Task Allocation Algorithms for 2D Torus Architecture

Lukasz Jakimczuk, Wojciech Kmiecik, Iwona Pozniak-Koszalka, and Andrzej Kasprzak
Department of Systems and Computer Networks
Wroclaw University of Technology
Wroclaw, Poland
e-mail: 170922@student.pwr.wroc.pl, {wojciech.kmiecik, iwona.pozniak-koszalka, andrzej.kasprzak}@pwr.wroc.pl

***Abstract* — Efficient allocation of computers to incoming tasks is crucial for achieving high performance in modern networks. A good allocation algorithm should identify available computers with minimum overhead and allocate incoming tasks in as short period of time as possible. This paper concerns allocation problem for torus-structured system. The new allocation mechanism, called Improved Tree Allocation for Torus (ITAT), based on tree architecture, has been proposed. ITAT-algorithm was compared with other known allocation algorithms on the basis of simulation experiments made with the designed and implemented experimentation system. The obtained results justify a conclusion that the created allocation algorithm seems to be very promising.**

Keywords-torus; allocation; algorithm; experimentation system; efectiveness

I. INTRODUCTION

Multicomputer systems, consisting of many processing elements connected through a high speed network, have become widespread in engineering and scientific applications [1]. Such networks are intended to deal with tasks which cannot be handled by single computer. Two-dimensional (2D) torus is one of the interconnection topologies developed for mentioned system [2]. For each topology including 2D torus, predefined allocation algorithms exist. In this paper, contiguous processor allocator for torus structured network is considered (Fig. 1).

The requirement here is to allocate incoming jobs to free subtorus of appropriate size in 2D torus connected system. The allocation scheme should provide maximal resource utilization what is done by minimizing any kind of fragmentation [3]. Allocation algorithm must be fast, deliver low overhead and be able to support systems with thousands of nodes. A critical attribute of all mechanisms is ability to find available subtoruses for incoming requests, if they exist, what is called subtorus recognition ability. An allocation algorithm has complete subtorus recognition ability when it can always find a free subtorus (if one is available) for an incoming job [4].

In this paper, recognition-complete allocation scheme based on non-binary tree called Improved Tree Allocation (ITAT) is presented. It was designed with intent of maximize the utilization.

The rest of the paper is organized as follows. Section II presents definitions and notations used throughout the paper. The existing job allocation mechanisms and accessory algorithms are reviewed in Section III. Section IV describes our novel scheme in detail. In Section V experimentation system is shortly presented. Within Section VI properties of the created algorithm are analysed and compared with other well-known algorithms. Future work and conclusion are finally included in Section VII.

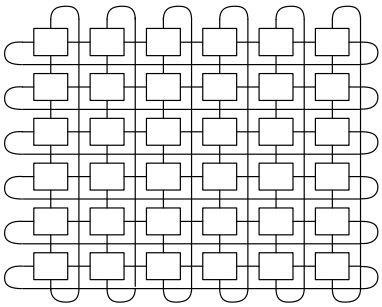

Figure 1. An example of 2D torus.

II. NOMENCLATURE

We use the classic notation presented, e.g., in [5][6][7]:

A 2D torus topology, denoted by *T(w,h)*, consists of $w \times h$ nodes arranged in a $w \times h$ 2D grid. The node in column c and row r is identified by address $<c,r>$ where $0 \le c < w$ and $0 \le r < h$. A node $<c,r>$ is connected by direct communication channel to its neighbouring nodes $<c\pm1,r>$ and $<c,r\pm1>$. Thus each node has four neighbouring nodes.

A 2D subtorus *S(p,q)* in the torus *T(w,h)* is a subgrid *T(p,q)* such that $1 \le p \le w$ and $1 \le q \le h$. A job requesting a subtorus $p \times q$ is denoted by *J(p,q)*. A subtorus *S* is identified by its base (lower left node) and end (upper right end) and is denoted as $S[<x_b,y_b> <x_e,y_e>]$. In contrast to the 2D-mesh topology, in torus x_b can be greater than x_e, and y_b can be greater than y_e. However, the base still remains as lower left corner with end on the upper right node.

A busy node is a node which has been allocated to a job. A busy subtorus β is a subtorus, where all of its nodes have been allocated to jobs.

A free node is a node which is not allocated to any job. A subtorus is free when all of its nodes have not been allocated to jobs.

A busy array of a torus *T(w,h)* is a bit map *B[w,h]*, in which element *B[c,r]* has a value 1 or 0 if node $<c,r>$ is busy or free, respectively.

A busy list is a set of all busy subtoruses in the system. Similarly, a free list is a set of all free subtoruses available.

 ISBN: 978-1-61208-245-5

The coverage of a busy subtorus β with respect to a job J is denoted by $\zeta_{\beta,J}$ and it is a set of processors such that use of any node in $\zeta_{\beta,J}$ as the base of free subtorus for the allocation of J will cause the job J to be overlapped with β. The coverage set with respect to J is denoted by C_J and it is the set of the coverages of all busy subtoruses.

A base block with respect to a job J is a subtorus whose nodes can be used as base for free subtoruses to allocate job J. A set of disjoint base blocks is called the base set.

External fragmentation is the ratio of the number of free processors to the total number of processors in the torus, when the allocation of incoming task fails but there is sufficient number of free processors.

The given definitions are illustrated in example of a torus *T(6,6)* with respect to *J(2,3)* and *J(2,2)* (see Fig. 2).

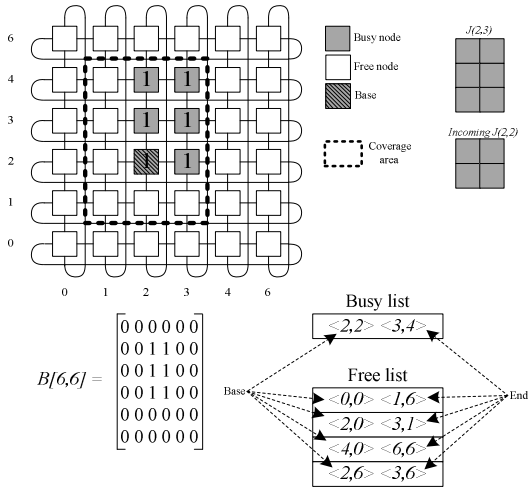

Figure 2. Busy and free nodes, coverage area, busy array and free list for a torus *T(6,6)*.

III. KNOWN ALLOCATION ALGORITHMS FOR TORUS ARCHITECTURE

The algorithms, based on busy list and busy array, create coverage area set [8] in one of the first steps. For *k-array 2-cube* four possible cases of task allocation can be distinguish and they are presented in Fig. 3.

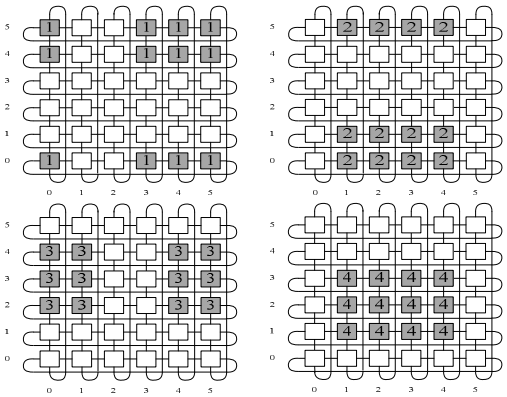

Figure 3. Four different cases of job *J(4,3)* allocation.

These cases are characterized by: 1) $x_b > x_e$ and $y_b > y_e$,; 2) $x_b \leq x_e$ and $y_b > y_e$,; 3) $x_b > x_e$ and $y_b \leq y_e$; 4) regular case known from 2D-mesh networks.

For every presented instance, coverage needs to be determined in a different way. For a given $\beta=[<x_b,y_b> <x_e,y_e>]$, its coverage with respect to $J(p,q)$ is $\zeta_{\beta,J}=[<x_1,y_1> <x_2,y_2>]$, where x_1, y_1, x_2, y_2 are determined according to the Construction of Coverage algorithm described in [1].

IBMAT Algorithm. First existing allocation algorithm is Improved Bit Map Allocation for Torus [9]. The general idea of the IBMAT is based on the approach used in IFF algorithm [4]. With respect to an incoming job, the busy array is scanned to create a coverage array C_T in the form of bit map. Each coverage $\zeta_{\beta,J}$ is divided into three regions: job coverage, left coverage and bottom coverage, presented in Fig. 4. In the worst case, two inspections through a C_T are required:

All rows from right to left, each row two times (determining the left coverage of a job).

All columns from top to bottom, each column two times (creating bottom coverage of a job).

The IBMAT is recognition complete by manipulating the job orientation. If for a given $J(p,q)$ the allocation fails and $p \neq q$, the scheme will change the orientation of the job and then $J(q,p)$ possibility is checked. When both attempts fail, the allocation of the job fails.

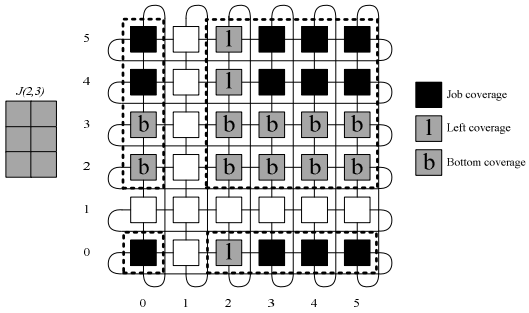

Figure 4. Coverage of job *J(4,3)* with respect to incoming job *J(2,3)*.

IBLAT Algorithm. Second existing allocation algorithm is Improved Busy List Allocation for Torus [10]. The IBLAT is based on the strategy employed in the IAS scheme [11]. For an incoming job $J(p,q)$, the IBLAT scans a busy list and creates coverage set C_T which is also in a list form. Both busy and coverage lists contains coordinates of each β and $\zeta_{\beta,J}$ respectively. When C_J is created, each node is tested for membership in C_J , what is done by inspecting the whole C_J for every node. Node which is not in C_J can be a base for given job, in the other case, the algorithm checks another node. The IBLAT scheme is recognition complete.

IRAT Algorithm. Third existing mechanism, based on randomness, is called Improved Random Algorithm. It can only pick random node from system and check if it can become base for an incoming request. If node is available as base job is allocated, otherwise scheme will change orientation of job J(p,q) and J(q,p) possibility is checked. When both attempts fail, the allocation of the job fails [12].

 ISBN: 978-1-61208-245-5

IV. IMPROVED TREE ALLOCATION FOR TORUS ALGORITHM

The task allocation algorithm proposed in this paper has complete subtorus recognition ability. The allocation scheme is particularly attractive for large systems, what is confirmed in experimentation section of this paper.

ITAT achieves recognition completeness by manipulating the orientation of the subtorus request. In allocation a job $J(p,q)$, the scheme first tries to allocate the task using the given orientation $p \times q$. If allocation fails, the algorithm creates a new request $J(q,p)$ by rotating the original orientation and tries to allocate rotated request. If this attempt also fails, the allocation of the job also fails.

The following definitions are introduced:

A busy tree, denoted by $T(n)$, incorporates n nodes including root, free and busy leaves and base nodes.

A root of the tree is the node with address <0,0>.

Subtree, denoted by $S_T(x)$, is a tree incorporates x nodes including only base nodes of tree $T(n)$ such that $0 < x < n$.

Free leave, denoted by $L_f(c,r)$, is the node with address $<c,r>$ such that it is not base of any existing subtoruses and it is not allocated to a job.

Busy leave, denoted by $L_b(c,r)$, is the node with address $<c,r>$ such that it is not base of any existing subtoruses but it is allocated to a job.

In Fig. 5(a), the introduced notions are illustrated for an exemplary torus $T(6,6)$.

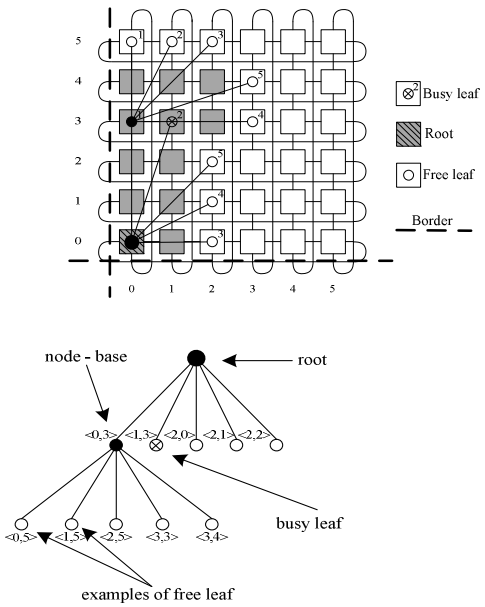

Figure 5(a). Illustration of the introduced notions for torus $T(6,6)$.

Allocation scheme always starts from node with address $<0,0>$ which is a root of busy tree. The root is a constant element of tree and exists even if it is not allocated as base to any job. With first allocated job, tree evolution process starts, that lasts until there are no more requests to allocate. Tree evolution process is divided into two main sections.

Base of every allocated job generates children nodes (leaves, shown in Fig. 5(a)) as follows:

Add adjacent nodes, from left to right, along top of the busy subtorus.

Add adjacent nodes, from bottom to top, along the right side of the busy subtorus.

After that, busy tree is recursively updated and every leaf receives its status – free or busy. Free leaves are potentially base nodes and hence are directly under consideration. Tree is searched level by level from left to right, start with root level. First free leaf that meets the requirement is returned as base for given job. Search scheme is presented in Fig. 5(b).

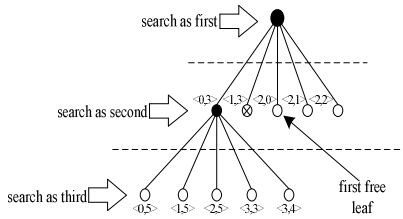

Figure 5(b). Search scheme for busy tree $T(11)$.

The search scheme, with respect to children creation mechanism, provides good fit of the incoming task to the existing configuration of torus, marked as *border* in Fig. 6(b).

Every allocated job, after elapse of its duration time is deallocated. It means that base node and its leaves are deleted from busy tree. Children that represent base are reattached to other base from current level or level up as follows:

If any base exists on the left side of deleted base descendants are attached to the base on the left side.

If none base does not exist on the left side of deleted base but does exist on the right side descendants are attached to the base on the right side.

If none base does not exist in current level descendants are attached to the parent base of deleted one.

V. EXPERIMENTATION SYSTEM

Experimentation system was developed in C++.

Input parameters:

The following task allocation problem parameters are taken into consideration:

P_1: the number of jobs in the queue (important data for static case of allocation).

P_2: the range of uniformly distributed pseudorandom sizes of each job in the queue (range of p and q).

P_3: the range of uniformly distributed pseudorandom numbers for execution time of each job in the queue.

P_4: the size of torus $T(w,h)$ i.e., the values of w and h.

Output parameters:

The following indices of performance (measures of efficiency, criteria) are treated as system outputs:

 ISBN: 978-1-61208-245-5

Simulation time t_s [ms]: defined as total time of simulation. It is the time needed to allocate and process all given jobs. This criterion is being analysed during static experiments.

Effectiveness E [%]: defined as percentage of process jobs with respect to all given jobs. This criterion is being analysed during static experiments. Effectiveness is also measured in specific period of time for dynamic experiments.

Unreliability U [%]: defined as complement of effectiveness in specific period of time. Unreliability can be calculated using eq. 1.

$$U = 1 - E \quad (1)$$

This criterion is not being analysed itself due to the fact it is calculated based on knowledge of effectiveness. It is important during dynamic experiments.

Remark: The series of simulation experiments were carried out on the Intel Pentium i5 machine with 4 GB of RAM memory.

VI. INVESTIGATION

A. Static Allocation – Experiment Design

First experiment was focused on comparing total simulation times t_s and effectiveness E for the considered algorithms. List of tasks was generated according to range of P_2 and P_3 presented in Table I.

TABLE I. RANGE OF INPUT PARAMETERS

Parameter	Range
Job size	2÷7
Job time [s]	5÷20

Parameter P_4 was equal: 25×25, 50×50, 75×75, and 100×100. For each system, ten measurements were made on the basis of which the average result for each system was calculated. Each algorithm had to allocate respectively 25, 50, 75, 100, 150, 200, 300, and 400 tasks.

B. Static Allocation – Results

The averaged results, concerning the total simulation time t_s and the effectiveness E are presented in Fig. 6 and Fig. 7.

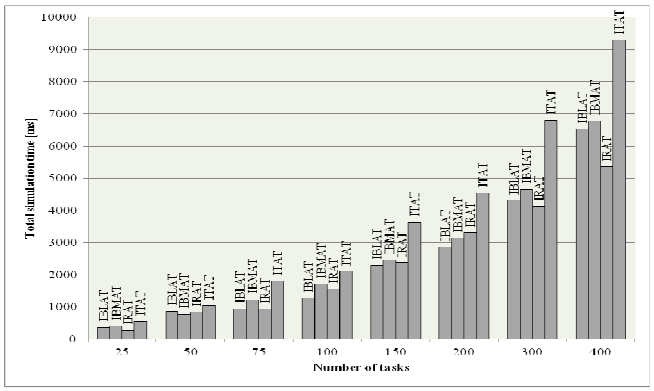

Figure 6. Average total simulation time t_s.

It may be observed, that the best algorithm, with respect to t_s is *IRAT* which is not reliable because it is based on randomness. Thus it is able to get through the whole list of tasks in short period of time but a substantial part of them will not be allocated and processed (Fig. 7). Due to the fact, that the best results were achieved for *IBLAT* and *ITAT* algorithms.

More complex structure of system does have noticeable impact on allocation time. Although differences are not significant and can be consider as neglected due to simulation error – experiments were done in the multitasking operating system that can cause measurement errors.

Testimony that the Improved Random Allocation Algorithm is not reliable is shown in Fig. 7. The *IRAT* works faster than the other mechanisms but a substantial part of tasks is not allocated and processed. Due to this fact the effective results were achieved for *IBLAT*, *ITAT* and *IBMAT* algorithms.

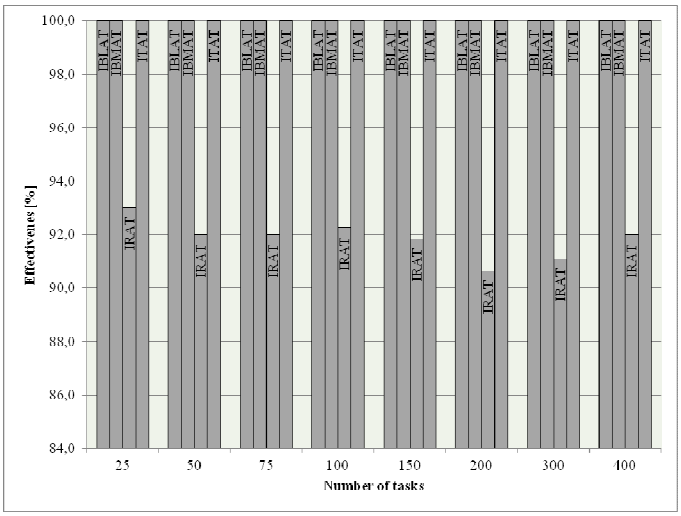

Figure 7. Average effectiveness *E[%]*.

C. Dynamic Allocation – Experiment Design

The second complex experiment was focused on comparing the total number of processed jobs N and unreliability U. Two cases were considered depending on the two sets of input parameters.

In the Case 1, incoming tasks were created according to the ranges of P_2 and P_3 parameters presented in Table II. Input parameter P_4 was equal: 25×25, 50×50, 75×75, and 100×100.

TABLE II. CASE 1: RANGE OF INPUT PARAMETERS P_2 AND P_3

Parameter	Range
Job size	2÷7
Job time [s]	10÷30

In the Case 2, incoming tasks were created according to range of P_2 and P_3 parameters presented in Table III. Input parameter P_4 was equal: 100×100, 200×200, and 300×300.

TABLE III. CASE 2: RANGE OF INPUT PARAMETERS P_2 AND P_3

Parameter	Range
Job size	20÷40
Job time [s]	50÷100

For the both sets of parameters and for each considered torus 10 measurements were performed on the basis of which the average result for each system was calculated. Each algorithm had to allocate incoming tasks during respectively 10, 20, 30, 40, 50, and 60 seconds.

D. Dynamic Allocation - Results

The averaged results of the experiments for both cases are presented in Fig. 8 and 9.

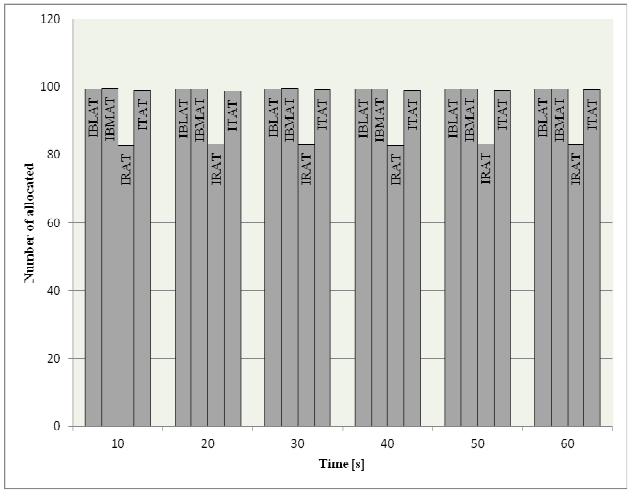

Figure 8. Average effectiveness *E* for Case 1

As expected, randomness has noticeable impact on the effectiveness and thus also on number of allocated tasks and unreliability. It allows processing large number of jobs (tasks) but at the cost of high unreliability what is a result of this, that system gets more tasks than it can be processed. It is important to know, that quality of algorithms cannot be determined solely by the number of allocated and processed tasks.

It can be said that effectiveness of algorithms for case 1 is comparable, or even the same - differences are barely noticeable. The effectiveness is not an ideal parameter because it is based on the speed of processing tasks within the system. Simplify algorithm, i.e. IRAT are faster, thus whole process of allocation for one task does not take a lot time. Because of that the number of processed tasks can be higher when compared with other algorithms, however in comparison with the all given jobs results are worse.

The probability of allocation success decreases with every processed job and so unreliability decreases.

It does not mean that every algorithm with ability to allocate large number of jobs, even at the cost of low effectiveness, is efficient. The efficient allocation technique needs to have balanced parameters.

Every incoming task can be described by the probability of its allocation. As the size of torus grows, the probability

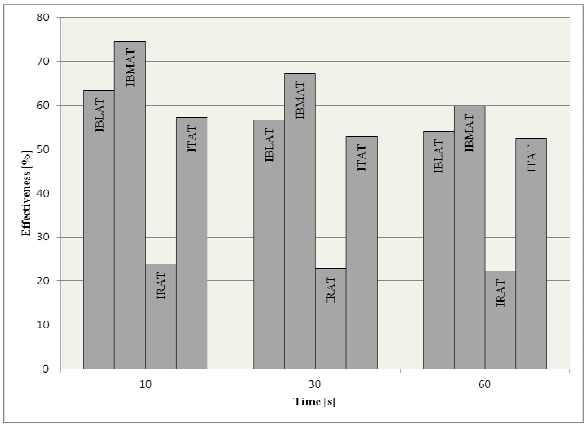

Figure 9. Average effectiveness *E* for Case 2.

of allocation grows as well. The same can be said about effectiveness and total allocation time parameters that achieve higher values for larger systems. The reverse situation can be observed for duration time and size of incoming tasks. As the size and duration time of tasks falls, probability of allocation, effectiveness and total allocation time grows.

VII. CONCLUSION AND FUTURE WORK

In this paper the Improved Tree Allocation for Torus (*ITAT*) was proposed. Based on experiments it may be observed that each allocation algorithm has its advantages and disadvantages. The proposed algorithm is particularly attractive for large toruses and large tasks. *ITAT* uses non-binary tree to identify free subtoruses which can be allocated to an incoming request.

Busy tree as the algorithm is a very complex structure that is why for smaller systems *ITAT* work worse than other schemes. It is important to know that modern networks with torus topology do have hundreds of nodes. Thus *ITAT*'s quality is comparable with other existing schemes.

The future work includes plans to extend *ITAT* by implementing more intelligent scheme for leaves creation process. It is necessary in order to maximize utilization of nodes. Other process that needs to be reviewed is recursively update of busy tree that has negative impact on total simulation time and allocation time. We also intend to combine leaves update scheme with leaves creation process.

REFERENCES

[1] T. Srinivasan,, J. Seshadri, A. Chandrasekhar, and J. B. Siddhart, "A minimal fragmentation algorithm for task allocation in mesh-connected multicomputers," Report in Department of Computer Science, College of Engineering, Sriperumbudur, India, 2004.

 ISBN: 978-1-61208-245-5

[2] W. J. Dally and C. L. Seitz, “The torus routing chip,” Journal of Distributed Computing, vol. 1, no. 4, 1986, pp. 187-196.

[3] W. Kmiecik, L. Koszalka, I. Pozniak-Koszalka, and A. Kasprzak, “Evaluation scheme of tasks allocation with metaheuristic algorithms in mesh connected processors,” Proceedings of 21st International Conference on Systems Engineering (ICSEng), IEEE CPS, 2011, pp. 241-246.

[4] S. Yoo and R. Das, “An efficient task allocation scheme for 2D mesh architectures,” IEEE Transaction on Computers, vol. 8, no. 9, 2002, pp. 934-938.

[5] T. Liu, W. Huang, F. Lombardi, and L. N. Bhuyan, “A submesh allocation scheme for mesh connected multiprocessor systems,” Proceedings of International Conference on Parallel Processing, vol. II, 1995, pp. 193-200.

[6] I. Pozniak-Koszalka, L. Koszalka, and M. Kubiak, „Allocation algorithm for mesh structured networks” Proceedings of IARIA International Conference on Systems, 2006, pp. 24-30.

[7] D. M. Zydek and H. Selvaraj, “Implementation of processor allocation schemes for mesh-based chip multiprocessors,” Journal of Microprocessors and Microsystems, ISSN 0141-9331, vol. 34, no. 1, 2011, pp. 39-48.

[8] J. Ding, and L. N. Bhuyan, L. N., “An adaptive submesh allocation strategy for two-dimensional mesh connected systems" Proceedings of International Conference on Parallel Processing, vol. II, 1993, pp. 193-200.

[9] Y. Zhu, “Efficient processor allocation strategies for mesh-connected parallel computers,” Journal of Parallel and Distributed Computing, vol. 16, no. 4, 1992, pp. 328-337.

[10] D. M. Zydek, “Processor allocator for chip multiprocessors,” PhD. Dissertation, University of Nevada, Las Vegas, USA, 2010.

[11] D. M. Zydek and H. Selvaraj, “Fast and efficient processors allocation algorithm for torus-based chip multiprocessors,” Journal of Computers & Electrical Engineering, ISSN 0045-7906, October 2010.

[12] T. Baba, Y. Iwamoto, and T. Yoshinaga, “A network-topology independent task allocation strategy for parallel computers,” Proceedings of ACM/IEEE conference on Supercomputing, IEEE Computer Society Press Los Alamitos, CA, USA, 1990, pp. 878-887.

 ISBN: 978-1-61208-245-5