# Methods of Parallel Processing of Constraint Satisfaction Using CCM — A Model for Emergent Computation

**Yasusi Kanada***
Tsukuba Research Center, Real World Computing Partnership
Takezono 1-6-1, Tsukuba, Ibaraki 305, JAPAN
kanada@trc.rwcp.or.jp

## Abstract

Two methods for solving large-scale constraint satisfaction problems using parallel processing are surveyed in the present paper. These methods are based on CCM, which is a model for emergent computation. The number of constraint violations is minimized in these methods. The minimization is performed by optimization of local functions. The computation is stochastic and no global information is used. An annealing method called FAM has been introduced to avoid "local maxima." FAM also works only with local information. Two types of parallel processing of CCM-based constraint satisfaction using FAM has been tested. One is a parallel search and the other is a cooperative search. Our experiments has shown that both methods improve performance almost linearly in large-scale graph coloring problems when the number of processors is ten or so.

## 1 Introduction

Problems in the real world are not only large-scale and complex but also open to the human society and/or the nature. The problems themselves are continually changing and only incomplete information is available for solving them. The changes may sometimes be unexpected, because the behavior of humans and other natural systems are sometimes unpredictable.

Conventional methods for problem solving, such as those used in operations research or various search methods in artificial intelligence do not work well in such situations. We believe that this is because global and complete planning is necessary and global information is used in these methods. Global planning and global information based computation are considered to be easily voided by small changes in the information, requirements or environment. Global information referencing also makes parallel processing difficult, because of the global reference creates a global data dependence.

We believe emergent computation [For 91], or local information based computation, is the key for solving these problems. Thus, we have been developing a model called CCM (the chemical casting model) [Kan 92, Kan 94a], which is a stochastic computational model for emergent computation. We have developed a method of solving constraint satisfaction problems (CSP) using CCM. The number of constraint violation is minimized in this method. The minimization is performed by optimization of local functions, i.e., partial sums of numbers of violations, but not by optimization of their total.

An annealing technique called FAM (the frustration accumulation method) has also been introduced in this method to avoid "local maxima." This method was developed because the annealing method that does not depend on global information is necessary. In FAM, No global parameter, such as temperature, is used. No global control, such as changing the value of a global temperature, is necessary, and the system thus works autonomously. The absense of global informtion is different from the usual annealing methods. FAM has made solving large-scale problems using CCM possible.

Because computation in CCM does not depend on global information, it continues to work even when information is changed dynamically and new information is added or removed. Thus, it seems to have the potential to process unexpected situations properly using implicit knowledge that emerges from interaction between local computation processes. New information may be added or preexisting information may be dynamically changed by environmental changes while solving problems. The nonexistence of global information references also makes parallel processing much easier.

CCM is briefly explained in Section 2. The basic method of solving CSP using CCM is explained in Section 3 and a CCM-based method with FAM is explained in Section 4. Two methods of solving CSP in parallel with FAM are explained in Section 5. The results of performance evaluation of these methods are shown in Section 6. Finally, our conclusion is given in Section 7.

## 2 Computational Model CCM

CCM [Kan 92, Kan 94a] is explained briefly here. CCM has been developed for computation using only local information. CCM is based on a forward-chaining production system [New 72], such as systems written in OPS5 [For 81]. Production systems are often used for develop-

*The author's current address is Central Research Laboratory, Hitachi Ltd., Kokubunji, Tokyo 185, JAPAN.

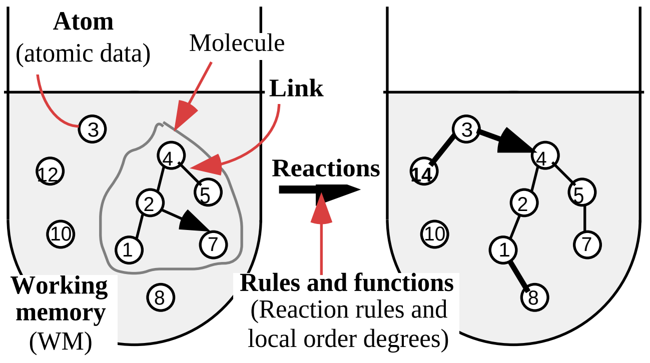

Figure 1: The elements of CCM

ing expert systems or when modeling the human brain in AI or cognitive sciences.

However, CCM differs from conventional forward-chaining production systems in two points. Firstly, evaluation functions, which are called local order degrees (LODs). LODs are similar to negated energy in chemical reaction systems, and are evaluated to decide whether to apply a rule or not. Rules and functions are computed using only local information. Secondly, stochastic control or randomized ordering of rule applications, is applied. Because of these features, we believe that CCM is much more similar and analogical to chemical reaction systems than conventional production systems in AI. Because chemical reactions occur in parallel, it is very natural to make reactions occur asynchronously in parallel in CCM, and the parallelization is much easier in CCM than in conventional production systems.

The structural 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 bonds, but the difference is that links may have directions.

A local state of the working memory is changed by a reaction rule. "Local" means that the number of atoms referred by a reaction rule is small. The reaction rules are written as production rules. However, reaction rules are at a lower level, or more primitive, than rules in expert systems. This means that the behavior of the system is not directly programmed using rules but it should emerge from repeated rule applications. Therefore, a reaction rule is more similar to the reaction formula of a single reaction in compex chemical reactions, and this model is thus called the chemical casting model.

The abstract syntax of reaction rules is as follows: LHS $\rightarrow$ RHS. LHS and RHS are sequences of patterns. 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 appear. So an atom may be modified, bonded to another atom, removed, or created by a reaction rule.

Local order degrees (LODs) are evaluation functions, or local objective functions, whose values are defined for a type or more types of atoms, using only local information. LODs are functions with one or two arguments. The arguments are atoms. An LOD may be regarded as a negated energy. For example, it is analogous to the bonding energy in chemical reaction systems.

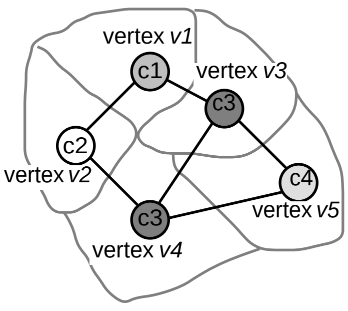

Figure 2: An example of a graph coloring problem

A reaction takes place when the following two conditions are satisfied. Firstly, there exists an atom that matches each pattern in the LHS. Secondly, the sum of the LODs of all the atoms that are involved 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 rules and atoms. The system enters a stationary state, or stops, when such a combination is exhausted. If the system is to solve a problem, this state must be the solution. However, reactions may occur again if the working memory is modified because of changes in the problem or the environment. A CCM-based system is a dynamical system rather than just a problem solving system. Thus, a CCM-based system can solve dynamical problems without additional rules or data.

Typically, there are two or more combinations that satisfy the two conditions at the same time. There may be two or more collections of atoms that satisfy the LHS of a reaction rule, or may be two or more reaction rules containing atoms that match the patterns in the LHS. In both cases, the order of the reactions are stochastic, and whether they occur in parallel or sequentially is arbitrary. The CCM-based systems that we have developed use random numbers for selecting rules and atoms. The same set of rules and LODs can be used both in sequential and parallel processing.

## 3 Constraint Satisfaction Using CCM

Kanada [Kan 94a] describes a method of problem solving using CCM, and uses the $N$ queens problem, which is a CSP, as an example. The same method can be applied to other CSPs. A CCM-based system to solve a coloring problem is described below.

The problem is how to color the vertices of a graph using a specified number of colors, for example, four. Each pair of neighboring vertices must be given different colors. A map coloring problem can be converted to a graph coloring problem, if areas of the map are converted to vertices and the area borders are converted to edges. Thus, the map coloring problem can be solved by the same rules and LODs as the graph coloring problems. For example, the problem of coloring the graph with five vertices, shown in Figure 2, is equivalent to the problem of coloring the map with five areas, which is also drawn in Figure 2. The data structure used for solving the problem is as follows. Vertices are represented by atoms. An atom of type vertex has a color as its internal state. In Figure 2, $c1$, $c2$, $c3$ and $c4$ are the colors.

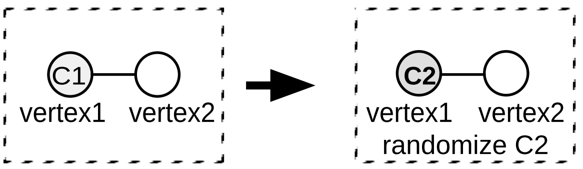

Figure 3: The reaction rule for the graph coloring system

The reaction rule and LOD to solve the problem are shown below. The only reaction rule is illustrated in a graphical form in Figure 3. This rule refers only to two neighboring vertices and the edge between them, and changes the color of one of the vertices randomly. The LHS of the rule contain two patterns; vertex1 and vertex2. C1 and C2 are variables to be matched to the colors of vertex1 and vertex2. There is a link between the atoms, and this link represents the edge between the vertices. Thus, vertex1 and vertex2 can only match two vertices that have a link between them. When a reaction occurs, the internal state of the atom that matches vertex1 is rewritten. That is, the color that was equal to C1 before the reaction becomes equal to C3, which is selected randomly from a predefined set of colors.

The LOD is defined between two vertices, $v_1$ and $v_2$. Its value is 1 when the constraint between $v_1$ and $v_2$ is satisfied, and otherwise it is 0.

$$o(v_1, v_2) = \begin{array}{l} 1 \textbf{ if not} \mathit{connected}(v_1, v_2) \textbf{ or } v_1.\mathit{color} \le v_2.\mathit{color} \\ 0 \textbf{ otherwise} \end{array}$$

This definition means that the value of the LOD is 1 (higher) if the vertices are not connected or have different colors, and otherwise that it is equal to 0 (lower). The value of LOD is 1 when the vertices are not connected in this case. The value is also 1 when the vertices are connected and have different colors, because the constraint between the vertices is satisfied. The value is 0 when the vertices are connected and have the same color, because the constraint is violated.

A reaction occurs when two connected vertices, $V_1$ and $V_2$, are selected by the system, and the following condition holds: $o(V_1, V_2) \le o(V_1', V_2)$, where $V_1'$ is $V_1$ after the reaction (with the new color).

A pattern of atoms, whose value is not changed by the rule, is called a catalyst [Kan 94a]. An atom that is not changed by a reaction is called a catalyst of the reaction. A pattern in a rule is also called a catalyst if the same (unchanged) pattern appears in both LHS and RHS of the rule. In the rule in Figure 3, vertex2 is a catalyst.

A slightly modified version of the rule and LOD, which is more complicated but performs better than the above rules and LODs, was coded by SOOC, which is a computational language based on CCM, and then tested. The coloring system on SOOC was applied to the map of the USA mainland, consisting of 48 states [Tak 92]. A correct solution was found in every run.

Coloring the USA mainland is a rather easy problem. Thus, it can be solved using the rule with only one catalyst. However, if the problem is more difficult, more catalysts are necessary. Otherwise, the computation time will be too long. If more catalysts are used, the computation can more easily fall into a "local maximum" of MOD [Kan 94a]. For example, a rule with a variable number of catalysts is more efficient than the single-catalyst rule. However, sometimes it fails to solve the problem but falls into a stationary state, which is a "local maximum." Or, sometimes it flips randomly between several states, which are "local maxima."

## 4 FAM: An Annealing Technique

A method of escaping from "local maxima" in CCM is proposed here. This method is called the frustration accumulation method (FAM) [Kan 94b]. In this method, the same rule and LOD shown in the previous section are used. However, the mechanism of computation, or the compiler of SOOC, must be modified.

Each atom, or vertex, has a type of energy called frustration, whose value is positive, in this method. The frustrations of all the atoms to be modified by the reaction are subtracted from the sum of the LODs. Thus, if a vertex has a non-zero frustration, reactions occur more easily. For example, if the rule in Figure 3 is used, a reaction occurs when the following condition holds:

$$o(V_1, V_2) - V_1.f \le o(V_1', V_2) - V_1.f',$$

where $V_1.f$ and $V_1.f'$ are the frustration of $V_1$ before and after the reaction. The frustrations of catalysts are not counted.

Each vertex initially has a certain level of frustration, $f_0$, which is, for example, $10^{-5}$. The frustration is increased, when the application of a reaction rule fails but there are unsatisfied constraints. That means that the frustration of an atom is increased when the following three conditions hold.

- The atom is tested for modification.
- The reaction does not occur.
- There are constraints, relating to the matched vertex which are not or will not be fully satisfied.

More specifically, if a rule and a set of atoms are tested for a reaction, if the reaction does not occur, and if the sum of the LODs is less than the maximum, then the frustration of each atom to be modified by the rule, $f$, is replaced by $cf$, where $c$ ($> 1$) is a constant. The value of $c$ is 2, for example. Thus, the procedure that runs after each test for a reaction can be described as follows.

```
**if** a reaction occurred **then**
  $f_i := f_0$; /* reset to the initial value */
**else if** not all the constraints are satisfied **then**
  $f_i := cf_i$; /* increased */
**end if**;
```

Assume that the sum of the LODs of the vertices matched to patterns in the rule before the reaction is represented by $O$, and that after the reaction is represented by $O'$. The sum of the frustrations before and after the reaction are represented by $F$ and $F'$. Then the reaction occurs when $O - F \le O' - F'$. Thus, the reaction occurs more easily when the value of $F$ is larger.

An example is shown in Figure 4. An improved version of the rule, which is called the variable catalyst rule [Kan 94a] is applied to the state shown in the upper figure. A reaction may or may not occur depending the color selected by the RHS of the rule. In case 1, no reaction occurs, and in case 2, a reaction does occur. The rule is applied to vertex $v3$ and its three neighboring vertices. The sum of LODs before the reaction, $o$, is 2, because the colors of $v1$ and $v3$ are different, that of $v4$ and $v3$ are the same, and that of $v5$ and $v4$ are different.

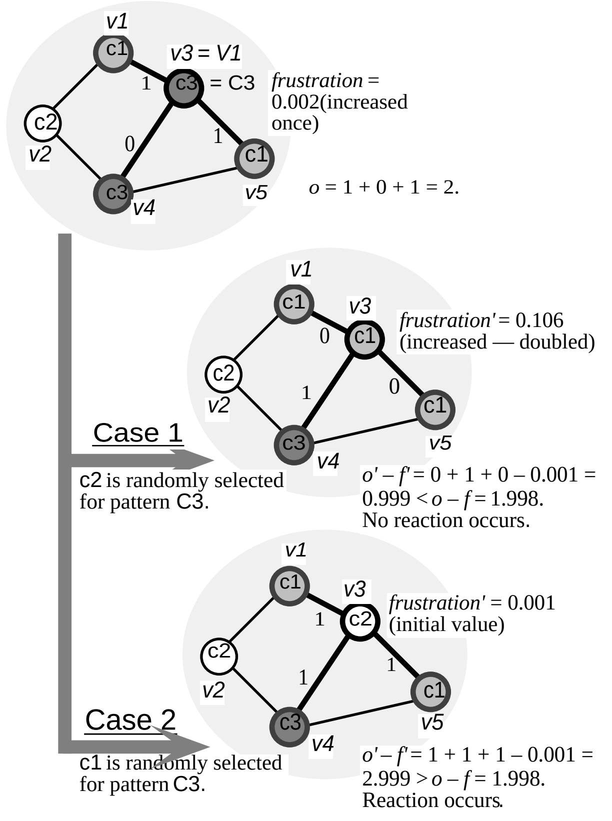
v1
v3 = V1
c1
1
c3 = C3
frustration = 0.002(increased once)
c2
v2
0
1
c1
c3
v4
v5
o = 1 + 0 + 1 = 2.
v1
c1
v3
frustration' = 0.106 (increased — doubled)
0
c1
c2
v2
1
0
c1
c3
v4
v5
Case 1
c2 is randomly selected for pattern C3.
o' – f' = 0 + 1 + 0 – 0.001 = 0.999 < o – f = 1.998.
No reaction occurs.
v1
c1
v3
frustration' = 0.001 (initial value)
1
c2
c2
1
1
c1
v2
c3
v4
v5
Case 2
c1 is randomly selected for pattern C3.
o' – f' = 1 + 1 + 1 – 0.001 = 2.999 > o – f = 1.998.
Reaction occurs.

Figure 4: The outline of FAM

The values labeled on the edges in the figures, 0 or 1, are the LOD values. The value of frustration is assumed to be 0.002, the value increased once from the initial value, $f_0 = 0.001$. The value of $c$ is 2.

In case 1, the randomly selected color for variable C3 (in Figure 4) is $c1$. The sum of LODs after the reaction, $o'$, is 1, because now the colors of $v1$ and $v3$ are the same. That of $v4$ and $v3$ are different, and that of $v5$ and $v3$ are the same. The sum of LODs would be decreased by the reaction. Thus, no reaction occurs, and the frustration of $v3$ is increased to 0.004. The subtraction of frustration does not affect the comparison result in this case, i.e., $o' < o$ and $o' - f' < o - f$. If the value of frustration before the reaction were 1.002 or more, the subtraction of $f'$ from $o'$ would change the comparison result, i.e., $o' < o$ but $o' - f' \geq o - f$, and a reaction would occur.

In case 2, the randomly selected color is $c2$. The sum after the reaction, $o'$, is 3, because the colors of all the neighrors of $v3$ are different from that of $v3$. The sum would be increased by the reaction. Thus, the reaction occurs, and the frustration of $v3$ is reset to the initial value, 0.001. A subtraction of frustration never changes the comparison result in cases that a reaction occurs.

$f_0$ and $c$, are constant during the execution, and they are considered to be properties of each atom or each type of atoms. Thus, this method works only with local information and no global parameters, such as temperature in simulated annealing, is used. The values of $f_0$ and $c$ are the same for all the vertices in our implementation.

## 5 Methods of Parallel Processing Using Annealed CCM

There are two types of parallel processing in CCM. One is *parallel reaction* or cooperative search, and the other is *parallel search*. Parallel reaction means that reactions in a working memory are processed in parallel. The processes cooperate to find a solution. Parallel search means that each process search a solution independently. Parallel reaction corresponds to and-parallelism and parallel search corresponds to or-parallelism in logic programming.

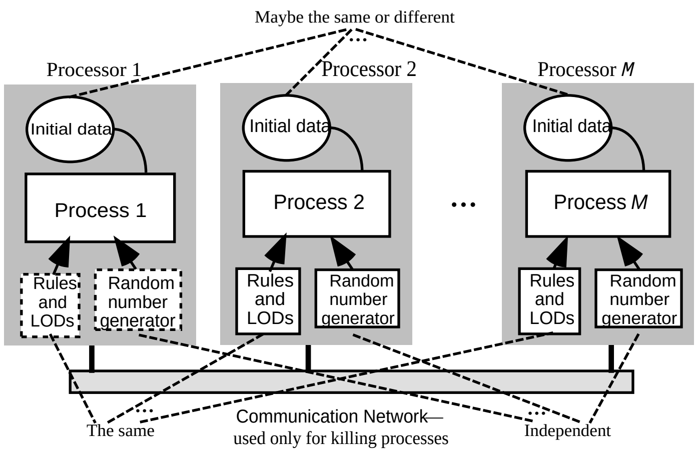

Figure 5: The method of parallel processing CCM-based computation

### 5.1 Parallel search

A parallel search method using non-communicating multiple processes is explained here.

A parallel computer with at least $M$ processors is used (Figure 5). Only one process runs on each processor. A processor and a process are thus identified here. The same reaction rule and LOD are stored in each processor. The initial data may be the same or different for all the processes. The computation of each process, which is based on CCM, is performed independently. Each process generates independent random numbers for selecting data to be reacted. This independence is very important. When a process finishes its computation, the solution found is output, and all other 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. This computation can be regarded as a search for a solution in combinatorial problem solving. Processes probably search in different areas in the search space, if the search space is large enough. The efficiency is thus in proportion to the number of processes. The reason for linear acceleration is explained theoretically by Kanada [Kan 94c].

### 5.2 Parallel reaction

A parallel reaction method of constraint satisfaction using CCM with FAM is shown in the present section.

Reactions may occur in parallel in CCM. Thus, if the working memory is shared between parallel processors, the computation is parallelized. This is shown in Figure 6. However, if two processes try to modify an atom concurrently, a wrong result may be obtained in general. The analogy to chemical reaction systems is useful here.

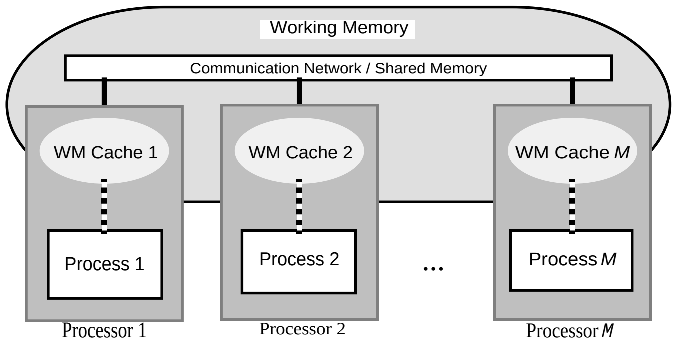

Figure 6: Parallel processing of CCM-based computation

In chemical systems, two reactions that affect the same bond do not occur at the same time. The same principle must be applied to CCM. Atoms to be modified by the reaction must be accessed exclusively.

However, fortunately, because the rules of coloring modify only one atom at a time, no wrong result can be obtained by parallel reactions without explicit mutual exclusion in this case. The only possible new effect caused by parallel reaction is that the sum of LODs may be decreased accidentally by parallel reactions even when the frustration is small. This is because, after a process tested the reaction condition but before rewriting an atom, another process may rewrite the atom. This effect seems to be small because the probability of two or more reactions occurring on the same atom concurrently is small, if there are enough atoms and the degree of parallelism is not massive. Thus, no mutual exclusion is necessary here. This non-strictness in parallel processing is made possible by the stochastic nature of CCM.

The following problem has to be solved to improve performance. Each parallel process works independently in principle in this method. A process tests termination condition without testing the states of other processes if it follows the principle strictly. However, this sometimes causes premature termination. Some processor may run for long time after other processors terminate, because the termination test is stochastic. This load imbalance lowers the degree of parallelism, and causes performance degradation. This actually occurred in our experiments.

To solve this load imbalance problem, weak global communication is introduced. The termination test is modified as follows. A counter variable is allocated and is set to the number of processes when the system is initialized. Each process decrements the counter (where mutual exclusion is necessary) when its local termination condition becomes satisfied. However, the process continues its task until the counter value becomes zero. This technique avoids premature termination and the consequent performance degradation.

## 6 Experiments on Large-scale Problems

The results of evaluating the two methods using several large-scale problems are shown here. The performance of sequential processing is shown first for comparison.

### 6.1 Sequential performance

The problems used to evaluate performance are chosen from DIMACS benchmarks [Tri][DIM]. Some have been used by Johnson, et al. [Joh 91] and by Selman and Kautz [Sel 93a].

Table 1: The performance of coloring by annealed CCM compared with annealing methods and GSAT ($c = 2$)

| Graph | $N_C$ | CCM Seq | CCM Par | SA | GSAT |
|---|---|---|---|---|---|
| 125.1 | 6 | 0.2 (−5) | 0.2 (−5) | < 360 | |
| 125.1 | 5 | 13.2 (−5) | 1.5 (−5) | 720 | |
| 125.5 | 18 | 34.0 (−15) | 2.4 (−15) | 360 | 44 |
| 125.5 | 17 | 3113 (−30) | 149 (−30) | 6700 | 1800 |
| 125.9 | 44 | 208 (−30) | 13.6 (−30) | 1080 | |
| 125.9 | 43 | - | - | - | |
| 250.1 | 9 | 2.9 (−4) | 0.3 (−4) | < 360 | |
| 250.1 | 8 | 298 (−10) | 17.6 (−10) | 9360 | |
| 250.5 | 31 | 239 (−35) | 18.9 (−35) | 360 | |
| 250.5 | 30 | 513 (−35) | 47.9 (−35) | 2900 | |
| 250.5 | 29 | 5111 (−45) | 360 (−45) | 22000 | 4000 |

The name of graph is DSJC$N.p$, where $N$ is the number of vertices and $p$ is the probability of existence of edge. The performance is shown by CPU time in seconds. The parentheses contain $\log_{10} f_0$. A single CPU of CS6400 is used for the sequential processing, and twelve CPUs are used for the parallel processing. The results of SA (simulated annealing) are by Johnson, et al. and those of GSAT are by Selman and Kautz. Hyphens mean that no solution could be found.

The results are shown in Table 1. The graphs used here, which were randomly generated and used by Johnson et al., have 125 or 250 vertices, and the probability of existence of edge, $p$, is 0.1 to 0.9. The performance of annealed CCM is comparable to GSAT or the methods tested by Johnson et al. in several cases. The parameters used here, which are kept constant during the computation, are shown in the table. Each case has been measured at least 20 times, and the results shown in the table are the averages except the cases in which the system failed to find a solution. The probability that the system fails to find a solution is less than 0.05 experimentally, with appropriate parameter values. Although a compiler and interpreter for SOOC is available, a reaction rule and LOD coded using C are used in these measurements, for better performance of the program written by C. The performance has been measured on a Cray Superserver 6400 (CS6400), which is a parallel computer with 60MHz SuperSPARC processors, but only one CPU is used here.

### 6.2 The performance of Parallel search

The parallel search method has been applied to several combinatorial problems, and the measured acceleration ratios by the parallelization are shown here.

The performance of solving the graph or map coloring problem has been evaluated by the method using simulation. The simulated performance is shown in Figure 7. 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.

Performance of real parallel processing has not yet been measured for the coloring problem. However, the performance of the $N$ queens problems was measured by Kanada [Kan 94c]. The method and the result are summarized here. The rule and LOD are hand-coded using C on a Cray Superserver 6400 (CS6400). The CS6400 has shared memory, and multiple threads (not UNIX processes) that run on different processors but share the same memory, are thus used for this measurement. However, the shared memory is used only for initial data distribution, final data output, and process termination.

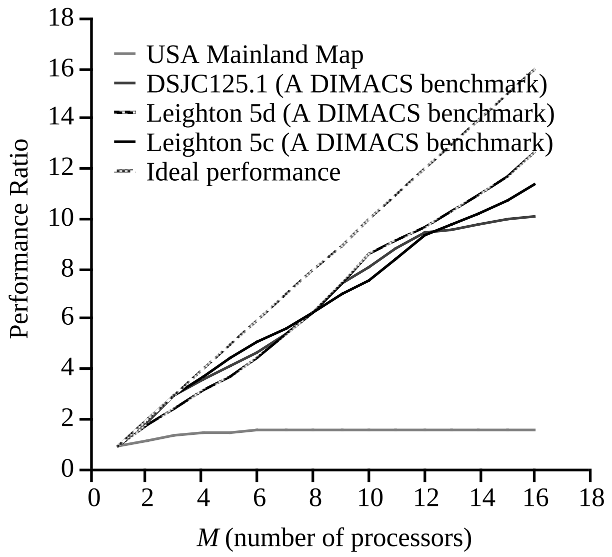

Figure 7: Simulated performance of the graph/map coloring problems

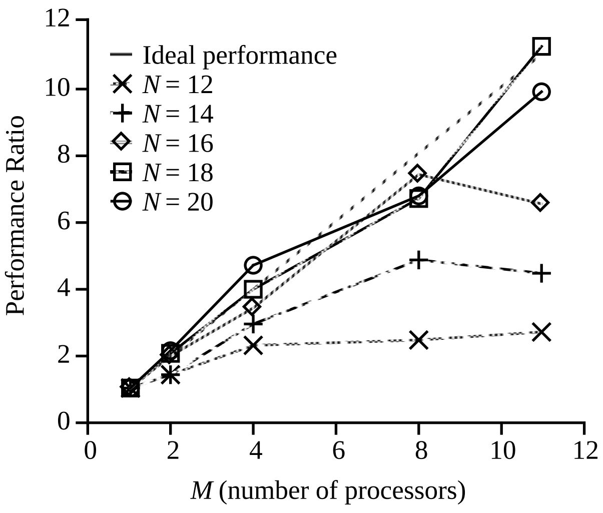

Figure 8: Parallel search performance of the $N$ queens problem

There is a parent thread that 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.) The parent then terminates and the other threads are killed by the operating system. 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 is difficult to be separated.

Each thread computes random numbers independently. The random number seed is generated using both the current time in microseconds and the address of input data for each thread. The method of seed generation was chosen carefully to guarantee the independence of random numbers between threads.

The execution time was measured 50 times for each $N$. The result of measurement is shown in Figure 8. The CS6400 used for the measurement has 12 processors, each of which a thread is assigned to, and there is also a parent thread. The maximum value of $M$ is thus 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 this problem. The reasons that the performance is worse than the simulation are probably as follows:

- There is parallelization overhead, which is caused by the time required for thread dispatching and final synchronization.
- The measured time includes the initialization time, which is not accelerated by the parallelization.

Both reasons cause the distribution of execution time to be different from an exponential distribution.

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. Other problems, including exchange sort and the traveling salesperson problem (TSP), are also simulated. The execution time of exchange sort, which can be regarded as a constraint satisfaction problem, is almost constant. Therefore, it is almost never accelerated. The acceleration ratio was 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 method because of the non-local nature of their computation. They are global optimizations.

### 6.3 The performance of parallel reaction

The parallel reaction performance of the same problems, as shown in the preceding sections, has been measured on CS6400 with 12 processors. A relatively small number of processors is used because the performance will not improve much by using hundreds or thousands of processors; only 125 atoms are used in these problems.

The performance of four problems, DSJC125.1 with 5 and 6 colors, and DSJC125.5 with 17 and 18 colors, is displayed in Figure 15. The parameter values of FAM used in the evaluations are fixed; $q = 8, c = 1.05, f_0 = 0.8$. Each case is measured at least 20 times, and the results are the averages. The performance of the 5 and 17 color problems is close to ideal. They are acceler-ated nearly linearly. However, the performance of the 6 and 18 color problems is less than the half the ideal. The reason for this is not yet known, but the performance is known to be better for some other parameter values. For example, in the case of 18 colors with $M = 12, c = 1.2$ and $f_0 = 0.2$, the acceleration ratio was 10.3.

The effect of load balancing is measured for DSJC125.1 with 5 colors. The result for 100 runs is shown in Table 2. The standard deviation of CPU time is reduced to 0.4, and the average CPU time is reduced to 0.75.

## 7 Conclusion

Two parallel processing methods for solving constraint satisfaction problems using CCM with FAM are surveyed in the present paper. FAM makes large-scale constraint

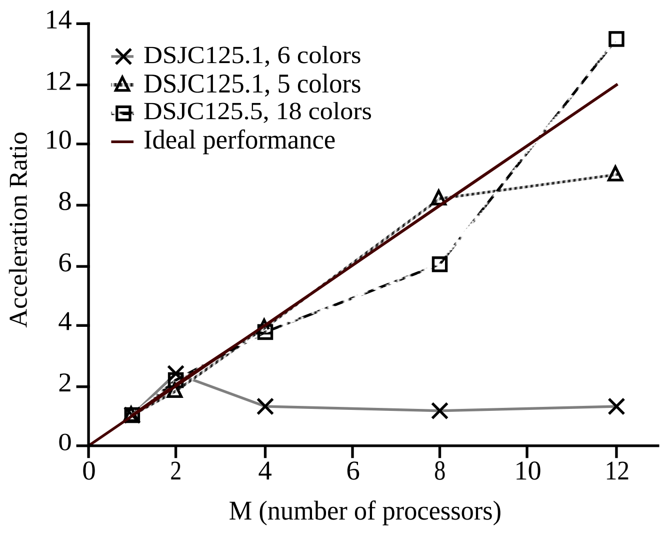

Figure 9: Parallel reaction Performance of coloring problems by parallel annealed CCM

Table 2: The effect of load balancing (DSJC125.1, 5 colors, 12 processors, $f_0 = 10^{-5}, c = 2$)

| Load balance | CPU time | $\sigma$ of CPU time |
|---|---|---|
| Balanced | 15.1 sec | 11.5 sec |
| Not balanced | 20.1 sec | 28.4 sec |

$\sigma$ means the standard deviation.

satisfaction using CCM possible, without spoiling the feature of CCM, i.e., local-information-based computation. No global functions or global parameters such as temperature are used in FAM. The performance is comparable to conventional simulated annealing or GSAT. Because of the nonexistence of global information references, CCM with FAM can be parallelized ealily The performance is improved nearly linearly by parallelization in in graph coloring problems in DIMACS benchmarks and several other problems both in parallel search and parallel reaction.

## Acknowledgment

The author thanks to Susumu Seki and Hironobu Takahashi of Tsukuba Research Center of Real World Computing Partnership, for their assistance in use and programming of the Cray Superserver 6400.

## References

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

[Fei 91] Feige, U., Goldwasser, S., Lovász, L, Safra, S., and Szegedy, M.: Approximating Clique is Almost NP-complete, *32th Symposium on Foundations of Computer Science*, 2–12, 1991.

[For 81] Forgy, C. L.: *OPS5 User's Manual*, Technical Report CMU-CS-81-135, Carnegie Mellon University, Dept. of Computer Science, 1981.

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

[Gu 93] Gu, J.: Local Search for Satisfiability (SAT) Problem, *IEEE Trans. on Systems, Man, and Cybernetics*, 23:4, 1108–1129, 1993.

[Joh 91] Johnson, D. S., Aragon, C. R., McGeoch, L. A., and Schevon, C.: Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning, *Operations Research*, 39:3, 378–406, 1991.

[Kan 92] Kanada, Y.: Toward Self-organization by Computers, *33rd Programming Symposium*, Information Processing Society of Japan, 1992 (in Japanese).

[Kan 93] Kanada, Y.: Features of Problem-Solving Method using Computation Model CCM, based on Production Rules and Local Evaluation Functions, *SWoPP '93*, Information Processing Society of Japan, 1993 (in Japanese).

[Kan 94a] Kanada, Y., and Hirokawa, M.: Stochastic Problem Solving by Local Computation based on Self-organization Paradigm, *27th Hawaii Int'l Conf. on System Sciences (HICSS-27)*, 82–91, 1994.

[Kan 94b] Kanada, Y.: Methods of Controling Locality in Problem Solving using CCM: A Model for Emergent Computation, *SWoPP '94*, 94-AI-95-4, 29-38, 1994 (in Japanese).

[Kan 94c] Kanada, Y.: A Method of Independent Parallel Processing of Constraint Satisfaction and Other Problems using CCM: A Model for Emergent Computation, *49th National Conference*, Information Processing Society of Japan, 4-321–322, 1994 (in Japanese).

[Kan 95] Kanada, Y.: Fuzzy Constraint Satisfaction Using CCM — A Local Information Based Computation Model, *FUZZ-IEEE/IFES '95*, 2319–2326, Yokohama, Japan, 1995.

[Meh 85] Mehrotra, R., and Gehringer, E. F.: Superlinear Speedup Through Randomized Algorithms, *14th ICPP*, 291–300, 1985.

[Min 92] Minton, S., Johnston, M. D., Philips, A. B., and Laird, P.: Minimizing Conflicts: a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems, *Artificial Intelligence*, 58, 161–205, 1992.

[Mor 93] Morris, P.: The Breakout Method For Escaping From Local Minima, *AAAI '93*, 40–45, 1993.

[New 72] Newell, A., and Simon, H. A.: *Human Problem Solving*, Prentice-Hall, N. J., 1972.

[Sel 92] Selman, B., Levesque, H. J., and Mitchell, D. G.: A New Method of Solving Hard Satisfiability Problems, *AAAI '92*, 440–446, 1992.

[Sel 93a] Selman, B., and Kautz, H.: Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems, *IJCAI '93*, 290–295, 1993.

[Sel 93b] Selman, B., and Kautz, H. A.: An Empirical Study of Greedy Local Search for Satis-fiability Testing, *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*.