ICOT Technical Report: TR-718

TR-718

Experimental Version of Parallel Computer Go-Playing System "GOG"

by
S. Sei, H. Oki, N. Sanechika,
T. Akaosugi & K. Taki

December, 1991

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Experimental Version of Parallel Computer Go-Playing System "GOG" [1]

Shinichi Sei†, Hiroaki Oki††, Noriaki Sanechika‡[2], Takashi Akaosugi†, Kazuo Taki†

† Institute for New Generation Computer Technology
1-4-28 Mita, Minato-ku, Tokyo 108, Japan
†† Future Technology Laboratories Inc.
‡ Electrotechnical Laboratory of the Agency of Industrial Science and Technology

abstract : We have been developing a parallel computer Go-playing system "GOG". Go has been a difficult game for the computer to play. We are trying to build a strong Go program using the computer power of the parallel inference machines. The first version was developed and reported [?]. A new version has been completed and tested the concept of *flying corps*, a technique for making a game playing program stronger without losing the real-time property.

1 Introduction

Unlike checker and chess playing computer programs which have attained or are approaching the highest human skills, there have been no Go-playing programs that match average human Go-player's skills.

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 search to be feasible -- the board of Go is 19×19 as compared to the chess board of 8×8, and the player can put the next stone on almost any vacant board position, and (2) a simple and good board evaluation function does not exist -- evaluation of a board configuration needs understanding of relative strengths of groups of stones, which involves pattern recognition.

We have developed a sequential Go-playing system in the Fifth Generation Computer System Project [?]. It is written in the Prolog-like language ESP and runs on the sequential inference machine PSI [?]. Currently, the system is stronger than an entry level human Go player, but considerably weaker than an average-level player.

There are a number of improvements (such as knowledge of set moves, tactics, better board evaluation, etc.) that could make the system stronger, but it would take too much processing time to incorporate them. Thus we started the development of a parallel Go-playing system which will be stronger than the sequential system but will retain a tolerable response time to make real-time play with humans possible. The system runs on the experimental parallel inference machine Multi-PSI [?, ?]. It is consist of the parallel code (written in parallel logic language KL1) for deciding the next move and the sequential code (that runs on the front-end processor (FEP)) for human interface. The final version will run on the parallel inference machine being developed at ICOT [?].

This paper first describes the process of move making in the sequential system, and discusses what parallelism to exploit in the parallel system. It then describes the parallel version and the concept of *flying corps* tested in the system.

2 The Sequential GOG System

We have been developing a sequential computer Go playing system called "Sequential GOG" on the sequential inference machine PSI since 1985. The current system is stronger than human Go

[1] "GOG" is stand for "GO Generation". "GO" means both "Go Game" and "5" in Japanese. It has been developing in 5th Generation Computer System Project.

[2] Currently at AI Language Research Institute Ltd. (AIR)

beginners, but is considerably weaker than an average-level amateur.

2.1 Move Making in the Sequential GOG System

Our system's approach is based on simulating the human player's way of thinking as faithfully as possible. The outline of the process in which the sequential GOG system determines its next moves comprises three stages.

1. Board Recognition
2. Candidate Move Generation
3. Next Move Decision

When system get enemy's move, it first recognizes board configuration. Based on the results of recognition of the board configuration, it generates many candidate moves. It rates those moves and selects the one with the highest value as the next move.

2.1.1 Board Recognition

When a player looks at the "Go" board, he sees not only the arrangement of stones on the board, but also sees their strategic meaning.

The raw data of 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 human player, the system starts from the raw board data and successively makes higher-level data structures — stones, strings (a string is connected stones of the same color), groups (strings of the same color that are close to each other), families ("loosely" connected groups), etc. —, and then determines their attributes (potential value, area of surrounded territory, etc.) in the recognition phase.

These objects in the data structure are constructed bottomup : point, string, group and family. This is basically the same as the process of human concept formation from concrete to abstract. Furthermore, we can find important area and stones through this deepening understanding.

2.1.2 Candidate Move Generation

The system has "Candidate Knowledge" which generate coordinate and evaluation value of candidate move. To decide the next move, many candidates are listed by execution of tasks invoked from Candidate Knowledge. Candidates knowledge which the sequential GOG has is shown in Table 1[?].

2.1.3 Next Move Decision

First of all, the local adjustment for candidates rearranges disharmonies between the different candidate knowledges. Next, the system sum up total proposed value of candidate at each point on the board. The system selects the one with the highest value as the next move, and play it.

3 The parallel "GOG" System

3.1 Design of the Parallel GOG System

The various tasks done by the sequential GOG system contain possibilities of various forms of parallel processing.

- Independent parallel recognition of distinct objects (strings, groups, etc.)
- Independent parallel capture searches of distinct strings
- Parallel processing within a search
- Parallel generation and rating of candidate moves

Table 1: Candidates

Task	characteristic	Contents
JOSEKI	a standard pattern of good play in the corner	JOSEKI (about 200 kinds)
Edge	a pattern along with the edge	extension, pincer, invasion into edge, etc
DAME	touched stones pattern	stone's competition move (HANE,NOBI,OSHI, etc)
Invasion	a pattern in the corner	Invade/protect at corner
FUTOKORO Expand/Reduce	edge pattern	expand/reduce territory in edge for a weak stone
Spheres' Contact Point	touched families	Expand/prevent MOYO move (RYO-GEIMA, etc)
Capture/Escape	String (3 DAMEs or fewer)	Capture/escape move
Cut/connect	weak linkages	Cut/connect move for peeping
Enclose/Escape	Slightly enclosed weak friend group	Enclose/escape move for weak group
Separate/Contact	a big separatable group	Separate/contact move
Sphere	a family with weak bordered	Enclose/prevent move
Life-and-Death/ Capturing Race	a race to capture between two vulnerable groups	a vital point for making two eyes or filling opponent's DAME

Also, some of the tasks, that could potentially strengthen the sequential GOG system but were not incorporated because of processing time limitation, can be incorporated in the parallel GOG system, and they give more parallelism.

3.2 Machine Environment

The Multi-PSI is an experimental parallel inference machine. It is a distributed-memory computer, in which up to 64 nodes (processors plus local memory) are connected by an 8×8 mesh network with worm-hole routing.

3.3 Parallel Processing

In the parallel system, one of the processors of the Multi-PSI serves as a manager processor, and the rest does as worker processors. Next move decision process is on the manager processor, and manager processor distributes tasks to worker processors.

The outline of process in the parallel GOG system is shown in Figure 1. When the system gets enemy's move, it recognizes board configuration and generate candidates moves. In those processes, it picks up large tasks such as local search which check string to be captured or not and dispatches to worker processors. The results are sent to the manager processor and then it decides the next move based on those results.

3.4 Devices on Parallel GOG System

- Interprocessor communication

  Generally, in programming a parallel system, we have to keep the interprocessor communication down to a minimum. In order to realize it, we designed that each worker processor maintains a local copy of the board and updates it each time the manager processor notifies a new move. In the board recognition or candidate move generation phase, worker processors need only which is the target of task to execute those tasks. It don't cause to concentrate message from worker processor on manager processor. This reduces interprocessor communication.

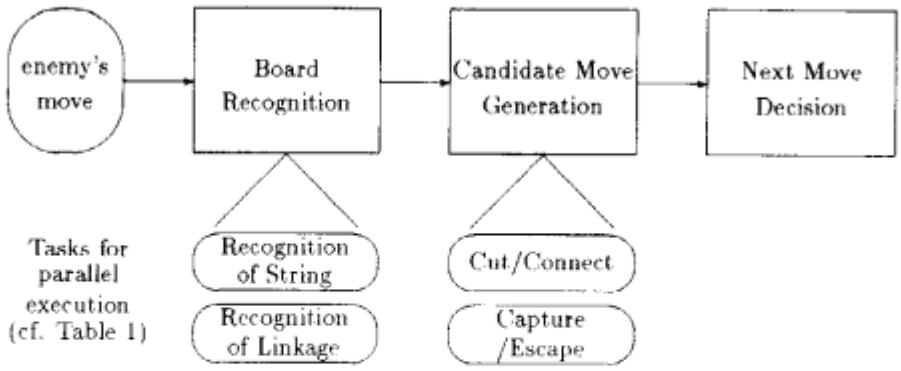


Figure 1: Outline of Process in The Parallel GOG

- Load Balance

  Good load balance among processors is a key to get high processor utilization. We used the dynamic load balancing technique which is one of good ways to realize the load balance. In our system, manager processor have a process which observe worker processors whether are in idle. The manager processor dispatches tasks to worker processors that are detected to be idle.

3.5 Flying Corps

The Parallel GOG system can be made stronger by incorporating more tasks, evaluating candidate moves more precisely, etc. But, that would increase the time to decide the next move, and the system might become too slow as real time game, even when parallel processing would be incorporated. To improve strength of the system considerably while retaining the real-time response of it, we propose the concept of *flying corps* which has been tested in the system.

This idea is to find the tasks which is important but not necessarily solved before next move and to make flying corps processes to execute these tasks. The system which incorporates flying corps idea consists of main corps processes and flying corps processes (Figure 2). A flying corps process and a main corps process are assigned on a same processor. Main corps processes consist of a manager and workers and flying corps processes are the same with them. Main corps processes executes necessary tasks to operate by Go rule and tasks to keep certain strength.

Main corps' processes have higher priority than flying corps' processes. Flying corps processes notify the task completion to a manager process of flying corps when the dispatched task is completed. (that might be several moves after the initiation of the task). Each time all tasks of main corps are finished, the manager process of main corps will collect the results of finished tasks on flying corps processes. With those results and the results by worker processes of main corps, the system decide the next move. The time to decide next move depends only on 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. [3] When a local situation which caused tasks for flying corps will be changed by some later move, the these tasks will be aborted.

This idea's merits are as follows.

- Flying corps processes do their tasks independently from the main move deciding procedure, and they don't obstruct main corps processes.

[3] This is common with a human player to consider such tasks which flying corps processes execute

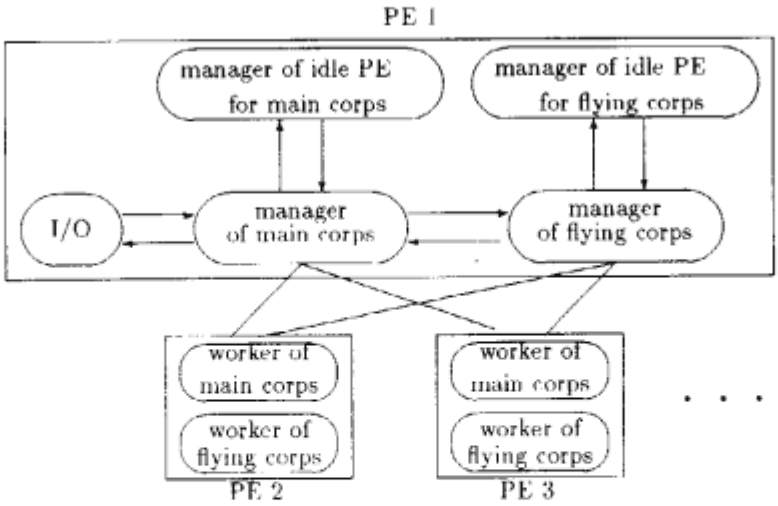


Figure 2: Configuration of System

A processor have a main corps process and a flying corps process. The main corps process have higher priority than flying corps process. The flying corps processes start to run when tasks of main corps is finished. Thus processor which finished tasks of main corps don't change into idle. Main move deciding procedure need not wait for results of flying corps processes. Therefore the time to generate the next move won't be increased in spite of the increasing process. It retains the real-time response with the system.

- The more idle processors there are, the smarter move parallel GOG can produce.

  A flying corps process and a main corps process are assigned on a same processing element. Flying corps process runs on the processor which there have no task to be executed by main corps process. When opponent takes long thinking time or the parallel machine have many processors, there are many idle processor. Thus many tasks of flying corps are execute and the system produce smarter move.

3.6 Experimental Results and Consideration

First, we programmed the parallel GOG system without flying corps idea. Table 2 shows its performance. From these results, the parallel execution shortens the processing time in Go.

Table 2: 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

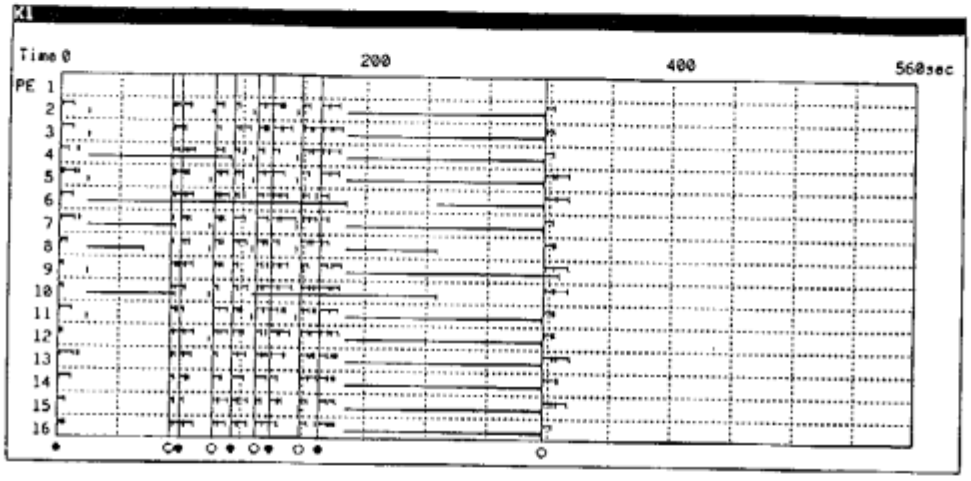
Next, we made the system including flying corps which compute capture search of string with 4 DAMEs. The capture search is the task which decides whether the string is in danger by

being captured. Capture search with 4 DAMEs is so large task that we didn't incorporate in the sequential GOG. Table 3 shows the comparing executing time with GOG including flying corps and without flying corps using 16 PEs. The generating time for the next move of the parallel GOG with flying corps did not increase in spite of the increasing process.

Table 3: Comparing executing time (sec) with GOG including flying corps and without flying corps using 16 PEs

	Data 1			Data 2		
Stage	30th move	90th move	180th move	30th move	90th move	180th move
Without flying corps	10.91	12.43	13.05	5.590	7.674	11.68
With flying corps	10.90	12.54	13.65	5.678	7.716	11.74

The origin : Data 1 is quoted from 1st of final match in 13th Kisei tournament and Data 2 is done from 5th of final match in 13th Meijin tournament.



Upper line represents time corresponding to a worker of main corps execution in each processing element, while lower line to a flying corps. PE 1 is assigned manager process. Virtical line represents when move is placed on board. In this execution, white side is human player and black's one is parallel GOG.

Figure 3: Log of Execution on every processing element

Figure 3 shows log of execution on every processing element. Upper line represents time corresponding to a main corps execution in each processing element, while lower line to a flying corps.

It shows that a processor executes both tasks of flying corps and main corps, and tasks of flying corps are executing over several moves. We can also find flying corps processes is being executed during main corps process are waiting task.

The strength of the system, including flying corps idea, is under evaluation. The parallel system handles more tasks than sequential GOG, thus we have to adjust evaluation values of candidate move to be well balanced.

4 Conclusion

We have developed the experimental version of parallel computer Go-playing system "GOG" and tested a technique of flying corps on the system. Our experiments represent the system using the

technique retains the real-time response in spite of incorporating much more processing.

We will incorporate more tasks gradually to make GOG stronger using the flying corps technique. We plan to incorporate much larger tasks such as Tsumego using the technique of flying corps. Tsumego is too large to execute in main corps, and it is able to execute independently from next move deciding procedure. Tsumego would require long processing time, even though flying corps is incorporated. We will consider dividing a large task into small subtasks and distribute to processors. Also, we consider that, if flying corps execute the task which have intermediate results such as search and have some effective information to decide next move on the working, flying corps processes report those information to main move deciding procedure.

We will improve the parallel GOG to aplly with those flying corps ideas.

References

[1] A. Goto, M. Sato, K. Nakajima, K. Taki, and A. Matsumoto: Overview of the parallel inference machine (PIM) architecture, In *Proceedings of FGCS'88*, pp. 208–229 (1988).

[2] K. Nakajima, Y. Inamura, N. Ichiyoshi, K. Rokusawa, and T. Chikayama. Distributed implementation of KL1 on the Multi-PSI/V2, In *Proceedings of the Sixth International Conference on Logic Programming*, pp. 436–451, 1989.

[3] H. Nakashima and K. Nakajima. Hardware Architecture of the Sequential Inference Machine PSI-II. In *Proceedings of 4th Symp. on Logic Programming*, pp. 104–113, 1987.

[4] N. Sanechika. "Go Generation": A Go Playing System. ICOT Technical Report TR-545, 1990.

[5] N. Sanechika, S. Sei, T. Akaosugi, K. Taki, S. Yoshikawa, T. Yoshioka, Y. Murasawa, S. Uchida, H. Oki, M. Ohshima, M. Ogiso, Y. Mizuno and J. Sakamoto. The Specifications of "GO Generation. In *Proceedings of Game Playing System Workshop*, 1991.

[6] S. Sei, N. Ichiyoshi and K. Taki. Experimental Version of Parallel Computer Go-Playing System "GOG". In *Proceedings of International Workshop on Parallel Processing for Artificial Intelligence in IJCAI'91*, pp. 184–190, 1991

[7] K. Taki. The Parallel Software Research and Development Tool: Multi-PSI System, In *Programming of Future Generation Computers*, K. Fuchi and M. Nivat (eds.), Elsevier Science Publishers B.V. (North-Holland), pp. 411–426, 1988.