The Sixteenth International Symposium on Artificial Life and Robotics 2011 (AROB 16th '11),
B-Con Plaza, Beppu,Oita, Japan, January 27-29, 2011

Parallelizing Fuzzy Rule Generation using GPGPU

T. Uenishi, T. Nakashima and N. Fujimoto
Osaka Prefecture University, 1-1 Gakuen-cho, Nakaku, Sakai, Osaka
(Tel : 81-72-254-9351; Fax : 81-72-254-9915)
(nakashi@cs.osakafu-u.ac.jp)

Abstract: This paper proposes a method to parallelize the process of generating fuzzy if-then rules for pattern classification problems in order to reduce computational time. The proposed method makes use of GPGPUs' parallel implementation with CUDA, a development environment. CUDA contains a library to perform matrix operations in parallel. In the proposed method, published source codes of matrix multiplication are modified so that the membership values of given training patterns with antecedent fuzzy sets are calculated. In a series of computational experiments, it is shown that the computational time is reduced for those problems that require high computational efforts.

Keywords: fuzzy if-then rule, parallel computation, GPGPU, pattern classification

I. INTRODUCTION

It is known that fuzzy systems based on fuzzy if-then rules perform well for pattern classification problems [1]. However, the computational cost of a fuzzy system is often huge when it is applied to high-dimensional problems with a large amount of training patterns. This is mainly due to explosive increase in the number of fuzzy if-then rules that are generated to construct a classification system. One solution to this problem is to reduce the number of rules. For example, the number of rules to be generated can be restricted, or a small number of fuzzy if-then rules can be selected by using genetic algorithms [2].

This paper proposes a method for speeding up the process of generating fuzzy if-then rules for pattern classification problems without reducing the number of rules. The proposed method is to implement the fuzzy-rule generation process on GPGPU (General Purpose computation on Graphics Processing Units) in order to reduce the computational time.

GPUs, which were originally developed for graphics processing, have a lot of multiprocessors and have potential for high-speed parallel computation. Implementation of GPUs is usually done in C programming language using CUDA (Compute Unified Device Architecture) [3]. To implement with CUDA, it should be considered that a GPU has its own memories which are only accessible from the GPU. GPU's memories are composed of several types of devices with different access speeds and capacities. Thus the efficiency of parallel computing with GPUs depends on the optimality of the memory access. However, it is difficult to design the memory access optimally without understanding the details of the hardware architecture of GPUs. In this paper, we adapt existing implementation to fuzzy classification system.

A library, called CUBLAS, is included in the CUDA package [4]. CUBLAS is an implementation of BLAS (Basic Linear Algebra Subprograms) computation for GPGPU. It allows us to develop parallel computing programs more easily without heavily modifying source codes. While all algorithms in CUBLAS are published as binary files, some source codes of SGEMM (Single precision General Matrix Multiply) algorithms have been published by the developer.

In the proposed method, calculation of the membership values is parallelized by viewing them as matrix calculation, using two matrices which represent antecedent fuzzy sets and training patterns. The published source codes of the matrix multiplication in SGEMM are modified so that the membership values of given training patterns with antecedent fuzzy sets are calculated in parallel. In a series of computational experiments, the computational time of the proposed method is compared with that of the traditional method that only uses a CPU. It is shown that the proposed method reduces the computational time for pattern classification problems that have high dimensionality and/or a large number of training patterns.

II. FUZZY RULE GENERATION

In this paper, we propose a method to parallelize fuzzy-rule generation that is formulated in the fuzzy system by Ishibuchi et al [1]. It should be noted that the method can be applied to any forms of fuzzy if-then rules because it parallelizes only membership calculation. An overview of the system in [1] is shown below.

In a pattern classification problem with n dimensionality and M classes, we suppose that m training patterns, $\boldsymbol{x}_p = \{x_{p1}, x_{p2}, \cdots, x_{pn}\},\ p = 1, 2, \cdots, m$, are given and each attribute of $\boldsymbol{x}_p$ is normalized to a unit interval $[0, 1]$. From training patterns we generate fuzzy if-then rules of the following type:

$$\begin{aligned} R_q : \text{If } x_1 \text{ is } F_{q1} \text{ and } \cdots \text{ and } x_n \text{ is } F_{qn} \\ \text{then Class } C_q \text{ with } CF_q, \\ q = 1, 2, \cdots, N, \end{aligned} \tag{1}$$

where R_q is the label of the q-th rule, $\boldsymbol{F}_q = (F_{q1}, \cdots, F_{qn})$ represents a set of antecedent fuzzy sets, C_q a the consequent class, CF_q is the confidence of the rule R_q, and N is the number of rules. We use triangular membership functions as antecedent fuzzy sets. Figure 1

shows triangular membership functions which divide the attribute axis into five fuzzy sets. Suppose that an attribute axis is divided into L fuzzy sets. The membership function of the k-th fuzzy set is defined as follows:

$$\mu_k(x) = \max\left\{1 - \frac{|x - x_k|}{v}, 0\right\}, k = 1, \cdots, L, \quad (2)$$

$$x_k = \frac{k-1}{L-1},\ k = 1, \cdots, L, \quad (3)$$

$$v = \frac{1}{L-1}. \quad (4)$$

Compatibility of a training pattern $\boldsymbol{x}_p$ with a fuzzy if-then rule R_q is denoted by $\mu_{\boldsymbol{F}_q}(\boldsymbol{x}_p)$ and is calculated as follows:

$$\mu_{\boldsymbol{F}_q}(\boldsymbol{x}_p) = \prod_{i=1}^{n} \mu_{F_{qi}}(x_{pi}),\ q = 1, 2, \cdots, N, \quad (5)$$

where $\mu_{F_{qi}}(x_{pi})$ is the compatibility of x_{pi} with the fuzzy set F_{qi} and x_{pi} is the the i-th attribute value of $\boldsymbol{x}_p$. $\mu_{F_{qi}}(x_{pi})$ is calculated by equation (2).

Equation (5) implies that the same procedure is iterated for calculating the compatibility of a training pattern with each fuzzy if-then rule: First calculating the compatibility for each attribute, and then multiplying them. Therefore, we can view this process as a function of two matrices. One matrix represents a set of fuzzy if-then rules. The size of this matrix is $N \times n$ and is composed of N row vectors whose lengths are n and elements are antecedent fuzzy sets $\boldsymbol{F}_q$. The other matrix represents a set of training patterns. This matrix is $n \times m$ and is composed of m column vectors whose lengths are n and each column is a training pattern $\boldsymbol{x_p}$. In the conventional matrix multiplication for two matrices, the (q, p) element of the product, r_{qp}, is represented as:

$$r_{qp} = \sum_{i=1}^{n} F_{qi} \times x_{pi}. \quad (6)$$

We adapt the above calculation to the calculation of membership value $\mu_{F_{qi}}(x_{pi})$ with the same access order as matrix multiplication. Thus the (q, p) element of the result, r'_{qp}, is represented as:

$$r'_{qp} = \prod_{i=1}^{n} F_{qi} \odot x_{pi}, \quad (7)$$

where $\odot$ denotes the membership calculation, i.e., equation (2). That is, the membership calculation (i.e., equation (7)) can be regarded as a matrix operation where product operation is replaced with a membership function and sum operation is replaced with a product operation.

The number of fuzzy rules to be generated is L^n. That is, the number of rules increases exponentially for the division number and the dimensionality.

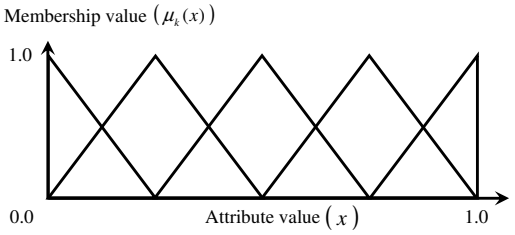

Fig. 1. Triangular fuzzy sets

III. GPGPU

GPUs have a lot of multiprocessors and thus have high potential for parallel computation. The performance of GPUs have been improved tremendously. NVIDIA have released CUDA, a development environment of GPGPU.

In CUDA, functions which are executed on a CPU are compiled with a c-compiler (we used GCC in this paper). While functions which are executed on a GPU are compiled with NVCC (NVIDIA CUDA Compiler). Data and control structures available for GPUs are the same as that of CPUs. However, a GPU has its own memories which are only accessible from it. Thus data transportation between a CPU and a GPU is required before the actual computation. The basic procedure of GPU computing is composed of four steps: A CPU transports data to a GPU, the CPU instructs the GPU to calculate, the GPU executes the calculation, and the GPU transports result to the CPU. Instructions for a GPU are composed of threads, blocks, and grids depending on the level of parallelization. A thread is an atomic execution of the instructions. A blocks is a set of threads, and a grid is a set of blocks. CPUs can only send instructions to grids. Upon receiving the instructions, threads in a block execute the calculations parallely depending on instructions, and instructions of blocks in a grid are also executed parallely. During the calculations, threads which belong to the same block can be synchronized and make use of shared memory whose access speed is higher than the global memory. The number of thread per block and the number of blocks per grid need to be determined. Thus the efficiency of parallel computing with GPUs depends on the memory access and the composition of threads, blocks, and grids. However, it is difficult to design the optimal memory access without understanding the details of the hardware architecture of GPUs. As an optimized library of BLAS(Basic Linear Algebra Subprograms) for CUDA, CUBLAS is published together with CUDA. In CUBLAS, SGEMM(Single precision General Matrix Multiply) and DGEMM(Double precision GEMM) are implemented by Volkov et al [5]. Some source codes of SGEMM are published by them. Since the memory access of CUBLAS is designed efficiently, user can implement their algorithms without any concern about memory access diverting the memory access of it. In this paper, we modify the source code of SGEMM so that the calculation of membership values

are parallelized in order to reduce the computational time. Representation and processing of floating point on GPUs follows IEEE754, and we suppose that real numbers on CPUs and GPUs are both single precision in this paper.

IV. IMPLEMENTATION

As mentioned in Section II, the formulation of membership values for fuzzy-rule generation is similar to that of matrix multiplication. We modify the SGEMM algorithm introduced in section III, to be the algorithm to generate fuzzy if-then rules. Volkov et al [5] published the SGEMM algorithm that calculates the following equation:

$$\boldsymbol{C}^{\mathrm{new}} = \alpha \times \boldsymbol{A} \times \boldsymbol{B}^{\mathrm{T}} + \beta \times \boldsymbol{C}^{\mathrm{old}}, \tag{8}$$

where $\boldsymbol{A}$ is a $x \times y$ matrix, $\boldsymbol{B}^{\mathrm{T}}$ is a $y \times z$ matrix, and $\boldsymbol{C}$ is a $x \times z$ matrix. α and β are scalar values. Equation (8) is calculated parallely by a GPU after initialization by a CPU. In this paper we specify that $\alpha = 1$ and $\beta = 0$ to consider only the matrix multiplication. By representing elements of the matrices as $\boldsymbol{A} = (a_{ij})$, $\boldsymbol{B}^{\mathrm{T}} = (b_{ij})$, and $\boldsymbol{C} = (c_{ij})$ and a temporal variable as t, the procedure to calculate an element of $\boldsymbol{C}^{\mathrm{new}}$, c_{ij}, can be shown as the pseudocode in Fig. 2(a). The parallel procedure to calculate compatibility is shown in Fig. 2(b), where a_{ik} is the label of the antecedent fuzzy set, and b_{kj} is the input value and $\mu(a_{ik}, b_{kj})$ is the membership function of the input value b_{kj} for the fuzzy set a_{ik}. Thus the order to access the elements of each matrices is the same as that of the original matrix multiplication. Therefore, the consistency of the parallel computation holds by replacing addition and multiplication of the elements in matrix multiplication to multiplication and membership calculation respectively. In addition, Volkov et al [5] employs 16×4 threads per block and $(x/64) \times (y/16)$ blocks per grid to make the memory access efficient. However, this limitation has no effect on the calculation of the equation. Now we can parallelize the membership calculation on GPUs by applying the above procedure to matrices which represent antecedent fuzzy sets and training pattern sets.

We suppose that $x_{F_{qi}}$ is an element of $\boldsymbol{A}$, where $x_{F_{qi}}$ is the mode of the fuzzy set F_{qi} computed by equation (3):

$$\boldsymbol{A} = \begin{bmatrix} x_{F_{11}} & \cdots & x_{F_{1n}} \\ \vdots & \ddots & \vdots \\ x_{F_{N1}} & \cdots & x_{F_{Nn}} \end{bmatrix}. \tag{9}$$

```
t = 0
for(k = 1 to y){
    t += a_ik × b_kj
}
c_ij = t
```

(a) Matrix multiplication

```
t = 1
for(k = 1 to y){
    t ×= μ(a_ik, b_kj)
}
c_ij = t
```

(b) Proposed method

Fig. 2. Pseudocodes of the matrix multiplication and the proposed method

And for a set of training patterns, we set a transposed matrix $\boldsymbol{B}^{\mathrm{T}}$ as follows:

$$\boldsymbol{B}^{\mathrm{T}} = \begin{bmatrix} x_{11} & \cdots & x_{m1} \\ \vdots & \ddots & \vdots \\ x_{1n} & \cdots & x_{mn} \end{bmatrix}. \tag{10}$$

By applying the calculation of compatibility modified from matrix multiplication to the above two matrices, a $N \times m$ matrix $\boldsymbol{C}$ is computed as:

$$\boldsymbol{C} = \begin{bmatrix} \mu_{\boldsymbol{F}_1}(\boldsymbol{x}_1) & \cdots & \mu_{\boldsymbol{F}_1}(\boldsymbol{x}_m) \\ \vdots & \ddots & \vdots \\ \mu_{\boldsymbol{F}_N}(\boldsymbol{x}_1) & \cdots & \mu_{\boldsymbol{F}_N}(\boldsymbol{x}_m) \end{bmatrix}, \tag{11}$$

where a row vector of $\boldsymbol{C}$ corresponds to the compatibility of rules for each patterns, i.e., equation (5). The procedure to parallelize fuzzy-rule generation with a GPU takes the following steps. First, by a CPU, $\boldsymbol{A}$ and $\boldsymbol{B}^{\mathrm{T}}$ are made, and transported to a GPU. Second, the GPU calculates $\boldsymbol{C}$ using the matrix operation for $\boldsymbol{A}$, $\boldsymbol{B}^{\mathrm{T}}$. Finally, $\boldsymbol{C}$ is transported to the CPU, and then it determines the consequents.

V. COMPUTATIONAL EXPERIMENTS

To verify the effect of parallelization with GPUs, the computational time to generate fuzzy rules with a GPU is compared to that of a CPU. Table 1 shows the environment of the experiments. Although GeForce GTX 295 has a dual-chip structure, we use only one chip. In the experiments, the computational time for solving a two-class problem is compared. 100 classification problems with different number of training patterns and dimensionalities were used to evaluate the computational time. The number of fuzzy sets for each axis is fixed to two. The results were averaged to compare the efficiency of the parallelization. The results of the experiments are shown in Figs. 3-6. Figure 3 shows how dimensionality of the problem has an effect on the computational time when the number of training patterns is 64. Figure 4 shows how dimensionality of the problem has an effect on the computational time when the number of training patterns is 816. In Fig. 3, the computational time with a CPU is shorter than that with a GPU when the dimensionality is small. As the dimensionality increased, the computational time with a CPU increased drastically while that with a GPU keeps short. In Fig. 4, the computational time with a GPU is shorter than that with a CPU constantly. Figures 5 and 6 show that the number of patterns has an effect on the computational time when the dimensionality of the problem is 12 or 18 respectively. In Fig. 5, the computational time with a CPU is shorter than that with a GPU when the number of patterns is small. However, when the number of training patterns is large, the computational time with a GPU is shorter than that with a CPU. In Fig. 6, the computational time with a GPU is shorter than that

with a CPU constantly. Thus the parallel computation for generating fuzzy rules with a GPU has the effect of reducing the computational time except in the case of low-dimensionality problems with small amount of training patterns.

Table 1. Environment of the experiments

CPU	Intel Core i7 Extreme 945
Clock Frequency	3.20 GHz
Memory Size	5.8 GB
Memory Clock	667 MHz
GPU	NVIDIA GeForce GTX 295
Processor Core	240
Processor Clock	1242 MHz
Memory Size	896 MB
Memory Clock	999 MHz
OS	Linux x86_64
Development Environment	CUDA(NVCC)2.2, GCC4.3

VI. CONCLUSIONS

In this paper, we proposed a method to parallelize fuzzy-rule generation with a GPU using matrix multiplication that is optimized for CUDA. Computational experiments showed that the method reduced the computational time when the dimensionality of the problem and/or the number of training patterns were large. For future works, we will try to parallelize fuzzy inference with GPU, or resolve lack of memory on GPU when the method is applied to problems with further dimensionality and/or the number of training patterns.

REFERENCES

[1] Ishibuchi H, Nakashima T and Nii M (2003), Classification and Modeling with Linguistic Information Granules. Springer

[2] Ishibuchi H, Nozaki K, Yamamoto N, and Tanaka H (1993), Selection of Fuzzy If-Then Rules by a Genetic Method(in Japanese). The Transactions of the Institute of Electronics, Information and Communication Engineers Vol.J76-A No.10:1465-1473

[3] NVIDIA CUDA, http://www.nvidia.com/object/cuda_home.html

[4] NVIDIA CUBLAS Library, http://developer.download.nvidia.com/compute/cuda/2_1/toolkit/docs/CUBLAS_Library_2.1.pdf

[5] Volkov V and Demmel JW (2008), LU, QR and Cholesky factorizations using vector capabilities of GPUs. Technical Report No.UCB/EECS-2008-49

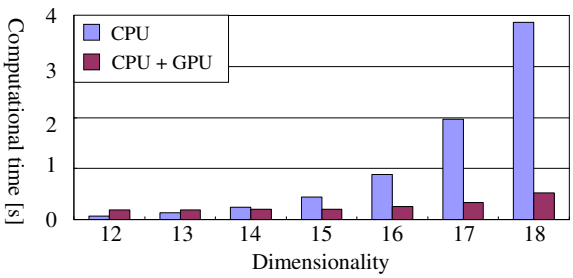

Fig. 3. Computational time for dimensions (64 training patterns)

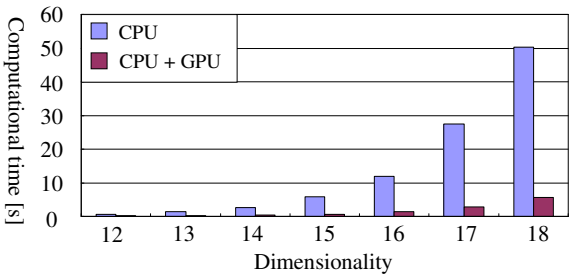

Fig. 4. Computational time for dimensions (816 training patterns)

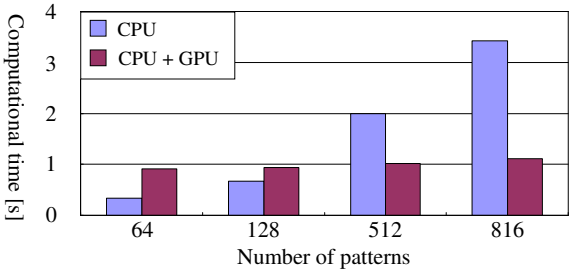

Fig. 5. Computational time for number of patterns (12 dimensionality)

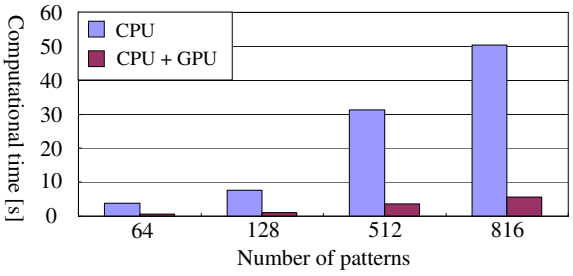

Fig. 6. Computational time for number of patterns (18 dimensionality)