

We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900

Open access books available

186,000

International authors and editors

200M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



Optimization of Mapping Graphs of Parallel Programs onto Graphs of Distributed Computer Systems by Recurrent Neural Network

Mikhail S. Tarkov

*A.V. Rzhanov's Institute of Semiconductor Physics
Siberian Branch, Russian Academy of Sciences
Russia*

1. Introduction

A distributed computer system (CS) is a set of elementary computers (ECs) connected by a network that is program-controlled from these computers. Each EC includes a computing module (CM) (processor with a memory) and a system unit (message router). The message router operates under CM control and has input and output ports connected to the output and input ports of the neighboring ECs, correspondingly. The CS structure is described by the graph $G_s(V_s, E_s)$, where V_s is the set of ECs and $E_s = V_s \times V_s$ is the set of connections between the ECs.

The topology of a distributed system may undergo changes while the system is operating, due to failures or repairs of communication links, as well as due to addition or removal of ECs (Bertsekas, Tsitsiklis, 1989). The CS robustness means that failures and recoveries of the ECs bring only to increasing and decreasing time of a task execution. Control on resources and tasks in the robust distributed CS suggested solution of the following problems (Tarkov, 2003, 2005): the CS optimal decomposition to connected subsystems; mapping parallel program structures onto the subsystem structures; static and dynamic balancing computation load among CMs of the computer system (subsystem); static and dynamic message routing (implementation of paths for data transfer), i.e. balancing communication load in the CS network; distribution of program and data copies for organization of fault tolerant computations; subsystem reconfiguration and redistribution of computation and communication load for computation recovery from failures, and so on.

As a rule, all these problems are considered as combinatorial optimization problems (Korte & Vygen, 2006), solved by centralized implementation of some permutations on data structures distributed on elementary computers of the CS. The centralized approach to the problem solution suggests gathering data in some (central) EC, solving optimization problem in this EC with the following scattering results to all ECs of the system (subsystem). As a result we have sequential (and correspondingly slow) method for the problem solution with great overhead for gathering and scattering data. Now a decentralized approach is significantly developed for solution of problems of control resources and tasks in computer

systems with distributed memory (Tel G., 1994), and in many cases this approach allows to parallelize the problem solution.

Massive parallelism of data processing in neural networks allows us to consider neural networks as a perspective, high-performance, and reliable tool for solution of complicated optimization problems (Melamed, 1994; Trafalis & Kasap, 1999; Smith, 1999; Haykin, 1999; Tarkov, 2006). The recurrent neural network (Hopfield & Tank, 1985; Wang, 1993; Siqueira, Steiner & Scheer, 2007, 2010; Serpen & Patwardhan, 2007; Serpen, 2008; da Silva, Amaral, Arruda & Flauzino, 2008; Malek, 2008) is a most interesting tool for solution of discrete optimization problems. A model of a globally converged recurrent Hopfield neural network is in good accordance with Dijkstra's self-stabilization paradigm (Dijkstra, 1974). This signifies that the mappings of parallel program graphs onto graphs of distributed computer systems, carried out by Hopfield networks, are self-stabilizing (Jagota, 1999). An importance of usage of the self-stabilizing mappings is caused by a possibility of breaking the CS graph regularity by failures of ECs and intercomputer connections.

For distributed CSs, the graph of a parallel program $G_p(V_p, E_p)$ is usually determined as a set V_p of the program branches (virtual elementary computers) interacting with each other by the point-to-point principle through transferring messages via logical (virtual) channels (which may be unidirectional or bidirectional) of the set $E_p = V_p \times V_p$. Interactions between the processing modules are ordered in time and regular in space for most parallel applications (line, ring, mesh, etc.) (Fig. 1).

For this reason, the maximum efficiency of information interactions in advanced high-performance CSs is obtained by using regular graphs $G_s(V_s, E_s)$ of connections between individual computers (hypercube, torus) (Parhami, 2002; Yu, Chung & Moreira, 2006; Balaji, Gupta, Vishnu & Beckman, 2011). The hypercube structure is described by a graph known as a m -dimensional Boolean cube with a number of nodes $n = 2^m$. Toroidal structures are m -dimensional Euclidean meshes with closed boundaries. For $m = 2$, we obtain a two-dimensional torus (2D-torus) (Fig. 2); for $m = 3$, we obtain a 3D-torus.

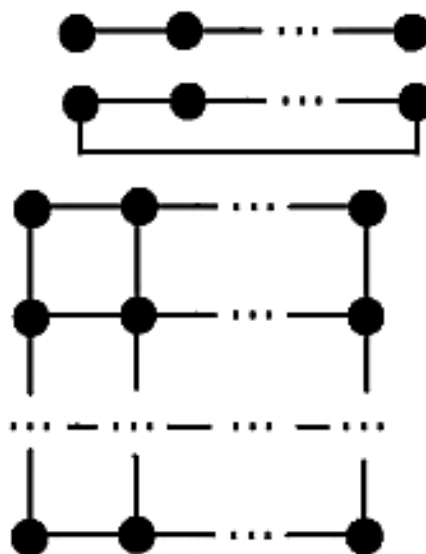


Fig. 1. Typical graphs of parallel programs (line, ring and mesh)

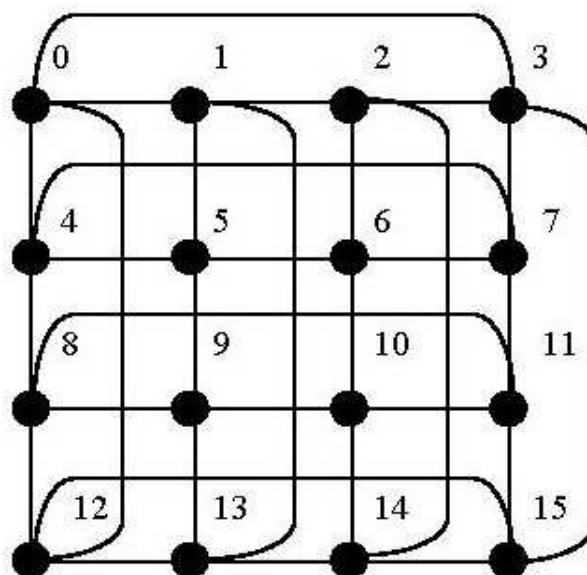


Fig. 2. Example of a 2D-torus

In this paper, we consider a problem for mapping graph $G_p(V_p, E_p)$ of a parallel program onto graph $G_s(V_s, E_s)$ of a distributed CS, where $n = |V_p| = |V_s|$ is a number of program branches (of ECs). The mapping objective is to map nodes of the program graph G_p onto nodes of the system graph G_s one-to-one to carry out mapping G_p edges onto edges of G_s (to establish an isomorphism between the program graph G_p and a spanning subgraph of the system graph G_s).

In section 2, we consider a recurrent neural network as a universal technique for solution of mapping problems. It is a local optimization technique, and we propose additional modifications (for example, penalty coefficients and splitting) to improve the technique scalability.

In section 3, we propose an algorithm based on the recurrent neural network and WTA ("Winner takes all") approach for the construction of Hamiltonian cycles in graphs. This algorithm maps only line- and ring-structured parallel programs. So, it is less universal than the technique proposed in section 2 but more powerful because it implements a global optimization approach, and hence it is very more scalable than the traditional recurrent neural networks.

2. Mapping graphs of parallel programs onto graphs of distributed computer systems by recurrent neural networks

Let us consider a matrix v of neurons with size $n \times n$, each row of the matrix corresponds to some branch of a parallel program and every column of the matrix corresponds to some EC. Each row and every column of the matrix v must contain only one nonzero entry equal to one, other entries must be equal to zero. Let the distance between the neighboring nodes of the CS graph is taken as a unit distance and d_{ij} is the length of the shortest path between nodes i and j in the CS graph. Then we define the energy of the corresponding Hopfield neural network by the Lyapunov function

$$\begin{aligned}
 L &= C \cdot L_c + D \cdot L_d, \\
 L_c &= \frac{1}{2} \left[\sum_x \left(\sum_j v_{xj} - 1 \right)^2 + \sum_i \left(\sum_y v_{yi} - 1 \right)^2 \right], \\
 L_d &= \frac{1}{2} \sum_x \sum_i \sum_{y \in Nb_p(x)} \sum_{j \neq i} v_{xi} v_{yj} d_{ij}.
 \end{aligned} \tag{1}$$

Here d_{ij} is the distance between nodes i and j of the system graph corresponding to adjacent nodes of the program graph (a "dilation" of the edge of the program graph on the system graph), $Nb_p(x)$ is a neighborhood of the node x on the program graph.

The value v_{xi} is a state of the neuron in the row x and column i of the matrix v , C and D are parameters of the Lyapunov function. L_c is minimal when each row and every column of v contains only one unity entry (all other entries are zero). Such matrix v is a correct solution of the mapping problem (Fig. 3).

$$\begin{bmatrix}
 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0
 \end{bmatrix}$$

Fig. 3. Example of correct matrix of neuron states

The minimum of L_d provides minimum of the sum of distances between adjacent G_p nodes mapped onto nodes of the system graph G_s (Fig. 4).

The Hopfield network minimizing the function (1) is described by the equation

$$\frac{\partial u_{xi}}{\partial t} = - \frac{\partial L}{\partial v_{xi}} \tag{2}$$

where u_{xi} is an activation of the neuron with indices x, i ($x, i = 1, \dots, n$),

$$v_{xi} = \frac{1}{1 + \exp(-\beta u_{xi})}$$

is the neuron state (output signal), β is the activation parameter.

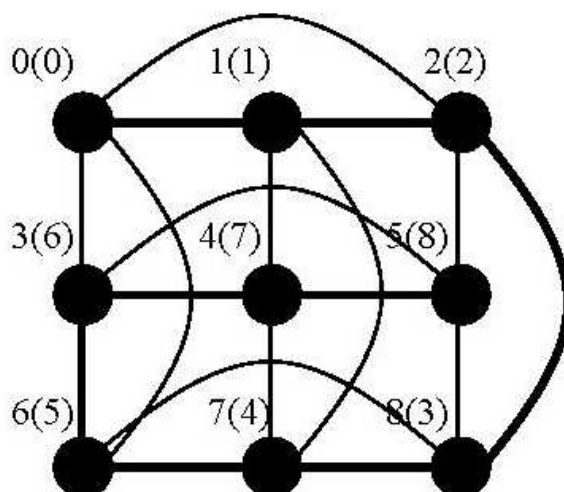


Fig. 4. Example of optimal mapping of “line”-graph onto torus (the mapping is distinguished by bold lines; the line-graph’s node numbers are shown in brackets)

From (1) and (2) we have

$$\frac{\partial u_{xi}}{\partial t} = -C \left(\sum_j v_{xj} + \sum_y v_{yi} - 2 \right) - D \sum_{y \in Nb_p(x)} \sum_{j \neq i} v_{yj} d_{ij}. \quad (3)$$

A difference approximation of Equation (3) yields

$$u_{xi}^{t+1} = u_{xi}^t - \Delta t \cdot \left[C \left(\sum_j v_{xj} + \sum_y v_{yi} - 2 \right) + D \sum_{y \in Nb_p(x)} \sum_{j \neq i} v_{yj} d_{ij} \right], \quad (4)$$

where Δt is a temporal step. Initial values u_{xi}^0 ($x, i = 1, \dots, n$) are stated randomly.

A choice of parameters $\beta, \Delta t, C, D$ determines a quality of the solution v of Equation (4). In accordance with (Feng & Douligieris, 2001) for the problem (1)-(4) a necessary condition of convergence is

$$C > D \frac{f_{\min}}{2(1-\alpha)}, \quad (5)$$

where $f_{\min} = \min \left\{ \sum_{y \in Nb_p(x)} \sum_{j \neq i} v_{yj} d_{ij} \right\}$, $\alpha \in [0, 1)$ and α being a value close to 1. For a parallel program graph of a line type $f_{\min} = 1$. For example, taking $\alpha = 0.995$ for the line we have

$$C > 100 \cdot D.$$

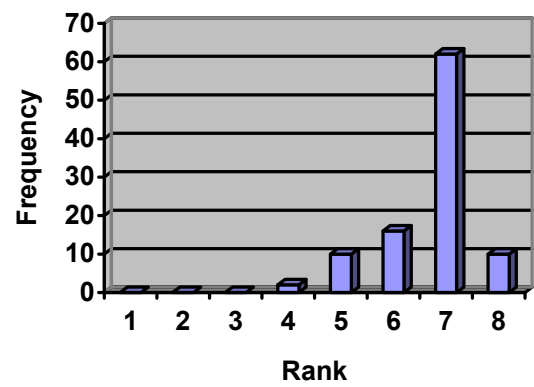
From (4) and (5) it follows that the parameters Δt and D are equally influenced on the solution of the equation (4). Therefore we state $\Delta t = 1$ and have the equation

$$u_{xi}^{t+1} = u_{xi}^t - C \cdot \left(\sum_j v_{xj} + \sum_y v_{yi} - 2 \right) - D \cdot \sum_{y \in Nb_p(x)} \sum_{j \neq i} v_{yj} d_{ij}. \tag{6}$$

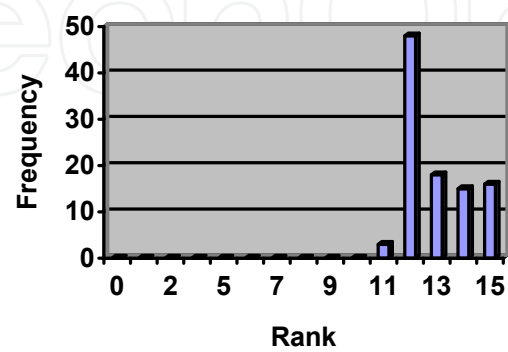
Let $\beta = 0.1$ (this value was stated experimentally). We will try to choose the value D to provide the absence of incorrect solutions.

2.1 Mapping by the Hopfield network

Let us evaluate the mapping quality by a number of coincidences of the program edges with edges of the system graph. We call this number a mapping rank. The mapping rank is an approximate evaluation of the mapping quality because the mappings with different dilations of the edges of the program graph may have the same mapping rank. Nevertheless, the maximum rank value, which equals to the number $|E_p|$ of edges of the program graph, corresponds to optimal mapping, i.e. to a global minimum of L_d in (1). Our objective is to determine the mapping algorithm parameters providing maximum probability of the optimal mapping.

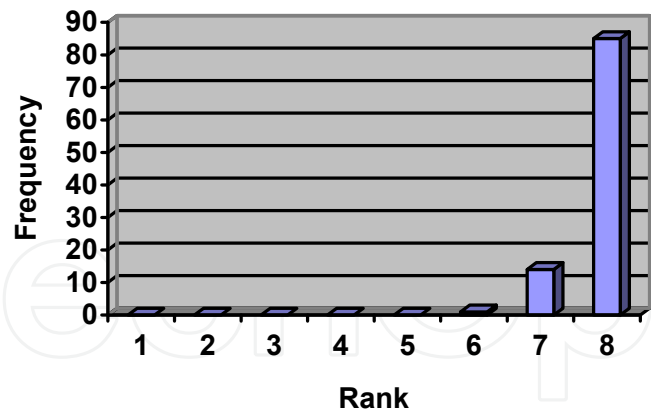


a) $n = 9$

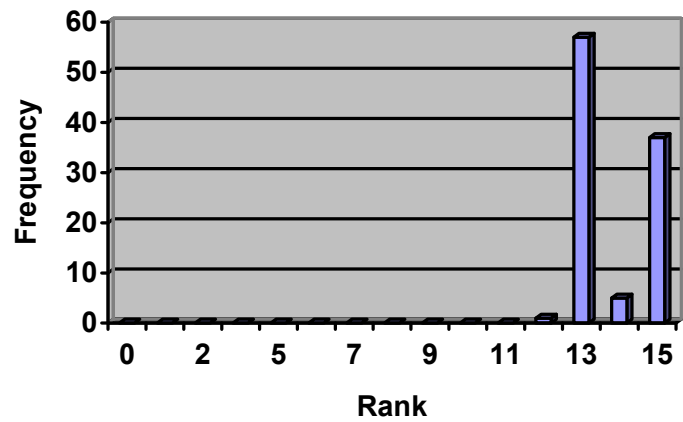


b) $n = 16$

Fig. 5. Histograms of mappings for the neural network (6)



a) $n = 9$



b) $n = 16$

Fig. 6. Histograms of mappings for the neural network (8)

As an example for investigation of the mapping algorithm we consider the mapping of a line-type program graph onto a 2D-torus. Maximal value of the mapping rank for a line with n nodes is obviously equal to $n-1$.

For experimental investigation of the mapping quality, the histograms of the mapping rank frequencies are used for a number of experiments equals to 100. The experiments for mapping the line onto the 2D-torus with the number of nodes $n = l^2$, $l = 3, 4$, where l is the cyclic subgroup order, are carried out.

For $D \geq 8$ the correct solutions are obtained for $n = 9$ and $n = 16$, but as follows from Fig. 5a and Fig. 5b for $D = 8$, the number of solutions with optimal mapping, corresponding to the maximal mapping rank, is small.

To increase the frequency of optimal solutions of Equation (6) we replace the distance values d_{ij} by the values

$$c_{ij} = \begin{cases} d_{ij} & d_{ij} = 1 \\ p \cdot d_{ij} & d_{ij} > 1 \end{cases} \quad (7)$$

where p is a penalty coefficient for the distance d_{ij} exceeding the value 1, i.e. for non-coincidence of the edge of the program graph with the edge of the system graph. So, we obtain the equation

$$u_{xi}^{t+1} = u_{xi}^t - C \cdot \left(\sum_j v_{xj} + \sum_y v_{yi} - 2 \right) - D \cdot \sum_{y \in Nb_p(x)} \sum_{j \neq i} v_{yj} c_{ij}. \quad (8)$$

For the above mappings with $p = n$ we obtain the histograms shown on Fig. 6a and Fig. 6b. These histograms indicate the improvement of the mapping quality but for $n = 16$ the suboptimal solutions with the rank 13 have maximal frequency.

2.2 Splitting method

To decrease a number of local extremums of Function (1), we partition the set $\{1, 2, \dots, n\}$ of subscripts x and i of the variables v_{xi} to K sets

$$I_k = \{(k-1)q, (k-1)q+1, \dots, k \cdot q\}, \quad q = n/K, \quad k = 1, 2, \dots, K,$$

and map the subscripts $x \in I_k$ only to the subscripts $i \in I_k$, i.e. we reduce the solution matrix v to a block-diagonal form (Fig. 7) and the Hopfield network is described by the equation

$$u_{xi}^{t+1} = u_{xi}^t - C \cdot \left(\sum_{j \in I_k} v_{xj} + \sum_{y \in I_k} v_{yi} - 2 \right) - D \cdot \sum_{y \in Nb_p(x)} \sum_{j \neq i} v_{yj} c_{ij}, \quad (9)$$

$$v_{xi} = \frac{1}{1 + \exp(-\beta u_{xi})}, \quad x, i \in I_k, k = 1, 2, \dots, K.$$

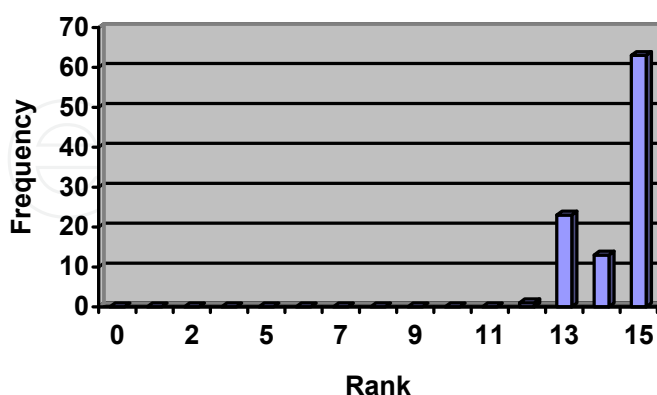
In this case $v_{xi} = 0$ for $x \in I_k, i \notin I_k, k = 1, 2, \dots, n$.

$$\begin{bmatrix} v_{00} & v_{01} & v_{02} & 0 & 0 & 0 \\ v_{10} & v_{11} & v_{12} & 0 & 0 & 0 \\ v_{20} & v_{21} & v_{22} & 0 & 0 & 0 \\ 0 & 0 & 0 & v_{33} & v_{34} & v_{35} \\ 0 & 0 & 0 & v_{43} & v_{44} & v_{45} \\ 0 & 0 & 0 & v_{53} & v_{54} & v_{55} \end{bmatrix}$$

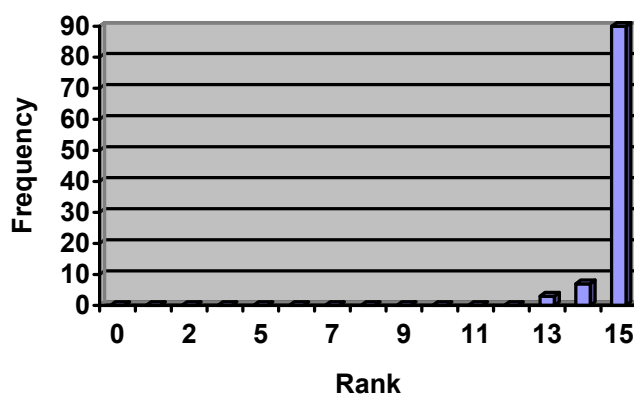
Fig. 7. Example of block-diagonal solution matrix for $K = 2$

In this approach which we call a splitting, for mapping line with the number of nodes $n = 16$ onto 2D-torus, we have for $K = 2$ the histogram presented on Fig. 8a.

From Fig. 6b and Fig. 8a we see that the splitting method essentially increases the frequency of optimal mappings. The increase of the parameter D up to the value $D = 32$ results in additional increase of the frequency of optimal mappings (Fig. 8b).



a) $n = 16, K = 2, D = 8$.



b) $n = 16, K = 2, D = 32$.

Fig. 8. Histograms of mappings for the neural network (9)

2.3 Mapping by the Wang network

In a recurrent Wang neural network (Wang, 1993; Hung & Wang, 2003) L_d in Expression (1) is multiplied by the value $\exp(-t/\tau)$ where τ is a parameter. For the Wang network Equation (9) is reduced to

$$u_{xi}^{t+1} = u_{xi}^t - C \cdot \left(\sum_{j \in I_k} v_{xj} + \sum_{y \in I_k} v_{yi} - 2 \right) - D \cdot \sum_{y \in Nb_p(x)} \sum_{j \neq i} v_{yj} c_{ij} \exp(-t/\tau), \quad (10)$$

$$v_{xi} = \frac{1}{1 + \exp(-\beta u_{xi})}, \quad x, i \in I_k, k = 1, 2, \dots, K.$$

We note that in experiments we frequently have incorrect solutions if for a given maximal number of iterations $t_{\max}$ (for example, $t_{\max}=10000$) the condition of convergence

$\sum_{x,i} |u_{xi}^t - u_{xi}^{t-1}| < \varepsilon$, $\varepsilon = 0.01$ is not satisfied. The introduction of factor $\exp(-t/\tau)$ accelerates the convergence of the recurrent neural network and the number of incorrect solutions is reduced.

So, for the three-dimensional torus with $n = 3^3 = 27$ nodes and $p = n, K = 3, D = 4096, \beta = 0.1$ in 100 experiments we have the following results:

- 1. On the Hopfield network (9) we have 23 incorrect solutions, 43 solutions with Rank 25 and 34 optimal solutions (with Rank 26) (Fig. 9).
- 2. On the Wang network (10) with the same parameters and $\tau = 500$ we have all (100) correct solutions, where 27 solutions have Rank 25 and 73 solutions are optimal (with Rank 26) (Fig. 10).

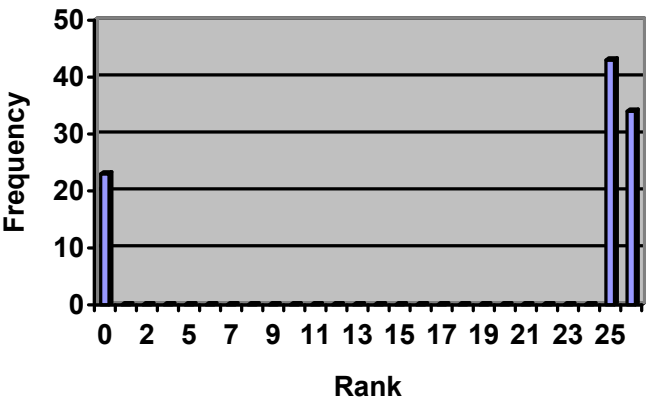


Fig. 9. Histogram of mappings for the Hopfield network ($n = 3^3 = 27$)

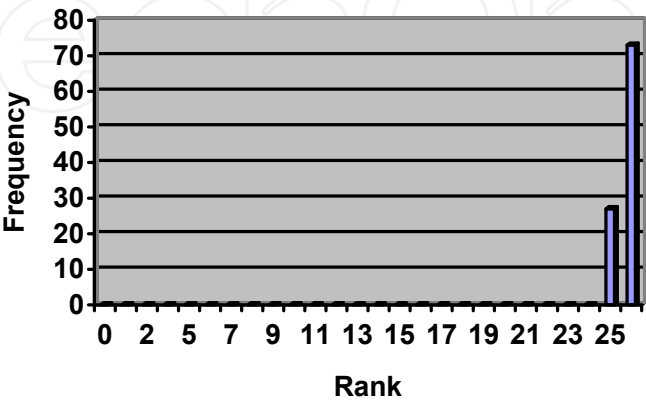


Fig. 10. Histogram of mappings for the Wang network ($n = 3^3 = 27$)

As a result we have high frequency of optimal solutions (for 100 experiments):

1. more than 80% for the two-dimensional tori ($n = 3^2 = 9$ and $n = 4^2 = 16$);
2. more than 70% for three-dimensional torus ($n = 3^3 = 27$).

Further investigations must be directed to increasing the probability of getting optimal solutions of the mapping problem when the number of the parallel program nodes is increased.

3. Construction of Hamilton cycles in graphs of computer systems

In this section, we consider algorithms for nesting ring structures of parallel programs of distributed CSs, which are based on using recurrent neural networks, under the condition $n = |V_p| = |V_s|$. Such nesting reduces to constructing a Hamiltonian cycle in the CS graph and is based on solving the traveling salesman problem using the matrix of distances $d_{ij}(i, j = 1, \dots, n)$ between the CS graph nodes, with the distance between the neighboring nodes of the CS graph taken as a unit distance.

The traveling salesman problem can be formulated as an assignment problem (Wang, 1993; Siqueira, Steiner & Scheer, 2007, 2010)

$$\min \sum_{i=1}^n \sum_{j \neq i}^n c_{ij} x_{ij}, \quad (11)$$

under the constraints

$$\begin{aligned} x_{ij} &\in \{0, 1\}, \\ \sum_{i=1}^n x_{ij} &= 1, j = 1, \dots, n, \\ \sum_{j=1}^n x_{ij} &= 1, i = 1, \dots, n. \end{aligned} \quad (12)$$

Here, c_{ij} , $i \neq j$ is the cost of assignment of the element i to the position j , which corresponds to motion of the traveling salesman from the city i to the city j ; x_{ij} is the decision variable: if the element i is assigned to the position j , then $x_{ij} = 1$, otherwise, $x_{ij} = 0$.

For solving problem (11) – (12), J. Wang proposed a recurrent neural network that is described by the differential equation

$$\frac{\partial u_{ij}(t)}{\partial t} = -C \left(\sum_{k=1}^n x_{ik}(t) + \sum_{l=1}^n x_{lj}(t) - 2 \right) - D \cdot c_{ij} \exp(-t / \tau), \quad (13)$$

where $x_{ij} = g(u_{ij}(t))$, $g(u) = 1 / [1 + \exp(-\beta u)]$. A difference approximation of Equation (13) yields

$$u_{ij}^{t+1} = u_{ij}^t - \Delta t \cdot \left[C \left(\sum_{k=1}^n x_{ik}(t) + \sum_{l=1}^n x_{lj}(t) - 2 \right) - D \cdot c_{ij} \exp\left(-\frac{t}{\tau}\right) \right], \quad (14)$$

Here $\beta, C, D, \tau, \Delta t$ are parameters of the neural network.

Siqueira et al. proposed a method of accelerating the solution of the system (14), which is based on the WTA ("Winner takes all") principle. The algorithm proposed below was developed on the basis of this method.

1. A matrix $\|x_{ij}(0)\|$ of random values $x_{ij}(0) \in [0, 1]$ is generated. Iterations (14) are performed until the following inequality is satisfied for all $i, j = 1, \dots, n$:

$$\sum_{k=1}^n x_{ik}(t) + \sum_{l=1}^n x_{lj}(t) - 2 \leq \delta,$$

where δ is the specified accuracy of satisfying constraints (12).

2. Transformation of the resultant decision matrix $\|x_{ij}\|$ is performed:

- 2.1. $i = 1$.

- 2.2. The maximum element $x_{i, j_{\max}}$ is sought in the i th row of the matrix ($j_{\max}$ is the number of the column with the maximum element).

- 2.3. The transformation $x_{i, j_{\max}} = 1$ is performed. All the remaining elements of the i th row and of the column numbered $j_{\max}$ are set to zero. Then, there follows a transition to the row numbered $j_{\max}$. Steps 2.2 and 2.3 are repeated until the cycle returns to the first row, which means that the cycle construction is finalized.

3. If the cycle returns to the row 1 earlier than the value 1 is assigned to n elements of the matrix $\|x_{ij}\|$, this means that the length of the constructed cycle is smaller than n . In this case, steps 1 and 2 are repeated.

To ensure effective operation of the algorithm of Hamiltonian cycle construction, the following values of the parameters of system (14) were chosen experimentally (by the order of magnitude): $D = 1, C = 10, \tau \geq 1000, \beta = 0.1$. Significant deviations of these parameters from the above-indicated values deteriorate algorithm operation, namely:

1. Deviations of the parameter C from the indicated value (at a fixed value of D) deteriorate the solution quality (the cycle length increases).
2. A decrease in τ increases the number of non-Hamiltonian ring-shaped routes.
3. An increase in β deteriorates the solution quality. A decrease in β increases the number of iterations (14).

It follows from (Feng & Douligieris, 2001) that $\Delta t \leq \Delta t_{\max}$, where $\Delta t_{\max} \leq \frac{1}{\beta C} = \frac{1}{0.1 \cdot 10} = 1$.

The experiments show that it is not always possible to construct a Hamiltonian cycle at $\Delta t = 1$, but cycle construction is successfully finalized if the step Δt is reduced. We reduced the step Δt as $\Delta t / 2$ if a correct cycle could not be constructed after ten attempts.

The parameters $c_{ij}, i \neq j$, are calculated by the formula (7) where d_{ij} is the distance between the nodes i and j of the graph, and $p > 1$ is the penalty coefficient applied if the distance d_{ij} exceeds 1. The penalty coefficient was introduced to ensure coincidence of transition in the travelling agent cycle with the edges of the CS graph.

We studied the use of iterative methods (Jacobi, Gauss-Seidel, and successive overrelaxation (SOR) methods (Ortega, 1988)) in solving Wang's system of equations. With the notation

$$r_{ij}^t = -C \left(\sum_{k=1}^n x_{ik}^t + \sum_{l=1}^n x_{lj}^t - 2 \right) - D c_{ij} \exp(-t / \tau)$$

the Jacobi method (method of simple iterations) of solving system (14) has the form

1. $u_{ij}^{t+1} = u_{ij}^t + \Delta t \cdot r_{ij}^t, i, j = 1, \dots, n;$
2. $x_{ij}^{t+1} = g(u_{ij}^{t+1}), g(u_{ij}^{t+1}) = \frac{1}{1 + \exp(-\beta u_{ij}^{t+1})}, i, j = 1, \dots, n.$

According to this method, new values of $x_{ij}^{t+1}, i, j = 1, \dots, n$, are calculated only after all values $u_{ij}^{t+1}, i, j = 1, \dots, n$, are found. In contrast to the method of simple iterations, the new value of x_{ij}^{t+1} in the Gauss-Seidel method is calculated immediately after finding the corresponding value of u_{ij}^{t+1} :

$$u_{ij}^{t+1} = u_{ij}^t + \Delta t \cdot r_{ij}^t, x_{ij}^{t+1} = g(u_{ij}^{t+1}), i, j = 1, \dots, n.$$

In the SOR method, the calculations are performed by the formulas

$$\begin{aligned} u_{new} &= u_{ij}^t + \Delta t \cdot r_{ij}^t, \\ u_{ij}^{t+1} &= \omega \cdot u_{new} + (1 - \omega) \cdot u_{ij}^t, \omega \in (0, 2), \\ x_{ij}^{t+1} &= g(u_{ij}^{t+1}), i, j = 1, \dots, n. \end{aligned}$$

With $\omega = 1$, the SOR method turns to the Gauss-Seidel method.

Experiments on 2D-tori with the group of automorphisms $E_2 = C_m \otimes C_m, n = m^2$ show that the Jacobi method can only be used for tori with a small number of nodes ($m \in \{3, 4\}$).

The SOR method can be used for tori with $m \in \{3, 4, 6\}$ with appropriate selection of the parameter $\omega < 1$. For $m \geq 8$, it is reasonable to use the Gauss-Seidel method ($\omega = 1$). Figure 11 shows an example of a Hamiltonian cycle constructed by a neural network in a 2D-mesh with $n = 16$ (the cycle is indicated by the bold line).

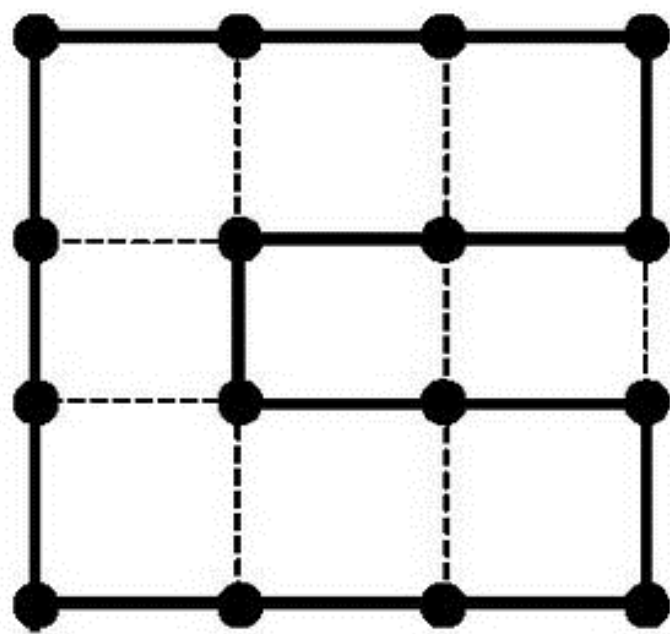


Fig. 11. Example of a Hamiltonian cycle in a 2D-mesh

In our experiments, we obtained Hamiltonian cycles (with the cycle length $L = n$) in 2D-meshes and 2D-tori for a number of experiments equals to 100 with up to $n = 1024$ nodes for $m = 2k$ and suboptimal cycle lengths $L = n + 1$ at $m = 2k + 1, k = 2, 3, \dots, 16$. The penalty coefficients p and the values of Δt with which the Hamiltonian cycles were constructed for $n = 16, 64, 256$, and 1024 , and also the times of algorithm execution on Pentium (R) Dual-Core CPU E 52 000, 2.5 GHz (the time equal to zero means that standard procedures did not allow registering small times shorter than 0.015 s) are listed in Tables 1 and 2.

n	$16 = 4^2$	$64 = 8^2$	$256 = 16^2$	$1024 = 32^2$
Δt	1	1	1	0,5
$p = n / 2$	8	16	128	512
Time, s	0	0.015	0.75	73.36

Table 1. 2D-mesh

n	$16 = 4^2$	$64 = 8^2$	$256 = 16^2$	$1024 = 32^2$
Δt	1	1	1	0.5
$p = n$	16	64	256	1024
Time, s	0	0	4.36	73.14

Table 2. 2D-torus

In addition to the quantities listed in Tables 1 and 2, Tables 3 and 4 give the relative increase $\varepsilon = \frac{L - n}{n}$ in the travelling salesman cycle length, as compared with the Hamiltonian cycle length, for a 3D-torus and hypercube.

n	$64 = 4^3$	$216 = 6^3$	$512 = 8^3$	$1000 = 10^3$
Δt	0.012	0.1	0.1	0.1
p	64	32	32	32
L	64	218	520	1010
ε	0	0.01	0.016	0.01
Time, s	0.625	0.313	12.36	97.81

Table 3. 3D-torus

n	16	64	256	1024
Δt	0.1	0.1	0.1	0,1
p	32	32	32	32
L	16	64	262	1034
ε	0	0	0.016	0.023
Time, s	0	0.062	99.34	1147

Table 4. Hypercube

It follows from Tables 3 and 4 that:

1. In 3D-tori, the Hamiltonian cycle was constructed for $n = 64$. With $n = 216, 512$, and 1000, suboptimal cycles were constructed, which were longer than the Hamiltonian cycles by no more than 1.6%.
2. In hypercubes, the Hamiltonian cycles were constructed for $n = 16$ and 64 (it should be noted that the hypercube is isomorphous to the 2D-torus with $n = 16$). For $n = 256$ and $n = 1024$, suboptimal cycles were constructed, which were longer than n by no more than 2.3%.

3.1 Construction of Hamiltonian cycles in toroidal graphs with edge defects

The capability of recurrent neural networks to converge to stable states can be used for mapping program graphs to CS graphs with violations of regularity caused by deletion of edges and/or nodes. Such violations of regularity are called defects. In this work, we study the construction of Hamiltonian cycles in toroidal graphs with edge defects. Experiments in 2D-tori with a deleted edge and with $n = 9$ to $n = 256$ nodes for $p = n$ were conducted. The experiments show that the construction of Hamiltonian cycles in these graphs by the above-described algorithm is possible, but the value of the step Δt at which the cycle can be constructed depends on the choice of the deleted edge. The method of automatic selection of the step Δt is described at the beginning of Section 3. Table 5 illustrates the dependence of the step Δt on the choice of the deleted edge in constructing an optimal Hamiltonian cycle by the SOR method in a 2D-torus with $n = 16$ nodes for $\omega = 0.125$.

Examples of Hamiltonian cycles constructed by the SOR method in a 2D-torus with $n = 16$ nodes are given in Figs. 12a and 12b. Figure 12a shows the cycle constructed in the torus without edge defects for $\omega = 0.5$ and $\Delta t = 0.25$. Figure 12b shows the cycle constructed in the torus with a deleted edge $(0, 12)$ for $\omega = 0.125$ and $\Delta t = 0.008$.

Edge	Δt	Edge	Δt
(0,12)	0,008	(5,9)	0,5
(0,3)	1,0	(6,7)	0,063
(0,1)	1,0	(6,10)	1,0
(0,4)	1,0	(7,11)	1,0
(1,13)	1,0	(8,11)	1,0
(1,2)	0,25	(8,9)	1,0
(1,5)	1,0	(8,12)	1,0
(2,14)	0,125	(9,10)	0,125
(2,3)	0,125	(9,13)	1,0
(2,6)	1,0	(10,11)	1,0
(3,15)	1,0	(10,14)	1,0
(3,7)	0,25	(11,15)	1,0
(4,7)	0,25	(12,15)	1,0
(4,5)	1,0	(12,13)	0,5
(4,8)	1,0	(13,14)	0,5
(5,6)	1,0	(14,15)	1,0

Table 5. Dependence Δt of the step on the choice of the deleted edge

Results discussed in this section should be considered as preliminary and opening the research field studying the relation between the quality of nesting of graphs of parallel algorithms to graphs of computer systems whose regularity is violated by node and edge defects and the parameters of neural network algorithms implementing this nesting.

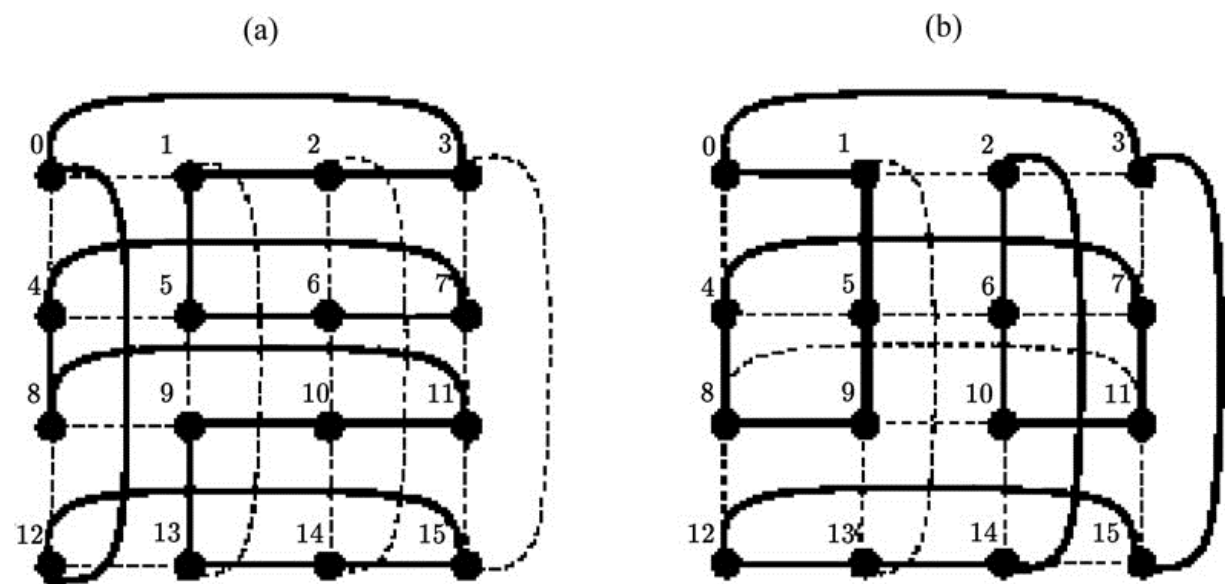


Fig. 12. Examples of Hamiltonian cycles in 2D-torus

3.2 Construction of Hamiltonian cycles by the splitting method

The time of execution of the above-described algorithm can be substantially reduced by using the following method:

- 1. Split the initial graph of the system into k connected subgraphs.
- 2. Construct a Hamiltonian cycle in each subgraph by the algorithm described above.
- 3. Unite the Hamiltonian cycles of the subgraphs into one Hamiltonian cycle.

For example, the initial graph of the system can be split into connected subgraphs by the algorithms proposed in (Tarkov, 2005).

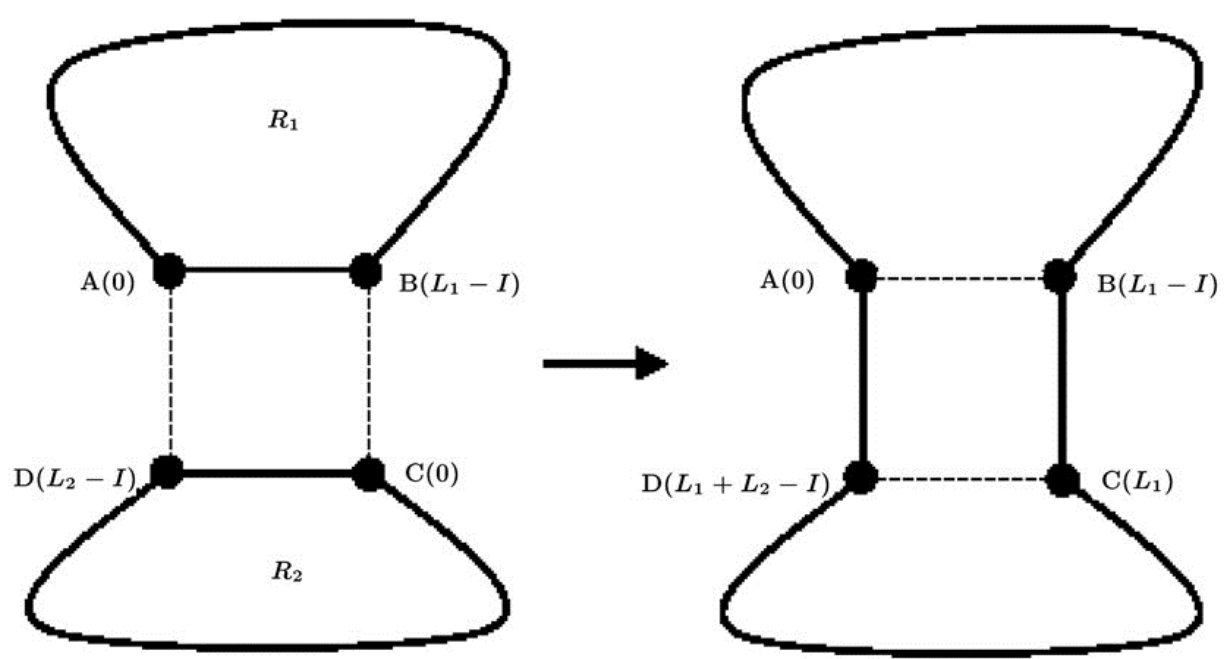


Fig. 13. Unification of cycles

For unification of two cycles R_1 and R_2 , it is sufficient if the graph of the system has a cycle ABCD of length 4 such that the edge AB belongs to the cycle R_1 and the edge CD belongs to the cycle R_2 (Fig. 13).

The cycles R_1 and R_2 can be united into one cycle by using the following algorithm:

1. Find the cycle ABCD possessing the above-noted property.
2. Eliminate the edge AB from the cycle and successively numerate the nodes of the cycle R_1 in such a way that to assign number 0 to the node A and assign number $L_1 - 1$, where L_1 is the length of the cycle R_1 , to the edge B. Include the edge BC into the cycle.
3. Eliminate the edge CD and successively numerate the nodes of the cycle R_2 so that the node C is assigned the number L_1 , and the node D is assigned the number $L_1 + L_2 - 1$, where L_2 is the length of the cycle R_2 . Include the edge DA into the cycle. The unified cycle of length $L_1 + L_2$ is constructed.

The cycles R_1 and R_2 , and also the resulting cycle are marked by bold lines in Fig. 12. The edges that are not included into the above-mentioned cycles are marked by dotted lines.

For comparison, Table 6 gives times (in seconds) of constructing Hamiltonian cycles in a 2D-mesh by the initial algorithm (t_1) and by the algorithm with splitting of cycle construction (t_2) with the number of subgraphs $k = 2$. The times are measured for $p = n$. The cycle construction time can be additionally reduced by parallel construction of cycles in subgraphs.

n	16	64	256	1024
t_1	0.02	0.23	9.62	595.8
t_2	0.01	0.03	2.5	156.19

Table 6. Comparison of cycle construction times in 2D-mesh

The proposed approach can be applied to constructing Hamiltonian cycles in arbitrary nonweighted nonoriented graphs without multiple edges and loops.

We can use the splitting method to construct Hamilton cycles in three-dimensional tori because the three-dimensional torus can be considered as a connected set of two-dimensional tori. So, the Hamilton cycle in three-dimensional torus can be constructed as follows:

1. Construct the Hamilton cycles in all of two-dimensional tori of the three-dimensional torus.
2. Unify the constructed cycles by the above unifying algorithm.

If the Hamilton cycles in all two-dimensional tori are optimal then the resulting Hamilton cycle in the three-dimensional torus is optimal too.

In the table 7 the times (in seconds) of construction of optimal Hamilton cycles in three-dimensional tori with $n = m \times m \times m$ nodes are presented: t_{seq} is the time of the sequential algorithm, t_{par} is the time of the parallel algorithm on processor Intel Pentium Dual-Core CPU E 52000, 2,5 GHz with usage of the parallel programming system OpenMP (Chapman, Jost & van der Pas, 2008), $S = t_{seq} / t_{par}$ is the speedup. The system of equations (14) was chosen for parallelization.

m	4	8	12	16	20	24	28	32
n	64	512	1728	4096	8000	13284	21952	32768
t_{seq}	0.125	0.062	0.906	6.265	31.22	133.11	390.61	2293.5
t_{par}	0.171	0.062	0.484	3.265	15.95	70.36	217.78	1397.6
S	0.73	1	1.87	1.92	1.96	1.89	1.79	1.64

Table 7. Construction of Hamiltonian Cycles in 3D-torus

So, the experiments show that the proposed algorithm:

1. Constructs optimal Hamilton cycles in 2D-tori with edge defects;
2. Allows to construct optimal Hamilton cycles in 3D-tori with tens of thousands of nodes (See Table 7).

4. Conclusion

A problem of mapping graphs of parallel programs onto graphs of distributed computer systems by recurrent neural networks is formulated. The parameter values providing the absence of incorrect solutions are experimentally determined. Optimal solutions are found for mapping a “line”-graph onto a two-dimensional torus due to introduction into Lyapunov function of penalty coefficients for the program graph edges not-mapped onto the system graph edges.

For increasing probability of finding optimal mapping, a method for splitting the mapping is proposed. The method essence is a reducing solution matrix to a block-diagonal form. The Wang recurrent neural network is used to exclude incorrect solutions of the problem of mapping the line-graph onto three-dimensional torus. This network converges quicker than the Hopfield one.

An efficient algorithm based on a recurrent neural Wang’s network and the WTA principle is proposed for the construction of Hamiltonian cycles (ring program graphs) in regular graphs (2D- and 3D-tori, and hypercubes) of distributed computer systems and 2D-tori disturbed by removing an arbitrary edge (edge defect). The neural network parameters for

the construction of Hamiltonian cycles and suboptimal cycles with a length close to that of Hamiltonian ones are determined.

Resulting algorithm allows us to construct optimal Hamilton cycles in 3D-tori with number of nodes up to 32768. The usage of this algorithm is actual in modern supercomputers having topology of the 3D-torus for organization of inter-processor communications in parallel solution of complicated problems.

Recurrent neural (Hopfield and Wang) network is a universal technique for solution of optimization problems but it is a local optimization technique, and we need additional modifications (for example, penalty coefficients and splitting) to improve the technique scalability.

The proposed algorithm for the construction of Hamiltonian cycles is less universal but more powerful because it implements a global optimization approach and so it is very more scalable than the traditional recurrent neural networks.

The traditional topology aware mappings ((Parhami, 2002; Yu, Chung & Moreira, 2006; Balaji, Gupta, Vishnu & Beckman, 2011)) are constructed especially for regular graphs (hypercubes and tori) of distributed computer systems. The proposed neural network algorithms are more universal and can be used for mapping program graphs onto graphs of distributed computer systems with defects of edges and nodes.

5. References

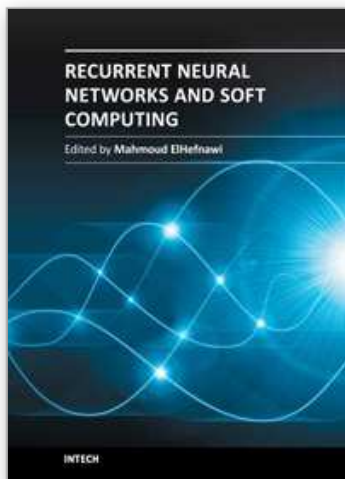
- Balaji, P.; Gupta, R.; Vishnu, A. & Beckman, P. (2011). Mapping Communication Layouts to Network Hardware Characteristics on Massive-Scale Blue Gene Systems, *Comput. Sci. Res. Dev.*, Vol. 26, pp.247-256
- Bertsekas, D.P. & Tsitsiklis, J.N. (1989). *Parallel and Distributed Computation: Numerical Methods*, Athena scientific, Belmont, Massachusetts: Prentis Hall
- Chapman, B., Jost, G. & van der Pas, R. (2008). *Using OpenMP : portable shared memory parallel programming*, Cambridge, Massachusetts :The MIT Press
- da Silva, I. N.; Amaral, W. C.; Arruda, L. V. & Flauzino, R. A. (2008). Recurrent Neural Approach for Solving Several Types of Optimization Problems, *Recurrent Neural Networks*, Eds. Xiaolin Hu and P. Balasubramaniam, Croatia: Intech, pp. 229-254
- Dijkstra, E.W. (1974). Self-stabilizing Systems in Spite of Distributed Control. *Commun. ACM*, Vol.17, No.11, pp. 643-644
- Feng, G. & Douligeris C. (2001). The Convergence and Parameter Relationship for Discrete-Time Continuous-State Hopfield Networks, *Proc. of Intern. Joint Conference on Neural Networks*
- Haykin, S. (1999). *Neural Networks. A Comprehensive Foundation*, Prentice Hall Inc.
- Hopfield, J.J. & Tank, D.W. (1985). Neural Computation of Decisions in Optimization Problems, *Biological Cybernetics*, Vol.52, pp.141-152
- Hung, D.L. & Wang, J. (2003). Digital hardware realization of a recurrent neural network for solving the assignment problem, *Neurocomputing*, Vol. 51, pp.447-461

- Jagota, A. (1999). Hopfield Neural Networks and Self-Stabilization, *Chicago Journal of Theoretical Computer Science*, Vol.1999,Article 6,
<http://mitpress.mit.edu/CJTCS/>
- Korte, B. & Vygen, J. (2006). *Combinatorial optimization. Theory and algorithms*. Bonn, Germany: Springer
- Malek, A. (2008). Applications of Recurrent Neural Networks to Optimization Problems, *Recurrent Neural Networks*, Eds. Xiaolin Hu and P. Balasubramaniam, Croatia: Intech, pp. 255-288
- Melamed, I.I. (1994). Neural networks and combinatorial optimization, *Automation and remote control*, Vol.55,No.11, pp.1553-1584
- Ortega, J.M. (1988). *Introduction to Parallel and Vector Solution of Linear Systems*, New York: Plenum
- Parhami, B. (2002). *Introduction to Parallel Processing. Algorithms and Architectures*, New York: Kluwer Academic Publishers
- Serpen, G. & Patwardhan, A. (2007). Enhancing Computational Promise of Neural Optimization for Graph-Theoretic Problems in Real-Time Environments, *DCDIS A Supplement, Advances in Neural Networks*, Vol. 14, No. S1, pp. 168--176
- Serpen, G. (2008). Hopfield Network as Static Optimizer: Learning the Weights and Eliminating the Guesswork, *Neural Processing Letters*, Vol.27, No.1, pp. 1-15
- Siqueira, P.H., Steiner, M.T.A., & Scheer, S. (2007). A New Approach to Solve The Travelling Salesman Problem, *Neurocomputing*, Vol. 70, pp.1013-1021
- Siqueira, P.H., Steiner, M.T.A., & Scheer, S. (2010). Recurrent Neural Network With Soft 'Winner Takes All' Principle for The TSP, *Proceedings of the International Conference on Fuzzy Computation and 2nd International Conference on Neural Computation*, pp. 265-270, SciTePress, 2010
- Smith, K. A. (1999). Neural Networks for Combinatorial Optimization: A Review of More Than a Decade of Research, *INFORMS Journal on Computing*, Vol.11,No.1, pp.15-34.
- Tarkov, M.S. (2003). Mapping Parallel Program Structures onto Structures of Distributed Computer Systems, *Optoelectronics, Instrumentation and Data Processing*, Vol. 39, No. 3, pp.72-83
- Tarkov, M.S. (2005). Decentralized Control of Resources and Tasks in Robust Distributed Computer Systems, *Optoelectronics, Instrumentation and Data Processing*, Vol. 41, No. 5, pp.69-77
- Tarkov, M.S. (2006). *Neurocomputer systems*, Moscow, Internet University of Inf. Technologies: Binom. Knowledge laboratory (in Russian)
- Trafalis T.B. & Kasap S. (1999). *Neural Network Approaches for Combinatorial Optimization Problems*, Handbook of Combinatorial Optimization, D.-Z. Du and P.M. Pardalos (Eds.), pp. 259-293, Kluwer Academic Publishers
- Tel, G. (1994). *Introduction to Distributed Algorithms*, Cambridge University Press, England, 1994
- Wang, J. (1993). Analysis and Design of a Recurrent Neural Network for Linear Programming, *IEEE Trans. On Circuits and Systems-I: Fundamental Theory and Applications*, Vol. 40, No.9, pp.613-618

- Yu, H.; Chung, I.H. & Moreira, J. (2006). Topology Mapping for Blue Gene/L Supercomputer, *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, ACM Press, New York, NY, USA, pp. 52–64

IntechOpen

IntechOpen



Recurrent Neural Networks and Soft Computing

Edited by Dr. Mahmoud ElHefnawi

ISBN 978-953-51-0409-4

Hard cover, 290 pages

Publisher InTech

Published online 30, March, 2012

Published in print edition March, 2012

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Mikhail S. Tarkov (2012). Optimization of Mapping Graphs of Parallel Programs Onto Graphs of Distributed Computer Systems by Recurrent Neural Network, Recurrent Neural Networks and Soft Computing, Dr. Mahmoud ElHefnawi (Ed.), ISBN: 978-953-51-0409-4, InTech, Available from:
<http://www.intechopen.com/books/recurrent-neural-networks-and-soft-computing/optimization-of-mapping-graphs-of-parallel-programs-onto-graphs-of-distributed-computer-systems-by-r>

INTECH
open science | open minds

InTech Europe

University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China

Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

INTECHOPEN

© 2012 The Author(s). Licensee IntechOpen. This is an open access article distributed under the terms of the [Creative Commons Attribution 3.0 License](https://creativecommons.org/licenses/by/3.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

IntechOpen

IntechOpen