TR-696

Parallel Randomized Search for Distributed Memory Machines

by
N. Iwayama & K. Satoh

October, 1991

© 1991, ICOT

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

Parallel Randomized Search for Distributed Memory Machines

Noboru Iwayama, Ken Satoh
Institute for New Generation Computer Technology
4-28 Mita 1-Chome Minato-ku, Tokyo 108 Japan
email: iwayama@icot.or.jp

September 3, 1991

Abstract

For the computation on parallel machines, each processor should be given even computational load. Generally speaking, there are two types of load balancing methods: dynamic ones and static ones. Static methods need the estimation of the task size and the communication cost, which is difficult in most cases. For Dynamic methods, communication problems should be resolved for efficient computing especialy on distributed memory machines.

We give a different type of load balancing method for problems searching a single solution on distributed memory machines. That is a parallelization of randomized search. Our method requires no communication during computation, and all processors are always busy. This parallelization on distributed memory machines is profitable for the problems which the above methods don't handle well.

1 Introduction

It is essential for the computation on parallel machines that processors don't become idle during computation. Each processor should be given even computational load to keep it busy. Generally speaking, there are two load balancing methods: a dynamic one (ex. [1, 2]), which dynamically distributes tasks to idle processors, and, a static one (ex. [3]), which gives each processor subtasks into which the whole task is divided in advance. Static methods are efficient when the task size and the communication cost between tasks can be estimated, however estimation is difficult in most cases. If you adopt dynamic methods, communication problems should be resolved for efficient computing. Since distributed memory machines have more expensive communication costs than shared memory machines, load balancing should be with less communication overheads in order to compute efficiently with distributed memory machines.

In this paper we give a different type of load balancing method for problems searching a single solution on distributed memory machines. That is a parallelization of randomized search. Our method requires no communication during computation, and all processors are always busy. This parallelization on distributed memory machines is profitable for the problems which the above methods don't handle well.

In section 2 and 3 we give two methods to parallelize randomized searches. In section 4 we compare our method with the parallelization of randomized backtracking, and we show conclusions and future work in section 5.

2 Parallel Randomized Search

For distributed memory machines, load balancing methods should not cause much communication, because the communication costs are more expensive than computation costs.

At first some static load balancing method, which only needs communication with each processor at the beginning to distribute divided tasks, should be tried. Unless tasks are divided equally based on the estimation size of the whole task, some processors easily fall into the idle state. Since it is difficult to estimate a task's size generally, static methods with distributed memory machines are not good for all problems. Next a dynamic load balancing method may be tried for the problems which static methods are not good at. When the problem needs a lot of communication, compared with the computation time, this communication becomes overheads and makes some processors idle. Even if both static and dynamic methods don't compute some problems quickly, we want to get a solution to those problems quickly with distributed memory machines.

If each processor of a parallel machine independently resolves the problem using different strategies to search a single solution, the search is done in the time taken for the fastest processor. This idea is suitable for distributed memory machines from the aspect of communication cost, because no communication is needed during computation except for the copy of the initial state of the problem at the beginning. We provide parallel search methods in this paper on the basis of this idea.

Here, we explain how each processor searches with our method. Each processor resolves the same problem with a randomized search. We mean the randomized search as a following naive search method for getting a single solution: at each point of choice each processor chooses at random from the candidates, and it retries from the beginning on failure until a solution is found.

Next we state how a parallel search method as a whole. With our method each processor searches at random. Since that randomness comes from the random number generator, each processor uses a different random number generator in order for each processor to resolve the problem using a different strategy. Therefore, each processor can choose independently from search candidates. As soon as one processor gets an answer, that processor instructs the other processors to stop their searching. We call this parallel search method the parallel randomized search.

After the initial state is copied, the parallel randomized search needs no communication until the answer is found. And the total computation time is the time taken for the fastest processor.

However there is a possibility of repeated choice of paths that result in failure. We should analyze the time complexity to investigate the ability of the parallel randomized search.

2.1 Time Complexity of Parallel Randomized Search

Since it takes different times to compute even the same problem because of randomness, we must consider the expected time complexity of the parallel randomized search.

Let p be the existential probability of a solution in a search path. Since the expected trials for getting one solution with one processor (i.e. failure occurrences $+1$) is $1/p$, T_1, the expected time for getting one solution with one processor is t/p if the time for one trial to result in failure or success is constant (t) for every path. By n processors, the probability of getting one solution at some processor is $p_n = 1 - (1-p)^n$, and the expected trials for getting one solution

Figure 1: Speed up ratio of 8queen problem

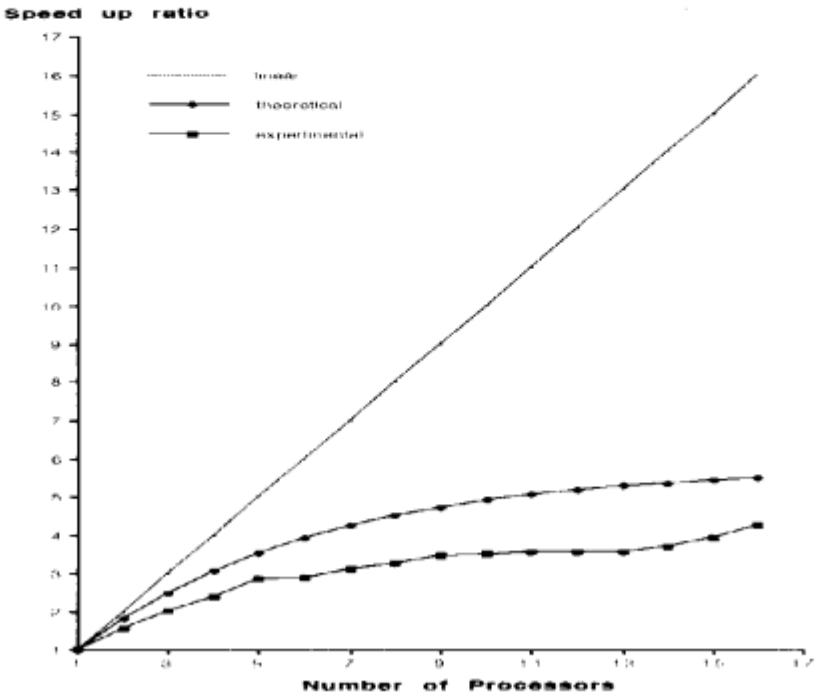
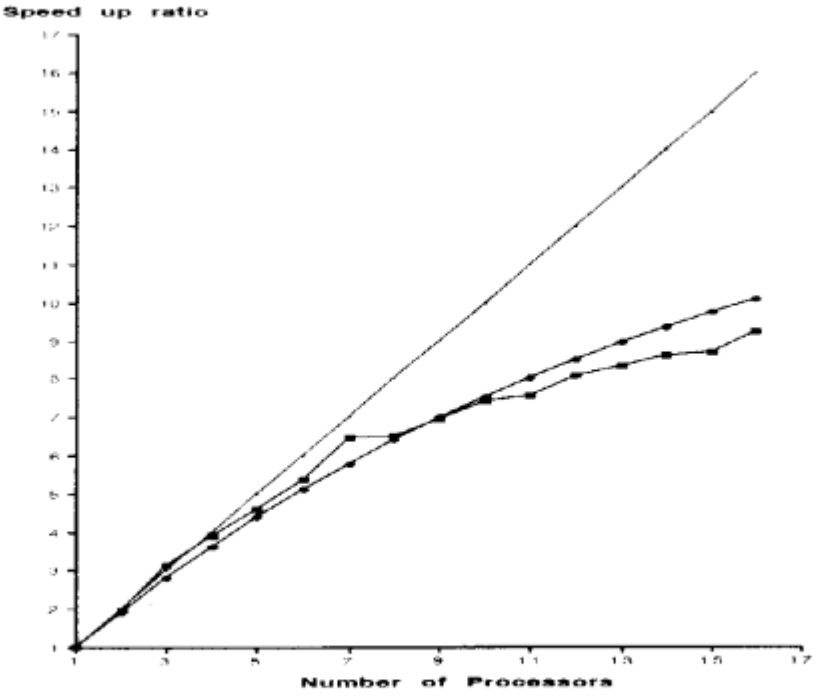


Figure 2: Speed up ratio of 11queen problem



is

$$\frac{1}{1-(1-p)^n}$$

Therefore, T_n, the expected time for getting one solution by n processors is

$$\frac{t}{1-(1-p)^n}$$

The speed up ratio is

$$\frac{T_1}{T_n} = \frac{1-(1-p)^n}{p}$$

In the case where there are few solutions in the search space, namely, $p \approx 0$,

$$\frac{T_1}{T_n} \approx n$$

This means there is no repeated choice of failure paths by n processors.

In the case of $p \approx 1$,

$$\frac{T_1}{T_n} \approx 1$$

means that when a solution is almost found on one trial, the speed up ratio is bad. In this case even one processor gets an answer quickly because the problem is very easy.

Moreover,

$$\begin{aligned}\frac{T_n}{T_{2n}} &= \frac{1-(1-p)^{2n}}{1-(1-p)^n} \\ &= 1+(1-p)^n\end{aligned}$$

means that speed up is $1+(1-p)^n$ though twice the number of processors is used. This shows that the repeated choices increase as the number of processors increases.

2.2 Experiments on MultiPSI Machine

We consider the Truth Maintenance System [1] as an experimental problem for the parallel randomized search. TMS consists of justifications which express reasons for beliefs and derives a rational state of belief from justifications. Some researchers in the AI field are devoted to studying TMS for hypothetical reasoning or non-monotonic reasoning. Since parallel TMS algorithms in [5, 6] are based on shared memory machines as the underlying architecture, the algorithms are not suitable for distributed memory machines.

We have implemented a non-deterministic TMS algorithm [7] with the parallel randomized search on the mesh-connected multiprocessor Multi-PSI [8]. We experimented on TMSs expressing the 8 and 11 queen problems with 16 processors. Figure 1, 2 shows the theoretical and the experimental speed up ratios. We had to take an average time of 100 executions because of randomness of the randomized search. And we used an experimental value of p to get the theoretical speed up ratio.

In 11 the queen problem the experimental result corresponds well with the theoretical one. In the 8 queen problem the experimental result doesn't correspond to the theoretical one. When over 10 processors were used for the 8 queen problem, a solution was almost found on 1 trial (for about 1 second). Although the work of each processor should be halted as soon as a solution is found, the work cannot necessarily be halted immediately because of our coding style. This overrun is not small compared when with 1 second, so it affects the total time of the 8 queen problem.

3 Combination of Randomized and Static Methods

Because all processors search the whole space independently with the parallel randomized search, there is a possibility of repeated choices of search paths by different processors. To cover this defect, we provide a combined method of the parallel randomized search with a static load balancing method.

By the combined method the search space is divided into subspaces so that each processor may search the only given subspace at random. We call this method the divided method.

Let's consider a situation where ten people search a ten room house for a needle.[1] By the former method (non-divided method) each person searches the whole house, however by the divided method each person searches a different room and each room is searched. Both are the same except that each processor searches the whole house or a room in the house. By the divided method we expected computation to be faster than computation by non-divided method, since the processor which is assigned to a subspace with a high existential probability of solutions can find a solution quickly. We show in the following that the method which is faster depends on divisions.

3.1 Comparing Methods with respect to Time Complexity

In this subsection, we compare the divided method with the non-divided method with respect to their time complexities.

Let's divide the search space of a given problem into n subspaces $S_1, \ldots, S_n$, and let the existential probabilities of a solution in a search path for subspaces be $q_1, \ldots, q_n$ respectively. If we choose, by non-divided method, a subspace S_i with probability $\alpha_i = \frac{1}{n} + \varepsilon_i$($-\frac{1}{n} \leq \varepsilon_i < 1 - \frac{1}{n}$, $\varepsilon_1 + \cdots + \varepsilon_n = 0$), the probability for getting a solution at each processor, p, is

$$\begin{aligned} p &= \sum_{i=1}^{n} \alpha_i q_i \\ &= \frac{1}{n} \sum q_i + \sum \varepsilon_i q_i. \end{aligned}$$

Since the probability of getting a solution at some processor is $Pr_{nondiv} = 1 - (1-p)^n$ by the non-divided method,

$$Pr_{nondiv} = 1 - (1 - \sum_{i=1}^{n} \alpha_i q_i)^n$$

and, the expected time for getting a solution is

$$T_{nondiv} = \frac{t}{Pr_{nondiv}}$$

By the divided method, each processor i searches only a subspace S_i. The probability of getting a solution at some processor of n processors by the divided method is

$$Pr_{div} = 1 - \prod_{i=1}^{n} (1 - q_i)$$

the expected time for getting a solution is

$$T_{div} = \frac{t}{Pr_{div}}$$

[1] This example was originally used in [10].

Which method is faster?

$$\begin{aligned} Pr_{div} - Pr_{nondiv} &= (1 - \sum_{i=1}^{n} \alpha_i q_i)^n - \prod_{i=1}^{n}(1 - q_i) \\ &= (\frac{1}{n}\sum(1 - q_i) - \sum \varepsilon_i q_i)^n - \prod(1 - q_i). \end{aligned}$$

If $\sum \varepsilon_i q_i \leq \frac{1}{n}\sum(1 - q_i) - \prod(1 - q_i)^{\frac{1}{n}}$,

$$\begin{aligned} Pr_{div} &\geq Pr_{nondiv}, \\ \text{because} & \\ \frac{1}{n}\sum(1 - q_i) - \sum \varepsilon_i q_i &\geq \prod(1 - q_i)^{\frac{1}{n}}. \end{aligned}$$

If $\frac{1}{n}\sum(1 - q_i) - \prod(1 - q_i)^{\frac{1}{n}} \leq \sum \varepsilon_i q_i$,

$$\begin{aligned} Pr_{nondiv} &\geq Pr_{div}, \\ \text{because} & \\ \frac{1}{n}\sum(1 - q_i) - \sum \varepsilon_i q_i &\leq \prod(1 - q_i)^{\frac{1}{n}}. \end{aligned}$$

In particular, if $\varepsilon_i = 0$, namely $\sum \varepsilon_i q_i = 0$, then $Pr_{div} \geq Pr_{nondiv}$ owing to geometric mean $\geq$ harmonic mean with $1 - q_i \geq 0$. We notice $\alpha_i = \frac{1}{n}$ in this case. If $q_1 = q_2 = \cdots = q_n$, then $Pr_{div} = Pr_{nondiv}$.

From the above discussion, a set of values of ε_i and q_i, that is, a way of dividing the search space, determines which of Pr_{nondiv} and Pr_{div} is larger.

3.2 Divided Method Experiments

We experimented with the 8 queen problem using the above two methods on a MultiPSI machine. There was no difference between the divided and non-divided methods(table 1) in the total time of 300 repetitions to get a single solution.

In the experiments, by the non-divided method each processor chooses at random from the queen position $\{1 \times 1, 1 \times 2, 1 \times 3, 1 \times 4, 1 \times 5, 1 \times 6, 1 \times 7, 1 \times 8\}$ at the first choice point. By the divided method, when 2 processors are used, we divide the search space into 2 independent subspaces, from which each processor chooses at random at the first choice point. One has candidate queen positions of $\{1 \times 1, 1 \times 2, 1 \times 3, 1 \times 4\}$ as the first choice. The other has candidate queen positions of $\{1 \times 5, 1 \times 6, 1 \times 7, 1 \times 8\}$ as the first choice. For 4 processors, we divide the search space into 4 independent subspaces. Candidate sets of queen positions as the first choice for each subspace are $\{1 \times 1, 1 \times 2\}$, $\{1 \times 3, 1 \times 4\}$, $\{1 \times 5, 1 \times 6\}$ and $\{1 \times 7, 1 \times 8\}$. For 8 processors we divide the search space into 8 independent subspaces. In each subspace one queen is placed from the beginning on the chess board at a position of $1 \times 1, 1 \times 2, \ldots, 1 \times 8$ respectively [2]. We notice that $\varepsilon_i = 0$ in this case.

Because $\varepsilon_i = 0$ owing to the divisions of the search space, and $q_1 = \cdots = q_n$ owing to the property of the queen problem, $Pr_{nondiv} = Pr_{div}$, so that there was no difference between the two methods. We need to experiment on problems which have imbalanced ε_i and q_i.

[2]This division method is similar to the static load balancing method for OR parallel search in [3].

PEs	divided method	non-divided method
8	334	341
4	464	458
2	700	698
1	0	1157

Table 1: Total times for 8 queen problem(sec)

4 Related Works

In [9, 10] the parallelization of the randomized backtrack search was provided. This is based on the same idea as in section 2 that each processor resolves the problem using different strategies to search a single solution. Though, by the divided and non-divided methods, each processor retries from the beginning on failure, by the parallel randomized backtrack search each processor goes back to the nearest choice point at the failure so that the processor may choose from remaining candidates.

We compare our randomized search with the randomized backtrack search about the computation on each processor. By the randomized backtrack search, each processor continues to search locally a subtree of a whole search tree until the chosen subtree is searched completely. If the chosen subtree has no solution, the search for the subtree is a waste of time. By our randomized search, since each processor chooses a candidate from a whole search space, the processor doesn't search locally. However, there is the possibility of repeated choices of the same search path at each processor, while there is no such possibility with the randomized backtrack search.

The framework of the expected time complexity in [9, 10] is different from ours. In [9, 10], the time to get a solution by one processor is considered as a random variable. Their speed up ratio of the expected time complexity depends on the distribution function of the random variable. Our random variable is a number of trials, and we assume that the time for one trial is constant.

5 Conclusions

We provided the parallel randomized search for getting a single solution by distributed memory machines. Then we provided the combined method of the parallel randomized search with a static load balancing method. For the divided method we considered the expected computation time and speed up ratio, and showed experimental results on the MultiPSI machine. And we compared the non-divided method to the divided method.

Since the parallel randomized search needs no communication during computation, this method is suitable for distributed memory machines from the aspect of communication cost.

The followings should be done as future tasks:

1. Pr_{div} and Pr_{nondiv} should be considered for different divisions of real problems' search spaces. This helps us choose the suitable method from divided and non-divided one.

2. Because, by the parallel randomized search, random choices at choice points are too arbitrary, namely, the probabilities for choosing each candidate are equal, we introduce heuristics for bias toward some paths to reach solutions quickly.

3. For the non-divided method, processors should exchange some information about each other to reduce repeated choice of search paths.

Acknowledgments

We wish to thank Dr.W.Ertel from Techniche Universität München, who gave us useful comments on early version of this paper. We wish to thank Dr.V.Kumar from University of Minnesota, and the researchers of ICOT for helpful suggestions and discussions.

References

[1] Masuzawa, H. et al.: "Kabuwake" Parallel Inference Mechanism and Its Evaluation, *FJCC-86*, pp. 955 - 962 (1986).

[2] Furuichi, M. et al.: A Multi-Level Load Balancing scheme for OR-Parallel Exhaustive Search Programs on the Multi-PSI, *PPoPP '90*, SIGPLAN NOTICES Vol. 25 No. 3, pp. 50 - 59 (1990).

[3] Burg, B.: Parallel Forward Checking First Part, *ICOT TR-594*, (1991).

[4] Doyle, J.: A Truth Maintenance System, *Artificial Intelligence*, **12**, pp. 231 - 272 (1979).

[5] Petrie, C. J. Jr.: A Diffusing Computation for Truth Maintenance, *Proc. ICPP-86*, pp. 691 - 695 (1986).

[6] Fulcomer, R. M., Ball W. E.: Correct Parallel Status Assignment for the Reason Maintenance System, *Proc. IJCAI-89*, pp 30 - 35 (1989).

[7] Satoh, K., Iwayama, N.: Computing Abduction by Using the TMS, *to appear in ICLP-91*,(1991).

[8] Nakajima, K. et al.: Distributed implementation of KL1 on the Multi-PSI/V2, *ICLP-89*,(1989).

[9] Janakiram, V. K. et al.: Randomized Parallel Algorithms for Prolog Programs and Backtracking Applications, *Proc. ICPP-87*, pp 278 - 280 (1987).

[10] Ertel, W.: Random Competition: A Simple, but Efficient Method for Parallelizing Inference Systems, *Techniche Universität München, Report FKI-143-90*,(1990).