

A Hopfield neural network based task mapping method

W. Zhu^{a,*}, T.-Y. Liang^b, C.-K. Shieh^b

^a*Department of Computer Science and Electrical Engineering, The University of Queensland, Queensland 4072, Australia*

^b*Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan*

Abstract

With a prior knowledge of a program, static mapping aims to identify an optimal clustering strategy that can produce the best performance. In this paper we present a static method that uses Hopfield neural network to cluster the tasks of a parallel program for a given system. This method takes into account both load balancing and communication minimization. The method has been tested on a distributed shared memory system against other three clustering methods. Four programs, SOR, N-body, Gaussian Elimination and VQ, are used in the test. The result shows that our method is superior to the other three. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Task mapping; Hopfield network; Optimization; Distributed shared memory

1. Introduction

Task assignment plays an important role in parallel processing, especially when distributed systems are increasingly used as an alternative to multiprocessors for parallel processing. An optimal assignment provides much better performance than an ad hoc one. In the past, mathematical programming and graph theory, based on various heuristics, have been successfully used in search of optimal/near-optimal assignment [1–5]. In this paper, we propose a technique that combines the robustness of annealing with an efficient local optimization method in the search of optimal/near-optimal assignments. The proposed method is a variation of mathematical programming. It, as mathematical programming, has an objective function and some constraints that determine optima. In contrast to other methods in search of optima, the proposed method uses a neural network to locate optima. It transfers the objective function of a task assignment problem into the energy function of a Hopfield neural network (HNN) it then depends on the network's natural tendency to evolve towards a potential energy minimum. To avoid local optima, the search algorithm adopts an annealing technique as suggested in [6] in its search. The annealing technique helps in locating good regions of the search space, while the HNN can rapidly hit the optimum. The advantage of this approach lies on its simplicity, efficiency and reusability.

The assignment problem has been described in various ways in the literature [7]. Generally, the assignment

problem can be viewed as assigning a given program with a number of tasks onto a number of processors/machines allocated to this program, so that some performance criteria, such as the completion time of the program, can be optimized. Using graph theory, a program can be represented in the form of a graph (directed or undirected). In this paper, we only consider undirected task graphs, i.e. no precedence constraint exists between tasks. A node in such a graph represents a task; an edge between two nodes represents the amount of communication between these two tasks. Similarly, machines and the network that connects the allocated machines can also be represented in a graph, in which a node represents a machine and an edge represents the communication cost between the two connected machines. With these two graphs, the assignment problem can be viewed as a graph matching problem, which has been known to be NP-complete in its general form [8]. In mathematical programming, some performance criteria are presented in the form of a set of objective functions with some restrictions. Depending on the number of those objective functions, the assignment problem can be transferred to a single- or multi-objective decision making problem. Then, an algorithm, such as branch-and-bound, is applied to identify a feasible and optimal solution. The goal of task assignment is, given a set of related tasks and a group of connected machines, to find the task assignment so as to obtain the optimal performance defined in the objective function.

The proposed method uses Competitive Hopfield Neural Network (CHNN) as a tool to eliminate task duplication that is usually not allowed in most applications. The network is implemented as a two-dimensional array with the rows

* Corresponding author.

representing machines allocated to a parallel program and the columns representing the tasks of the program. A winner-take-all learning rule is imposed on network evolution, which ensures that only one neuron in a column is in firing state, all other in resting. The winner-take-all rule handily guarantees a feasible result that eliminates task duplication and accelerates the searching process. When the number of tasks within a program is more than the number of machines allocated to the program, the proposed method identifies an optimal or near-optimal clustering strategy.

Four programs, SOR, N-Body, Gaussian Elimination and VQ, were used in the experimental study which was carried out on a distributed shared memory system. The first three programs are iterative-based computation programs; the last one is an image compression algorithm and involves intensive memory searches. To compare with other methods, three widely used assignment methods, interleaving, batch, and random, are also implemented. The four methods are evaluated under the same condition. The results show that the proposed method in all cases either outweighs or equals to the performance achieved by the best of the other three methods. For a system that is comprised by identical computers, the proposed method found the optimal assignments. When a system consists of computers with different speeds, the method can find near-optimal assignments.

The rest of the paper is organized as follows. Section 2 formulates the task assignment problem, which includes modeling and objectives. Section 3 focuses on task mapping method that is developed based on the model established in Section 2. In Section 4, SOR, N-body, Gaussian Elimination and VQ are used as benchmarks to evaluate the performance of the proposed method against the other three methods. Finally, Section 5 is devoted to concluding remarks.

2. Problem formulation

In the following analysis, we assume the cost of communication between tasks executing on the same machines, intra-machine communication, is negligible because the cost of inter-machine communication is much higher than that of intra-machine communication in a distributed system. We further assume that inter-machine messages take approximately the same amount of time to deliver from one machine to another. Therefore, to find an optimal assignment, one must consider how to find a tradeoff between load balancing and minimization of communications. A compromise between these two can ensure an assignment that has the best performance in terms of completion time. We also assume there is no precedence relation between tasks, all tasks can start at the same time.

2.1. Modeling

Under the above assumptions, task assignment can be

formulated into an optimization problem with five elements

$$TM = \{T, M, N, O, AL\},$$

where

- T represents a set of n related tasks, $T = \{T_i | i = 1, \dots, n\}$. Let $exec_i$ denote the execution time of tasks i . If computers with different speeds exist in M , $exec_i$ refers to the execution time of task i on the slowest computer of M .
- $M = \{m_i | i = 1, \dots, m\}$ is a set of m computers allocated to T . All the computers in M must have the same architecture, but can be different in speed, that makes task migration possible. Let $speed_i$ ($i \in \{1, \dots, m\}$) indicate the relative speed of computer i to the slowest computer in M . The slowest computer has its $speed_i$ set to 1. A subset of T allocated to computer x is expressed as T_x .
- $N = \{(c_{xij}) | x, y \in M \text{ and } i, j \in T\}$ represents the communication cost between task i executing on machine x and task j executing on machine y .
- O is a set of objectives for task assignment, such as, minimizing the completion time, minimizing load difference, and minimizing the total execution time.
- AL denotes a set of algorithms that can be used in the search task assignments.

Given T , M and N , task assignment can be viewed as an optimization process which selects an algorithm from AL , the algorithm based on the defined objective, O , searches the possible solution space for an either optimal or near optimal assignments.

2.2. Objective function

In contrast to use a min–max form objective in the search of an optimal assignment, in this study we use a different objective that considers load balancing and communication minimization together. Instead of having two separate objectives, one for load balancing, the other for communication minimization, we integrate these two into one equation and minimize the sum of these two factors. Let $ld(A)$ denote the sum of differences in completion time and the total communication cost created by an assignment A . Let $|T_i|$ denote the number of replicas of task T_i . Then, the proposed objective takes the following form:

$$ld(A_{op}) = \min (LD_{bal} + bC_{total}) \quad (1)$$

$$s.t. \quad \forall |T_i| = 1, (i \in (1, \dots, n)),$$

where A_{op} represents an optimal assignment, LD_{bal} sums the time difference between machines in completing the assigned workload, and C_{total} sums the cost of inter-machine communication. Condition $\forall |T_i| = 1, (i \in (1, \dots, n))$ eliminates the possibility of task duplication.

For a particular machine x , $t_x^e(A)$ denotes the total time spent on executing the tasks assigned to it, which is defined as $t_x^e = (\sum_{i \in T_x} exec_i) / speed_x$. Similarly, $t_x^c(A)$ denotes the total number of inter-machine communications caused by

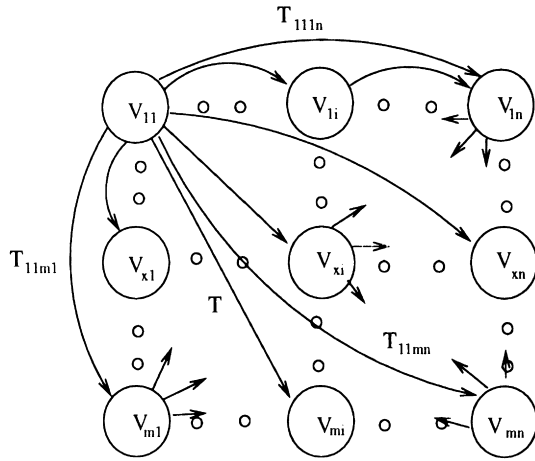


Fig. 1. An mn Hopfield neural network.

executing these tasks on x . The values of t_x^e and t_x^c depends on how tasks are assigned to machines. In this paper, we use the differences in completion time between a machine and the average one to represent load difference contributed by the machine. LD_{bal} then takes the following form:

$$LD_{bal} = \sum_{x=1}^m (t_x^e(A) - t_{avg}^e(A))^2, \quad (2)$$

where $t_{avg}^e(A)$ denotes the average completion time which is defined as

$$t_{avg}^e(A) = \frac{\sum_{i=1}^m t_i^e(A)}{m}. \quad (3)$$

Substituting $\sum_{x=1}^m t_{avg}^e(A)$ of (2) by (3), LD_{bal} becomes

$$LD_{bal} = \sum_{x=1}^m (t_x^e(A))^2 - \frac{\left(\sum_{x=1}^m t_x^e(A)\right)^2}{m}. \quad (4)$$

If all machines are identical, the second term of (4) is a constant and does not have any impact on the outcome of an optimization process. Let LD_{bal}^{iden} denote the load difference in a system with identical machines. Discarding the second term from (4), we obtain

$$LD_{bal}^{iden} = \sum_{x=1}^m (t_x^e)^2 = \sum_{x=1}^m \left(\sum_{i \in T_x} exec_i \right)^2. \quad (5)$$

Mathematically, for a system with identical machines and if $\sum_{x=1}^m t_x^e = c$, c is a constant, then the minimal value for LD_{bal}^{iden} is to let $t_x^e = (c/m)(x \in \{1, \dots, m\})$, i.e. workload should be evenly divided among machines, thus, all machines complete their work at the same time. Note that t_x^e is the normalized workload that makes the workloads comparable between machines with different speeds. In

this situation,

$$LD_{bal} = \sum_{x=1}^m \left(\frac{\sum_{i \in T_x} exec_i}{speed_x} \right)^2 - \frac{1}{m} \left(\sum_{x=1}^m \frac{\sum_{i \in T_x} exec_i}{speed_x} \right)^2. \quad (6)$$

When there are speed differences between machines minimizing the LD_{bal} does not necessarily minimize the completion time. Thus, apart from minimizing load difference, some extra measures must be taken to ensure that the assignment identified based on (6) corresponds to the assignment with the minimal completion time. We impose such a condition on an valid assignment that has the machines complete their tasks in the order of the machine speeds, i.e. a slower machine should complete its work before a faster one.

In (1), C_{total} represents the time spent on passing messages among machines. b is a weight parameter and is used to guide the mapping process to find an assignment that can either favor to load balancing or to communication reduction. For instance, if $b \ll 1$, the mapping process pays more attention on load balancing since communication has little impact on the objective function. On the contrary, if $b \gg 1$, the mapping process bias its weight on alleviating communication impact.

3. Task mapping

The HNN, due to its simple architecture and well-defined time behaviors, has been widely used in various applications. State evolution of this network is based on the decrease in an energy function. Hence, it is inherently suitable for solving combinatorial optimization problems, such as job-shop scheduling or travel salesman problem [9]. To map a parallel program of n tasks to a system of m machines, a two-dimensional binary Hopfield network is created, it contains mn fully interconnected neurons, as depicted in Fig. 1. In this figure, a row corresponds to a machine and a column to a task. A possible assignment corresponds to a state of the neural net. An optimal assignment corresponds to a state that has the minimal energy, if we can translate the objective function defined in the previous section into the energy function of a HNN.

A neuron in the network can be either in firing state (1) or in resting state (0), depending on the input received from other neurons. Let V_{xi} denote the state of the (x,i) th neuron and let T_{xij} be the strength of the connection from neuron (x,i) to neuron (y,j) . During the network evolution, a neuron (x,i) receives the weighted input of $T_{yxi}V_{yj}$ from neuron (y,j) . The total input to the neuron (x,i) , U_{xi} , is the sum of inputs from all the other neurons in the network and a bias input I_{xi} from outside. Mathematically,

$$U_{xi} = \sum_{y=1}^m \sum_{j=1}^n T_{yxi} V_{yj} + I_{xi}. \quad (7)$$

During the network evolution, the state of neuron (x,i) , V_{xi} , is determined by

$$V_{xi}^{\text{new}} = \begin{cases} 1, & U_{xi} > 0 \\ V_{xi}, & U_{xi} = 0. \\ 0, & U_{xi} < 0 \end{cases}$$

Following to this evolution equation, if $I_{xi} = 0, (1 \leq x \leq m, 1 \leq i \leq n)$, the network achieves a stable state when the network energy function of the two-dimensional Hopfield network, given by

$$E = \sum_{x=1}^m \sum_{y=1}^m \sum_{i=1}^n \sum_{j=1}^n T_{xiyj} V_{xi} V_{yj} \quad (8)$$

is converged.

Before applying this technique on task assignment, the objective function defined in the last section must be translated to the form of the network energy function of HNNs as shown in (8). Then, the optimum principle of the HNN is applied to locate a minimal energy point in the network, which corresponds to an optimal or a near-optimal assignment. In the view of task assignment, the state of neuron (x,i) , V_{xi} , indicates whether task T_i is assigned to machine x . If so, V_{xi} is set to 1; otherwise, it is set to 0. The workload on machine x can be expressed as $t_x^c = \sum_{i=1}^n \text{exec}_i V_{xi} / \text{speed}_x$. Then, for the load balancing based objective, LD_{bal} can be expressed by using neuron states

$$\begin{aligned} LD_{\text{bal}} &= \sum_{x=1}^m \left(\frac{\sum_{i=1}^n \text{exec}_i V_{xi}}{\text{speed}_x} \right)^2 - \frac{1}{m} \left(\sum_{x=1}^m \sum_{i=1}^n \frac{\text{exec}_i V_{xi}}{\text{speed}_x} \right)^2 \\ &= \sum_{x=1}^m \sum_{i=1}^n \sum_{j=1}^n \frac{\text{exec}_i \text{exec}_j V_{xi} V_{xj}}{\text{speed}_x \text{speed}_x} \\ &\quad - \frac{1}{m} \sum_{x=1}^m \sum_{y=1}^m \sum_{i=1}^n \sum_{j=1}^n \frac{\text{exec}_i \text{exec}_j V_{xi} V_{yj}}{\text{speed}_x \text{speed}_y}. \end{aligned}$$

If we introduce an δ function defined as

$$\delta_{xy} = \begin{cases} 1, & \text{if } x = y \\ 0, & \text{if } x \neq y \end{cases}$$

LD_{bal} can be simplified to

$$LD_{\text{bal}} = \sum_{x=1}^m \sum_{y=1}^m \sum_{i=1}^n \sum_{j=1}^n \frac{\text{exec}_i \text{exec}_j}{\text{speed}_x \text{speed}_y} \left(\delta_{xy} - \frac{1}{m} \right) V_{xi} V_{yj}.$$

When all machines are identical, the formula is changed to

$$LD_{\text{bal}}^{\text{iden}} = \sum_{x=1}^m \sum_{y=1}^m \sum_{i=1}^n \sum_{j=1}^n \text{exec}_i \text{exec}_j V_{xi} V_{yj} \delta_{xy}.$$

The overall communication cost of a program must also be transferred to the form of the energy function of a HNN,

it is

$$C_{\text{total}} = \sum_{x=1}^m \sum_{y=1, y \neq x}^n C_{xy} = \sum_{x=1}^m \sum_{y=1}^m \sum_{i=1}^n \sum_{j=1}^n C_{xiyj} V_{xi} V_{yj} (1 - \delta_{xy}). \quad (9)$$

Adding the LD_{bal} and C_{total} together, the objective function becomes

$$\begin{aligned} ld(A) = \min &\left(\sum_{x=1}^m \sum_{y=1}^m \sum_{i=1}^n \sum_{j=1}^n \left[\frac{\text{exec}_i \text{exec}_j}{\text{speed}_x \text{speed}_y} \left(\delta_{xy} - \frac{1}{m} \right) \right. \right. \\ &\left. \left. + b C_{xiyj} (1 - \delta_{xy}) \right] V_{xi} V_{yj} \right) \end{aligned}$$

which has the form of the energy function of a HNN. Let

$$T'_{xiyj} = \frac{\text{exec}_i \text{exec}_j}{\text{speed}_x \text{speed}_y} \left(\delta_{xy} - \frac{1}{m} \right) + b C_{xiyj} (1 - \delta_{xy})$$

and

$$T_{xiyj} = T_{yjxi} = T'_{xiyj} + T'_{yjxi}.$$

A symmetric HNN is created, in which T_{xiyj} is the strength of the synaptic link that connects V_{xi} and V_{yj} the total inputs to the (x,i) th neuron is,

$$U_{xi} = \sum_{y=1}^m \sum_{j=1}^n T_{xiyj} V_{yj}.$$

We proved that using (T_{xiyj}) as the energy function, the network energy decreases monotonously during its evolution¹. Thus, an optimal or near optimal task assignment can be identified.

To eliminate task duplication, which could introduce inconsistency in execution, CHNN is used in the network evolution. It depends on a winner-take-all rule to impose such a condition that only one neuron in a column is in firing state. Using this rule, the neurons in a column compete with each other to be the winner and only the winner, which is the neuron that receives the minimal total inputs is set to 1. In other words, the input–output function for the neurons in the i th column is

$$V_{xi} = \begin{cases} 1, & \text{if } U_{xi} = \min\{U_{1i}, U_{2i}, \dots, U_{mi}\} \\ 0, & \text{otherwise} \end{cases}.$$

If two neurons in the same column have the same minimal input, we select that the one with $\min(\max(\sum_{i \in T_x} \text{exec}_i) / \text{speed}_x)$ as the winner since it has a smaller completion time.

3.1. Searching algorithm

An asynchronous update scheme is used in the network evolution. Fig. 2 outlines how a local optimum is identified. After an initial assignment is given, the mapping algorithm

¹ See Appendix for proof.

```

Repeat
  continue_flag = FALSE
  For i=0 to n-1 do // P denotes the machine for task i
    For x=0 to m-1 do // compute the state of ith column
      Calculate  $U_{xi}$ 
       $P_{new} = P$ 
      For x=0 to m-1 do // applying the winner-take-all rule
        If  $U_{xi} < U_{p_{newi}}$ , then
           $P_{newi} = x$ 
          If  $P_{new} \neq P$  then
            { reassign task i to machine  $P_{new}$ 
              continue_flag = TRUE }
      Until continue_flag = FALSE

```

Fig. 2. The local optimization algorithm.

starts to migrate a task from its initial location to another and calculates the impact of this migration on the objective function. The same process is also applied to the other tasks. If the value of the objective function is reduced, the migration is accepted. The mapping algorithm continues this process until the objective function is converged. Heuristics that can accelerate the convergence are also used in the search of an optimal solution, such as, balancing the workload between heavily loaded machines and lightly loaded ones, and clustering heavily communicated tasks into a group.

As the simulated annealing, a probability method is applied to avoid local minima. It starts from an initial mapping. We then run the local optimization algorithm shown in Fig. 2 to identify the nearest local optimum. The value of the energy function at the local optimum is used as the value for a new initial point, and an acceptance or rejection decision is made according to metropolis criterion. Experiments show this strategy has a high probability to hit optimum solutions in the sense of statistics. The time spent on searching varies with applications. Even with the same application, the time also varies, depending on the number of machines allocated to the application. The more the machines are allocated, the longer the time spent on searching because of the increase of searching space. For the four applications used in the following performance evaluation, the searching times, running on an Intel 80486 PC, are shown in Table 1.

Table 1
Searching time of the proposed method

Applications	2-node	4-node	8-node
SOR	1.58s	1.65s	20.48s
N-body	0.05s	3.42s	3.46s
Gaussian	0.11s	0.82s	16.59s
VQ	2.09 s	15.49 s	37.02 s

In addition to optimization, another advantage of using HNN relies on its capability to store certain memories or patterns in a manner rather similar to our brain. Using this feature, we are able to store all assignments found so far into files. These files can be used later as initial assignment for similar applications and platforms. In that situation, a file can be pre-loaded to our system. When the searching starts, the HNN immediately recalls its *memory* and a good assignment, perhaps an optimal one, can be identified very quickly.

4. Performance analysis

All experiments were carried out on Cohesion [10], a distributed shared memory (DSM) system, since DSMs is more preferable to message-passing in parallel processing [11]. The system is built on a network of personal computers connected by a dedicated 10 Mbps Ethernet. Each computer contains an INTEL 80×86 microprocessor, 8 Mbyte memory and a 3Com Ethernet interface. Three indices: (1) the execution time; (2) the number of messages exchanged between machines; and (3) the amount of data transferred, measured in KB, are used in the evaluation.

According to the model established in Section 2, we need not only know T and M , but also know N . Nevertheless a programmer is hardly able to supply either the execution time of a task or communication cost between two tasks in most cases. For a DSM system, an application programmer normally has no idea about the number of inter-machine communications. In addition, the programmer unlikely knows the consistent protocol used in the DSM system. Even the programmer knows the consistent protocol, it is almost impossible for him/her to estimate the communication cost in a reasonable precision. To assist the collection of the execution time and the amount of communication, an automatic collector was implemented on Cohesion. The tasks of an application should be first executed on the collector. Whenever a task is executed on the collector, the collector records the execution time of the task it also counts all page faults. Based on the information, it can work out the amount of communication between tasks. With this information for an application, we can execute mapping algorithms for assignments. Finally, according to these assignments, the tasks are allocated to the machines for execution. Data related to the three performance indices are collected in execution, which are used to determine the qualities of these mapping algorithms.

4.1. Experimental results

Four application programs are used in the performance evaluation. Three of the four programs are scientific computing programs, namely Successive Over Relaxation (SOR), N-Body and Gaussian Elimination, and the other one is an image compression algorithm, Vector

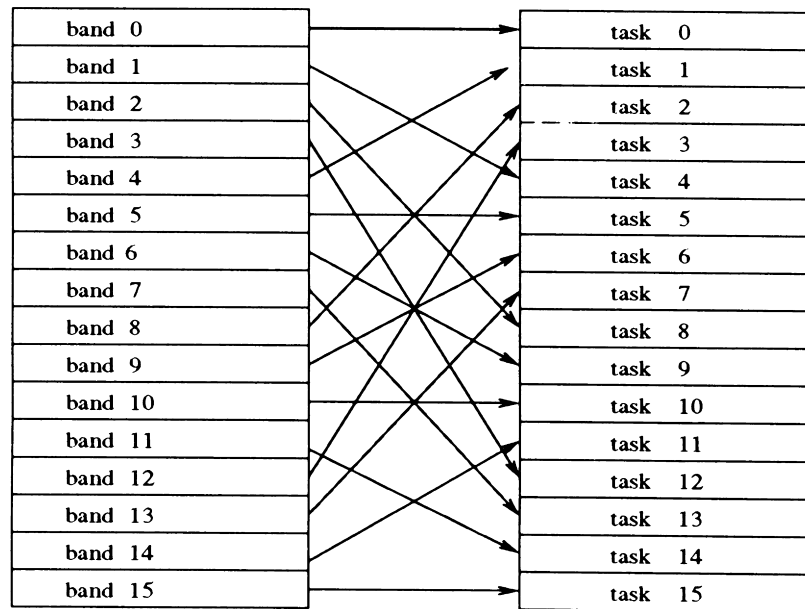


Fig. 3. The decomposition for SOR.

Quantization (VQ). The evaluation was conducted in two phases. In the first phase, homogeneous systems comprised by identical processors, 80486-33s, were used to carry out experiments. In the second phase, processors with different speeds, 80486-33 and pentium-90, were used in experiments. In the second phase only SOR and N-Body programs were used in the evaluation.

Instead of having on task for a machine in our experiments, an application is divided into a number of small tasks that are four times of the number of machines. The advantages of having multiple tasks on a machine concurrently are: (1) overlap computation with communication; (2) make dynamic task relocation possible. The first advantage can reduce the delay caused by communication, i.e. while a task on a machine is communicating, instead of waiting for the completion of the communication another task on the same machine start to execute. On the contrary, if we want to support dynamic load balancing based on task migration, the number of tasks must be more than the number of machines.

To evaluate the effectiveness of the proposed method, another three widely used methods are employed, namely interleaving, batch, and random [10,12]. The interleaving method distributes tasks among machines in a round-robin fashion, i.e. if there are m machines, tasks $i, m + i, 2m + i, \dots, (i \in (0, \dots, m - 1))$ form a group and assigned to machine i . The batch method, on the contrary, divides tasks into a number of groups, depending on the number of machines. Each group is composed of the same number of tasks that are next to each other. The random method arbitrarily divides tasks into a number of groups, each group has the same amount of workload and is assigned to a machine. When there are machines with different speeds in a system, the batch and random methods cluster tasks into

groups on the basis of the relative speeds of machines². If the number of tasks is divisible by the number of machines and each task has the same workload, load balancing can be easily achieved. In this situation, the quality of a mapping method depends on its ability to identify an assignment that has minimized inter-machine communication.

SOR is a method that uses iteration to approximate the solution of a partial differential equation. Basically, a grid of points in a pending area of problem is represented by a matrix. In a parallel version, the matrix is divided into roughly equal size bands of rows each band is assigned to a task for processing. Since each entry of the matrix is updated by its neighboring entries for next iteration, communication occurs at the boundary rows between bands.

We ran SOR on a 1024×1024 of floating point number matrix for 20 iterations. Three experiments were carried out on the 2-, 4-, and 8-machine systems, respectively. The rows of the matrix are evenly divided into bands, one for a task. These bands are assigned to tasks randomly. An example is shown in Fig. 3 which assigns the bands interleavely to the tasks.

Since each machine receives the same amount of workload, the performance difference between assignments depends on the amount of inter-machine communication produced by the assignments. For SOR with the above data assignment, inter-machine communication can only occur at the boundary of neighboring data bands. To identify the optimal assignment, a mapping method needs to find a way to put the tasks into groups, and ensure such a grouping strategy has the minimized communication between groups. For example, for the 4-machine case, there are 16 tasks

² We have only compared our method against these two in the second phase.

Table 2
Performance comparison for SOR

SOR	2-node	4-node	8-node
<i>Execution time (s)</i>			
Random	105.396	65.482	49.501
Batch	109.406	66.476	50.224
Interleaving	102.215	55.962	37.66
CHNN	102.215	55.962	37.66
Optimal	102.215	55.962	37.66
<i>Number of inter-machine messages</i>			
Random	412	1326	3697
Batch	495	1638	3864
Interleaving	219	701	2061
CHNN	219	701	2061
Optimal	219	701	2061
<i>The amount of communication (KB)</i>			
Random	973	3172	7341
Batch	1752	3669	7591
Interleaving	247	745	1751
CHNN	247	745	1751
Optimal	247	745	1751

assigned the same workload, as shown in Fig. 3. An optimal grouping strategy divides the tasks into four groups, $\{T_0, T_4, T_8, T_{12}\}$, $\{T_1, T_5, T_9, T_{13}\}$, $\{T_2, T_6, T_{10}, T_{14}\}$, and $\{T_3, T_7, T_{11}, T_{15}\}$, one for a machine. By using the interleaving allocation method, an assignment as above is obtained. On the contrary, the batch method groups the tasks in a different way, i.e. $\{T_0-T_3\}$, $\{T_4-T_7\}$, $\{T_8-T_{11}\}$, $\{T_{12}-T_{15}\}$. Such an assignment performs even worse than that produced by the random method since it has the biggest number of inter-machine messages, as shown in Table 2. The number of inter-machine messages is proportional to the amount of communication, as indicated in Table 2. Given an initial assignment as Fig. 3, the proposed method can easily find the optimal assignment, as the CHNN rows in Table 2 indicated.

The N-body problem simulates the evolution of a system with N-bodies, where the force exerted on each body arises due to its interaction with all the other bodies in the system. N-body algorithms have numerous applications in area of astrophysics, plasma physics, molecular dynamics, computer graphics, etc. The simulation proceeds over timesteps, each time it calculates the force on every body, updates its position and other attributes in a self-gravitating space system according to the Newtonian acceleration theorem. There are $N(N-1)/2$ forces among all pairs of bodies. To

Table 3
The task access patterns of N-body (access pattern of tasks)

N-body	T1	T2	T3	T4	T5	T6	T7	T8
1st iteration	1,9	2,10	3,11	4,12	5,13	6,14	7,15	8,16
2nd iteration	2,9	3,10	4,11	5,12	6,13	7,14	8,15	24,25
3rd iteration	3,9	4,10	5,11	6,12	7,13	8,14	24,26	23,25
4th iteration	4,9	5,10	6,11	7,12	8,13	24,27	23,26	22,25
5th iteration	5,9	6,10	7,11	8,12	24,28	23,27	22,28	21,25

Table 4
The task access patterns of N-body (groups produced by the batch method)

Groups	T1, T2, T3, T4	T5, T6, T7, T8
Accessed pages	1-12	5-8,13-16,21-28
Shared pages		5,6,7,8

calculate their impacts on each other, a sequential code as follows should be executed:

```
for (i = 0; i < N - 1; i + +)
  for (j = i + 1; j < N; j + +)
    gravity (body[i], body[j]);
```

In our experiments, 8192 bodies are simulated. The number of tasks used in the simulations is also four times of the number of machines allocated to the simulations. Each task is responsible for the calculation of part of the gravity between bodies. The intermediate results are temporarily stored into four arrays. The final result can be calculated by adding up these four arrays. We ran the simulations on 2-, 4- and 8-machines, respectively. The results are shown in Table 6. In the case of 2-machines, eight tasks are created. As depicted in Fig. 4, the calculation is vertically divided into eight columns, one for a task. For parallelism without contention, tasks running on different machines store their intermediate results into different

Table 5
The task access patterns of N-body (groups produced by the interleaving method)

Groups	T1, T3, T5, T7	T2, T4, T6, T8
Accessed pages	1-9,11,13,15,22-24,26,28	2-8,10,12,14,16,21-25,27
Shared pages		2-8,22,23,24

Table 6
Performance comparison for N-body

N-body	2-node	4-node	8-node
<i>Execution time (s)</i>			
Random	712.800	321.256	185.697
Batch	701.745	315.646	159.773
Interleaving	722.796	325.917	193.606
CHNN	701.745	315.646	159.773
Optimal	701.745	315.646	159.773
<i>Number of inter-machine messages</i>			
Random	178	1211	8840
Batch	142	719	4305
Interleaving	231	1579	12211
CHNN	142	719	4305
Optimal	142	719	4305
<i>The amount of communication (KB)</i>			
Random	351	2563	11295
Batch	216	879	3693
Interleaving	529	3581	1527
CHNN	216	879	3693
Optimal	216	879	3693

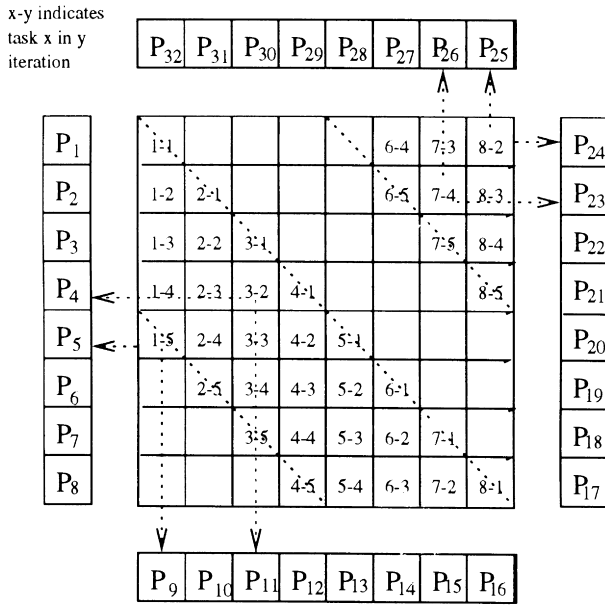


Fig. 4. The algorithm for N-body.

Table 7
Performance comparison for Gaussian elimination

Gauss	2-node	4-node	8-node
Random	136.985	102.875	99.137
Batch	154.799	96.271	72.189
Interleaving	136	90.563	76.480
CHNN	115.43	71.419	66.063
Optimal	114.727	70.742	65.869

regions of the arrays in a DSM system, each region of the array can be mapped to a distinct page to eliminate *false sharing*. The data access patterns for the eight tasks in different iterations are illustrated in Tables 3–5. In addition to organize related tasks into evenly loaded groups, an optimal assignment for this problem wants to ensure such a grouping strategy that produces the minimal number of page sharing between groups.

Table 3 shows the page access patterns for each

individual tasks in the 2-machine case. For instance, T_1 needs to access six different pages in the first five iterations, they are pages 1–5 and 9. Remember in DSM the number of pages shared between machines directly related to the communication cost. Using the batch method, T_1, T_2, T_3, T_4 will be in a group, and T_5, T_6, T_7, T_8 in the other group. Four pages are shared between these two groups, as shown in Table 4. On the contrary, if we use the interleaving method to group tasks there are 10 pages shared between groups, depicted in Table 5. Compared with the interleaving method, the batch method is better in this case since the groups produced by the batch method share a smaller number of pages than that produced by the interleaving method. Given an initial assignment, our method converges to the same assignment as that produced by the batch method in all the three cases of 2-, 4- and 8-machines. Table 6 shows that a near-linear speedup can be achieved as we increase the number of machines in the simulations.

A classic Gaussian elimination algorithm has been used to solve linear equations ($\mathbf{AX} = \mathbf{b}$). The algorithm marches the pivot down the main diagonal of the matrix. A *kij-form* elimination scheme was used in our experiments. $\mathbf{AX} = \mathbf{b}$ is partitioned into a number of blocks horizontally. Each block has the same number of equations and is allocated to a task. Each machine executes four tasks. The problem used in our experiment has its \mathbf{A} be a 512×512 floating-point matrix. Three experiments were conducted on 2-, 4- and 8-machine systems, the results shown in Table 7 indicates the CHNN can identify assignments that is very close to the optimal ones.

Vector Quantization is a technique used in image compression. As shown in Fig. 5, an original image is first decomposed into an N -dimension image vector. For example, a 2×2 block of pixel values can be ordered to form a 4-dimensional vector. Each image vector is then compared with a set of code-vectors, stored in a previously generated code-book. The code-vectors are also of N -dimension. The best match code-vector is chosen such that the distortion incurred in replacing the image vector with the code-vector is minimal. The distortion measure used here is the square of the Euclidean distance between the two vectors. After the

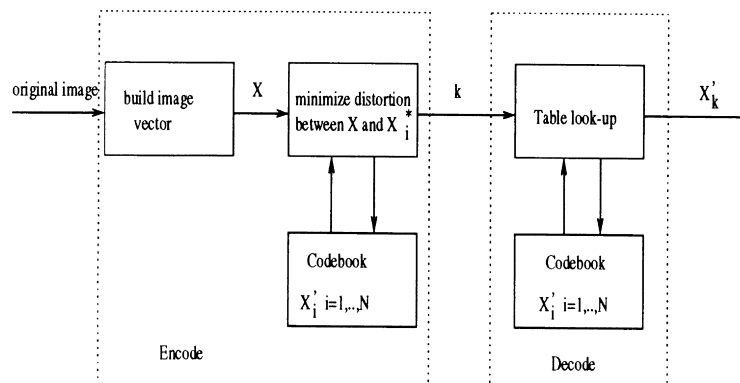


Fig. 5. A block diagram for VQ.

1-1	1-2	2-1	2-2	3-1	3-2	4-1	4-2
1-3	1-4	2-3	2-4	3-3	3-4	4-3	4-4
5-1	5-2	6-1	6-2	7-1	7-2	8-1	8-2
5-3	5-4	6-3	6-4	7-3	7-4	8-3	8-4
9-1	9-2	10-1	10-2	11-1	11-2	12-1	12-2
9-3	9-4	10-3	10-4	11-3	11-4	12-3	12-4
13-1	13-2	14-1	14-2	15-1	15-2	16-1	16-2
13-3	13-4	14-3	14-4	15-3	15-4	16-3	16-4

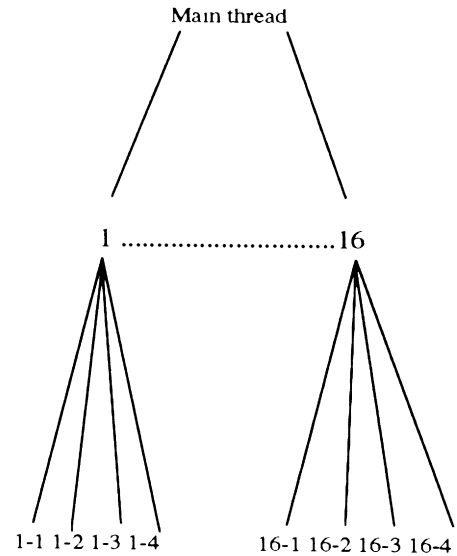


Fig. 6. The problem decomposition of VQ.

minimum distortion code-vector has been selected, its index k is encoded for the image vector in $\log_2 N$ bits, where N is the total number of code-vectors. At the decoder, this index is used as an entry to a duplicate code-book to produce code-vector. In our experiments, an image has 512×512 pixels and there are 256 code-vectors in the code-book. In addition, we use two matrices to restore the results from the encoder and the decoder. A *divide and conquer* approach is used to carry out the encoding and decoding tasks. As shown in Fig. 6, the code-book is divided into a number of areas, according to the number of tasks. An area is assigned to a task, the task searches the area for the best match. The outcome of a task then compares with those produced by

other tasks to find the global best. In practice, the main task of this program forks 16 child-tasks, these child-tasks are evenly distribute to machines. A child-task then creates another four tasks on its local machine, as illustrated in Fig. 6.

The experiment results of VQ is shown in Table 8. In the cases of 2- and 4-machines, the batch method produces the optimal assignment, and the same assignment is also obtained by using our mapping method. However, in the case of 8-machines, our mapping method can find the optimal solution but the batch method cannot. The results shown in Table 8 indicates the superiority of our method to the others in all three measurements.

Table 8
Performance comparison for VQ

VQ	2-node	4-node	8-node
<i>Execution time (s)</i>			
Random	145.987	85.516	53.331
Batch	134.893	68.875	43.554
Interleaving	150.436	114.407	67.446
CHNN	134.893	68.875	43.554
Optimal	134.893	68.875	43.554
<i>Number of inter-machine messages</i>			
Random	663	1693	2846
Batch	125	387	1777
Interleaving	771	2323	3341
CHNN	125	387	1777
Optimal	125	387	1777
<i>The amount of communication (KB)</i>			
Random	1605	3413	4659
Batch	4	20	42
Interleaving	1729	5002	5053
CHNN	4	20	42
Optimal	4	20	42

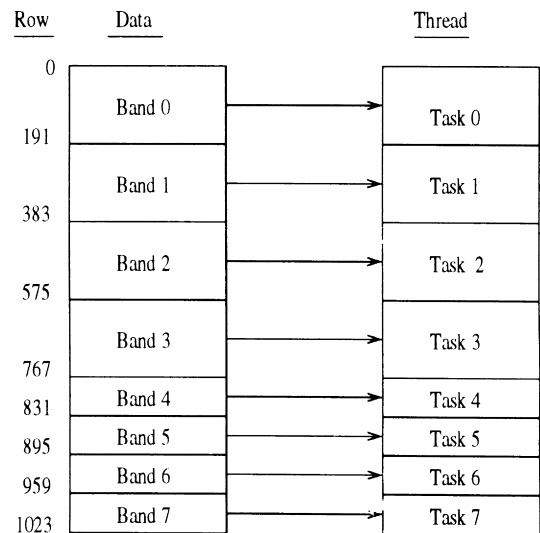


Fig. 7. Data decomposition for uneven SOR.

Table 9
Performance comparison for uneven SOR

	2-node	4-node	8-node
<i>Execution time (s)</i>			
Batch	141.609	78.325	49.501
Interleaving	109.162	64.787	46.177
CHNN	104.398	60.128	40.562
Optimal	104.398	58.516	40.201
<i>Number of inter-machine messages</i>			
Batch	219	700	2065
Interleaving	453	1636	3932
CHNN	296	859	2497
Optimal	296	1013	2670
<i>The amount of communication (KB)</i>			
Batch	249	748	1756
Interleaving	1699	3668	7595
CHNN	492	1235	3220
Optimal	492	1722	3706

To evaluate the capability of the proposed method on load balancing, we conducted another series of tests. This time, different workload is assigned to individual task. For instance, Fig. 7 shows such an assignment for SOR on a 2-machine system, where the first four tasks are assigned the same workload, while the other four tasks are assigned the same workload which is one-third of the workload assigned to the first four tasks.

The results of SOR using 2-, 4- and 8-machines are shown in Table 9. As before the batch method can identify a group strategy that has the minimum inter-machine communication, but with very different workloads assigned to machines. In contrast, workload balance can be achieved by using the interleaving method at the cost of inter-machine communication. Therefore, both the batch and the interleaving methods perform badly in the experiments.

Nevertheless, our method makes a compromise between load balancing and communication reduction. As a result, for the 2-machine case, our method divides the eight tasks into two groups, $\{T_0, T_4, T_5, T_6, T_7\}$ and $\{T_1, T_2, T_3\}$. This strategy eliminates the most inter-machine communication and make both machines have similar workload. In fact, the grouping identified by our method is the optimal one. In the cases of 4- and 8-machines, our mapping method achieved very good results though completion time are not optimal. Table 9 shows that from alleviating inter-machine communication viewpoint, the assignments identified by our method for 4- and 8-machines are the best. This suggests that we should adjust b value used in the search. A smaller b will guide the search process to have a bias toward load balancing. The assignment identified by our algorithm for 8-machines is shown in Table 10.

Table 10
Task assignment for uneven SOR with 8-machines

Group	0	1	2	3	4	5	6	7
Tasks	2,3,12	6–8	9–11	13–15	0,16–19	1,20–23	4,24–27	5,28–31

In the second phase of our experiments, machines with different speeds were introduced. To fully exploit the computation power of faster machines, the workload of a machine must be normalized as discussed previously. We found that the relative speed between machines varies not only from one application to another, but also from one configuration to another. For instance, if there are two-type CPUs, 80486-33 and pentium-90, in different 8-machine configurations, the relative speed between these CPUs in a system that has $7 \times 80486-33$ s and 1 pentium-90 differs to that has $6 \times 80486-33$ s and $2 \times$ pentium-90s. Based on our measurement, when the number ratio between 80486-33s and pentium-90s is 1:1, the speed ratio between these two CPUs is 1:7; when the number ratio changes to 3:1, the speed ratio changes to 1:5, further, if the number ratio is 7:1, the speed ratio becomes 1:3. The experiment results for SOR and N-Body are shown in Table 11). In the second phase, the assignments identified by our method are superior than that produced by the batch and interleaving methods.

Compared with Table 2 and Table 11, the performance is significantly improved when faster computers are introduced, especially for 2- and 4-machine cases. However, when we increase the computation power of a system without changing the size of a problem, the performance can be improved, but the margin decreases. In addition, the margin decrease depends on applications. For example, the performance margin of SOR becomes much smaller than that of N-body when there are 8-machines in a system, as shown in Table 11. This phenomenon confirms the finding of the Princeton team—DSMs perform very well for some applications, in particular for larger problems [13].

5. Conclusion and future works

Results on variety of applications with different characteristics suggest the proposed method is an effective optimization method. Using the HNN as the local optimizer, it can hit local optimum rapidly. In simple and/or unimodal spaces, the local optimum is also the global one. For complicated search spaces, equipped with a probabilistical *redirection* capability, the proposed method is able to obtain fast and effective solutions. A number of experiments were carried out and we found that our mapping algorithm effectively identifies optimal assignments in these experiments. Note that hill climbers other than HNN can also be used to search problem space, so does other probabilistical *redirections*. The effectiveness of the other methods on task assignment requires further study.

Table 11
Performance for systems with different machines

80486:pentium	1:1	3:1	2:2	7:1	6:2
<i>Execution time for SOR (s)</i>					
Batch	49.166	49.120	41.628	43.841	38.761
Random	49.166	45.402	39.789	45.778	49.775
CHNN	47.653	41.725	32.002	32.191	31.980
<i>Execution time for N-body (s)</i>					
Batch	274.423	179.000	134.955	120.353	104.641
Random	274.423	186.860	138.015	123.640	114.638
CHNN	274.423	178.295	132.622	115.512	94.497

Our future work is to solve the mapping problem of the loop applications with the irregular access patterns. In addition, we will improve the method proposed in this paper to keep track of the temporal order of the memory reference patterns of tasks on a DSM. We are going to use this knowledge to guide the proposed method to determine the optimal initial data location in a DSM to minimize page migration between machines. It can also eliminate *false sharing*.

Appendix A

The HNN has been found to always converge to a stable state in the network evolutions. This is guaranteed by the monotonicity of its energy function. For task mapping, we can prove that the property is preserved, i.e. ΔE , the change of energy between two consecutive steps in the network evolution, is negative. First, consider the energy function of the HNN

$$E = \sum_{x=1}^m \sum_{y=1}^m \sum_{i=1}^n \sum_{j=1}^n T_{xijy} V_{xi} V_{yj}.$$

The energy can be divided into four terms, consisting of ($i = k, j = k$), ($i = k, j \ni k$), ($i \ni k, j = k$) and others, then

$$\begin{aligned} E = & \sum_x^m \sum_y^m T_{xkyk} V_{xk} V_{yk} + \sum_x^m \sum_y^m \sum_{j=1, j \neq k}^m T_{xkyj} V_{xk} V_{yj} \\ & + \sum_x^m \sum_y^m \sum_{i=1, i \neq k}^n T_{xiky} V_{xi} V_{yk} \\ & + \sum_x^m \sum_y^m \sum_{i=1, i \neq k}^n \sum_{j=1, j \neq k}^n T_{xijy} V_{xi} V_{yj} \end{aligned} \quad (A1)$$

The fourth term of Eq. (A1) is independent of the states of a specific column k . Therefore, only the first three terms are affected by the state change of a specific column k . The winner-take-all rule guarantees that each column contains only one active neuron. Without loss of generality, we assume that neuron (b, k) is the only active neuron on the k -th column before a specific updating cycle of column k , i.e. $V_{bk}^{\text{old}} = 1$. While, all other neurons in the column are resting, $V_{xk}^{\text{old}} = 0, \{\forall x | x \neq b\}$. Further assume that after an update cycle, neuron (a, k) becomes the winning node, i.e.

$V_{ak}^{\text{new}} = 1$ and $V_{xk}^{\text{new}} = 0, \{\forall x | x \neq a\}$. According to the winner-take-all rule, it has

$$\sum_{y=1}^m \sum_{j=1}^n (T_{akyj} + T_{yjak}) = \min_{x=1,2,\dots,m} \left[\sum_{y=1}^m \sum_{j=1}^n (T_{xkyj} + T_{yjxk}) \right],$$

which implies that

$$\sum_{y=1}^m \sum_{j=1}^n (T_{akyj} + T_{yjak}) < \sum_{y=1}^m \sum_{j=1}^n (T_{bkyj} + T_{yjbk}).$$

Thus, the energy change ΔE due to the update on the k -th column can be calculated as follows:

$$\begin{aligned} \Delta E = & E(G^{\text{new}}) - E(G^{\text{old}}) \\ = & \left\{ T_{akak} + \sum_{y=1}^m \sum_{\substack{j=1 \\ j \neq k}}^n T_{akyj} V_{yj} + \sum_{x=1}^m \sum_{\substack{i=1 \\ i \neq k}}^n T_{xiak} V_{xi} \right\} \\ & - \left\{ T_{bkkk} + \sum_{y=1}^m \sum_{\substack{j=1 \\ j \neq k}}^n T_{bkyj} V_{yj} + \sum_{x=1}^m \sum_{\substack{i=1 \\ i \neq k}}^n T_{xibk} V_{xi} \right\} \\ = & \left\{ -T_{akak} + \sum_{y=1}^m \sum_{j=1}^n T_{akyj} V_{yj} + \sum_{x=1}^m \sum_{i=1}^n T_{xiak} V_{xi} \right\} \\ & - \left\{ -T_{bkkk} + \sum_{y=1}^m \sum_{j=1}^n T_{bkyj} V_{yj} + \sum_{x=1}^m \sum_{i=1}^n T_{xibk} V_{xi} \right\} \\ = & \left(\sum_{y=1}^m \sum_{j=1}^n T_{akyj} V_{yj} + \sum_{y=1}^m \sum_{j=1}^n T_{yjak} V_{yj} \right) \\ & - \left(\sum_{y=1}^m \sum_{j=1}^n T_{bkyj} V_{yj} + \sum_{y=1}^m \sum_{j=1}^n T_{yjbk} V_{yj} \right) + T_{bkkk} - T_{akak} \\ = & \left(\sum_{y=1}^m \sum_{j=1}^n T_{akyj} V_{yj} + \sum_{y=1}^m \sum_{j=1}^n T_{yjak} V_{yj} \right) \\ & - \left(\sum_{y=1}^m \sum_{j=1}^n T_{bkyj} V_{yj} + \sum_{y=1}^m \sum_{j=1}^n T_{yjbk} V_{yj} \right) + \text{comp}_k^2 \\ & - \text{comp}_k^2 \\ = & \sum_{y=1}^m \sum_{j=1}^n (T_{akyj} + T_{yjak}) V_{yj} - \sum_{y=1}^m \sum_{j=1}^n (T_{bkyj} + T_{yjbk}) V_{yj}. \end{aligned}$$

where $V_{ak}^{\text{new}} = 1; V_{xk}^{\text{new}} = 0, \{\forall x | x \neq a\}; V_{bk}^{\text{old}} = 1; V_{xk}^{\text{old}} = 0, \{\forall x | x \neq b\}$; and $T_{akbk} = T_{bkkk} = \text{comp}_k^2$ are used in the above derivation, we know that $\Delta E < 0$, implying that

the energy change in the updating is negative. Therefore, the convergence of the HNN is guaranteed.

References

- [1] C.C. Chen, W.H. Tsai, A graph matching approach to optimal task assignment in distributed computing systems using a minmax criterion, *IEEE Transactions on Computers* C-34 (1985) 197–203.
- [2] V.M. Lo, Heuristic algorithms for task assignment in distributed systems, *IEEE Transactions on Computers* C-37 (11) (1988) 1384–1397.
- [3] H. El-Rewini, T.G. Lewis, Scheduling parallel program tasks onto arbitrary target machines, *Journal of Parallel and Distributed Computing* 9 (1990) 138–153.
- [4] V. Chaudhary, J.K. Aggarwal, A generalized scheme for mapping parallel algorithms, *IEEE Transactions on Parallel and Distributed Systems* 4 (3) (1993) 328–346.
- [5] D. Fernandez-baca, Allocating modules to processors in a distributed system, *IEEE Transactions on Software Engineering* 15 (11) (1989) 1427–1436.
- [6] R. Desai, R. Patil, SALO: combining simulated annealing and local optimization for efficient global optimization, *Proceedings of the ninth Florida AI Research Symposium*, June 1996.
- [7] T.L. Casavant, J.G. Kuhl, A taxonomy of scheduling in general-purpose distributed computing systems, *IEEE Transactions on Software Engineering* SE-14 (2) (1988) 42–45.
- [8] H. El-Rewini, T.G. Lewis, H.H. Ali, *Task scheduling in parallel and distributed systems*, Prentice Hall, Englewood Cliffs, NJ, 1994.
- [9] J.J. Hopfield, D.W. Tank, Neural computation of decisions in optimization problems, *Biol. Cybern.* 52 (1985) 141–152.
- [10] C.K. Shieh, A.C. Lai, J.C. Ueng, Cohesion: an efficient distributed shared memory system supporting multiple memory consistency models, *Proceedings of the First Aizu International Symposium on Parallel Algorithms/Architecture Synthesis*, March 1995.
- [11] A.S. Tanenbaum, *Distributed operating systems*, Prentice Hall, Englewood Cliffs, NJ, 1995.
- [12] J.B. Carter, D. Khandekar, L. Kamb, distributed shared memory: where we are and where we should be headed, *Proceedings of the fifth Workshop on Hot Topics in Operating Systems*, 1995.
- [13] L. Iftode, J.P. Singh, K. Li, Understanding application performance on shared virtual memory systems, *Proceedings of the 23rd Annual International Symposium of Computer Architecture*, May 1996.