Performance Comparison of Models for Multiple Rule Firing

Steve Kuo and Dan Moldovan
skuo@gringo.usc.edu and moldovan@gringo.usc.edu
Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-1115
DRB-363, (213) 740-9134

Abstract

The performance of production programs can be improved by firing multiple rules in a production cycle. Although considerable amount of research has been done on parallel processing of production programs, the problem of multiple rule firing has not been thoroughly investigated yet. In this paper, we begin by identifying the problems associated with multiple rule firing systems: the *compatibility problem* and the *convergence problem* and present three multiple rule firing models which address them. The *rule dependence model (RDM)* addresses the compatibility problem using inter-rule data dependence analysis. The *single-context-multiple-rules (SCMRJ model* and the *multiple-contexts-multiple-rules (MCMR) model* address both the compatibility and the convergence problems. A production program executed under the SCMR and the MCMR models is guaranteed to reach a solution which is equivalent to the sequential execution. These three multiple rule firing models have been simulated on the RUB1C simulator, and the MCMR model, which has the highest performance, has been implemented on the Intel iPSC/2 hypercube. The simulation and implementation results are reported.

1 Introduction

Multiple rule firing production systems increase the available parallelism over parallel match systems by parallelizing not only the match phase, but all phases of the inference cycle. To speedup the derivation of correct solutions by firing multiple rules in a production cycle, two multiple rule firing problems — the *compatibility* and *convergence* problems — need to be addressed. The compatibility problem arises from the interferences between production rules. If a set of rules does not have inter-rule data dependence among themselves, they are said to be *compatible* and are allowed to execute concurrently. The convergence problem arises from the need to follow the problem solving strategy used in a production program. If the problem solving strategy is ignored in a multiple rule firing system, then two tasks may be executed out of sequence or two actions for the same task may be executed in the wrong order resulting in an incorrect solution.

There are three approaches to address the compatibility and convergence problems. The first approach considers only the compatibility problem and resolves it by data dependence analysis [6] [9] [11]. Both synchronous and asynchronous execution models have been proposed. In these models, rules which are compatible are fired concurrently in a production cycle. Because the convergence problem is not addressed in these models, the problem solving strategy for a production program may be violated when multiple rules are fired simultaneously. The second approach addresses the compatibility and convergence problems by developing parallel production languages. CREL [9] and Swarm [4] are two such languages. Production programs written in these languages do not use control flow or conflict resolution to ensure that the right rules are fired. Instead, production rules are fired as soon as they are matched. The correctness of these parallel production programs is guaranteed by showing that for any arbitrary execution sequence the correct solutions are always obtained [2]. A potential hurdle for CREL and Swarm is the possible difficulty to prove the correctness of a large production program. In addition, to be able to fire production rules as soon as they are matched may not be the same as being able to fire multiple rules concurrently. These questions will be answered when the benchmark production programs have been translated into CREL and Swarm programs and their performance measured.

The multiple rule firing models presented in this paper represents the third approach. They address the compatibility problem by data dependence analysis and the convergence problem by analyzing the

control flow in a production program to maintain the correct task ordering. In a production program, a complex problem can be solved by dividing it into smaller tasks until they are easily solved. These tasks are called *contexts* and each context is solved by a set of *context rules.* The multiple rule firing models improve the performance of a production program by activating multiple contexts and firing multiple rules concurrently. They guarantee the correctness of the solution by determining the conditions under which multiple contexts and multiple rules can be activated and fired. The multiple rule firing models have been simulated on the RUBIC simulator and implemented on the Intel iPSC/2 hypercube. The results indicate that these models have successfully addressed the problems associated with multiple rule firing.

2 Resolving the Compatibility and Convergence Problems

To resolve the compatibility and convergence problems successfully, one needs to understand how problems are solved in production programs. A useful method in general problem solving is the method of stepwise refinement, which resolves a complex problem by dividing it into smaller and smaller subproblems (or tasks) until they are easily solved. If other subproblems need to be solved before solving a subproblem, the program control is transferred from one subproblem to another. Production programs also employ the method of stepwise refinement to solve complex problems. First, the production rules in a production program are divided into subsets of rules, one subset for each subproblem. A subset of rules is called a *context* and each individual rule in the subset is called a *context rule.* Every context rule in the same context has a special *context WME.* Rules in different contexts have different context WMEs. A programmer can control which context is active by adding and removing the context WMEs. Context rules are divided into *domain rules* and *control rules.* Domain rules address the subproblem associated with the context and conflict resolution is used to select the right rule to fire. If other subproblems need to be solved before solving a subproblem, the control rules transfer the program control to the appropriate contexts by modifying the context WMEs. By analyzing the control rules, the control flow between different contexts can be determined. The problem solving strategy and the control flow diagram for an example production program is shown in Figure 1.

In this paper, we present three multiple rule firing models: the *rule dependence (RDM) model,* the *single-context-multiple-rules (SCMR) model,* the *multiple-contexts-multiple-rules (MCMR) model.* They resolve the compatibility and convergence problems at two

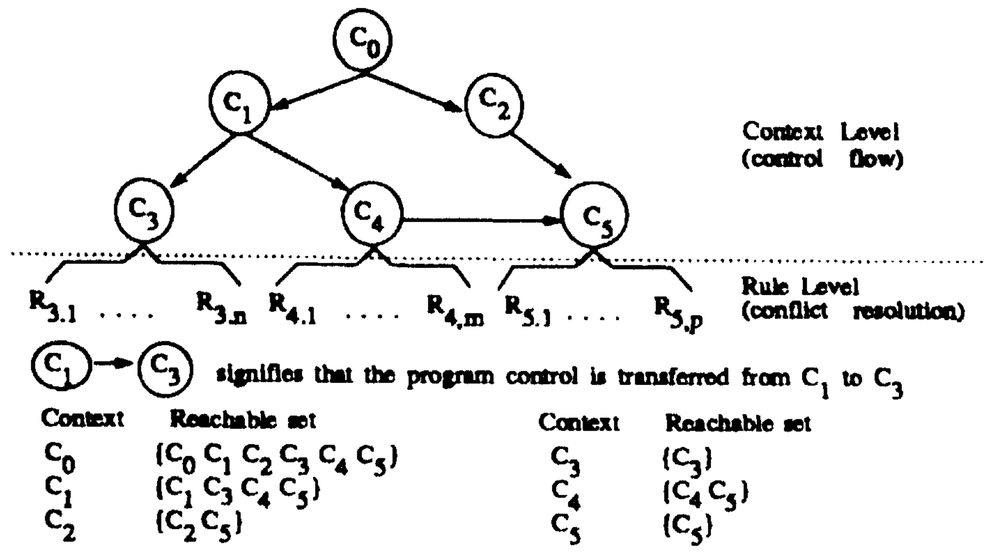


Figure 1: Control in Production Program

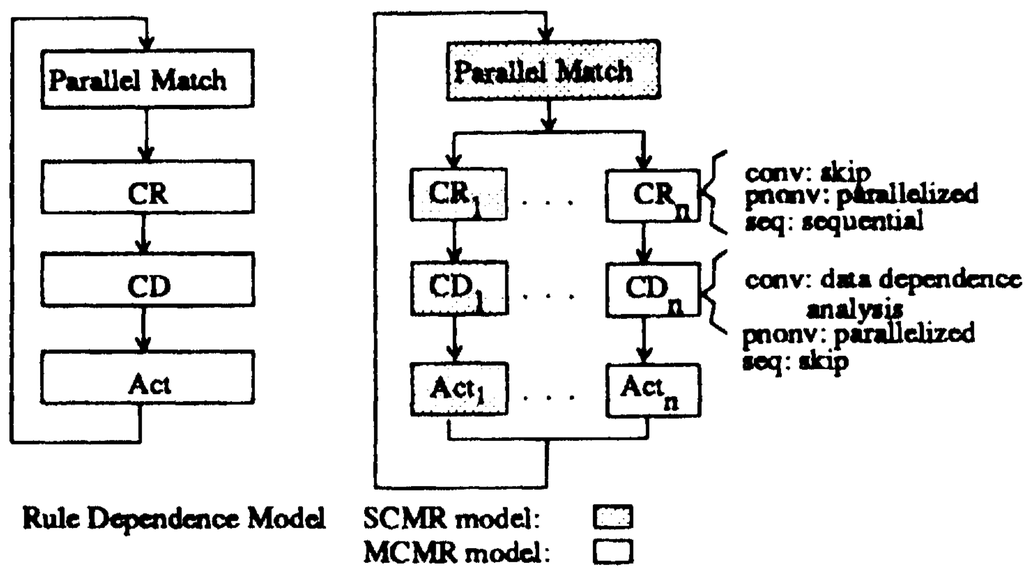
levels: the *rule* level and the *context* level. At the rule level, the compatibility problem is resolved by data dependence analysis. A set of rules is allowed to fire concurrently and are said to be *compatible* if executing them either sequentially or concurrently, the same state is reached. This is the case if there are no data dependences among rules in the set. The data dependence analysis is performed at compile time to construct a parallelism matrix P = $[p_{ij}]$ and a communication matrix C = $[c_{ij}]$. Rules R_i and Rj are compatible if p_{ij} = 0; they are incompatible if pij — 1. The communication matrix C is used for communication purpose when production rules are partitioned and mapped onto different processing nodes in a message-passing multiprocessor. Rules R_i and Rj need to exchange messages to update the database if c_{ij} = 1; they do not need to if C_{IJ} = 0.

The convergence problem is resolved at the rule level by dividing contexts in a production program into three different types: (1) *converging* contexts, (2) *parallel nonconverging* contexts and (3) *nonconverging* or *sequential* contexts. A context C is a converging context if starting at a state satisfying the initial condition INIT for that context, all execution sequences result in states satisfying the post condition POST for that context [2] [4]. Otherwise context C is a nonconverging context. The conflict resolution can be eliminated for a converging context because all execution sequences converge to the correct solution. Compatible rules can be fired simultaneously within a converging context without error. This is because firing a set of compatible rules concurrently is equivalent to executing them in some sequential order and all execution sequences reach the correct solution for a converging context (for proof, see [8]). For a nonconverging context, conflict resolution must be used to reach the correct solution. The performance of a nonconverging context can be improved by parallelizing its conflict resolution. A parallel nonconverging context is a nonconverging context whose conflict resolution is parallelizable and as a result multiple rules may be selected. The conflict

resolution for a sequential context is not parallelizable and only sequential execution is possible. By dividing contexts into different types and applying the correct execution model for each type, the compatibility and convergence problems are resolved at the rule level.

The compatibility and convergence problems are resolved on the context level by analyzing the control flow diagram to determine which contexts are allowed to be active at the same time. These contexts are called *compatible contexts.* Two contexts are compatible if their reachable sets do not intersect and rules in the two reachable sets do not have data dependences (for proof, see [7]). The reachable set for a context C_i, is the set of contexts which are reachable by following the directed arcs in the control flow diagram starting from C_i. Context C_i, is included in its own reachable set. The reachable set for context C_1 for the example production program in Figure 1 is $\{C_1, C_3, C_4, C_5\}$.

The production rules are analyzed at compile time to generate the compatibility context matrix $CC = [cc_{ij}]$. Two contexts C_i, and Cj are compatible and are allowed to be active at the same time if $cc_{ij} = 0$; they are incompatible if $cc_{ij} = 1$. The programmer then consults the CC matrix and modifies the production rules if needed so that only the compatible contexts will be activated concurrently in production program execution. In this way, the compatibility and the convergence problems are resolved on the context level.



1. Assume that there are n contexts in the production program.
2. conv = converging context
   pnonv = parallel nonconverging context
   seq = sequential context
3. CR = conflict resolution
   CD = compatibility determination

Figure 2: New Parallel Inference Cycle

3 Multiple Rule Firing Models

In this section, we present three multiple rule firing models which incorporate the solutions presented in Section 2 in varying degrees. The RDM model resolves the compatibility problem by data dependence analysis. It does not address the convergence problem. The SCMR and MCMR models address both the compatibility and convergence problems. The difference between the two models is that only one context is active at a time for the SCMR model but multiple contexts may be active simultaneously for the MCMR model. A new parallel inference cycle, shown in figure 2, is needed for the RDM, SCMR and MCMR models. It consists of four phases: the *match* phase, the *conflict resolution (CR)* phase, the *compatibility determination (CD)* phase, and the *act* phase. The RDM, SCMR, and MCMR models follow this new parallel cycle with some modifications, depending on how each handles contexts.

3.1 Data Dependence Model

The rule dependence (RDM) model addresses only the compatibility problem, not the convergence problem. The data dependence analysis is performed at compile time to construct the parallelism matrix $P = [p_{ij}]$ and the communication matrix $C = [c_{ij}]$. To capture the maximum parallelism, all four phases of the new parallel inference cycle are parallelized. For the match phase, either parallel shared-memory match algorithm [5] or parallel message-passing algorithm [1] is used, depending on the underlying architecture. At the conflict resolution phase, a dominant rule R_d is selected according to the conflict resolution strategy or at random. At the compatibility determination phase, a compatible set is chosen. Initially the compatible set contains only R_d. For each eligible rule R_i, if R_i, is compatible with all rules in the compatible set according to the parallelism matrix P, it is added to the compatible set. This step is repeated until every eligible rule is checked. At the act phase, all the rules in the compatible set are fired concurrently. The data dependence model is similar to the models proposed by Ishida [6], Miranker [9] and Schmolze [11].

Because the RDM model fires rules from different contexts concurrently, it may activate incompatible contexts or fire multiple rules simultaneously for sequential contexts. If all contexts are converging contexts and are compatible with each other, the RDM model would reach the correct solution and obtain high performance because it captures all the parallelism there is in each cycle. If this is not the case, the RDM model fails.

3.2 SCMR and MCMR Models

The SCMR and MCMR models extend the RDM model and address the convergence problem by dividing contexts into converging and nonconverging contexts and analyzing whether two contexts can be activated concurrently. For the match phase, the same technology used for the RDM model is used for the

SCMR and MCMR models. But unlike the RDM model which performs the conflict resolution, compatibility determination and act phases for all rules globally, the SCMR and MCMR models execute these phase for each context individually. In addition, different actions are performed for different types of contexts during the conflict resolution, compatibility determination and act phases.

For a converging context, the conflict resolution phase is skipped. For a parallel nonconverging context, the conflict resolution is parallelized by a special algorithm. For a sequential context, the conflict resolution is retained. During the compatibility determination phase, compatible rules are determined by data dependence analysis and fired concurrently for a converging context. For a parallel nonconverging context, a special algorithm is used to determine the set of compatible rules. For a sequential context, the compatibility determination phase is skipped. During the act phase, converging and parallel nonconverging contexts execute the set of compatible rules concurrently, and the sequential contexts fire the dominant rule selected by the conflict resolution. By allowing compatible contexts to be executed independently, the SCMR and MCMR models avoid unnecessary synchronizations and improve the available parallelism. The main difference between the SCMR model and the MCMR model is that only one context is active for the SCMR model, but multiple contexts may be active for the MCMR model.

4 Results

Six production programs developed at USC, CMU and Columbia have been simulated on the RUBIC simulator using four models: the sequential, RDM, SCMR and MCMR models. The RUBIC simulator is written in LISP and currently running on the Sun Sparc workstation. By analyzing the simulation results, we can determine the validity of the RDM, SCMR and MCMR models and measure their performance.

Table 4.1 lists and describes the six test programs simulated on the RUBIC simulator. The test programs were first simulated under the sequential model and the sequential simulation results are summarized in Table 4.2. The test programs were then simulated under the RDM model, in which all compatible rules were fired in a production cycle. The RDM simulation results are summarized in Table 4.3. Under the RDM model, production programs Toru-Waltzl6, Cafeteria, Snap-2d and Snap-TA reached the correct solutions but Tournament and Hotel failed. The reason that Toru-Waltzl6, Cafeteria, Snap-2d and Snap-TA were successful was because they contained only converging contexts. Tournament and Hotel failed because they contained nonconverging contexts and firing multiple rules concurrently violating the problem solving strategies. If the SCMR and MCMR models work, then by using a special algorithm to parallelize the nonconverging context in Tournament and executing the nonconverging context in Hotel sequentially, the two test programs should reach the correct solutions. In this way, the validity of the SCMR and MCMR models can be verified.

Table 4.1 Test Production System

Program	Description
A	Tournament: scheduling bridge tournaments
B	Toru-Waltz16: implementing the Waltz's edge labelling algorithm
C	Cafeteria: setting up cafeteria
D	Snap-2d[1]: a two-dimensional semantic network
E	Snap-TA: verifying the eligibility of TA candidates
F	Hotel: modeling hotel operations

Table 4.2 Sequential Simulation Results

	Production Programs					
Measurements	A	B	C	D	E	F
# of rules	26	48	94	574	574	832
# of sequential cycles = α	528	207	493	590	1175	5115

The test programs have been simulated under the SCMR model and the simulation results are summarized in Table 4.4. All six production programs reached the correct solutions under the SCMR model as expected. By using a special algorithm for the nonconverging context in Tournament, multiple rules were fired concurrently without error and a speedup of 6.21-folds was achieved. By firing rules sequentially for the sequential context and concurrently for the converging contexts for Hotel, a speedup of 8.77-folds was obtained. For Toru-Waltzl6, Cafeteria, Snap-2d, and Snap-TA, speedups of 3.06 to 8.94-folds have been obtained. The simulation results indicate that the SCMR model was able to capture a significant amount of available parallelism in production programs and we expected that the MCMR model would give even better performances.

The simulation results for the test programs under the MCMR model are summarized in Table 4.5. Like the SCMR model, all six production programs reached the correct solutions under the MCMR model as expected. For Tournament, Toru-Waltzl6 and Snap-2d, only one context can be activated at at time. As a result, their MCMR performances were the same

[1] Snap is a simulator for semantic network array processor under development at USC [10].

Table 4.3 Simulation Results for Rule Dependence Model

	Production Programs					
Measurements	A	B	C	D	E	F
Are RDM solutions correct	no	yes	yes	yes	yes	no
# RDM cycles = β	165	65	78	66	126	254
$Speedup_{\alpha/\beta} = \alpha/\beta$	N/A[2]	3.18	6.13	8.94	9.40	N/A

Table 4.4 SCMR Simulation Results

	Production Programs					
Measurements	A	B	C	D	E	F
Are SCMR solutions correct	yes	yes	yes	yes	yes	yes
Max # rules fired per SCMR cycle	120	14	10	15	100	40
Max # contexts activated per cycle	1	1	1	1	1	1
Ave # contexts activated per cycle	1	1	1	1	1	1
# SCMR cycles = γ	85	65	156	66	158	475
$Speedup_{\alpha/\gamma} = \alpha/\gamma$	6.21	3.18	3.06	8.94	7.46	8.77

Table 4.5 MCMR Simulation Results

	Production Programs					
Measurements	A	B	C	D	E	F
Are MCMR solutions correct	yes	yes	yes	yes	yes	yes
Max # rules fired per MCMR cycle	120	14	14	15	106	77
Max # of contexts activated per cycle	1	1	5	1	4	7
Ave # of contexts activated per cycle	1	1	2.00	1	1.21	2.33
# MCMR cycles = δ	85	65	78	66	125	247
$Speedup_{\alpha/\delta} = \alpha/\delta$	6.21	3.18	6.13	8.94	9.40	20.38
$Speedup_{\gamma/\delta} = \gamma/\delta$	1.00	1.00	2.00	1.00	1.26	2.32

as their SCMR performances. For Cafeteria, Snap-TA and Hotel, multiple contexts were activated concurrently and they achieved speedups of 6.13, 9.40 and 20.38-folds or 2.00, 1.26 and 2.32-folds better than their SCMR speedups. This indicates that the MCMR model does capture more parallelism than the SCMR model, but the additional speedups obtainable using the MCMR model over the SCMR model depend on the nature of the production programs. To compare the MCMR speedups and the RDM speedups for production programs Toru-Waltz, Cafeteria, Snap-2d and Snap-TA is instructive. We see that for these programs, their MCMR and RDM speedups were the same. This is not surprising since the RDM model also activates multiple contexts and fires multiple rules concurrently like the MCMR model does. The problem is that sometimes it erroneously activates incompatible contexts and fires multiple rules concurrently for sequential context.

N/A = Non-applicable

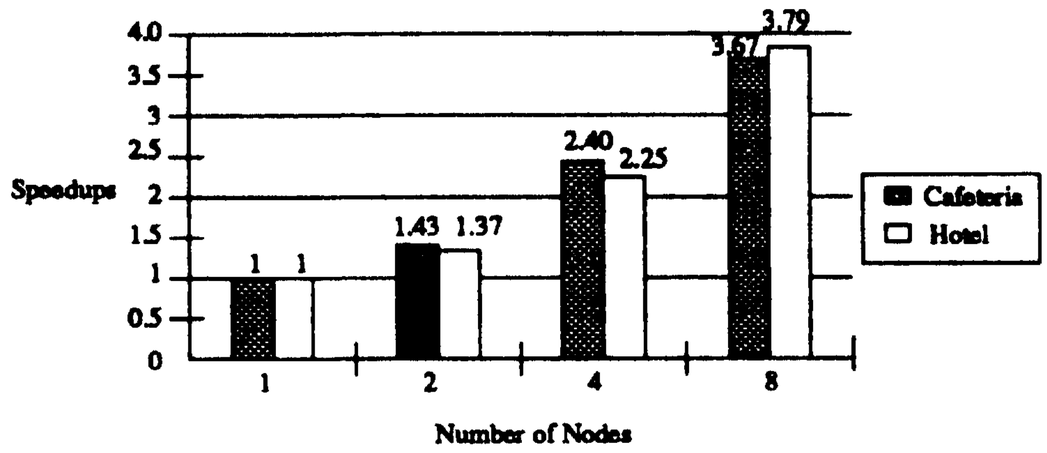


Figure 3: Performance for Partition-by-Context

4,1 Hypercube Performance Results

The MCMR model has been implemented on the Intel iPSC/2 hypercube because it provides the highest performance. Only eight nodes can be run concurrently due to the large memory requirement of LISP program. We have run two test programs, Cafeteria and Hotel, using two static partitioning schemes: partitioning-by-context and round-robin partitioning. When rules are partitioned by context, all rules in the same context are mapped to the same processing node. In round-robin partitioning, rules are allocated to the processing nodes in a circular fashion regardless of the contexts.

Cafeteria and Hotel were able to achieve good speedups when rules are partitioned by contexts. Their performance is shown in Figure 3. Due to the timing limitations of LISP on iPSC/2 only the real time was measured. The simulated speedup for Cafeteria is 6.13 and it achieved a speedup of 3.67 for 8 nodes. The simulated speedup for Hotel is 8 (only 8 nodes are available) and it achieved a speedup of 3.79. Since only eight nodes were available, the upward bound for speedup is eight. For this reason, the performances for Cafeteria and Hotel were quite close. But if more nodes were available, we expect the performance of Hotel to continue to increase with the number of the nodes while the performance of Cafeteria would stay the same.

The performance of Cafeteria using the round-robin partitioning, shown in Figure 4, is disappointing. However it is important to understand the cause in order to prevent it. The speedups dropped as more processing nodes were added. When rules in the same context are mapped to different nodes, they need to communicate with each other to perform the conflict resolution, the compatibility determination and the RHS actions. As the number of nodes increase, the number of messages increases. This is exactly what happened in the round-robin partitioning. As more nodes were used, the number of messages increased. In fact, the number of messages proliferated and the reduction in computation time was outweighed by the message communication time.

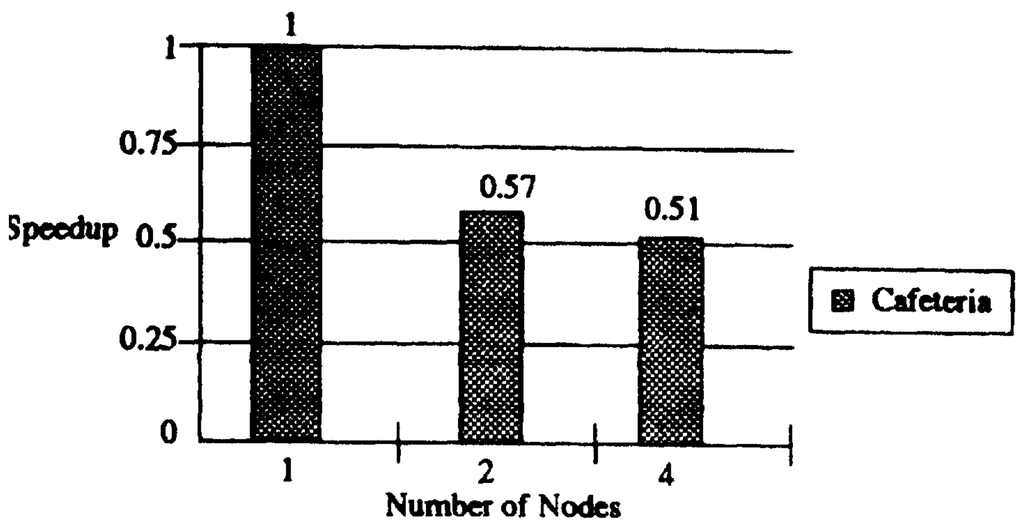


Figure 4: Performance for the Round-Robin Partitioning

Even though good speedups were obtained by allocating rules from a context to the same node, it is not clear whether this is the best partitioning. We intend to study the partitioning problem by developing an allocation algorithm based on simulated annealing. The allocation algorithm will read in the run-time information and use parallelism and communication matrices to estimate the computational and the communication costs for each partitioning. To accomplish this goal, we will also extend the timing functions to provide better run-time information.

5 Conclusion

In this paper, we have present three multiple rule firing models. The RDM model failed because it viewed a production program only as a collection of rules and did not consider the effect of multiple rule firing has on the problem solving strategy. On the other hand, the SCMR and MCMR models were successful because they addressed the multiple rule firing problems at both levels. Speedups of 3.18 to 8.94-folds were obtained for the test programs under the SCMR model. Speedups of 3.18 to 20.38-folds have been obtained for the MCMR model. The large speedups achieved under the SCMR and the MCMR models indicate that there is considerable amount parallelism in many production programs.

The execution of production programs on real parallel machines forces us to address the allocation and message communication problems. Production programs Cafeteria and Hotel have been executed on the Intel iPSC/2 hypercube using two partitioning schemes. When Cafeteria and Hotel are allocated by context, good speedups were obtained, but when Cafeteria was allocated in a round-robin fashion, the performance dropped as more nodes were added. We intend to use run-time traces and a simulated annealing method to further study partitioning. The run-time traces will be used to estimate the computation and the communication costs for different partitions. By comparing the simulated performance with the iPSC/2 performance for different allocations, the partitioning problem will be better understood.

References

[1] Acharya, A., Tambe, M. *"Production Systems on Message Passing Computers: Simulation Results and Analysis/"* Proceeding of International Conference on Parallel Processing, 1989.

[2] Chandy, K. M., Misra, J. *"Parallel Program Design: A foundation."* Addison Wesley, Reading, Massachusetts, 1988.

[3] Cunningham, H.C., Roman, G.-C. *"A UNITY-Style Programming Logic for Shared Dataspace Programs."* IEEE Transactions on Parallel And Distribued Systems, July 1990.

[4] Gamble, R. *"Transforming Rule-based Programs: form the sequential to the parallel."* Third Int'l Conference on Industrial and Engineering Applications of AI and Expert Systems, July 1990.

[5] Gupta, A., Forgy, C, Kalp, D., Newell, A., Tambe, M.S. *"Parallel OPS5 on the Encore Multimax".* In proceedings of the International Conference on Parallel Processing. August, 1988.

[6] Ishida, T. el at *"Towards the Parallel Execution of Rules in Production System Programs".* International Conference on Parallel Processing, 1985, 568-575.

[7] Kuo, S., Moldovan, D., Cha, S. *"Control in Production Systems with Multiple Rule Firings."* Technical Report No. 90-10. Parallel Knowledge Processing Laboratory, USC.

[8] Kuo, S., Moldovan, D., Cha, S. *"A Multiple Rule Firing Modele - The MCMR Model."* Technical Report. Parallel Knowledge Processing Laboratory, USC.

[9] Miranker, D.P., Kuo, C, Browne, J.C. *"Parallelizing Transformations for A Concurrent Rule Execution Language."* In Proceeding of the International Conference on Parallel Processing, 1990.

[10] Moldovan, D., Lee, W., Lin, C. *"SNAP: A Marker-Propagation Architecture for Knowledge Processing."* Technical Report No. 90-1. Parallel Knowledge Processing Laboratory, USC.

[11] Schmolze, J. *"A Parallel Asynchronous Distributed Production System."* Proceeding of Eight National Conference on Artificial Intelligence. AAAI90. Page 65-71.