A Parallel CLIPS-based Course Timetabling Expert System

Chao-Chin Wu, Lien-Fu Lai
Dept. Comp. Sci. & Info. Eng.
Nat'l Changhua Univ. of Education
{ccwu, lflai}@cc.ncue.edu.tw

Liang-Tsung Huang
Dept. Information Communication
MingDao University
larry@mdu.edu.tw

Wei-Chao Chang
Dept. Information Communication
National Tsing Hua University
s92610030@mail.ncue.edu.tw

***Abstract*—Course timetabling is a complex problem that cannot be dealt with by using only a few general principles. We proposed an artificial intelligence approach that integrates expert systems and constraint programming to implement a course timetabling system. By adopting the approach, it is easy to formulate and customize for supporting requirement changes and the difference between hard and soft constraints can be also addressed. However, it is very time consuming to achieve a feasible timetable because the inference engine is CLIPS-based. CLIPS is a rule-based language and relies on repeatedly matching facts and rules to draw conclusions. To address the problem, we propose parallelizing the execution of the timetabling system in emerging cluster systems. We parallelize the inference process of one course by partitioning CLIPS rules into multiple running pieces, where each running piece is inferred by a slave process. To ensure achieving correct solutions, we identify four possible problems that might occur if rules are divided improperly. Furthermore, how to avoid these problems and how to deal with them are also introduced. In the implementation of the system, the MPICH library is embedded into the *C*-based inference engine for interprocess communication. In addition, a programming model, which we transmit facts in *C* and infer rules in CLIPS, is also proposed. Experimental results show that the proposed parallel timetabling system achieves superlinear speedup when running in a cluster system.**

***Index Terms*—Course timetabling, parallel computing, CLIPS, expert system, cluster system.**

I. Introduction

The course timetabling problem is to allocate a set of courses into predetermined time slots (typically within a week), while satisfying a set of constraints of various types. It is a complex problem that cannot be dealt with by using only a few general principles. To address the timetabling problem, many approaches have been proposed for dealing with a variety of timetabling programs [2, 4-7, 11-14, 15].

We propose an artificial intelligence approach that integrates expert systems and constraint programming to implement a course timetabling system. The proposed approach has the following advantages. (1) The timetabling systems are easily capable of reformulation or customization to support changes since the timetabling problem varies significantly from institution to institution, in terms of specific requirements and constraints. (2) It is easy to capture knowledge and incorporate it into the timetabling system since expertise helps in reducing the search space and in fitting the solution to the context. (3) The difference between hard constraints and soft constraints can be addressed.

However, because the inference engine of the proposed expert system is CLIPS-based [1], it is very time-consuming to achieve a feasible solution. To address the problem, we propose parallelizing the execution of timetabling in emerging cluster system. Because it is hard to assign courses in parallel without having to solve the assignment conflict, we assign courses one by one and parallelize the assignment of one course. The CLIPS rules are divided into multiple running pieces for parallel inferences. Moreover, we propose how to divide rules and how to design safe CLIPS codes because a simple partitioning method will result in runtime errors. Four problems have been pointed out if rules are not divided properly. We introduce how to avoid them and how to deal with them by careful coding.

To execute a CLIPS application in parallel in a cluster system, the MPICH library is used. However, we cannot parallelize CLIPS programs directly because the CLIPS language does not support the feature. We modified the CLIPS inference engine by augmenting MPICH library functions because it is *C*-based. Moreover, we remove the data from the CLIPS program to an additional C file, leaving only the rules in the original CLIPS program. To integrate the CLIPS data into the C file, we have to convert the data to C data format, declare and process them. The proposed timetabling system has been implemented in a cluster system. Experimental results show that the proposed parallel system achieves superlinear speedup.

II. Related Work

A wide variety of approaches to the timetabling problem have been proposed. Monfroglio et al. [12] propose a Prolog-based system that employs backtracking for finding feasible timetables. The system decomposes and classifies constraints with respect to message passing and constraint ordering in order to minimize the backtracking and maximize the parallelism. Deris et al. [4] propose a constraint-based reasoning algorithm to model and solve the timetabling problem. The proposed system is implemented via an object-oriented approach, and can therefore be easily adapted to support changes. In [12], operational research models and local search techniques are used to assist the constraint programming search process by effectively reducing the solution search space. The authors propose a minimum cost matching algorithm to relax the constraint satisfaction model. Constraint logic programming [2] integrates logic programming and constraint solving so as to tackle combinatorial problems such as planning, scheduling, and resource allocation. This combination helps to make constraint logic programs expressive and flexible. Gunadhi et al. [6] introduce an automated timetabler that combines a data model and a knowledge base, developed via object-oriented methodology. Separating out the data, the knowledge, and the algorithms provides the flexibility to deal with changes, and the incorporation of human expertise helps to reduce the feasible solution search space. Solotorevsky et al. [14] develop a rule-based language, called RAPS, for specifying resource allocation problems and timetabling problems. The language enables the specification of a problem in terms of resources, activities, allocation rules, and constraints, and thereby provides a convenient knowledge acquisition tool. Dhar and Ranganathan [5] propose the use of an expert system, called PROTEUS, for the allocation of teachers to courses, and compare it with integer programming techniques. For predictive scheduling of passenger trains, Isaai et al. [7] introduce a lookahead, constraint-based algorithm that is designed using an object-oriented approach. In their approach, expert knowledge is used as a heuristic for finding practical solutions and is combined with the constraint-propagation technique.

CLIPS (C Language Integrated Production System) is a popular tool for building expert systems [1]. It consists of three components: facts, rules, and an inference engine, where the rules form the knowledge base. To solve a problem, CLIPS 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.

Though CLIPS is very suitable for developing expert systems, it is very time-consuming to run a CLIPS application because of inference processes. To address the problem, several studies have been proposed to execute a CLIPS program in parallel [3, 16-19].

III. Course Timetabling Expert System

We propose integrating expert systems and constraint programming to implement a course timetabling system [20]. Expert systems are utilized to incorporate knowledge into the timetabling system and to provide a reasoning capability for knowledge deduction. The constraint hierarchy and the constraint network are utilized to capture hard and soft constraints and to reason about constraints by using constraint satisfaction and relaxation techniques.

We propose a 4-tier system framework to implement timetabling systems. In the presentation tier, clients can manipulate data (e.g., courses, instructors, classrooms, preference time slots and exclusion time slots) via a browser. The flow control tier receives requests from clients and controls the system flow. In the business logic tier, the application software receives system messages from the web server and accesses the database in the DB server. Scheduling rules and domain knowledge are incorporated into the knowledge base. When scheduling starts, data stored in the DB server are translated into facts that are loaded into the working memory. By matching facts and rules, the inference engine can make inferences that achieve a solution based on the scheduling rules and domain knowledge. The results are stored in the database and are displayed in the web page. The clients then decide to approve, adjust or reschedule according to the analysis of results.

Expert systems are utilized to incorporate knowledge into the timetabling system and to provide the reasoning capability for knowledge deduction. The C Language Integrated Production System (CLIPS) is a productive development and delivery expert system tool that provides a complete environment for the construction of expert systems. The Web pages and the application software are implemented using JSP and Java. We adopt JClips [10] to combine CLIPS with Java by embedding the CLIPS engine in Java applications. In our previous work [13], we proposed a Knowledge Management through Knowledge Engineering (KMKE) approach to capturing knowledge Conceptual Graphs (CGs) and translating knowledge into CLIPS rules. Both scheduling rules and domain knowledge are represented as CLIPS rules and are stored in the knowledge base. A feasible solution can be inferred from these rules and existing facts. The inference engine makes inferences by deciding which rules are satisfied by facts and then applies the satisfied rules. Separating out the knowledge base, the facts, and the inference engine in expert systems provides greater flexibility in supporting changes. Changes in requirements can be mapped to the modification of corresponding rules in the knowledge base, while changes in data can be mapped to the modification of facts. When facts need to be changed, the clients can modify the database via a web page. New data can then be translated automatically into CLIPS facts in the working memory. On the other hand, knowledge engineers can add or modify the corresponding rules when the requirements are changed. Furthermore, as the inference engine is independent of the actual rules and facts, it can remain unchanged while the rules and facts are changed.

Course timetabling can be formulated as a constraint satisfaction problem by (1) treating the time slots of courses as a set of variables, each of which must be instantiated in a particular domain and (2) considering constraints as predicates on variables. A solution means a state in which the values of variables satisfy all predicates simultaneously. Constraints can be classified as hard or soft. A feasible solution to course timetabling should satisfy all hard constraints and as many soft constraints as possible. We utilize a constraint hierarchy to capture hard and soft constraints and a constraint network to reason about constraints.

A constraint hierarchy [8] can be established in terms of the strength (denoted as C_0, …, C_n) associated with each constraint. Constraints at the C_0 level are the strongest and cannot be violated. The remaining constraints are classified into different levels of strength (i.e., C_1, …, C_n) and can be relaxed to attain a feasible solution. This constraint hierarchy is useful for reasoning about constraints using constraint satisfaction and relaxation techniques.

A constraint network [9] can then be built level by level from the constraints at the top level of the hierarchy downwards. A feasible solution is achieved when all top-level constraints and as many weak constraints as possible can be satisfied simultaneously.

We adopt the salience of rules (the priority of rules) in CLIPS to realize the strength of constraints. Rules with higher salience are executed and satisfied first. All scheduling constraints in the constraint hierarchy can be represented as CLIPS rules with salience. The clients manipulate data via web pages, and the data stored in the Oracle DB server are translated into CLIPS facts. The CLIPS inference engine can then make inferences that achieve a feasible solution for the constraint network automatically.

To construct the course timetabling expert system, the requirements of instructors and resources are translated into facts; expert knowledge, hard and soft constraints are converted into rules; and the inference can draw a feasible timetable according to the facts and rules, as shown in Fig. 1.

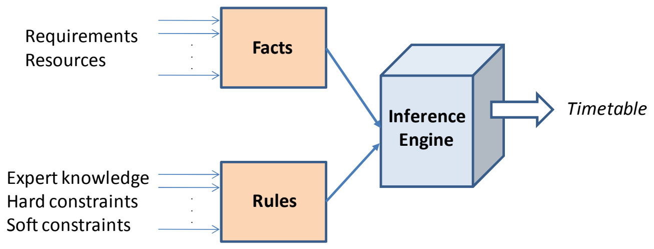


Fig. 1 The construction of the course timetabling expert system

IV. Parallel Vresion

Though the proposed course timetabling expert system is applicable to any department by establishing their won scheduling rules, it takes time before the result is available because the inference engine is CLIPS-based. To cope with this problem, we propose to execute the proposed course timetabling system in parallel on the emerging cluster system.

However, the big challenge is that the inherent serialization of the inference on course timetabling. We have to schedule courses one by one. When scheduling a course, we have to check if the course satisfies all the constraints and if it has any conflicts with other already scheduled courses. Even if the time tabling is parallelized by partitioning the inference according to some resource such as class or professor, it is still required that we have to merge all the inferred results sequentially because the final merged result should satisfy all the constraints. For instance, if the timetabling is divided according to classes, different classes can be scheduled independently and executed in parallel. However, due to parallel timetabling, more than one class timetable may require the same classroom at the same time slots. Similarly, a professor may have two classes at the same time slots. Consequently, we have to solve the above problems after parallel timetabling. The already scheduled timetables have to compare with one another to see if any conflicts occur. Therefore, we have to check all already scheduled results in the timetables one by one, which in turn may become the bottleneck because it has to be executed sequentially by the CLIPS inference engine. Even if the timetabling is divided by other resources, such as professor, we still have similar problems: to allocate the same resource to more than one timetables at the same period. To cope with the problem mentioned above, we propose the following parallelization approach.

4.1 Parallelization method

Following the feature of CLIPS, we schedule courses one by one. However, we partition the scheduling of one course into multiple prioritized running pieces and execute these pieces in parallel, where each running piece consists of many rules. Every running piece will return if it has a feasible solution. After receiving all the results, the course will be scheduled according to the result of the running piece that has the highest priority among the pieces having feasible solutions.

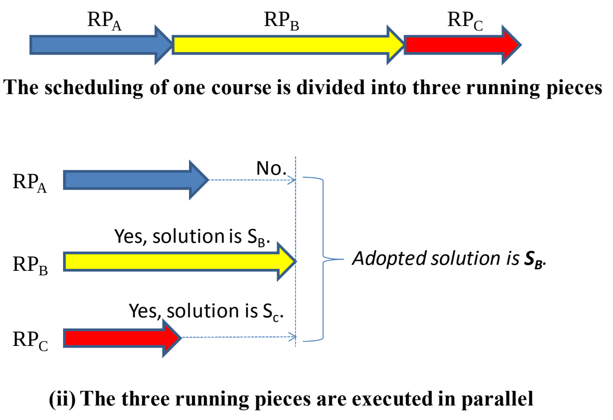
We explain the approach more detailed. CLIPS contains an inference engine which controls the execution of the rules in the knowledge base. The basic strategy used is known as forward chaining; this leads naturally to bottom-up, data-driven reasoning. The key to understanding forward chaining is the agenda. The agenda is how CLIPS keeps track of which rules are to be executed next. A rule is added to the agenda when all its conditions are satisfied. When a *run* command is issued, all the rules on the agenda are executed. To specify which rule should be matched before which rule, each rule is associated with a user-defined *salience* value. A rule with a larger salience value implies it has higher priority of performing pattern matching. Therefore, all the rules will be matched in the ordering specified by their salience values. If all the conditions of a rule are satisfied, its actions listed on the right hand side will be fired, usually resulting in that new rules are added into the agenda and the other rules of the same type will not be inferred.

In the problem of course timetabling, when a course, represented as a fact, is selected for inference, all the rules will be matched with the course one by one according to their salience values. If a course is scheduled to some time slots by a certain rule, the remaining untouched rules will not be inferred and another course will be selected for next scheduling.

For instance, assume that the scheduling of one course consists of three running pieces, RP_A, RP_B and RP_C as shown in Fig. 2(i). The salience values of the rules in RP_A are all larger than any one in RP_B, and that in RP_B are all larger than any one in RP_C. Note that the running piece having larger salience values has a higher priority of reasoning. Consequently, if a course can be scheduled by any one of the rules in RP_A, all the rules in RP_B and RP_C will not be matched with the course. However, if on rule in RP_A can be matched with the course, the rules in RP_B will be inferred for the course. Similarly, if the course can be scheduled by any one of the rules in RP_B, all the rules in RP_C will not be matched with the course. The rules in RP_C will be inferred only when no rule in RP_A and RP_A can be matched with the course. The behavior is the same as the nested *if-then-else* structure in conventional algorithmic language such as *C* and *Java*.

To parallelize the nested-*if-then-else*–like inference for scheduling one course, we propose a two phase parallelization approach. In the first phase, the inferences of all running pieces are performed in parallel and each of them returns if it has found a solution for scheduling the course and the corresponding solution. Each running piece is inferred by an independent inference engine. In the second phase, we decide which solution will be adopted by selecting the one that are inferred in the running piece having highest priority of reasoning among all the running pieces having solutions. Based on the decided result, another course will be selected for next scheduling.

We use the example shown in Fig. 2(i) for further explanation. The three running pieces can be inferred in parallel and only RP_B and RP_C have solutions, as shown in Fig. 2(ii). Since RP_B has higher priority than RP_C, the solution S_B will be adopted for the course. Before scheduling next course, the solution S_B has to be asserted into every inference engine. It is because every inference engine has to know which time slots have been allocated to the course before it starts to schedule the next course.



(i) The scheduling of one course is divided into three running pieces

(ii) The three running pieces are executed in parallel

Fig. 2. Timetabling by a sequential method and by a parallel method

4.2 How to divide into running pieces

According to the basic idea of parallelization proposed in the previous subsection, the rules will be divided into several running pieces according to their individual salience values. However, improper division might result in wrong solutions or infinite loops. Consider the following example. A rule, called R_x, has one action, A_m, listed on its right hand side, where A_m has to be inferred by another rule R_y. If all its conditions are satisfied at runtime, the action, A_m, will be added into the knowledge base for further inference. The inference engine will use R_y to infer A_m. If R_x is in the running piece RP_A and R_y is in RP_B after division, there are two possible problems incurred.

The first problem is that the action A_m cannot be inferred by the inference engine responsible for RP_A because R_y is in RP_B rather than RP_A. To cope with the first problem, RP_B should include R_y to ensure that A_m can be inferred by R_y. However, after inferring A_m by R_y, the inference engine might require other rules not in RP_B for further inference.

As a result, more and more rules have to be included in RP_B to draw the conclusion by further inference. That is, all the rules in the inference chain have to be included in the same running piece, which we call the property of *self-containment*. The property of self-containment might lead to no possible ways of rule partitioning because a rule may included in more than one running piece, which may result in duplicated conclusions. We have to compare the results and return only one result. If this problem occurs, parts of the inference of scheduling one course have to be executed concurrently on multiple running pieces, resulting in a poorer performance improvement. Fortunately, we can avoid this problem by carefully designing the application of course timetabling. No rules will fire any action that requires rules out of their individual belonged running pieces.

The second problem is that the action A_m can be inferred only by the rules in RP_B. This problem occurs when there are multiple rules, distributed in different running pieces, which can be matched with the action A_m, only the rules in RP_B can be matched with the action A_m, resulting in an undesirable result or infinite loop. For instance, assume that the action A_m can be inferred by two rules, R_x and R_y, where R_x has a higher salience value than R_y. Assume that R_x and R_y are RP_A and RP_B, respectively. Following the sequential execution of the original CLIPS inference, R_x will be selected for inference first. Assume that R_x will be matched with the action A_m, resulting in that the inference by R_x will not be performed for A_m. However, because RP_B contains only the rule R_y, A_m will be matched with R_y instead of R_x. The match between A_m and R_y have three possible results. First, assume that the match is successful. The inference result may be different from that inferred by R_x. To cope with the problem, we have to compare the results from running pieces and decide the result by the running priority. Second, if the match is failed, the inference engine will conclude that no feasible solution. This result is not harmful because it will be filtered out when it returns to the master. Third, if the match is failed and the rule R_y fire an action to retract some facts from the knowledge base, the inference engine can re-match the action A_m only with the rules in RP_B one by one again though the rules in RP_A should be matched first, as shown in Fig. 3. If no retraction can result in an inference result, the inference engine will fall in infinite loop. In the application of course timetabling, we will retract an already scheduled course by the lowest-prioritized rule if the current selected course cannot be scheduled by any other rules. After retracting an already scheduled course, the current course will be inferred from the highest-prioritized rule to the lowest-prioritized rule again. To cope with the third problem and according to the feature of the reasoning of course timetabling, we propose that for a rule having retraction actions, it has to be the lowest prioritized one in its running piece and the retraction action will be executed only once.

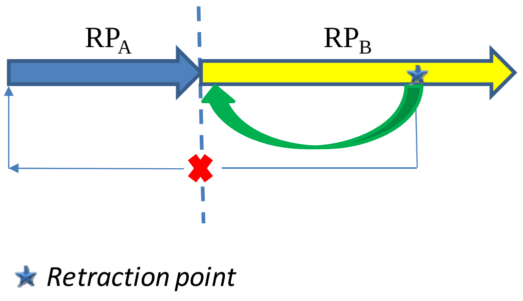


Fig. 3 The inference process when a retraction action occurs

4.3 Implementation

To execute a CLIPS-based expert system in parallel on a cluster system, the application has to be parallelized with MPICH library functions. However, CLIPS language does not support the feature, i.e., we cannot invoke MPICH functions directly from a CLIPS program. Therefore, we modified the CLIPS inference engine to execute the CLIPS file in parallel because the inference engine is written in the *C* language. Moreover, the inference engine is coded based on the SPMD model. When being executed in parallel, each inference engine will read a CLIPS file to build the knowledge base.

To assign course one by one and execute the assignment of one course in parallel, we propose the following programming model, as shown in Fig. 4, for parallel CLIPS program execution. A CLIPS file contains only the rules and all the processes will read a CLIPS file for making inference. The facts are managed in a *C* file that will be linked with the inference engine. Therefore, the facts have to be converted into *C*-format data. At the run time, the data will be distributed to multiple processes to reduce the execution time of making inference. Meanwhile, we have to convert these *C*-format data and insert into the fact list in the inference engine. Similarly, the inference results have to be extracted from the inference engine to display the reasoning conclusions.

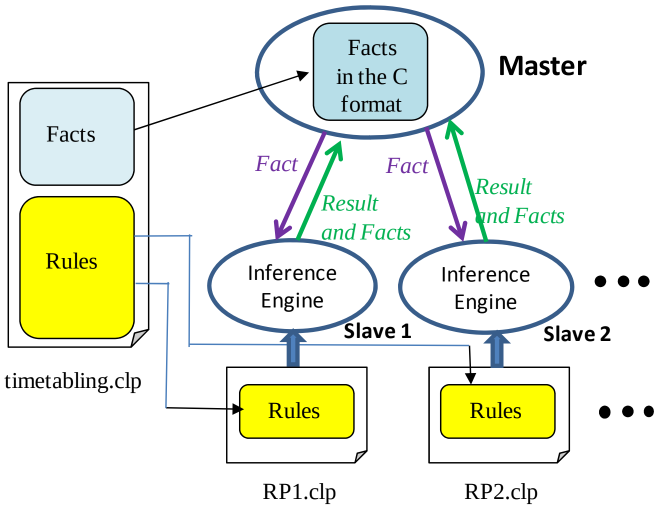


Fig. 4 The programming model of parallel CLIPS program execution

We divide the rules of the parallel expert system into four running pieces according to the partition method mentioned in the previous subsection. Each running piece is saved as an independent CLIPS file. At runtime, four slave MPI processes will be forked for the inferences of these four running pieces. Each slave process is associated with a CLIPS file containing one running piece. The master will send facts one by one to all slaves. After receiving one fact, each slave process perform the inference according to the fact and its associated CLIPS file. As soon as the inference is done, each slave sends the inference result back to the master process. In addition, the facts in the knowledge base will be sent back to the master. Then, the master determines which result will be adopted for the current course. The result from the slave that has the largest priority among all the salves having feasible solution will be chosen for scheduling the course. Before the master sends next course to slaves for further inference, the master will broadcast all slaves the facts from the chosen slave. After receiving the facts, each slave will replace the knowledge base by these facts, which assuring a consistent view of the knowledge base before scheduling the next course. The scheduling will be done when there is no course in the master.

V. Performance Evaluations

We have constructed a cluster consisting of four Intel PCs to evaluate the parallel course timetabling expert system. The configuration of our cluster system is shown in Table 1. The MPICH2 library is adopted for inter-process communication.

Table 1 The configuration of our cluster system

Intel Pentium III PC ×6	
CPU	Intel Pentium III, Coppermine 1000Mhz
Memory	256MB + 128MB
Swap	779144 KB
HD	IDE 10GB
OS	RedHat 9.0

The execution times of the sequential and the parallel course timetabling expert systems are compared in Fig. 5. The sequential version is executed on one PC in our cluster system while the parallel version is executed on five PCs in our cluster system. As we can see, the parallel version can cut the execution time significantly. The speedup is 12.47, derived by dividing the execution time of the sequential version by that of the parallel version. The speedup is superlinear because we use only five PCs but the speedup is more than five. The reason that we can obtain superlinear speedup is explained as follows. The total memory space of the cluster system is larger than that of one PC. Because CLIPS-based applications are memory-intensive programs, the parallel version can take more advantage from larger memory space.

VI. Conclusions

In this paper, we investigate into how to construct a parallel course timetabling expert system. We proposed an artificial intelligence approach that integrates expert systems and constraint programming to implement a course timetabling system. Expert systems are utilized to incorporate knowledge into the timetabling system and to provide a reasoning capability for knowledge deduction. Separating out the knowledge base, the facts, and the inference engine in expert systems provides greater flexibility in supporting changes. The constraint hierarchy and the constraint network are utilized to capture hard and soft constraints and to reason about constraints by using constraint satisfaction and relaxation techniques.

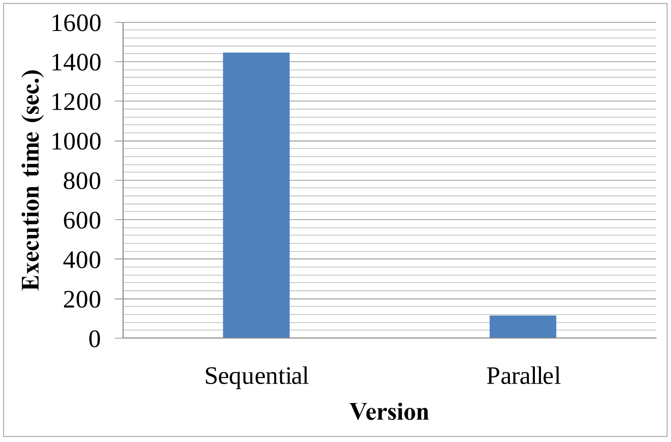


Fig. 5 Execution time comparison between the sequential and parallel versions.

The inference of the expert system is implemented in the CLIPS language. To address the problem that it is very time consuming to achieve a feasible timetable by CLIPS programs, we propose parallelizing the execution of assigning one course in a cluster system. The rules are divided into multiple running pieces and each running piece is executed by one slave process. We pointed out that improper rule division would result in four possible problems. The first problem should be avoided by restricting that no rules will fire any action in other running piece. The second and the third problems can be dealt normally. The fourth problem can be solved by the following restriction. A retraction action should be performed only once and the associated rule has the smallest salience value in its running piece.

Finally, we propose a programming model that separating facts and rules into two types of files. Facts are processed, transmitted by the *C* language and rules are inferred by the CLIPS language. Inter-process communication in a cluster system is enabled by incorporating the MPICH library into the CLIPS inference engine. Experimental results show that the proposed parallel timetabling system achieves superlinear speedup when running in a cluster system.

ACKNOLWDGEMENT

This research is supported by the National Science Council of Taiwan under contract number: NSC98-2221-E-018-008-MY2.

REFERENCE

[1] CLIPS, http://www.ghg.net/clips/CLIPS.html
[2] P. Boizumault, Y. Delon and L. Peridy, Constraint logic programming for examination timetabling, The Journal of Logic Programming , pp. 217-233, 1996.
[3] Dana Petcu, “Parallel Jess,” ISPDC 2005, pp. 307–316, 2005.
[4] S. Deris, S. Omatu and H. Ohta, Timetable planning using the constraint-based reasoning, Computers and Operations Research, Vol. 27, pp. 819-840, 2000.
[5] V. Dhar and N. Ranganathan, Integer programming vs. expert systems: An experimental comparison, Communications of ACM, Vol. 33, No. 3, pp. 323-336, 1990.
[6] H. Gunsdhi, V.J. Anand and Y.W. Yong, Automated timetabling using an object-oriented scheduler, Expert System with Applications, Vol. 10, No. 2, pp. 243-256, 1996.
[7] M.T. Isaai and N.P. Cassaigne, Predictive and reactive approaches to the train-scheduling problem: A knowledge management perspective, IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews Vol. 31, No. 4, pp. 476-484, 2001.
[8] V. Kumar, Algorithms for constraint-satisfaction problems: A survey, AI Magazine, Vol. 13, No. 1, pp. 32-44, 1992.
[9] A.K. Mackworth, Consistency in networks of relations, Artificial Intelligence, Vol. 8, No. 1, pp. 99-118, 1977.
[10] M. Menken, JClips - CLIPS for Java, http://www.cs.vu.nl/~mrmenken/jclips.
[11] MPICH, http://www.mcs.anl.gov/research/projects/mpi/mpich1/
[12] A. Monfroglio, Time-tabling through a deductive database: A case study, Data and Knowledge Engineering, Vol. 3, pp. 1-27, 1988.

[13] L.F. Lai, A knowledge engineering approach to knowledge management. Information Science, Vol. 177, pp.4072-4094, 2007.

[14] G. Solotorevsky, E. Gudes and A. Meisels, Raps: A rule-based language specifying resource allocation and time-tabling problems, IEEE Transactions on Knowledge and Data Engineering, Vol. 6, No. 5, pp. 681-697, 1994.

[15] C. Valouxis and E. Housos, Constraint programming approach for school timetabling, Computers and Operations Research, Vol. 30, pp. 1555-1572, 2003.

[16] C.-C. Wu, L.-F. Lai, Y.-S. Chang, "Towards Automatic Load Balancing for Programming Parallel Fuzzy Expert Systems in Heterogeneous Clusters", Journal of Internet Technology, Vol. 10, No. 2, pp. 179-186, 2009.

[17] C.-C. Wu, L.-F. Lai, Y.-S. Chang, "A Study of Designing a Grid-Enabled Expert System Language", Journal of the Chinese Institute of Engineers, Vol. 31, No. 7, pp. 1165-1179, 2008.

[18] C.-C. Wu, L.-F. Lai, Y.-S. Chang, "Using MPI to Execute a FuzzyCLIPS Application in Parallel in Heterogeneous Computing Systems," IEEE CIT 2008, pp. 279-284, 2008.

[19] C.-C. Wu, L.-F. Lai, K.-C. Lai, W.-C. Chang, Syun-Sheng Jhan, "Parallelizing Expert Systems with CLIPS Language for Grid Systems," WoGTA'07, pp. 125-131, 2007.

[20] L.F. Lai, C.-C. Wu, N.L. Hsueh, L.T. Huang, and S.F. Hwang. "An Artificial Intelligence Approach to Course Timetabling", International Journal on Artificial Intelligence Tools, Vol. 17, No. 1, pp. 223-240, Feb. 2008.