Towards Automatic Load Balancing for Programming Parallel Fuzzy Expert Systems in Heterogeneous Clusters

Chao-Chin Wu, Lien-Fu Lai, Yu-Shuo Chang
Department of Computer Science and Information Engineering
National Changhua University of Education, Taiwan
ccwu@mail.ncue.edu.tw, lflai@mail.ncue.edu.tw, asouchang@gmail.com

Abstract*-FuzzyCLIPS is a rule-based language for developing the fuzzy expert systems. Due to the characteristics of rule-based languages, the execution of the system would be more time-consuming than most conventional algorithmic languages. To address this problem, we propose to execute a FuzzyCLIPS application in parallel on the emerging heterogeneous cluster system. Furthermore, to maximize the speedup of the parallel execution and to minimize the burden of programmers, we have implemented built-in self-scheduling schemes in the interpreter for better load balancing. The developer only needs to use our proposed directives to specify where the parallelisms are, no explicit and complicated send and receive routines have to be invoked in the parallel program. Experimental results show that the built-in load balancing schemes can improve the system performance significantly.*

Keywords: Expert System, Cluster System, FuzzyCLIPS, Load Balancing, Self Scheduling.

1. Introduction

Conventional computer programs generally solve problems having algorithmic solutions. Algorithmic languages include C, Java, and VB. On the contrary, for non-algorithmic languages, the programmer does not give exact details on how the program is to be solved. Instead, the programmer only has to specify the goal. The underlying mechanism of the implementation itself tries to satisfy the goal. This kind of languages includes LISP, Prolog, and FuzzyCLIPS. Because the key feature of expert systems is to solve problems for which there are no known algorithms, the latter kind of language is more suitable for building expert systems. Among these languages, FuzzyCLIPS has been widely adopted because it is a multiparadigm programming language that provides support for rule-based programming, object-oriented programming, and procedural programming [1].

FuzzyCLIPS consists of three components: facts, rules, and an inference engine, where the rules form the knowledge base. To solve a problem, FuzzyCLIPS must have data or information with which to reason. Each chunk of information is called a fact. By matching unknown facts with the rules, the inference engine draws conclusions from the knowledge base. If all the patterns of a rule match facts, the rule is activated and put on the agenda. The activated rule with the highest priority in the agenda will be selected and executed repeatedly until the agenda becomes empty.

A lot of fuzzy expert systems have been constructed for solving various complex problems [2-5]. However, it is very time-consuming to run the rule-based applications. To address this problem, we propose to use the MPICH Library [6] to parallelize the execution of a FuzzyCLIPS application on cluster systems. Meanwhile, we hope that the programmers can parallelize their programs without the need of understanding the complicated parallel programming skills. To achieve this goal, we define several easy-use directives for parallelization. The FuzzyCLIPS programmers only need to use these directives to specify which facts can be inferred with which rules in parallel. Accordingly, the master process will dispatch these facts to the slave processes. Furthermore, the number of facts will be processed by a slave process depends on its dynamic computing power at runtime. The master process employs the concept of self-scheduling schemes to dispatch facts to slaves for better load balancing [7]. Therefore, the proposed programming model is easy to parallelize the FuzzyCLIPS application and the application can be executed more efficiently by the built-in load balancing mechanism.

To verify our proposed approach, the search engine of a human resources website has been parallelized and evaluated on a heterogeneous cluster system. The search engine is written in FuzzyCLIPS, and it performs the fuzzy logical

evaluations based on the user's query to find the satisfied records with different levels of fitness. In a sequential version, if the number of records exceeds 6000, the end-user has to wait for more than two minutes to get the result. After applying our proposed approach, even if the number of records is equal to 10000, the response time is only 16 seconds. Therefore, our approach is very easy to parallelize FuzzyCLIPS applications and the speedup can be superlinear.

2. Related Work

FuzzyCLIPS is an extended version of CLIPS (C Language Integrated Production System) [8] that is a tool for helping the developer to design the expert system [9]. FuzzyCLIPS extends CLIPS by adding the concept of fuzzy logic, i.e. fuzziness and uncertainty. The extension let the FuzzyCLIPS inference engine be able to do the inference with the facts and rules with fuzzy expressions.

Due to the rule-based characteristics of CLIPS, it makes the execution very time-consuming. Several research projects have proposed different solutions to address this problem [10-13]. However, these solutions were all based on some specific platforms, such as shared memory, Intel Hypercubes, distributed system, and PVM (Parallel Virtual Machine). All of them were based on the MPMD programming model [14] and none of them adopted the MPICH Library for parallel executions on the cluster system.

In our previous researches, we have proposed two methods to parallelize the CLIPS-based expert systems. They adopted SPMD programming model for easy maintenance [14]. In the ECS (Embedded Subroutine Calls) method, we separate the facts and the rules into two parts [15]. The facts are embedded into the C-based inference engine and the rules are left in the CLIPS file. The MPICH Library is used in the inference engine to exchange these facts between different computing nodes. After receiving the facts, the slave process will use the built-in subroutines to insert the facts into the local knowledge base. Finally, the facts are matched with the rules by the inference engine to draw the conclusion.

In the EFD (External Function Definitions) method, we use the interface of the External Function Definition provided by the CLIPS interpreter to extend the languages [16, 17]. In this way parallel syntax can be defined based on the original CLIPS programming style. As a result, the proficient CLIPS programmers have no need to learn the C language to develop parallel expert systems. They only need to learn the basic parallel programming knowledge and use the simple CLIPS-style parallel routines such as *send*, *receive*, and *synchronization* to develop parallel CLIPS applications. The supported parallel syntaxes are simplified for the CLISP language by implementing various complicated message passing mechanisms with the MPICH Library in the interpreter.

3. The proposed approach

In this section, we propose a programming model that allows programmers to write parallel FuzzyCLIPS applications in an easy way. No explicit send and receive routines are invoked in the program. The extended FuzzyCLIPS interpreter will execute the application in parallel automatically. In addition, because of the heterogeneity of the cluster system, each slave process will be assigned different chunk of tasks for better load balancing.

3.1. The main idea

If a FuzzyCLIPS expert wants to execute his parallel program more efficiently on a heterogeneous cluster system, he has to write additional codes for load balancing. However, it is preferred that the expert system developers do not need to care about the coding of load balancing. Hence, we propose a SPMD-based programming model that hides message passing subroutine calls from the programmers. In other words, there are no *Send* or *Recv* function calls needed in the parallel code of a fuzzyCLIPS expert system. Instead, several simple directives are employed for parallelization. The programmers only have to figure out the following information. (1) Which facts will be processed by slave processes in parallel? These facts are called the *parallel facts*. (2) What kinds of facts should be sent back after each slave process finishes its work? These facts are called the *returned facts*. (3) Which rules will be applied to the parallel facts? These rules are called the *applied rules*. (4) Which rules the master process has to apply to the returned facts from slave processes? These rules are called the *reduce rules*.

Several directives are designed to specify the first two kinds of the above information in a SPMD-based FuzzyCLIPS application code. In addition, the applied rules and the reduce rules are specified by asserting particular facts to the left-hand-side of the rules. At runtime, a master process and multiple slave processes will be created from the code as shown in Figure 1. First, the master process would call the preprocessor to

analyze these directives, called FuzzyCLIPS Parallelism Directives, to identify the parallel facts, and the returned facts. The parallel facts are identified and stored into the memory of the master process. The type names of the facts which should be returned are stored by slave processes. Next, the master process will perform some self scheduling algorithm to assign parallel facts to slave processes. On the other hand, the slave processes will match the assigned parallel facts with the applied rules and send all the asserted returned facts back to the master process. Finally, the master process will perform the reduced operations on the returned facts after all the parallel facts have been processed.

In the following two sections, we will introduce the FuzzyCLIPS Parallelism Directives and the way of identifying the applied rules and the reduce rules.

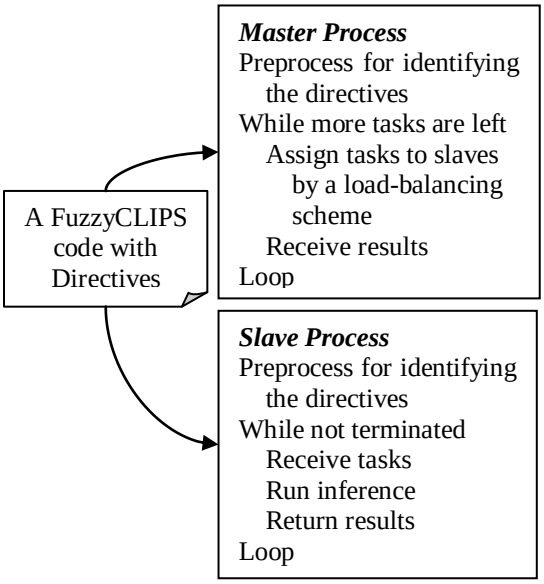

Figure 1. A SPMD-based FuzzyCLIPS code with directives

3.2. FuzzyCLIPS Parallelism Directives

The main function of the directives is to identify the parallel facts and the returned facts. Each directive begins with ";#". We define three directives: *";#TASK_BEGIN"*, *";#TASK_END"*, and *";#RETURN"*. The first two directives are a pair of directives used together to enclose all the parallel facts between them. The directive *";#TASK_BEGIN"* is placed at the beginning and the directive *";#TASK_END"* is placed at the end. The third directive *";#RETURN"* is for specifying the returned facts. However, because the contents of the returned facts cannot be known until the runtime, the type name of the returned facts is adopted as the argument of the directive *";#RETURN"*. Accordingly, a slave process will send all the facts of the specified type names in the directive *";#RETURN"* back to the master process. In this way, the programming can be simplified subtaintially.

Backus-Naur Form and Action Symbols

<CMD>→DIRECTIVE <TYPE>
<TYPE>→KEY(TASK_BEGIN)<TASKS>
<END>
<TYPE>→RETURN WORD #addRtn
{WORD #addRtn}
<END>→DIRECTIVE KEY(TASK_END)
<TASKS>→FACT #addFact
{FACT #addFact}

Functions of Action Symbols

#addRtn	To store the type name of the returned facts.
#addFact	To store a fact belonging to the parallel facts.

Token Types and Definitions

DIRECTIVE	Begin with a semicolon and a number sign, i.e., *";#"*.
FACT	Begin with a semicolon and a CLIPS fact followed. E.g. *;(SAMPLE (SLOT 1))*
COMMENT	A line beginning with a semicolon.
WORD	The regular expression is $(\alpha\|\eta)^+$, where α means an alphabet and η means a number.
KEY(…)	It is a keyword. The keyword must one of words listed in the parenthesis.

Figure 2. The BNF, action symbols and token types of our preprocessor

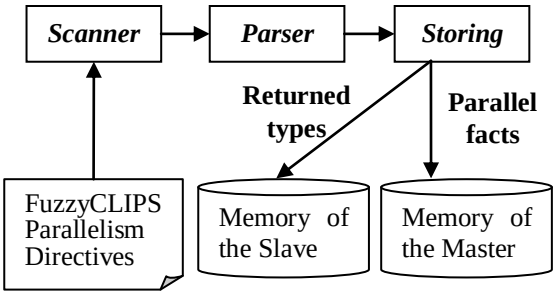

Figure 3. Preprocessor architecture

The Backus-Naur Form with the action symbols of our preprocessor and the token type are shown in Figure 2. Our preprocessor is an interpreter, as shown in Figure 3, derived from *LL(1)* parser [5]. The entry point of the preprocessor is the parser

routine. The parser routine calls the scanner routine to get tokens, and then checks the syntax correctness. Next, it passes the necessary information to the storing routine that is responsible for storing the information or data specified by the directives.

3.3. Identification of the rules

For a SPMD-based FuzzyCLIPS program, the rules that the master process and the slave processes should perform are all written in the same file. Because there are no *if-then-else* statements in FuzzyCLIPS, the programmers cannot divide the rules into two partitions by control statements. Hence, we propose to use the following fact template to specify if a rule will be executed by the master process or by the slave process.

(deftemplate MPI (slot TYPE))

The above template consists of only one slot, the *TYPE* slot. Based on the template, the fact *(MPI (TYPE MASTER))* indicates the associated rule will be executed by the master process and the fact *(MPI (TYPE SLAVE))* denotes the associated rule will be executed by the slave process. Therefore, the fact *(MPI (TYPE SLAVE))* is inserted into the left hand side of every applied rule and the fact *(MPI (TYPE MASTER))* is added into the left hand side of the reduce rules.

3.4. A simple example

Figure 4 shows a complete example code by using our proposed programming model. The main function of the program is to find the unusual weather of July in Changhua City during the last ten years. The *data* facts, ranging from Line 2 to Line 5, record the information of weather of July for each year in Changhua. They are the parallel facts that will be processed by the slave processes. Therefore, they are enclosed between the paired directives: "*;#TASK_BEGIN*" and "*;#TASK_END*". The "*;#RETURN*" directive followed by the type name of *unusual*, listed on Line 7, indicates that the slave process will return the facts of the type name of *unusual*. The *find-unusual-weather* rule, listed from Line 13 to Line 18, is the applied rule with the fact *(MPI (TYPE SLAVE))* on its left hand side. The action *(assert (unusual (year ?y) (month July)))* on the right hand side of the rule will assert all the *unusual* facts that will be sent back to the master process. Finally, the *printout-result* rule, listed from Line 19 to Line 23, is the reduce rule with the fact *(MPI (TYPE MASTER))* on its left hand side. After the master process receives the returned *unusual* fact, it will perform the rule.

As we can see, there are no explicit message passing between the master and the slaves in the code. However, at the runtime, the master will dispatch the *data* facts to the slaves by the interpreter based on the specified load-balancing scheduling scheme. The number of facts to be assigned to a slave depends on the computing power of the slave at the runtime. Therefore, the programming model is very easy for the FuzzyCLIPS experts to develop parallel FuzzyCLIPS applications.

```
1.  ;#TASK_BEGIN
2.  ;(data (year 1998) (month July) (location
    Changhua) (average-temperature (30.7 0)
    (30.7 1) (30.7 0)) (total-rainfall (261.3 0)
    (261.3 1) (261.3 0)))
3.  ;(data (year 1999) (month July) (location
     Changhua) (average-temperature (29.2 0)
     (29.2 1) (29.2 0)) (total-rainfall (174.1 0)
     (174.1 1) (174.1 0)))
4.  ……
5.  ;(data (year 2007) (month July) (location
    Changhua) (average-temperature (31.2 0)
    (31.2 1) (31.2 0)) (total-rainfall (252.1 0)
    (252.1 1) (252.1 0)))
6.  ;#TASK_END
7.  ;#RETURN unusual
8.  (deftemplate MPI (slot TYPE))
9.  (deftemplate temperature
    0 100 Celsius
    ((cold (z 10 25))(warm (pi 4 26))(hot (s 27
    32))))
10. (deftemplate rainfall
    0 1000 mm
    ((light (z 100 160))
    (medium (pi 30 180))
    (heavy (s 200 260))))
11. (deftemplate data
    (slot year)(slot month)(slot location)
    (slot average-temperature (type
    FUZZY-VALUE temperature))
    (slot total-rainfall (type FUZZY-VALUE
    rainfall)))
12. (deftemplate unusual (slot year)(slot month))
13. (defrule find-unusual-weather
14. (MPI (TYPE SLAVE))
15. (data (year ?y) (month July) (location
    Changhua) (average-temperature very hot)
16. (total-rainfall very heavy))
17. =>
18. (assert (unusual (year ?y) (month July)))
19. (defrule printout-result
20. (MPI (TYPE MASTER))
21. ?f <- (unusual (year ?y) (month July))
22. =>
23. (printout t "The weather is unuasal at July, " ?y
    crlf))
```

Figure 4. An example of parallel expert system with our proposed model

4. Experimental Results

A human resources Web site has been implemented to evaluate the performance. Because users' query requirements are usually imprecise and uncertain, instead of matching the input phrases with the records in the database, the search engine uses fuzzy logic to find the records with different levels of fitness. If the search engine is powered by a single processor, the response time will be too long to tolerate. When the number of records is 8000, the user has to wait for about four minutes before the results are displayed as shown in Figure 5. As the number of records is enlarged, the response time is increased substantially. To address the problem, we employ the proposed programming model to parallelize the search engine and construct a cluster system with 16 processor cores as shown in Table 1.

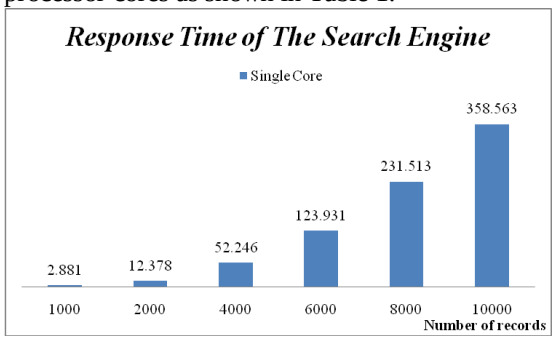

Figure 5. The response time (seconds) of the system

We compare the performance improvements of different load-balancing schemes by using 8 processor cores as shown in Figure 6. The performance improvement is derived from dividing the parallel execution time by the sequential execution time. Therefore, the label of *Single Core* represents the performance improvement by using the Intel Core 2 Duo processor only. The label of *Static Scheme* represents the improvement by dispatching equal-sized chunk of parallel facts to every slave process. The label of *CSS(x)* denotes the improvement by dispatching facts based on Chunk Self-Scheduling (CSS) scheme with the chunk size equal to *x*. No matter which kind of scheduling schemes is adopted for parallelization, our approach provides superior performance improvements. If the number of records is enlarged, the improvements are also increased. Moreover, the dynamic load balancing scheme is better than the static scheme. On the other hand, the chunk size influences the performance. Increasing the chuck size does not necessarily improve the performance. The appropriate chunk size depends on the number of records.

Table 1. The configuration of our cluster system

Intel Core 2 Duo (2 Duo-core PCs)	
CPU	Intel Core 2 Duo E2160, 1809Mhz
Memory	512MB DDR2-667 ×1
Swap	1024MB
HD	SATA 80GB
OS	CentOS 4.4, kernel 2.6.9-42.ELsmp
AMD Athlon @1250 (1 Single-core PC)	
CPU	AMD Athlon XP 2800+, 1243Mhz
Memory	256MB DDR-400 ×1
Swap	2048MB
HD	ATA133 80GB
OS	Slackware 12.0, kernel 2.6.21.5-smp
AMD Athlon @2000 (3 Single-core PCs)	
CPU	AMD Athlon XP 2800+, 1243Mhz
Memory	256MB DDR-400 ×1
Swap	2048MB
HD	ATA133 80GB
OS	Slackware 12.0, kernel 2.6.21.5-smp
Intel Core 2 Quad (2 Quad-core PCs)	
CPU	Intel Core 2 Quad, 2394Mhz
Memory	2048MB DDR2-533 ×1
Swap	1024MB
HD	SATA 160GB
OS	Slackware 12.1, kernel 2.6.24.5-smp

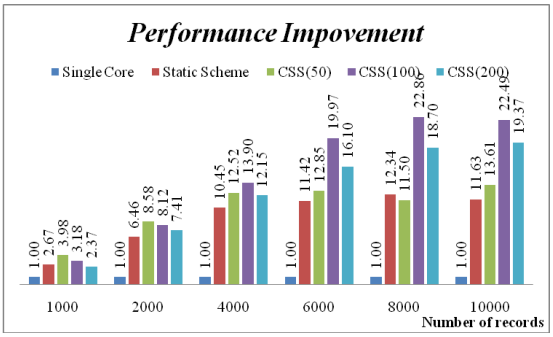

Figure 6. The performance comparison of different load balancing schemes by using 8 processor cores

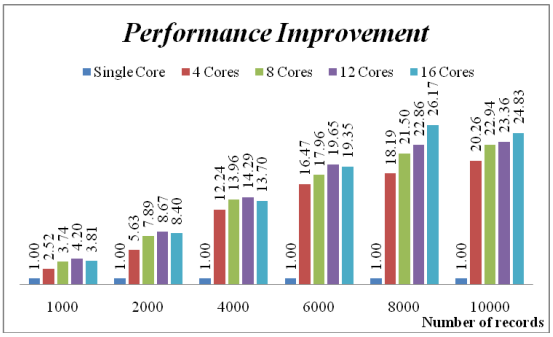

Figure 7. The performance improvements by using different number of processor cores

Next, we compare the performance improvements by using different number of processor cores as shown in Figure 7. The *CSS(100)* scheme is adopted for all parallel executions. Increasing the number of processor cores used can lead to performance improvement only when the number of records is larger enough. In this experiment, the number of records should be larger than 6000 for scalable performance improvements. Furthermore, our approach can provide superlinear speedups if the number of records is larger than 6000. The best performance speedup is 26.17 even though the search engine is parallelized by only 16 processor cores.

5. Conclusions and future work

In this paper, we have proposed a dynamic load balancing programming model for the parallel FuzzyCLIPS application. The programmers do not need to write any code for load balancing. Instead, with our proposed directives, they only need to indicate (1) which facts can be executed in parallel, (2) which asserted facts should be sent back to the master, (3) what the slave process should do, and (4) what are the reduce operations after the master receives the returned facts. Accordingly, the interpreter will assign the appropriate chunk of facts to each slave depending on the individual computing power, resulting in better load balancing. Therefore, it is very easy for programmers to write parallel FuzzyCLIPS applications with load balancing capacities. Experimental results show our method can improve the system performance with superlinear speedups.

In the future, we will extend the FuzzyCLIPS Parallelism Directives to support more complex applications. In addition, more advanced load balancing schemes will be developed based on the dynamic characteristics of grid systems.

Acknowledgement

This research is supported by the National Science Council of Taiwan under contract number: NSC97-2815-C-018-011-E.

References

[1] FuzzyCLIPS, 2007, FuzzyCLIPS, Available, http://www.iit.nrc.ca/IR_public/fuzzy/fuzzyClips/fuzzyCLIPSIndex2.html

[2] A. Iqbal, N. He, and L. Li, "A fuzzy expert system for optimization of high-speed milling process", in *Proceedings of the Fifth International Workshop on Robot Motion and Control*, pp. 297-304, 2005.

[3] N. Allahverdi, S. Torun, and I. Saritas, "Design of a fuzzy expert system for determination of coronary heart disease risk," in *Proceedings of the 2007 international conference on Computer systems and technologies*, Article No. 36, 2007, Bulgaria.

[4] I. Saritas, N. Etik, N. Allahverdi, and I.U. Sert, "Fuzzy expert system design for operating room air-condition control systems," in *Proceedings of the 2007 international conference on Computer systems and technologies*, Article No. 23, 2007, Bulgaria.

[5] C.A. Reyes-Garcia, and E. Corona, "Implementing Fuzzy Expert System for intelligent buildings," in *Proceedings of the 2003 ACM symposium on Applied computing*, pp. 9-13, Mar. 9-12, 2003, Melbourne, Florida, USA.

[6] MPICH, 2008, MPICH Home Page, Available: http://www-unix.mcs.anl.gov/mpi/mpich1/

[7] C.-T. Yang, K.-W. Cheng, K.-C. Li, "An Enhanced Parallel Loop Self-Scheduling Scheme for Cluster Environments", *Journal of Supercomputing*, Vol. 34, No. 3, pp. 315–335, 2005.

[8] CLIPS, 2007, CLIPS: A Tool for Building Expert Systems, Available, http://clipsrules.sourceforge.net/

[9] J.C. Giarratano, and G.D. Riley. *Expert Systems: Principles and Programming*, Thomson Course technology, Taiwan, 2005.

[10] G.D. Riley, "Implementing clips on a parallel computer," in *Proceedings of SOAR'87*, NASA/Johnson Space Center, 1987.

[11] L. Hall, B.H. Bennett, and I. Tello, "Pclips: Parallel clips," in *Proceedings of Clips'94*, pp. 307-314, 1994.

[12] D. Gagne, and A. Garant, "Dai-clips: Distributed, asynchronous, interacting clips," in *Proceedings of Clips'94*, pp. 297-306, 1994.

[13] L. Myers, and K. Pohl, "Using pvm to host clips in distributed environments," in *Proceedings of Clips'94*, pp. 177-186, 1994.

[14]B. Wilkinson, and M. Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*, Second Edition, Prentice Hall Publisher, 2005.

[15] C.C. Wu, L.F. Lai, K.C. Lai, W.C. Chang, and S.S. Jhan, "Parallelizing Expert Systems with CLIPS Language for Grid Systems," in *Proceedings of the 4th Workshop on Grid Technologies and Applications (WoGTA'07)*, pp. 125-131, Dec. 13-14, 2007, Providence University, Taiwan.

[16] C.C. Wu, L.F. Lai, and Y.S. Chang, "Developing SPMD-based CLIPS Applications by Using External Function Definitions for Grid Systems," in *Proceedings of 2008 International Conference on Advanced Information Technologies (AIT 2008)*, pp. 107, Apr. 25-26, 2008, Chao Yang University of Technology, Taiwan.

[17] C.C. Wu, L.F. Lai, and Y.S. Chang, "Using MPI to Execute a FuzzyCLIPS Application in Parallel in Heterogeneous Computing Systems," in *Proceedings of IEEE 8th International Conference on Computer and Information Technology (IEEE CIT 2008)*, pp. 279-284, Jul. 8-11, 2008, Sydney, Australia.