Developing SPMD-based CLIPS Applications by Using External Function Definitions for Grid Systems

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

Abstract

CLIPS is a rule-based language designed to help construct expert systems more easily. However, because of the language characteristics, it is very time-consuming to execute a CLIPS application when compared with other conventional languages. To address the problem, several parallel execution approaches have been studied. These approaches were all on specific platforms or based on MPMD Model. To our best knowledge, none of them focus on cluster or grid systems. Thus, in this paper, we propose an approach to parallelizing the CLIPS application for cluster and grid system. On the other hand, to maintain the CLIPS application program in an easier way, we propose to adopt the SPMD Model for programming. In our preliminary study, our approach can garner significant improvement. Therefore, it is very suitable to execute the CLIPS application in parallel on cluster and grid systems.

Keywords: Expert System, CLIPS, Parallel Computing, Cluster System, Grid System

1. Introduction

CLIPS is a rule-based language used to build an expert system [1]. Due to the characteristics of the CLIPS language, it is easier to construct an expert system by using CLIPS than other conventional languages. But this kind of programming language suffers from longer execution time than conventional languages. To reveal the phenomenon more concretely by the execution time, the *Cellular Automata* written in CLIPS language are evaluated with several different problem sizes.

Cellular Automata is a discrete model in computation theory. It consists of $n \times n$ grids, and each grid contains exactly a cell inside. A cell has two states: alive and dead. The next state of each cell depends on the current states of its neighbors. Usually a cell has eight neighbors. We have run the CLIPS-based Cellular Automata program with different grid sizes on a desktop PC and the execution times are shown in Figure 1. The execution time increases significantly when the dimension grows. It is so time-consuming that some previous researches suggested executing CLIPS applications in parallel to mitigate the problem. Nowadays, cluster and grid systems have been evolved into the framework for high performance computation. As far as we know, there is no previous work focusing on how to parallelize CLIPS application for cluster or grid systems. Thus, in this paper, we will propose an approach on this issue.

In previous researches [6-9], they parallelized the CLIPS programs for some specific platforms based on MPMD (Multiple Program Multiple Data) programming model. In this paper, we would parallelize the CLIPS applications with MPI library for cluster and grid systems based on SPMD (Single Program Multiple Data) programming model. SPMD programming model is easier to maintain and develop CLIPS applications than MPMD.

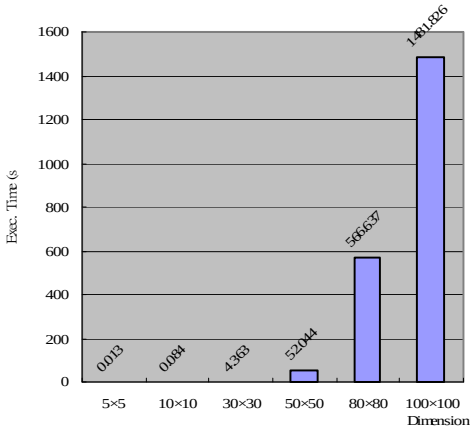

Figure 1. Execution time of the CLIPS-based Cellular Automata running on a PC

2. Related Works

2.1. Expert System

Expert systems are intelligent computer programs that use knowledge and inference procedures to solve problems that are difficult enough to require human expertise for their solutions [1, 10]. These systems are usually not easy to implement in the conventional programming languages, such as C, Pascal, FORTRAN. Instead, they are usually implemented in rule-based languages. There are two major components in this kind of programming paradigm – rules and facts. A rule is composed of criteria of facts and actions. The inference engine only executes the rules whose criteria are matched with some facts.

2.2. CLIPS

CLIPS (C Language Integrated Production System) is a tool for building expert system. The original version of CLIPS was developed and maintained by NASA from1985 to 1990s. CLIPS is now maintained independently from NASA as public domain software. Just like other expert system languages, the program coded in CLIPS contains only two primary elements: facts and rules. The inference engine would match the pattern of the rules with the facts and then fire the actions of the matched rules. There are several advantages of CLIPS: (1) efficient, (2) free, (3) portable, (4) high ability of extension. The extended functions of CLIPS can be written in C and combined with CLIPS engine, and CLIPS is able to call the functions simply in CLIPS programs. This feature is so-called “External Function Definition.”

2.3. Cluster System

Cluster system is comprised of loosely coupled computers that work together in parallel by dividing a big job into several small jobs. Nowadays, cluster system is usually connected by local area networks, and composed of many commodity PCs instead of expensive workstations. They are very cost-effective to provide high performance and high availability.

The typical clusters are homogeneous. The available resources, including computing power, network bandwidth, and so on, of computing nodes are all the same [12]. On the other hand, if the available resources of each node in cluster are different from each other, the system is called heterogeneous cluster. In the latter case, there exists a load balancing problem to make each node be allocated with proper workloads.

To parallelize a program, the MPICH Library is usually required. MPICH is an implementation of Message Passing Interface 1.1 standard and used in parallel computing [3]. It is cross-platform, including UNIX, Microsoft Windows, and so on.

2.4. Grid System

Grid system is a system connecting different cluster systems by wide area networks, usually the Internet [11]. There are some major differences between grid and cluster systems. (1) Because grid system is connected by the Internet, the bandwidth between clusters is fluctuant. (2) The resources of each cluster are different from each other. Thus, how to gather available resources to improve performance is a key issue on grid systems.

To build computing grid, we need to install the grid middleware, Globus Toolkit. Globus Toolkit is an open source toolkit to construct grid [5]. It is maintained by Globus Alliance. The implementation consists of some standards such as Grid Security Infrastructure (GSI), Job Submission Resource Framework (WSRF), and so on. Also, it is usually combined with MPICH-G2. MPICH-G2 is a grid-enabled implementation for Message Passing Interface 1.1 standard [4]. It allows the user to use the services provided by Globus Toolkit directly.

2.5. Previous Work

CLIPS is suitable to design expert system. Due to the characteristics of rule-based language, it would take very long time to complete execution. There have been some researches proposed previously to improve the performance by parallelization. Their works were established on shared memory [6], Intel Hypercubes [7], and distributed system [8]. Myers proposed to parallelize CLIPS program by PVM [9]. These methods are all implemented on some specific architecture and based on MPMD (Multiple Program Multiple Data) programming model. None of them adopted MPICH Library for parallelization on cluster and grid systems.

Riley parallelized the CLIPS on FLEX 32 platform that is a large-grain shared-memory parallel computer [6]. He proposed to divide the whole set of rules into several subsets and to allocate these subsets to several processors.

Hall and Bennett implemented the parallelization on Intel Hypercubes architecture [7]. The user interface is same as the original CLIPS. They only inserted some parallel calls and commands to make the CLIPS can

run on each node in Hypercubes. Furthermore, the parallel commands can assert or retract the facts into/from the memory of remote nodes.

Gagne and Garant combined the CLIPS and distributed system architecture [8]. Dai-clips is a distributed computational environment. Each CLIPS in it is an active independent computational entity. They can communicate with other CLIPS. Also, they can create, modify, or delete the expertise of CLIPS. The approach was implemented on distributed system architecture.

Myers and Pohl parallelized CLIPS by PVM (Parallel Virtual Machine) [9]. PVM is a library of C and FORTRAN. It supports distributive computing on distributed UNIX system. They used the MPMD model to parallelize CLIPS, and made it be able to run in the heterogeneous distributive computing experimentation.

3. Approaches

3.1. Parallelization

The SPMD Master-Slave programming model, as shown in Figure 2, requires some necessary information and subroutines to specify how to parallelize a program. All workers execute the same program and each of them can recognize which parts of the program he should do after evaluating the IF-THEN-ELSE expression. Thus, if we want to parallelize the CLIPS program, we must (1) construct SPMD Model in CLIPS syntax, and (2) build some subroutines into the CLIPS interpreter for exchanging messages. To achieve this goal, we have to provide some necessary information for SPMD and embed MPICH Library into CLIPS Engine. Also, we have to define some additional syntax or function calls for encapsulating MPICH library. After that, the CLIPS programmers can use the syntax which we defined for communication and synchronization through MPICH Library calls. The final architecture is shown in Figure 3. The CLIPS program contains some extended syntax we defined. It would be processed by the CLIPS interpreter directly.

To apply SPMD Master-Slave programming paradigm, there are two major key points. One is to construct SPMD Model, and the other is to define the extended subroutines properly. We can simply construct SPMD Model by assert a fact, named *MPI*, into CLIPS. The fact contains the *rank* of the process and the total number of processes. Then the fact can be used to tell which rules should be executed by which process as shown in Figure 4. Because each process

```
SPMD Master-Slave Program()
{
   if Master then // Master Process
      Divide the data into smaller parts
      for each slave i do
         Send the data to the salve_i
      loop
   else // Slave Process
      Receive the data from Master
      Do something with the data
   end if
}
```

Figure 2. SPMD Master-Slave Programming Model

has its own unique rank value contained in the *MPI* fact, for each rule, the process will test if its rank matches with all the patterns on the left-hand side of the rule. If they are matched, the actions in the right-hand side of the rule will be activated. The example shows how we can tell the master from the slaves: the process with the *RANK* slot equal to zero is the master.

Another goal can be implemented by adding new external function definition, the interface provided by the CLIPS interpreter. We have defined two additional syntaxes, mapping to *MPI_SEND* and *MPI_RECV* in MPICH Library, for sending and receiving data between CLIPS parallel processes. Also, to interpret

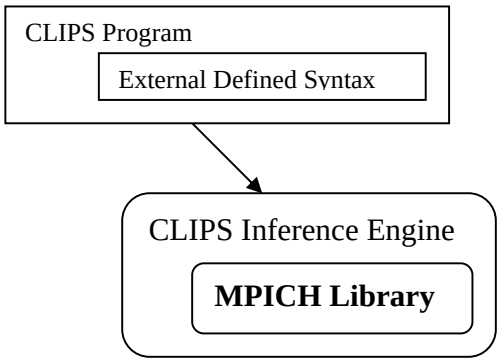

Figure 3. Embedding MPICH Library into the CLIPS inference engine.

these two syntaxes, we have implemented two corresponding subroutines, called *packageSendTo* and *packageRecvFrom*. To reduce the complexity of CLIPS program, we have to make the communication procedure easier. We define *packFact* for this purpose. Assume that we have *n* computing nodes available, there are *n* buckets in the CLIPS viewpoint. Each bucket has a serial number, *0* to *n-1*. Suppose the CLIPS programmer wants to pass some facts to some specific computing node, he just simply pushes the facts into the corresponded buckets by *packFact* subroutine, as shown in Figure 5. After all facts have been pushed into the corresponding buckets, the programmer only needs to call *packageSendTo*

```
SPMD Master-Slave CLIPS Procedure
(deftemplate MPI (slot RANK) (slot PROCS))
(defrule MASTER_DO
   (MPI (RANK 0))
   =>
   (printout t "I AM MASTER.")
)
(defrule SLAVE_DO
   (MPI (RANK ?r&:(neq ?r 0)))
   =>
   (printout t "I AM SLAVE " ?r )
)
```

Figure 4. SPMD Master-Slave CLIPS Procedure Example

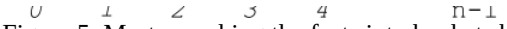

Figure 5. Master pushing the facts into buckets by invoking *packFact* subroutine, each bucket associated with one process.

subroutine to notify the CLIPS engine to send out all the facts in buckets. On the other hand, the receiver just simply called the subroutine *packageRecvFrom* to retrieve facts from the corresponding bucket in the CLIPS engine, as shown in Figure 6. Also, the receiving subroutine would pass the facts to the CLIPS program automatically. The message passing programming model is shown in Figure 7. In the example, there are four rules defined in the codes. The first two would be executed by master and the other two by slave. The rules cannot be executed out of order. The *packFact* must be done before

Table 1. External Function Definitions for Parallelization

packFact	
Argumets	*integer(r), FactPointer*
Comments	*to push the facts into the #r virtual buckets*
packageSendTo	
Argumets	*none*
Comments	*to send out all the facts in the buckets*
packageRecvFrom	
Argumets	*integer(r)*
Comments	*to receive the facts which are sent by process #r*
packageISendTo	
Argumets	*none*
Comments	*same as packageSendTo, but non-blocking*
packageIRecvFrom	
Argumets	*integer(r)*
Comments	*same as packageRecvFrom, but non-blocking*
packBFact	
Argumets	*FactPointer*
Comments	*to push the facts into the bucket which is used to broadcast*
packageBcast	
Argumets	*integer(f)*
Comments	*broadcast the facts from process #f to all the other processes*
MPISynchronization	
Argumets	*none*
Comments	*to synchronize all the processes*

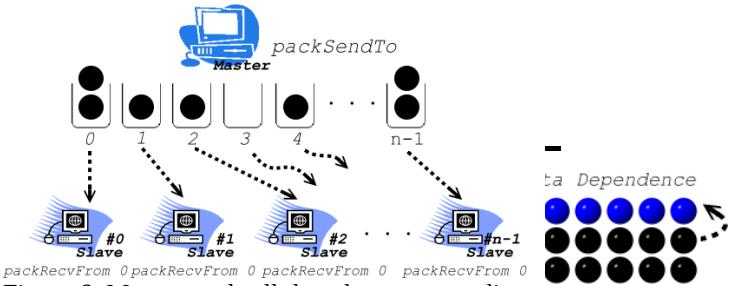

Figure 6. Master sends all data the corresponding workers by invoking *packageSendTo* subroutine.

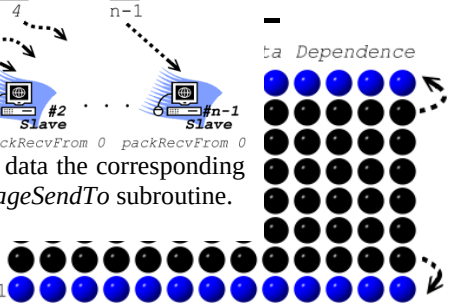

Figure 8. Cellular Automata Data Dependence

```
   packFact RE
)
(defrule MAST
   (declare (sal
   (MPI (RANK
   =>
   packetSendTo
)
(defrule SLAVE_RECV
   (declare (salience 1000))
   (MPI (RANK ?r&:(neq ?r 0)))
   =>
   packetRecvFrom 0
)
(defrule SLAVE_DO_SOMETHING
   (declare (salience 0))
   (MPI (RANK ?r&:(neq ?r 0)))
   =>
   ; Do something
)
```

Figure 7. An example CLIPS code of message exchanging between the master and the slaves.

packageSendTo. Thus, we declare appropriate salience values to ensure the correct execution order. Furthermore, we have defined some subroutines in CLIPS which are usually used in SPMD Master-Slave Programming Model, shown in Table 1.

3.2. An Example of Parallelized CLIPS Program

In our preliminary study, we aim to parallel the *Cellular Automata* application based on SPMD Master-Slave Programming Model. Assume we have *n* processes, and the dimension of the problem is $r \times r$ cells, the target generation is *g*. Each cell is coded as a fact with three slots: row number, column number, and status. At first, the master randomly generates the initial state of each cell. Then, the master divides *r* rows into *n* equal parts, and sends each slave a division. For instance, if a worker is responsible for working on the cells ranging from row *i* to row *j*, colored in black as shown in Figure 8, it also needs the statuses of the cells in row *i-1* and row *j+1*, colored in blue, because the next state of a cell is determined by its eight neighbors. In other words, each worker needs to deal

with *r*×*r*/*n* cells but it has to receive *(r*×*r/n)+2r* facts. After finishing the assigned job, the worker would return all the latest statuses back to the master. These procedures would be repeated for *g* times. Ideally, the performance would be improved up to *n* times.

4. Experimental Results

We have constructed a cluster system and a grid cluster system, as shown in Table 2, for experimentation in National Changhua University of Education. The cluster system, named Uranus, is composed of two dual-core Intel Pentium D personal computers. The grid cluster system consists of two clusters. The first cluster, named Sun and located in Bao-Shan Campus, consists of four dual-core PCs. The second cluster located in Jin-Der Campus, named CC, is comprised of two SMPs, each SMP has two dual-core CPUs on board.

4.1. Cluster System

We run the *Cellular Automata* CLIPS Program in non-parallel fashion on our Uranus Cluster. The cluster is set up by MPICH 1.2.7p1, and no grid middleware installed. Figure 9 shows the comparison of the total execution time with different cell dimension by using one Intel Pentium D E2160 processor. All the execution times are normalized to the one of 5×5 cells. We can clearly see that when the dimension is increased, the total execution time is raised up dramatically. For instance, when the dimension of cells grows from 10×10 to 100×100, i.e., enlarged by 100 times, the required execution time becomes to be 17536.4 times. It explains that the CLIPS execution time is increased exponentially when the dimension becomes larger.

Table 2. The configuration of Uranus Cluster, SUN Cluster and CC Cluster

Uranus Cluster (2 dual-core PCs)	
	2.0GHz, 2 Cores per CPU
Memory	1 GB DDR-400 Registered ECC ×4
Swap	2048 MB
Hard Disc	SATAII 320 GB
OS	CentOS 4.4, kernel 2.6.9-42.ELsmp

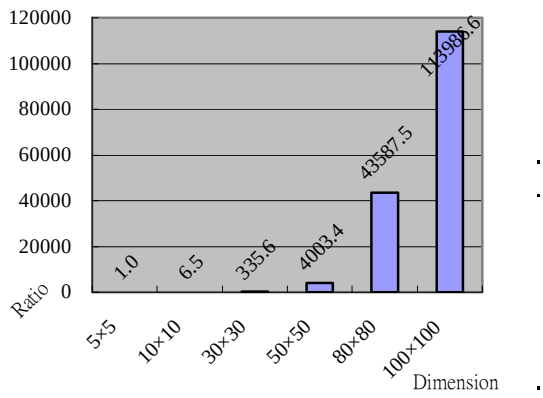

Figure 9. Comparison of normalized execution time of different dimensions

Next, we run the CLIPS Program in parallel on our Uranus Cluster System. The Figure 10 shows that the execution times with different cell dimensions by using different number of processes. Obviously, the larger dimension to be dealt with, the more improvement we have. We can find out that there is a substantial improvement if we compare the execution time of executing 100×100 cells by one process with the one by two processes. The reason is as follows. We divide the facts into two equal parts and give one part to one worker. And due to the facts to be processed for one process is decreased, the probability of cache miss and memory swapping becomes lower. Thus it derives the significant improvement.

The following Figure 11 shows the performance improvement ratio. The ratio is derived from dividing the execution time running on one process by the one on *n* processes. The ideal case is that if we run the program using n processes, the performance would be *n* times of that using one processor. We can clearly see that when the dimension is above 50 × 50, the improvement ratio is about the same as the ideal case. But if the dimension is not large enough, the parallelization is not cost-effective because of the parallelization overhead, including message exchanging. Furthermore, we can find that the best improvement occurs at the case of 50×50 cells. If the data size exceeds 50×50, the improvement would be decreased gradually. It might be caused by the high time complexity of CLIPS. Assumed there are *n* cells. To test if two cells are neighbors, the CLIPS inference engine has C_2^n matches to verify. The matches required to be tested for 50×50 and 100×100 cells are

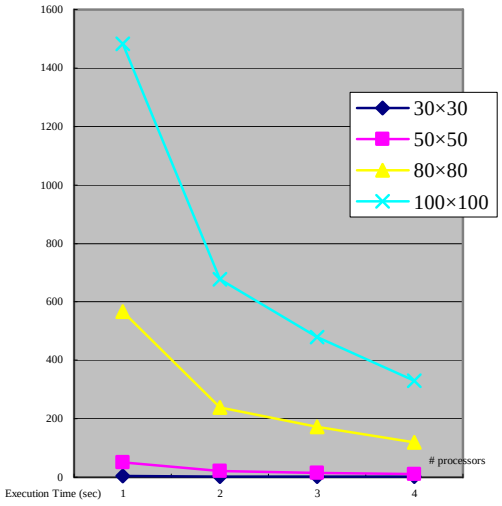

Figure 10. Execution time using different number of processors and different number of dimension

C_2^{50} and C_2^{100}, respectively. If we divide the data into four equal parts, each process has to work on 12.5 rows and 25 rows for these two problem sizes, respectively. For each process, there are $C_2^{12.5}$ and C_5^{25} matches to be verified for these two problem

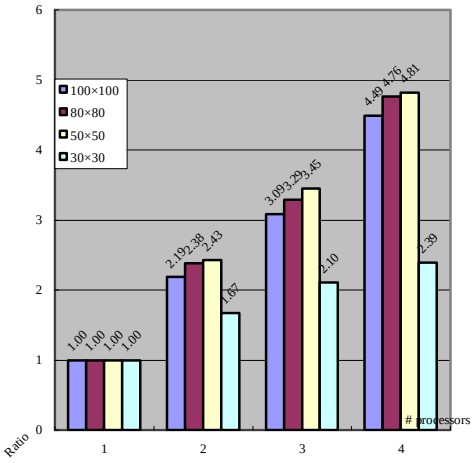

Figure 11. Comparison of performance improvements on a cluster system

sizes, respectively. As a result, the computational complexity is reduced to about 1/18.6 for the case of 50×50, and 1/16.5 for 100×100.

4.2. Grid Cluster System

In this subsection, we run the *Cellular Automata* in parallel on our Grid System. Our Grid System is set up by Globus Toolkit 4.0.5 and MPICH-G2. The grid system is comprised of Sun Cluster and CC Cluster. These two clusters are located in two different campuses of National Changhua University of Education and connected by 400 Mbps Internet. Consequently, the available bandwidth of the network is fluctuant during any time interval. Also, the communication time would be longer than that on cluster system. Thus, the message passing would play a more important role on grid system. Unfortunately, because of the character of CLIPS, it is hard to measure the communication time on runtime. If we do that, it would be an overhead for execution. Thus, in the following, we just focus on the total execution time.

Figure 12 shows the performance improvement ratio under different combinations of computing nodes. *CC8* and *Sun8* stand for the configurations of using 8 cores on CC Cluster and Sun Cluster, respectively. "*CC4 + Sun4*" represents the configuration consisting of four cores on CC Cluster and four cores on Sun Cluster. Finally, "*CC8 + Sun8*" represents the configuration comprised of 8 cores on CC Cluster and 8 cores on Sun Cluster. The ratio is the result of dividing the execution time of *n* processors by the time of one processor. The execution time of one processor is measured by running the application on one of the

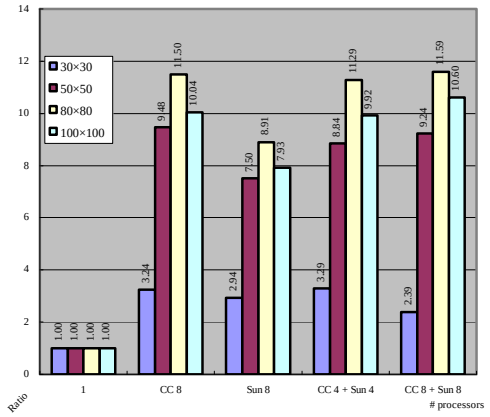

Figure 12. Comparison of performance improvements on a grid system

PCs in Sun Cluster. Because the computing power of CC Cluster is better than Sun Cluster as shown in Table 2, the *CC8* configuration has better performance than Sun8. Note that the eight parallel processes are running on the same cluster when we use 8 cores, resulting in shorter-latency communications.

Let's focus on *CC8*, *Sun 8*, and "*CC4 + Sun4*". We can find that the performance improvement provided by the configuration of "*CC4 + Sun4*" is between those by *CC8* and by Sun 8. It is reasonable because the computing power of CC Cluster is better than that of Sun Cluster. Furthermore, the communication between CC and Sun is over Internet. It would increase communication cost. On the other hand, we also find that the improvement of "*CC8 + Sun8*" is similar to that of *CC8* and of "*CC4 + Sun4*". The reason is that the data size is not large enough for each process when there are 16 parallel processes. Moreover, 8 out of 16 processes communicate with the master node through the Internet. The communication cost of the Internet is much larger than that of the local area network. It is expected that the performance might improve more when the data size is increased.

5. Conclusion and Future Works

In this paper, we have proposed a preliminary parallelization model for CLIPS programming

language based on the SPMD Model, which is easy to design and maintain. Also, we have defined some extended function calls for parallelizing the CLIPS application. The CLIPS programmers can parallelize their application without having to alter their original codes. They only need to insert the extended function calls to the beginning and the ending of their codes.

According to our experimental results, under this model, the performance improvements are significant. In the future, we would consider the characteristics of heterogeneity of clusters and grids to design a model under considering the load balancing issue based on SPMD Model. Furthermore, we would also investigate on how to support more parallelization model and features in CLIPS for higher degree of the parallelization of CLIPS application.

6. References

[1] CLIPS, http://www.ghg.net/clips/WhatIsCLIPS.html
[2] Joseph C. Giarratano, Gary D. Riley, *Expert Systems: Principles and Programming*, Thomson Course technology, 2005.
[3] MPI, http://www-unix.mcs.anl.gov/mpi/
[4] MPICH-G2, http://www3.niu.edu/mpi/
[5] Globus, http://www.globus.org/
[6] G. Riley, "Implementing clips on a parallel computer," In *Proc. SOAR'87*, NASA/Johnson Space Center, 1987.
[7] L. Hall, B.H.Bennett, and I.Tello, "Pclips: Parallel clips," In *Procs. Clips'94*, pages 307–314, 1994.
[8] D. Gagne and A. Garant, "Dai-clips: Distributed, asynchronous, interacting clips," In *Procs. Clips'94*, pages 297–306, 1994.
[9] L. Myers and K.Pohl, "Using pvm to host clips in distributed environments," In *Procs. Clips'94*, pages 177–186, 1994.
[10] Joseph C. Giarratano, Gary D. Riley, *Expert Systems: Principles and Programming*, Thomson Course technology, 2005.
[11] Ian Foster, Carl Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers Inc., 2003.
[12] B. Wilkinson, M. Allen, *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*, Second Edition, Prentice Hall Publisher, 2005.