Control and Cybernetics

vol. **31** (2002) No. 2

Binary neural networks for N-queens problems and their VLSI implementations

by

Nobuo Funabiki[1], Takakazu Kurokawa[2], and Masaya Ohta[3]

[1] Department of Communication Network Engineering,
Okayama University, Okayama, 700-8530, Japan

[2] Department of Computer Science,
National Defense Academy, Yokosuka, 239-8686, Japan

[3] Department of Electrical and Electronic Systems,
Osaka Prefecture University, Sakai, 599-8531, Japan

Abstract: Combinatorial optimization problems compose an important class of mathematical problems that include a variety of practical applications, such as VLSI design automation, communication network design and control, job scheduling, games, and genome informatics. These problems usually have a large number of variables to be solved. For example, problems for VLSI design automation require several million variables. Besides, their computational complexity is often intractable due to NP-hardness. Neural networks have provided elegant solutions as approximation algorithms to these hard problems due to their natural parallelism and their affinity to hardware realization. Particularly, binary neural networks have great potential to conform to current digital VLSI design technology, because any state and parameter in binary neural networks are expressed in a discrete fashion. This paper presents our studies on binary neural networks to the N-queens problem, and the three different approaches to VLSI implementations focusing on the efficient realization of the synaptic connection networks. Reconfigurable devices such as CPLDs and FPGAs contribute the realization of a scalable architecture with the ultra high speed of computation. Based on the proposed architecture, more than several thousands of binary neurons can be realized on one FPGA chip.

Keywords: binary neural network, NP-hard, combinatorial optimization, VLSI design, algorithm

1. Introduction

Since Hopfield and Tank introduced an innovative idea of solving an NP-hard

field and Tank, 1985), the studies on neural network approaches have played active roles in various areas such as computations, algorithms, VLSI designs, signal processing, and machine learning. Although the deficiencies of their works have been strongly criticized, (Wilson and Pawley, 1988; Paielli, 1988), several ideas and new methods have been presented, means toovercome this difficulty and open the door to a new era. These new approaches include the binary neural network, (Takefuji, 1992), the Boltzmann machine, (Hinton, Sejnowski and Ackley, 1984), the Cauchy machine, (Takefuji and Szu, 1989), the chaotic neural network, (Nozawa, 1992), the mean field annealing (Jagota, 1995), the Lagrange multiplier method (Wieselheier, Barnhart and Ephremides, 1994), and the gradient ascent learning, (Wang, Tang and Cao, 2002).

The Boltzmann machine incorporates the mechanism of stochastic state transitions in the parallel computation of neural networks, to avoid local minimum convergence. The stochastic state transitions originated from simulated annealing, (Kirkpatric, Gelatt Jr. and Vecchi, 1983). The Cauchy machine is based on a different distribution from the Boltzmann machine, so as to speed up convergence. The chaotic neural network introduces the chaotic behavior of state transitions to the neural network so that it can avoid the undesired local minimum convergence. However, it suffers from the troublesome requirement of adjusting many parameters for better performance. The mean field annealing is an approach to dynamically change the state of the system from continuous to discrete, so that it can avoid the local minimum convergence. The computation of the meaning field annealing is usually slow in terms of convergence. The discrete usually generates more local minima than the continuous one. The Lagrange multiplier method dynamically changes coefficients of the energy function to be minimized during the state update computations. The energy function describes the constraints and the objective function of the problem, and is usually given by the linear sum of the terms representing the conditions. The gradient ascent learning dynamically changes coefficients of the energy function for the positive gradient direction of the energy function.

The binary neural network adopts the binary neuron model, which has a long history in the neural network research since McCulloch and Pitts (1943) proposed the first neuron model in the binary form. In addition to the binary discrete state of neuron outputs, our binary neural network requires the discrete state for neuron inputs and the integer formulations of the system parameters including state transition thresholds for neuron outputs and coefficients for the energy function. The integer neuron input may take a limited range such as $[-32, +31]$ or $[-64, +63]$ so that the required bit width for neuron inputs becomes only six or seven. This limited bit requirement provides a great advantage when the binary neural network is actually implemented on VLSI hardware.

As the past research has expressed the deficiency, however, the discrete form of the solution search can easily bring on the problem of the local minimum convergence. The neural network in the local minimum cannot move to other

network always generates a number of undesired local minima because of the discrete state definition. In order to deal with this negative fact, several heuristic methods have been proposed to escape from the local minimum so that the binary neural network can be applied to a variety of NP-hard combinatorial optimization problems. The first method is the *hill-climbing term* that encourages the necessary number of neurons to have nonzero outputs. This term is actually added to the motion equation as an additional term besides the necessary terms that are derived from the energy function. As seen later, the motion equation usually provides each neuron more negative force than positive force so as to satisfy the constraints of the problem. The hill-climbing term supplements the positive force to satisfy the constraints. The second method is the *omega-function* as another form meant to alleviate the negative force of the motion equation. The omega-function allows the negative force to be active only for the neurons that have nonzero outputs. Besides, the *reinforced self-feedback* in the motion equation is effectively introduced as another method to avoid the local minimum convergence.

It is a huge advantage for the binary neural network to form a massive parallelism in a natural manner, and to be suitable for the VLSI implementation. Thus, we present VLSI implementations of the binary neural network in this paper, focusing on the efficient realization of the synaptic connection networks. Our implementations include a bus-connected architecture of the binary neural network, a systolic array implantation, and a logical synaptic connection in the maximum neural network with the reinforced self-feedback.

The paper is organized as follows: Section 2 briefly runs through the relationship between the combinatorial optimization problem and the binary neural network. Section 3 reviews the application of the binary neural network for the N-queens problem and presents a new binary neural network approach called the "maximum neural network with the reinforced self-feedback". Section 4 presents three VLSI implementations of the binary neural network for the N-queens problem. Finally, Section 5 provides the concluding remarks of this paper.

2. Binary neural network and combinatorial optimization problem

Let $x = (x_1, x_2, \ldots, x_n)$ be a set of n variables whose values should be found in a combinatorial optimization problem. Then, the combinatorial optimization problem requires finding an integer value assignment to each of n variables, such that a set of the constraints is not only satisfied, but also the objective function is optimized. A constraint in a problem may be described by:

$$f(x) = 0 \tag{1}$$

where the function $f(x)$ is usually nonlinear for NP-hard problems. An objective function may be described by:

Generally, each variable x_i for a combinatorial optimization problem takes an integer value within certain range. However, through the variable transformation, we can always adopt a binary formulation of variables y_{ij}, such that $y_{ij} = 1$ represents $x_i = j$, and $y_{ij} = 0$ represents $x_i \neq j$. Thus, we can focus on a combinatorial optimization problem with binary variables.

In a binary neural network in this paper, each neuron has an integer input U_{ij} and a binary output V_{ij}, and a nonlinear, nondecreasing function called the neuron function, to compute the output using the input:

$$V_{ij} = s(U_{ij}). \tag{3}$$

The simplest form of the neuron function is the binary model by McCulloch and Pitts:

$$\text{if } U_{ij} > 0 \text{ then } s(U_{ij}) = 1, \text{ else } s(U_{ij}) = 0. \tag{4}$$

Thus, a binary neural network can be seen as a nonlinear function connecting multiple inputs and multiple outputs. Then, the outputs are fed back into the inputs through a set of differential equations, which is called the motion equation. After the initial states for neuron inputs and outputs are given, this closed loop between neuron inputs and outputs cause dynamical state transitions, until the state of the binary neural network reaches a stable state. The stable state may represent a solution of the problem. Here, the binary output directly corresponds to the binary variable of the combinatorial optimization problem, and the state of the binary neural network means the state of neuron outputs.

The energy function should be defined to solve each combinatorial optimization problem by a binary neural network. The output of the energy function becomes minimum when the state of the binary neural network reaches a solution of the problem. The energy function E is a linear sum of terms representing the constraints and the objective function:

$$E = Af(V) + Bg(V) \tag{5}$$

where A and B are weight coefficients and V is a vector of neuron outputs. Note that these coefficients are constant in the binary neural network, whereas they are dynamically changed in the Lagrange multiplier method and the gradient ascent learning. The details on the application of the binary neural network to combinatorial optimization problems will be discussed in the following section.

Then, the motion equation is derived through the partial derivative of the energy function:

$$\Delta U_{ij} = -\frac{\partial E}{\partial V_{ij}}, \qquad U_{ij} = U_{ij} + \Delta U_{ij}. \tag{6}$$

We note that, Wang (1997), shows that Eq. (6) does not always lead to convergence for an arbitrary energy function. The motion equation determines the

constraints and the objective function in the target combinatorial optimization problem. The motion equation guides the state transitions so that the final stable state represents a solution of the problem. In the final stable state, it is requested that all the constraints of the problem be satisfied and the objective function minimized.

3. Application to N-queens problem

In this section, we introduce two binary neural network approaches to the N-queens problem. The N-queens problem is a typical combinatorial optimization problem of finding a set of locations of N queens on an $N \times N$ chessboard subject to the constraint that no pair of queens may be in each other's line of movement. Since Gauss failed to find all of the 92 different solutions for the 8 queens problem, Falkowski and Schmitz (1986), the N-queens problem has been widely used as a benchmark problem to evaluate a variety of search algorithms, such as the backtracking method, Bitner, Reingold (1975); Stone and Stone (1987), the branch-and-bound method, Abramson and Yung (1989), and the variable ordering heuristic method, Kale (1990). An algorithm for finding at least one solution for every N has also been proposed, using the symmetry of the solution, Reichling (1987); Falkowski and Schmitz (1986).

3.1. Binary neural network and three updating modes

In this subsection, we show the binary neural network formulation for the N-queens problem from Yoshio, Baba, Funabiki and Nishikawa (1997), so as to introduce a binary neural network approach for a combinatorial optimization problem. Then, we compare the performance of three modes to update the state of the neural network, namely, the sequential mode, the N-parallel mode, and N^2-paralle mode. The binary neural network is composed of $N \times N$ neurons to solve the N-queens problem. In the sequential mode, the state of each neuron is updated sequentially, where a state represents the input and the output of the neuron. In other words, states of no two neurons are updated simultaneously. In the N-parallel mode, the N^2 neurons are first divided into disjoint N groups such that each group is composed of N neurons. The states of N neurons in the same group are updated simultaneously, while each group is updated sequentially. Actually, in this binary neural network, the N neurons representing N diagonal locations of $((i+k \bmod N)+1, (j+k \bmod N)+1)$ for $k = 1, \ldots, N$, are grouped together into the same group. Note that "mod" represents the modulo function. Then, each of N neurons for the same row or column is assigned to different groups. In the N^2-parallel mode, all the states of N^2 neurons are updated

3.1.1. Energy function and motion equation

In a binary neural network approach, the binary output of a neuron should be defined to solve the N-queens problem. The binary output V_{ij} represents whether a queen be assigned at the i-th row and the j-th column, or not. The output $V_{ij} = 1$ represents a queen assigned there, and the output $V_{ij} = 0$ represents no assignment. Then, the computational energy function E must be defined as described in Section 2. The energy function becomes zero, if and only if every constraint of the N-queens problem is satisfied. Actually, two constraints must be satisfied for this problem: 1) one queen must be located in each row and each column, and 2) no more than one queen must be located in any diagonal line. The corresponding energy function is given by:

$$E = \frac{A}{2}\Big\{\sum_{i=1}^{N}\Big(\sum_{k=1}^{N}V_{ik}-1\Big)^2+\sum_{j=1}^{N}\Big(\sum_{k=1}^{N}V_{kj}-1\Big)^2\Big\}$$
$$+\frac{B}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}V_{ij}\Big(\sum_{1\le i+k,j+k\le N,k\neq 0}V_{i+k,j+k}$$
$$+\sum_{1\le i+k,j-k\le N,k\neq 0}V_{i+k,j-k}\Big) \tag{7}$$

where A and B are coefficients. The A-term in Eq. (7) represents the first constraint, and the B-term represents the second one.

Then, the motion equation is derived to seek the state of the neural network satisfying $E = 0$ by the gradient descent method:

$$\Delta U_{ij} = -A\Big\{\Big(\sum_{k=1}^{N}V_{ik}-1\Big)+\Big(\sum_{k=1}^{N}V_{kj}-1\Big)\Big\}$$
$$-B\Big(\sum_{1\le i+k,j+k\le N,k\neq 0}V_{i+k,j+k}+\sum_{1\le i+k,j-k\le N,k\neq 0}V_{i+k,j-k}\Big)$$
$$+C\Big\{h\Big(\sum_{k=1}^{N}V_{ik}\Big)+h\Big(\sum_{k=1}^{N}V_{kj}\Big)\Big\} \tag{8}$$

where $A = B = 1$, and $C = 4$ if (t mod $20 < 5$), $C = 1$ otherwise; t is the number of iteration steps for solving the motion equation by the first-order Euler method. The C-term is the hill-climbing term for escaping from local minimum. The hill-climbing function $h(x) = 1$ if $x = 0$, 0 otherwise. The C-term is activated when no neuron for each row or column has output = 1.

3.1.2. Performance comparison between three modes

The neural network for N-queens problem is simulated on three computation

Table 1. Simulation results of the binary neural network for N-queens problem.

N	sequential		N-parallel		N^2-parallel	
	rate	step	rate	step	rate	step
8	94%	24	89%	97	54%	66
10	88%	122	80%	114	26%	88
20	100%	66	99%	79	47%	131
50	100%	50	100%	60	90%	169
100	100%	56	100%	63	90%	171
200	100%	67	100%	77	0%	-
300	100%	65	100%	91	0%	-
400	100%	75	100%	93	0%	-
500	100%	95	100%	107	0%	-

a total of 100 simulation runs are performed from different randomized initial states of the neurons, and the convergence rate and the average number of iteration steps required for convergence are evaluated. The iterative computation of the neural network is terminated by the time-out procedure when it cannot converge to a solution after 500 iteration steps have passed. The convergence rate is the number of simulation runs among 100 runs in which the neural network can find a solution of locating N queens without violating the constraints within 500 steps. Table 1 shows the simulation results for this neural network, where the average number of iteration steps is calculated from the results only where the neural network converges to a solution. "-" in the table means that no result is obtained there because of no convergence.

The results in Table 1 indicate that the sequential mode provides the best performance in terms of the convergence rate and the speed, while the N^2-parallel mode does the worst one. In the sequential mode, any neuron updates its state to minimize the energy function after all the states of the other neurons are updated to the latest ones. On the other hand, in the N^2-parallel mode, any neuron updates its state based on the previous states of neurons. This delay of state updating causes the oscillation of the state transitions. In the N-parallel mode, the N neurons for the same row or column, which are competing each other, are updated sequentially, while the N neurons for the same diagonal line are updated simultaneously. This combination of the sequential updating and the parallel updating realizes the highly parallel computation with the suppression of oscillations at the same time.

3.2. Maximum neural network with reinforced self-feedback

The maximum neural network (MNN) has been proposed in Lee, Funabiki and

neural network. MNN can always satisfy one constraint in a combinatorial optimization problem. MNN is also suitable for the computation on the N-parallel mode. However, MNN often suffers from the problem of the local minimum convergence. Thus, we combine the idea of the chaotic neural network (CNN) from Nozawa (1992) with MNN to alleviate this negative effect. A neuron in CNN has a negative self-feedback to cause the chaotic oscillation, Ohta, Ogihara, Takamatsu and Fukunaga (1995), so that the state in CNN can escape from a local minimum by using its chaotic behavior. Ohta, Anzai, Yoneda and Ogihara (1993); Ohta, Ogihara and Fukunaga (1994) have clarified that the self-feedback connection between neurons in the binary neural network is very effective for avoiding local minima. Ohta (1999; 2002) have shown that the control algorithm for the self-feedback gain named the *reinforced self-feedback* improves the performance of CNN. This subsection presents the introduction of the reinforced self-feedback into MNN. The MNN with the reinforced self-feedback can be realized with much smaller efforts than CNN, although our MNN and CNN are both based on the non-periodical behavior so as to overcome the problem of the local minimum convergence.

3.2.1. Energy function and motion equation

In MNN for the N-queens problem, the output of the maximum neuron always satisfies the constraint that one and only one queen is assigned to each row. Thus, the output V_{ij} in MNN is given by

$$V_{ij} = \begin{cases} 1 & \text{if } U_{ij} = \max_k\{U_{ik}\} \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

Then, the energy function of MNN for the N-queens problem is defined by

$$\begin{aligned} E = & \frac{1}{2}\sum_{j=1}^{N}\sum_{i=1}^{N}\sum_{\substack{k=1 \\ k\neq i}}^{N} V_{ij}V_{kj} \\ & + \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{1\leq i+k,j+k\leq N,k\neq 0} V_{ij}V_{i+kj+k} \\ & + \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{1\leq i+k,j-k\leq N,k\neq 0} V_{ij}V_{i+kj-k} \end{aligned} \tag{10}$$

The first term represents the first constraint of the N-queens problem. While the second and third terms represent the second constraint.

The motion equation to update the neuron input U_{ij} is given by

$$U_{ij}(t+1) = U_{ij}(t) + \Delta U_{ij}(t), \quad \Delta U_{ij}(t) = -\frac{\partial E}{\partial V_{ij}}$$

$$-\Big(\sum_{\substack{k=1\\k\neq i}}^{N} V_{kj} + \sum_{1\leq i+k,j+k\leq N,k\neq 0} V_{i+kj+k} + \sum_{1\leq i+k,j-k\leq N,k\neq 0} V_{i+kj-k}\Big) \quad (11)$$

where r is the dumping constant to satisfying $|r| < 1$, and T_{ij} is the self-feedback gain for the ijth neuron. In this MNN, the state of neurons are updated on the N-parallel mode. However, in this MNN, the states of N neurons for the same row are updated simultaneously, whereas the states of N neurons for the same column are updated sequentially due to the nature of the maximum neuron model in Eq. (9).

In this MNN with the reinforced self-feedback, the self-feedback gain T_{ij} should be positive when the state of MNN needs to converge to a local minimum, and be negative when it needs to escape from the local minimum by causing the non-periodical behavior. Thus, we propose the control algorithm of the self-feed back gain as follows:

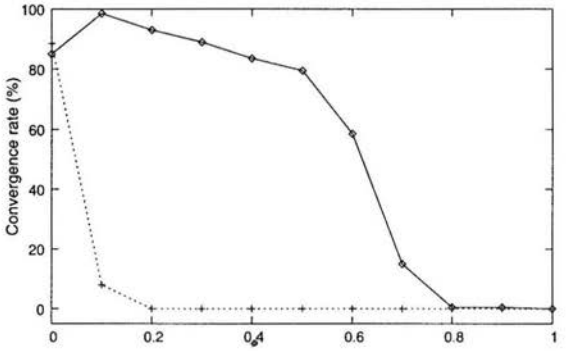
$$T_{ij} = \begin{cases} T_{ij} - \delta T & \text{if } V_{ij} = 1 \\ \omega & \text{otherwise} \end{cases} \quad (12)$$

where $\delta T\,(\geq 0)$ is a constant small value for the self-feedback gain control, and ω is the default value for the gain. This algorithm indicates that when a neuron becomes active ($V_{ij} = 1$), the self-feedback gain starts decreasing gradually, and when the neuron becomes non-active ($V_{ij} = 0$), it is reset to the default value. Eq.(11) shows that when the neuron is active, the self-feedback term increases the input U_{ij} when the gain is positive, and it decreases the input when the gain is negative. Thus, by reaching a negative gain by decreasing it, the input also decreases for the neurons with active outputs, and it encourages other neurons to be eventually active. That is the mechanism of escaping from a local minimum by changing the state in our MNN. Here, the time to change from a positive gain to a negative gain is varied for each neuron. Thus, it avoids the oscillation of the state by simultaneous state updates. Note that this method does not guarantee the convergence to a local minimum, and the convergence to a solution must be checked at every iteration step.

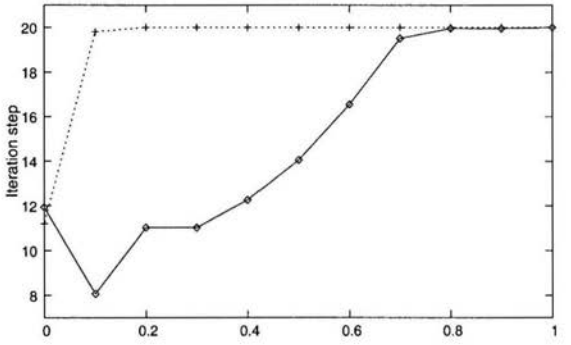
3.2.2. Performance evaluation

We evaluate the performance of MNN with the reinforced self-feedback for the N-queens problem. The problem size N is fixed at $N = 1,000$, the default value of the self-feedback gain is $\omega = 0.0$, and the decrement is $\delta T = 0.01$. Note that $\delta T = 0.0$ is used to evaluate MNN without the reinforced self-feedback. Because the proposed MNN with the reinforced self-feedback usually converges to a solution very quickly unlike the conventional binary neural network of Section 3.1, the maximum number of iteration steps is set at 20. Here, we note that MNN with $\delta T = r = 0$ is equivalent to the *min-conflict algorithm*, which is one of the well-known methods in heuristic approach for constraint satisfaction

row randomly. Then, it continues to locate a queen at the location that has minimum number of attacking queens in each row. It has been known that this min-conflict algorithm is surprisingly effective for many constraint satisfaction problems, and is able to solve the million-queens problem in an average of less than 50 iteration steps, Russell and Norvig (1995).



(a) Convergence rate

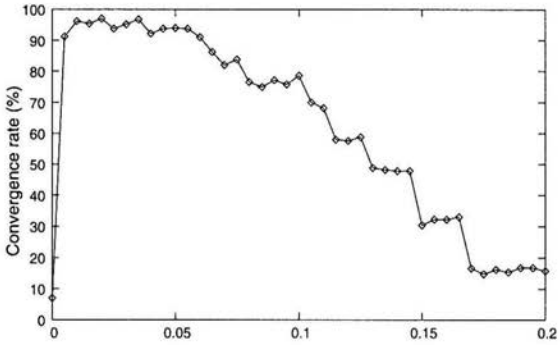


(b) Average number of iteration steps

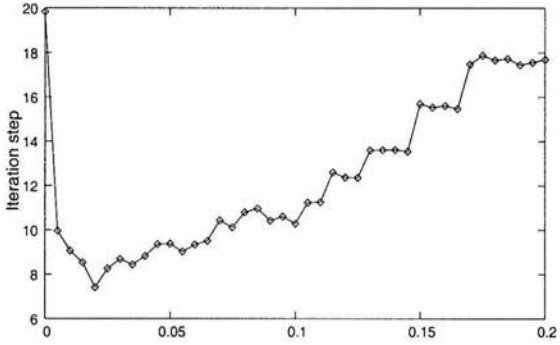
Figure 1. Performance sensitivity of the proposed MNN to dumping constant r ($N = 1{,}000$).

Fig. 1 (a) and (b) shows the convergence rates and the average number of iteration steps among 200 simulation runs with different initial states. We note that convergence means that MNN found out the solution satisfying all

iteration steps. Fig. 1 (a) shows that the proposed scheme of the reinforced self-feedback improves the convergence rate from 88.5 % with $\delta T = r = 0$, to 98.5 % with $\delta T = 0.01, r = 0.1$. Fig. 1 (b) shows that the average number of iteration steps decreases from 11.2 to 8.1. Fig. 2 shows the change of the convergence



(a) Convergence rate

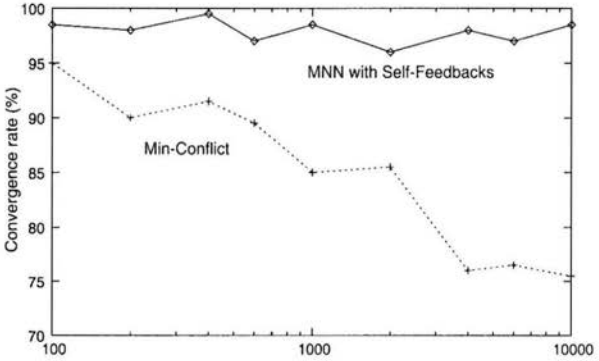


(b) Average number of iteration steps

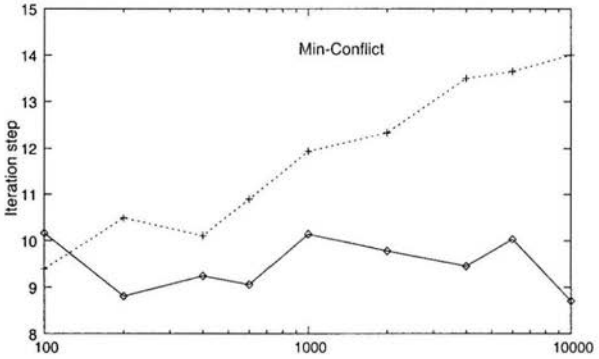
Figure 2. Performance sensitivity of the proposed MNN with repsect to δT (N=1,000).

property, when δT is varied from 0 to 0.2 among 500 simulation runs, while r is fixed to 0.1. For example, when δT is set at 0.01, the convergence rate becomes 96.2% and the average number of iteration steps is 9.1. Fig. 3 shows the change of performance between two algorithms when the number of queens N is varied from 100 to 10,000 among 200 simulation runs. In the 10,000-queens problem, our proposed MNN improves the convergence rate from 75.5%

The performance of our MNN is not disproved as N increases. Therefore, we conclude that our MNN with the reinforced self-feedback is very effective in solving large size N-queens problem.



(a) Convergence rate



(b) Average number of iteration steps

Figure 3. Performance sensitivity of the proposed MNN with repsect to problem size.

Lastly, the proposed MNN is compared with the conventional CNN and CNN with the reinforced self-feedback from Ohta (in printing). The problem size N is set at $1,000$ and the maximum number of iteration steps is $1,000$ for CNN, and 20 for the proposed MNN. The results of Table 3.2.2. show that the proposed MNN achieves a far better convergence property with much smaller number of

Table 2. Comparison of CNN and MNN (N=1,000).

method	Convergence rate [%]	Average steps
Conventional CNN	30	795.5
CNN with reinforced self-feedback	90	666.5
MNN with reinforced self-feedback	98.5	8.1

4. Hardware implementation of binary neural networks

The hardware implementation of a neural network can improve its computational ability by fully taking advantage of parallel computation by a massive number of neurons. There have been many approaches to implement neural network hardware systems, Shriver (1988); Sundararajan and Saratchandran (1998). Particularly, binary neural networks are very suitable for implementation on digital VLSI circuits. However, when we consider the digital VLSI implementation of a binary neural network for a combinatorial optimization problem, we encounter a serious problem that the implementation of synaptic connections between neurons occupies enormous area on a VLSI chip. For this problem, we restrain the chip area by limiting the applicable combinatorial optimization problem by the neural network. In this section, we propose three architectures for hardware implementation of binary neural networks focusing on the efficient synaptic connection network.

4.1. Bus connected architecture

In this subsection, we propose a bus connected architecture for the binary neural network, adopting the sequential mode to update the state of the binary neural network. The bus connected architecture is a simple static network that is typically used for mutual communication in a multiprocessor system. In this architecture, the communication through the common bus becomes the bottleneck of the performance improvement when the number of processing elements increases. First, we have checked the necessary volume of data communicated through the bus when a binary neural network is solving a combinatorial optimization problem. In this architecture, the information on active neurons must be broadcasted to every processing element to realize the neuron function through the bus. Thus, the number of active neurons becomes proportional to the total volume of communicated data. As an example, we have considered the binary neural network for the (16, 20, 5, 4, 1)-BIBD problem from Kurokawa and Takefuji (1992). Fig. 4 illustrates the change of the number of active neurons during the computation of the binary neural network. This data was derived after 1,000 simulation runs have been performed. Here, the number of active neurons is about 20 among the total of 1,820 neurons in the network. This fact supports the feasibility of using the bus connected architecture for the

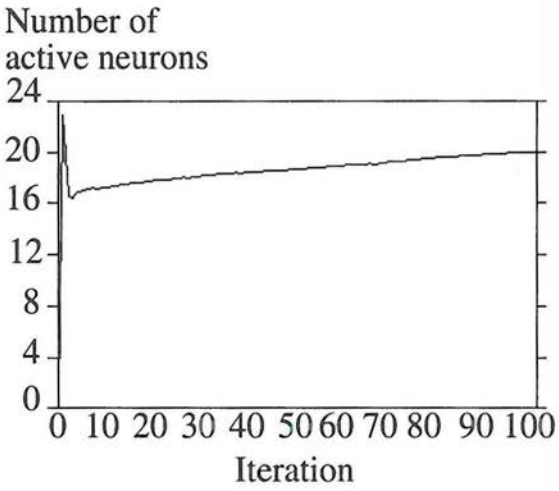


Figure 4. The transition of the number of active neurons for the (16, 20, 5, 4, 1)-BIBD problem.

4.1.1. Architecture

Figure 5 depicts our proposed bus connected architecture for the binary neural network. Each binary neuron has a unique ID number and one local arbiter to decide a bus master in parallel. The decision process of the bus master is the same as that of Futurebus system from IEEE (1988).

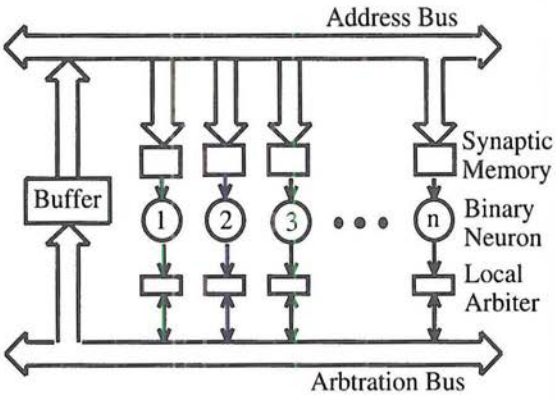


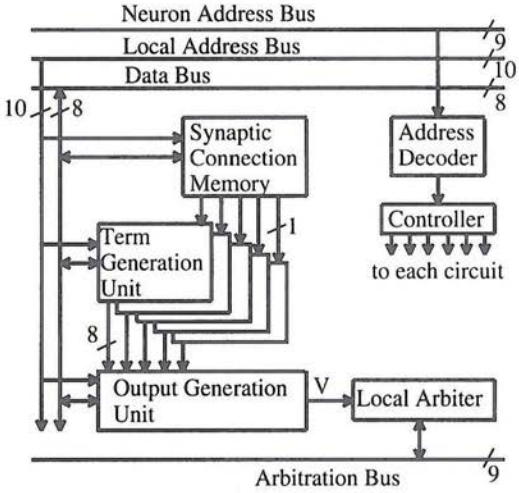
Figure 5. Bus connected neural network system.

1. Each active neuron has the right to participate in arbitration.
2. Among all participating neurons, one neuron that has the largest ID number is selected as the bus master.
3. The arbitration bus holds the inverse bit pattern of the bus master's ID number.
4. All neurons receive the ID number of the bus master through the arbitration bus, buffer, and address bus, and calculate their inputs by the motion equation.
5. All neurons update their outputs by the binary function.
6. If the state of the binary neural network converges to a local minimum, terminate the iteration, else if more than one neurons have not become bus masters, go to Step 1 after denying the participation right of the present bus master, else permit the right of all active neurons and then go to Step 1.

By limiting the rights of active neurons to participate in arbitration, we can reduce the number of bus accesses drastically.

4.1.2. Implementation

In our architecture, a single binary neuron is actually implemented as the combination of a synaptic connection memory, five term-generation units, an output-generation unit, a local arbiter, and a controller as shown in Fig. 6. The synaptic connection memory stores the information of a neuron's synaptic connection to the other neurons. Each of the term-generation units calculates one term in the



motion equation. After analyzing a variety of binary neural networks for combinatorial optimization problems, we have concluded that five term-generation units are sufficient in this architecture. The output-generation unit calculates the output of a binary neuron by adding the outputs of term-generation units to its input.

Fig. 7 shows the printed circuit board of our implementation, named a "neuron-board". This board includes eight binary neurons. The binary neurons are actually implemented on the combination of an FPGA, a PLA, a 2kbyte S-RAM, and several SSIs. Five term-generation units and one output-generation unit are realized on an FPGA including 9,000 system gates (Xilinx XC3090PC84). The local arbiter and the synaptic connection memory are realized on a PLA (Palce16V8HD) and a 2kbyte S-RAM (MN4416S-l5), respectively. As a prototype, we have actually developed five neuron-boards that are stored in a VME bus rack. Because this VME bus rack can store 512 boards, this developed system can equip up to 4,096 binary neurons by adding the neuron-boards.

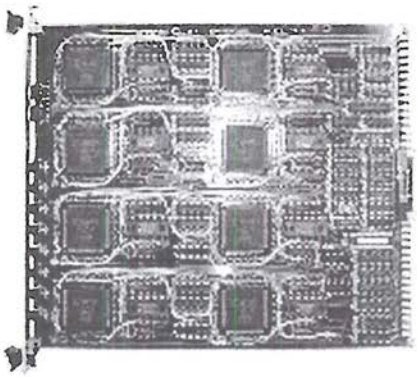


Figure 7. Neuron board.

Let us consider the maximum size of the N-queens problem to be solved by this architecture when we adopt Xilinx XC2V10000 in Virtex II family for FPGAs. This chip has 10M system gates that can contain about 1,000 binary neurons. So, 512 neuron-boards with 8 XC2V10000 chips for each board can contain about $2,000 \times 2,000$ binary neurons. Thus, up to the 2,000-queens problem can be solved on this architecture using the current FPGA technology. A personal computer controls the neuron boards as a host computer with two

4.1.3. Evaluation

The proposed architecture requires $100ns$ to determine a bus master and to broadcast its ID number through the arbitration bus. The output computation using the motion equation by binary neurons requires $200ns$. Thus, binary neurons can update their inputs by receiving an ID number of an active neuron within $300ns$. Thousands of simulation runs reveal that the binary neural network to solve 6-queens problem with 36 binary neurons requires 15.3 iteration steps to converge to a solution on average. The total computational time was $27.6\mu s$. This computational time was about 28.6 times less than that by a software simulation on a 400MHz IBM PC/AT. The features of the developed binary neural network architecture are summarized in Table 3.

Table 3. Features of binary neural network system.

Number of boards		5 (Max=512)
Number of neurons in a Neuron Board		8
Number of neurons in the whole system		40 (Max=4,096)
Clock frequency		10MHz
Arbitration time		100ns
Calculation time of neuron		200ns
Performance		156.5MCPS
Data transfer rate		6.0MB/s
Power consumption	Whole system	116.28W
	one Neuron Board	4.89W

4.2. Systolic array architecture

In this subsection, we propose a systolic array architecture for the binary neural network adopting the N^2-parallel mode to update the state of the binary neural network. Conventional researches for systolic arrays have concentrated on the VLSI design as systolic cells, Fortes (1987). The advancement of VLSI technology enables us to realize a special purpose chip with reconfigurable devices such as CPLDs and FPGAs. A hardware system with reconfigurable devices introduces a flexible scalability. Following these circumstances, we have developed a prototype of a systolic array architecture composed of 9×9 systolic cells to realize a 9×9 binary neural network for the 9-queens problem.

4.2.1. Architecture

For the N-queens problem, we present the architecture of $N \times N$ systolic cells arranged in the 2-D matrix fashion similar to a chessboard. Each systolic cell accords with a hysteresis neuron, where an active neuron means that a queen is placed at the corresponding position of the chessboard as shown in 3.1.1. The octagonal cell shown in Fig. 8 represents a systolic cell that is placed at the (i, j)-th position in our architecture. When this architecture starts the neural network computation, all systolic cells are connected in a line. Then, $N \times N$ initial values for inputs U_{ij} $(i, j = 1, 2, \ldots, N)$ are randomly generated with negative integers at a host computer. These values are transmitted to $N \times N$ systolic cells synchronizing with the system clock from the host computer systolically, and are stored at "U-Reg." in each systolic cell. At the same time, "V-Reg." in each systolic cell is initialized to zero.

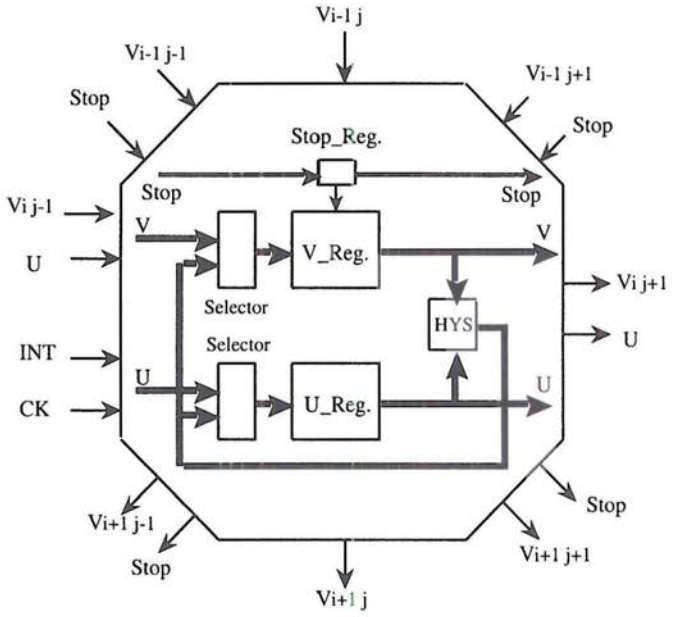


Figure 8. Block diagram of a systolic cell.

During the neural computation process, the systolic cell at the (i, j)-th position receives the information on active neurons from the systolic cells placed in the same row, the same column, and the same diagonal directions. For this purpose, the systolic array transmits the neuron output V_{ij} $(i, j = 1, 2, \ldots, N)$

movements. Then, the hysteresis function block ("HYS") calculates the neuron input U_{ij}. After that, the systolic cells update their values on "V-Reg." and "U-Reg.". This procedure is repeated in 500 iteration steps in our experiments. If the binary neural network finds a solution of an N-queens problem within 500 iteration steps, the host computer receives the final neuron inputs U_{ij} from all the systolic cells through the same systolic path as the initial load. Otherwise, the host computer begins the next computation by changing initial values of neuron inputs U_{ij}.

The neuron outputs V_{ij} are transmitted through the connected lines as shown in Fig. 9 systolically by synchronizing with the system clock. Each systolic cell is connected with its adjacent eight cells for these systolic data movements as shown in Fig. 9. The total of N clock cycles are required for these data movements in the row direction and the column direction. For the diagonal direction, the required number of clock cycles differs by the position of a cell. We prepared a "stop" signal to prevent the unnecessary data movements in these directions.

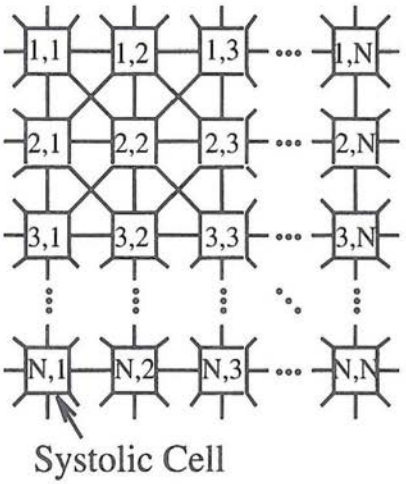


Figure 9. Structure of our systolic array.

4.2.2. Implementation

As the prototype, we have developed a circuit board named "Systolic array board". This board includes 11 CPLDs (XC95108-PC84) with 2,400 usable gates for each one. Fig. 10 shows our systolic array board containing: (a) 9 CPLDs as a systolic array, (b) 9×9 LEDs arranged in 2-D array fashion, (c)

between the host computer and our systolic array board, and (e) a clock oscillator. In our systolic array board, we have designed 9 systolic cells within one CPLD. Thus, the board contains 81 systolic cells totally. Table 4 summarizes the evaluation result of our systolic array architecture from the standpoints of features and required resources.

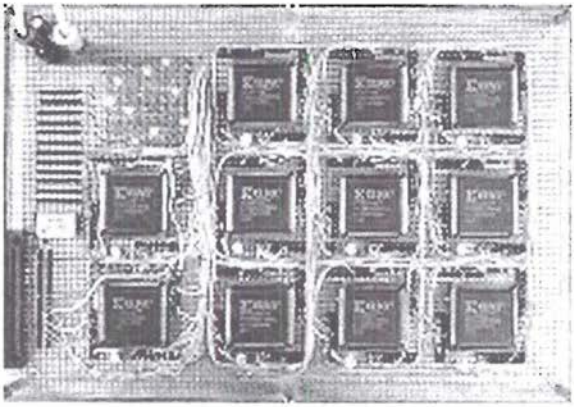


Figure 10. The developed systolic array board.

This proposed architecture is based on the systolic array with a flexible scalability using reconfigurable devices. Therefore, our architecture can be easily expanded by adding extra systolic cells and/or by changing the size of reconfigurable devices. Let us assume that we use XC2V10000 in Virtex II family instead of CPLDs. This chip can contain about 4,500 systolic cells. So, 3×3 XC2V10000 chips for the systolic array can contain about 200×200 binary neurons to solve up to the 200-queens problem. However, the N^2-parallel method could not find a solution over 100-queens problems. Currently, we are modifying this architecture to N-parallel method.

Table 4. Features of the developed systolic board.

Systolic cells	81
Operation Frequency	17.6MHz
Computation Time	14.1usec
Computation Speed	3.71GCPS
CPLD Chip	XC95108-PC84
Macrocells	100 (92%)
Product terms	350 (64%)
I/O pins	64 (92%)

4.3. Logical synaptic connection architecture

In this subsection, we propose a logical synaptic connection architecture for the maximum neural network with reinforced self-feedback. As described before, our architecture for MNN adopts the N-parallel mode. In our MNN for the N-queens problem, the neuron input can be updated, if the information on the existence of active queens on the same column and its diagonal lines is given there. When a neuron encounters no violation from conflicting neurons, a queen should be located on the corresponding place. Therefore, the proposed architecture has only logical sum (OR-gate) to realize connections between neurons (see Fig. 11). This logical sum operation calculates whether the neuron encounters violations or not.

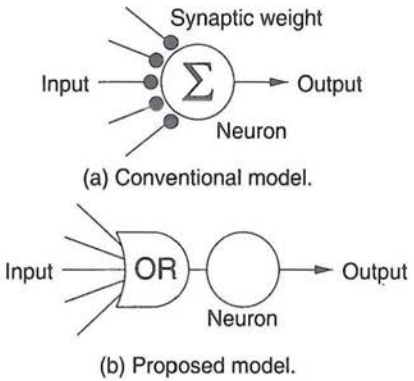


Figure 11. Logical synaptic connection.

4.3.1. Logical synaptic connection

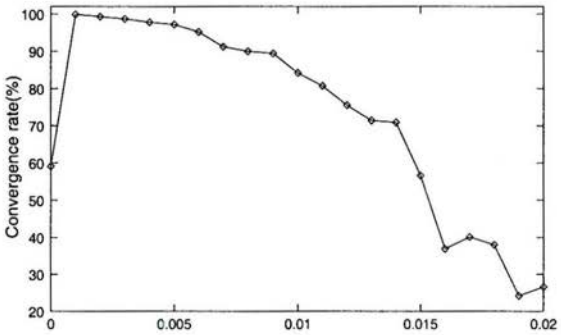
In our architecture, the motion equation for the N-queens problem, (11), is slightly modified as

$$U_{ij} = rU_{ij} + T_{ij}V_{ij} - \Big(\bigcup_{\substack{k=1 \\ k \neq i}}^{N} V_{kj} \cup \bigcup_{1 \leq i+k, j+k \leq N, k \neq 0} V_{i+kj+k} \cup \bigcup_{1 \leq i+k, j-k \leq N, k \neq 0} V_{i+kj-k} \Big). \quad (13)$$

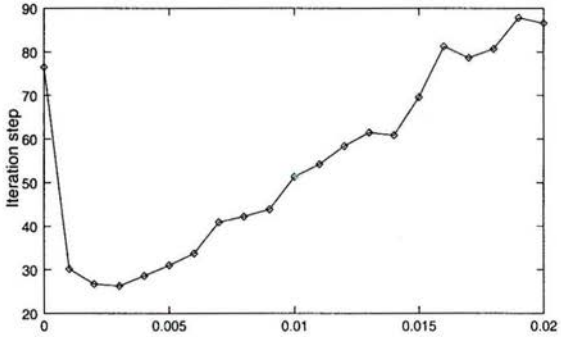
In this motion equation, the sum of neuron inputs is replaced by the logical sum of them. It is proven mathematically that Eq. (13) has the same equilibrium points as Eq. (11). This modification can reduce the hardware size necessary to implement our MNN.

Fig. 12 shows the performance of our MNN with this modified motion equa-

$r = 0.125$ among 1,000 runs. The maximum number of the iteration steps is set to 100. The convergence rate is 99.9 % and the average number of iteration steps for convergence is 30.2 at $\delta T = 0.001$. Although the number of iteration steps increases about 3 times from Eq. (11), it still keeps the high convergence rate.



(a) Convergence rate



(b) Average number of iteration steps

Figure 12. Change of performance with various δT.

4.3.2. Architecture

Now, we present our implementation of the maximum neural network with the reinforced self-feedback by the logical synaptic connection architecture. Fig. 13 shows the architecture for the N-queens problem. This architecture consists of the neuron array, the processing units, and RAM. The neuron array consists of $N \times N$ processing elements that realize the neurons arranged on a chessboard. The detailed structure of each processing element is illustrated in Fig. 14. The D flip-flop (DFF) stores 0−1 neuron output V_{ij}, and three lookup tables (LUT1,2,3) implementing OR-gates calculate the logical sum of outputs from the other neu-

state of MNN is updated in the N-parallel mode. So, there are N units for this state update in the system. At each iteration step, each processing unit loads the neuron input U_{ij} and the self-feedback gain T_{ij} from RAM. Then, it updates U_{ij}, V_{ij} and T_{ij}, using V_{ij} and the logical sum generated from the neuron array. Lastly, it stores V_{ij} in the neuron array, and inputs U_{ij} and T_{ij} into RAM.

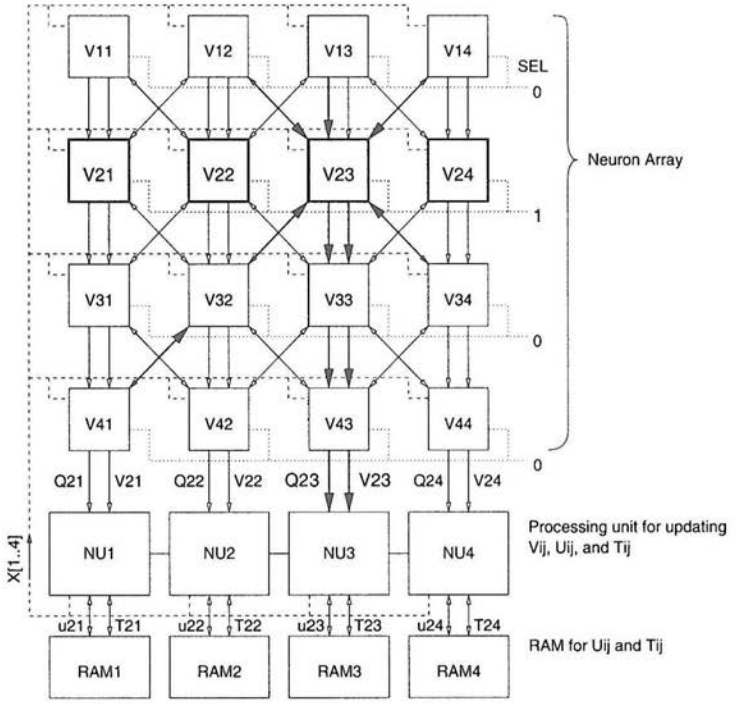
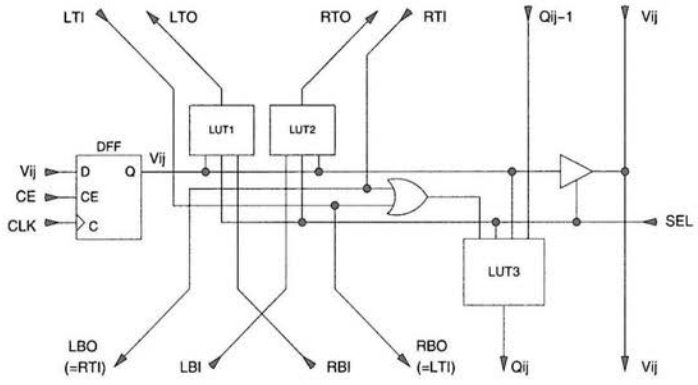


Figure 13. Architecture of MNN with reinforced self-feedback.



4.3.3. Implementation

We have designed the single element of this neuron array on an FPGA (Xilinx XCV200 with 230k system gates). This device contains 2,352 slices, and one slice consists of two configurable logic blocks (CLBs). Each CLB contains two function generators based on lookup tables, which can implement any function of four input variables. The single element of the neuron array in Fig. 13 occupies three slices in our design, thus the device can contain 784 elements. The processing unit in Fig. 13 occupies 69 slices in our design, and the device can contain 34 units. Hirai (1998) proposed a 1,000-neuron VLSI system with one million 7-bit physical interconnections based on the pulse-density-modulating (PDM). It consists of 1,120 chips, where each one is fabricated using CMOS gate array with 250k gates. This system can be applied to solve the 31-queens problem. Although our architecture is only applicable to limited problems, the architecture for the 31-queens problem can be built with *only three* chips.

The neuron array in our architecture has a simple structure, and several chips for the array can be connected in cascade to build a system for a large scale problem. Besides, if the neuron array consists of reconfigurable FPGAs, we can rewrite all connections between neurons in the neuron array to solve other combinatorial optimization problems.

5. Conclusions

The paper presented the maximum neural network with the reinforced self-feedback as a new binary neural network approach to the N-queens problem, and three VLSI implementations for this typical combinatorial optimization problem, after a brief review of the background and the conventional approach. Our VLSI implementations focus on the efficient realization of the synaptic connection networks between neurons, because they usually occupy a large area on a VLSI chip. Besides, we adopt reconfigurable devices such as CPLDs and FPGAs to realize the scalable hardware with very high speed of computation. Although the actually implemented hardware systems contain less than a hundred binary neurons, we estimate that more than several thousands of binary neurons can be implemented with the current FPGA technology. In our experiments we have introduced several new ideas and methods to make use of this attractive approach. The binary neural networks have open-ended possibilities of solving hard problems in real worlds. Through active research in this field, we would like to contribute to the advancement of this innovative technology in the future.

References

ABRAMSON, B. and YUNG, M. (1989) Divide and conquer under global constraints: a solution to the N-queens problem. *J. Para. Dist. Comp.*, **6**,

BITNER, J. R. and REINGOLD, E. M. (1975) Backtrack programming techniques. *Commun. ACM*, **18**, 651-656.

FALKOWSKI, B.-J. and SCHMITZ L. (1986) note on the queen's problem. *Inf. Proc. Lett.*, **23**, 39-46.

FORTES, J. A. B (1987) Systolic arrays: A survey of seven projects, *IEEE Computer*, **20** 91-103.

HINTON, G. E., SEJNOWSKI, T. J. and ACKLEY, D. H. (1984) Boltzmann machines: constrained satisfaction networks that learn. *CMU-CS-84-119*, Carnegie Mellon University.

HIRAI, Y. (1998) *A 1,000-Neuron System with One Million 7-bit Physical Interconnections, in M.J.Jordan, M.J.Kearns and S.A.Sola eds., Advances in Neural Information Processing Systems*, **10**, *The MIT Press, Cambridge, Massachusetts*, 705–711.

http://www.viplab.is.tsukuba.ac.jp/PDM/

HOPFIELD, J. J. and TANK, D. W. (1985) Neural computation of decisions in optimization problems. *Bio. Cybern.*, **52**, 141-152.

IEEE (1988) *IEEE standard backplane bus specification for microprocessor architecture; Futurebus*, ANSI/IEEE Std.896.1-1987.

JAGOTA, A. (1995) Approximating maximum clique with a Hopfield network. *IEEE Trans. Neural Networks*, **6**, 724-735.

KALE, L. V. (1990) An almost perfect heuristic for the N nonattacking queens problem. *Inf. Proc. Lett.*, **34**, 173-178.

KIRKPATRICK, S., GELATT JR., C. D. and VECCHI, M. P. (1983) Optimization by simulated annealing, *Science*, **220**, 4598.

KUROKAWA, T. and TAKEFUJI, Y. (1992) Neural network parallel computing for BIBD problems, *IEEE Trans., CAS.*, **39**, 243-247.

LEE, K. C., FUNABIKI, N. and TAKEFUJI, Y. (1992) A parallel improvement algorithm for the bipartite subgraph problem, *IEEE Trans. Neural Networks*, **3**, 139–145.

MCCULLOCH, W. S. and PITTS, W. H. (1943) A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, **5**, 115.

NOZAWA, H. (1992) A neural network model as a globally coupled map and applications based on chaos. *Chaos*, **2**, 377-386.

OHTA, M. (1999) On the Self-Feedback Controlled Chaotic Neural Network and its Application to N-Queen Problem, *Proc. Int. Joint Conf. Neural Networks.*

OHTA, M. (2002) Chaotic neural networks with reinforced self-feedbacks and its application to N-Queen problem, *Mathematics and Computers in Simulation*, **59**, 305-317.

OHTA, M., ANZAI, Y., YONEDA, S. and OGIHARA, A. (1993) A Theoretical Analysis of Neural Networks with Nonzero Diagonal Elements, *IEICE Trans. Fundamentals*, **E76–A**, 284–291.

OHTA, M., OGIHARA, A. and FUKUNAGA, K. (1994) Binary Neural Network

IEICE Trans. Inf. Syst., **E77-D**, 459–465.

OHTA, M., OGIHARA, A., TAKAMATSU, S. and FUKUNAGA, K. (1995) A Study on the Mechanism of the Minimum Searching by the Chaotic Neural Network, *Proc. IEEE Int. Conf. Neural Networks*, 1517–1520.

PAIELLI, R. A. (1988) Simulation tests of the optimization method of Hopfield and Tank using neural networks. *NASA Tech. Memo. 101047.*

REICHLING, M. (1987) A simplified solution of the N queen's problem. *Inf. Proc. Lett.*, **25**, 253-255.

RUSSELL, S. and NORVIG, P. (1995) *Artificial Intelligence A Modern Approach*, Prentice Hall.

SHRIVER, B. D. ed. (1988) *Artificial Neural Systems*, IEEE Computer, **21**, 8–117.

STONE, H. S. and STONE, J. M. (1987) Efficient search techniques: an empirical study of the N-queens problem. *IBM J. Res. Dev.*, **31**, 464-474.

SUNDARARAJAN, N. and SARATCHANDRAN, P. (1998) Parallel Architectures for Artificial Neural Networks: paradigms and implementations, *IEEE Computer Society Press, Los Alamitos, California*, 379.

TAKEFUJI, Y. (1992) *Neural network parallel computing*, Kluwer Academic Publishers.

TAKEFUJI, Y. and SZU, H. (1989) Design of parallel distributed Cauchy machines. *Proc. Int. Joint Conf. Neural Networks.*

WANG, L. (1997) Discrete-time convergence theory and updating rules for neural networks with energy functions. *IEEE Trans. Neural Networks*, **8**, 445–447.

WANG, R.-L., TANG, Z. and CAO, Q.-P., A. (2002) A near-optimum parallel algorithm for bipartite subgraph problem using the Hopfield neural network learning. *IEICE Trans. Fundamentals*, **E85-A**, 497-504.

WIESELHEIER, J. E., BARNHART, C. M. and EPHREMIDES, A. (1994) A neural network approach to routing without interference in multihop radio networks. *IEEE Trans. Commun.*, **42**, 166-177.

WILSON, G. V. and PAWLEY, G. S. (1988) On stability of the travelling salesman problem algorithm. *Biological Cybernetics*, **58**, 63-70.

YOSHIO, H., BABA, T., FUNABIKI, N. and NISHIKAWA, S. (1997) Proposal of an N-parallel computation method for a neural network for the N queens problem. *Elect. Commun. Japan, Part 3*, **80**, 12-20.