ICOT Technical Report: TR-669

TR-669

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

by

S. Sei, N. Ichiyoshi & K. Taki

July, 1991

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

(Extended Abstract)

Shinichi Sei
sei@icot.or.jp

Nobuyuki Ichiyoshi
ichiyoshi@icot.or.jp

Kazuo Taki
taki@icot.or.jp

Institute for New Generation Computer Technology
1-4-28 Mita, Minato-ku, Tokyo 108, Japan

Abstract

We are developing a parallel computer Go-playing system "GOG". The experimental version has been completed and a dynamic load balancing technique is being tested. The next version will incorporate the concept of *flying corps*, a technique to make 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 basics of Go is given in the appendix.)

The difficulty of constructing a Go-playing program comes mainly from the fact that (1) the fanout 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 [3]. 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 (one who knows a number of established tactics such as opening moves, and has played tens of games with stronger players).

There are a number of improvements (such as move knowledge of set moves, tactics, better board evaluation, etc) that could make the system stronger, but it would take much more 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 the tolerable response time to make real-time play with humans possible. The final system will run on the parallel inference machine being developed at ICOT [1]. It will 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. We have redesigned part of the existing sequential program in KL1, and constructed an experimental version (intermediate system) in which both the sequential and parallel code take part in the next move decision. The system runs on the experimental parallel inference machine Multi-PSI [4, 2].

This extended abstract 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 experimental version and the load balancing technique employed in the system with its results. Lastly, the concept of *flying corps* to be tested in the next version is described.

2 The Sequential GOG System

2.1 Move Making in the Sequential GOG System

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

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. In the recognition phase, 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.).

Based on board recognition, the system lists up plausible next moves together with heuristic value of those moves. Candidate moves include moves to enlarge a friendly territory, moves to limit an opponent territory, moves to capture an opponent string (we call the task which decide whether the string is in danger by being captured *capture search*), moves to avoid capture of a friendly string, etc.

Finally, based on the proposed values of the candidate moves, the system decides on the next move, and plays it.

2.2 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 of single capture search
- Parallel generation and rating of candidate moves

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 added parallelism.

3 Intermediate System

The intermediate parallel GOG system consists of the parallel code that runs on the Multi-PSI and the sequential code that runs on the front-end processor (FEP).

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 cut-through routing. A FEP is a sequential inference machine.

In the intermediate system, one of the processors of the Multi-PSI serves as a manager processor, and the rest are worker processors. Each worker processor maintains a local copy of the board, and updates it each time the manager processor notifies a new move. This reduces interprocessor communication, since the master processor needs only to specify which string is the target of the search.

The system configuration is shown in Figure 1. In this partly parallel GOG system, the front-end sequential GOG system notifies the enemy's moves to the Multi-PSI manager processor. The manager processor dispatches the capture search task to the worker processors. (The capture search tasks are generated only by the dangerous string in order to keep executing time short.) When a capture search is completed, a worker processor requests for a next capture search task. After all capture search tasks of dangerous strings have been dispatched, a particular kind of plausible move generation tasks are dispatched. Those are *keshi* candidate moves that may restrict the enemy's potential territories. The results are sent to the manager processor and then to the front-end processor. After the front-end sequential GOG system have received the candidate moves from the Multi-PSI, it evaluates them with the candidates generated by itself. It then decides the next move.

4 An Experiment of a Load Balance Technique

Good load balance between processors is key to high processor utilization. The dynamic load balancing technique is one of way to realize it. It is that the manager processor dispatches tasks to worker processor which are detected to be idle. But, even if we use the technique, load imbalance is caused by uneven granularity.

To alleviate this problem, we devised the following technique. At first, we classify all of tasks into two groups: a task group with larger granularity and a task group

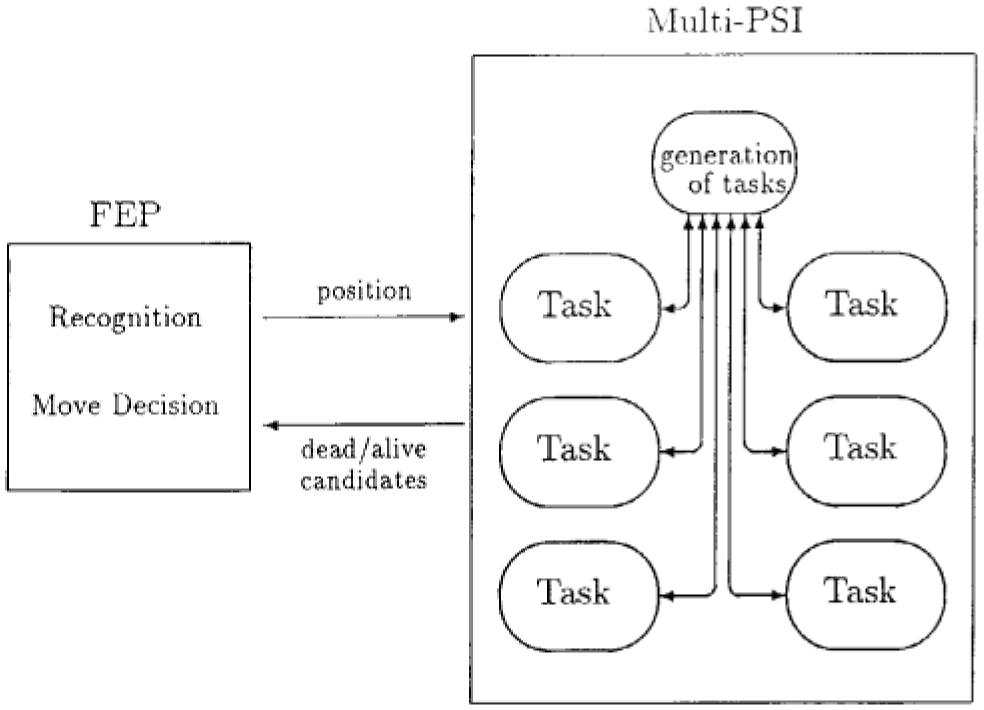

Figure 1. System Configuration

with smaller granularity. Larger tasks are given higher priority than smaller tasks. Therefore after all larger tasks have been dispatched, smaller tasks are dispatched. Then the smaller tasks tend to smooth out the load imbalance caused by uneven granularity of larger tasks (Figure 2). This makes processor utilization rate higher.

We tested this technique on the experimental GOG system. In the intermediate system, the larger-grain task group consists of capture search tasks, and smaller-grain task group consists of keshi candidate move's, because the former tasks are usually much larger than the latter.

We give the measurement results in following graphs (Figure 3).

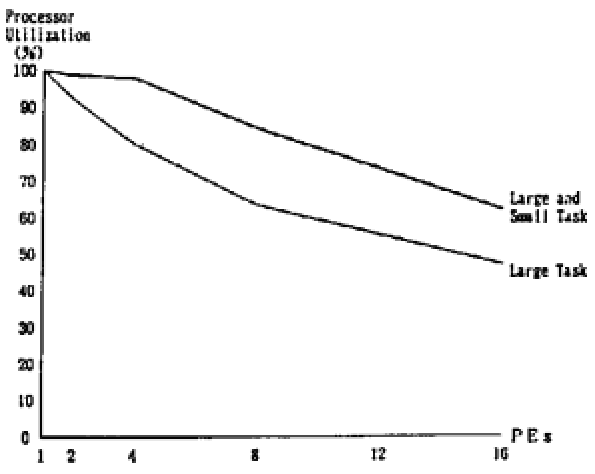

Figure 3. Processor Utilization

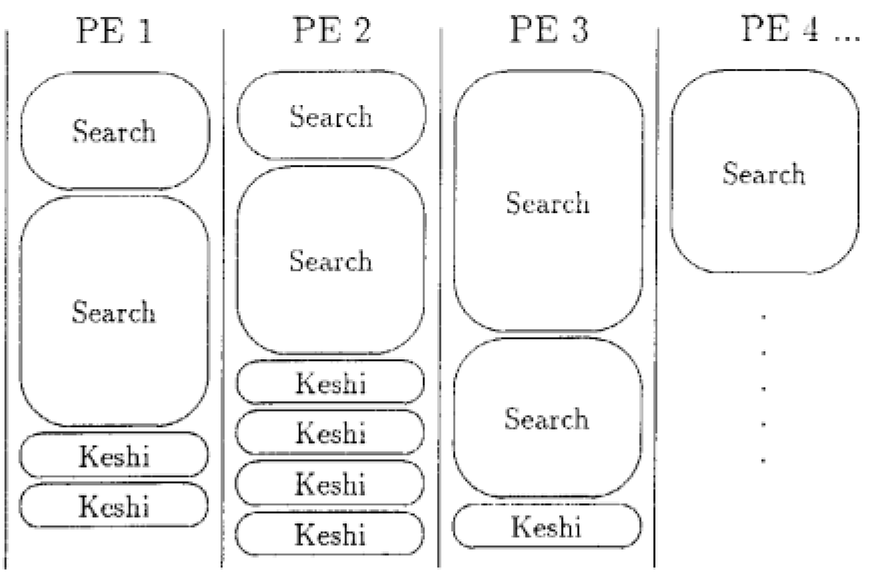

Figure 2. The smaller tasks tend to smooth out the load imbalance

In general, this technique can be employed when (relative) task sizes can be roughly estimated in advance.

5 Flying Corps

The parallel GOG system can be made stronger by incorporating more searches, making more precise evaluation of candidate moves, etc. But, that would lengthen the time to decide the next move, and even the parallel system might become too slow for real-time game playing. To make the system considerably stronger while retaining the real-time response of the system, we propose the concept of *flying corps*.

The idea is to find out the possibilities of potentially large gain (such as capturing a large opponent group, invasion of a large opponent territory) or loss, and assign the investigation of those possibilities to *flying corps* processes. The future system which incorporates flying corps idea is consist of *main corps* processes and flying corps processes (Figure 4). Main corps treats necessary tasks to play Go and to keep strength standard level we can permit. A flying corps does the investigation independently from the immediate next move decision process, and it notifies the main corps when the investigation task is completed (that might be several moves from the initiation of the task). Note that flying corps keep on running, while the opponent is thinking of the next move.[1] A flying corps may be aborted if it has become irrelevant or unimportant in the overall situation, or the local situation that motivated it has changed by some later move. Main corps processes have higher priority than flying corps processes. The time to decide next move depends only on

[1]This is common with a human player

main corps.

A well-known technique in game tree search of doing whatever can be done in a limited amount of time is iterative deepening search. Flying corps is different from iterative deepening in that (1) it is independent from the main move deciding procedure, that (2) it extends more than one move, and that (3) it is aborted by reasons that are not solely time-related.

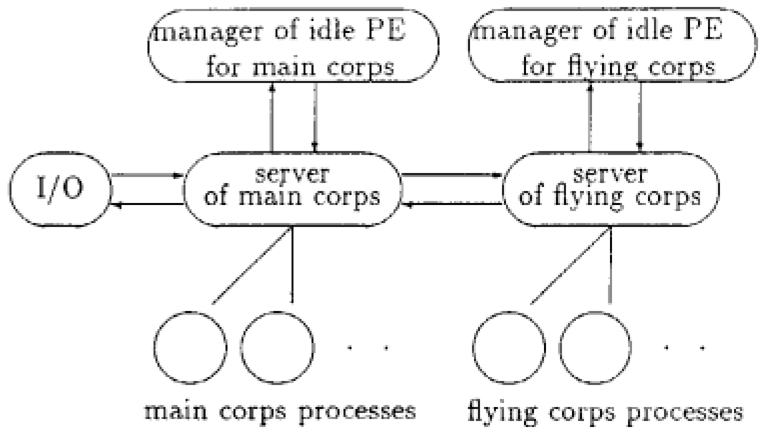

Figure 4. Configuration of Future System

The initial design of flying corps is completed. We would like to test the utility of flying corps in the next version of GOG which is currently under development. The candidate tasks for flying corps are large scale tasks like *Tsumego* (A Tsumego task is the same as a capture search task, except the object can be a group or a family).

6 Conclusion

We have developed the experimental version of parallel computer Go-playing system "GOG" and tested a dynamic load balancing technique on the system. Our experiments show that our technique minimized imbalance caused by uneven granularity. We also described the flying corps idea.

We will develop flying corps. A new version of the "GOG" system is currently under development. It will include an experimental flying corps mechanism. We plan to demonstrate the final version in the FGCS'92 conference.

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

[4] 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.

Appendix: Basics of Go

Go is a board game, and is popular in China, Korea and Japan. The board is a 19×19 grid. The two players are named black and white. The black and the white place a stone on a vacant intersection in turn (the black and the white place black and white stones, respectively). Each player tries to gain as much territory as possible (a black's (white) territory is a vacant area surrounded by black (white) stones). A group of solidly connected stones of the same color is captured when all adjacent positions are occupied by the opponent's stones. The adjacent positions that are vacant are called *dames*. When stones are captured, they are removed from the board and are added to the opponent's territory count. Thus, a player places stones as efficiently as possible to surround vacant area to maximize his/her territory. Inevitably, the black and white stones clash, which leads to compromise or fight for capturing the opponent's stones.