Artificial Neural Nets on CUDA

ECE 5720 Introduction to Parallel Computing

Project Presentation

Zhixin Lai(zl768) Jingwen Ye(jy879) Shizhe Zhang(sz592)

Introduction of Project

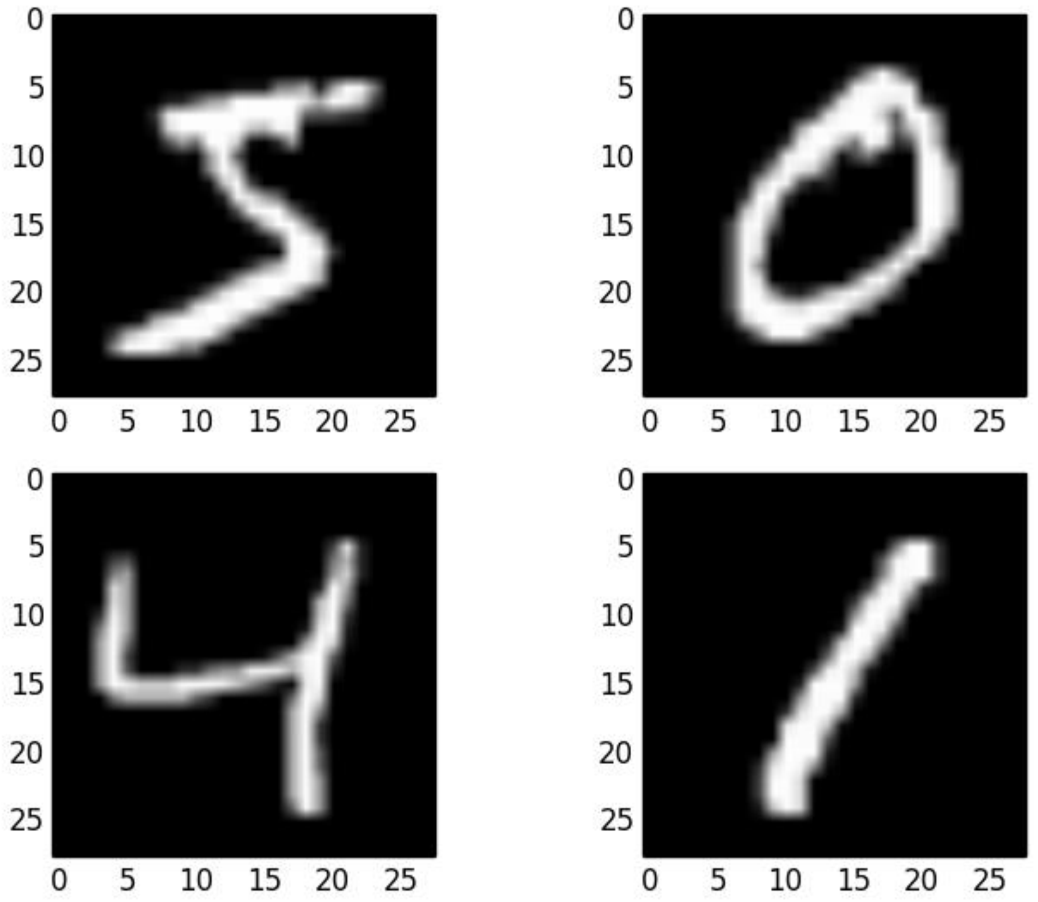
- **Goal:**

  Image classification with ANN

  Speed up with GPU
- **Dataset:**

  Handwritten digits dataset (MNIST)
- **Model:**
  Artificial Neural Networks
- **Language:**

  C/C++ (CUDA)



Model of Artificial Neural Networks

- **Input layer:**

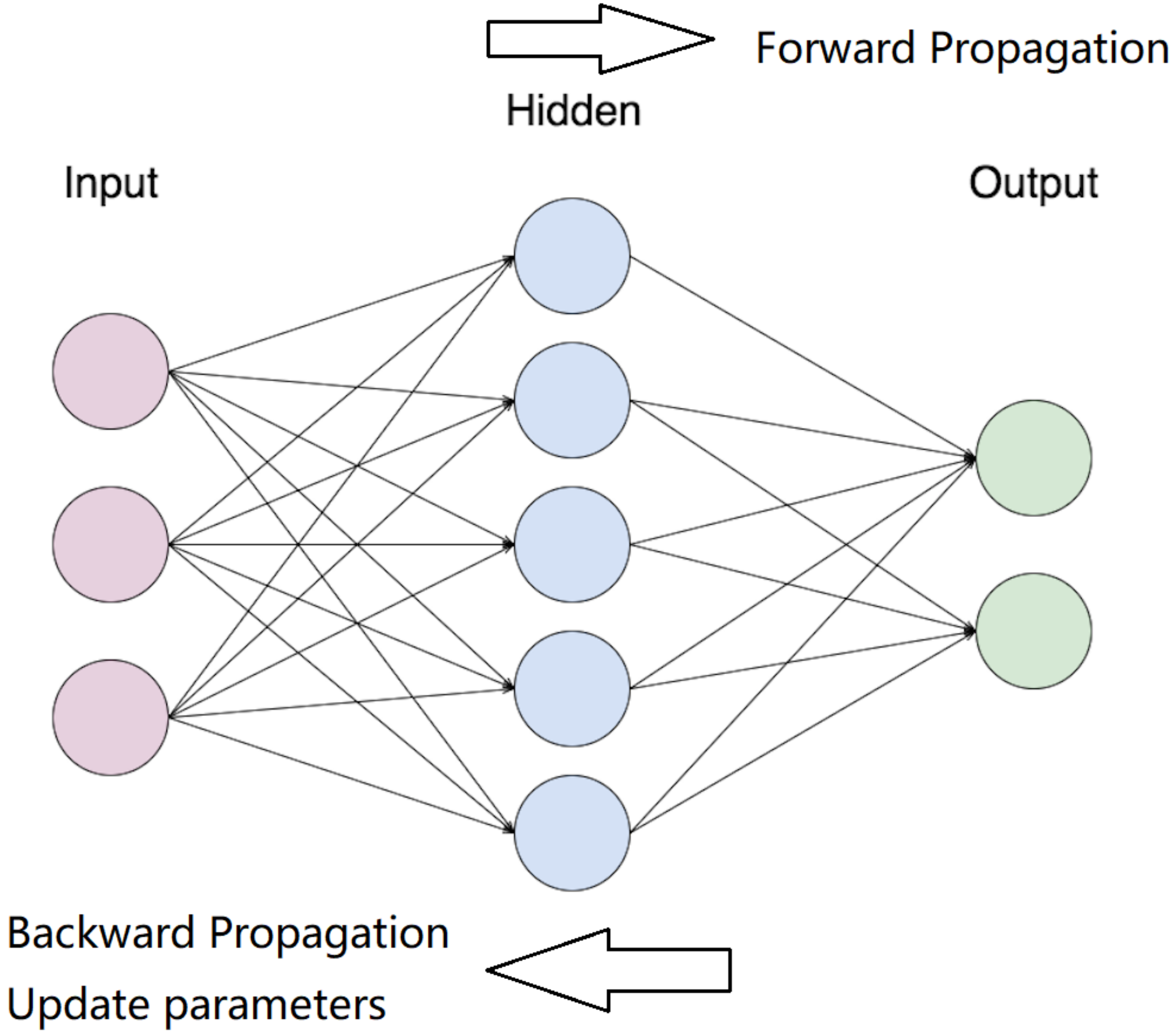
  image(14 * 14 * 964) for train

  image(14 * 14 * 414) for test
- **Hidden layer:**

  20 Nodes + Activation function (ReLU / Sigmoid)
- **Output layer:**

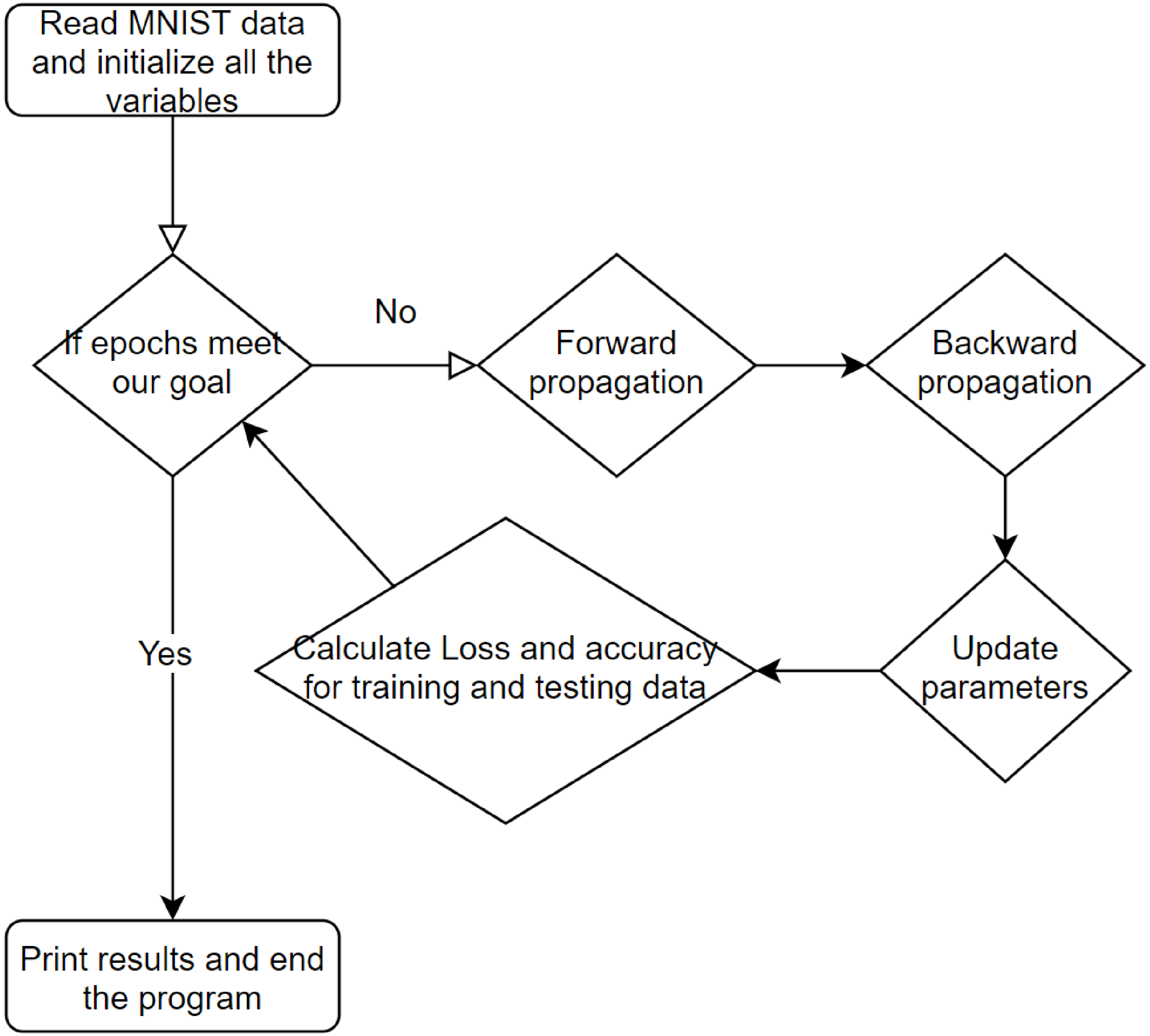
  2 Nodes + Softmax
- **Loss:**

  Cross-Entropy Loss



Logic of ANN

- Simple version of MNIST
- 14*14 per image
- 964 images for training
- 414 images for testing



Introduction of CUDA

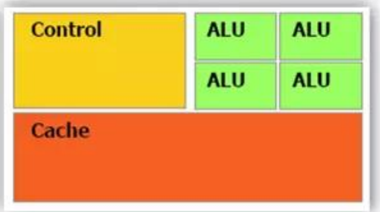
- Compute Unified Device Architecture
- Use GeForce series of Video Card GPU for computing.
- In this project, GeForce GTX980 and Tesla K40C are provided
- GPU is good for repeated tasks, such as Matrix Computation.

Powerful ALU
- Reduced operation latency

Large caches
- Convert long latency memory accesses to short latency cache accesses

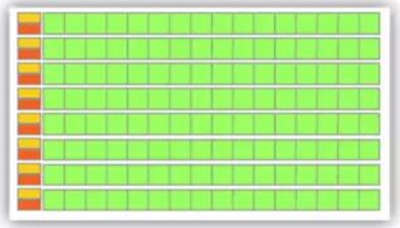
Sophisticated control
- Branch prediction for reduced branch latency
- Data forwarding for reduced data latency

CPU



* Low compute density
* Complex control logic
* Large caches (L1$/L2$, etc.)
* Optimized for serial operations
  - Fewer execution units (ALUs)
  - Higher clock speeds
* Shallow pipelines (<30 stages)
* Low Latency Tolerance
* Newer CPUs have more parallelism

GPU



* High compute density
* High Computations per Memory Access
* Built for parallel operations
  - Many parallel execution units (ALUs)
  - Graphics is the best known case of parallelism
* Deep pipelines (hundreds of stages)
* High Throughput
* High Latency Tolerance
* Newer GPUs:
  - Better flow control logic (becoming more CPU-like)
  - Scatter/Gather Memory Access
  - Don't have one-way pipelines anymore

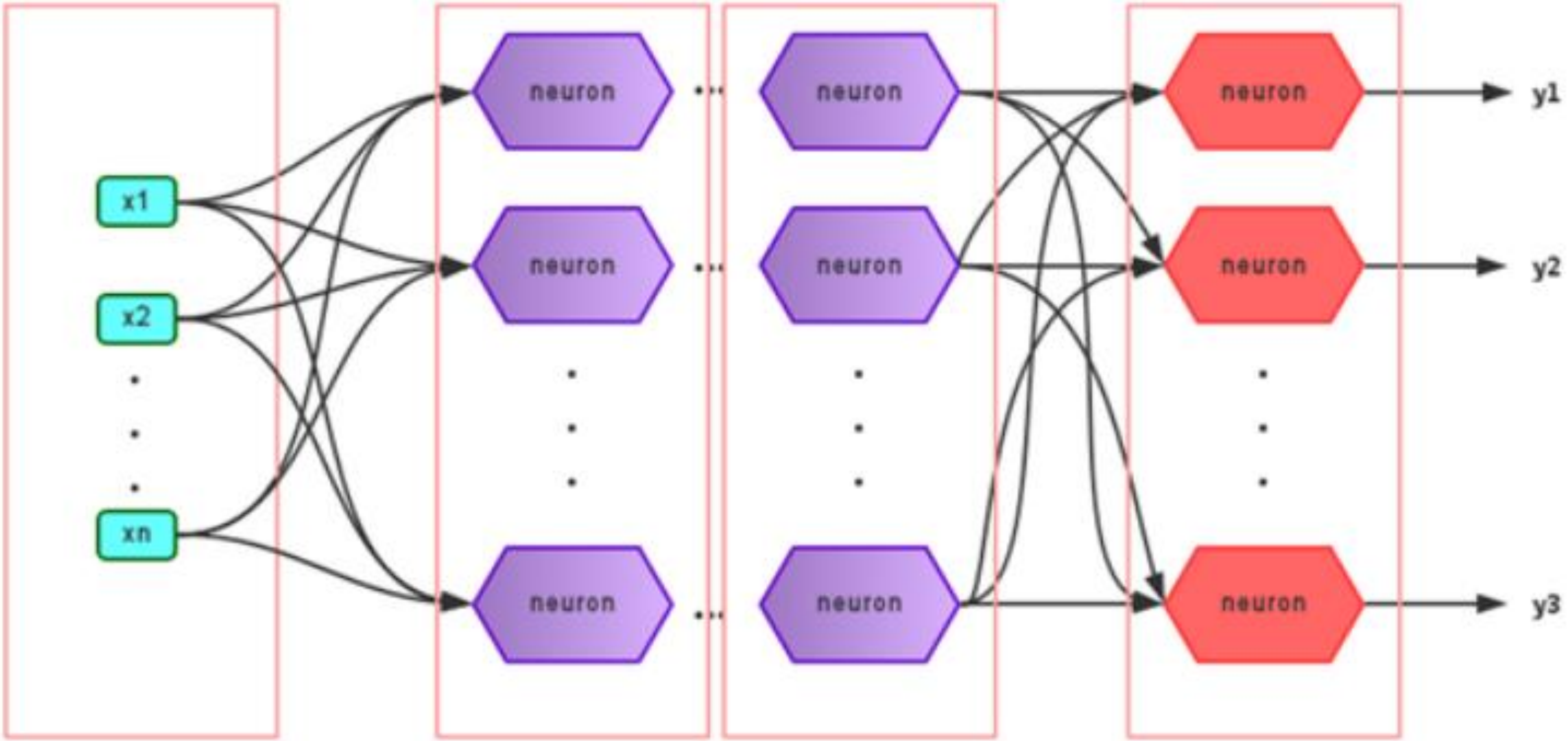
Small caches
- To boost memory throughput

Simple control
- No branch prediction
- No data forwarding

Energy efficient ALUs
- Many, long latency but heavily pipelined for high throughput

Require massive number of threads to tolerate latencies

Parallel Computing Design



Input Layer
No parallel

Hidden Layer
Parallel Computing

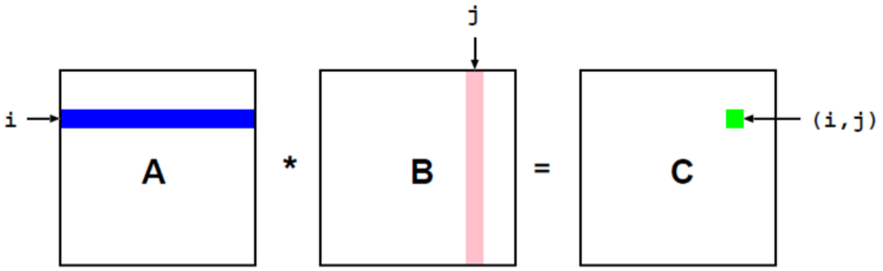
Output layer
Parallel Computing

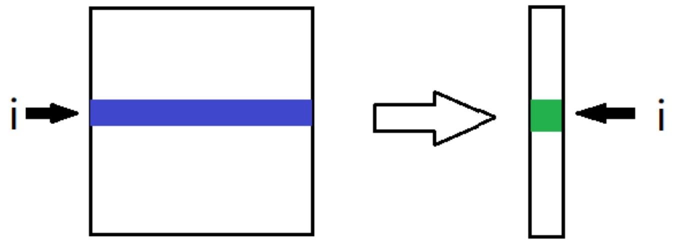
Loss
No parallel

Parallel Computing Design

(1) Forward

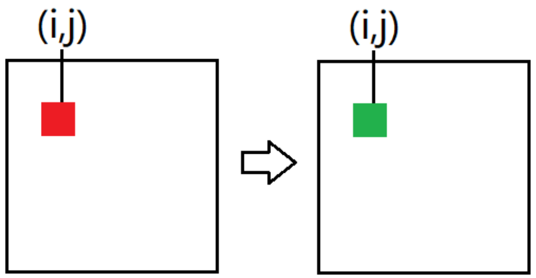
- Matrix-matrix multiply: W*X
- Softmax: Matrix elements add
- Activation function:
  Sigmoid / ReLU





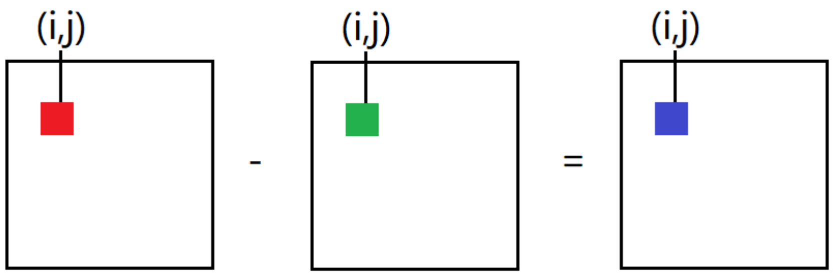
(2) Backward

- Matrix-matrix multiply: W*X
- Activation function: Sigmoid / ReLU



(3) Update parameter

- Matrix-matrix add: W-dW*rate



i= blockIdx.y * blockDim.y + threadIdx.y

j= blockIdx.x * blockDim.x + threadIdx.x

Experiment Design

(1) Different versions

- **Sequential computing version**
  Code without CUDA
- **V1: Basic version of parallel computing**
  Parallel compute for each part of ANN (Forward, backward....)
- **V2: Reduce times of memory copy and allocation**
  Some duplicate memory copy between CPU and GPU are removed
  Some duplicate memory allocation for each epoch can be removed
- **V3: Remove the if-else**
  Remove if-else of __global__ code ran on device to avoid warp divergence
- **V4: Use stream**
  Use stream to allows overlapping between CPU(Memory copy) and GPU
  cudaMemcpyAsync(), cudaStreamSynchronize()

(2) Each parallel computing experiment

- Block size is 1*1, 2*2, 4*4, 8*8, 16*16, 32*32

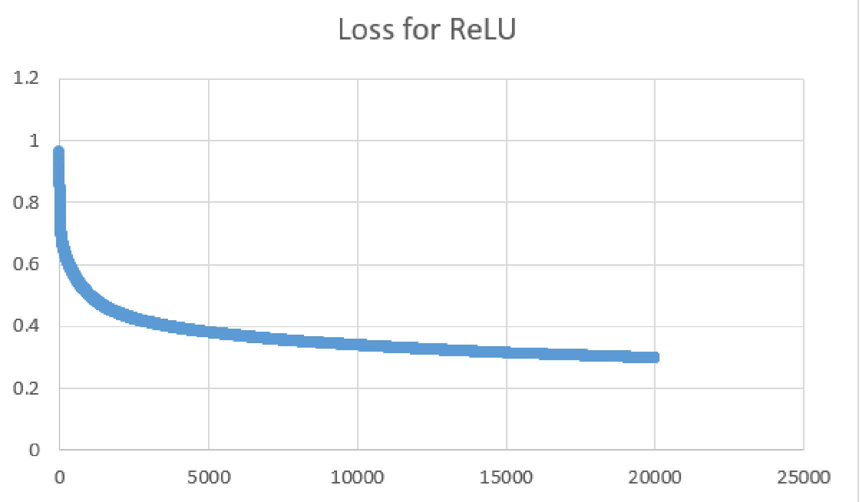
Result and Analysis

(1) Sequential computing

Loss:

Sigmoid:

ReLU:



Accuracy:

Sigmoid:

Training set: 0.852697, testing set: 0.842995

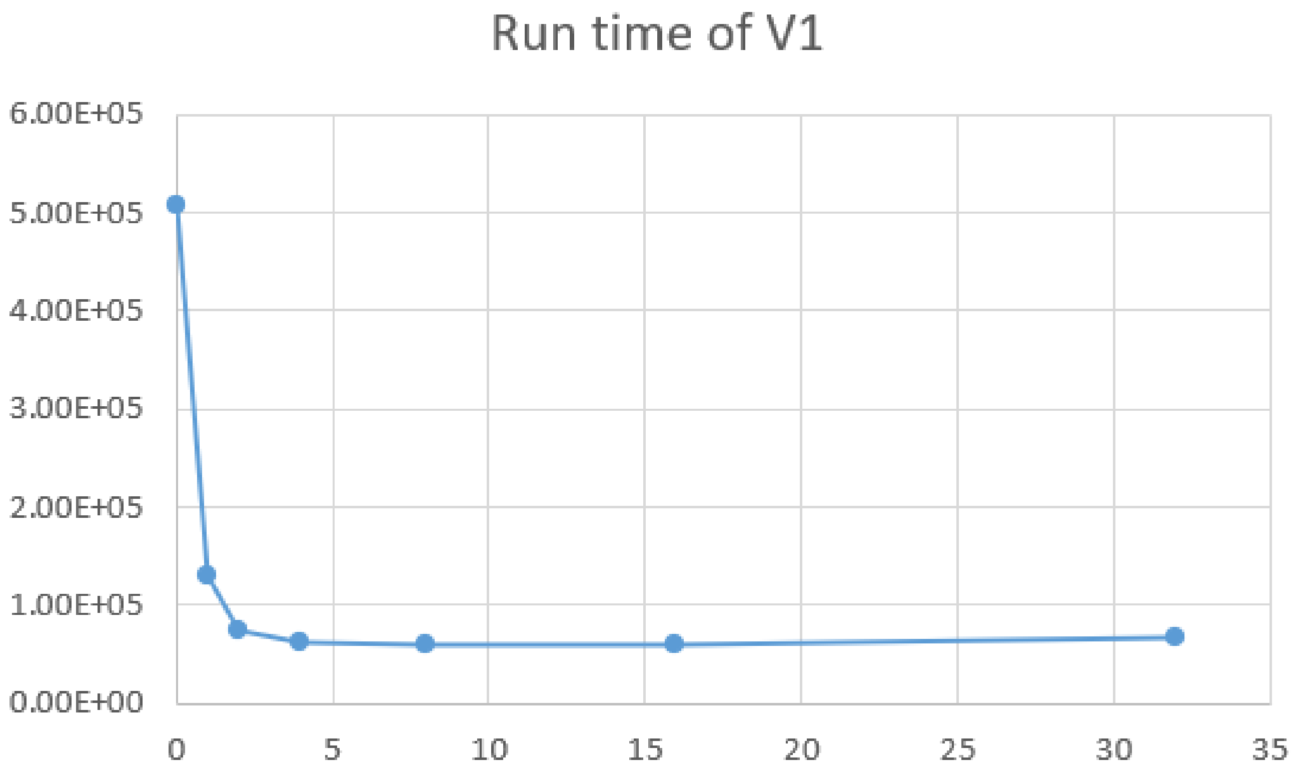
ReLU:

Training set: 0.870332, testing set: 0.840580

Convergence speed: ReLU > Sigmoid

Result and Analysis

(1) V1: Basic version of parallel computing



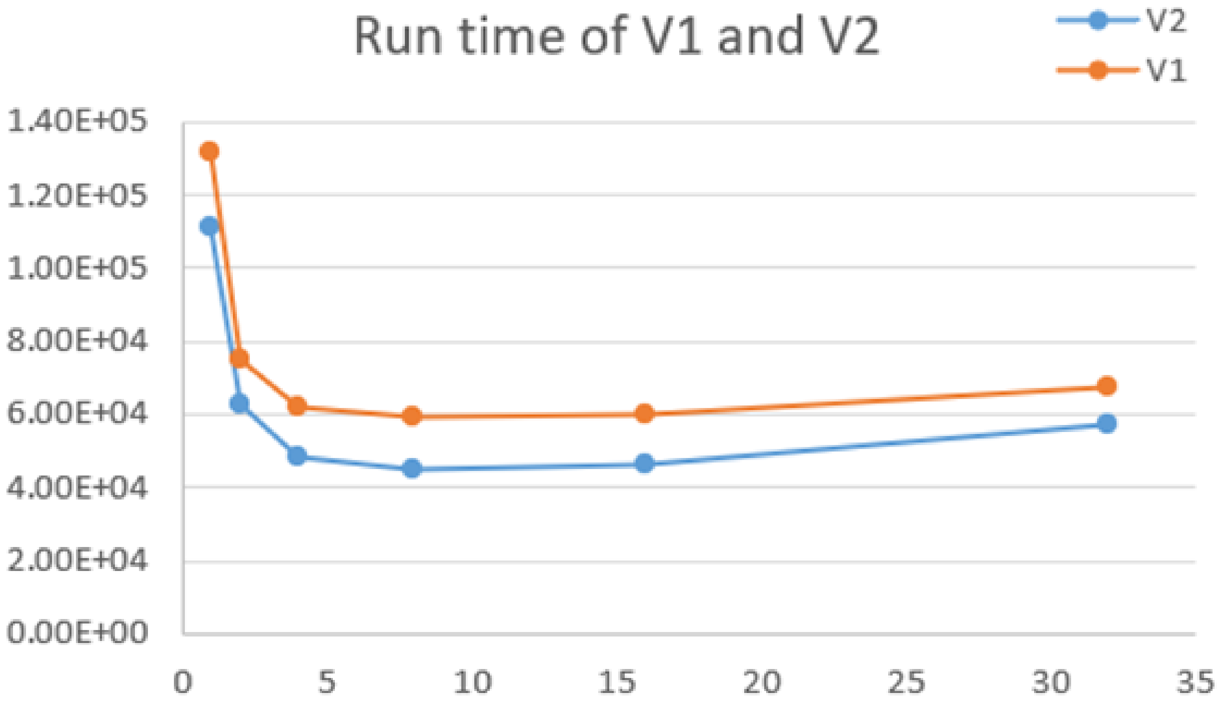
x - block size (0 means series computing without CUDA)
y - run time (ms)

- **Parallel vs Sequential**
  Speed up by 10x with CUDA

- **Influence of block size**
  Run time decreases first and increases then with the increase of the block size

Result and Analysis

(2) V2: Reduce times of memory copy and allocation



x - block size; y - run time (ms)

- **Compared with V1**

Reducing the times of memory copy and allocation can save running time

Result and Analysis

(3) V3: Remove the if-else

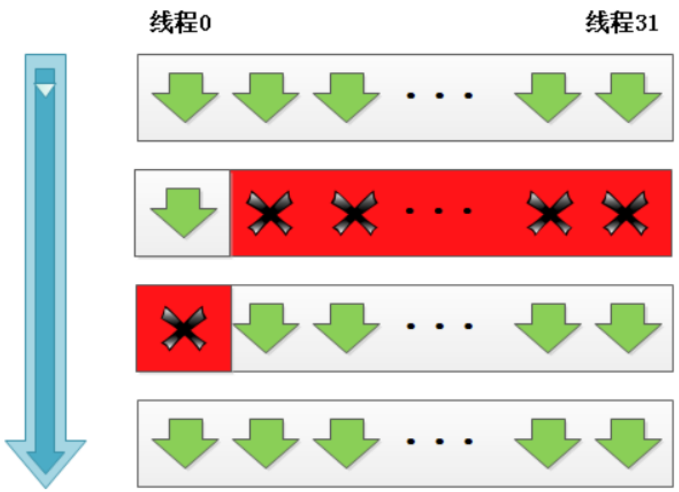
```
__global__ void HiddenLayer(double* dev_X, double* d
{
  .............

  // Sigmoid
  if (acti_type == 1)
   dev_A1(i,j) = 1 / (1 + exp(0 - dev_Z1(i,j)));

  // ReLU
  if (acti_type == 2) {
   if (dev_Z1(i,j) < 0)
    dev_A1(i,j) = 0;
   if (dev_Z1(i,j) >= 0)
    dev_A1(i,j) = dev_Z1(i,j);
  }

 ...............
}
```

warp divergence



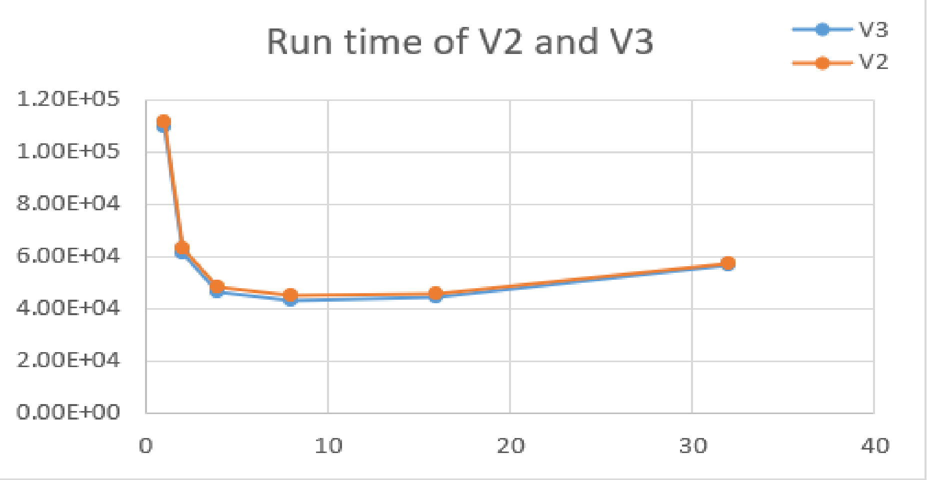
divide

```
__global__ void HiddenLayer_ReLU(double* dev_X, double* dev
{
  ..........
  dev_A1(i,j) = dev_Z1(i,j) * (dev_Z1(i,j)>0);
  ..........
}
```

```
__global__ void HiddenLayer_Sigmoid(double* dev_X,
{
  ...............
  dev_A1(i,j) = 1 / (1 + exp(0 - dev_Z1(i,j)));
  ...............
}
```

Result and Analysis

(3) V3: Remove the if-else



Run time for the whole code

	V2	V3
1	1.71E+03	1.71E+03
2	5.28E+02	5.22E+02
4	1.98E+02	1.97E+02
8	9.64E+01	9.56E+01
16	6.72E+01	6.67E+01
32	9.15E+01	8.25E+01

Run time for the local code of if-else

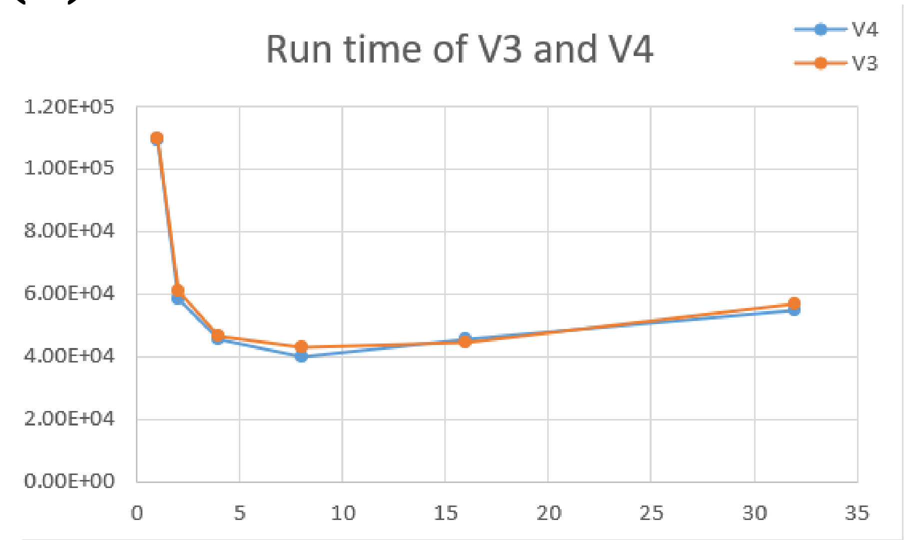
- **Compared with V2**

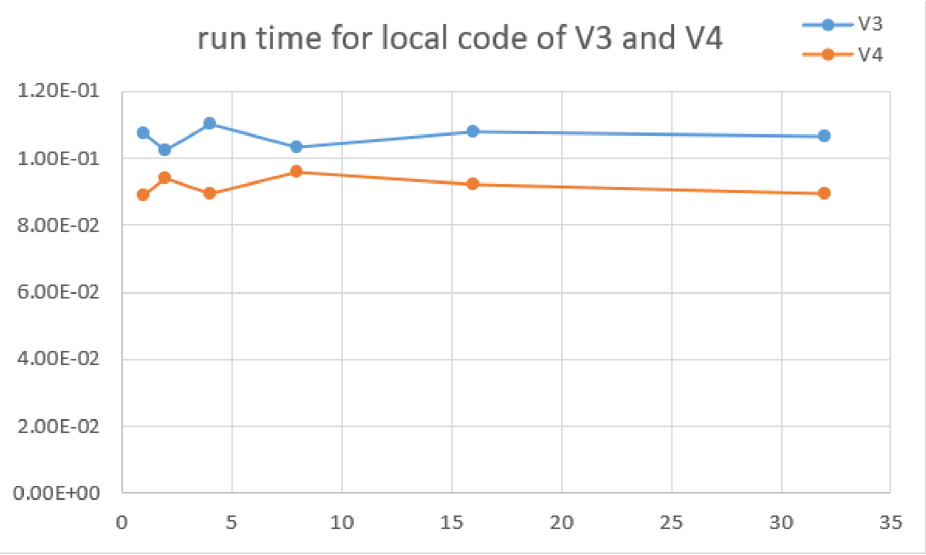
Remove if-else of code ran on device to avoid warp divergence. It helps save time.

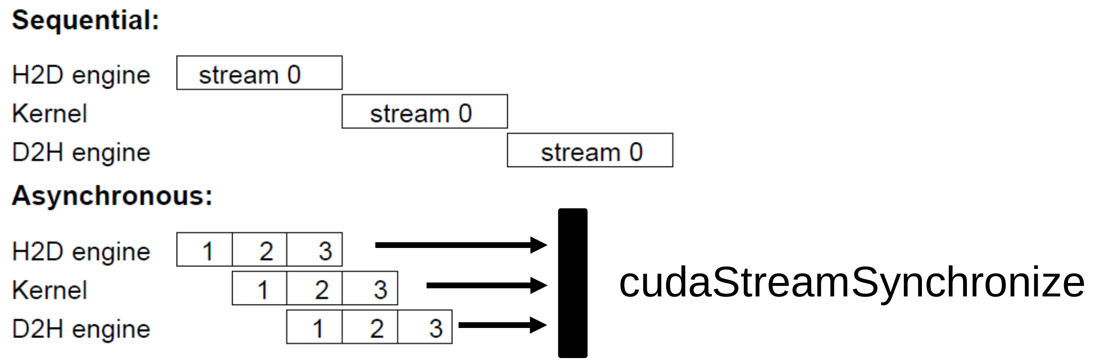
The local run time of V4 is less than the local run time of V3. Therefore, avoid warp divergence helps save time.

Result and Analysis

(4) V4: Use stream





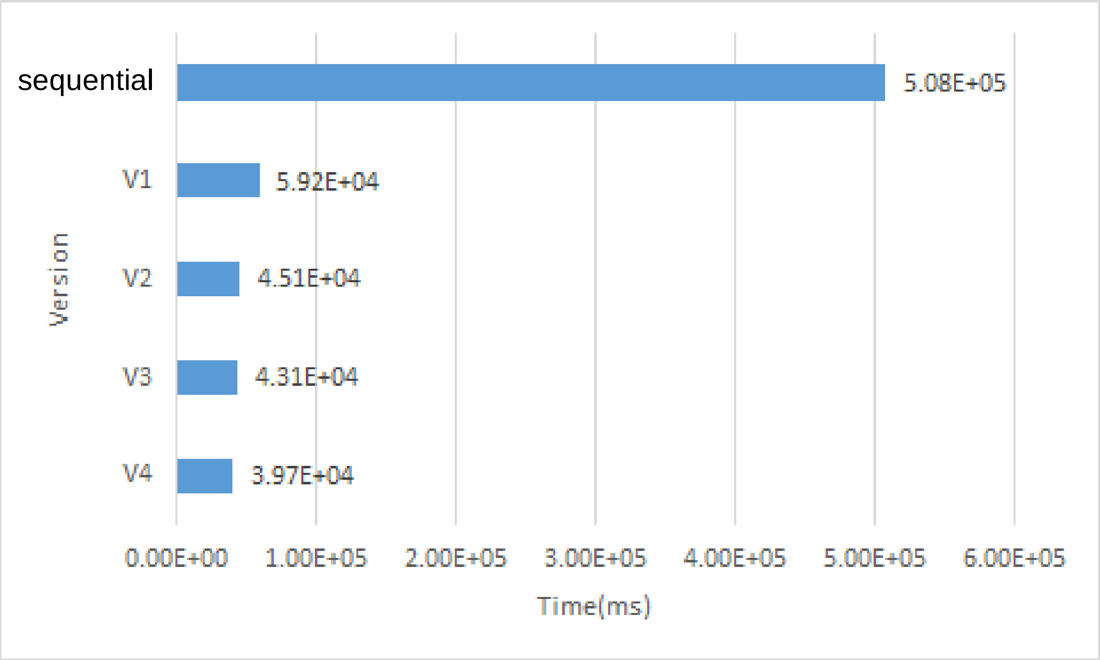


- **Compared with V3**

Use stream to allows overlapping between CPU(Memory copy) and GPU.

The local run time of V4 is less than the local run time of V3. Therefore, stream helps save time.

Conclusion



Series no CUDA: 5.08E+05 ms
V1 block size = 8: 5.92E+04 ms
V2 block size = 8: 4.51E+04 ms
V3 block size = 8: 4.31E+04 ms
V4 block size = 8: 3.97E+04 ms

Run time decreases!
Efficiency improved!

- Parallel computing improves the efficiency by more than 12 times to Sequential computing;
- Block size influence the efficiency and find the best size is also important;
- Remove if-else code on device to avoid warp divergence and improve efficiency;
- Using stream to allow overlapping between CPU and GPU improves the efficiency.

Much more can do to improve efficiency: optimize the memory access, dynamic parallelism…………

Thank You

Q & A