Speedup of Type-1 Fuzzy Logic Systems on Graphics Processing Units Using CUDA

Durlabh Chauhan[1], Satvir Singh[2], Sarabjeet Singh[3] and Vijay Kumar Banga[4]
[1,2]*Department of Electronics & Communication Engineering,*
SBS State Technical Campus, Ferozepur–152004, (Punjab) India
[3]*Department of Computer Science & Engineering,*
SBS State Technical Campus, Ferozepur–152004, (Punjab) India
[4]*Department of Electronics & Communication Engineering,*
Amritsar College of Engg. & Tech., Amritsar–143001, (Punjab) India
E-mail: [1]*er.durlabh@gmail.com,* [2]*drsatvir.in@gmail.com,*
[3]*sarabjeet_singh13@yahoo.co,* [4]*v_banga@rediffmail.com*

***Abstract*—Parallelcomputing is one of significant components of the High Performance Computing (HPC) and is being used to solve problems, which are large and complex in nature. Fuzzy Logic System (FLS) is a problem that becomes computationally intensive with increase in number of inputs and/or fuzzy rules. Running an FLS is highly parallel in nature, therefore, can be implemented in parallel on GPU using CUDA. In this paper, various fuzzy computations viz. rule firing, implication, aggregation and defuzzification are performed in parallel. Multiple threads are run at a time to fire multiple fuzzy rules, simultaneously, that reduces the overall FLS execution time. It is observed from simulation results that GPU works faster as compared to CPU when either number of inputs is increased or number of fuzzy rules.**

Keywords: High Performance Computing, Fuzzy Logic Systems, General Purpose Computing on Graphics Processing Unit, Compute Unified Design Architecture

I. INTRODUCTION

Artificial Intelligence (AI) undoubtedly is the backbone of the emerging technological advancements, however, implementation involves intensive computation owing to increased data sizes. Methods to improve the runtime performance construe the areas of research to reduce mathematical computations and parallelization of algorithm in hardware. Graphics Processor Units (GPU), plays a major role in gaming and graphics applications and allows for general purpose programming on remarkably fast parallel hardware using a Single Instruction Multiple Data (SIMD) programming architecture. Fuzzy Logic, introduced by Zadeh [1], [2], is one of the exigent part of AI and possess inherent parallel nature. General Purpose computing on GPU (GPGPU) is targeted by many researchers to speed up complicated algorithms especially, AI algorithms and their applications [3]. Ander-son *et al.* presented a GPU solution for the fuzzy C-means clustering algorithms [4]. Earlier this solution used OpenGL and Cg (graphics libraries) to achieve approximately two folds of computational speedup for some clustering profiles using NVIDIA 8800 GPU. They later generalized the system for the use of non-Euclidean metrics [5]. Further, Sejun Kim describes the method used to adapt a multilayer tree structure composed of fuzzy adaptive units into CUDA (Compute Unified Device Architecture) platforms [6].

In [7], Chiosa and Kolb present a framework for mesh clustering solely implemented on the GPU with a new generic multilevel clustering technique. Chia *et al.,* have proposed the implementation of a zero-order TSK-Fuzzy Neural Network (FNN) on GPUs to reduce training time in [8].

Harvey *et al.,* have presented a GPU solution for fuzzy inference system in [9]. Anderson *et al.,* present a parallel implementation of fuzzy inference on GPU using CUDA in[10]. Two folds of speed improvement of this naturally parallel algorithm have been achieved under typical inference profiles. One problem with this system and the implementation on GPU is that they both rely upon OpenGL and Cg libraries, which makes system generalization difficult for new comers to GPGPU. Further, Ngo *et al.,* report an implementation of Interval Type-2 FLS on GPU using CUDA with a tremendous speedup of 30 folds in [11].

In this paper, parallel functionality with CUDA programming model, for parallel implementation of a Type-1 Sugeno FLS on GPU, is investigated with increased number of inputs and fuzzy rules. After this brief historical background rest of this paper is outlined as follows: Section II targets CUDA programming model for GPGPU. Section III introduces Type 1 FLS along with the scope of parallelism wherever possible. Section IV presents simulation results and a speedup comparison of serial and parallel computing using CPU and GPU, respectively. Finally, section V concludes the research workand presents future off-shoots.

II. CUDA PROGRAMMING MODEL

CUDA is a parallel computing platform and programming model introduced by NVIDIA that increases computing performance substantially by harnessing the power of parallelism of the GPU. CUDA gives program developers the direct access to the virtual instruction set and memory of the parallel computational elements in CUDA enabled GPUs. The CUDA platform is accessible to software developers through CUDA accelerated libraries, compiler directives (such as Open ACC), and extensions to industry-standard programming languages, including C, C++ and FORTRAN. C/C++ programmers use

CUDAC/C++, compiled with nvcc which is NVIDIA'S LLVM-based C/C++ compiler. A C/C++ program using CUDA can interface with one GPU or multiple GPUs and can be identified and utilized in parallel, allowing for unprecedented processing power on desktop computers.

CUDA allows multiple kernels to be run simultaneously on GPU cores. CUDA refers to each kernel as a grid. A grid is a collection of blocks. Each block runs the same kernel, however, is independent of each other (this has significance in terms of access to memory types). A block contains threads, which are the smallest divisible unit on a GPU.

A thread block is a number of SIMD threads that work on core at a given time. Threads can exchange information through the shared memory and can be synchronized. The operations are systematized as a grid of thread blocks. For parallel operation the programming model allows a developer to partition a program into several subprograms, each of which is executed independently on a block. Each subprogram can be further divided into finer pieces that perform the same function but execute on different threads independently within the same block. For data set parallelism, data sets can be divided in to smaller chunks that are stored in the shared memory, and each chunk is visible to all threads of the same block. This local data arrangement approach reduces the need to access off-chip global memory, which reduces data access time.

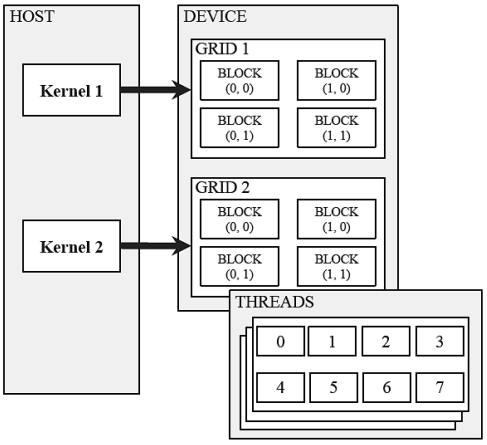

Fig. 1 CUDA Architecture

The next critical component of a CUDA application is the memory model. There are multiple types of memory and each has different access times. The GPU is broken up into read-write per thread registers, read-write per thread local memory, read-write per-block shared memory, read-write per-grid global memory, read-only per-grid constant memory, and read-only per-grid texture memory. Texture and constant memory have relatively small access latency times, while global memory has the largest access latency time. Applications should minimize the number of global memory reads and writes. This is typically achieved by having each thread read its data from global memory and store its content into shared memory.

The basic structure of a CUDA code comprises of allocation of memory space (using cuda Malloc function) on device (GPU) and (using regular malloc function) on host (CPU). Data which is copied from the host to the device for the call of kernel routine to be executed on the GPU (using function cuda Memcpy) also defines the number of threads and their physical structure. Kernel is prefixed with the global keyword. Results are transferred from GPU to CPU in the same fashion as data is copied from host to device.

III. SCOPE OF PARALLELISM IN TYPE-1 FLS

Theory of FLS given by Zadeh a fuzzy set is defined for a particular domain, and it is characterized by a membership function that maps elements from the domain to a real valued numbers [1], [2]. Mendel and many researchers gave numerous methods to design and implement FLS for various applications [12]–[15]. Theory of FLS given by Zadeh a fuzzy set is defined for a particular domain, and it is characterized by a membership function that maps elements from the domain to a real valued numbers.

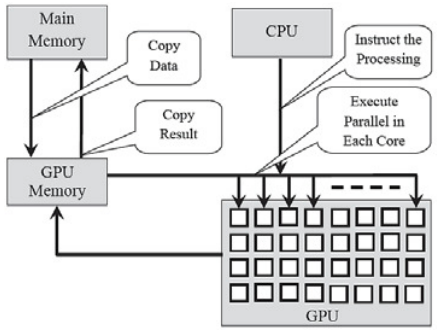

Fig. 2 CUDA Process Flow

In this paper every input has been fuzzified using four arbitrary located Gaussian membership functions that are characterized by two parameters, i.e., mean (m) and standard deviation (σ). Gaussian membership function is symmetrical about its mean and is expressed mathematically as (1)

$$\mu(x) = \exp\left(\frac{-(x-m)^2}{2\sigma^2}\right) \tag{1}$$

In this paper, an FLS with 4 inputs and 1 output is subjectedto parallel computation for forecasting of Mackey-Glass timeseries [14]. For implication and aggregation, min and maxoperator are investigated. The defuzzification is performedwith the height defuzzification method as symmetrical shaped Gaussian fuzzy sets have been used to fuzzify consequent.Output of the height defuzzifier is given by (2)

$$y = \frac{\sum_{i=1}^{M} C_i \mu(x_i)}{\sum_{i=1}^{M} \mu(x_i)} \quad (2)$$

Here C represents the location of singleton consequents fuzzy sets and (x_i) represent clipping level for each rule after implication and M represents total number of fuzzy rules. However, for implementation of the FLS defined above in CUDA, the foremost and the most crucial step is to allocate the memory space for the data sets to be used in the system,as the choice and format of data affects performance of thealgorithm.

Scope of possible parallel computational processing is discussed as follows: Harvey *et al.,* [9] and Anderson *et al.,* [10] in their respective work have given a vast scope of parallelism for Type-1 FLS. In the same fashion Ngo *et al.,* [11] presented a novel scope of parallelism for Interval Type-2 FLS. However, in all these implementations the emphasis was laid to parallelize the number of fuzzy rules and discrete levels for a typical FLS with two inputs and single output. So, a single FLS was computed with parallel rule inference on GPU using CUDA. However, a typical FLS can be processed multiple times with fixed fuzzy rules and discrete levels but varying inputs. Here lies our scope of parallelism, we construe our code to compute multiple FLS in parallel on GPU as a FLS serially on CPU will consume more time.

IV. Parallel I Mplementation

Two $M \times N$ dimensional 'Mean' and 'Sigma' matrices are used in this implementation where M denotes number of fuzzy rules and N is number of antecedents. These matrices hold mean and sigma values of Gaussian membership function in accordance with fuzzy rules used in the FLS. An $M \times 1$ dimensional 'Consequent' matrix contains only mean values of the consequent fuzzy sets as systems uses height defuzzificationmethod given by equation (2). Multiple inputs are provided to the systems collectively in the form of an $L \times N$ dimensional input matrix, X, where L is the number of inputs.The CPU executes rule firing sequentially with a single input at a time and causes more computational time. Whereas, multiple threads are run at the same time to fire multiple rules simultaneously using system matrices, i.e., 'Mean', 'Sigma' and 'Consequent' matrices,which reduce the execution time for the FLS. A kernel function is initialized from CPU to pass inputs in parallel to various GPU cores. Copying system matrices everytime along with input vectors and increases the GPU processing time. Therefore, system matrices are copied only once and subsequently require only input vectors those are passed to GPU in parallel to enhance the GPU performance.

V. Results Discussion

The speed up performance with GPU implementation of a Type-1 Sugeno FLS was compared with that of CPU implementation. Intel Core 2 Duo system under experimentation has 2GB of system RAM, and Windows 7 platform. The GPU used here works on nVIDIAGeforce GTX 650 with 1024 MB of texture memory, 192 stream processors, and PCI Express X16.

The number of antecedents is fixed to 4 and consequentto 1, the number of rules is varied between 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 and inputs were varied as 128, 256, 512, and 1024. A set of input data is obtained from Mackey-Glass time series for experimentation. Ratio of CPU to GPU runtimes with respect to number of fuzzy rules has been tabulated in Table I and presented graphicallyin Fig. 3.

TABLE I Fuzzy Rules *vs.* CPU to GPU Speedup Ratio

Number of Fuzzy Rules	CPU to GPU Runtime Speedup Ratio			
	128 Inputs	256 Inputs	512 Inputs	1024 Inputs
10	1.937	4.875	5.034	5.488
20	2.437	3.032	4.548	6.132
30	1.340	2.319	4.319	6.544
40	2.000	2.476	3.968	7.063
50	1.516	3.532	4.410	7.063
60	1.730	3.000	4.365	7.185
70	1.602	3.205	4.509	7.134
80	1.500	2.742	4.500	7.832
90	1.709	2.577	4.446	5.488
100	1.624	2.504	4.652	6.132

Here, it can be observed clearly that advantage of GPU computing has increased with increased input data vector sizes. In another simulation results, it is observed that CPU to GPU speedup time improves with increase in fuzzy rules. Computational time on CPU varies significantly due to already always running applications at the back end.Therefore, to present fair comparison serial and parallel timing analysis experiments with same FLS have been repeated 30 times. Average of CPU to GPU speedups for 30 monte-carlosimulations during serial and parallel computations of rule firing, implication, aggregation and defuzzification have beenpresented in Fig. 4–7.

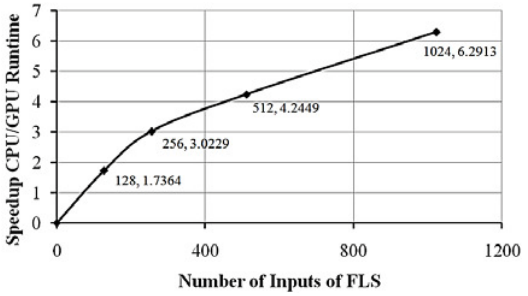

Fig. 3 CPU to GPU Speedup Ratio *vs* Inputs Data Length

The overall performance of CPU to GPU speedup timingswith respect to collective number of inputs, presented to the FLS, is shown in Fig. 8 that depicts that larger the number offuzzy rules better is the speedup performance.

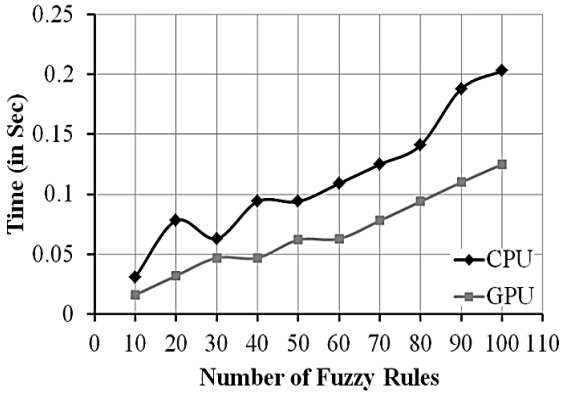

Fig. 4 CPU and GPU Runtime Comparison for 128 Inputs

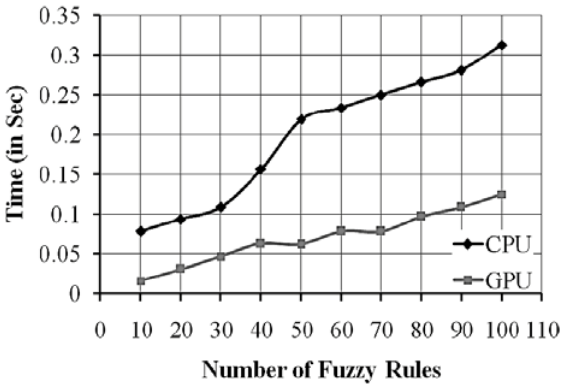

Fig. 5 CPU and GPU Runtime Comparison for 256 Inputs

VI. Conclusion and Future Work

This paper has demonstrated the implementation and comparative runtime performances of a typical Sugeno Type-1 FLSon a GPU and CPU without the use of a graphics API which is flexible, scalable, and can be used by any researcher with knowledge of C. It has been demonstrated that the CPU works equally fast as GPU when the system is small. As the number of rules or the number of inputs increase the GPU outperforms the CPU runtime. Here, in the work nearly 7.83 times speedup could be achieved as 1024 inputs supplied in parallel to the FLS ported on GPU. The GPU has an initial setup overhead of kernel loading and memory transfer, however, subsequent parallel computations leads to a small increase in processing time despite a substantial increase in computational load. On the other hand, CPU has no initial cost, but computation time grows linearly with computational load much beyond GPGPU runtime.

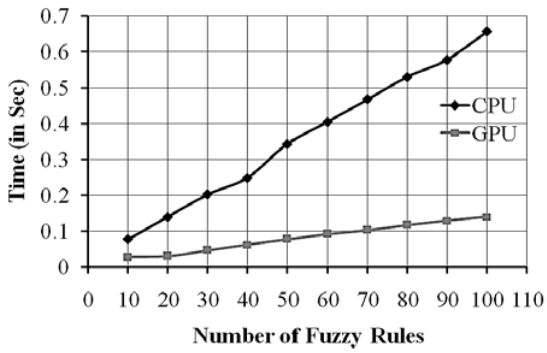

Fig. 6 CPU and GPU Runtime Comparison for 512 Inputs

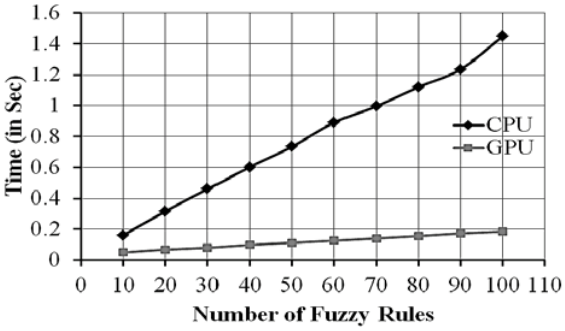

Fig. 7 CPU and GPU Runtime Comparison for 1024 Inputs

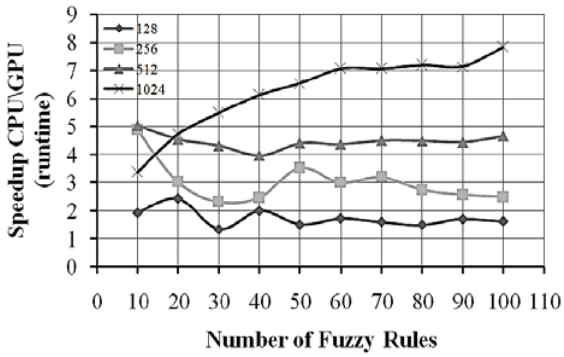

Fig. 8 CPU to GPU Speedup Ratio for Various Input Data Sets

The parallelization of more FLS applications is next on ouragenda. That will follow implementation of Interval Type-2 FLSs and Generalized Type-2 FLSs for various applications that require much more computational time otherwise.GPGPU is also possibly investigated to on Evolutionary Algorithms those are computational intensive and parallel innature.

References

[1] L.A. Zadeh, "Fuzzy Sets," *Information and Control*, Vol. 8, No. 3, pp. 338–353, 1965.

[2] L.A. Zadeh, "Fuzzy Logic and Approximate Reasoning," *Synthese*, Vol. 30, pp. 407–428, 1975.

[3] S. Singh, S. Singh, V.K. Banga, and D. Chauhan, "CUDA for GPGPU Applications-A Survey," in *National Conference on Contem-poraryTechniques & Technologies in Electronics Engineering*, Murthal,Sonepat, India, March 2013, p. Accepted.

[4] D.T. Anderson, R.H. Luke, and J.M. Keller, "Speedup of FuzzyClustering through Stream Processing on Graphics Processing Units," *IEEE Transactions on Fuzzy Systems*, Vol. 16, No. 4, pp. 1101–1106, 2008.

[5] D. Anderson, R.H. Luke, and J.M. Keller, "Incorporation of non-EuclideanDistance Metrics into Fuzzy Clustering on Graphics Processing Units," in *Analysis and Design of Intelligent Systems using Soft Computing Techniques*. Springer, pp. 128–139, 2007.

[6] S. Kim and D. Wunsch, "A GPU based Parallel Hierarchical Fuzzy ART Clustering," in *The 2011 International Joint Conference on Neural Networks (IJCNN)*, pp. 2778–2782, 2011.

[7] I. Chiosa and A. Kolb, "GPU-based Multilevel Clustering," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 17, No. 2, pp. 132–145, 2011.

[8] C.F. Juang, T.C. Chen, and W.Y. Cheng, "Speedup of Implementing Fuzzy Neural Networks with High-dimensional Inputs through Parallel Processing on Graphic Processing Units," *IEEE Trans-actions on Fuzzy Systems*, Vol. 19, No. 4, pp. 717–728, 2011.

[9] N. Harvey, R. Luke, J.M. Keller, and D. Anderson, "Speedup of Fuzzy Logic through Stream Processing on Graphics Processing Units,"in 2008. *CEC IEEE World Congress on Computational Intelligence* and *Congress on Evolutionary Computation*, pp.3809–3815, 2008.

[10] D. Anderson and S. Coupland, "Parallelisation of Fuzzy Inference on a Graphics Processor Unit using the Compute Unified Design Architecture,"in *Proceedings of the UK Workshop on Computational Intelligence (UKCI'08)*, pp. 1–6, 2008.

[11] L.T. Ngo, D.D. Nguyen, C.M. Luong *et al.,*., "Speedup of Interval Type 2 Fuzzy Logic Systems based on GPU for Robot Navigation," *Advances in Fuzzy Systems*, Vol. 2012, pp. 4, 2012.

[12] J.M. Mendel, "Fuzzy Logic Systems for Engineering: A Tutorial,"*Proceedings of the IEEE FUZZ*, Vol. 83, No. 3, pp. 345–377, 1995.

[13] G.C. Mouzouris and J.M. Mendel, "Non-singleton Fuzzy Logic Systems: Theory and Application," *IEEE Transactions on Fuzzy Systems*, Vol. 5, No. 1, pp. 56–71, 1997.

[14] N.N. Karnik and J.M. Mendel, "Applications of Type-2 Fuzzy Logic Systems to Forecasting of Time-series," *Information Sciences*, Vol. 120, No. 1, pp. 89–111, 1999.

[15] J.M. Mendel, "Uncertainty, Fuzzy Logic, and Signal Processing," *Signal Processing*, Vol. 80, No. 6, pp. 913–933, 2000.