COMPUTING OF NEURAL NETWORK ON GRAPHICS CARD

S. Kajan, J. Slačka

Institute of Control and Industrial Informatics, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava, Slovak Republic
slavomir.kajan@stuba.sk, xslacka@stuba.sk

Abstract

This paper deals the potential of parallel computing on graphics cards. In many applications we encounter time-consuming mathematical or general computing operations, which handle large amounts of data. One of these applications is artificial neural network. One solution to speed up such calculations is to get them done on a graphics card. Graphics cards with CUDA (*Compute Unified Device Architecture*) interface allow programmers to use parallel performance of modern graphics cards. In computing of neural network were used libraries Jacked and GPUmat created for Matlab enable parallel computing on the graphics processing unit (GPU). Then we tested the acceleration of chosen mathematical operations on selected graphics cards. Computing of training algorithm for neural network contains mathematical matrix operations, which is possible very good parallel execute on GPU. As practical example we implemented hand written digit recognition using artificial neural networks.

1 Introduction

Currently we can see various software applications with high computation demands on mathematical or general operations that processes large amount of data. Acceleration of such time-consuming calculations is usually done by increasing computational power of a single PC, or by connecting multiple PC's into a computer Grid or Cluster. Such a solution requires a significant investment in the hardware equipment. Another way to speed up mathematical calculations is to compute them on graphics card. Accelerating of general calculations on graphics cards massively expanded in year 2007 (February), when *nVidia* released to the public the first official version of application interface for general calculations on graphics cards. This interface is called CUDA - *Compute Unified Device Architecture* and it enables the programmers to use massive parallel performance of modern graphics cards.

2 Principle of parallel computing on GPU

The graphics cards with older architecture used to calculate graphics scenes by vertex and pixel units. In modern games it happened sometimes that at the same moment there were used only the pixel units and the vertex units had nothing to do or vice versa. That's why *nVidia* in the new generation of the graphics cards (family 8000) introduced a new architecture of graphics processing unit. This new graphics card does not use separated vertex and pixel units, but for all types of calculations it uses the stream processors, which are more complex. These stream processors are all identical and they dynamically share the computing tasks as required by actual situation. With release of CUDA interface programmers have the opportunity to work with these processors and they can perform on them other than graphics calculations. Nowadays, the graphics cards witch CUDA support are used in scientific research centers and universities in various sectors as biology, medicine, physics, astrophysics, artificial intelligence, chemistry, etc... [5, 9].

If we compare a stream processor with a central processing unit (CPU) in serial calculation, the CPU will definitely win. The power of graphics cards is in the parallel data processing. In one graphic chip we can commonly find hundreds of stream processors. Their combined performance is many times better than the CPU performance. Graphics processing unit consisting of stream processors and graphics memory is in short known as GPU (see fig. 1)

To determine theoretical computational performance of CPU or GPU we use the number of flop‘s (floating point operations per second). For example a two-core CPU Core 2 Duo E7400 has the theoretical computational performance of 22,4 Gflop and the graphics card Nvidia Geforce GTS 250 with 128 stream processors has the theoretical computational performance of 700 Gflop. From these numbers it can be seen that this graphics card is theoretically about 28 times faster than Core 2 Duo in single precision calculations [7].

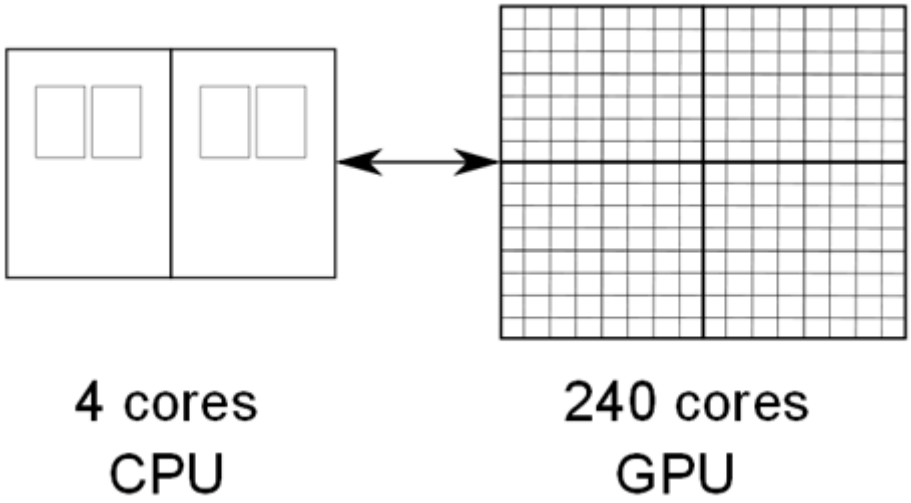

Figure 1: Comparison of CPU and GPU

Of course not every problem can be well parallelized, but there are applications in many fields such as edge detection in images, artificial intelligence and neural networks, where scientists by using powerful graphics cards reached speed up 50-100 times over CPU.

3 Principle of parallel computing on GPU

During the massive expansion of parallel computing on the GPU's, companies as AccelerEyes and GP-You Group have developed an extension for Matlab, which directly allows users to use in Matlab the performance of graphics cards. The company AccelerEyes offers a library for parallel computing on the GPU called Jacket [1, 3]. Conversely, the company GP-you Group has developed a similar library called GPUmat which is like Jacket and allows parallel computations on the GPU, but it is distributed for free (freeware) [2, 4]. Both libraries provide basic functions for handling calculations on graphics cards. These functions differ only in few commands. The benefit of the Jacket library is that there is existence of other extensions such as application functions for graphics, neural networks and other demonstration examples. The main advantage of the library GPUmat is providing free access to a very good basis for parallel computing on graphics card. Library GPUmat is designed to be easy to integrate with Matlab and other existing programs can be easily converted to run on the GPU. Installation of the GPUmat library is very easy. You just have to copy it to a location on the disk and run GPUstart. The program will check for compatibility with CUDA interface graphics card and add routes to the directory structure of MATLAB. Before installation of *GPUmat* you should have:

- Installed graphics card with driver software
- Installed development tools - CUDA SDK version 2.1 or 2.2 for your OS
- Installed software CUDA Toolkit version 2.1 or 2.2 for your OS

To use the computing power of the graphics cards we have to create a data variable which will be located in the memory of the graphics card. This variable must be type single, and we can create it by the command GPUsingle. Here is a revealing handicap of GPU calculations that GPU’s can work only with a single type of data, rather than the type of double, which reduces the accuracy of the calculation. However GPU's with core G200 or newer can process double data format, but this ability is not implemented in Jacked or GPUmat library. When we create such a variable, it can be applied to any function in Matlab while GPUmat will care that the calculation will run on the graphics card.

In the example of multiplication of two matrices A, B, we will explain how the calculation works. We can write matrices A and B into the graphics memory by using the command GPUsingle. Multiplication of matrices A and B is preformed by multiplying the row of matrix A and the column of matrix B. This computation is carried on each GPU stream processor and the result is written to the position of the matrix C in the graphics memory. (see fig. 2).

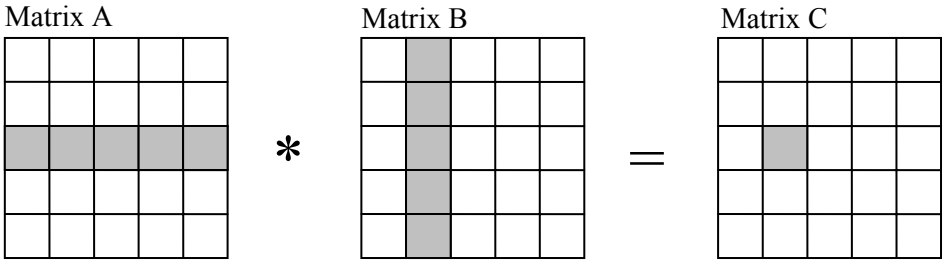

Figure 2: Image of matrix multiplication on GPU

In Matlab we can do this as follows:

```
>> A=GPUsingle(rand(100)); % create matrix A on GPU
>> B=GPUsingle(rand(100)); % create matrix B on GPU
>> C=A*B;        % multiply A and B on GPU
```

Similarly, the GPU can perform various mathematical operations. If we want the data from graphics memory to pass back into the CPU cache we can use the command CPUsingle.

Calculation of mathematical operations on the GPU has two main handicaps. The first one is that not all mathematical operations can be performed equally well in parallel, thereby speeding up the calculation is different and cannot be accurately determined. Another handicap is that the higher speed of calculation is dependent on the number of the processed data. The acceleration can usually be seen when the number of processed data is about 1e4 to 1e5. This threshold depends on the ratio of computational power of GPU and CPU and also on the number of stream processors on the GPU.

4 Examples of parallel computations on chosen graphics cards

The possibilities of accelerating the mathematical calculations on the GPU in Matlab using the library GPUmat are demonstrated on the matrix multiplication and matrix multiplication by elements examples, which are used in computing of neural networks. Data used for mathematical operations were randomly generated. Calculations were made on three graphics cards on three PC's with the following parameters:

1. PC 1

- CPU AMD Athlon 64 X2 Dual Core Processor 6000 + 3.02 GHz, 2GB RAM
- GPU nVidia 8600GT (256MB RAM, 32 stream processors)

2. PC 2

- CPU Intel Core2 Duo E6750 Processor+2 GHz, 4GB RAM
- GPU nVidia GTS250 (512MB RAM, 128 stream processors)

3. PC 3

- CPU Intel Core2 Duo E6750 Processor + 2.66 GHz, 4GB RAM
- GPU nVidia GTX275 (896MB RAM, 240 stream processors)

4.1 Examples of matrix multiplication

In testing, we monitored the dependence of calculation acceleration on the number of processed data, where speedup of calculation has been calculated as a fraction of CPU time - T_{CPU} and the GPU

time - T_{GPU}. Time calculation T_{CPU} and T_{GPU} were calculated as the statistical median of ten consecutive running calculations, thereby eliminating the random error.

$$Speedup = \frac{T_{CPU}}{T_{GPU}} \tag{1}$$

The results of the individual measurements on three different PCs are shown in figures 4 to 6.

In figure 4 you can see the results of the accelerating multiplication of two randomly generated matrixes A and B. In Matlab we calculated it as follows:

```
>> C=A*B;        % matrix multiplication
```

Graphics card GTX275 had the best results thanks to the highest number of the stream processors.

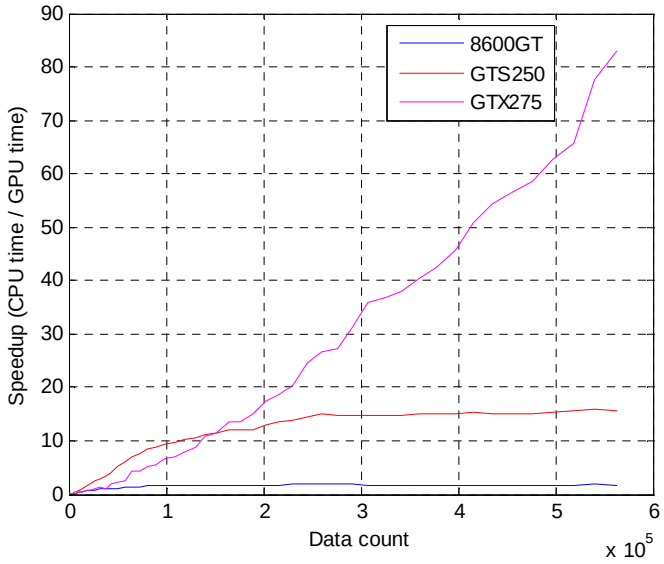

Figure 3: Computation speedup of matrix multiplication

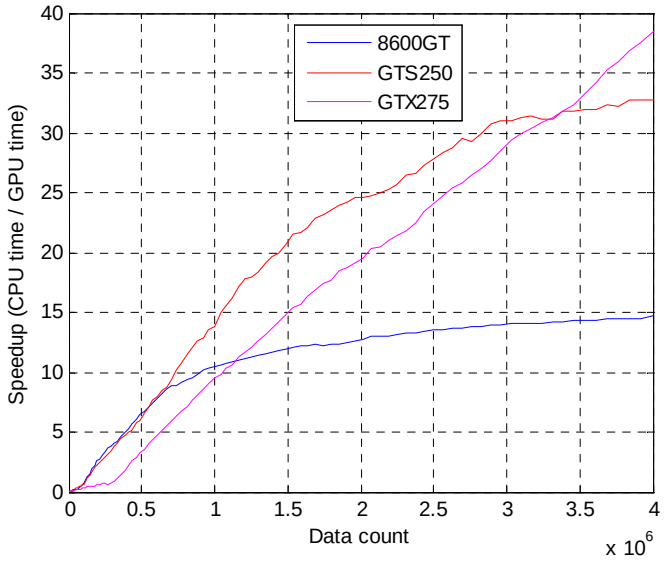

Figure 4: Computation speedup of matrix element multiplication

In figure 5 are shown the results of the speedup of the multiplication (element by element) between two randomly generated matrixes A and B. In Matlab we calculated it as follows:

```
>> C=A.*B;        % matrix multiplication by element
```

4.2 Example of written digits recognition by neural network

As a practical example of the computation accelerated on the graphics card, we used an application for hand written digit recognition by neural network. We chose a neural network because the learning of the neural network can be easily parallelized and it is appropriate to demonstrate the application of parallel computing on graphics card. Computing of training algorithm for neural network contains mathematical matrix operations, which is possible very good parallel execute on GPU. For digit recognition we used multi-layer perceptron network (MLP) with one hidden layer. The structure of MLP network is displayed in Figure 5. In the all layers of network, there was used logical sigmoid function (logsig) in the following form:

$$\varphi(a) = \frac{1}{1+e^{-\beta a}} \tag{2}$$

The network inputs are represented by parameters in range (0, 1), on the basis which is realized classification to classes [8]. The written digits shown in fig. 6 were scanned into a grid of 30x30 that means there are 900 inputs in neural network in range from 0 to 1, where 0 is white and 1 is black colour. The number of network outputs is defined according to recognition digits. Every network output has the value in the range (0, 1) and represents classification rate for defined digit. For MLP network training, there was used the modified algorithm of back-propagation errors with momentum parameter. To simulate a different number of processed data, we used a change in the number of neurons in hidden layer from range 500 to 12.000.

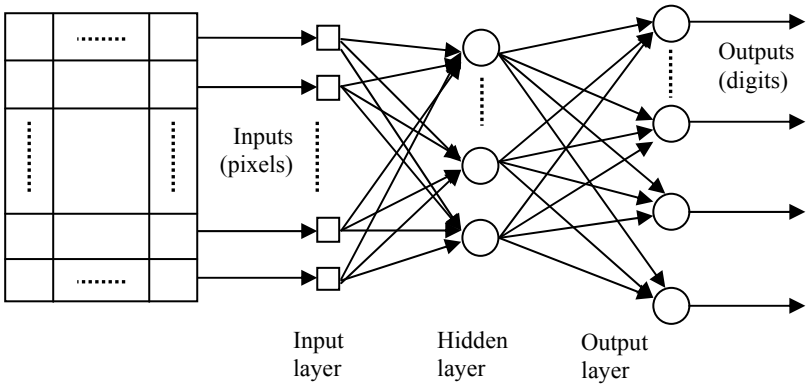

Figure 5: The structure of multilayer perceptron network

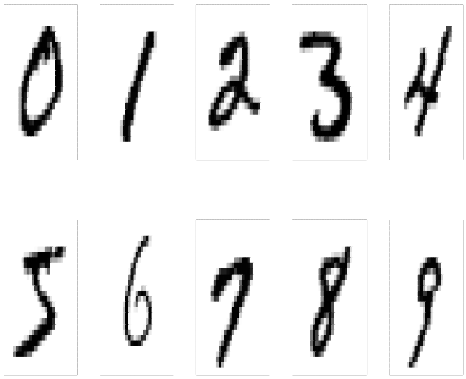

Figure 6: Written digits for recognition

Calculations of the outputs and adjusting the neural network weights can be rewritten in a matrix form [6]. This enables us to use graphics card to accelerate computations. The results of the calculation speed up against CPU are shown in fig. 7. Simulations were computed on PC 3 with graphics card Nvidia GTX275.

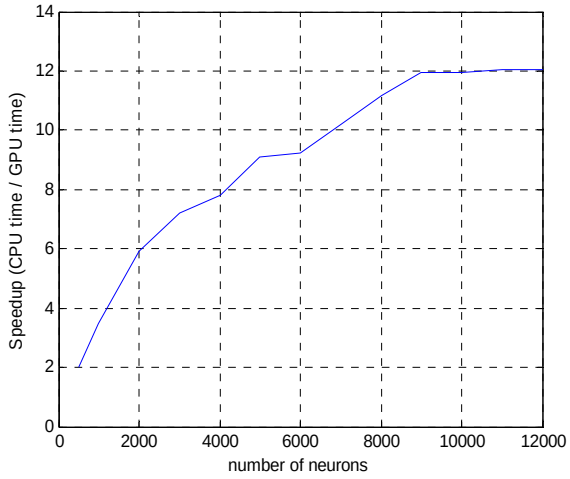

Figure 7: Computation speedup of neural network training

5 Conclusion

The existence of a CUDA application interface for general computations on graphics cards results in growing number of the applications using parallel computing. The library as a GPUmat or Jacket for MATLAB gives users a relatively simple way to implement intensive mathematical calculations on the graphics cards and use their advantage in parallel data processing. By measuring the performance of three graphics cards we verified, that by using the library GPUmat (and Jacket) in Matlab a speedup (up to ten) of the calculations of mathematical operations can be achieved. The advantage of parallel computing on GPU over CPU is considerably lower financial costs and vice versa, the disadvantage is the lack of support for double precision calculations by these libraries. The main disadvantage, as mentioned in the paper, is that such an approach to accelerate the calculation cannot be applied universally to every application, but only for special applications with large amounts of data such as pattern recognition, artificial intelligence, neural networks, graphics simulations, etc.

Acknowledgement

The work has been supported by the grants agency VEGA no. 1/0544/09 and no. 1/0690/09.

References

[1] The AccelerEyes: *Jacket User Guide*, jun 2009
[2] The GP-you group: GPUmat User Guide, April 2009
[3] http://www.accelereyes.com – internet page of company AccelerEyes
[4] http://gp-you.org – internet page of company GP-you
[5] http://www.nvidia.com – internet page of company nVidia
[6] A.Jadlovská. *Modelovanie a riadenie dynamických procesov s využitím neurónových sietí.* Edícia vedeckých spisov FEI TU Košice, ISBN 80-8894122-9, 2003
[7] S.Kajan, J.Slačka. *Matematické výpočty na grafickej karte v prostredí Matlab*, internet portal Posterus, http://www.posterus.sk/?m=200907, jul 2009
[8] M. Negnevitsky, *Artificial Intelligence*. Pearson Education Limited, 2005
[9] M.Visconti. *Heterogeneous GPU Computing In Computational Science*, http://www.osc.edu/supercomputing/training/customize/Mark_Visconti_slides.pdf
[10] The Mathworks. Neural Network Toolbox,User's Guide, documentation on CD

Ing. Slavomír Kajan, PhD, E-mail: slavomir.kajan@stuba.sk
Institute of Control and Industrial Informatics, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava

Bc. Juraj Slačka, E-mail: xslacka@stuba.sk
Institute of Control and Industrial Informatics, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava