PARALLEL FUZZY LOGIC CONTROLLER IMPLEMENTATION USING MPICH2

By

Prof. Dr. Bakir A.R. AL-Hashemy
Electrical Engineering Department
Baghdad University

AboTalib H. Mahfoodh
Electrical Engineering Department
Baghdad University

ABSTRACT

In this work FLC program is implemented using C++ codes. Two implementations are presented one with the rules stored inside the program, the other with rules in a rulebase file. The execution times of these two implementations, along with MATLAB FLC implementation, are compared using different simulated FLCs. Furthermore, to reduce the rulebase searching time, a parallel FLC is implemented using C++ and MPI (Message Passing Interface). The MPICH2 package is used to run the parallel FLC. A cluster of four computers is used as the parallel environment. The execution time of this FLC program is evaluated using servomotor, Anti Skid System, and other simulated applications. The speedup and efficiency are studied using different number of computers. The results show that decomposing the rulebase searching operation to more than a computer reduce the execution time significantly.

الخلاصة

في هذا العمل برنامج FLC يكتب في سي ++. هناك طريقتين لتطبيق هذا البرنامج، واحد بالقواعد مخزونة داخل البرنامج، الآخر بالقواعد مخزونة في ملف. إنّ أوقات تنفيذ هذه التطبيقين، سويّة مع تطبيق ال MATLAB، تقارن بإستعمال انظمة FLC مختلفة. بالإضافة إلى ذلك، لتخفيض وقت البحث في ملف القواعدينفذ FLC متوازي بإستعمال سي ++ و MPI (وصلة مرور رسالة). رزمة MPICH2 تستعمل لتنفيذ الـ FLC المتوازي. البيئة المتوازية عبارة عن عنقود من أربع حاسبات متصلة في شبكة واحدة. يقيم وقت التنفيذ بإستعمال نظام FLC لـمحرك تؤازري، نظام ضدّ التزلج ، وتطبيقات وهمية أخرى. إنّ التعجيل والكفاءة تدرس بإستعمال عدد مختلف من الحاسبات. النتائج تدل بانه تجزئة عملية البحث في ملف القواعد إلى أكثر من حاسوب يخفّض وقت التنفيذ بشكل ملحوظ. \

KEY WORDS: Fuzzy Logic, Fuzzy Logic Controller, Parallel Computing, MPI.

INTRODUCTION

Nowadays, the areas where Fuzzy Control has been applied comprise a wide variety of applications, with different complexities and performance. Fuzzy Controllers can be found in washing machines, automatic focusing for video cameras, automatic TV tuner, servo motor control, automotive anti-skid brake, and many other consumer appliances. The application of Fuzzy Logic exceeds the control domain since it is also employed for others knowledge-based decision making tasks. Among the latter, medical diagnosis, business forecasting, traffic control, network management, image processing, signal processing, computer vision, geology, and many more [1].

Table 1 covers a range of research areas related to Fuzzy Logic as reported in the IEEE 2001 International Conference on Fuzzy Systems [1].

Table 1 Main Research areas in Fuzzy Logic (FUZZ-IEEE 2001)

Research area	Main Topics
Fuzzy Mathematics	Foundations of fuzzy logic, approximate reasoning, evolutionary computation, identification and learning algorithms, rule base optimization.
Control systems	Fuzzy control theory and applications, process and environmental control, stability criterions issues, multilevel supervisory control.
Pattern recognition and image processing	Supervised and unsupervised learning, classifiers design and integration, signal/image processing and analysis, computer vision, multimedia applications.
Soft computing and hybrid system	Intelligent information systems, database systems, data mining, intelligent agents, reliability engineering, Nero-Fuzzy systems, internet computing, networks traffic modeling and control.
Electronic systems	Fuzzy hardware implementation and embedded applications.
Robotic and Automation	Fuzzy logic in robotics, industrial automation and other industrial applications.

There is rapid increasing in number of applications of FL (Fuzzy Logic), in the domain of Image Processing, Signal Processing and Power Electronics that have been reported in the last decade. Most of them need real-time processing, fast transient behavior, low-power consumption and/or autonomy. In such cases, an effective implementation is required. Implementing these applications on parallel computer can have direct effect on the system efficiency.

A parallel computer is a set of processors that are able to work cooperatively to solve a computational problem. This definition is broad enough to include parallel supercomputers that have hundreds or thousands of processors, networks of workstations, multiple-processor workstations, and embedded systems. The need for faster computers is driven by the demands of both data-intensive applications in commerce and computation-intensive applications in science and engineering [2].

In 1986 Gupta, Forgy, Newell, and Wedig [3] stated that Rule-based systems, appear to be capable of exploiting large amounts of parallelism. It is possible to match each rule to the data memory in parallel. In practice, however, they have showed that the speed-up from parallelism is quite limited, less than 10-fold. Howard, Taylor, and Allinson [4] presented Cellular automata (CA) mechanisms. York Fuzzy Automata Machine (FAMe) is a massively parallel fuzzy CA machine. They have described the structure of the Fuzzy Automata Machine and showed how large, complex fuzzy parallel systems may be constructed in consequence. Lees, Campbell, and Devlin [5] presented an application specific parallel rule inference architecture which is capable of performing an entire rule inference within one clock cycle. The design is targeted for high capacity Complex Programmable Logic Devices (CPLDs), whose ability to be reconfigured allows the application specific rule structure to be practical for real world systems.

In this work different type of FLC implementation were provided to reduce the execution time of an FLC program. An efficient FLC algorithm is designed, and implemented in C++. Complex FLC programs spend long time in searching the rulebase for fired rules. To decrease the program execution time, parallel FLC algorithm is designed and implemented in C++ using MPICH2 (Message Passing Interface Chameleon) package. MPICH2 is used to manage and

distribute the jobs over computers. The parallel FLC algorithm decreases the execution time by decomposing the rulebase searching operation into more than a task that can be done concurrently. Since minimum execution time is desired, number of computers in the parallel environment should be determined precisely to obtain the desired execution time.

* FUZZY LOGIC CONTROLLERS

The concept of FL was conceived by L. Zadeh [6] in 1965, and presented not as a control methodology, but as a way of processing data by allowing partial set membership rather than crisp set membership or non-membership. This approach to set theory was not applied to control systems until the 70's due to insufficient small-computer capability prior to that time.

2.1 Fuzzy Logic Controller Structure

The kind of a structure a fuzzy logic controller (FLC) will have, primarily depend on the controlled process and the demanded quality of control. Since the application area for fuzzy control is really wide, there are many possible controller structures, some differing significantly from each other by the number of inputs and outputs, or less significantly by the number of input and output fuzzy sets and their membership functions forms, or by the form of control rules, the type of inference engine, and the method of defuzzification. All that variety is at the designer's disposal, and it is up to the designer to decide which controller structure would be optimal for a particular control problem [7].

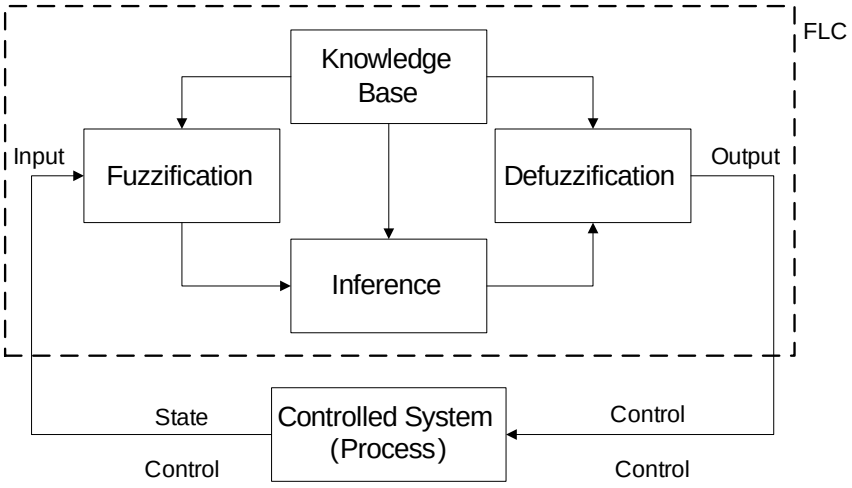


Fig. 1 The FLC structure.

The basic configuration of FLC is shown in **Fig. 1**. The configuration consists of four main components: fuzzification, knowledge base, Inference, and defuzzification [8].

- The fuzzification transforms input crisp values into fuzzy values and it involves the following functions.
  a. Receives the input values,
  b. Transforms the range of values of input variable into corresponding universe of discourse,
  c. Converts input data into suitable linguistic values (fuzzy sets). This part is necessary when input data are fuzzy sets in the fuzzy inference.
- The knowledge base contains knowledge of the application domain and the control goals. It consists of the rulebase.
- The inference performs the following functions:

- simulates the human decision-making procedure based on fuzzy concepts,

d. Infers fuzzy control actions employing fuzzy implication and rules.

- The defuzzification involves the following functions

e. A scale mapping which converts the range of output values into corresponding universe of discourse

f. Defuzzification which yields a non-fuzzy control action from an inferred fuzzy control action.

Design Procedure of Fuzzy Logic Controller

- To design an FLC, bellow procedure can be followed [8]:
- Determination of state variables and control variables: In general, the control variable is determined depending on the property of process to be controlled. The state variables should be selected. In general, state, state error and error difference are often used. The state variables are input variables, and the control variables are output of the controller to be developed.
- Determination of inference method: one of the inference methods can be selected. The decision is dependent upon the properties of process to be studied.
- Determination of fuzzification method: It is necessary to study the property of measured data of state variables. If there is uncertainty in the data, the fuzzification is necessary, and the fuzzification method and membership functions of fuzzy sets should be selected. If there is no uncertainty, singleton state variables can be used.
- Discretization and normalization of state variable space: In general, it is useful to use discretized and normalized universe of discourse.
- Partition of variable space: The state variables are input variables of the controller and thus the partition is important for the structure of fuzzy rules. At this step, partition of control space (output space of the controller) is also necessary.
- Determination of the shapes of fuzzy sets: It is necessary to determine the shapes of fuzzy sets and their membership functions for the partitioned input spaces and output spaces.
- Construction of fuzzy rule base: Control rules can be built now. The variables and corresponding linguistic terms in antecedent part and consequent part of each rule are determined. The architecture of rules is dependent upon the inference method determined in step 2.
- Determination of defuzzification strategy: In general, the crisp control values are used and thus a defuzzification method should be determined.
- Test and tuning: It is almost impossible to obtain a satisfactory fuzzy controller without tuning. In general it is necessary to verify the controller and tune it until satisfactory results are achieved.

Complex Fuzzy Logic Controller

The size of the fuzzy rulebase depends on the number of fuzzy rules, while the number of fuzzy rules depends on the number of inputs and on the number of linguistic values (fuzzy sets) associated with each of the variables [7].

In the case of having many input and output variables, it is efficient to parallelize and to inference many fuzzy rules simultaneously. For example for a system with five input variables, respectively the number of total rules are 16,807 (= 7^5) assuming that the number of the linguistic variables for each fuzzy variable is 7 [9].

* PARALLEL COMPUTING

Introduction

The past decade has seen tremendous advances in microprocessor technology. Clock rates of processors have increased from about 40 MHz (e.g., a MIPS R3000, circa 1988) to over 3.0 GHz (e.g., a Pentium 4, Intel 2006). At the same time, processors are now capable of executing multiple instructions in the same cycle. The average number of cycles per instruction (CPI) of high end processors has improved by roughly an order of magnitude over the past 10 years. All this translates to an increase in the peak floating point operation execution rate (floating point operations per second, or FLOPS) of several orders of magnitude. Even though it is expected that technology development will be continue to hold for the near future, there is a limit that will eventually be reached. The most easily understood physical limit is that imposed by the finite speed of signal propagation along a wire. This is sometimes referred to as the speed-of-light argument (or limit). The speed-of-light argument suggests that once the above limit has been reached, the only path to improve performance is the use of multiple processors [10].

The motivations for parallel processing can be summarized as follows:

1. Higher speed, or solving problems faster. This is important when applications have "hard" or "soft" deadlines. For example, there is at most a few hours of computation time to do 24 hour weather forecasting or to produce timely tornado warnings.

2. Higher throughput, or solving more instances of given problems. This is important when many similar tasks must be performed. For example, banks and airlines, among others, using transaction processing systems that handle large volumes of data.

3. Higher computational power, or solving larger problems. This would allow the use of very detailed, and thus more accurate, models or to carry out simulation runs for longer periods of time (e.g., 5-day, as opposed to 24-hour, weather forecasting).

The ultimate efficiency in parallel systems is to achieve a computation speedup factor of p with p processors. Although in many cases this ideal cannot be achieved, some speedup is generally possible. The actual gain in speed depends on the architecture used for the system and the algorithm executed on it [10].

Parallelism Type Classification

Parallel computers can be divided into two main categories of control flow and data flow. In 1966, M. J. Flynn [10] proposed a four-way classification of computer systems based on the notions of instruction streams and data streams. Flynn's classification has become standard and is widely used. Flynn coined the abbreviations Single Instruction Single Data (SISD), Single Instruction Multiple Data (SIMD), Multiple Instruction Single Data (MISD), and Multiple Instruction Multiple Data (MIMD) for the four classes of computers shown in **Fig. 2**, based on the number of instruction streams (single or multiple) and data streams (single or multiple). The SISD class represents ordinary "uni-processor" machines [10].

Computers in the SIMD class, with several processors directed by instructions issued from a central control unit, are sometimes characterized as "array processors." Machines in the MISD category have not found widespread application, but one can view them as generalized pipelines in which each stage performs a relatively complex operation (as opposed to ordinary pipelines found in modern processors where each stage does a very simple instruction-level operation). The MIMD category includes a wide class of computers. For this reason, in 1988, E. E. Johnson [10] proposed a further classification of such machines based on their memory structure (global or distributed) and the mechanism used for communication/synchronization (shared variables or message passing). Again, one of the four categories (GMMP) is not widely used.

The GMSV class is what is loosely referred to as (shared-memory) multiprocessors. At the other extreme, the DMMP class is known as (distributed-memory). Finally, the DMSV class, combining the implementation ease of distributed memory with the programming ease of the shared-variable scheme, is sometimes called distributed shared memory. When all processors in a MIMD-type machine execute the same program, the result is sometimes referred to as single-program multiple data [10]. The parallel environment in this work falls into the DMMP section of the MIMD category.

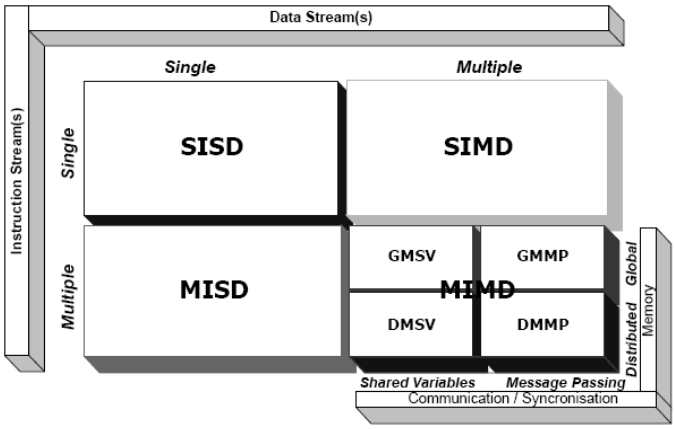


Fig 2 The Flynn-Johnson classification of computer systems.

Parallel algorithm design

Algorithm development is a critical component of problem solving using computers. A sequential algorithm is essentially a recipe or a sequence of basic steps for solving a given problem using a single computer. Similarly, a parallel algorithm is a recipe that tells how to solve a given problem using multiple computers or processors. However, specifying a parallel algorithm involves more than just specifying the steps. At the very least, a parallel algorithm has the added dimension of concurrency and the algorithm designer must specify sets of steps that can be executed simultaneously. This is essential for obtaining any performance benefit from the use of a parallel computer. In practice, specifying a nontrivial parallel algorithm may include some or all of the following [8]:

- Identifying portions of the work that can be performed concurrently.
- Mapping the concurrent pieces of work onto multiple processes running in parallel.
- Distributing the input, output, and intermediate data associated with the program.
- Managing accesses to data shared by multiple processors.
- Synchronizing the processors at various stages of the parallel program execution.
- Typically, there are several choices for each of the above steps, but usually, relatively few combinations of choices lead to a parallel algorithm that yields performance adequate with the computational and storage resources employed to solve the problem. Often, different choices yield the best performance on different parallel architectures or under different parallel programming paradigms [8].

The process of dividing a computation into smaller parts, some or all of which may potentially be executed in parallel, is called *decomposition*. *Tasks* are programmer-defined units of computation into which the main computation is subdivided by means of decomposition. Simultaneous execution of multiple tasks is the key to reducing the time required to solve the entire problem. Tasks can be of arbitrary size, but once defined, they are regarded as indivisible units of computation. The tasks into which a problem is decomposed may not all be of the same size [8].

The tasks, into which a problem is decomposed, run on physical processors. The term process is used to refer to a processing or computing agent that performs tasks. The term process does not adhere to the rigorous operating system definition of a process. Instead, it is an abstract

entity that uses the code and data corresponding to a task to produce the output of that task within a finite amount of time after the task is activated by the parallel program. During this time, in addition to performing computations, a process may synchronize or communicate with other processes, if needed. In order to obtain any speedup over a sequential implementation, a parallel program must have several processes active simultaneously, working on different tasks. The mechanism by which tasks are assigned to processes for execution is called *mapping*. A good mapping should seek to maximize the use of concurrency by mapping independent tasks onto different processes, it should seek to minimize the total completion time by ensuring that processes are available to execute the tasks on the critical path as soon as such tasks become executable, and it should seek to minimize interaction among processes by mapping tasks with a high degree of mutual interaction onto the same process. In most nontrivial parallel algorithms, these tend to be conflicting goals [8].

- Performance Indices of Parallel Computation

To measure the performance of parallel computation some performance indices have been defined. Due to the level of complexity involved, no single measure of performance can give a truly accurate measure of a computer systems performance. Different indices are needed to measure different aspect. Some indices for global measurements are as follow [9]: (for more information about parallel algorithm performance indices refer to [10; 12]).

- Execution time: the execution time measures the time required to run the program, in this work it is the time it takes since the inputs are ready until the output is calculated.
- Speedup (S_p): The speedup factor of a parallel computation using p processors is defined as the ratio: $S_p = \frac{T_1}{T_p}$ Where T_1 is the time taken to perform the computation on one processor and T_p is the time taken to perform the same computation on p processors. Normally the speedup factor is less than the number of processors because of the time lost to synchronization, communication time, and other overheads required by the parallel computation: $1 \leq S_p \leq P$. However as mentioned before there are some cases where this does not apply.
- Efficiency (E_p): The efficiency of a parallel computation is defined as the ratio between the speedup factor and the number of processors: $E_p = \frac{S_p}{P} = \frac{T_1}{pT_p}$ Efficiency is a measure of the cost-effectiveness of computations.
  - Fired rules job: is the task of checking a rule to see if it is a fired rule or not, and if it is a fire rule, calculating its strength.
  - Communication overhead or time: is the time it takes to transfer the required data between computers.

* FUZZY LOGIC CONTROLLER IMPLEMENTATION

Introduction

There are different FLC implementation methods. Deciding which method is best appropriate for a plant depends on several parameters like plant circumstances, cost, execution time, performance, etc. Among FLC implementation classifications are stand alone hardware implementation and software implementation on a computer [13].

In this work C++ code is used to program a general FLC, with few changes, any type of FLC can be achieved. Visual Studio .NET 2003 [14] is used to compile the codes. FLC program have two types, one with the rules implemented inside the program and the other with rules stored in a rulebase file. These two types implementation are compared with FLC implementation using

MATLAB Fuzzy Toolbox. Finally to decrease the program execution time a parallel FLC algorithm is designed and implemented using MPICH2.

Serial FLC Algorithm

The flowchart of the serial FLC algorithm is shown in **Fig. 3**.

Three FLC programs (MATLAB, C++ Type 1 (Rules stored inside the program), and C++ Type 2 (Rules stored in a file on Hard Disk)) are used to test the design. The execution time is the time difference between the time that output is ready and the time of applying inputs. 500 random inputs used to determine the average execution time.

To evaluate the serial FLC algorithm, a serial FLC for servomotor is implemented. The servomotor process shows nonlinear properties, and thus the fuzzy logic control is applied to the motor control. The task of the control is to rotate the shaft of the motor to a set point without overshoot. The set point and process output is measured in degree [8]. State variables (input variable of controller):

Error equals the set point minus the process output (e): $e(k) = \theta_r(k) - \theta(k)$

While θ is the shaft position, θ_r is the shaft set point.

Change of error (ce) equals the error from the process output minus the error from the last process output: $ce(k) = \frac{\theta(k)-\theta(k-1)}{T}$

Control variable (output variable of the controller): Control input (v) equals the voltage applied to the process.

FLC for servomotor has two inputs and 22 rules provided by the expert.

- If e is PB and ce is any, then v is PB.
- If e is PM and ce is NB, NM, or NS, then v is PS.
- If e is ZE and ce is ZE, PS, or PM, then v is ZE.
- If e is PS and ce is NS, ZE, or PS, then v is ZE.
- If e is NS and ce is NS, ZE, PS, or PM, then v is NS
- If e is NS or ZE and ce is PB, then v is PS.

The execution time of the serial FLC for servomotor on MATLAB and C++ are shown in Fig. 4.

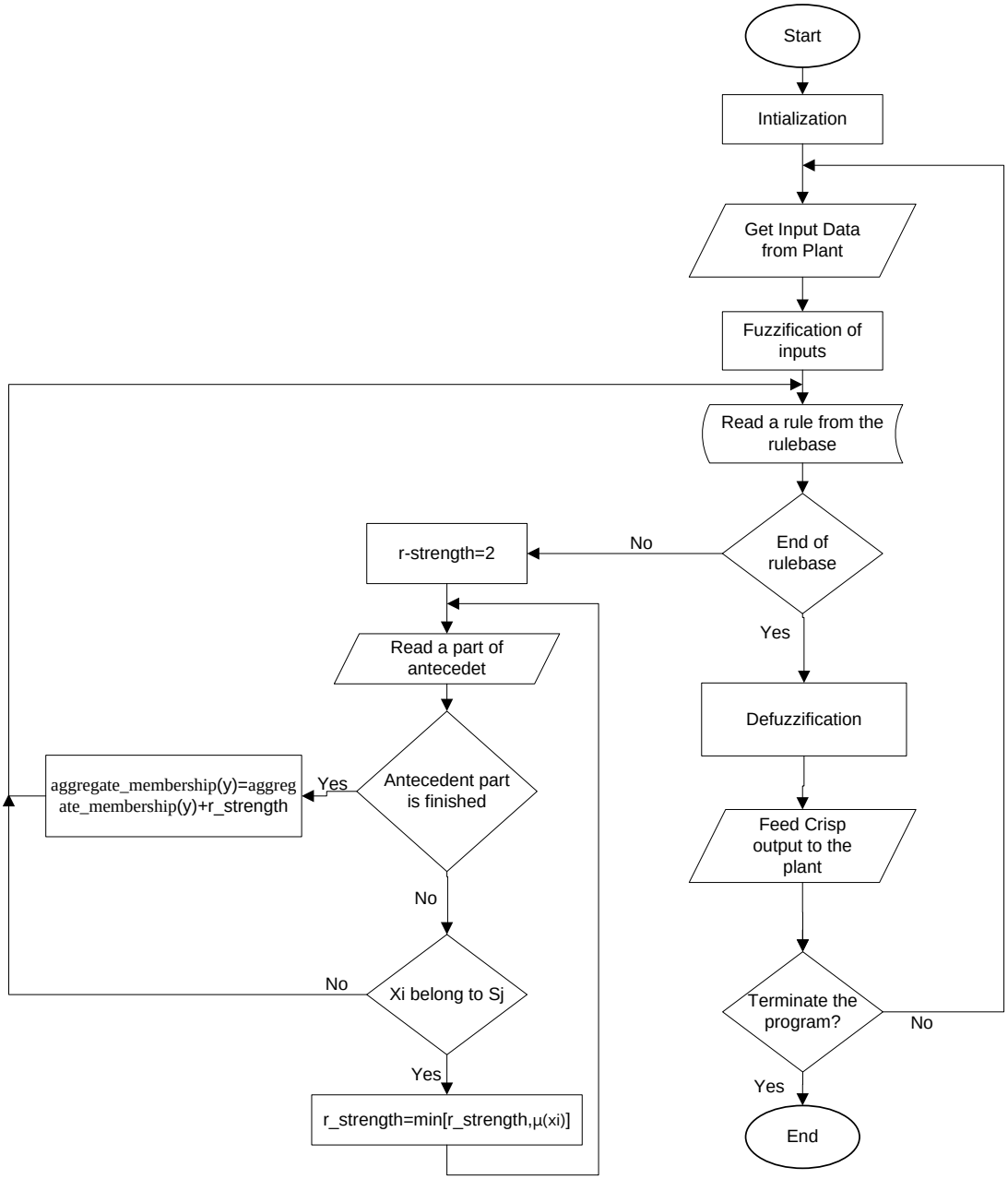


Fig. 3 Flowchart of the serial Fuzzy Logic Controller algorithm.

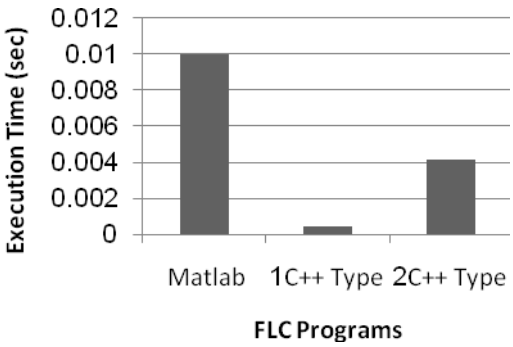


Fig. 4 The execution time of servomotor FLC programs

Fig. 5 shows the execution time of the three above mentioned serial implementations of various virtual FLC systems with different number of inputs and rules. As it is shown in **Fig. 5** execution time of all three programs grows rapidly with increasing number of rules.

The execution time of all FLC programs is combined in **Fig. 5.d**, while MATLAB has the highest execution time, the C++ type 1 shows the smallest execution time. Although C++ Type 1 program is show slow increase in execution time and even for 2401 rules it have very small execution time in comparison with other programs but this type of FLC cannot be used for complex problems because of memory problem.

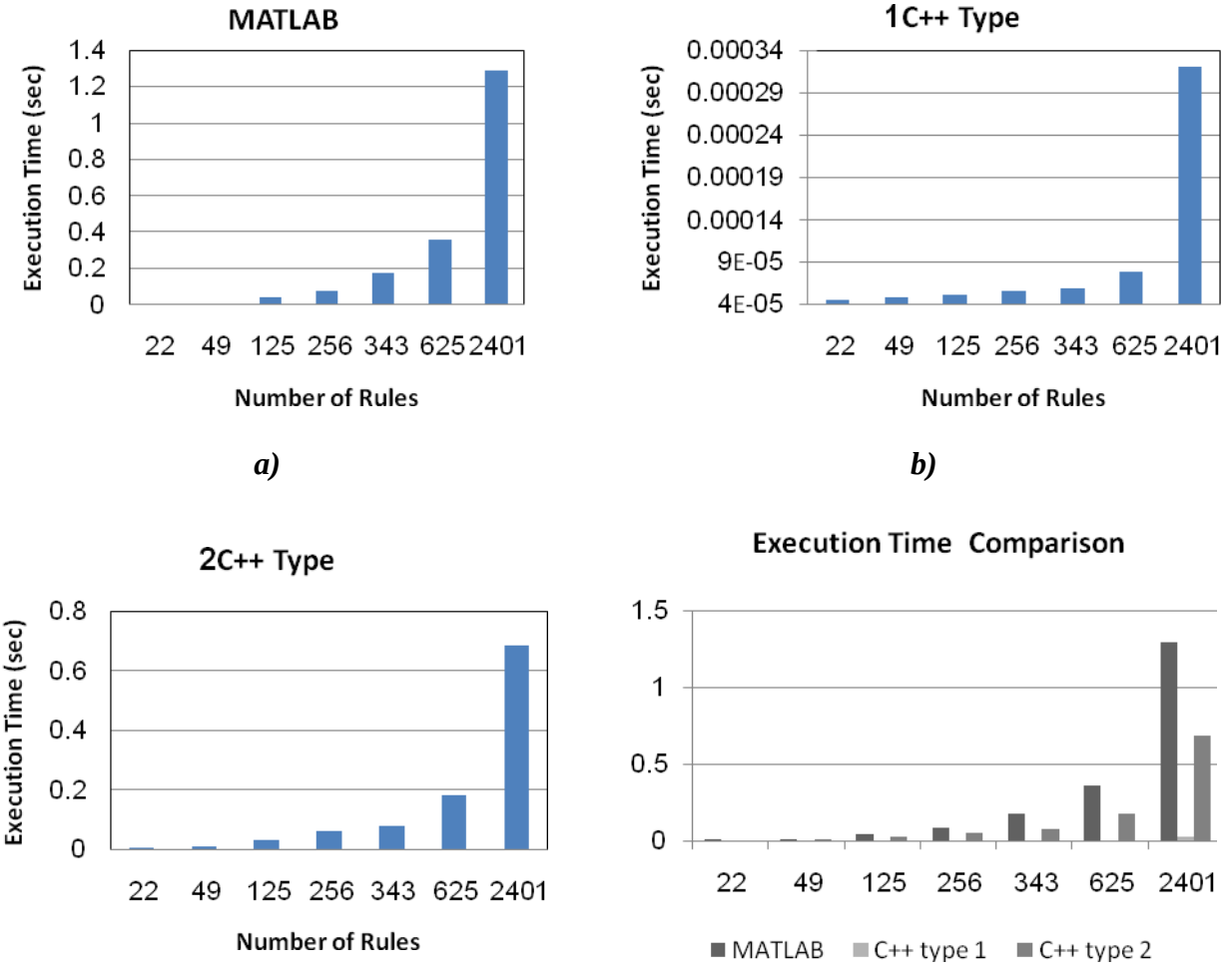


Fig. 5 Execution Time of ***a)*** MATLAB FLCs, ***b)*** C++ type 1 FLCs, ***c)*** C++ type 2 FLCs, ***d)*** comparison of execution time of three FLC programs.

C++ type 2 FLC program can handle complex systems with any number of rules but as mentioned before, the execution time will grows exponentially with increasing number of rules. A solution for the high execution time problem is to use a parallel environment and to design a parallel algorithm for FLC program.

Parallel Programming Environment

Computer systems are classified into four groups SISD, SIMD, MISD, and MIMD. Parallel FLC implemented in this work mimics a MIMD system. All processors execute the same program; the result referred to as single-program multiple data [10]. Parallel programming environment can be classified as multi-processors or multi-computers; each one has either bus or switch connection subdivision (**Fig. 6**) [12].

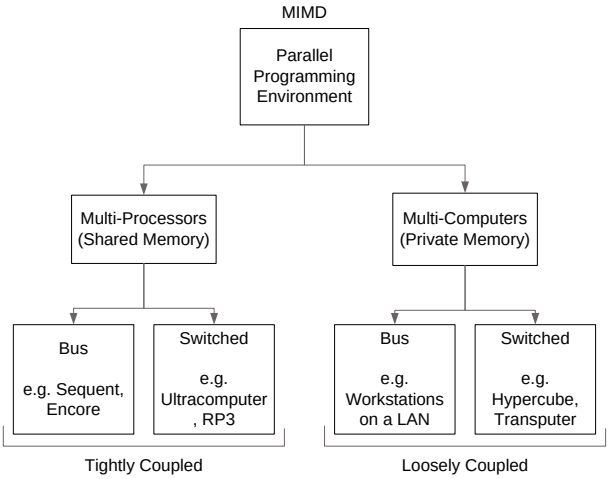


Fig. 6 Classification of parallel and distributed computers.

In this work the parallel programming environment consists of four computers connected in a LAN through a switch. All computers belong to the same workgroup called "WORKGROUP". The computers IP addresses are from 192.168.0.1 to 192.168.0.4. All computers have the same characteristics. This parallel programming environment is called a cluster. **Fig. 7** shows the physical organization of a cluster of four computers.

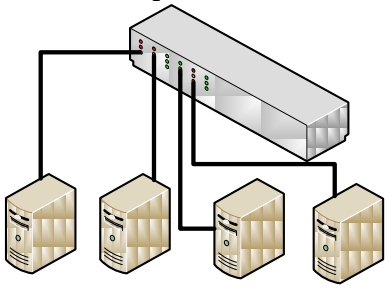


Fig. 7 Physical organization of the cluster.

For this cluster, software is needed to distribute the jobs, synchronize processes, send and receive data, and manage the cluster. MPICH2 [15] is used to execute a parallel program. MPICH2 is one of the most perfect implementation of MPI-2 standard by Argonne National Laboratory.

Parallel FLC Algorithm

To design the parallel FLC algorithm, the problem should be decomposed into more than one task that can be done concurrently on different computers. Most of computation time in a FLC program is due to the searching of rulebase for fired rules. Especially for a large rulebase file the time to open and search the file is large.

The reading and searching of the rulebase file operations can be decomposed, and more than one computer can handle these operations concurrently. To do so, the rulebase file should be divided into number of files equal to the number of computers in the parallel environment.

Each part of decomposed rulebase is stored in a computer and the operations of reading and searching this file are mapped to that computer. All computers in the cluster read their rulebases and search them for the fired rules concurrently during the execution of the program. They also aggregate the numerator and denominator of the output value based on the fired rules strength.

The input should be distributed to all computers in the cluster from the root computer. The root computer could be any computer in the cluster. It is the interface between the FLC and controlled plant, it also participate in problem solution. The inputs from the plant provided to the root computer then they will be distributed to all computers by the root computer. When all outputs in all computers are ready, they will be transferred to the root computer to calculate the final output. This final output will be fed to the plant by the root computer. The flowchart of the parallel FLC algorithm is shown in **Fig. 8**.

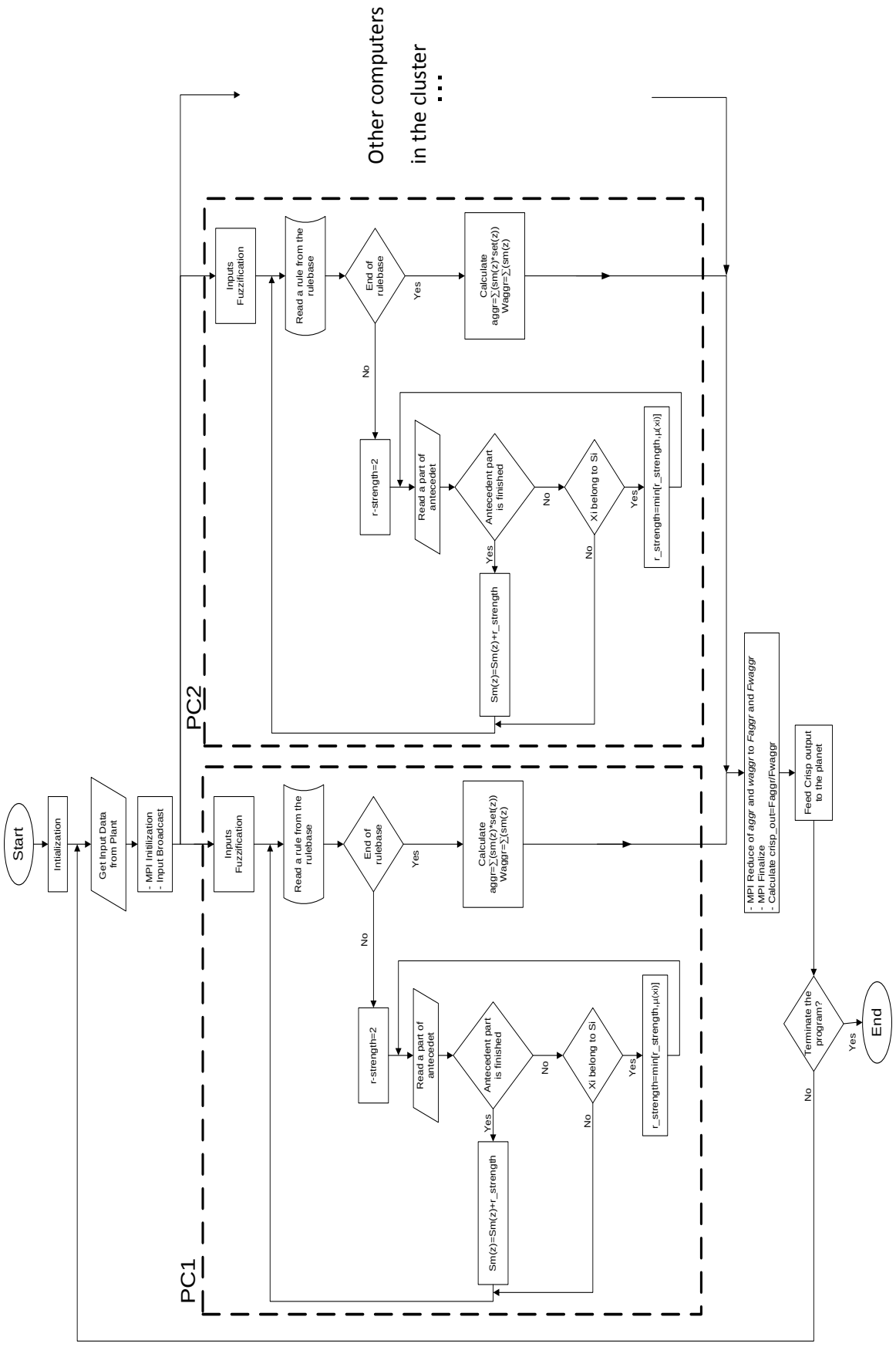


Fig. 8. Flowchart of the parallel FLC algorithm.

The FLC for servomotor is used here to evaluate the parallel FLC. The execution time of this FLC is shown in **Fig. 9**. It can be seen that using 4 computers increase the execution time in comparison with the employment of 3 computers. The servomotor FLC has 22 rules, executing of this controller on 4 computers will not reduce the execution time as much as the increase in communication time.

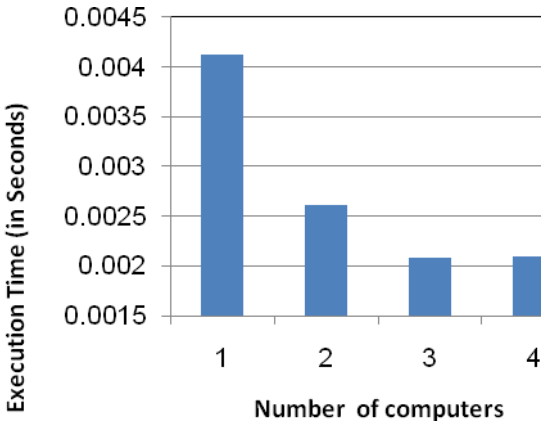
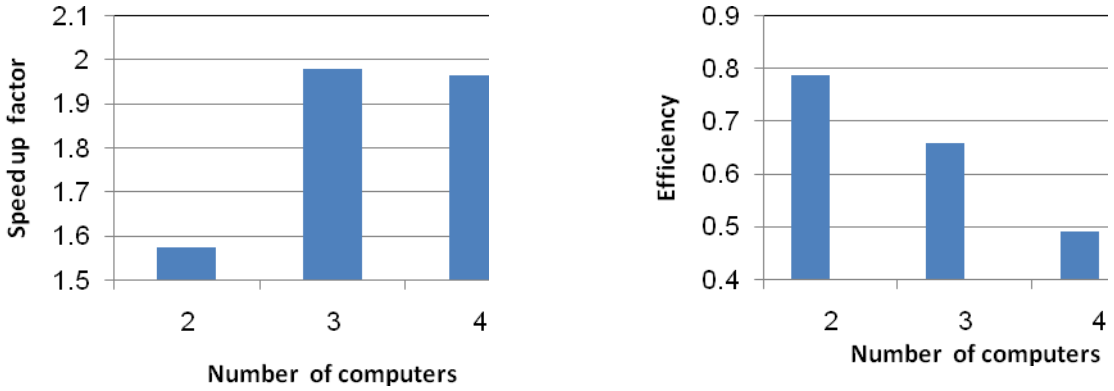


Fig. 9. Execution time of FLC for Servomotor.

The speed up factor and efficiency result of executing FLC for the servomotor using different number of computers in a cluster is shown in **Fig. 10**. It is obvious that maximum speedup for this simple FLC can be achieved by using three computers to solve the problem concurrently. Using more than 3 computers will increase the communication time without noticeable reduction in execution time.



a) ***b)***

Fig. 10. Parallel servomotor FLC ***a)*** speed up, ***b)*** Efficiency.

The maximum efficiency of FLC for servomotor can be achieved using 2 computers in a cluster to execute the parallel servomotor FLC program.

Antiskid Steering System (ASS) is another FLC system used to evaluate the parallel FLC implementation. ASS is one of the most complex fuzzy-logic embedded systems ever developed. It reduces the steering angle applied by the driver through the steering wheel to the amount the road can take. It optimizes the steering action and avoids sliding since a sliding car is very difficult to re-stabilize, especially for drivers not accustomed to such situations [16].

The execution time of the ASS FLC with 600 rules using different number of computers is shown in **Fig. 11**.

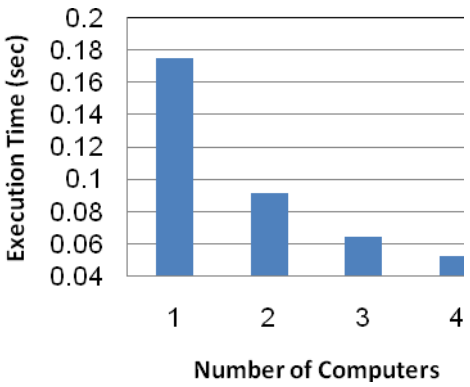
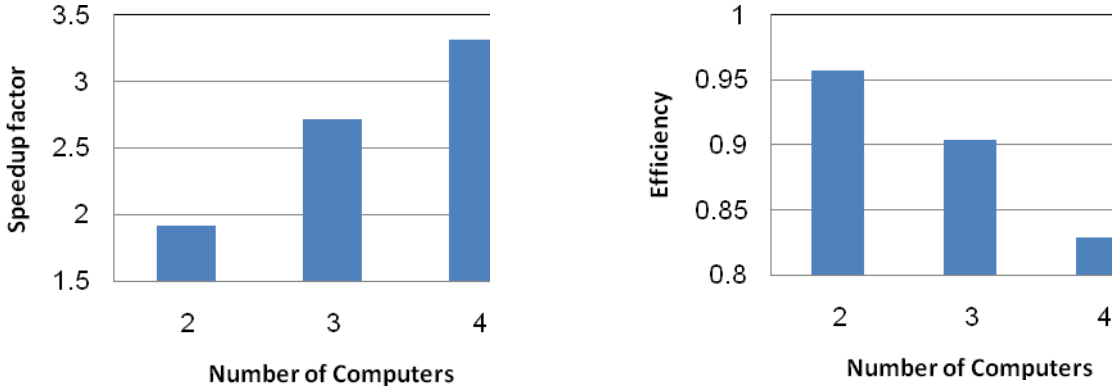


Fig. 11. Parallel ASS FLC execution time.

Fig. 12.a shows the speedup factor and **Fig. 12.b** shows the efficiency.

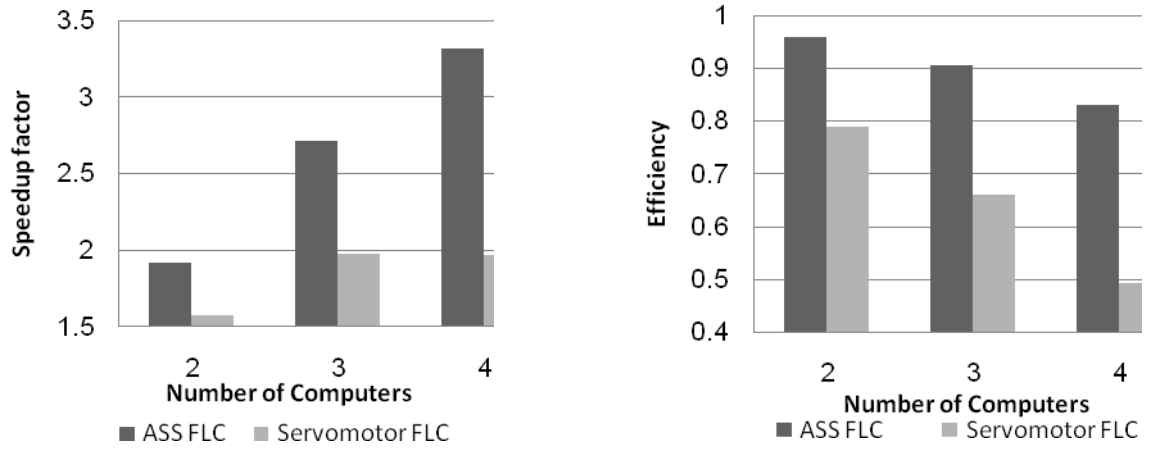
The result in **Fig. 12** shows that the maximum speedup is obtained when using 4 computers while the efficiency is minimum.



a) ***b)***

Fig. 12. Parallel ASS FLC ***a)*** Speedup, ***b)*** Efficiency.

Comparing servomotor FLC with ASS FLC it is obvious that the speedup factor and efficiency are higher in Parallel ASS FLC. **Fig. 13** shows the comparison between speedup and efficiency of servomotor and ASS FLCs.



a) ***b)***

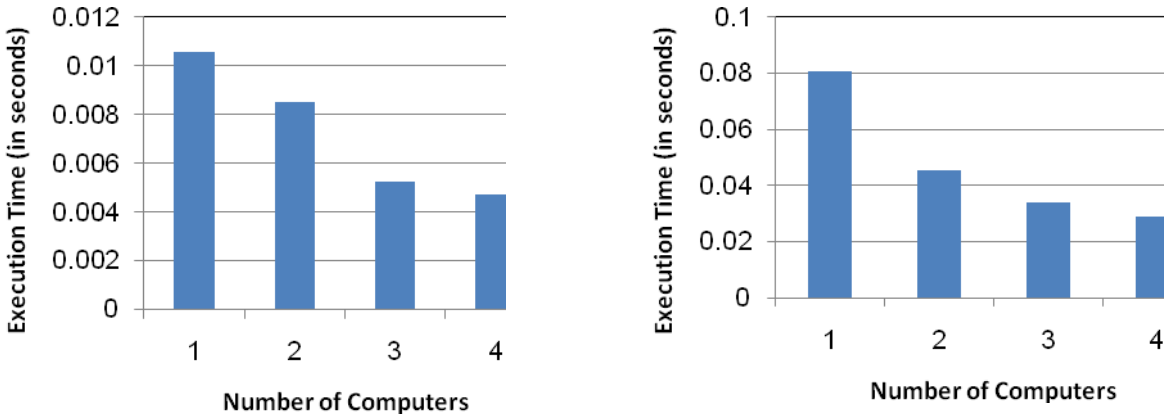
Fig. 13. ***a)*** Speedup, ***b)*** Efficiency; comparison between Servomotor and ASS FLCs.

Another four hypothetical FLC systems are used to evaluate the speedup and the efficiency of the parallel FLC program.

1- A FLC system with 2 inputs, 7 membership function, and 49 rules.
2- A FLC system with 3 inputs, 7 membership function, and 343 rules.
3- A FLC system with 4 inputs, 7 membership function, and 2401 rules.
4- A FLC system with 5 inputs, 7 membership function, and 16807 rules.

The above FLC systems will be used to calculate the execution time, speedup, and efficiency when executing on a cluster of 1, 2, 3, and 4 computers.

The execution time of these FLC systems are shown in **Fig. 14**.



a) ***b)***

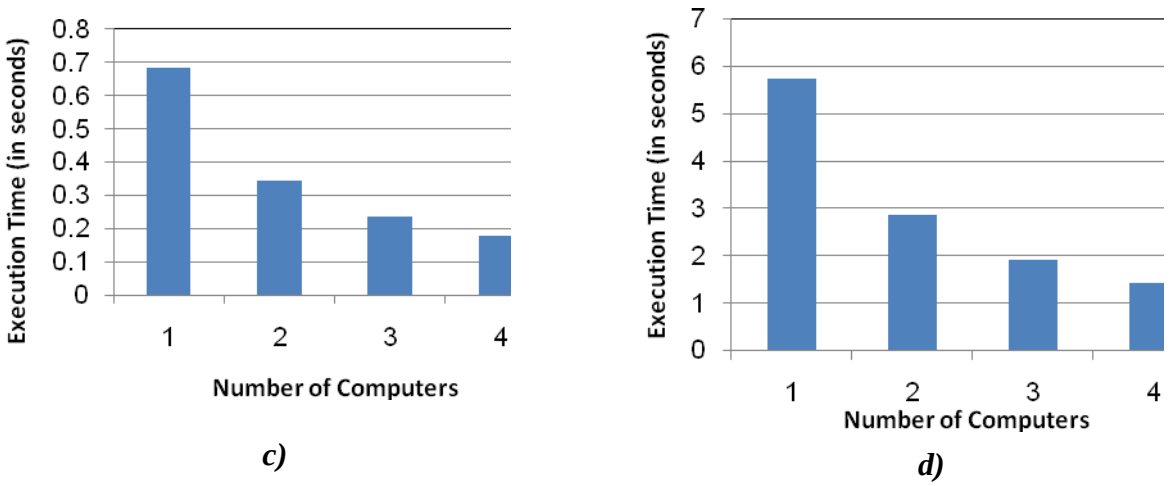


Fig. 14. Execution time of ***a)*** A FLC with 49 rules, ***b)*** A FLC with 343 rules, ***c)*** A FLC with 2401 rules, ***d)*** A FLC with 16807 rules.

To compare the effect of distributing these FLCs in a cluster the speed up factor must be calculated. The speedup factors are shown in **Fig. 15**. The efficiency values of the above FLCs are shown in **Fig. 16**.

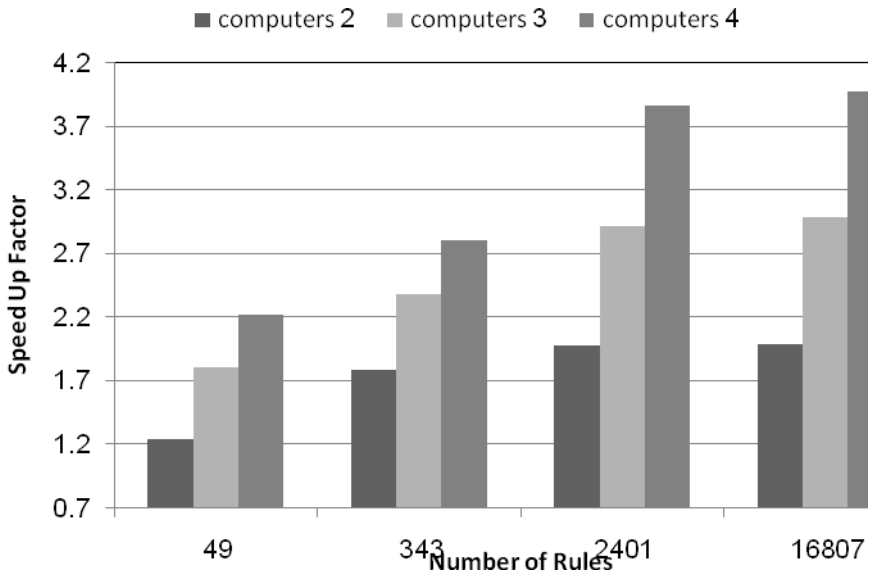


Fig. 15. The speedup Factors of various FLCs on clusters of different number of computers.

The results show that increasing in number of rules make the speedup factors becomes near to the number of computers. This means that for complex FLCs when speedup factor is close to its maximum amount, the communication time can be neglected. Also it is clear that for small FLCs increasing number of computers in the cluster will increase the communication time which will reduce the amount of speedup.

Fig.16 shows that the maximum efficiency is achieved for the most complex FLC (the one with maximum number of rules). Although increasing number of computers in the cluster increase the speedup, but in other hand it reduces the efficiency factor.

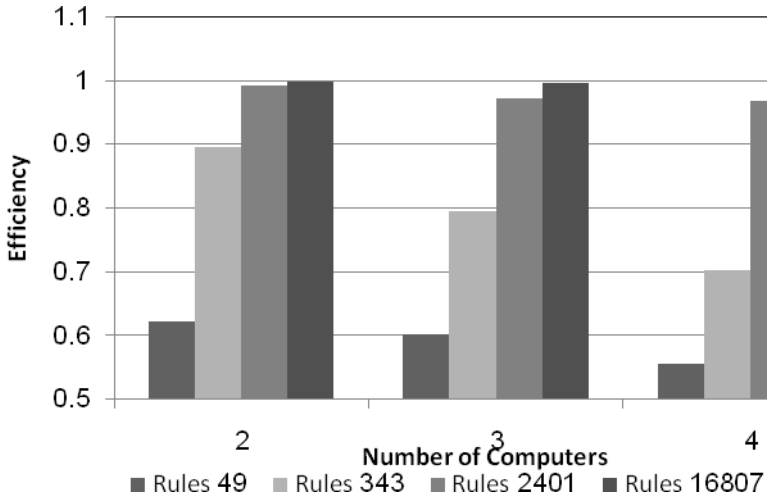


Fig. 16. Comparison of various FLC systems Efficiency.

The results show that choosing number of computers in the cluster that will participate in solution of the FLC problem has direct effect to speedup, efficiency, and communication time. So choosing the number of computers in the cluster is a tradeoff between speedup and efficiency.

CONCLUSIONS

In this research three types of FLC implementations, were introduced. The first one implemented with the rules stored inside the program. Comparing this implementation with MATLAB fuzzy toolbox implementation, shows that the former one has less execution time.

The experimental results show that above two implementations are not appropriate for complex and large FLCs, since both have rulebase size limitation.

The second FLC implementation with rules stored in a file on hard disk solves the memory problem of the previous implementations on the account of execution time. Rulebase searching time is growth by increasing number of rules. Complex FLC spend much time in searching the rulebase file for fired rules. This operation can be decomposed and mapped to more than a computer. Running the FLC program, on a parallel environment, decrease the execution time especially for complex FLCs.

The third FLC implementation is based on decomposing the second FLC implementation to be executed on more than a computer. The experimental results shows that increasing number of computers, decreases the execution time until a point that because of the communication overhead, the execution time will not be decreased anymore. For simple FLCs this point is reached using few computers. But for complex FLCs with large number of rules, using more computers in the cluster can decrease the execution time.

Although adding more computers in the cluster increase the speedup factor but in the other hand, the efficiency will be decreased. Choosing number of computers in the cluster to solve a FLC program is a tradeoff between speedup and efficiency.

REFERENCES

- [1] C. Dualibe, M. Verleysen, P.G.A. Jespers, Design of Analog Fuzzy Logic Controllers in CMOS Technologies Implementation, Test and Application, Kluwer Academic Publishers, 2003.
- [2] I. Foster, Argonne National Laboratory, Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering, Addison Wesley, 1995.
- [3] A. Gupta, C. Forgy, A. Newell, and R. Wedig, Parallel Algorithms and Architectures for Rule-Based Systems, Vol. 14 , Issue 2 , Proceedings of the 13th annual international symposium on Computer architecture (ISCA '86), P 28-37,ACM, 1986.
- [4] N. Howard, R. Taylor, N. Allinson, The Design and Implementation of a Massively-Parallel Fuzzy Architecture, Fuzzy Systems, IEEE International Conference, P552-554, IEEE Trans., 1992.
- [5] M.J. Lees, D. A. Campbell, J. C. Devlin, A Reconfigurable Parallel Inference Processor for High Speed Fuzzy Systems, Circuits and Systems, ISCAS '96., IEEE International Symposium,Vol. 3, P 539-542, 1996.
- [6] L.A. Zadeh, "Fuzzy Logic Systems, Origin, Concepts, and Trends", ALTANA Chair for Applied Computer Science, 2004.
- [7] Z. Kovačić, S. Bogdan, Fuzzy Controller Design Theory and Applications, Published in, CRC Press Taylor & Francis Group, 2006.
- [8] K. H. Lee, First Course on Fuzzy Theory and Applications, Springer, 2005

- [9] S.G. Lee, H.H. Lee, M. Miyazaki, K. Akizuki, Parallel Fuzzy Inference on Hypercube Computer, Fuzzy Systems Conference Proceedings, FUZZ-IEEE '99. IEEE International, Vol. 1, P 309-314, 1999.
- [10] B. Parhami, Introduction to Parallel Processing Algorithms and Architectures, Kluwer Academic Publishers, 2002.
- [11] A. Grama, A. Gupta, G. Karypis, V. Kumar, Introduction to Parallel Computing, Second Edition, Addison Wesley, 2003.
- [12] R.S. Morrison, Cluster Computing Architectures, Operating Systems, Parallel Processing & Programming Languages, 2003
- [13] B. Giacalone, M.L. Presti, F.D. Marco, Hardware Implementation Versus Software Emulation of Fuzzy Algorithms in Real Applications, Fuzzy Systems Proceedings, IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference, Vol. 1, P 7-12, 1998.
- [14] http://support.microsoft.com/ph/3040, Visual Studio .NET 2003 solution center, Microsoft, June 13, 2008.
- [15] http://www.mpi-forum.org, The official Standard documents for MPI-1 and MPI-2, Message Passing Interface Forum, April 22, 2008.
- [16] C.v. Altrock, Fuzzy Logic in Automotive Engineering,Issue 88, Circuit Cellar INK, 1997.

LIST OF SYMBOLS AND ABBREVIATIONS

ASS	Antiskid Steering System
CPI	Cycles Per Instruction
DMMP	Distributed Memory Message Passing
DMSV	Distributed Memory Shared Variables
E_p	Efficiency Factor
FL	Fuzzy Logic
FLC	Fuzzy Logic Controller
FLOPS	Floating Point Operations Per Second
GMMP	Global Memory Message Passing
GMSV	Global Memory Shared Variables
MIMD	Multiple Instruction Multiple Data
MIPS	Million Instructions Per Second
MISD	Multiple Instruction Single Data
MPICH2	Message Passing Interface CHameleon version 2
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
S_p	Speedup Factor