Parallel Processing Method of Combinatorial Problem Solving Based on Implicit Stochastic Divide-and-Conquer*

Yasusi Kanada
Tsukuba Research Center, Real World Computing Partnership
Takezono 1-6-1, Tsukuba, Ibaraki 305, Japan
E-mail: *kanada@trc.rwcp.or.jp*, WWW: *http://www.rwcp.or.jp/people/yk/*

Abstract

A method of solving combinatorial problems, such as the N queens problem or graph coloring problems using independent parallel processes, is proposed. This method is stochastic (or randomized). Problems are decomposed for parallel processing implicitly and stochastically. This method is based on CCM, which is a computational model proposed by the author. A program consists of production rules and local evaluation functions in CCM. Each process uses the same set of rules and functions, and it may use the same set of initial data. However, the performance is approximately in proportion to the number of processors in average in certain cases. The theoretical reason of this linear acceleration is explained, and several results of experiments are also shown.

Keywords

Combinatorial problem solving, Randomized algorithms, Emergent computation, Stochastic computation, Divide-and-conquer method

1. Introduction

Most of conventional methods of solving complex problems have, as we believe, two major problems. The one problem is that these methods rely on explicit problem decomposition, or divide-and-conquer method. Problems are asserted to be able to be divided into sub-problems by humans, which can be solved mostly independently. For example, to search a solution of a combinatorial problem, a search tree is built for the problem decomposition, or to solve a problem in conventional parallel processing method, the problem must be divided by human into sub-problems that do not have strong mutual data dependence. However, complex problems are not easy to be divided into independent sub-problems, and problems in real world are often impossible to be divided because of their *non-reductionistic* or *holistic* [Koe 78, Kan 94] nature.

The other problem of conventional methods is that they are very weak for unexpected situations because they are deterministic and explicitly designed, and thus, they cannot go beyond the human planner's explicit knowledge. Artificial systems in real world, such as on-line banking systems, are open to natural systems, i.e., human society or nature. Natural systems are *autonomous* and *nondeterministic* and thus their behavior is often unpredictable. Thus, artificial systems should be ready to process such unexpected situations, or at least they must be able to use implicit knowledge that the human planners do not know explicitly. However, deterministic and explicitly designed systems can, we believe, only process expected situations.

Thus, we should develop methods of problem solving which are not based on strict divide-and-conquer method and which are stochastic, or randomized, and thus, not deterministic. Not all methods that satisfy these conditions are acceptable, of course. However, the methods that we should develop are, as we believe, among the methods that satisfies these two conditions, and not among the methods that are deterministic or based on strict divide-and-conquer method.

Kanada [Kan 92, Kan 94] proposed a computational model called CCM (Chemical Casting Model), which is a model of stochastic and local- or partial-information-based computation. Local-information-based computation seems to have a chance to process unexpected situations using implicit knowledge that is emerged from interaction between local computation processes. Solving combinatorial problems, such as the *N* queens problem [Kan 94], graph or map coloring problem [Kan 93b, Kan 95b], traveling salesperson problems [Kan 93a], or fuzzy coloring problem [Kan 95a] has been experienced. However, it has been tested on sequential computers.

A method of parallel processing of solving combinatorial problems using CCM is proposed in the present paper. A problem is solved using multiple independent processes in this method. The word, "independent," means that the processes do not communicate each other. This method is stochastic or randomized. This method is not based on divide-and-conquer method in the conventional sense. No

* A preliminary summary (in Japanese) of this paper was presented at the 49th National Conference of the Information Processing Society of Japan.

search tree is built in this method. This method can be regarded as a problem-solving method based on implicit and stochastic divide-and-conquer. It is not yet certain that this method lead to a problem solving method that is holistic and adaptive to natural systems. However, there is possibility.

The computational model, CCM, is briefly explained in Section 2. A method of combinatorial problem solving using CCM is explained in Section 3. A probabilistic model of CCM-based computation, which is called the Markov chain model, is explained in Section 4. The method of independent parallel processing using CCM is explained in Section 5. The results of experiments on this method are shown and analyzed in Section 6. Related work is mentioned in Section 7. Finally, conclusions are given in Section 8.

2. Computational Model CCM

CCM (the chemical casting model) [Kan 92, Kan 94] is explained briefly in the present section.

CCM has been developed for emergent computation [For 91][Lan 89–94], which is computation based on local and partial information. CCM is based on a production system. Production systems are often used for developing expert systems or modeling human brains. However, CCM is different from conventional production systems. Firstly, evaluation functions, which are evaluated using local information only, are used. Secondly, stochastic control, or randomized ordering of rule applications, is used. Production rules are also applied only using local information.

The system components in CCM are shown in **Figure 1**. The set of data to which the rules apply is called the *working memory*. A unit of data in the working memory is called an *atom*. An atom has a type and an internal state, and may be connected to other atoms by *links*. Links are similar to chemical bonding, but the difference is that links may have directions. Any discrete data structures such as lists, trees, graphs or networks can be represented using atoms and links.

The state of the working memory is changed locally by *reaction rules*. “Locally” means that the number of atoms referred by a reaction rule is small.[1] The reaction rules are written as forward-chaining production rules, such as rules in expert systems. However, reaction rules are at a lower level, or more primitive, than rules in expert systems. So, the reaction rules are more similar to reaction formulae in chemical reactions, and thus, this model is called the *chemical* casting model. The syntax of reaction rules is as

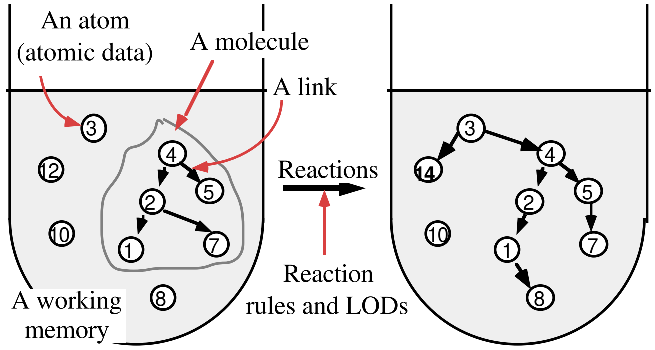


Figure 1. The elements of CCM

follows:

LHS → RHS.

The left-hand side (LHS) and the right-hand side (RHS) are sequences of patterns.

For example, the following reaction rule, which is a rough sketch, simulates the generation of water from oxygen and hydrogen:

H-H, O-? → H-O-H, ?

(This approximately means $H_2 + \frac{1}{2}O_2 \rightarrow H_2O$).

There are four patterns both in the LHS and RHS: two H’s, an O, and “?” (an unknown atom). Each pattern matches an atom of type oxygen or type hydrogen in the working memory.

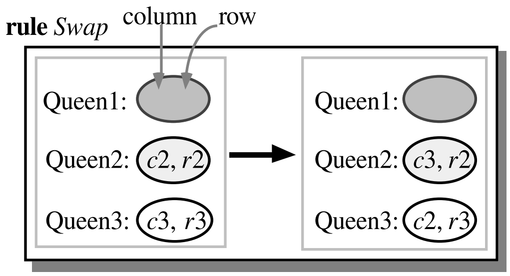
The reaction rule can be activated when there is a set of atoms that matches the LHS patterns. If the reaction rule is activated, the matched atoms vanish and new atoms that match the RHS patterns are generated. A single reaction rule is enough for solving a simpler optimization or constraint satisfaction problem like the graph vertex coloring problem, which is described later, or the 0–1 integer programming problem. Two or more reaction rules are needed in more complex systems, in which there are two or more ways of changing atoms.

Local order degrees (LODs) are a type of evaluation functions. LODs express the degrees of local “organization” or “order.” They are defined by the user to take a larger value when the local state of the working memory is better. An LOD may be regarded as a negated *energy*. For example, it is analogous to bonding energy in chemical reaction systems.

A reaction takes place when the following two conditions are satisfied. First, there exists an atom that matches each pattern in the LHS. Second, the sum of the LODs of all the atoms concerned in the reaction, i.e., the atoms that appear on either side of the reaction rule, does not decrease as a result of the reaction. Reactions repeatedly occur while the above two conditions are satisfied by a combination of any rule and atoms. The system stops, i.e., becomes

[1] Because (physical) distance is not a factor in CCM unlike systems such as a chemical reaction system, “locally” does not mean the distance is small.

■ **Reaction rule**



■ **Local order degree (mutual order degree)**

The local order degree is defined between two queens.

$$o(x, y) = \begin{cases} 0 & \textbf{if } x.column - y.column = x.row - y.row \textbf{ or} \\ & \quad x.column - y.column = y.row - x.row, \\ 1 & \textbf{otherwise}. \end{cases}$$

Figure 2. A rule and LOD of the *N* queens problem

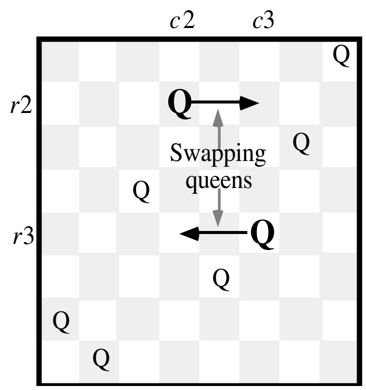


Figure 3. The meaning of the *N* queens rule

a stationary state, when such a combination is exhausted. However, reactions may occur again if the working memory is modified because of changes in the problem or the environment. Thus, open and dynamic problems as mentioned beforehand can probably be handled properly using CCM.

Typically, there are two or more combinations that satisfy the two conditions at the same time. There are two possible causes that generates multiple combinations. One cause is that there are two or more collections of atoms that satisfy the LHS of a reaction rule. The other cause is that there are two or more reaction rules containing atoms that match the patterns in the LHS. In each case, the order of the reactions, or the order of selection of such combinations, and whether they occur in parallel or sequentially is determined stochastically or randomly. Therefore, although the microscopic behavior of CCM, i.e., a reaction, may be deterministic in a sense, the macroscopic behavior is nondeterministic or random.

3. Combinatorial Problem Solving Using CCM

The *N* queens system, an CCM-based system for finding a solution to the *N* queens problem, is briefly explained in the present section. The *N* queens system is explained more detailed by Kanada [Kan 94].

The *N* queens problem is an extension of the eight queens problem. The LOD and the rule in a visual form for the *N* queens system are shown in **Figure 2**. This system contains only one rule and a definition of the mutual order degree, $o(x, y)$, i.e., an LOD defined between two queens. This rule swaps the rows of two queens (Queen2 and Queen3 in Figure 2). See **Figure 3**. Queen1, which can be called a *catalyst*, remains unchanged by the swapping. The role of the catalyst is explained later. No link is used in this rule. The value of LOD $o(x, y)$ is defined to be higher (i.e., 1) when queens x and y are not diagonally oriented, and lower (i.e., 0) when they are diagonally oriented.

However, the rule shown in Figure 2 contains three patterns for queens both in LHS and RHS. The third pattern (i.e., Queen1 in Figure 2) does not change the contents of the working memory, but it affects the computation of order degrees. A reaction takes place when the following condition holds:

$$O_b(\mathrm{Q1}, \mathrm{Q2}) + O_b(\mathrm{Q1}, \mathrm{Q3}) + O_b(\mathrm{Q2}, \mathrm{Q3}) \le O_a(\mathrm{Q1}, \mathrm{Q2}) + O_a(\mathrm{Q1}, \mathrm{Q3}) + O_a(\mathrm{Q2}, \mathrm{Q3}),$$

where Q1, Q2 and Q3 are the queens that matched patterns Queen1, Queen2 and Queen3, and O_b and O_a denote the LOD before and after the reaction. A pattern that does not change the data is called a *catalyst*. The catalyst in the rule in Figure 2 drives the system toward a solution. The effect of catalysts is explained by Kanada [Kan 94].

If the rule is executed with an appropriate initial state, the system repeats the selection of three queens and reaction of the instance. The initial state must satisfy the following condition: there is only one queen in each row and each column. The easiest layout that satisfies this condition is to put all queens on a diagonal line. If this condition holds, it holds at any time in the system because the reaction preserves the condition. The system stops when a solution of the *N* queens problem is found.

A mean value of LOD is called a mean order degree (MOD). An MOD can be local or global, i.e., a mean of either small or large number of data can be computed. An MOD changes continually while solving a problem. A sample path of MOD time sequence is shown in **Figure 4**. The MOD shown in this figure is the mean value of the LODs of the eight (i.e., all the) queens. The value of MOD changes stochastically, but it increases in average and becomes the maximum value, i.e., 1, when the number of

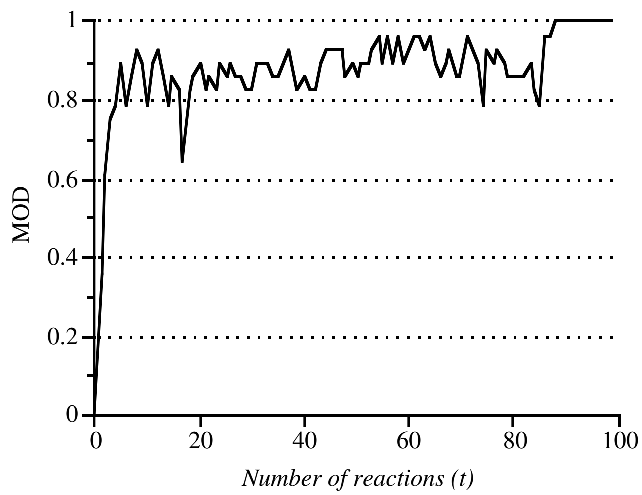


Figure 4. A sample of MOD time sequence in the eight queens problem solving

reactions is 88 in this case. This means that the system found a solution. The system becomes stationary when it finds a solution.

Other constraint satisfaction problems can be solved in the similar method as the N queens problem, if all the constraints are expressed as a relation between two objects [Kan 93b]. The values of LODs can be defined to be 0 or 1. Thus, the values of MODs are between 0 and 1. The values of MODs become 1 when the problem has been solved.

4. Markov Chain Model of CCM-based Computation

It is shown that the distribution of execution time is close to an exponential distribution under certain conditions in the present section.

Time sequences of MODs can be approximated by a Markov chain in the processes of solving several problems, including the N queens problem and graph/map coloring problems, experimentally [Kan 93b]. MODs are asserted to take discrete values, $o_0, o_1, \ldots, o_I$ $(0 = o_0 < o_1 < \ldots < o_I = 1)$, here.[1] Time is measured by the number of reactions from the beginning. The value of MOD at time t, $O(t)$, is a random variable. The probability that $O(t)$ is equal to o_i is described as $p(O(t) = o_i)$, where the following condition holds:

$$p(0 \le O(t) \le 1) \;=\; \sum_{i=0}^{I} p(O(t) = o_i) \;=\; 1$$

A vector, whose elements are $p(O(t) = o_i)$ $(i = 0, 1, \ldots, I)$, is denoted by $\boldsymbol{p}_t$. Then, the following relation approximately holds.

$$\boldsymbol{p}_{t+1} \;=\; T\boldsymbol{p}_t \,.$$

The transition matrix, T, has I rows and I columns. The values of elements of T are asserted *not* to depend on time.

If the eigen values of T are denoted by $\lambda_0, \lambda_1, \ldots, \lambda_I$ $(|\lambda_0| \ge |\lambda_1| \ge \ldots \ge |\lambda_I|)$, then $\lambda_0 = 1$. T^t can be expressed as the following well-known form:

$$T^t \;=\; T_0 + \lambda_1{}^t\, T_1 + \lambda_2{}^t\, T_2 + \ldots + \lambda_I{}^t\, T_I$$

If $\lambda_2, \lambda_3, \ldots, \lambda_I$ are equal to zero, the probability that the system is in a state other than the solution state, i.e., $p(O(t) < 1)$, decreases exponentially. Thus, the distribution of computation time until the system becomes the stationary state (solution state) is an exponential distribution. As explained in the previous section, the MOD often becomes close to 1 in an earlier stage of computation when solving a CSP. In such a system, $|\lambda_1|$ is close enough to 1, and $|\lambda_2|, \ldots, |\lambda_I|$ are far below 1. So, T^t is approximately equal to $T_0 + \lambda_1{}^t\, T_1$ when $t \gg 0$, and $p(O(t) < 1)$ decreases exponentially.

In the case of the eight queens problem, the eigen values are estimated to be as follows [Kan 93b] by measurements:[2]

$$\lambda_1 = 0.986,\;\; \lambda_2 = 0.5,\;\; \lambda_3 = 0.2,\;\; \ldots$$

Thus, the above condition holds for the eight queens system.

The probability of each MOD value is measured statistically in the case of the eight queens problem. The result is displayed in **Figure 5**. The probabilities of the states except the solution state, i.e., the state in which MOD = 1, decreased almost exponentially as expected. The distribution of computation time is measured statistically. The result is displayed in **Figure 6**. The distribution is close to an exponential distribution except when time (the number of reactions) is near zero.

5. A Method of Independent Parallel Processing

A method of parallel processing of CCM-based computation using non-communicating multiple processes is explained in the present section. The reason of linear acceleration is explained both intuitively and theoretically.

The method of parallel processing is as follows. (See **Figure 7**.) A parallel computer with at least M processors is used. Only one process runs on each processor. Thus, the processor and the process are identified in the present section. The same reaction rule and LOD are stored in each processor. The initial data may be the same or different for all the processors. The computation of each processor, which is based on CCM, is performed independently.

[1] Kanada [Kan 93a] shows a method of constructing a Markov chain model when the MOD takes continuous values.

[2] It is currently not possible to estimate the eigen values only from theories.

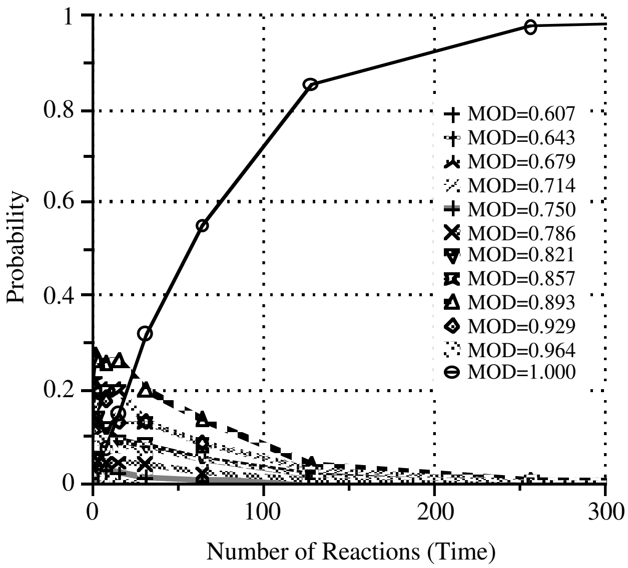


Figure 5. Probabilities of the states in which MOD is 0.607 to 1 in the eight queens

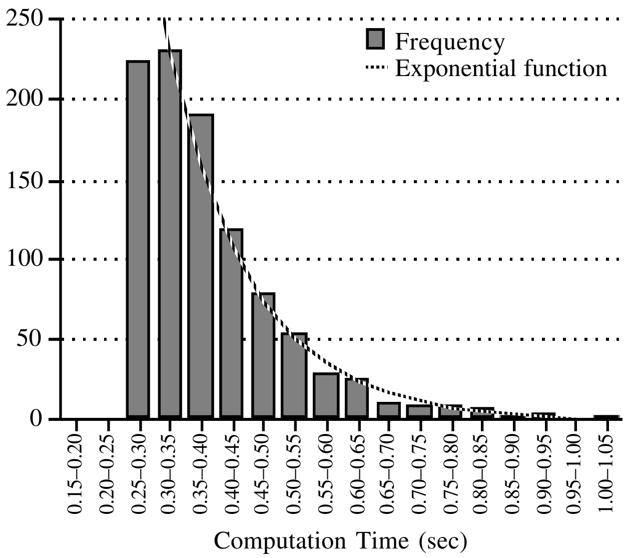


Figure 6. A sample distribution of the computation time of the eight queens problem

Each processor generates independent random numbers for selecting data to be reacted. This independence is very important.[1] When a process finishes the computation, the solution found by the process is outputted, and all the processes are killed. Communication between processors is used only for this process termination, and is never used during the computation. If the distribution of sequential computation time is exponential, the performance will be improved in proportion to M by this method.

The reason why the acceleration is linear can be intuitively explained as follows. The computation can be regarded as a search of solution in combinatorial problem solving. Processes probably search different places in the search space, if the search space is large enough. So the efficiency is in proportion to the number of processors.

The reason of linear acceleration is explained theoretically using the following theorem, in which the value of each random variable is interpreted as execution time.

Theorem: If random variables, $X_1, X_2, \ldots, X_m$, follows an exponential distribution, whose density function is $p_1(x) = \lambda e^{-\lambda x}$, then the random variable, $min(X_1, X_2, \ldots, X_m)$, follows the exponential distribution, whose density function, $p_m(x)$, is equal to $m\lambda e^{-m\lambda x}$.

Proof: The probability that the value of the random variable $min(X_1, X_2, \ldots, X_m)$ is larger than x is expressed as follows.

$$\int_x^\infty p_m(y)\,\mathrm{d}y$$

$$= \mathrm{P}\left\{x < min(X_1, X_2, \ldots, X_m)\right\}$$

where $\mathrm{P}\{C\}$ denotes the probability that condition C is satisfied

$$= \mathrm{P}\left\{x < X_1\right\} * \mathrm{P}\left\{x < X_2\right\} * \ldots * \mathrm{P}\left\{x < X_m\right\}$$

because $X_1, X_2, \ldots,$ and X_m are independent random variables, and the condition $x < min(X_1, X_2, \ldots, X_m)$ means that $x < min(X_1)$ and $x < min(X_2)$ and … and $x < min(X_m)$

$$= \left(\int_x^\infty p(y)\,\mathrm{d}y\right)^m$$

$$= e^{-m\lambda x}.$$

Thus, $\frac{\mathrm{d}}{\mathrm{d}x}\int_x^\infty p_m(y)\,\mathrm{d}y = \frac{\mathrm{d}}{\mathrm{d}x} e^{-m\lambda x}$,

which means $p_m(x) = m\lambda e^{-m\lambda x}$. ■

The distribution of the parallel processing time using M processors is equal to $p_M(x)$ by this theorem. The expectation of random variable X' that follows distribution $p_M(x')$ is $1/M$ of the expectation of X that follows distribution $p_1(x)$. Thus, the average parallel processing time is $1/M$ of the sequential processing time.

6. Experiments

The method proposed in the previous section is applied to several combinatorial problems. The measured acceleration ratios by the parallel processing are shown in the present section.

Parallel processing time of the N queens problem by the rule and LOD shown in Figure 2 has been estimated using 50 measurement results of sequential processing time.

[1] If the random numbers are not independent, the performance degrades. For exapmle, if all the processors use the same duplicated sequence of random numbers, they compute the same result. Thus, the performance is the same as using one processor.

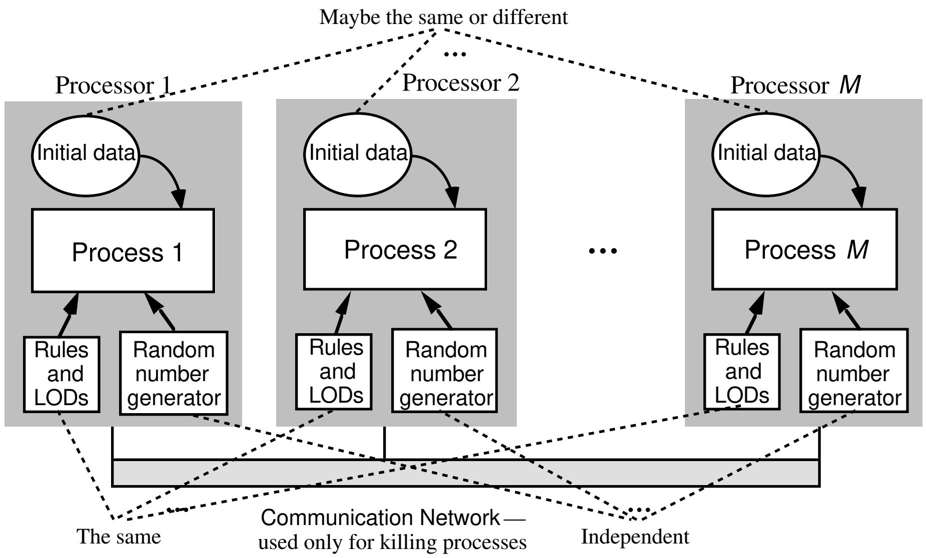


Figure 7. The method of parallel processing CCM-based computation

Namely, sequential processing time is repeatedly measured using computational language called SOOC[1] and its interpreter, and the average of minimum values of 2 to 16 measured values are computed. The initialization time, i.e., the time for making initial placement of queens, is excluded from the measured time. The i-th measured execution time of the N queens problem is denoted by $t(N, i)$ here ($i = 1, 2, \ldots, 50$). The average values of $t(N, M, j)$ ($j = 1, 2, \ldots$), which represents a measured value for the N queens by M processors, are plotted for $M = 1, 2, 4, \ldots, 16$, and $N = 4, 6, \ldots, 14$, in **Figure 8**, where $t(N, 2, 1) = min\big(t(N, 1), t(N, 2)\big)$, $t(N, 2, 2) = min\big(t(N, 3), t(N, 4)\big)$, …, and $t(N, 4, 1) = min\big(t(N, 1), t(N, 2), t(N, 3), t(N, 4)\big)$, …, and so on. Because the distribution of the execution time is close to an exponential distribution when N is larger, the performance is accelerated nearly linearly.

Real parallel processing time has also been measured using the same rule and LOD that are hand-coded using C on Cray Superserver 6400 (CS6400). CS6400 has shared memory, and thus, multiple threads (but not UNIX processes), which run on different processors but share memory, are used for this measurement. However, the shared memory is used only for initial data distribution, final data output, and process termination. There is a parent thread and it distributes the input data, i.e., the value of N, to M child threads. Only the thread that finds the solution first outputs the solution and synchronizes with the parent. (The parent is busy-waiting for this synchronization.) Then the parent terminates and other threads are killed by the operating system. Thus, the threads are not explicitly killed in the program. The measured time includes the initialization time because the initialization is also performed in parallel and it is difficult to be separated.

Each thread computes random numbers independently (using `rand_r` library routine). The random number seed is generated using both the time in micro second and the address of input data for each thread. I have chosen the method of seed generation carefully to guarantee the independence of random numbers between threads.

The execution time has been measured 50 times for each N. The result of measurement is shown in **Figure 9**. The CS6400 used for the measurement has 12 processors, each of which a thread is assigned to, and there is a parent thread. Thus, the maximum value of M is 11, where the parent is not counted. The performance is worse than the simulation. When $N = 14$, the acceleration is nearly linear in the simulation, but it is far below linear in the real parallel execution. However, the acceleration is almost linear when N is 18 or 20. Thus, the method shown in the previous section has been proved to be effective for the N queens problem. The reasons that the performance is worse than the simulation are probably as follows.

(1) There is parallelization overhead, which is caused by

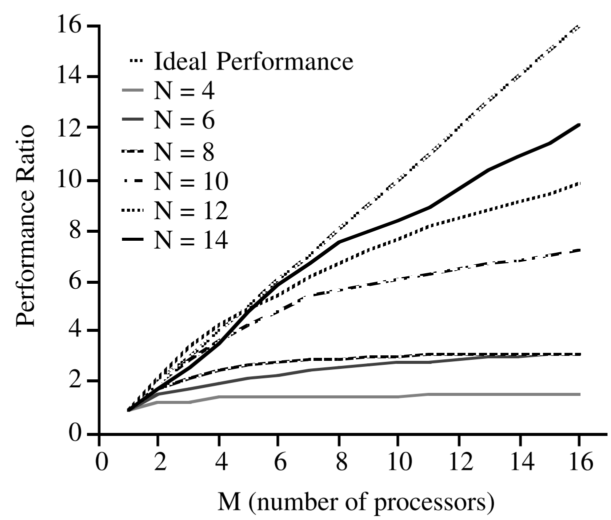


Figure 8. Simulated performance of the N queens problem

[1] SOOC is a computational language designed for experiments of CCM-based computation. SOOC is built on Common Lisp.

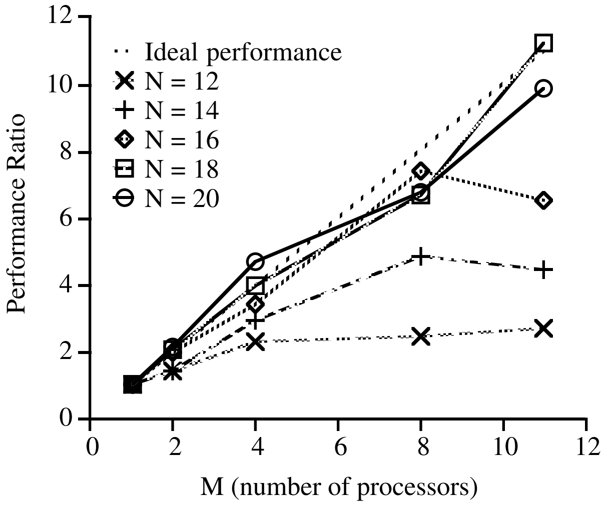


Figure 9. Parallel performance of the *N* queens problem

time for thread dispatching and final synchronization[1].

(2) The measured time includes the initialization time, which is not accelerated by the parallelization.

Both reasons make the distribution of execution time apart from an exponential distribution.

Computation of solving the graph or map coloring problem is also evaluated by the method using simulation.[2] The simulated performance is shown in **Figure 10**. The performance of the USA mainland map [Tak 92] is not good probably because the problem is too small for parallelization. However, the performance of several problems in the DIMACS benchmarks [Tri][DIM] is good, and this method has been proved to be effective for the coloring problems.

Not all constraint satisfaction problems are linearly accelerated by this method. Even the performance of the same problem can be different, if a different set of rules is used. For example, a simulated performance of the *N* queens problem has been measured and is shown in **Figure 11**. In this measurement, a rule that is different from Figure 2 and that has less locality is used. This rule refers a vertex and all its neighbors. This rule refers more data than the rule shown in Figure 2, i.e., it is less local. The ratio of performance improvement is less than 3 even when $M = 16$ and $N = 50$. This result shows that the high degree of locality in the computation is indispensable for performance improvement. The real parallel performance has not been measured because much improvement is not expected.

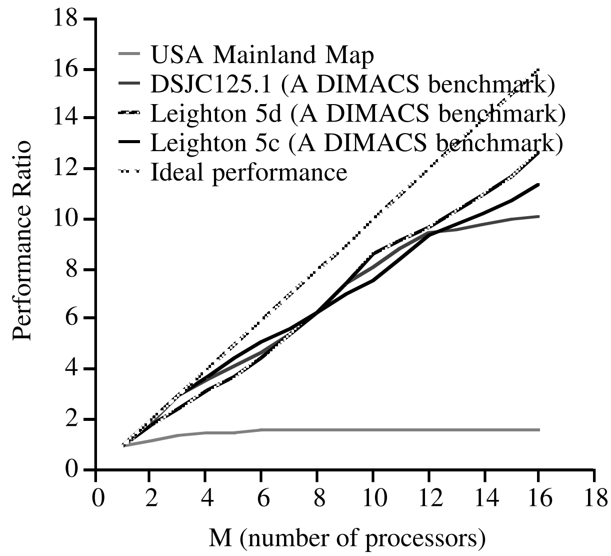


Figure 10. Simulated performance of the graph/map coloring problems

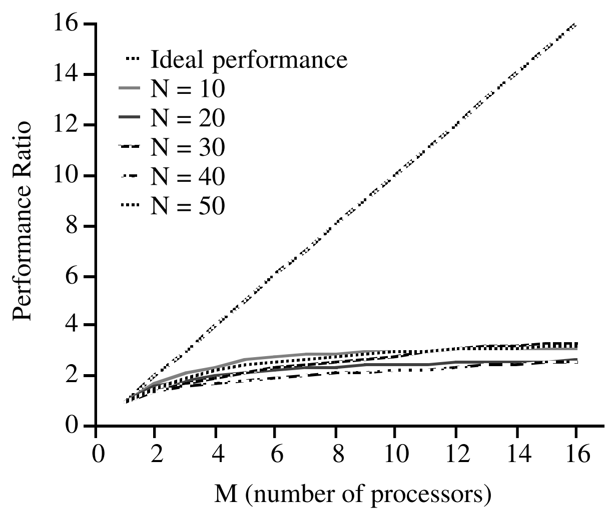


Figure 11. Simulated performance of the *N* queens problem using a less local rule

Other problems, including exchange sort and traveling salesperson problem (TSP), are also tested by simulation. The execution time of exchange sort, which can be regarded as a constraint satisfaction problem, is almost constant. Thus, it is almost never accelerated. The simulation results of TSP with 10 to 20 cities are shown in **Figure 12**. The acceleration ratio is far less than 2, even when the number of processors is 16. No global optimization problem that can be accelerated nearly linearly has yet be found. It is probably difficult to improve the performance of solving global optimization problems by this

1 Time for killing threads is not included in the measured time, but it is known to be almost negligible when N is large. This overhead is measured to be 1% (when $N = 18$) to 3% (when $N = 12$) in average.

2 A rule with variable number of catalysts [Kan 94, Kan 95a] is used for this measurement.

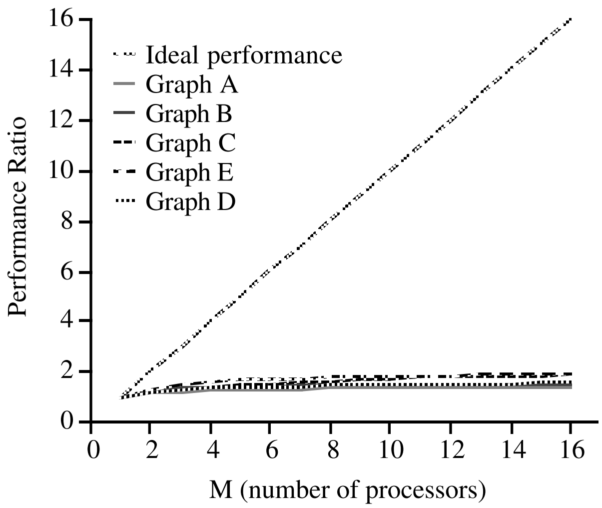


Figure 12. Simulated performance of TSP with 10 to 20 cities

method because of non-local nature of their computation; they are *global* optimizations.

7. Related Work

Mehrotra [Meh 85] stated that super-linear acceleration is made possible by a randomized parallel processing of tree search problems. Mehrotra compared a deterministic sequential version and a randomized parallel version of an algorithm.[1] However, Mehrotra did not compare the performance of randomized parallel processing with different numbers of processors.

8. Conclusion

The distribution of computation time is close to an exponential distribution in some cases in CCM-based computation. In such cases, it is proved that almost linear performance improvement is possible by both theory and practice. This method makes implicit and stochastic decomposition of problems possible. Although the result shown in the present paper is only one step toward the goal, we believe that researches on such implicit stochastic divide-and-concur methods will lead us to a new methodology of problem solving and software development, which is not only applied to parallel processing, in future.

Acknowledgment

The author thanks to Shoji Hatano from TRC (the Tsukuba Research Center, RWCP), for showing a better proof of the theorem described in Section 5. He also thanks to Susumu Seki and Hironobu Takahashi from TRC for tutoring him the usage and programming techniques of CS6400.

[1] The function of these two versions of programs are different. Thus, it is not fare to state that a super-linear acceleration was performed.

References

[DIM] Center for Discrete Mathematics and Theoretical Computer Science, *http://dimacs.rutgers.-edu/*.

[For 91] Forrest, S., ed.: *Emergent Computation*, MIT Press, 1991.

[Kan 92] Kanada, Y.: Toward a Model of Computer-based Self-organizing Systems, *Proc. 33rd Programming Symposium*, 1992 (in Japanese).

[Kan 93a] Kanada, Y.: Optimization using Production Rules and Local Evaluation Functions, *11th Meeting of the Technical Group on Software Engineering*, The Society of Instrument and Control Engineers (SICE), 1993 (in Japanese).

[Kan 93b] Kanada, Y.: Computations as Stochastic Processes — Necessity and Examples of Macroscopic Models of Computation Processes —, *IEICE Technical Groups on Computation / Software Science*, COMP92-93, SS92-40, 1–10, 1993 (in Japanese).

[Kan 94] Kanada, Y., and Hirokawa, M.: Stochastic Problem Solving by Local Computation based on Self-organization Paradigm, *27th Hawaii International Conference on System Sciences,* 82–91, 1994.

[Kan 95a] Kanada, Y.: Fuzzy Constraint Satisfaction Using CCM — A Local Information Based Computation Model, *FUZZ-IEEE/IFES ’95*.

[Kan 95b] Kanada, Y.: Large-Scale Constraint Satisfaction by Optimization of Local Evaluation Function with Annealing, *submitted for IJCAI ’95*.

[Koe 78] Koestler, A. : *JANUS*, Hutchinson and Co. Ltd., 1978.

[Lan 89–94] Langton, C. G.: *Artificial Life I*, *II* and *III*, Addison Wesley, 1989, 1991 and 1994.

[Meh 85] Mehrotra, R., and Gehringer, E. F.: Superlinear Speedup Through Randomized Algorithms, *14th Int'l. Conf. on Parallel Processing*, 291–300, 1985.

[Mor 93] Morris, P.: The Breakout Method For Escaping From Local Minima, *12th National Conference on Artificial Intelligence (AAAI-93)*, 40–45, 1993.

[Sel 93] Selman, B., and Kautz, H. A.: An Empirical Study of Greedy Local Search for Satisfiability Testing, *12th National Conference on Artificial Intelligence (AAAI-93)*, 46–51, 1993.

[Tak 92] Takefuji, Y.: *Neural Network Parallel Processing*, Kluwer Academic Publishers, 1992.

[Tri] Tricks, M.: The Second DIMACS Challenge, *http://mat.gsia.cmu.edu/challenge.html*.