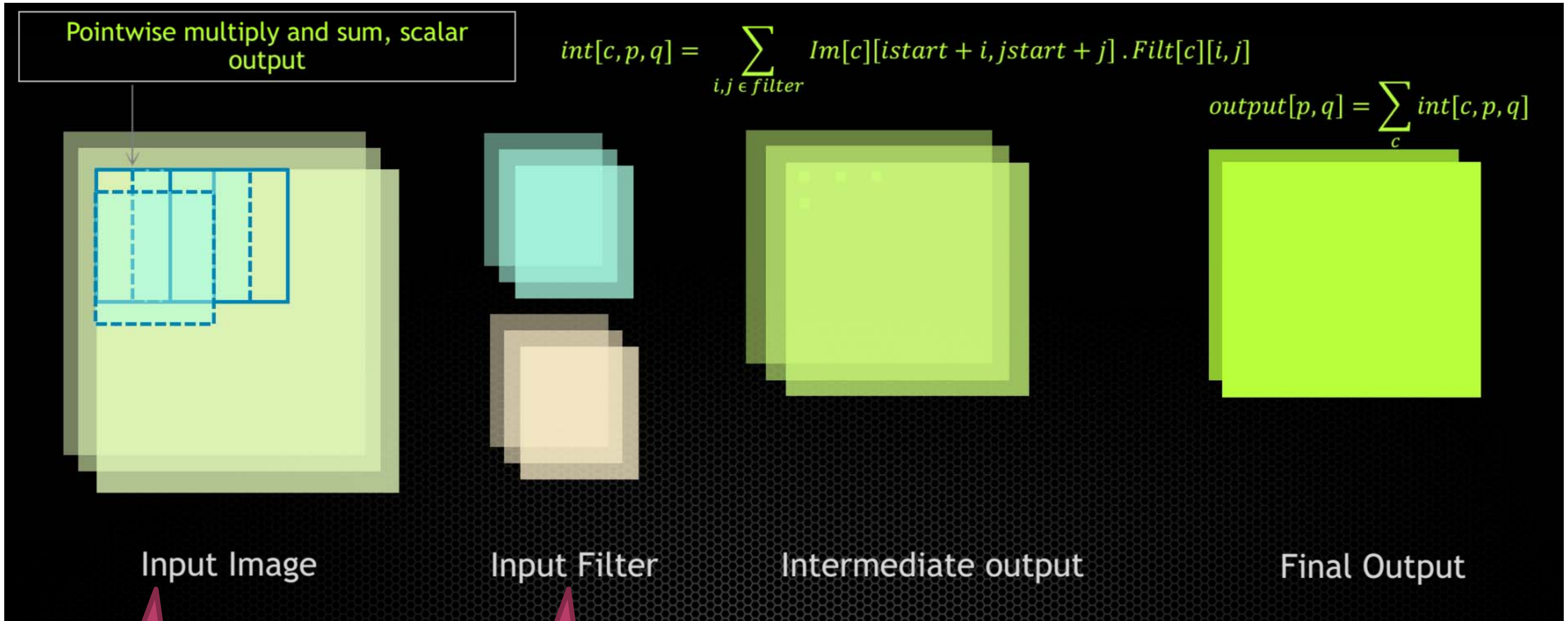
- Anyone Enlighten me?

- You can bring more brilliant applications

Multi-convolve overview

- Linear Transformation part of the CNN neuron
  - Main computational workload
  - 80-90% of execution time
- Generalization of the 2D convolution (a 4D tensor convolution)
- Very compute intensive, therefore good for GPUs
- However, not easy to implement efficiently

Multi-convolve, pictorially



Good Parallelism

Good Parallelism

Why do it once if you can do it n times ?
Batch the whole thing, to get parallelism.

cuDNN-GPU accelerated CNN lib

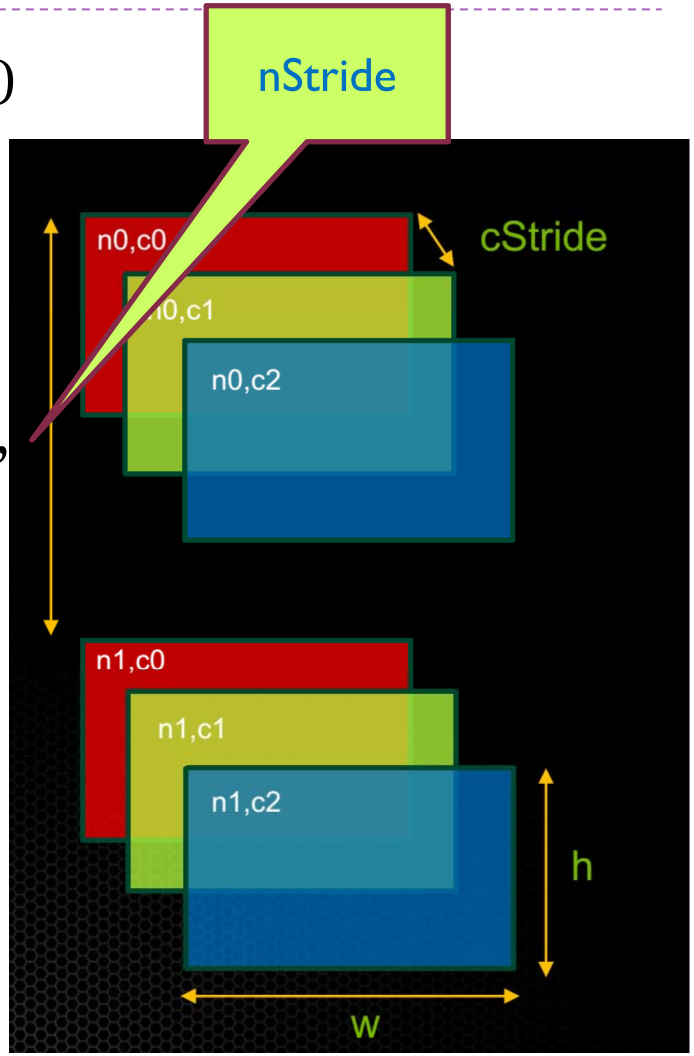
- Low-level Library of GPU-accelerated routines; similar
  in intent to BLAS
- Out-of-the-box speedup of Neural Networks
- Developed and maintained by NVIDIA
- Optimized for current and future NVIDIA GPU generations
- First release focused on Convolutional Neural Networks

cuDNN Features

- Flexible API : arbitrary dimension ordering, striding, and sub-regions for 4d tensors
- Less memory, more performance : Efficient forward and backward convolution routines with zero memory overhead
- Easy Integration : black box implementation of convolution and other routines - ReLu, Sigmoid, Tanh, Pooling, Softmax

Tensor-4d: Important

- Image Batches described as 4D Tensor
  [n, c, h, w] with stride support
  [nStride, cStride, hStride, wStride]
- Allows flexible data layout
- Easy access to subsets of features (Caffe's "groups")
- Implicit cropping of sub-images
- Plan to handle negative



Example - OverFeat Layer 1

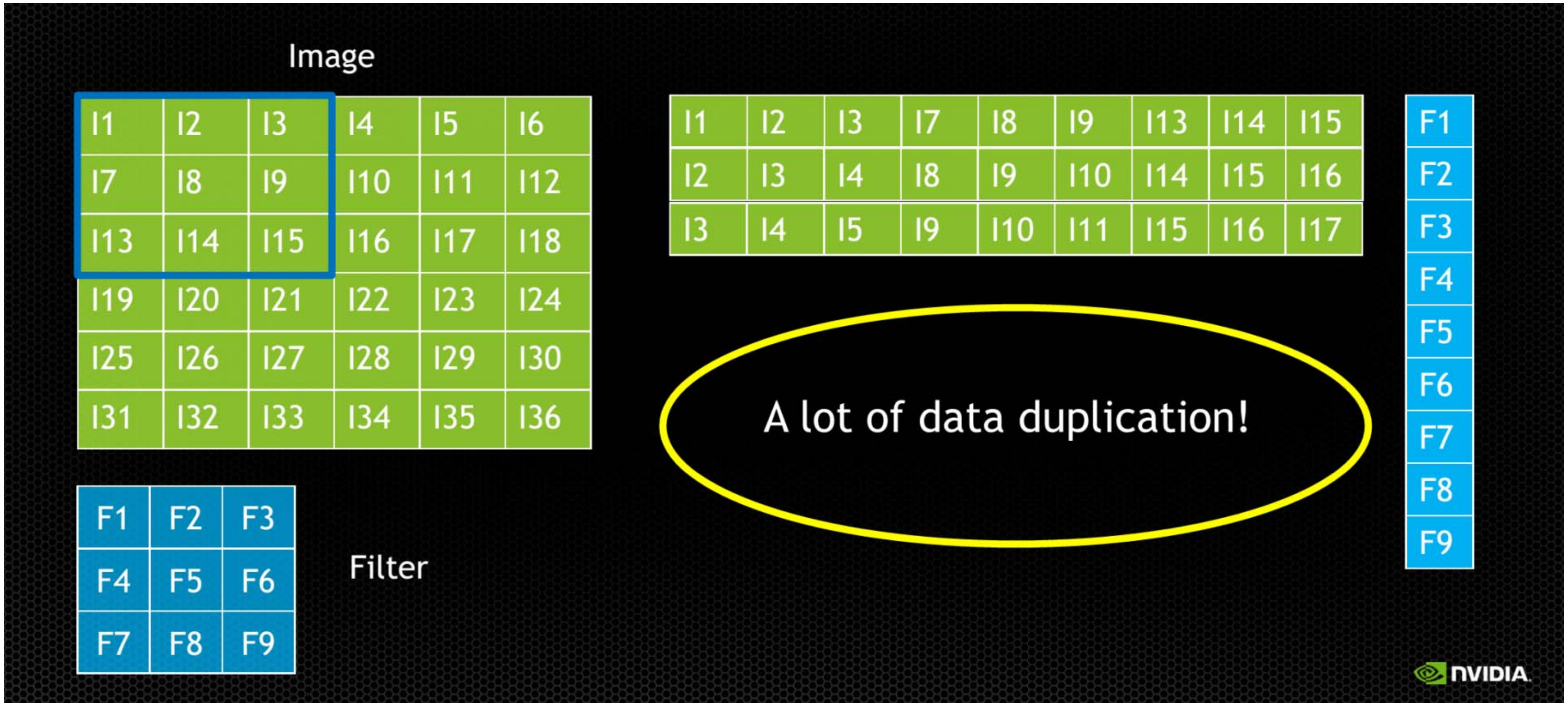
```
/* Allocate memory for Filter and ImageBatch, fill with data */
cudaMalloc( &ImageInBatch , ... );
cudaMalloc( &Filter , ... );
...

/* Set descriptors */
cudnnSetTensor4dDescriptor( InputDesc, CUDNN_TENSOR_NCHW, 128, 96, 221, 221);
cudnnSetFilterDescriptor( FilterDesc, 256, 96, 7, 7 );
cudnnSetConvolutionDescriptor( convDesc, InputDesc, FilterDesc,
    pad_x, pad_y, 2, 2, 1, 1, CUDNN_CONVOLUTION);

/* query output layout */
cudnnGetOutputTensor4dDim(convDesc, CUDNN_CONVOLUTION_FWD, &n_out, &c_out, &h_out, &w_out);

/* Set and allocate output tensor descriptor */
cudnnSetTensor4dDescriptor( &OutputDesc, CUDNN_TENSOR_NCHW, n_out, c_out, h_out, w_out);
cudaMalloc(&ImageBatchOut, n_out * c_out * h_out * w_out * sizeof(float));

/* launch convolution on GPU */
cudnnConvolutionForward( handle, InputDesc, ImageInBatch, FilterDesc, Filter, convDesc,
    OutputDesc, ImageBatchOut, CUDNN_RESULT_NO_ACCUMULATE);
```

NVIDIA

Real Code that runs

- Under Linux
- Demostration

Implementation 1: 2D conv as a GEMV



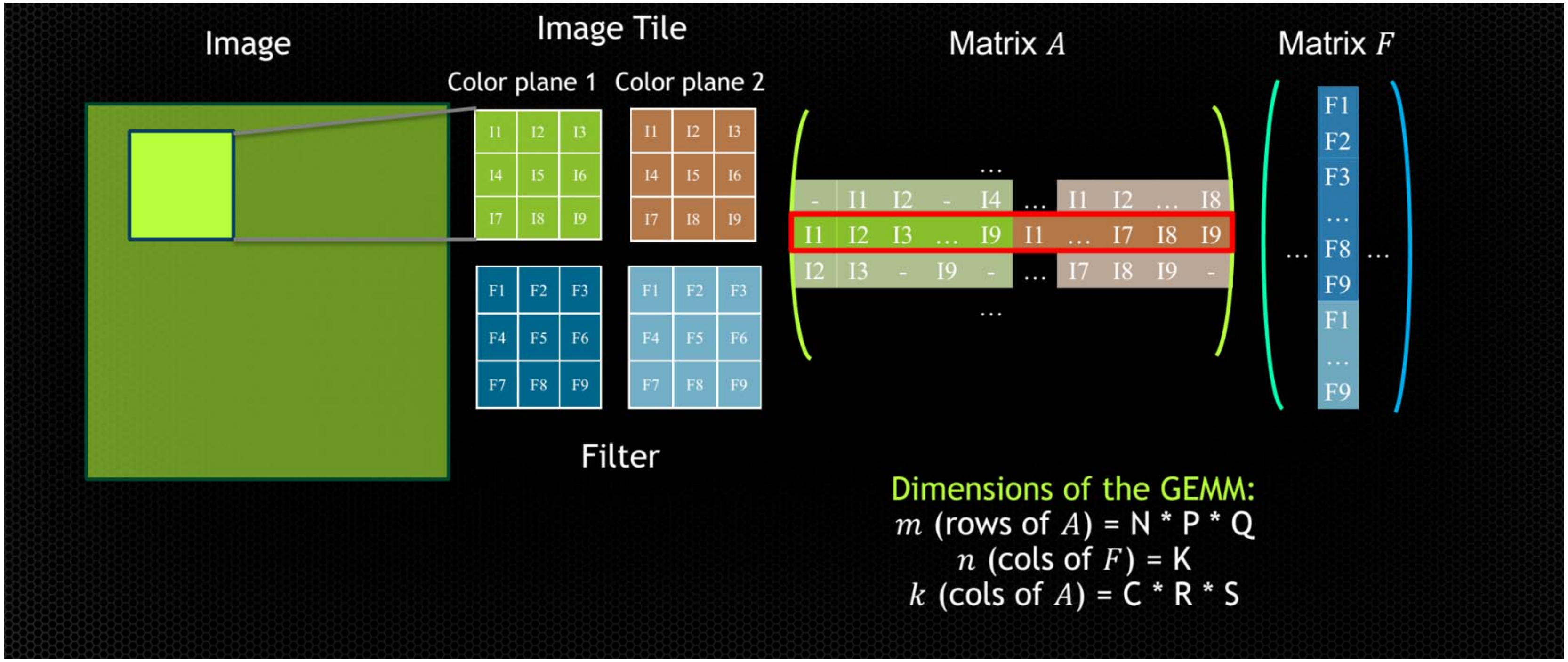
Multi-convolve

- More of the same, just a little different
  Longer dot products
  More filter kernels
  Batch of images, not just one
  Mathematically:

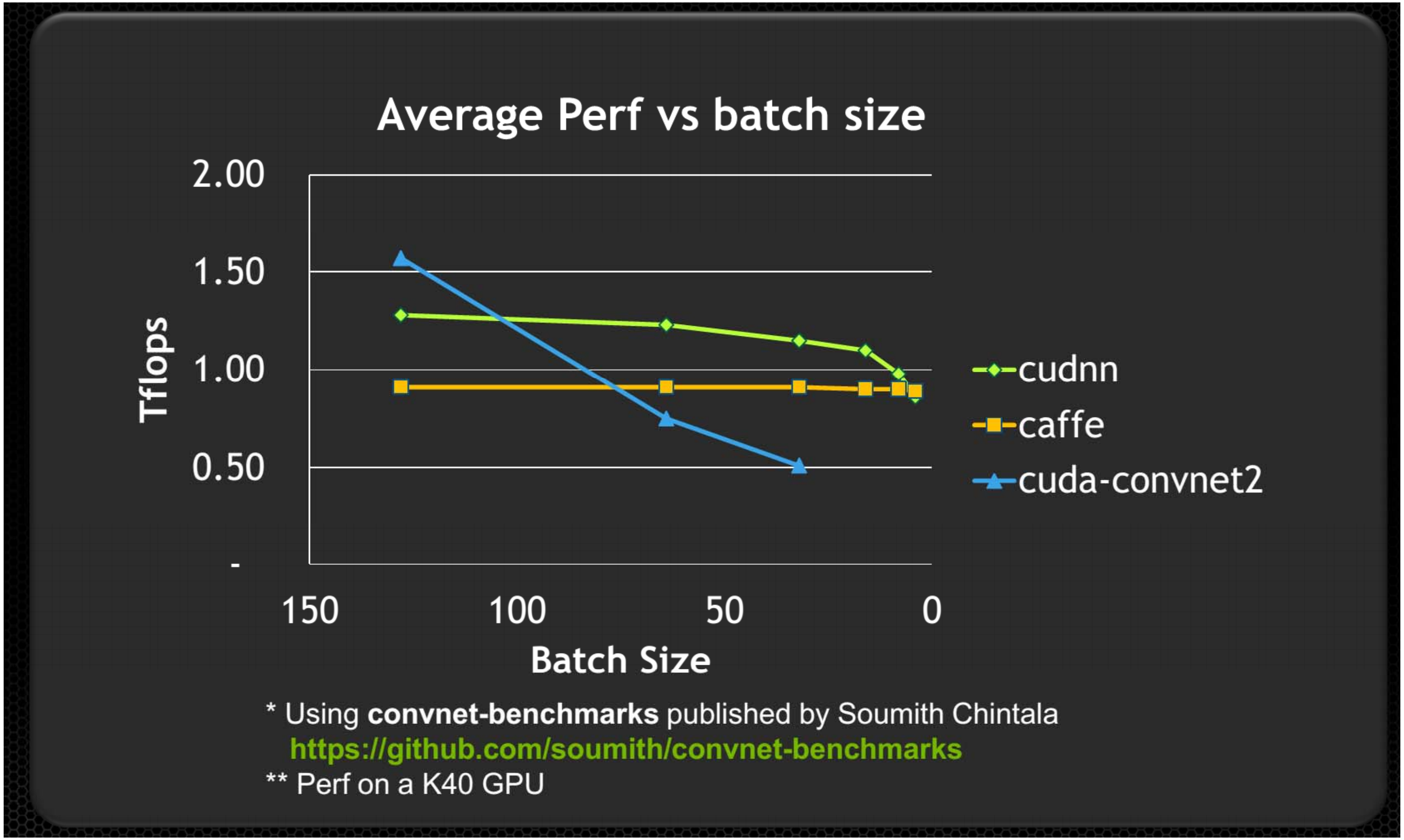
$$out[k,p,q] = \sum_{c \epsilon\, input\ color\ planes} \left(\sum_{i,j\ \epsilon\ filter} Im[c][istart+i, jstart+j]\,.\,Filt[k][c][i,j] \right)$$

$$\forall k\ \in output\ color\ planes, (p,q) \in output\ image$$

Implementation 2: Multi-convolve as GEMM



Performance

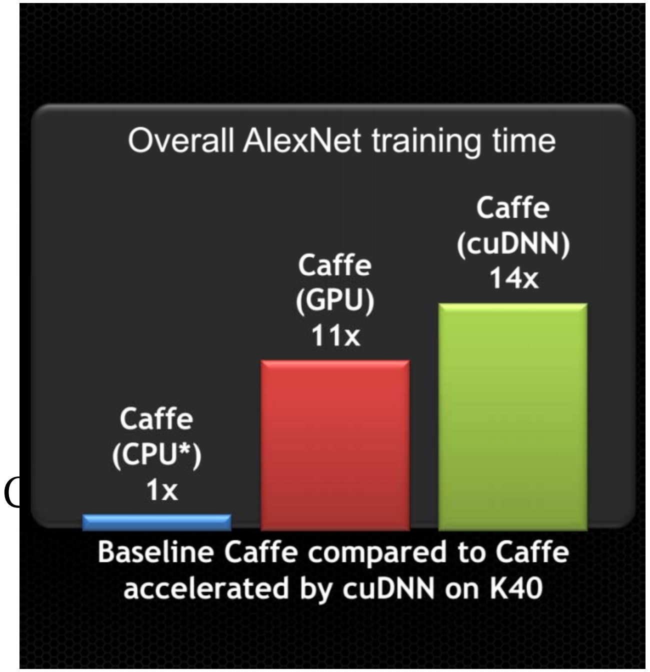


cuDNN Integration

- cuDNN is already integrated in major open-source frameworks
  - Caffe
  - Torch

Using Caffe with cuDNN

- Accelerate Caffe layer types by 1.2 - 3x
- On average, 36% faster overall for training on Alexnet
- Integrated into Caffe dev branch today!(official release with Caffe 1.0)
- Seamless integration with a global switch



*CPU is 24 core E5-2697v2 @ 2.4GHz Intel MKL 11.1.3

Caffe with cuDNN: No Programming Required

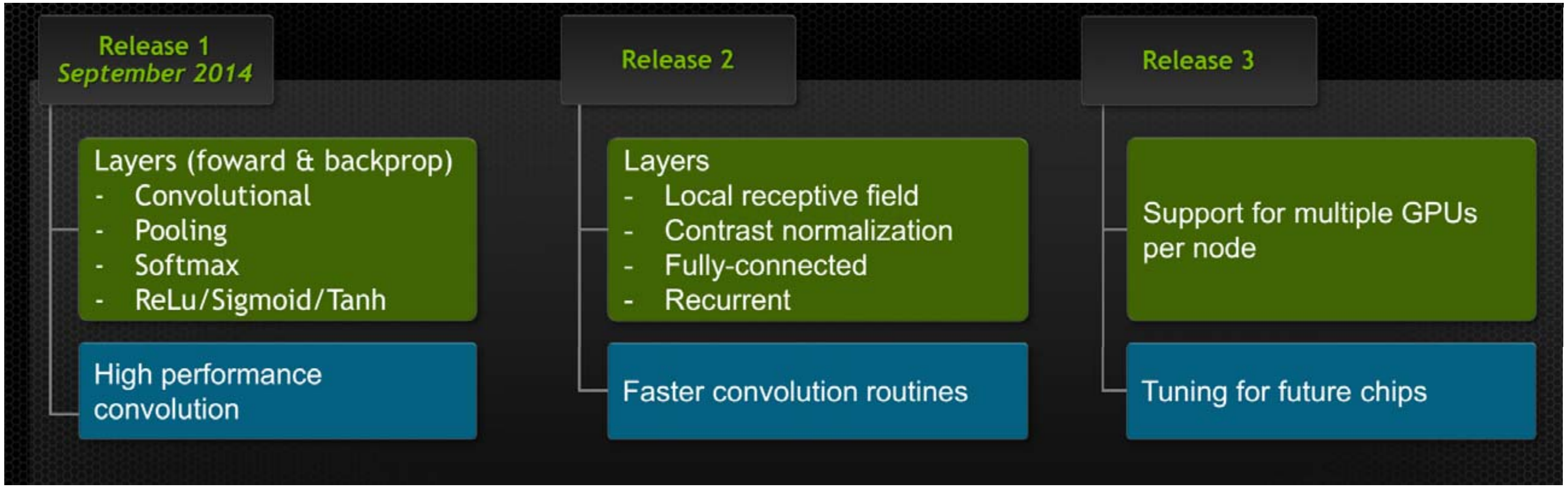
- layers {
  name: "MyData"
  type: DATA
  top: "data"
  top: "label"
  }
- layers {
  name: "Conv1"
  type: CONVOLUTION
  bottom: "MyData"
  top: "Conv1"
  convolution_param {
  num_output: 96
  kernel_size: 11
  stride: 4
  }

layers {
name: "Conv2"
type: CONVOLUTION
bottom: "Conv1"
top: "Conv2"
convolution_param {
num_output: 256
kernel_size: 5
}

Caffe with cuDNN : Life is easy

- install cuDNN
- uncomment the USE_CUDNN := 1 flag in Makefile.config when installing Caffe.
- Acceleration is automatic

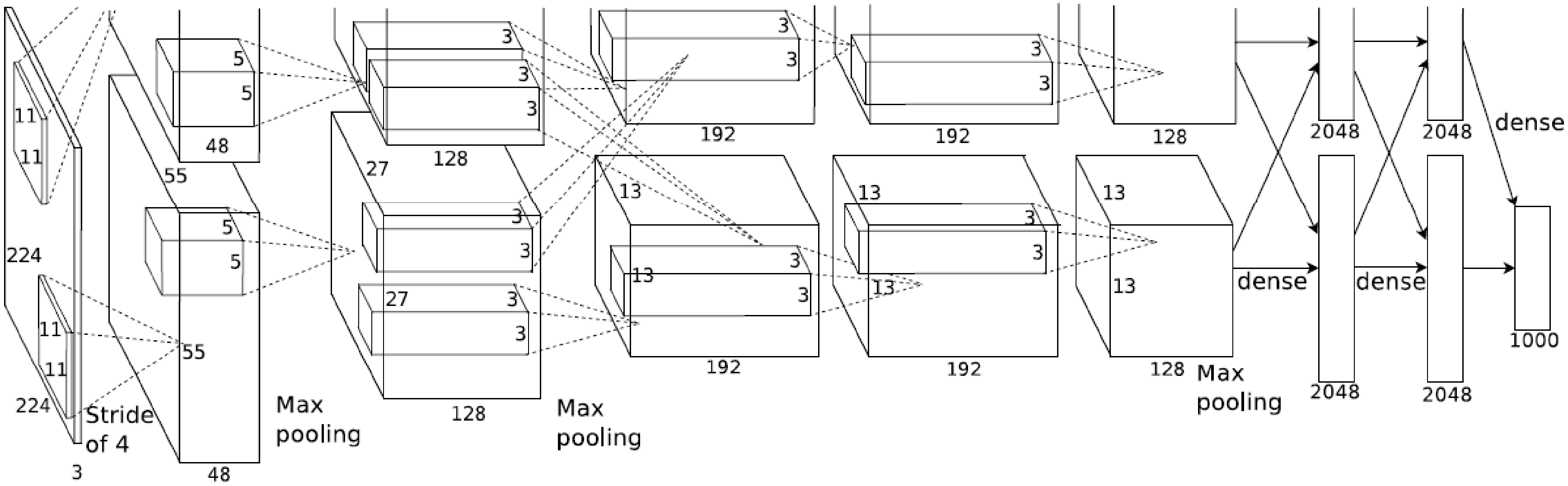
NVIDIA® cuDNN Roadmap



cuDNN availability

- Free for registered developers!
- Release 1 / Release 2 - RC
  - available on Linux/Windows 64bit
  - GPU support for Kepler and newer
- Already Done:
  Tegra K1 (Jetson board)
  Mac OSX support

Multi-GPU with CNN



Problem:

1) Single GPU has limited memory, which limits the size of the network
2) Single GPU is still too slow for some very large scale network

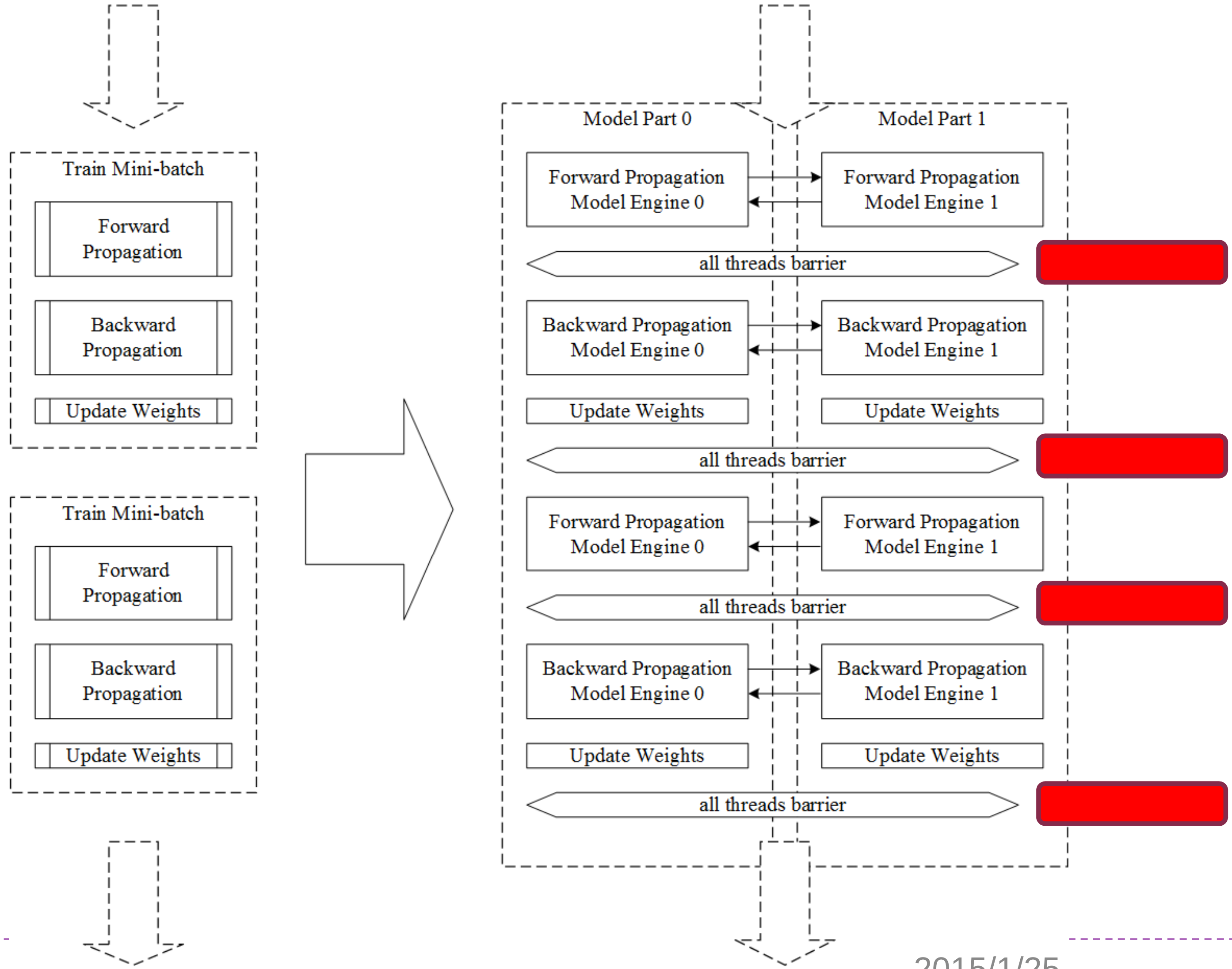
Multi-GPU Challenge

- First, How to parallelize the whole process, to avoid or reduce data dependency between different nodes

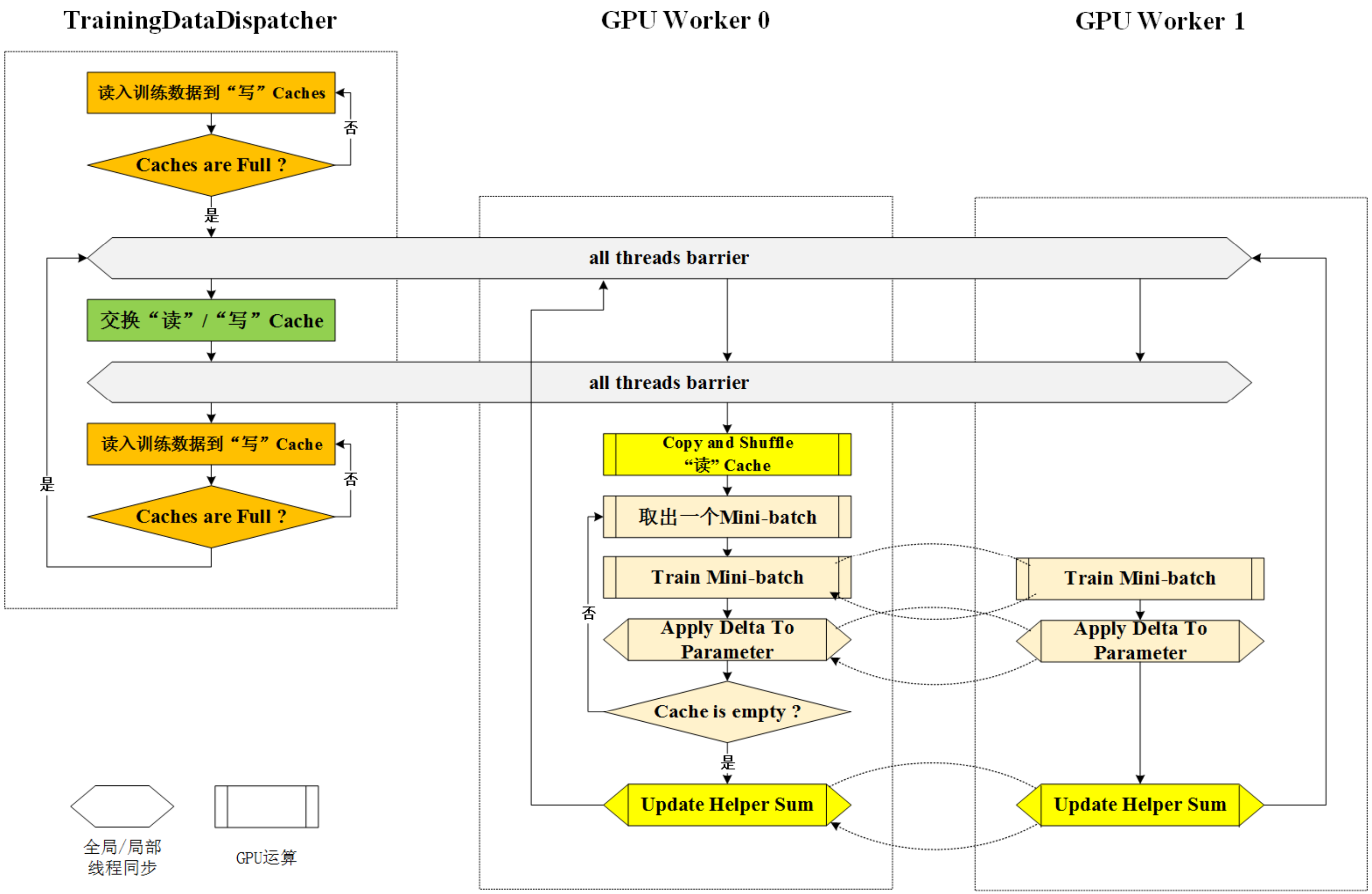
- Data IO and distribution to different Nodes
  - Pipeline and IO/Execution overlap to hide latency

- Synchronize between all the nodes??

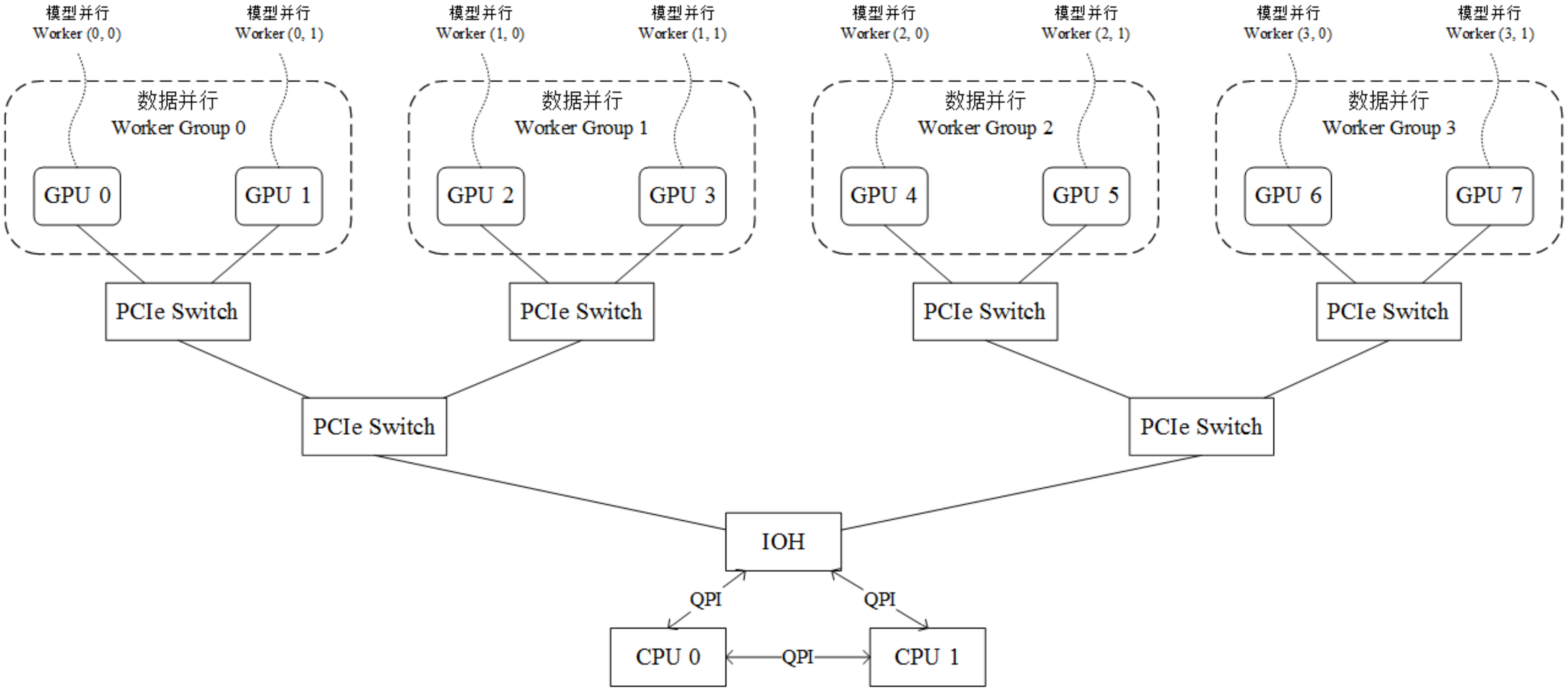
Multi-GPU Strategy



Data distribution IO/Exe Overlap

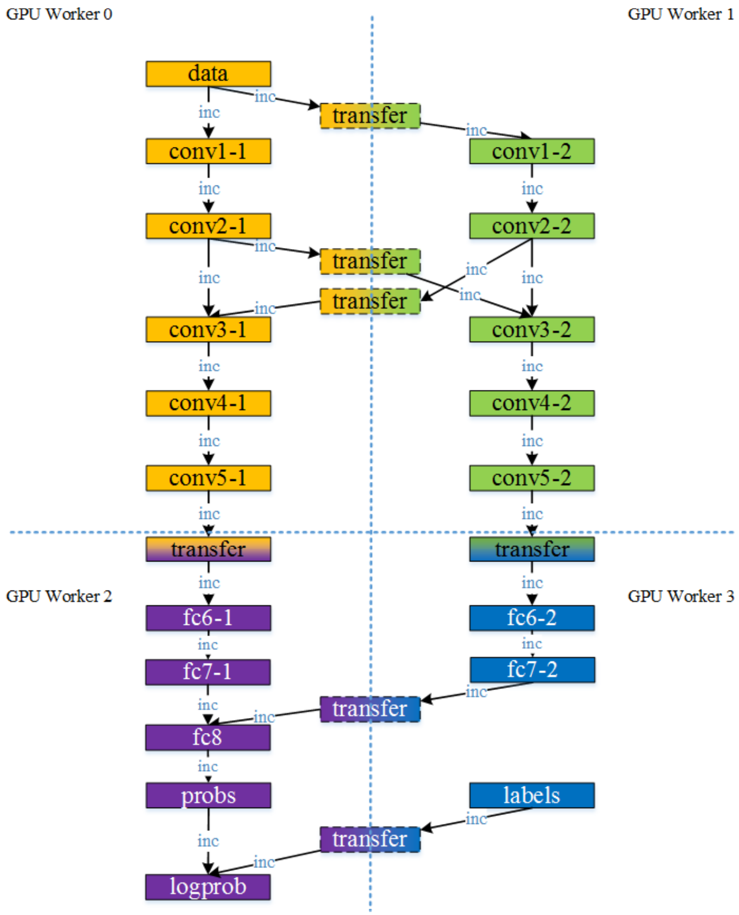


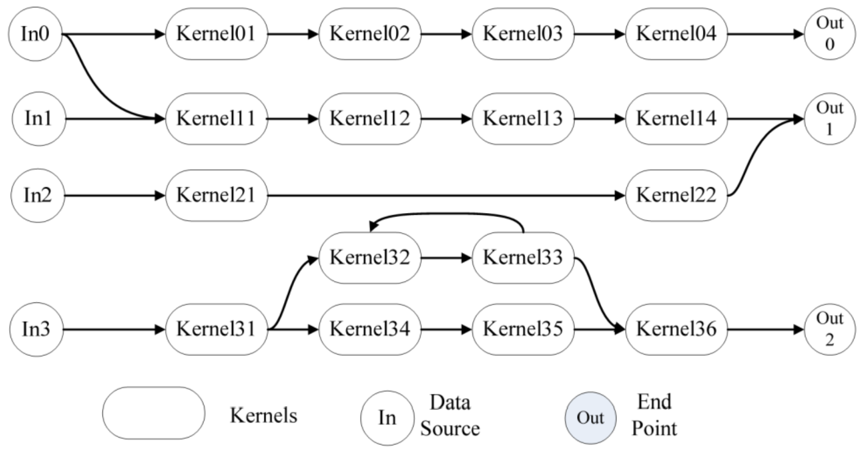
8-GPU server



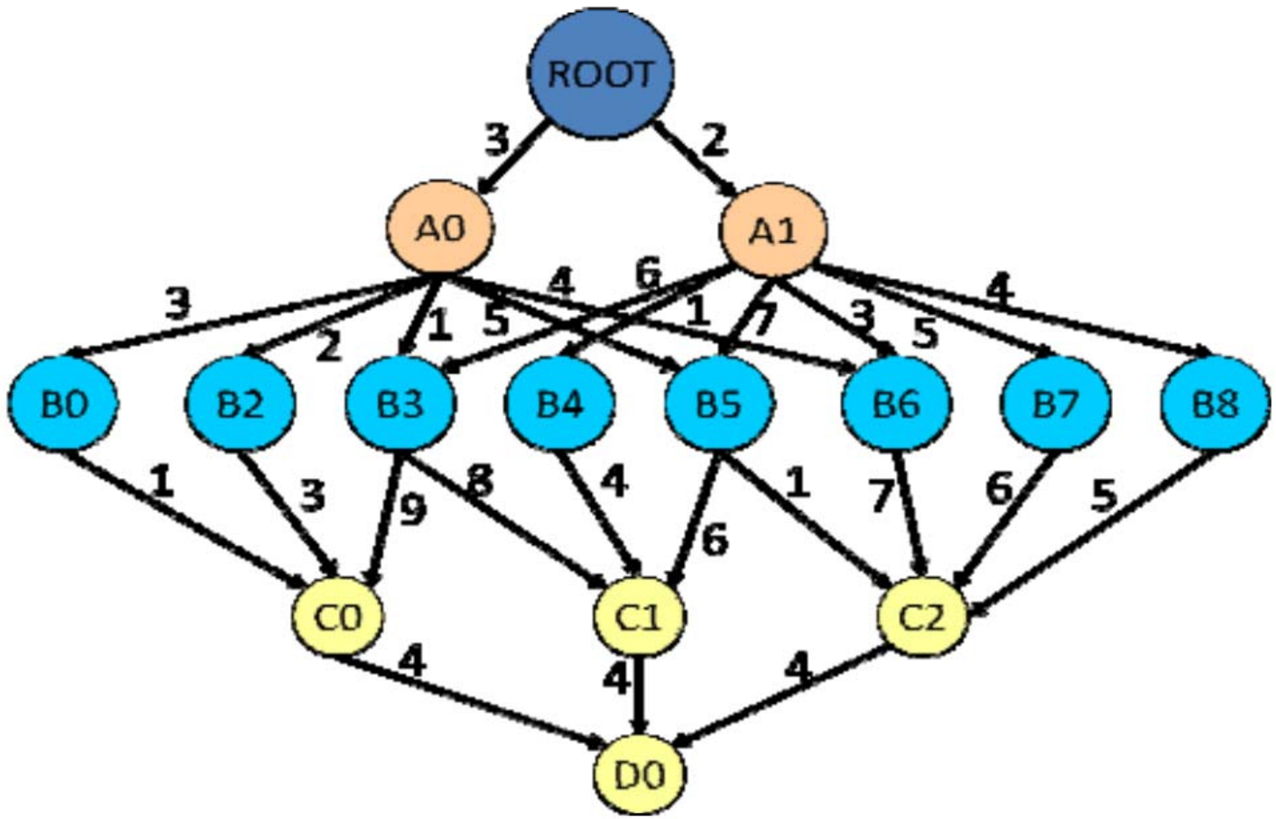
模型并行
Worker (0, 0)
模型并行
Worker (0, 1)
模型并行
Worker (1, 0)
模型并行
Worker (1, 1)
模型并行
Worker (2, 0)
模型并行
Worker (2, 1)
模型并行
Worker (3, 0)
模型并行
Worker (3, 1)
数据并行
Worker Group 0
数据并行
Worker Group 1
数据并行
Worker Group 2
数据并行
Worker Group 3
GPU 0
GPU 1
GPU 2
GPU 3
GPU 4
GPU 5
GPU 6
GPU 7
PCIe Switch
PCIe Switch
PCIe Switch
PCIe Switch
PCIe Switch
PCIe Switch
IOH
QPI
QPI
CPU 0
QPI
CPU 1

Pipeline and Stream processing in CNN





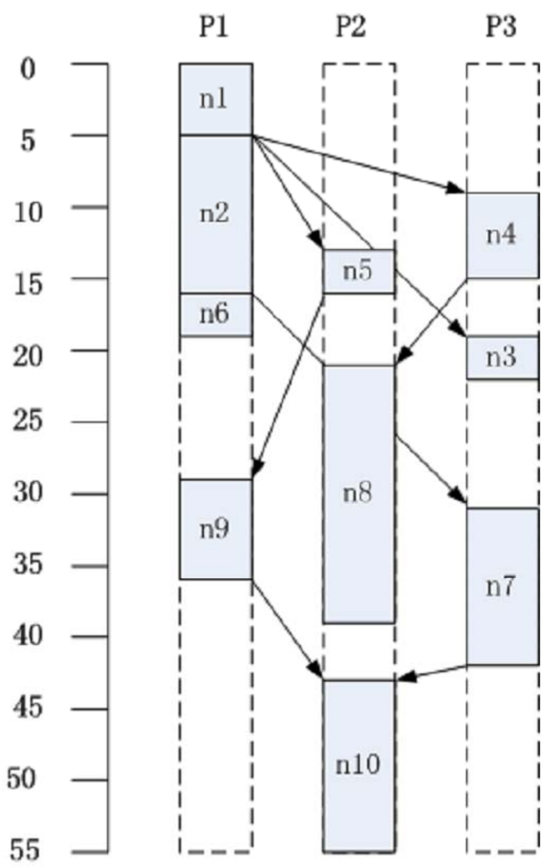
Familiar?? It’s a DAG!

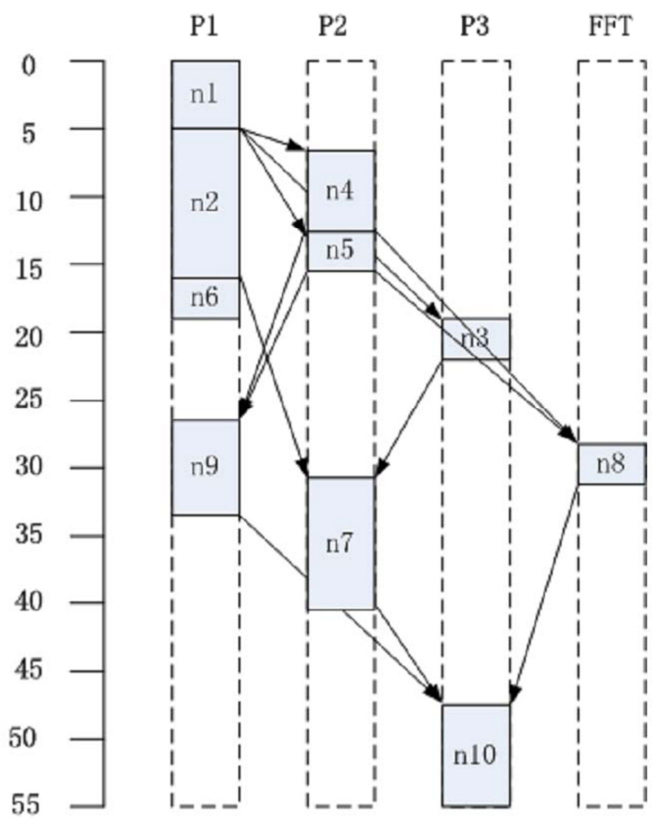


My Algorithm for DAG auto-Parallelization

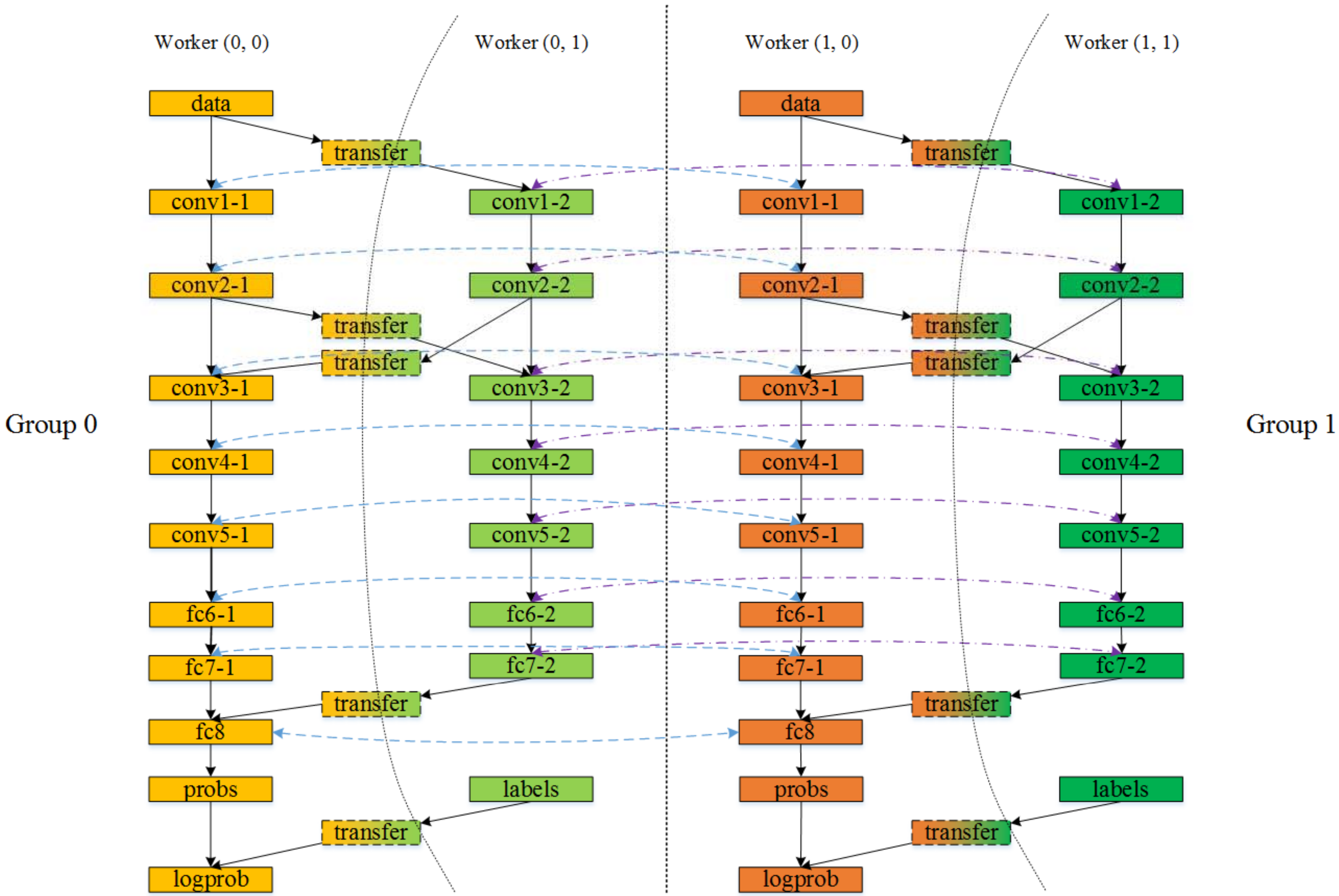
$$
\begin{aligned}
&\mathit{WHILE}\ \ Task\_List\ != \varnothing\ \mathit{DO}\\
&\quad \mathit{GET}\ first\ task\ n_i;\\
&\quad \mathit{FOR}\ \ every\ p_j\ \ \mathit{DO}\\
&\qquad EST(n_i, p_j) = \max\{available(p_j), \max_{n_m \in pred(n_i)} (AFT(n_m) + c_{m,i})\};\\
&\qquad f(n_i, p_j) = EST(n_i, p_j) + \sum_{n_m \in pred(n_i)} c_{m,i} \Big/ speed_j\\
&\quad \mathit{ENDFOR}\\
&\quad \mathit{SORT}(f(n_i, p_j));\\
&\quad \mathit{SCHEDULE}(n_i,\ p_j)\ with\ smallest\ f(n_i, p_j);\\
&\quad AST(n_i) = EST(n_i, p_j); AFT(n_i) = AST(n_i) + w'_{i,j};\\
&\quad available(p_j) = EFT(n_i, p_j);\\
&\mathit{ENDWHILE}
\end{aligned}
$$

Test Case


P1
P2
P3
0
5
10
15
20
25
30
35
40
45
50
55
n1
n2
n6
n9
n5
n8
n10
n4
n3
n7


P1
P2
P3
FFT
0
5
10
15
20
25
30
35
40
45
50
55
n1
n2
n6
n9
n4
n5
n7
n3
n10
n8

A More Complex Case



Speed with Multi-GPUs

Configuration	Speedup vs. 1 GPU
2 GPUs Model P.	1.71
2 GPUs Data P	1.85
4 GPUs Data P. + Model P.	2.52
4 GPUs Data P.	2.67

Conclusion

- GPU is very well suitable for CNN
- cuDNN is easy to use and good performance
- Multi-GPU is improving more.
- Carefully Designed parallel design on multi-GPU could get adequate scalability