Available online at www.isr-publications.com/jmcs
J. Math. Computer Sci., 18 (2018), 1–10
Research Article

Online: ISSN 2008-949x

Journal of Mathematics and Computer Science

Journal Homepage: www.tjmcs.com - www.isr-publications.com/jmcs

Accelerate the Training Process of BP neural Network with CUDA Technology

Yinfen Xie

School of Information Science and Technology, Linyi University, Linyi, Shandong 276005, P. R. China.

Abstract

NVIDIA GPUs is a typical Stream Processor device, and have a high performance of floating-point operations. CUDA uses a bran-new computing architecture, and provides greater computing ability for large scale data computing application than CPU. The learning algorithm of BP neural network has a high compute-intensive and rules, and be very suitable for the Stream Processor architecture. Using CUDA technology, the CUBLAS mathematical library and self-Kernels library, supported by NV Geforce GTX280 as hardware, modify the study algorithm ecome parallel, definite a parallel data structure, and describe the mapping mechanism for computing tasks on CUDA and the key algorithm. Compare the parallel study algorithm achieved on GTX280 with the serial algorithm on CPU in a simulation experiment. Improve the training time by as much as nearly 15 times.

Keywords: Stream processor, GTX280, CUDA, CUBLAS, BP neural network.

1. Introduction

With the continuous development of stream computing architecture [7, 6], stream processing and stream computing inaugurate a new era in extensive parallel computing. Stream processor can be considered as the concrete implementation of stream computing model, as well as NVIDIA GPUs being typical stream processor device. But GPGPU (General-Purpose computing on Graphics Processing) [5, 10] greatly limit the powerful parallel processing ability of GPU for too much reliance on graphical API interface. CUDA (Compute Unified Device Architecture) technique provides direct access interface of hardware, and extricates from using graphical API interface to access GPU which is indirect realization of GPGPU. CUDA uses a bran-new computing architecture, and provides greater computing ability for large scale data computing application than CPU.

2. Preparation

2.1. GTX280

As a new generation of Stream Processor device with unified architecture, NVIDIA GTX280 works at a frequency of 1296 MHz. It has thirty multi-core Stream Processors (SMs) and every SM has eight Stream

Email address: xieyinfen@lyu.edu.cn (Yinfen Xie)

doi: 10.22436/jmcs.018.01.01

Received 2017-04-14

processors (SPs). It can execute two addition instructions and a multiply instruction at the same time in its special Dual-Issue mode. So the formula of GTX200 serial for computing capability of theoretical peak floating-point is

number of Stream Processors $\times$ number of instructions $\times$ frequency=$30\times8\times3\times$ 1296=933GFLOPS.

Furthermore, GTX280 provides double- precision support and accord with official Version 1.3 [4] of computing capability.

2.2. CUDA

CUDA is a new infrastructure, which can help users solve complex problem in commerce, industry and science [1, 3]. It is a complete solution which provide direct access interface for hardware without depending on graphic interface to access just like international way. In infrastructure, CUDA use hardware resource form users by a bran-new computing architecture, then it can provide more powerful computing capability than CPU for extensive calculation. Using C language for programming language can provide a great deal of high performance computing instruction development ability and it make developers establish a more effective solution for intensive data computing basing on powerful computing capability of GPU.

On the components of architecture, CUDA includes development library, operation period environment and drive. Development library is application development library based on CUDA technique. At present the official version 2.0 provides two normative mathematical libraries, realization of CUFFT (Fast Fourier Transformation) [4, 3] and CUBLAS(Basic Linear Algebra) [4, 3]. What the two mathematical libraries solve is typical extensive parallel computing which is a very common type is in intensive data computing. Basing on development library, we can fast conveniently create the computing applications. In addition, the developer can realize more libraries.

Operation period environment provides application development interface and components including the definition of basic data type and all kinds of functions of computing, type conversions, memory management, device accessing and executing or scheduling. The programme codes based on CUDA development are divided into two kinds when actually executed, and one is host code being run in CPU, another is device Code in GPU.

3. Learning algorithm of BP neural networks

3.1. *Basic principal of BP learning algorithm*

Rumelhart and McClelland put forward learning algorithm of error back propagation of BP Network [9, 8]. Its basic idea is least squares, using gradient searching technique to minimize the error of network's real output and expected output value.

Learning process of BP algorithm includes two processes, forward propagation and back propagation. In process of forward propagation, input information is successively processed from input layer to implicit layer and transported to output layer. The state of neurons in each layer only affects the next layer neural status. If the value is unexpected in output layer, it turns to back propagation and backtracked error massage along the connecting path, modifying the weight of neural node of every layer to minimize the error massage.

3.2. *Computing Process of BP Learning Algorithm in batch mode*

Firstly, if batch mode being adopted, only after all output specimens are submitted, network weight and bias would be updated. Secondly adjustment rules of network weight adopt the fast Gradient Descent method.

Let input layer have n neurons, implicit layer has p neurons, output layer has q neurons, input vector is $x = (x_1, x_2, \ldots, x_n)$, input vector of implicit layer is $hi = (hi_1, hi_2, \ldots, hi_p)$, output vector of output layer is $ho = (ho_1, ho_2, \ldots, ho_p)$, input vector of output layer is $yo = (yo_1, yo_2, \ldots, yo_q)$, expected output vector is $yo = (yo_1, yo_2, \ldots, yo_q)$, connecting weigh of input layer and middle layer is w_{ih}, connecting weigh of implicit layer and output layer is *who*, threshold value of each implicit layer neurons is b_h, threshold value of each output layer neurons is b_o, Number of specimen data is $\{k = 1, 2, \ldots, m,\}$ Sigmoid activation function is f(), error function is

$$e = \frac{1}{2}\sum_{o}^{q}(d_o(k) - yo_o(k))^2.$$

Computing process of BP learning algorithm in batch mode are shown as follows (1)-(12).

(1) Initialization of network. Assign every connecting weigh a random number in interval (-1,1), specify computing precision value ε, learning velocity η and maximum learning time M.

(2) Select the first K output specimen $x(k) = (x_1(k), x_2(k), \ldots, x_n(k))$ randomly from m output specimens, corresponding expected output is $d_o(k) = (d_1(k), d_2(k), \ldots, d_q(k))$.

(3) Compute input and output of implicit layer naturals.

$$hi_h(k) = \sum_{i=1}^{n} w_{ih}x_i(k) - b_h \ , \quad h = 1, 2, \cdots, p \ ,$$

$$ho_h(k) = f(hi_h(k)) \ , \quad h = 1, 2, \cdots, p \ ,$$

$$yi_o(k) = \sum_{h=1}^{p} w_{ho}ho_h(k) - b_o \ , \quad o = 1, 2, \cdots, q \ ,$$

$$yo_o(k) = f(yi_o(k)) \ , \quad o = 1, 2, \cdots, q \ .$$

(4) Repeat step (2) and (3), select next learning specimen and corresponding expected output to compute, until all specimens input are completed.

(5) Compute global error,

$$E = \frac{1}{2}\sum_{k=1}^{m}\sum_{0-1}^{q}(d_o(k) - yo_o(k))^2.$$

(6) Use network expected output and actual output to compute partial derivative of output layer neuron.

$$\delta_o(k) = (d_o(k) - yo_o(k))f'(yi_o(k)) \ , \quad o = 1, 2, \cdots, q \ .$$

(7) Use connecting weigh from implicit layer to output layer, $\delta_o(k)$ of output layer and partial derivative $\delta_{h(}k)$ of input computing error function of implicit layer to neuron of implicit layer.

$$\delta_h(k) = \left(\sum_{o=1}^{q}\delta_o(k)w_{ho}\right)f'(hi_h(k)) \ , \quad h = 1, 2, \cdots, p \ .$$

(8) Using learning efficiency η, output layer neuron $\delta_o(k)$ and output computing connecting weigh w_{ho} corrections of implicit layer neuron,

$$\Delta w_{ho}(k) = \eta\delta_o(k)ho_h(k) \ , \quad o = 1, 2, \cdots, q, \ \ h = 1, 2, \cdots, p \ .$$

(9) Using learning efficiency η, implicit layer neuron $\delta_h(k)$ and input computing connecting weigh w_{ih} corrections of input layer neuron,

$$\Delta w_{ih}(k) = \eta\delta_h(k)x_i(k) \ , \quad i = 1,2,\cdots,n, \quad h = 1,2,\cdots,p \ .$$

(10) Repeat step (6) to (9), computing corrections of w_{ih} and w_{ho} of all specimens, $k = k+1$, until specimens number $k > m$.

(11) Using the sum of corrections of all specimens connecting weigh to correct connecting weigh w_{ih} and who. The t is current learning times, $\{t = 0, 1, \ldots, M-1\}$.

$$w_{ho}^{t+1} = w_{ho}^{t} + \sum_{k=1}^{m} \Delta w_{ho}^{t}(k) \ , \quad o = 1,2,\cdots,q, \quad h = 1,2,\cdots,p \ ,$$

$$w_{ih}^{t+1} = w_{ih}^{t} + \sum_{k=1}^{m} \Delta w_{ih}^{t}(k) \ , \quad i = 1,2,\cdots,n, \quad h = 1,2,\cdots,p \ .$$

(12) Judging learning times is greater than the set maximum times or not, if $t+1 > M$, then stop network learning process, else return step (2) and go on next learning.

4. Learning algorithm parallelization of BP neural networks

4.1. Pending problem

The pending problem may be described vividly through Fig.1. BP neural network identifies the status of the located area of two-dimensional point sets in *xy* coordinate system.

CUBLAS standardized mathematical computing library of CUDA is used being coordinated some self-Kernels to create the third-party development library and modify the learning algorithm of BP neural network to become parallel, so the learning speed is improved on the premise of keeping learning convergence and precision to the extent.

The status of point sets in Fig.1 is shown in three binaries. There are two stationary areas in Fig.1, one is right triangle, another is rectangle, and their ranges of xy coordinate axes are both [-2, 2]. In Fig.1 the point set marked ◯ shows not in above two areas or the boundary, then the status is 100, namely 3. The point set marked + shows in right triangle area or the boundary, the status is 010, namely 2. The point set marked shows in rectangle area or the boundary, the status is 001, namely 1. Thus the construction of network structure is shown at Fig.2.

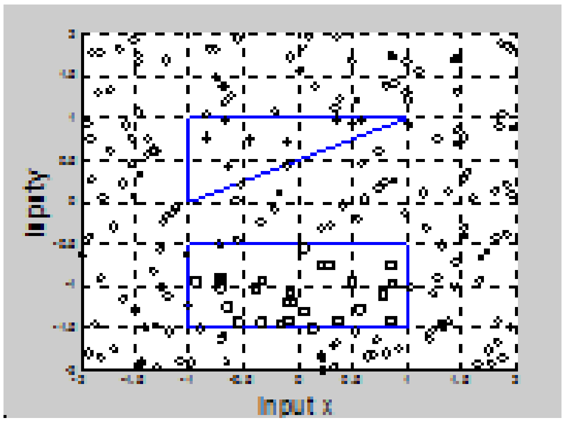

Figure 1: Status indentation of Located Area of Two-dimensional Point Sets

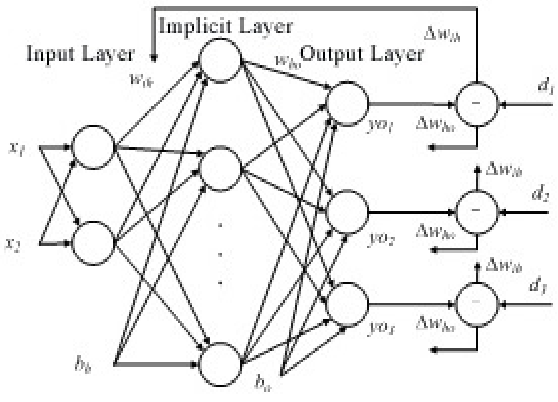

Figure 2: BP Neuron Network Structure ″2-10-3″ Used to Identify Point Set Status

The “2-10-3” BP Neuron Network Structure is designed for above pending problem. The symbol “2” shows two inputs, using input vector x =(x_1,x_2) to indicate point’s coordinate and x_1, x_2 belongs to [-2, 2]. The symbol ″10″ indicates that there are ten neurons in implicit layer. The symbol “3” shows three outputs, output vector is yo = (yo_1, yo_2, yo_3) and yo_1, yo_2, yo_3 belongs to {0,1}. So output and input in output layer and implicit layer both use logarithm Sigmoid to active function “logsig”.

$$f(net) = \frac{1}{1+e^{-net}}.$$

Logarithm Sigmoid maps the input range of neuron from (-∞, +∞) to (0, 1), which is Differentiable function applying to BP trained neuron. Its derived function is

$$f'(net) = \frac{1}{1+e^{-net}} - \frac{1}{(1+e^{-net})^2} = y(1-y).$$

4.2. *Mapping of data structure*

Organization and management of data structure is always an important part of parallelization. For increasing the efficiency of CUDA, to definition a data structure to adapt well to CUDA structure has a far-reaching significance.

Data decomposition in CUDA is a process of recombining and re-decomposition. It needs to recombine data structure of serial algorithm and mapping to CUDA, and then recognize re-decomposition. However, in decomposition process, index mechanism is dependable in fact. Specifically, how CUDA decide effective location of different data for thread operation is achieved by index mechanism, such one-dimensional vector as *BlockIdx(x,y)* and *ThreadIdx(x,y,z)*. But that is a decision that the data structure used in CUDA must satisfy a certain regulation. How to well mapping the data structure to CUDA and construct certain data decomposition regulation to adapt to index strategy of data location are the key to follow-up work.

The following is parallel learning algorithm of BP neural network.

Parallel learning algorithm of BP neural network is given in detail as follows.

Let input vector dimensionality *InDim=2*, specimen number *SamNum=200*, implicit layer neural number *HiddenUnitNum=10*, output vector dimensionality *OutDim=3*.

4.2.1. *Data structure in Host*

- Input specimen matrix

h_SamIn [InDim][SamNum]:

$$h\_SamIn = \left\{ \begin{array}{cccc} x_1(1) & x_1(2) & \cdots & x_1(200) \\ x_2(1) & x_2(2) & \cdots & x_2(200) \end{array} \right\}.$$

- Object output matrix
  h_SamOut[OutDim][SamNum]:

$$h\_SamOut = \left\{ \begin{array}{cccc} d_1(1) & d_1(2) & \cdots & d_1(200) \\ d_2(1) & d_2(2) & \cdots & d_2(200) \\ d_3(1) & d_3(2) & \cdots & d_3(200) \end{array} \right\}.$$

- Weigh from input layer to implicit layer wih matrix (including threshold bh)
  h_W1Ex[HiddenUnitNum] [InDim+1]:

$$h\_W1Ex = \left\{ \begin{array}{ccc} w_{11} & w_{21} & b_1 \\ w_{12} & w_{22} & b_2 \\ \vdots & \vdots & \vdots \\ w_{110} & w_{210} & b_3 \end{array} \right\}.$$

- Weigh from implicit layer to output layer w_{ho} matrix (including threshold b_o)
  h_W2Ex[OutDim][HiddenUnitNum+1]

$$h\_W2Ex = \left\{ \begin{array}{ccccc} w_{11} & w_{21} & \cdots & w_{101} & b_1 \\ w_{12} & w_{22} & \cdots & w_{102} & b_2 \\ w_{13} & w_{23} & \cdots & w_{103} & b_3 \end{array} \right\}.$$

- Error mean square array *Error[M]*

$$Error = \left\{ \begin{array}{cccc} E_1 & E_2 & \cdots & E_M \end{array} \right\}.$$

4.2.2. Data structure of Host mapping to Device

- Input specimen array *d_SamInEx[(InDim+1)*SamNum]*, mapping from *h_SamIn*, sequence is main and among it added full ”1” line is threshold coefficient.

$$d\_SamInEx = \{x1(1) \quad x2(1) \quad 1.0 \quad x1(2) \quad x2(2) \quad 1.0 \quad \cdots \quad x1(200) \quad x2(200) \quad 1.0\}.$$

- Object output array *d_SamOut[OutDim*SamNum]*, mapping *fromh_SamOut*, and sequence is main.

$$d\_SamOut = \{d_1(1) \quad d_2(1) \quad d_3(1) \quad d_1(2) \quad d_2(2) \quad d_3(2) \quad \cdots \quad d_1(200) \quad d_2(200) \quad d_3(200)\}.$$

- Weigh from input layer to implicit layer w_{ih}(including threshold b_h) *d_W1Ex[HiddenUnitNum* (InDim+1)]*, mapping from *h_W1Ex*, and sequence is main.

$$d\_W1Ex = \{w_{11} \quad w_{12} \quad \cdots \quad w_{110} \quad w_{21} \quad w_{22} \quad \cdots \quad w_{210} \quad b_1 \quad b_2 \quad \cdots b_{10}\}.$$

- Weigh from implicit layer to output layer w_{ho} matrix (including threshold b_o) *d_W2Ex[OutDim* (HiddenUnitNum+1)]*, mapping *fromh_W2Ex*, and sequence is main.

$$d\_W2Ex = \{w_{11} \quad w_{12} \quad w_{13} \quad w_{21} \quad w_{22} \quad w_{23} \quad \cdots \quad w_{101} \quad w_{102} \quad w_{103} \quad b_1 \quad b_2 \quad b_3\}.$$

4.2.3. Data structure using for subtotaling in Device

- Output array of implicit layer HiddenOutEx[(HiddenUnitNum+1)*SamNum],

$$\mathrm{HiddenOutE}\ x = \{ho_1(1) \quad ho_2(1) \quad \cdots \quad ho_{10}(1) \quad \cdots \quad ho_1(200) \quad ho_2(200) \quad \cdots \quad ho_{10}(200)\}.$$

- Output array of actual network NetworkOut[OutDim*SamNum], and sequence is main.

$$\begin{aligned}\mathrm{NetworkOut} = \{ & yo_1(1) \quad yo_2(1) \quad yo_3(1) \quad yo_1(2) \quad yo_2(2) \\ & yo_3(2) \quad \cdots \quad yo_1(200) \quad yo_2(200) \quad yo_3(200)\}.\end{aligned}$$

- Weigh modifier from input layer to implicit layer Delta1[HiddenUnitNum *SamNum], and sequence is main.

$$\mathrm{Delta1} = \{\Delta w_{i1}(1) \quad \Delta w_{i2}(1) \quad \cdots \quad \Delta w_{i10}(1) \quad \cdots \quad \Delta w_{i10}(200) \quad \Delta w_{i10}(200) \quad \cdots \quad \Delta w_{i10}(200)\}$$

- Weigh modifier from implicit layer to output layer Delta2[Out*SamNum], and sequence is main.

$$\mathrm{Delta2} = \{\Delta w_{h1}(1) \quad \Delta w_{h2}(1) \quad \Delta w_{h3}(1) \quad \cdots \quad \Delta w_{h1}(200) \quad \Delta w_{h2}(200) \quad \Delta w_{h3}(200)\}.$$

4.3. *Primary loading process*

The primary loading process is included five processes: CUDA hardware initialization, CUBLAS mathematics library initialization, saving space allocation of Host end data, saving space allocation of Device end data, user data loading (mapping).

4.4. *Kernels execution threads allocation setting*

CUBLAS mathematics library API need not set up executing configuration, namely thread setting. Computing ability of GTX280 is 1.3 [4] specification and in each block of a SM of CPU a group of 32 threads execute currently, so each block is set up 256 threads, i.e. a multiple of 32. It can just efficiently hide latency of thread switch, and then dimension is (16,16).

Referring to the above related regulation, specific thread setting is shown in Table.

4.5. *Thread synchronization*

Using API "_syncthreads( )" provided by CUDA to activate lightweight barrier grid synchronization in the tail of each Kernel, instructions can be transmitted to all warp blocks which in one block and accomplish synchronization. But in MPI, OpenMP and etc. communication overhead of thread synchronization is very expensive, by comparison, internal synchronization overhead of CUDA block in four clock cycles is quite lower.

4.6. *Data collection in stream processor*

Algorithm of BP neural network learning has two processes for data collection in stream processor:

Firstly, when initial loading data, data from Host side is collected and mapped to Device, which is the interaction between Host side memory and global memory of CPU using cudaMemcpy(*dst*, *src*, *size*, cudaMemcpyHostToDevice).

Secondly, when modifying weigh d_W1Ex and d_W2Ex , from gradient descent method, d_W2Ex will be modified firstly. But when d_W1Ex is modified, value of d_W2Ex before modified is needed, so d_W2Ex should be saved temporarily. This is the interaction of themselves for global memory of GPU using cudaMemcpy(dst, src, size, cudaMemcpyDeviceToDevice).

4.7. *Data recycling to data area*

Data recycling includes two phases:

Firstly, generalized error d-y recycling. After the actual output of every turn of learning is obtained, generalized error is computed through kernel dotsub and recycled to Host side for variance summing.

Secondly, after completing learning, recycle actual output, weigh wih andw_{ho}.

The above two processes both use cudaMemcpy(dst, src, size, cudaMemcpyDeviceToHost).

5. Simulation comparison experiment

The experiment is carried out on NVIDIA GeForce GTX280, and onboard global memory is 1GB. The GPU is embarked on PC equipped with Intel Core2 E8400 3.0GHz.

For the experiment's comparability, on GPU when using CUBLAS and self-assembled Kernels to realize the above mentioned serial learning algorithm, on CPU we both use the neural network toolbox of Matlab R2009a and simulate the learning process, Number of specimens for training are 200, learning velocity is 0.06, the maximum learning time is 10000, objective error is 0.01.

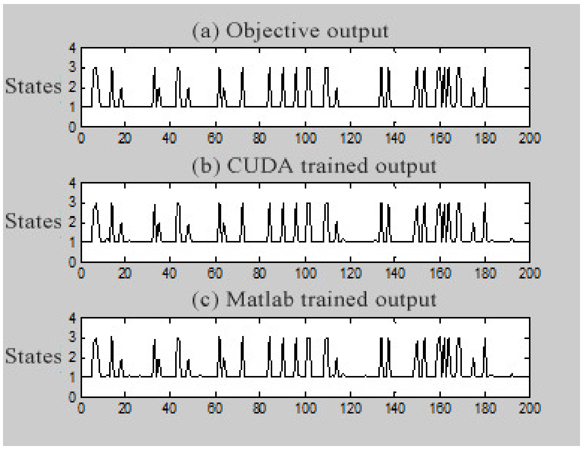

Figure 3: Output and objective output of 200 specimens after training

Fig.3 shows 200 specimens' objective output and actual output after trained in parallel and serial algorithm. The output of Matlab in Fig.3(b) is basically consistent with that in Fig.3(c), the jitter is not very obvious, and they are basically conformed to objective output of Fig.3(a), so the correctness of using CUDA technique to realize BP neural network is affirmative.

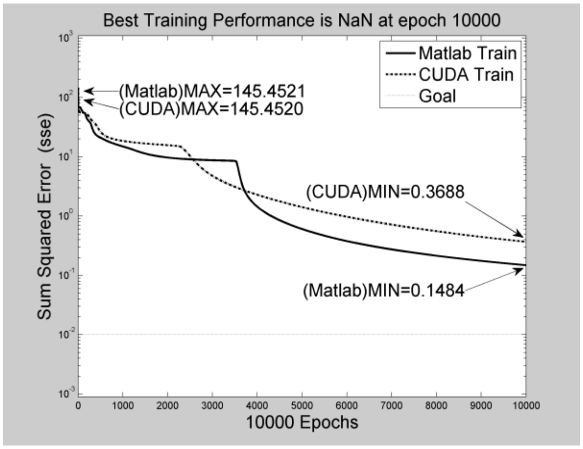

Figure 4: The fitting curve of error mean square sum of network training in learning velocity 0.06

Fig.4 shows if learn velocity 0.06, the maximum learning time 10000, objective error 0.01, the same input and output vector, the same threshold and weigh combination are established, the error mean square and fitting curve after network training by 2-10-3 network learning algorithm individually realized by CUDA technique and Matlab neural network tool box. The curves of CUDA error mean sum square is roughly identical with that of Matlab, and the velocity of convergence (gradient descent) is roughly the same. Two curves differ in the minimum and the error is about 0.22. According to the final output result from figure 3, the error of 0.22 has not greatly affected the actual output result. The precision of parallelization learning algorithm realized by CUDA technique is proportional to that of serial algorithm by Matlab neural network box analog simulation.

Table 1 shows the result of testing 5000 specimens which is randomly generated by using CUDA and Matlab to train network and get the weigh. The table indicates, using CUDA technique to realize BP neural network parallel algorithm, on the premise of ensuring training cycle shorter and training speed faster, its correct test rate is equivalent to that of serial algorithm.

Table 1: Examination results of test specimen

Platform	Number of specimens	Number of correct test	Correct rate%
Matlab	5000	4764	95.28
		4747	94.94
		4730	94.60
CUDA	5000	4742	94.84
		4761	95.22
		4768	95.36

After the above analyses, correctness of using CUDA technique to realize BP neural network parallel algorithm is certainly validated. Meanwhile its comparing of learning speed and serial algorithm on Matlab can be shown in Fig.5. For BP neural network learning, with learning velocity 0.06, maximum learning time 10000, objective error 0.01 and the number of implicit layer 10, CUDA consumes time 2.9s and Matlab 44s, then CUDA is 15 times faster than Matlab, so performance of CUBLAS library is remarkable.

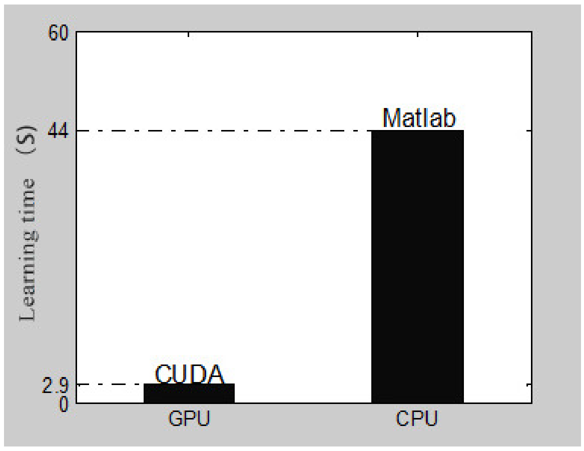

Figure 5: Comparing CPU with GPU in learning time

6. Conclusion

Using CUDA technology, the CUBLAS mathematical library and self-Kernels library in GPU to realize BP neural network parallel algorithm comparing to serial algorithmconvergencecomputing accuracy and correct rate are equivalent to serial algorithmand it achieves 15 times faster in 2-10-3 BP neural network parallel algorithm which proves an advantage well in contrast to CPU. Experiments show CUDA program model provides extendibility, intuitiveness and operability of GPU programming with united stream processor architecture, and its CUBLAS library has nice performance in actual application, the stream processor architecture satisfies related application of neural network.

Acknowledgment

This work is supported by Science and Technology Research Program for Colleges and Universities in Shandong Province (J17KA049).

References

[1] X. Chu, K. Zhao, M. Wang, *Massively parallel network coding on GPUs*, In Proceedings of IEEE Int'l Symp. On Performance Computing and Communications Conference, Austin, Texas, (2008), 144–151. 2.2

[2] K. Huang, Z. Xu, *Extensible Parallel Computing: Technique,Structure and Programming*, Beijing Industry Press, Beijing, (2000).

[3] D. Luebke, *CUDA: Scalable parallel programming for high-performance scientific computing*, In Proceedings of the 5th IEEE Int'l Symp. on Biomedical Imaging: From Nano to Macro. Paris, France, (2008), 836–838. 2.2

[4] NVIDIA Corporation, *NVIDIA CUDA Compute Unified Device Architecture Programming Guide 2.0[EB/OL]*, http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf. (2008). 2.1, 2.2, 4.4

[5] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, J. C. Phillips, *GPU computing*, Proceedings of the IEEE, **96** (2008), 879–899. 1

[6] M. Wen, *Research on key technology of stream architecture*, National University of Defense Technology, Changsha, **9** (2006), 1–15. 1

[7] X. Yang, X. Yan, T. Tang, *Research and development of Stream Processor technology*, Comput. Eng. Sci., **30** (2008), 114–117. 1

[8] J. Yi, Y. Hou, *Intelligent control technique*, Beijing Industry Press, Beijing, (2004). 3.1

[9] C. Yu, Y. Tang, *To improve the training time of BP neural networks*, In Proceedings of IEEE Int'l Symp. on Info-tech and Info-net. Beijing, China, **3** (2001), 473–479. 3.1

[10] Y. Zhang, X.-J. Yang, G.-B. Wang, I. Rogers, G. Li, Y. Deng, X.-B. Yan, *Scientific computing applications on a stream processor*, IEEE International Symposium on Performance Analysis of Systems and software (ISPASS), Austin, TX, USA, (2008), 105–114. 1