GPU-Accelerated Learning of Neuro-Fuzzy System Based on Fuzzy Truth Value

Sergey Vladimirovich Kulabukhov(✉) and Vasily Grigorievich Sinuk

Belgorod State Technological University Named After V.G. Shukhov, Belgorod, Russian Federation
qlba@ya.ru, vgsinuk@mail.ru

Abstract. The article is devoted to the problem of the computational complexity of fuzzy inference and neuro-fuzzy system learning in the case of fuzzy inputs. We resort to parallel computations to reduce computation time. In the article, we suggest an algorithm using GPU to efficiently perform fuzzy inference based on fuzzy truth values and the extension of this algorithm to neuro-fuzzy system learning by evolution strategy. We demonstrate the importance of the algorithm and include a benchmark to compare the computation time on CPU against GPU.

Keywords: Fuzzy inference systems · Neuro-fuzzy systems · Fuzzy truth value · Evolution strategies · GPGPU · Parallel computations

1 Introduction

Fuzzy inference systems are gaining popularity. They are used in many fields of practical interest. As a matter of fact, they are more consistent with the nature of human thinking than systems of traditional formal logic, since they allow for building models that reflect various aspects of uncertainty in a more adequate manner [6]. Such models are defined via fuzzy rule bases. Fuzzy inference systems have applications in such fields as control of technical systems, speech and image recognition, and diverse expert systems.

Fuzzy inference systems do not address the formation of a rule base. Being a formal representation of knowledge, fuzzy rule bases can be constructed manually. Nevertheless, some applications also provide training sets, i.e. data consisting of pairs "input values–desired output values" that can be used to adjust the parameters of the inference system. Such situations are the subject of machine learning and are common in applications of artificial neural networks.

The interpretation of neural network parameters is generally difficult. This fact prevents the explicit use of knowledge of domain experts in a network and the extraction of knowledge from a trained network. The combination of basic

This work was supported by the RFBR (grant No. 19-07-00133).

methods of fuzzy inference systems and artificial neural networks led to the creation of neuro-fuzzy systems. Various options for combining these methods have been presented in the literature. For example, parameters of membership functions, triangular norms, or even the whole rule base can undergo learning.

The parameters of fuzzy inference systems have a clear meaning, and their values after learning may reveal previously unknown knowledge about a subject area. In addition, if a fuzzy inference system can handle fuzzy input values, then they can also be used for neuro-fuzzy system learning, provided that the training set contains uncertainty. In many applications, input data contain either nonnumerical (linguistic) assessments [2,7] or input signals that are received with noise [8,9].

In this paper, we consider logical-type neuro-fuzzy systems based on fuzzy truth values [5,10]. Such systems allow for inference in case of multiple fuzzy inputs of polynomial computational complexity by using any triangular norm [11]. On the other hand, this method of fuzzy inference leads to discretization of membership functions since they undergo complex transformations which are difficult to implement in analytical form.

An evolutionary algorithm was used in [8,9] for neuro-fuzzy system learning. This algorithm assumes that the computation of objective function values is simultaneous for all elements of the offspring population created at each generation. Neuro-fuzzy system learning can be very time-consuming even in simple tasks. The parallel implementation of both inference and learning processes in such systems by using GPU is the subject of the present article. The implementation is based on NVIDIA CUDA technology.

2 Neuro-Fuzzy System Based on Fuzzy Truth Values

The problem that is to be solved by using a fuzzy inference system is formulated as follows. Consider a system with n inputs $\boldsymbol{x} = [x_1, \ldots, x_n]$ and a single output y. The relationship between inputs and the output is defined using N fuzzy rules expressed as

$$R_k\colon \text{If } x_1 \text{ is } A_{1k} \text{ and } \ldots \text{ and } x_n \text{ is } A_{nk}, \text{ then } y \text{ is } B_k, \quad k = \overline{1, N}, \tag{1}$$

where $\boldsymbol{x} \in \boldsymbol{X} = X_1 \times X_2 \times \cdots \times X_n$, $y \in Y$, and $\boldsymbol{A_k} = A_{1k} \times A_{2k} \times \cdots \times A_{nk} \subseteq \boldsymbol{X}$, $B_k \subseteq Y$ are fuzzy sets.

According to the classification proposed in [12], the specific feature of logical-type systems is that the rules expressed in (1) are formalized via fuzzy implication as fuzzy $(n+1)$-ary relations $R_k \subseteq X_1 \times \cdots \times X_n \times Y$, namely

$$R_k = A_{1k} \times \cdots \times A_{nk} \times Y \to X_1 \times \cdots \times X_n \times B_k, \;\; k = \overline{1, N},$$

where "$\to$" denotes a fuzzy implication expressing a causal relationship between the antecedent "x_1 is A_{1k} and ... and x_n is A_{nk}" and the consequent "y is B_k". The task is to determine the inference result $B'_k \subseteq Y$ for a system given in the form expressed in (1), provided that the inputs are given as $\boldsymbol{A'} = A'_1 \times \cdots \times A'_n \subseteq \boldsymbol{X}$ or "x_1 is A'_1 and ... and x_n is A'_n".

The specific feature of the considered approach to fuzzy inference is that the inference is made within a single truth space for all premises, which is achieved by transforming the relationships between premise and fact into a so-called fuzzy truth value. By using the truth modification rule (see [1]), we can write

$$\mu_{A'}(x) = \tau_{A|A'}(\mu_A(x)),$$

where $\tau_{A|A'}(\cdot)$ is the fuzzy truth value of the fuzzy set A relative to A', which represents the compatibility $CP(A, A')$ of the term A with respect to A' [3,13]:

$$\tau_{A|A'}(t) = \mu_{CP(A,A')}(t) = \sup_{\substack{\mu_A(x)=t \\ x\in X}} \{\mu_{A'}(x)\}, \quad t \in [0,1]. \tag{2}$$

Denote $t = \mu_A(x)$. Then

$$\mu_{A'}(x) = \tau_{A|A'}(\mu_A(x)) = \tau_{A|A'}(t). \tag{3}$$

Thus, the generalized fuzzy modus ponens rule for single-input systems can be written as

$$\mu_{B'_k}(y) = \sup_{t\in[0,1]} \{\tau_{A|A'}(t) \,\mathrm{T}\, I(t, \mu_{B_k}(y))\}, \quad k = \overline{1,N},$$

where T is a t-norm and I is a fuzzy implication.

In systems with n inputs ($n > 1$), the convolution of fuzzy truth values $\tau_{A_i|A'}$ is performed for all inputs $i = \overline{1,n}$. For rules of the form (1), the fuzzy truth value of the antecedent $\boldsymbol{A_k}$ with respect to the inputs $\boldsymbol{A'}$ is defined as

$$\tau_{\boldsymbol{A_k}|\boldsymbol{A'}}(t) = \mathop{\mathbf{T}}_{i=\overline{1,n}} \tau_{A_{ki}|A'_i}(t), \quad t \in [0,1], \tag{4}$$

where $\mathbf{T}$ is an n-ary t-norm extended by the extension principle (see [9]). With this in mind, the inference of the output value B'_k based on the fuzzy truth value, for systems with n inputs, can be written in the form

$$\mu_{B'_k}(y) = \sup_{t\in[0,1]} \{\tau_{\boldsymbol{A_k}|\boldsymbol{A'}}(t) \,\mathrm{T}\, I(t, \mu_{B_k}(y))\}, \quad k = \overline{1,N}. \tag{5}$$

The fuzzy set B' (the output of the system as a whole) is obtained by accumulation, and in a logical approach, it is defined as an intersection operation [9]:

$$B' = \bigcap_{j=\overline{1,N}} B'_j. \tag{6}$$

Accordingly, the membership function B' is defined by means of the t-norm:

$$\mu_{B'}(y) = \mathop{\mathrm{T}}_{j=\overline{1,N}} \mu_{B'_j}(y). \tag{7}$$

The center-of-gravity defuzzification method is used in the neuro-fuzzy system to define the crisp output $\overline{y}$ of the system:

$$\overline{y} = \frac{\int_Y y \cdot \mu_{B'}(y)\, dy}{\int_Y \mu_{B'}(y)\, dy}. \tag{8}$$

In addition, a transformation is introduced to regulate the effect of each rule j on the accumulation result in accordance with its weight w_j. This transformation is pointwise. Now (7) can be written in the form

$$\mu_{B'}(y) = \underset{j=\overline{1,N}}{\mathrm{T}} f\left(\mu_{B'_j}(y), w_j\right), \tag{9}$$

where $f(a, w)$ is an arbitrary function that associates a membership-function value a to a new value according to the rule weight w. The function $f(a, w)$ must be nondecreasing on the first argument and, for logical-type systems, nonincreasing on the second argument.

Within this study, neuro-fuzzy system learning is performed by means of an evolution strategy (μ, λ) [9]. As it was already noted, this algorithm assumes that the computation of objective function values $R(\mathbf{p})$ is simultaneous for all forms of the offspring population $\boldsymbol{O}$ at each generation. An objective function is a function of neuro-fuzzy system parameters that reflects how far the results obtained deviate from the training set. The lower $R(\mathbf{p}_l)$ is, the more $\mathbf{p}_l$ corresponds to the training set. Let us denote the training set as

$$T = \{T_r = \langle A_1'^{(r)}, \ldots, A_n'^{(r)}, \overline{y}^{(r)} \rangle\}_{r=\overline{1,M}},$$

where $A_i'^{(r)}$ is the value of the i-th input for the r-th element of the training set, $\overline{y}^{(r)}$ is the desired output value for these input values, and M is the training set size.

The objective function for the neuro-fuzzy system is defined as

$$R(\mathbf{p}) = \frac{1}{|Y|}\sqrt{\frac{1}{M}\sum_{r=1}^{M}\left(F_{\mathbf{p}}\left(A_1'^{(r)}, \ldots, A_n'^{(r)}\right) - \overline{y}^{(r)}\right)^2}, \tag{10}$$

where $|Y|$ stands for the width of the domain of definition of the output variable, while

$$F_{\mathbf{p}}\left(A_1'^{(r)}, \ldots, A_n'^{(r)}\right)$$

is the inference result $\overline{y}$ of the neuro-fuzzy system with parameter values $\mathbf{p}$ and input values $A_1'^{(r)}, \ldots, A_n'^{(r)}$.

3 Parallel Implementation of the Learning Process

In this section, we consider the implementation of the learning process of the neuro-fuzzy system, in which all computations associated with fuzzy inference are performed on graphics processing units (GPUs).

3.1 Learning Process Overview

From a computational point of view, the evaluation of the objective function $R(\mathbf{p})$ for each element of the offspring population $\boldsymbol{O}$ is the most complex phase of neuro-fuzzy system learning. To compute $R(\mathbf{p})$ for a single $\mathbf{p}$ (a single vector of parameter values), we must compute the value of the expression under the summation symbol in (10) for each element T_r of the training set T, each time computing the inference result for the given input values. This means that we have to perform a fuzzy inference for each combination $\langle \mathbf{p}, T_r \rangle \in \boldsymbol{O} \times T$, i.e. $|\boldsymbol{O}| \cdot M$ times, and each of these inferences is independent of the others. Thus, all pairs $\langle \mathbf{p}, T_r \rangle \in \boldsymbol{O} \times T$ can be processed in parallel. In the implementation of the algorithm using CUDA technology, every pair is processed in an individual block.

Each block is allocated its own memory, a part of which is allocated for storing intermediate results; output results are placed in another part of the memory and a third part contains a sequence of commands defining fuzzy inference operations. Before launching the kernel, command sequences that are executed by the fuzzy system to compute output values are built for each block and written to the corresponding memory location. The values of inputs and parameters are embedded explicitly in these sequences. Since commands are specified separately for each block, the developed algorithm does not impose restrictions on the set of adjusted parameters of the neuro-fuzzy system. In general, it allows parallel inference to be done for several completely different fuzzy systems. This fact expands the possibilities of neuro-fuzzy system learning.

As previously mentioned, this fuzzy inference method leads to discretization of membership functions into arrays of samples. These samples are distributed over the threads and then processed. The implementation of individual fuzzy inference operations will be described below in this section.

3.2 Data Distribution in GPU Memory

Each block is allocated the same amount of memory, which consists of three parts (Fig. 1). Let N_b be the number of blocks. The first part of the allocated memory is used for storing the results of the execution of intermediate commands; these results are arrays of samples of membership functions. The part consists of N_c arrays of N_d elements each, where N_d is the number of samples and N_c is the maximum number of commands that result in a membership function. The array elements are single-precision real numbers. The second part of the allocated memory stores a real number which is the final result of the computation, i.e. the crisp output $\overline{y}$ of the system. The third part stores the sequence of fuzzy inference commands. The size of this part is S_c, which is an estimated maximum size for command sequences (in bytes) as the size of sequences for different blocks in the general case can vary, for example, due to rule base changes during training. If the command sequence for a block is shorter than S_c, then this part is aligned with unused memory to size S_c.

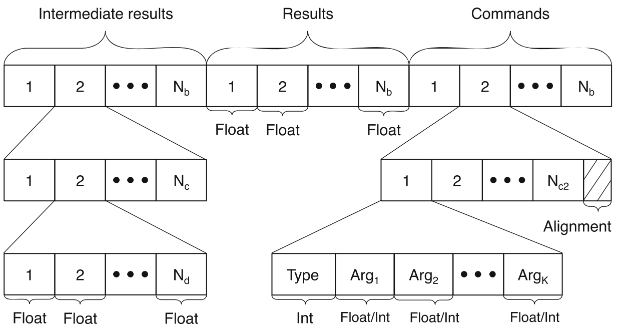

Fig. 1. Data distribution in GPU memory

Intermediate results for every block are stored first in GPU memory. Final inference results for each block are placed next, followed by command sequences. Such a distribution of parts in memory has an advantage over their grouping in blocks, namely when transferring the results from the device memory into the host memory, a continuous memory segment containing $N_b * 4$ bytes is copied in the first case, whereas, in the second, N_b four-byte segments are copied. The same happens when transferring command sequences into the device memory. Storage of all intermediate results in memory is not required; however, it provides detailed information when designing and debugging neuro-fuzzy systems.

Each command is represented as a sequence of numbers. The first number denotes the operation type; subsequent numbers are its arguments and parameters. The operation type determines the amount of numbers. If arguments include membership functions, then the command size is also determined by their type. Each argument or parameter is either a 32-bit integer or a single-precision real number.

3.3 The Main Kernel Function

The main kernel function accepts four parameters: the start address of the allocated memory a_0; the amounts of memory $N_c * N_d * 4$ and S_c, which are needed for storing, respectively, the intermediate results and the commands; and finally, the number of samples N_d. Kernel launch parameters are also specified: the grid size is set to $|\boldsymbol{O}| \cdot M$ (single dimension); the block size is set to the maximum possible number not exceeding N_d; the amount of shared memory is $N_d * 4$ bytes. When the main kernel function is started, each thread calculates the addresses of each memory part for the block $b =$ `blockIdx.x` to which the thread belongs. The first part has the address $a_1 = a_0 + b * N_c * N_d * 4$, the second $a_2 = a_0 + N_b * N_c * N_d * 4 + b * 4$, and the third $a_3 = a_0 + N_b * N_c * N_d * 4 + N_b * 4 + b * S_c$. This function also declares the variable `dp` which contains both the number of saved results of intermediate commands (initially equal to zero) and a pointer to the beginning of the next command `ip` (initially set to a_3). Then a loop begins, in whose body the current command is read and executed, and the pointer `ip` is increased by the size of the command. Implementations of individual operations are allocated to separate subroutines; the values of arguments and parameters that are passed to them are determined in the main subroutine. If the result of the current operation is a membership function, then its array of samples is placed at $a_3 + \texttt{dp} * N_d$ and `dp` is incremented. The loop ends when `ip` points to a dummy command that marks the end of the command sequence. We describe each operation below.

3.4 Computing Fuzzy Truth Values

This operation is analytically defined by (2). Its arguments are membership functions of the term $\mu_A(x)$ and the fact $\mu_{A'}(x)$; the result is the fuzzy truth value $\tau_{A|A'}(t)$ of the term with respect to the fact. In the discrete case, we calculate sample values of the function $\tau_{A|A'}(t)$. Since this function is defined in the numerical range $[0;1]$, we split it into N_d samples and define the value of each sample as follows:

$$\tau_{A|A'}(t_i) = \sup_{\substack{\mu_A(x) \in [t_i; t_{i+1}] \\ x \in X}} \{\mu_{A'}(x)\}, \quad t_i = \frac{i}{N_d}, \quad i = \overline{0, N_d - 1}, \tag{11}$$

whence, taking into account (3), it follows that

$$\tau_{A|A'}(t_i) = \sup_{t \in [t_i; t_{i+1}]} \{\tau_{A|A'}(t)\}. \tag{12}$$

The procedure for computing $\tau_{A|A'}(t_i)$ for each t_i is divided into three subroutines. Flowcharts of two of them at the level of individual threads are shown in Fig. 2. The main subroutine of the operation is $\texttt{FTV}(\texttt{dst}, \mu_A(x), \mu_{A'}(x), N_d)$, which fills an array of N_d elements at `dst` with values according to (11). Membership functions $\mu_A(x)$ and $\mu_{A'}(x)$ are expressed as numerical sequences in the same way as the commands: the first number determines the type of the membership function, the other numbers are its parameters. The destination address `dst` points to an array in the memory space allocated for the results of the execution of intermediate commands for this block.

The computation of individual samples of $\tau_{A|A'}(t_i)$ is distributed among all threads of the block. `FTV` starts with getting the thread index `threadIdx.x` and the number of threads `blockDim.x` in the block. They will be henceforth referred to as variables `x` and `h`, respectively. Then a loop is executed for $i = \overline{0, N_d - 1}$, each thread performs every `h`-th iteration starting with the `x`-th. Since the value of the i-th sample equals the upper boundary of the fuzzy truth value in the range $[t_i; t_{i+1}]$ (see (12)), the ends of this range, denoted by $t_{\min}$ and $t_{\max}$, are calculated in the loop body. Then the function $\texttt{FTV1}(\mu_A(x), \mu_{A'}(x), t_{\min}, t_{\max})$ is invoked, and it returns the value of (11) for the given range $[t_{\min}; t_{\max}]$, which is then assigned to the i-th element of the array at `dst`.

Depending on both the type of the membership function $\mu_A(x)$ and the values of its parameters, `FTV1` determines all ranges such that $\mu_A(x) \in [t_{\min}; t_{\max}]$.

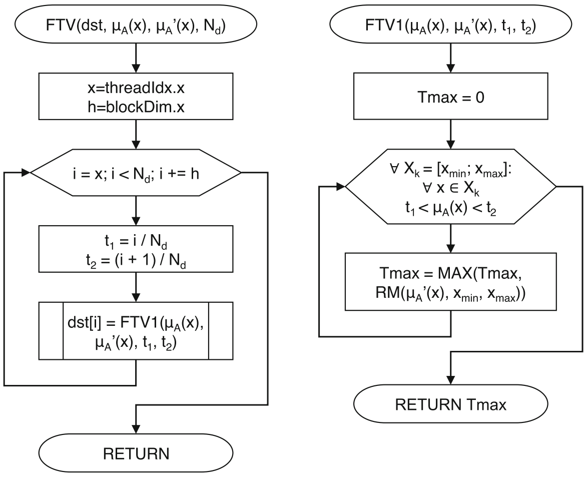

Fig. 2. Flowcharts of fuzzy truth value computation

`FTV1` features algorithms to determine these ranges for all supported types of membership functions. For example, the ranges for a Gaussian membership function with center in m and standard deviation σ are

$$\left[m-\sigma\sqrt{-\ln t_{\min}};m-\sigma\sqrt{-\ln t_{\max}}\right],\left[m+\sigma\sqrt{-\ln t_{\max}};m+\sigma\sqrt{-\ln t_{\min}}\right].$$

For each of these ranges, `FTV1` invokes the function `RM`($\mu_{A'}(x), x_{\min}, x_{\max}$), which returns the maximum membership degree of the fact within this range. `FTV1` returns the maximum of the values returned by `RM`.

The flowchart of `RM` is not shown in the figure as it contains only the formulas that express the maximum value within a given range $[x_{\min};x_{\max}]$ for all supported types of membership functions. In the aforementioned case of a Gaussian function, this value is given as

$$\sup_{x\in[x_{\min};x_{\max}]}\{\mu_{A'}(x)\}=\begin{cases}1, & \text{if } x_{\min}\le m\le x_{\max},\\ \exp\left((x_{\max}-m)^2/\sigma^2\right), & \text{if } x_{\max}<m,\\ \exp\left((x_{\min}-m)^2/\sigma^2\right), & \text{if } x_{\min}>m.\end{cases}$$

3.5 Convolution of Fuzzy Truth Values

If a rule contains more than one subcondition, then the fuzzy truth value of the entire antecedent $\boldsymbol{A_k}$ with respect to the inputs $\boldsymbol{A'}$ is computed according to (4). This formula contains an n-ary t-norm extended by the extension principle. For $n=2$, it is defined as

$$\underset{i=\overline{1,2}}{\mathbf{T}}\tau_{A_{ki}|A_i'}(t)=\sup_{\substack{t_1\,\mathrm{T}\,t_2=t\\ t_1,t_2\in[0;1]}}\{\tau_{A_{k1}|A_1'}(t_1)\,\mathrm{T}\,\tau_{A_{k2}|A_2'}(t_2)\},\quad t\in[0,1]. \tag{13}$$

If $n > 2$, then $\mathbf{T}$ is applied as an associative binary operator to the result of the convolution of the previous $(n-1)$ arguments and the n-th argument.

The algorithm for computing the result of this operation is depicted in Fig. 3. Its computational complexity is $O\left(N_d^2\right)$. The operation is implemented for $n = 2$; the convolution for larger values of n is defined by using multiple convolution commands. The arguments `A`, `B`, and the result `dst` are fuzzy truth values in discrete form (arrays of samples). Therefore, t_1 and t_2 take on values in the discrete set $\{i/N_d\}_{i=\overline{0,N_d-1}}$.

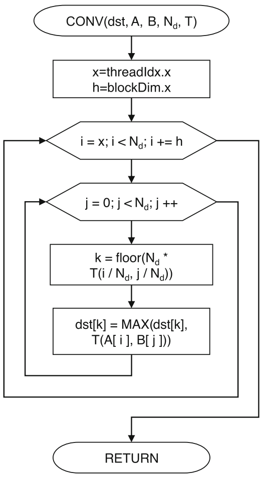

Fig. 3. Flowchart for the convolution of fuzzy truth values

The enumeration of all values of t_1 and t_2 is implemented by a double loop for the variables `i` and `j`, respectively. The iterations of the external loop for the variable `i` are distributed among all threads of the block in the same manner as in the case of `FTV` (Fig. 2). The internal loop for `j` is entirely executed by a single thread. The values of `i` and `j` correspond to $t_1 = \mathtt{i}/N_d$ and $t_2 = \mathtt{j}/N_d$, respectively. Then $t = t_1 \mathrm{~T~} t_2$ is computed. The index of the result sample corresponding to t is $\mathtt{k} = \lfloor t * N_d \rfloor$. Let us denote by $\hat{\tau}_t$ the argument of the supremum in (13), i.e. $\hat{\tau}_t = \tau_{A_{k1}|A_1'}(t_1) \mathrm{~T~} \tau_{A_{k2}|A_2'}(t_2)$, which is calculated as `T(A[i], B[j])`, where `T` is the implementation of the t-norm. If $\hat{\tau}_t$ is greater than the current value of `dst[k]`, then `dst[k]` is assigned the value of $\hat{\tau}_t$.

The index `k` is calculated using the t-norm. In the implemented distribution of iterations for `i` and `j`, threads may process elements of the destination array with the same index. If multiple threads read the value of `dst[k]` and then conditionally update it, then data races may occur. However, distributing the iterations in such a way that no pair of threads obtains similar values of `k` depends on the t-norm and is computationally inefficient since threads will get different numbers of payloads. Data races can be eliminated by using the atomic maximum operation provided by the CUDA framework, which can be invoked by calling the function `atomicMax`. This function accepts `address` and the value `val` as arguments. If the value at `address` is less than `val`, then `val` is written at `address`. Since the function accepts only integer numbers, $\lfloor \hat{\tau}_t * \texttt{INT\_MAX} \rfloor$ is passed instead of $\hat{\tau}_t$. Given that $\hat{\tau}_t \in [0; 1]$, no overflow can occur.

During the execution of the algorithm, the integer representation of the resulting array is located in the shared memory. This memory is used since it has shorter access time than global memory. When the execution is completed, the result is transferred to global memory at `dst` with each element converted into a floating-point number and divided by `INT_MAX`. This process is also iterative and involves all threads of the block in the same way as in previous cases.

3.6 Computation of the Result of Rule Inference

This operation is performed according to (5) after obtaining the truth value of the antecedent of the rule with respect to the inputs. The algorithm is shown in Fig. 4.

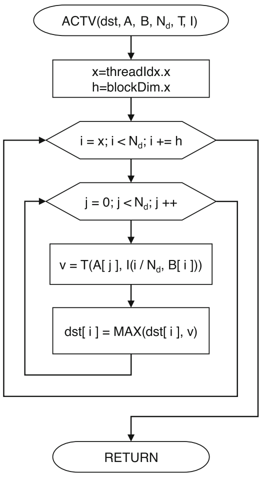
ACTV(dst, A, B, N_d, T, I)
x=threadIdx.x
h=blockDim.x
i = x; i < N_d; i += h
j = 0; j < N_d; j ++
v = T(A[j], I(i / N_d, B[i]))
dst[i] = MAX(dst[i], v)
RETURN

Fig. 4. Flowchart of the computation of rule inference result

The fuzzy truth value of the antecedent `A` is represented in discrete form, whereas the membership function of the consequent term `B` is given in analytical form (type and values of parameters). The discrete representation of the result is placed at `dst`. According to (5), it is necessary to have the values of `B` in the same points in which the result $\mu_{B'_k}$ must be computed. The discrete representation of `B` is prepared by the main kernel function and stored in shared memory; this process is parallelized by distributing the samples among threads. Similarly to the previous operation, a double loop is executed for two variables, `i` and `j`, to enumerate the samples of y and t, respectively. The computational complexity of this algorithm is $O\left(N_d^2\right)$.

The iterations of the loop for `i` are distributed among all threads; the loop for `j` is entirely executed by a single thread. The values of `i` and `j` correspond to $y = \mathtt{i}/N_d$ and $t = \mathtt{j}/N_d$, respectively. The values of $T\left(\mathtt{A[j]}, I\left(\mathtt{j}/N_d, \mathtt{B[i]}\right)\right)$ are computed in the body of the inner loop; the maximum of these values is recorded at `dst[i]` after leaving the loop. Unlike the convolution of fuzzy truth values, this algorithm does not cause data races since every sample `dst[i]` of the result array is processed by a single thread.

3.7 Applying Rule Weights

The transformation of the membership function of the rule inference result is pointwise, i.e. it transforms every sample value separately (see (9)). The function $f\left(a, w\right)$ is implemented in the program as

$$f\left(a, w\right) = a * w + \left(1 - w\right).$$

This function was designed by analogy with [4], where $f\left(a, w\right)$ was defined as $f\left(a, w\right) = a * w$, which is equivalent to $f\left(a, w\right) = a * w + 0 * \left(1 - w\right)$. Thus, w determines the proportion of the original function in a linear combination with $\mu_{B'_j}\left(y\right) = 0$, which is the neutral element for the union of fuzzy sets. The logical-type fuzzy system under consideration defines the accumulation through intersection, whose neutral element is $\mu_{B'_j}\left(y\right) = 1$.

The flowchart of the algorithm is depicted in Fig. 5. Its computational complexity is $O\left(N_d\right)$. The arguments of the operation are the membership function of the rule inference result `A` in discrete representation and the rule weight `w`. The algorithm contains a single loop for the variable `i` which denotes the sample index; the iterations are distributed among threads. The argument `A` and the result are both located in global memory.

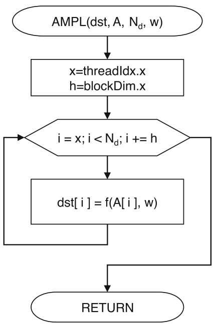

Fig. 5. Flowchart for application of rule weights

3.8 Accumulation

Accumulation is the computation of the inference B' of the fuzzy system as a whole. For logical-type systems, it is defined through the intersection of the inference results of all rules B'_j, $j = \overline{1, N}$ (see (6)). The intersection is performed pointwise; the membership degree of each sample is determined by applying the t-norm to the membership degrees of the samples in each set B'_j. The flowchart of this algorithm is not given here since it is completely similar to the one shown in Fig. 5, except that the operator in the loop body is `dst[i]=T(A[i],B[i])`. The operation is implemented for $N = 2$; accumulation for greater numbers N is programmed at command sequence level, similarly to the convolution of fuzzy truth values. The arguments `A`, `B` and the result are membership functions in discrete form, located in global memory. The enumeration of samples is implemented by a loop for the variable `i` (sample index); the iterations are distributed among threads.

3.9 Defuzzification

Defuzzification determines the crisp output of the system by accumulation result. In this research, the center-of-gravity defuzzification method was used. In the continuous case, it has the form given in (8), whereas in the discrete case, it can be defined as

$$\overline{y} = \sum_{i=0}^{N_d - 1} y_i \cdot \mu_{B'}(y_i) \Big/ \sum_{i=0}^{N_d - 1} \mu_{B'}(y_i) \, .$$

Both summations, on the numerator and on the denominator, can be distributed among threads; every thread computes its own partial sums, then they are reduced and divided. Barrier synchronization must be done before division; the division itself must be executed by a single thread. The algorithm flowchart is depicted in Fig. 6. The complexity of this algorithm is $O(N_d)$. The argument of the operation is `A`, the membership function of the system inference result in discrete form, which is stored in global memory. The result is placed at `dst`; it points to the memory part for final results. The boundaries `a` and `b` of the base set of the output variable are also passed to the function that implements the algorithm.

In the implementation of the algorithm, the whole sums `numTotal` and `denTotal` of the numerator and the denominator, respectively, are stored in shared memory. Initially, they are assigned zero values by one of the threads; then barrier synchronization is performed (not shown in the flowchart). Next, every thread computes its own partial sums of the numerator and the denominator, executing its iterations of the loop for the sample index `i`. The values y_i are calculated regardless of the original domain of definition [`a`; `b`] of the output variable; it is assumed to be $[0; 1]$ instead. After completion of the loop, the threads increase the values of the whole sums by the values of their partial sums. Since all threads are modifying the same memory locations, we use the atomic addition operation `atomicAdd` which is provided by the CUDA framework. This function

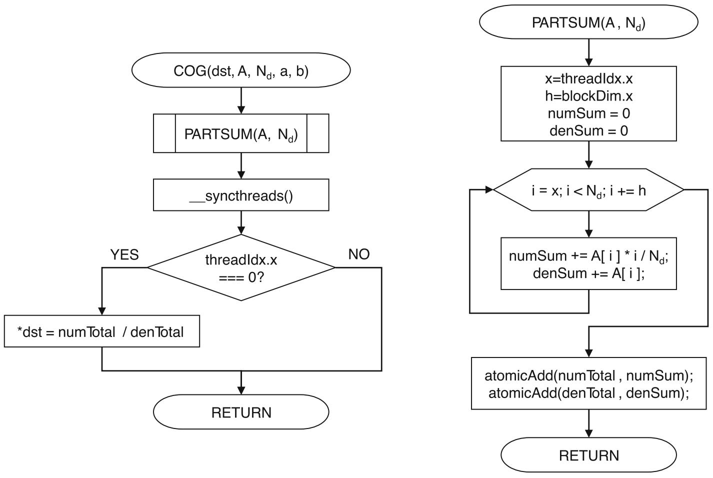

Fig. 6. Algorithm flowchart of the center-of-gravity defuzzification method

has `address` and value `val` as its arguments and increases the value at `address` by `val`. Then barrier synchronization is performed (`__syncthreads()`). Finally, the zeroth thread divides the whole sums of the numerator and the denominator, casts the quotient y_0 to the boundaries of the base set $[\mathtt{a};\mathtt{b}]$, according to the formula $\overline{y} = a + (b - a)\, y_0$, and records the result as $\overline{y}$ at `dst`, while other threads do nothing.

4 Benchmark

Let us consider a neuro-fuzzy system that approximates the analytical dependency $f(x_1, x_2) = (x_1 - 7)^2 \cdot \sin(0.61x_2 - 5.4)$. The plot of this dependency is shown in 7. The rule base compiled according to the shape of the plot is given in Table 1.

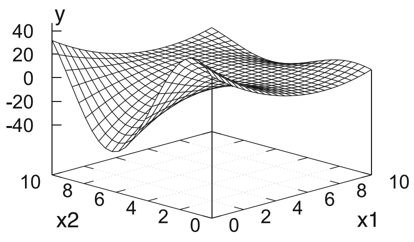

Fig. 7. Approximate dependency plot

Table 1. Rule base

x_1	x_2	y
Low	Low	High
Low	Medium	Low
Low	High	High
Medium	—	Medium
High	Low	Above medium
High	Medium	Below medium
High	High	Above medium

The neuro-fuzzy system has two inputs, each having three terms, six rules with two subconditions, and one rule with one subcondition. The computation of fuzzy truth values must be performed six times, for each term of each input variable. The convolution of fuzzy truth values occurs six times since the rule base contains six rules having two subconditions. The rule inference result is computed seven times, then each of them is applied a weight factor transformation. Finally, accumulation is performed $N - 1 = 6$ times, followed by defuzzification. The neuro-fuzzy system performs 33 operations in total; 13 of them have complexity $O\left(N_d^2\right)$. All terms are defined by Gaussian membership functions; each of them has two parameters for adjustment. Moreover, each rule has its own adjusted weight. The system contains altogether $(3 + 3 + 5) * 2 + 7 = 29$ parameters.

Let us estimate the computational complexity of system learning. A common number of generations for evolution strategies is $N_g = 300$; $\mu = 40$, $\lambda = 4 * \mu = 160$. Let the size of the training set be $M = 80$. In this situation, fuzzy inference is executed $N_g * \lambda * M = 3.84$ million times. Every inference requires $13 * N_d^2 + 20 * N_d$ iterations of loops. For $N_d = 1024$, it results in 13.6 millions of iterations; so the entire learning process requires approximately $5.2 * 10^{13}$ iterations. Figure 8

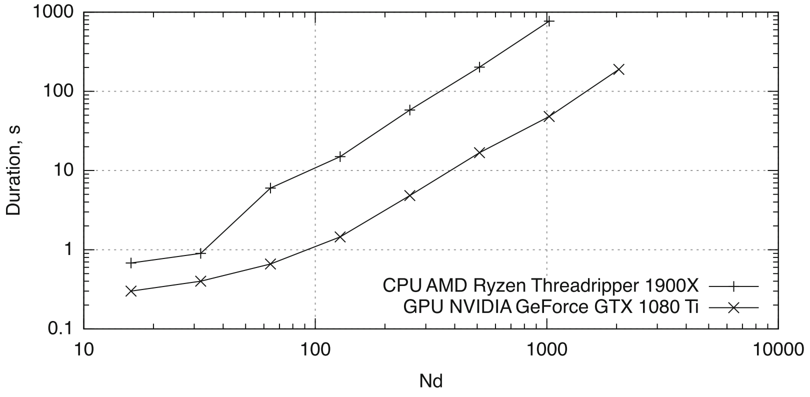

Fig. 8. A comparison of the learning process duration on CPU and GPU

portrays the results of an experiment comparing the duration of the learning process on CPU and GPU for different values of N_d. We used a parallel CPU-based implementation of the neuro-fuzzy system. For $N_d = 1024$, learning on CPU took 769.33 seconds, whereas on GPU it completed in 48.06 seconds. Thus, the GPU-based implementation accelerated the learning process by a factor of 16 approximately.

5 Conclusions

The parallel algorithms we have developed make it possible to significantly accelerate neuro-fuzzy system learning, which makes them more useful in practical applications. Owing to its high time complexity, the problem of acceleration becomes essential. The benchmark we provided demonstrates that the GPU-accelerated implementation of learning shortens its duration by a factor of 16. The suggested neuro-fuzzy system also allows for handling fuzzy input values, in contrast to most modern fuzzy modeling frameworks (see [6]).

References

1. Borisov, A., Alekseev, A., Krumberg, O., et al.: Decision Making Models Based on Linguistic Variable. Zinatne, Riga (1982). (in Russian)
2. Borisov, V., Kruglov, V., Fedulov, A.: Fuzzy Models and Networks. Hot Line - Telecom, Moscow (2007). (in Russian)
3. Dobuis, D., Prade, H.: Possibility Theory. Applications to the Representation of Knowledge in Informatics. Radio and Communication, Moscow (1990). (in Russian)
4. Programmable controllers - part 7: Fuzzy control programming. International Standard Electrotechnical Commission, Geneva, Switzerland (2000)
5. Kutsenko, D., Sinuk, V.: Inference method for systems with multiple fuzzy inputs. Bull. Russ. Acad. Sci. Control Theory Syst. **3**, 48–56 (2015). https://doi.org/10.7868/S0002338815030129. (in Russian)
6. Leonenkov, A.: Fuzzy Modeling in MATLAB and FuzzyTech Environment. BHV - Petersburg, Saint Petersburg (2003). (in Russian)
7. Rothstein, A., Shtovba, S.: Identification of nonlinear dependence by fuzzy training set. Cybern. Syst. Anal. **2**, 17–24 (2006). (in Russian)
8. Rutkowska, D., Pilinsky, M., Rutkowsky, L.: Neural Networks, Genetic Algorithms and Fuzzy Systems. Hot Line - Telecom, Moscow (2004). (in Russian)
9. Rutkowsky, L.: Methods and Techniques of Computational Intelligence. Hot Line - Telecom, Moscow (2010). (in Russian)
10. Sinuk, V.G., Polyakov, V.M., Kutsenko, D.A.: New fuzzy truth value based inference methods for non-singleton MISO rule-based systems. In: Abraham, A., Kovalev, S., Tarassov, V., Snášel, V. (eds.) Proceedings of the First International Scientific Conference "Intelligent Information Technologies for Industry" (IITI' 16). AISC, vol. 450, pp. 395–405. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33609-1_36

11. Sinuk, V.G., Kulabukhov, S.V.: Neuro-fuzzy system based on fuzzy truth value. In: Kuznetsov, S.O., Osipov, G.S., Stefanuk, V.L. (eds.) RCAI 2018. CCIS, vol. 934, pp. 91–101. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00617-4_9
12. Zadeh, L.: Outline of a new approach to the analysis of complex systems and decision processes. IEEE Trans. Syst. Man Cybern. **3**(1), 28–44 (1973)
13. Zadeh, L.: PRUF - a meaning representation language for natural language. Intern. J. Man-Mach. Stud. **10**, 395–460 (1978)