The Role of Massive Parallelism in Parallel Inference Applications

Katsumi Nitta Stephen Wong
nitta@icot.or.jp wong@icot.or.jp
Institute for New Generation Computer Technology (ICOT), Japan*

Abstract

The research conducted by ICOT is firmly based on the paradigm of parallel logic programming. We have developed a fifth generation computer system (FGCS) prototype and evaluated its performance and appropriateness with applications from various domains. Our experience in this area so far indicates that the functions of the FGCS can benefit from the use of massive parallelism for computationally intensive tasks such as pattern matching and brute-force searching. The logical inference, however, should retain its control over the entire problem solving process. As an example, in this extended abstract, we provide a brief overview of two parallel inference applications; one in the domain of legal reasoning and one is a Go game playing program. We, then, describe how massive parallelism can play a role in enhancing the performance of these applications.

1 Introduction

The core of the FGCS developed at ICOT includes a parallel inference machine, PIM, a parallel logic language, KL1, and the operating system, PIMOS [5]. The PIM includes five hardware modules containing about 1,000 element processors in total. The design and the theory of these tools are firmly grounded on the paradigm of logic programming, in contrast to low-level connectionist and artificial neural network paradigms.

To evaluate the effectiveness of these tools for building large scale, practical AI systems, we have developed several knowledge based, parallel programs. The application domains of these programs are carefully chosen so that each program has to explore different knowledge processing techniques and algorithms for problem solving. Parallel AI applications developed in ICOT are as follows: (i) a logic simulator for LSI circuits, (ii) a LSI layout system, (iii) a protein sequence analyzer, (iv) a protein folding simulator, (v) a protein structure analyzer, (vi) a legal reasoning system, and (vii) a go-playing program. Recently, these applications have been demonstrated in the International Conference on Fifth Generation Computer Systems, 1992, held in Tokyo [9]. In this presentation, we provide a brief overview of applications (vi) and (vii).

2 Legal Reasoning System "HELIC-II"

Legal knowledge consists of statutory laws and old cases. Since a statutory law is a set of legal rules, inference by a statutory law can be realized as rule-based reasoning. Legal rules, however, often contain legal predicates (legal concepts). Some legal concepts are ambiguous in the sense that their strict meanings are not fixed until the rules are applied to actual facts. To apply legal rules to facts, rule interpretation and matching between legal concepts and concrete facts are needed. To realize this, old cases are referenced and their explanations are reused. Thus, legal reasoning systems are often modeled as a mixed paradigm of rule-based reasoning and case-based reasoning [1, 6].

There are some major difficulties in developing a practical legal reasoning system. First, it takes a long time to search for similar cases and to draw conclusions from them, as there are many legal rules and old cases. Second, a complex mechanism is required to manage several inference engines. To solve these problems by parallel inference, we developed a legal reasoning system, HELIC-II, on the 64-node module of PIM.

*The authors would like to thank the members of parallel applications laboratory at ICOT in this research.

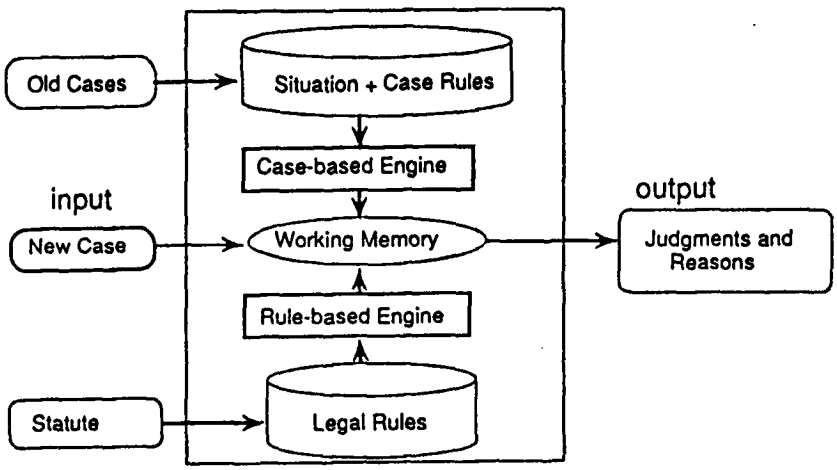

Figure 1: Architecture of HELIC-II

2.1 Overview

HELIC-II draws legal conclusions for a given case by referencing a statutory law and old cases and outputing them in the form of inference trees. The system consists of a rule-based engine and a case-based engine (Figure 1). The rule-based engine refers to legal rules and draws legal consequences logically. The case-based engine generates legal concepts from given facts by referring to similar old cases.

Rule-Based Inference Since there are many legal rules, a fast rule-based engine is needed. Moreover, there may be exceptions to the legal rules, so the rule-based engine must have the capability to handle nonmonotonic reasoning. The rule-based engine of HELIC-II is based on the parallel theorem prover MGTP (Model Generation Theorem Prover) [8]. Given a set of non-Horn clauses, MGTP generates models which satisfy all input clauses by parallel inference. To use MGTP as a rule-based engine of legal rules, and to obtain high performance by pipeline effect, we added several extended functions to the original MGTP.

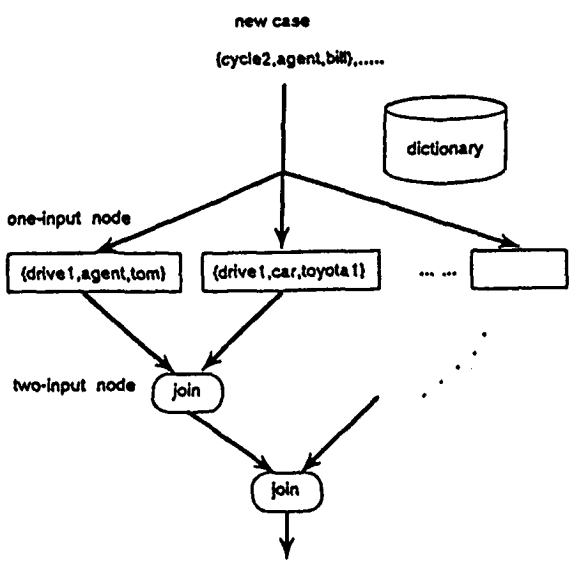

Figure 2: Rete-like network

Case-Based Inference A judicial precedent (old case) consists of arguments by both sides and the opinion of judges and a final conclusion. We represent an old case as a *situation* and some *case rules*. A *situation* contains

information on the occurrences of the case and represents a set of events/objects and their *temporal relations* in the form of semantic network. Arguments by both sides are represented as a set of *case rules*. The condition part of a case rule is a subset of the situation and the action part is a legal concept. A case rule is fired if its condition part is similar to a new case. The similarity is measured by mapping nodes and links of the semantic network of the condition part to those of a new case partially [3].

The function of the case-based engine is to generate legal concepts by referring to similar old cases. In the first stage, the engine searches for similar cases from the case base. Old cases are distributed to each processor (PE) of PIM and similarities between the new case and old cases are evaluated in parallel. In the second stage, similarities between case rules of selected cases and the new case are measured using a Rete-like network (Figure 2), and new arguments are constructed.

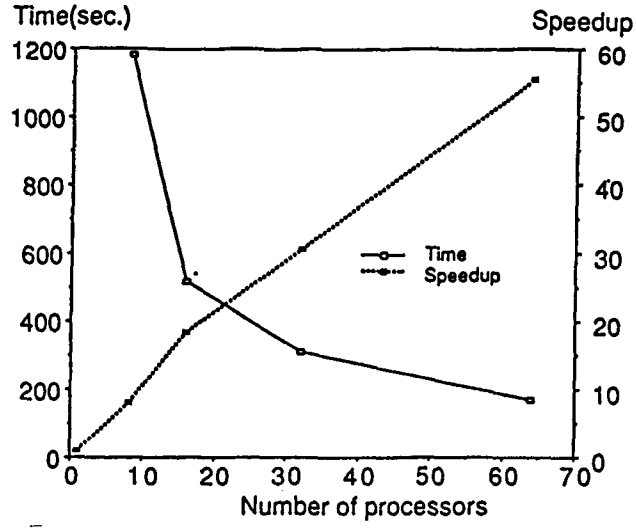

Figure 3: Performance of the case-based engine

2.2 Results

We observed that HELIC-II can solve several Penal Code cases. Figure 3 shows the speedup of the case-based engine at the second stage. We obtained more than 50-fold speedup using 64 PEs of the PIM.

3 Go Playing Game System "GOG"

Go is a popular board game played in the Far East. The game is played using black and white stones and a 19 × 19 grid. The two players alternately place black and white stones on the grid intersections. The goal is to gain more secure territories than your opponent. It is a zero-sum, perfect information game.

Go has long been a difficult game for computers to play. There have been no go-playing programs that match the ability of the average human go-player. The difficulty of constructing a go-playing program comes mainly from the fact that (1) the branching factor of an average game tree is too large for brute force searches to be feasible, and (2) a simple and good board evaluation function does not exist. The computational complexity of the computer go game is of several orders higher than that of computer chess.

Since a go-playing program requires basic AI techniques such as searching, processing ambiguous patterns, exceptional processing, and cooperative problem solving, it is a suitable research subject for knowledge processing technologies. In ICOT, we are building a strong go program using parallel inference algorithms and the computing power of parallel inference machines. We are aiming at GOG (GO Generation) with strength equal to the ability of the average human player.

3.1 Overview

GOG has the following three features.

1. It simulates the thinking mechanism of a human player.
2. It performs large grain tasks in parallel.
3. It applies new "flying corps" technique to improve the strength of GOG considerably while retaining its real-time response.

Simulating the Human Thinking Mechanism The process in which the GOG system determines its next moves comprises three stages (Figure 4). When the system receives the enemy's move, it first recognizes the board configuration. It then generates many candidate moves. It rates those moves and selects the one with the highest value as the next move.

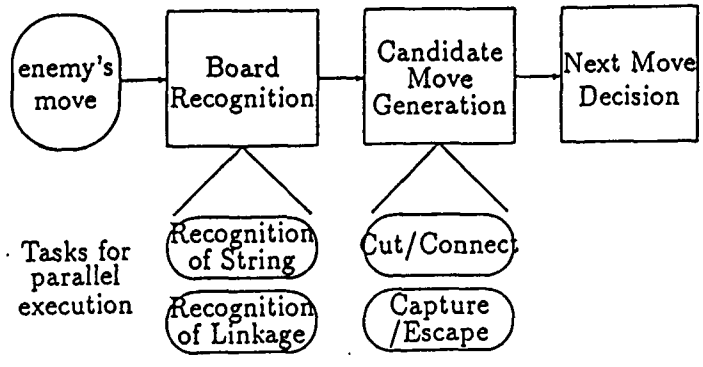

Figure 4: Outline of Process in The Parallel GOG

- Board Recognition The raw data of the board configuration is simply the state of every board position, which is either (a) vacant, (b) occupied by a white stone, or (c) occupied by a black stone. Just like a human player, the system starts from raw board data and successively makes higher-level data structures — stones, strings (connected stones of the same color), groups (strings of the same color that are close to each other), families (loosely connected groups), etc. The system then determines their attributes (potential value, area of surrounded territory, etc.) in the recognition phase.
- Candidate Move Generation The system has *candidate knowledge* which generates the coordinate and evaluation value of a candidate move. To decide the next move, many candidates are listed by executing tasks invoked from candidate knowledge. GOG has 12 kinds of candidate knowledge (e.g., JOSEKI, Edge, DAME, Invasion, Spheres' Contact Point, Capture/Escape, Cut/connect, Enclose/Escape).
- Next Move Decision The local adjustment for candidates rearranges disharmonies between the different candidate knowledges. Then, the system sums the total proposed values of candidates at each point on the board, and selects the one with the highest value as the next move and plays it.

Parallel Processing In GOG, one PIM processor serves as a manager processor, and the rest act as worker processors (may be scalable up to the full 1000 processing elements). The next move decision process is made on the manager processor, which also distributes tasks to worker processors.

When the system receives the enemy's move, it recognizes the board configuration and generates candidate moves. In those processes, it picks up large tasks such as local searches, which check whether a string to be captured or not, and dispatches the worker processors. The results are sent to the manager processor which, then, decides the next move based on those results.

Flying Corps To improve the strength of the system considerably while retaining its real-time response, we proposed the concept of *flying corps*. The idea is to find the tasks which are important but don't have to be solved before the next move and to make flying corps processes execute these tasks. The system which incorporates the flying corps idea consists of main corps processes and flying corps processes (Figure 5). A flying corps process and a main corps process are assigned to the same processor. Main corps processes consist of a manager and workers. Flying corps processes use the same manager and workers. Main corps processes execute necessary tasks to operate by go rules and tasks to maintain their strength.

Main corps processes have a higher priority than flying corps processes. Flying corps processes notify task completion to a flying corps manager process when the dispatched task is completed (which may be several moves after the initiation of the task). Whenever the main corps tasks are finished, the manager process of the main corps will collect the results of finished tasks on flying corps processes. With those results and the results by main corps worker processes, the system decides on the next move. The time to decide the next move depends only on the main corps processes.

Flying corps processes execute these tasks independently from the immediate next move decision process (in main corps processes). When the opponent is thinking of the next move, the flying corps processes keep on running. When a local situation, for which flying corps were assigned a task, is changed by some later move, these tasks are aborted.

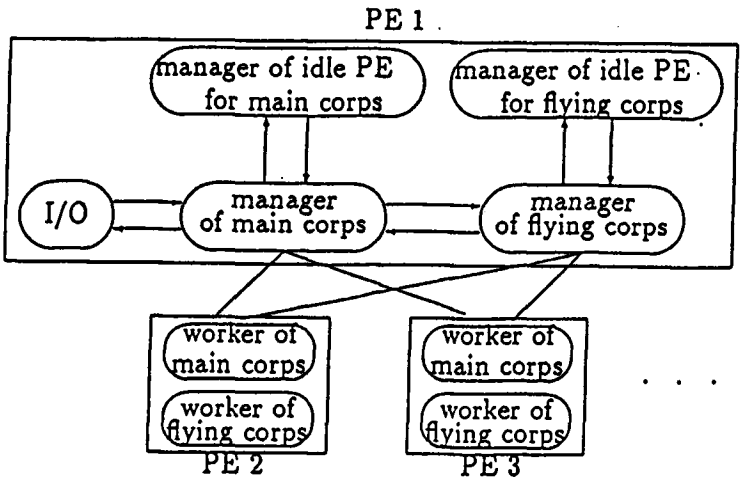

Figure 5: System Configuration.

3.2 Results

Table 1 shows the GOG's performance in parallel execution. These results show that parallel execution shortens the processing time in playing go. The strength of GOG, including the flying corps idea, is now under evaluation.

We have also been developing sequential GOG. The purpose is to test the new algorithm ideas of recognition, candidate knowledge, and next move decision. Last November, the sequential GOG and seven other computer go programs, including last year's top five programs, participated in a tournament at the Game Playing System Workshop [7]. The result of our GOG was 2 wins and 3 loses. This confirms that GOG is a top-class computer go-program. The current system is stronger than an entry level human go player, but considerably weaker than an average player.

4 Role of Massive Parallelism

Matching of Similar Rules The case-based inference in a single processor system normally involves many repetitive operations, such as searching for cases and matching similar rules in selected cases. These operations can be executed concurrently in a multiprocessor environment. The speedup of HELIC-II using PIM is linear when the number of processors is about the same as the number of old cases (see Figure 3).

Our study on HELIC-II reveals that case-based inference takes up most of the computational time (about 70% with 50 old cases in the case base). The system performance deteriorates when more cases are added. Most of the

Table 1: Speedup in Parallel Execution

1st of final match, 13th Kisei tournament			
Stage	1 PE	4 PE	16 PE
30th move	1.0	3.3	5.1
90th move	1.0	3.4	5.3
180th move	1.0	3.7	7.5
5th of final match, 13th Meijin tournament			
Stage	1 PE	4 PE	16 PE
30th move	1.0	3.2	5.4
90th move	1.0	3.4	5.6
180th move	1.0	3.6	5.9

time spent in case-based inference is at the second stage of matching for similar case rules. As the second stage is the partial matching of two semantic networks using the conceptional dictionary which contains hierarchical relations between concepts, the matching cost is very expensive. It is thus essential to remove this bottleneck. To this end, we see that the use of massive parallelism to solve such a problem is possible. The use of massive parallelism is related to the underlying knowledge representation formalism. HELIC-II, as with many case-based reasoning systems, encodes the knowledge about legal cases in a semantic network. Each case is composed of a set of legal concepts as nodes and a set of relations among these concepts as links. The inference also includes inheritance search. Such a network representation may be realized as a massively parallel network with simpler processing elements than PIM processors. This approach to inheritance and matching problems of semantic networks has been studied in [4, 11].

Note that we retain the rule-based engine for high-level cognitive tasks such as explanation and debate between the computer and the user.

Branching Search Factor A major problem faced in go-playing programs is that the branching factor of a search tree for Go is an order of magnitude greater than that for chess (around 150 versus 30). That is, a 7-ply search in Go would take 10^7 times as much effort as would one for chess. Many heuristics to reduce such searches have been studied, but so far their results have been poor. Simply judging by the successes in computer chess, *massive parallel search* would look like the best candidate. However, it seems unlikely that we will see massive parallel computers that are 10^7 times more powerful than today's for quite some time yet.

Another school of thought is that although Go and chess bear some resemblance, the perception of meaningful patterns is much easier in Go than it is in chess – the individual elements organize themselves readily before our eyes, thus, people are better at playing Go than they are at playing chess [2]. All it takes is an oversight in a tactical tree to lose a piece, and losing a piece in chess or Go is often decisive. Thus, a brute-force program win by not making the first oversight. If this theory holds, then the problem may not lie in the computational power, but in developing better knowledge representation languages to model human mental states.

5 Concluding Remarks

In this extended abstract, we have presented an overview of two AI systems, one in the domain of law and the other a go-playing program. Both were built using parallel inference technology developed here at ICOT. We have pointed out the need for a massive parallel capability in these systems, but this is mostly for speeding up low-level matching and searching tasks. Our position is that the control and coordination of the overall problem solving process should rest on high-level logical inference. The parallel inference remains the centerpiece of FGCS. We are interested in exchanging ideas about this view with researchers at the workshop.

References

[1] Ashley, K. D., *Modeling legal argument*, MIT Press, Cambridge, Massachusetts, 1990.

[2] Brown, D. J. H. "The challenge of Go," *Proc. Sixth IJCAI*, Tokyo, 1979, pp. 114-116.

[3] L.K.Branting "Representing and Reusing Explanations of Legal Precedents" *Proc. Int. Conf. on Artificial Intelligence and Law*, 1989.

[4] Fahlman, S. E., *NETL: A system for representing and using real-world knowledge*, MIT Press, Cambridge, MA, 1979.

[5] *Proc. Int. Conf. on Fifth Generation Computer Systems 1992*, ICOT Session, Vol. 1, ICOT, Tokyo, 1992.

[6] Gardner, A.v.d.L., *An artificial intelligence approach to legal reasoning,* MIT Press, Cambridge, Massachusetts, 1987.

[7] *Proc. of the Game Playing System Workshop*, Nov. 18-20, Tokyo, Japan, 1991.

[8] H. Fujita, et. al. "A model generation thereom prover in KL1 using a ramified-stack algorithm." ICOT TR-606 1991.

[9] K. Nitta et. al. "HELIC-II: A legal reasoning system on the parallel inference machine," *Proc. Int. Conf. on Fifth Generation Computer Systems 1992*, ICOT, Tokyo, 1992, pp.1115-1124.

[10] K. Nitta et. al. "Experimental parallel inference software," *Proc. Int. Conf. on Fifth Generation Computer Systems 1992*, ICOT, Tokyo, 1992, pp.166-190.

[11] Touretzky, D., *The mathematics of inheritance systems.* Morgan Kaufmann Publishers, San Mateo, CA, 1986.