Task Allocation within Mesh Networks: Influence of Architecture and Algorithms

Aleksandra Postawka and Iwona Poźniak Koszałka

1 Introduction

More and more computational power is needed to solve various problems. Supercomputers constantly advance in processing speed, but expensive CPU time may be wasted by inefficient Processor Allocator (PA). In order to increase the efficiency of task allocation process - many ideas, approaches, and methods in constructing task allocation algorithms are proposed in literature, including the algorithms based on First Fit [1–4] and its modifications based on Busy List [1, 2], or based on stack ideas so-called Stack Based Algorithm - family [1, 2, 5–8]. These algorithms are dedicated to the allocation process within 2D rectangle meshes. We focus not only on rectangle but also on the two other mesh processor structures (architectures) - cylinder and torus. For such scenarios a few algorithms are proposed in literature [3, 4, 9–11]. In the paper the own allocation algorithms (being some modifications of the known ideas) called Generalized Recognition Complete First Fit (GRCFF) and Generalized Stack-Based Allocation (GSBA) are proposed. The properties of these algorithms have been checked on the basis of the results of simulation experiments which were carried out with the designed own experimentation system. The efficiency of the algorithms was tested for three mesh structures consisting of the same number of the processors (nodes in mesh). To measure the algorithms' efficiency we introduced four indices of performance. The research was undertaken on task allocation algorithms with static task queue (the whole set of tasks is known before starting the allocation process). In such a case the tasks may be sorted without worrying about the starvation problem. Thus, the influence of sorting has been also taken into consideration.

The rest of the paper is organized as follows. The considered allocation problem is formulated in Section 2. In Section 3 the proposed algorithms for solving the problem are presented. Section 4 contains the description of the experimentation system and the results of investigations. The conclusions appear in Section 5.

2 Problem Formulation

Terminology. *Node* is a single *processor* and it is identified by (*x*, *y*) coordinates. Rows and columns are counted starting from the bottom-left corner - (0, 0) coordinate. A *mesh* M(W, H) is a composition of nodes. The three considered 2D structures of the mesh are defined in the following way:

- a rectangular mesh $M(W,H) = \{(x,y) : x,y \in \mathbb{Z} \wedge x \in [0,W) \wedge y \in [0,H)\}$;
- a cylinder mesh $M(W,H) = \{(x,y) : x,y \in \mathbb{Z} \wedge x \in (-W, W) \wedge y \in [0,H)\}$, where $x = (x+W) \, mod \, W$;
- a torus mesh $M(W,H) = \{(x,y) : x,y \in \mathbb{Z} \wedge x \in (-W,W) \wedge y \in (-H,H)\}$, where $x = (x+W) \, mod \, W \wedge y = (y+H) \, mod \, H$.

A *subgrid* $S_M(<i,j>, <w,h>) = \{(x,y) : x,y \in \mathbb{Z} \wedge x \in [i,i+w) \wedge y \in [j,j+h)\}$, where $i+w \in (0,W] \wedge j+h \in (0,H]$ is a rectangular grid which completely belongs to *M*(*W*, *H*). A node $< i, j >$ indicates bottom-left corner and $< w, h >$ denotes width and height of the subgrid. A *free subgrid* is a subgrid, where all the processors are currently free and an *allocated subgrid* is a subgrid, where all the processors are currently allocated by the tasks. A rectangular *task* *J*(*w*, *h*, *t*), where *w* denotes width, *h* denotes height, *t* denotes time needed to accomplish the task, can be allocated at the node-point (*i*, *j*),

A. Postawka (✉)
Faculty of Electronics Computer Architecture Group, Wroclaw University of Technology, Wroclaw, Poland
e-mail: aleksandra.postawka@pwr.wroc.pl

I.P. Koszałka
Faculty of Electronics, Department of Systems and Computer Networks, Wroclaw University of Technology, Wroclaw, Poland
e-mail: iwona.pozniak-koszalka@pwr.wroc.pl

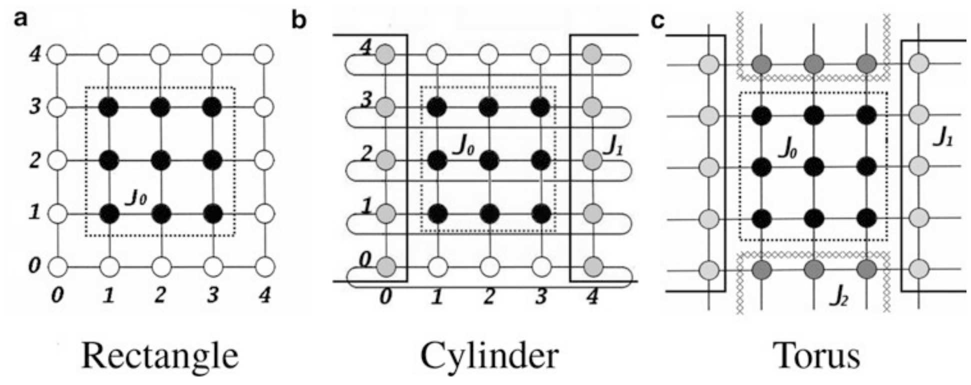

Fig. 1 Task allocation for different structures

causing that a submesh $S_M(< i, j > , < w, h >)$ is allocated. The task occupies a set of busy nodes completely until it is finished. Before allocation tasks are waiting in a queue. A *coverage* c_J of a busy subgrid with respect to waiting in queue task J is such a set of processors that allocation of J at any of these nodes would cause the task J to be overlapped with another already working task. The *coverage set* related to J is denoted by C_J and it is a set of the coverages of all busy subgrids. A *base subgrid* B_J for the task J is such a set of processors that J can be allocated at any of its nodes (without overlapping tasks already being executed). A *candidate area* CA_J for the task J is such a set of nodes which is considered to become the *base subgrid*. A *reject area* R_J for the task J is such a set of nodes that use of any of them as a *base subgrid* causes crossing the boundary of the mesh.

Example. Let us consider a queue of tasks $J_0(3, 3, 5)$, $J_1(2, 5, 7)$, $J_2(2, 3, 5)$. For the rectangle mesh, there may occur the situation shown in Fig. 1(a). Free nodes are located at the edges of the grid, but $J_1(2, 5, 7)$ cannot be allocated. For the cylinder structure, the possible allocation is shown in Fig. 1(b) - the next J_1 task can be allocated, but the third task in a queue not. A possible allocation for torus structure is shown in Fig. 1(c). The task $J_0(3, 3, 5)$ occupies the subgrid $S_M(< 1, 1 > , < 3, 3 >)$. The J_1 task is allocated on the subgrid $S_M(< 4, 0 > , < 2, 5 >)$. During the allocation the pointer for OY axis exceeds its maximum value of 4, thus it is changed to the value within the range by performing a modulo operation. The last task to allocate $J_2(2, 3, 5)$ can be placed on the submesh $S_M(< 1, 4 > , < 3, 2 >)$.

Problem statement. The considered allocation problem can be formulated as follows:

For given: (i) a set of tasks $J_0, J_1, J_2 \ldots, J_N$ with known sizes and execution times (w'_k, h'_k, t'_k), $k = 0,1 \ldots N$; (ii) a number of $(W \times H)$ nodes arranged in a way defined by the chosen structure $M(W, H)$.
To find: an allocation of tasks within the mesh.
Such that: the considered indices of performance reach the best value.
Subject to the constraint: the tasks should not overlap each other.

The indices of performance. To evaluate the quality of the algorithms the following indices of performance are introduced:

Allocation time t_a is the time needed to find a free subgrid and allocate the given task J

$$t_a = t_s - t_q,$$

where t_s - the time of successful allocation, t_q - the time of taking the task J from the queue.

Processing time t_p is the total time needed to execute all the given tasks.

External fragmentation F_e is the ratio of the number of free processors (N_f) to the total number of processors in the system (N_a):

$$F_e = \frac{N_f}{N_a} \cdot 100\%.$$

Latency L is the number of mesh time ticks between adding the task to the queue (for the static queue $t_0 = 0$) and its successful allocation (t_s):

$$L = t_s - t_0 = t_s.$$

3 Algorithms

Generalized Recognition Complete First Fit (GRCFF). The algorithm GRCFF is based on FF algorithm, but it is recognition complete in that both node by node verification and two job orientations are taken into consideration [1, 9]. The algorithm works in the following way: firstly C_J and R_J for the new task J are prepared. Next, all the nodes are sequentially searched in order to find the first one which does not belong to C_J or R_J (thus it can be used for J allocation). If this operation fails and J's width is different than its height, then J is rotated and algorithm attempts to allocate it again. If allocation is unsuccessful again, the algorithm waits (until one of the tasks being executed will

Table 1 GRCFF algorithm

```
GRCFF
set nodes from reject area as busy
rotated = false
do
    for each task $w$ from the working tasks vector $w_J$
        calculate bottom-left coordinates for $c_J$ ($x_c = x_w - h_J + 1$, $y_c = y_w - w_J + 1$)
        calculate width and hight for $c_J$ ($w_c = w_J + w_w - 1$, $h_c = h_J + h_w - 1$)
        for $x_c \leq i < x_c + w_c$
            for $y_c \leq j < y_c + h_c$
                $x = (i + W) \mathrm{mod} W$
                $y = (j + H) \mathrm{mod} H$
                set node $(x, y)$ as busy
    for $0 \leq i < W$
        for $0 \leq j < H$
            if node $(i, j)$ is not busy
                allocate task $J$ in subgrid $S_M(<i,j><w_J, h_J>)$
                exit(success)
    if rotated
        exit(fail)
    else
        rotate task $J$
        rotated = true
while rotated
```

be completed and new free space on mesh occurs). The modification refers to wrapping coverage areas and defining R_J. The abbreviated description of GRCFF algorithm is given in Table 1. In order to explain how the algorithm works - let us consider the task $J(w, h, t)$ and $M(W, H)$ for different architectures. In the case of rectangle, to allocate J, there are the coordinates $0 \leqslant i \leqslant W - w$ and $0 \leqslant j \leqslant H - h$ considered (not belonging to R_J). The R_J example for task $J(1, 2, 5)$ (Fig. 2(a)) and rectangle mesh is shown in Fig. 2(b). In the case of cylinder, tasks can be allocated at one of the edges as well. The tasks are wrapped and placed on the opposite edge, thus R_J is calculated only for OY axis (Fig. 2(c)). In order to allocate J within cylinder mesh, nodes with the coordinates $0 \leqslant i \leqslant W - 1$ and $0 \leqslant j \leqslant H - h$ are considered. For torus there are no borders and the first task can be placed at any node. The special feature of torus is the lack of R_J (Fig. 2(d)). To allocate J the nodes with coordinates $0 \leqslant i \leqslant W - 1$ and $0 \leqslant j \leqslant H - 1$ are considered. Despite the fact that in torus there is no R_J, additional c_J's have to be considered, which are inessential in the rectangle mesh (Fig. 2(e) and 2(f)). Not only the tasks are wrapped around the edges, but C_J as well.

Generalized Stack-Based Allocation (GSBA). This algorithm uses elementary areas subtraction operation. In the first part of its performance, it follows SBA in action (which is recognition complete [1]). Firstly, C_J for the current task J is calculated. Together with CA_J (at the beginning it is the entire mesh without R_J) it is placed on the top of the stack. The c_J's are subtracted from CA_J's and the results (new CA_J's with reduced C_J's) are placed on the stack again. The algorithm works until the stack is empty or CA_J without c_J's to subtract is found. If the task J cannot be allocated, then it is rotated and the algorithm attempts to allocate it again. If allocation fails, algorithm waits. GSBA is the alteration of Improved Stack Based Algorithm (ISBA) [1]. The modifications refer to (i) C_J and initial CA_J determination, (ii) area subtraction operation and (iii) test for areas overlapping. GSBA in action is shown in Table 2. The tasks can be allocated on the borders as well, so they are wrapped around two edges of the grid. It causes that C_J wraps together with them. Wrapping C_J causes that some coordinates are negative (see definition of S_M), thus it is complicated to determine how the two areas overlap. In the worst case CA_J inside the mesh can be overlapped with c_J in four places (see Fig. 3(a)). The problem was solved by changing the coordinate system. The bottom-left corner of CA_J is moved to the point (0,0) as in Fig. 3(b) and the coordinates are recalculated properly for c_J too. The attention has to be paid to the condition of wrapped task's coordinates signs. Since CA_J is anchored in the center of coordinate system it is easy to subtract areas once overlapped (now it can be done in the same way as for rectangle). If the x_c coordinate of c_J is negative and it fulfills the condition $x_c + W < w_c$ (where w_c is the CA_J's width), it is clear that for OX axis the areas are overlapped twice. The same should be checked for the OY axis.

4 Investigations

Experimentation system. The system consists of the following elements:

Controlled input: A - allocation algorithm {GRCFF, GSBA}; S - queue sorting {unsorted, time descending, time ascending, task size descending, task size ascending}.

Problem's parameters: P_1 - the number of tasks in the queue; P_2 - the range of values from which the task's dimensions will be chosen by random; P_3 - the range of execution time for each task in the queue; P_4 - the size ($W \times H$) of structured mesh.

Outputs: t_a-allocation time; t_p-processing time; F_e-external fragmentation; L-latency.

Experimentation system was written in *C++* and compiled by *g++ 4.1.2* with *-O3* optimization flag enabled. Simulations were performed on Intel(R) Xeon(R) 2.67 GHz processor. This research was supported in part by PL-Grid Infrastructure. For each simulation (single experiment) the tasks are randomly generated due to given parameters and inserted into the queue. The queue can be sorted in ascending or descending order by such properties as task execution time or task dimension. After all the tasks are executed, the calculated statistics are returned.

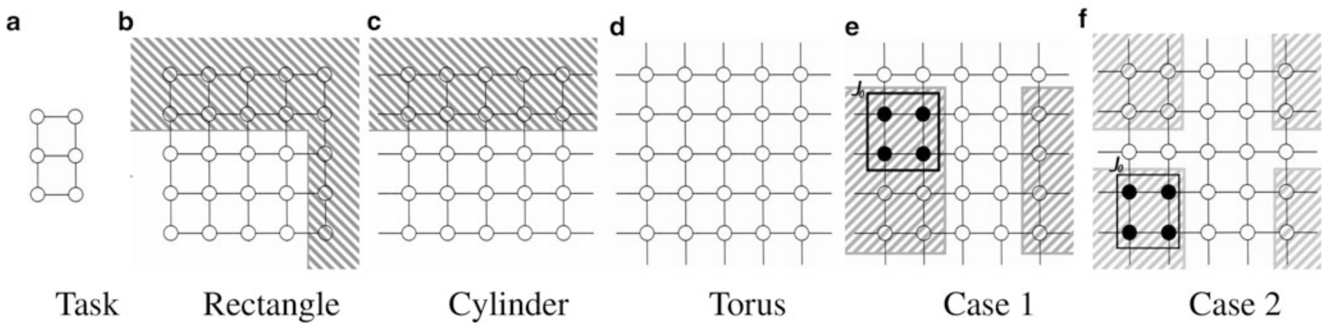

Fig. 2 Reject areas (b, c, d) for different architectures and torus coverage (e, f)

Table 2 GSBA algorithm

```
GSBA
rotated = false
do
  create initial candidate area $CA_J$
  for each task $w$ from the working tasks vector $w_J$
    calculate bottom-left coordinates for $c_J$ ($x_c = x_w - h_J + 1$, $y_c = y_w - w_J + 1$)
    calculate width and hight for $c_J$ ($w_c = w_J + w_w - 1$, $h_c = h_J + h_w - 1$)
    add $c_J$ to vector $vec_{cJ}$
  push onto the stack pair $(CA_J, vec_{cJ})$
  while stack is not empty
    pop from the stack pair $(CA_J, vec_{cJ})$
    for each $c_J$ in $vec_{cJ}$
      remove $c_J$ from $vec_{cJ}$
      if $c_J$ overlaps $CA_J$
        $vec_{CA}$ = (area subtraction) $CA_J - c_J$
        for each new $CA_J$ in $vec_{CA}$ =
          push onto the stack pair $(CA_J, vec_{cJ})$
    if $vec_{cJ}$ is empty
      allocate task $J$ in bottom-left node of $CA_J$
      exit(success)
    if rotated
      exit(fail)
    else
      rotate task $J$
      rotated = true
while rotated
```

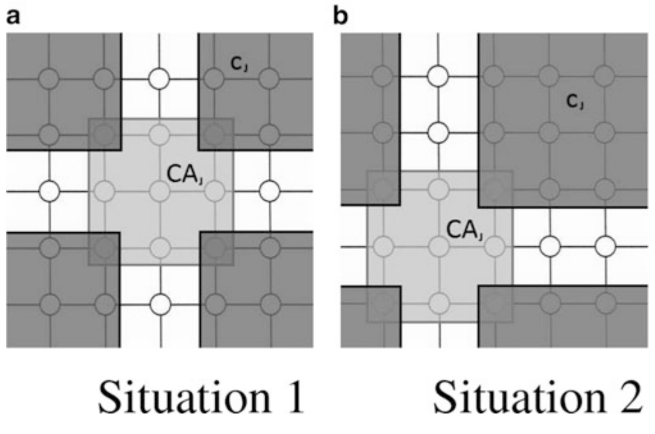

Fig. 3 Changing the coordinate system for torus architecture

Experiment 1. Influence of used architecture on average allocation time. *Experiment design:* $W = H = X$ and $P_1 = X^2$ P_2: $1 \div \lceil 0.4 \cdot X \rceil$

P_3: $1 \div 1000[ticks]$ P_4: X^2 for $X \in \{$ 100, 200, 300, 400, 500, 600, 700, 800$\}$

Results: Figures 4(a), 4(b) show the dependency of average t_a and grid size for each of the three investigated mesh architectures, for GRCFF and GSBA algorithms. GRCFF algorithm is $O(W \cdot H)$ what can be noticed in Fig. 4(a). Cylinder and torus architectures are more complex than rectangular one, but t_a are not significantly greater (for the same dimensions), e.g., for grid size of (800×800) t_a for cylinder and torus architecture is greater respectively by 17% and 27% than for rectangle structure. GSBA has almost the same t_a for all grid dimensions (Fig. 4(b)). When the number of tasks and their sizes increase proportionally to the growth of network size, t_a for GSBA is constant. The complexity of mesh architecture has an impact on t_a, but for each structure the dependencies are very similar to others. In line with expectations the average time needed to allocate the task in the cylinder architecture is longer than in the rectangular mesh and for torus allocation takes the most time. Removing each of the borders makes the subtraction operation more complex, because more special cases have to be taken into consideration. It can be observed (see Fig. 4(b)) that t_a in torus is nearly twice longer than in rectangular mesh. For example, for the biggest grid sizes t_a for cylinder and torus is greater by 36% and 91% than for rectangle architecture, respectively.

Experiment 2. Influence of tasks dimensions on the processing time. *Experiment design:* $P_1 = 10000$ P_2: $(1 \div 10), (10 \div 25), (15 \div 30), (20 \div 35), (25 \div 40), (30 \div 45)$ P_3: $1 \div 1000[ticks]$ P_4: 100×100

Results: It can be observed (Fig. 4(c), 4(d)) that for almost all investigated cases torus architecture achieves the shortest t_p. Moreover, for both GRCFF and GSBA it can be noticed that the greater tasks surface area, the more profitable is using more complex architectures - e.g., in the case of GSBA (Fig. 4(d)) and task sizes in the ranges $(10 \div 25)$, $(20 \div 35)$, $(30 \div 45)$ profit from the use of the torus instead

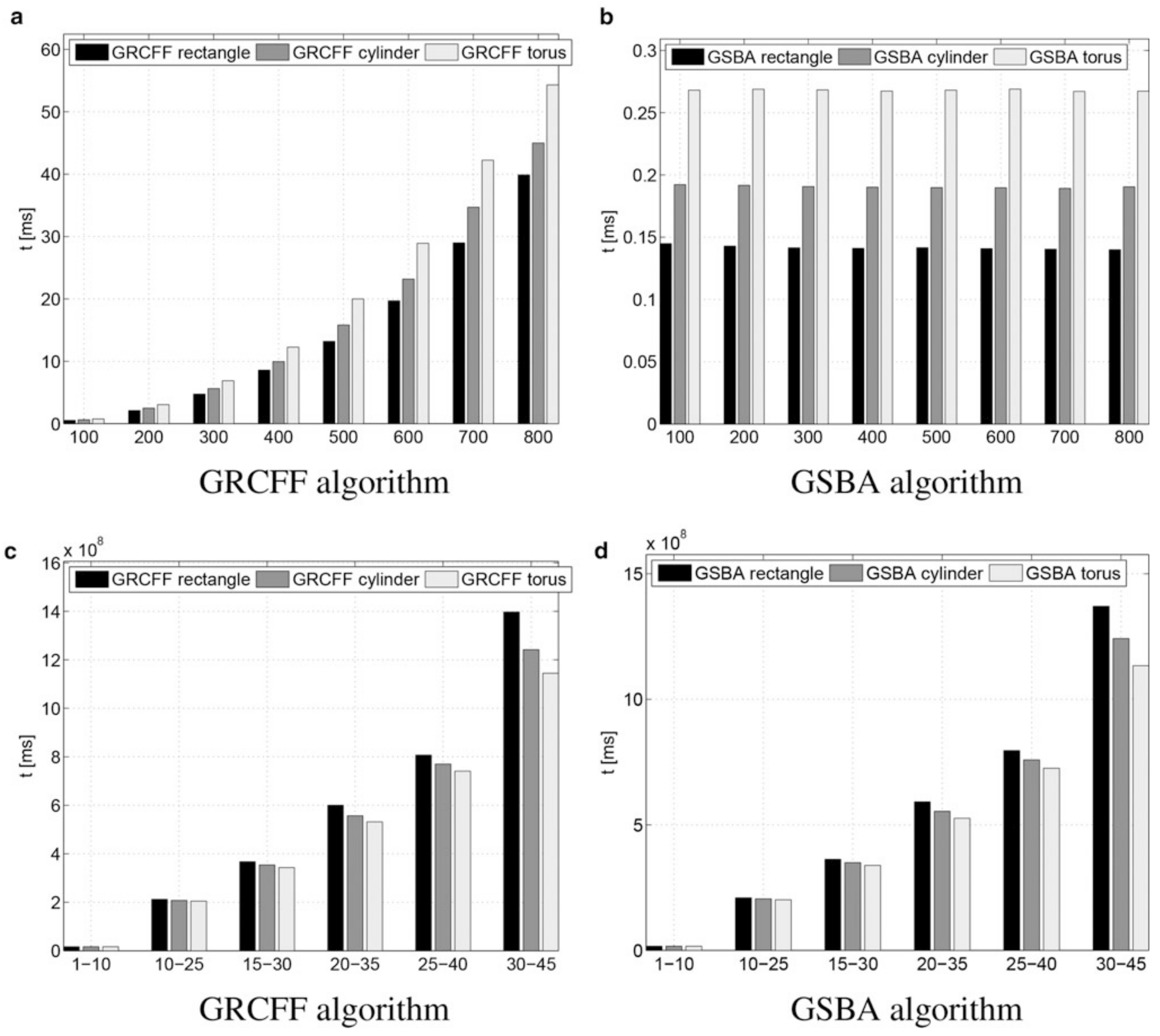

Fig. 4 Average tasks allocation time for different structures and different grid's side length (a, b) The dependency of processing time for tasks with different sizes - grid size 100 × 100 (c, d)

of a rectangular architecture is: 3.7%, 11.1% and 17.3%, respectively. For smaller tasks it is not so obvious and sometimes other architectures return shorter t_p than torus, e.g., for tasks from the range $1 \div 10$ the algorithm GRCFF achieved the shortest t_p for cylinder and the longest for rectangular structure - but when the algorithm GSBA is used (Fig. 4(c)), statistics are inverted. However, the differences are so small, that could be negligible (0.01%-0.2%).

Experiment 3. Examination of mesh architecture and sorting queue. *Experiment design:* P_1: 10000 P_2: $(5 \div 15) \times (5 \div 15)$, $(5 \div 30) \times (5 \div 30)$, $(15 \div 30) \times (15 \div 30)$ P_3: $1 \div 1000[ticks]$ P_4: 50x50

Results: For each mesh architecture and for each value of P_2 parameter five different simulations have been performed. Each of them used another sorting. The data set is either completely unsorted or sorted by task execution time and task surface area in ascending or descending order. In case 'unsorted' tasks are inserted into the queue directly from the random task generator. The obtained results are presented in Fig. 5. The characteristics of the results for F_e and t_p are very similar because of close relation between these parameters - the greater average number of unused nodes, the more time will take the execution of the same tasks. Therefore only diagrams for F_e have been presented (Fig. 5(a) and Fig. 5(b)). Tasks with the smallest

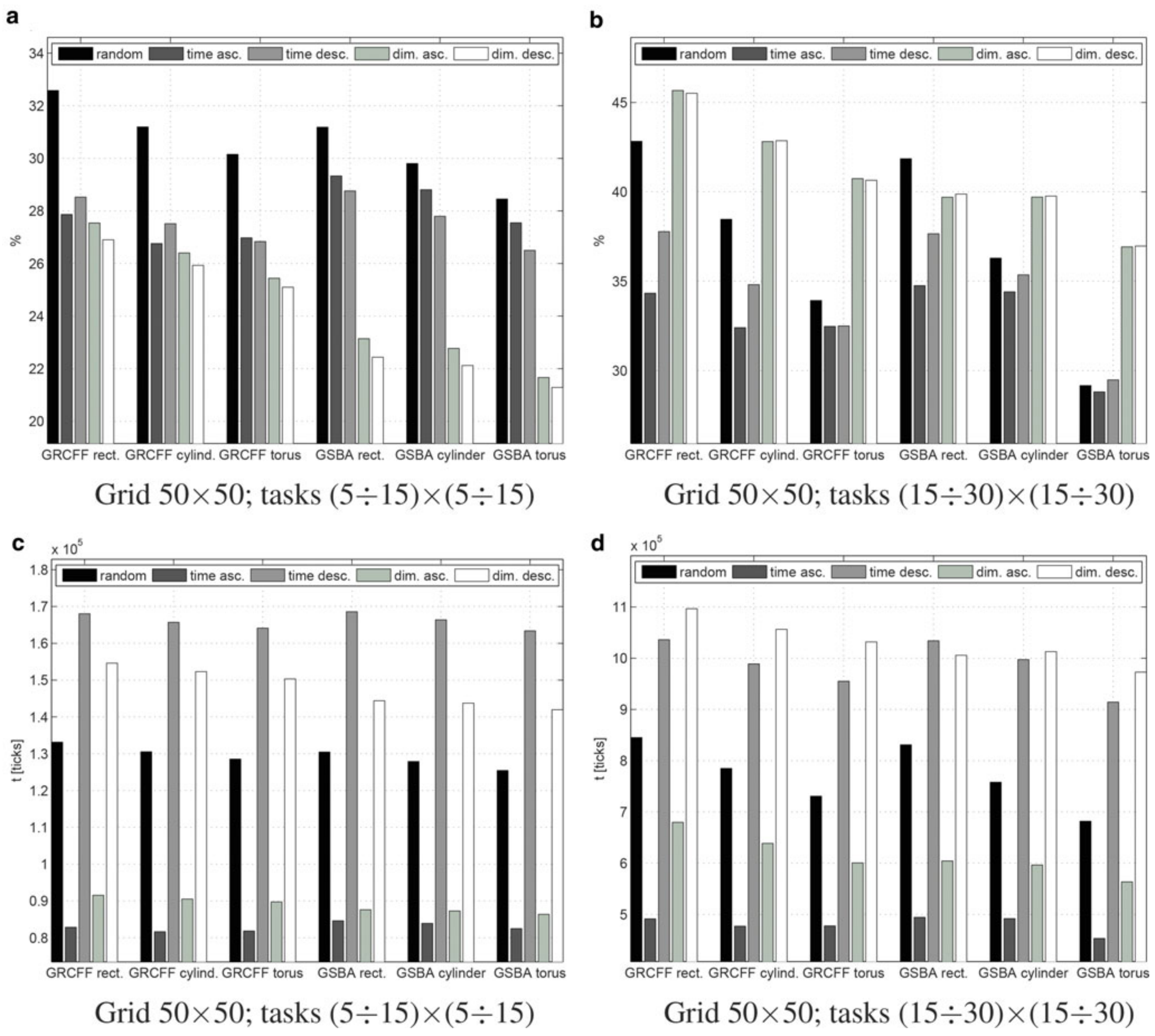

Fig. 5 External fragmentation (a, b) and latency (c, d)

surface area $(5 \div 15) \times (5 \div 15)$ are characterized by the least F_e (Fig. 5(a)). It is quite obvious because smaller tasks can fit better. As expected, torus shaped mesh usually achieves the best results, because it occurs that tasks, which would not be allocated in rectangular mesh, fit when wrapped. Looking at figures 5(a) and 5(b) it can be easily observed that sorting tasks can both improve and worsen achieved results. Sorting tasks by execution time in ascending order gives less F_e than for the random queue. Moreover, it can be noticed that for GRCFF algorithm it is better to sort tasks by time in ascending order, while for GSBA sometimes better results are obtained when sorting descending (Fig. 5 (a)). However, the greater tasks surface area, the more profitable it becomes for GSBA to sort ascending. For example, in the case of grid size (50×50) and torus architecture, F_e for sorting ascending in comparison to sorting descending (see Fig. 5) is higher by 1.1% for task sizes $(5 \div 15)$, lower by 0.1‰ for task sizes $(5 \div 30)$ and lower by 6.7‰ for task sizes $(15 \div 30)$ - see Fig. 5(b). Sorting tasks by their dimensions often makes the F_e greater and consequently gives longer t_p (Fig. 5(b)). In the case of sorting ascending at the beginning of such simulation tasks fit very well, but later there are only the tasks with the biggest surface area and the gaps between them cannot be filled. In the case of sorting descending large tasks at the beginning make F_e greater. In some special cases sorting tasks queue by dimensions helps, but if there is time available for sorting tasks it is better to sort them by execution time.

Completely different conclusions can be drawn from the observation of average L factor. The best results are achieved when tasks are sorted in ascending order, either by time or by surface area. In the first case, the average L for GSBA and tasks from $(5 \div 15)$ for torus in Fig. 5(c) is lower by 34%, and in the second case it is lower by 31% (in comparison to 'unsorted' queue). It is quite obvious that allocating the shortest tasks first and leaving the longest tasks at the end shortens the average waiting time. The same effect is gained when smaller tasks, which fit better, are placed earlier. Average is counted for many tasks with short latency and few tasks with greater value.

Sorting tasks by investigated parameters in descending order is never a good choice, because it extends the task's waiting time for its execution. In such cases the L value is greater by 30% for sorting by time and by 13% for sorting by tasks dimensions (considering GSBA in Fig. 5(c) and comparing with unsorted queue).

5 Conclusions

On the basis of the experiments it may be concluded that better performance was obtained for GSBA algorithm than in case of GRCFF. Although the allocation time was longer, better results were gained for processing time, external fragmentation and latency. As expected, in comparison to other investigated mesh architectures, in most cases torus structured mesh ensured achieving the best results.

Torus architecture achieves the best performance for the tasks with greater surface area. Because of wrapping tasks on the edges it occurs that there can be allocated task, which would not fit in other structures. Although there is no reject area in torus, more coverage areas have to be taken into consideration (these wrapped on the edges). This is the reason why the performance does not increase much. In the case of smaller tasks the processing time for torus architecture is not always shorter than for other structures, thus such network is more profitable for tasks requiring more processing nodes.

The usage of sorting algorithms can significantly improve the performance of the allocation algorithms. The values for processing time and external fragmentation are the least when tasks in the queue are sorted by time, in any order. In case of latency the lowest values are obtained for sorting by time or by task's surface area, both in ascending order. The average latency is very important because it indicates how long average system's user will have to wait for data. But the processing time is also significant. The only operation which improves all the indices of performance is sorting the task queue by time in ascending order. One of the advantages of GSBA algorithm is quite constant allocation time for different grid sizes. GRCFF algorithm can be considered only for small grids, because allocation time increases radically.

In the nearer future, the authors plan to expand the experimentation system following ideas proposed in [6] and to implement other allocation algorithms presented in [2].

Acknowledgement This work was supported by the statutory funds of the Faculty of Electronics, Wroclaw University of Technology, Wroclaw, Poland.

References

1. Zydek, D., Selvaraj, H.: Fast and efficient processor allocation algorithm for torus-based chip multiprocessors. Computers and Electrical Engineering 37, 91–105 (2011)
2. Majkowska, A., Zydek, D., Koszałka, L.: Task Allocation in Distributed Mesh-Connected Machine Learning System: Simplified Busy List Algorithm with Q-Learning Based Queuing. Advances in Intelligent Systems and Computing 226, Springer, 763–772 (2013)
3. Jakimczuk, Ł., Kmiecik, W., Poźniak-Koszałka, I.: Task Allocation Algorithms for 2D Torus Architecture. The Twelfth IARIA International Conference on Networks, 169–174 (2013)
4. Zhu, Y.: Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers. Journal of Parallel and Distributed Computing 16, 328337 (1992)
5. Yoo, B.S., Das, Ch.R.: A fast and efficient processor allocation scheme for mesh-connected multicomputers. IEEE Transactions on Computers 51, 46–60 (2002)
6. Koszałka, L., Kubiak, M., Poźniak-Koszałka, I.: Comparison of SBA - Family Task Allocation Algorithms for Mesh Structured Networks. LNCS, Springer 4331, 21–30 (2006)
7. Poźniak-Koszałka, I., Proma, W., Koszałka, L., Pol, M., Kasprzak, A.: Task Allocation in Mesh Structure: 2Side LeapFrog Algorithm and Q-learning Based Algorithm, LNCS, Springer 7336, 576–587 (2012)
8. Kaminski, R., Koszałka, L., Poźniak-Koszałka, I., Kasprzak, A.: Evaluation and Comparison of Task Allocation Algorithms for Mesh Networks. Proc. of 9th ICN, IEEE CPS, 104–108 (2010)
9. Geunmo, K., Hyunsoo, Y.: On Submesh Allocation for Mesh Multicomputers: A Best-Fit Allocation and a Virtual Submesh Allocation for Faulty Meshes. IEEE Transactions on Parallel and Distributed Systems 9, 1127–185 (1998)
10. Borowiec G., Postawka A., Koszaka L.: Static task allocation algorithms and influence of architecture on Mesh Structured Networks. Computer Systems Engineering PBW, 66–80 (2014)
11. Dally, W.J.: Performance analysis of k-ary n-cube interconnection networks. IEEE Transactions on Computers 39, 7127–785 (1990)