International Journal of Grid and High Performance Computing
Volume 15 • Issue 1

On Allocation Algorithms for Manycore Systems With Network on Chip

Abeer Shdefat, Al al-Bayt University, Jordan
Saad Bani-Mohammad, Al al-Bayt University, Jordan*
Ismail Ababneh, Al al-Bayt University, Jordan

ABSTRACT

Single-chip multicore processors and their network on chip interconnection mechanisms have received extensive interest since the early 2000s. The mesh topology is popular in networks on chip. A common issue in mesh is that it can result in high energy consumption and chip temperatures. It has been recently shown that mapping communicating tasks to neighboring cores can reduce communication delays and the associated power consumption and improve throughput. This paper evaluates the contiguous allocation strategy first fit and non-contiguous allocation strategies that attempt to achieve a degree of contiguity among the cores allocated to a job. One of the non-contiguous strategies is a new strategy, referred to as neighbor allocation strategy, which decomposes the job request so that it can be accommodated by free core submeshes and individual cores that have degree of contiguity. The results show that the relative merits of the policies depend on the job's communication pattern.

KEYWORDS

2D Mesh, Communication Overhead, Manycore Systems, Network on Chip

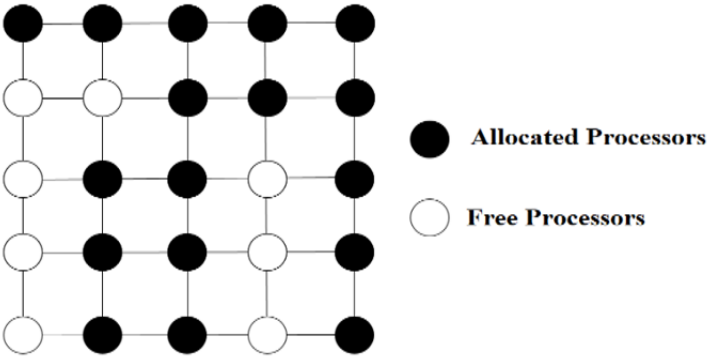
1. INTRODUCTION

Single-chip multicore processors and their network on chip (NoC) interconnection mechanisms have received extensive interest since the early 2000s. They have become attractive as the number of transistors that can be placed on a single die has continued to increase, as per Moore's law. A multicore processor consists of multiple processing cores that can execute instructions in parallel. Multicore processors that consist of a large number (tens to thousands) of relatively simple cores are referred to as manycore processors. Manycore processors have for goal achieving a high level of explicit parallelism.

The mesh interconnection topology is popular in NoCs. This includes both 2D and 3D meshes (Matsutani et al., 2007; Wentzlaff et al., 2007). An example is the 2D 8×10 mesh interconnection network used in an Intel manycore research chip that integrates 80 cores. Each of the cores contains a processing element (PE) and a 5-port router for communication (Vangal et al., 2007). Four ports are used for communicating with the east, west, north, and south neighbors of internal PEs or cores, as in Figure 1. Edge and corner cores have fewer neighbors. The fifth port is for communication with the

DOI: 10.4018/IJGHPC.320789 *Corresponding Author

Figure 1. An example of a 5×5 2D mesh



PE. Another example is the TRIPS On-Chip Network (OCN) that uses a 4x10 wormhole-routed 2D mesh interconnection network (Gratz et al., 2007). A 1024-node manycore system that has a mesh topology, and is partitioned into 32 clusters, where each cluster is 4×8 mesh, has been proposed recently. To decrease the system's overall diameter, each cluster has in its middle a Radio Hub (RH) that attaches to the four routers of the middle cluster node. Communications between clusters takes place through the Radio Frequency (RF) waveguides of the RHs. Within clusters, communication takes place over a mesh interconnection network (Lahdhiri et al., 2020).

A 2D mesh interconnection network (see Figure 1) is an example of direct networks, in which each processer or core is connected directly to its neighbors. In addition to being used over the past two decades in multicore systems, the 2D mesh interconnection network had been used in earlier large-scale multicomputer systems. This is due to its simplicity, regularity, scalability (i.e., it can be scaled up as the system needs), as well as its ease of implementation, and ability to benefit from the locality property in communication for many parallel applications. The 3D mesh is an extension of the 2D system, where several 2D mesh chips are stacked vertically on top of each other with additional top and down interconnection links, as appropriate.

A common issue in mesh NoCs is that they can suffer from high energy consumption and temperatures. Mapping communicating tasks to neighboring cores can reduce communication delays and associated power consumption and improve throughput and job execution times (Mosayyebzadeh et al., 2016; Agyeman et al., 2018; Dahir et al., 2021).

1.1. Core Allocation

We assume in this paper that a subclass of manycore systems implemented as a system on chip (SOC) will be used as general purpose multicomputers. In such systems, performance of an application depends on the job scheduling strategy, the core allocation strategy, and the characteristics of the SOC, including the NoC used. The job scheduler determines the order in which jobs are executed. The core allocation strategy determines the set of cores or PEs on which a job that is selected for execution will run. In addition, we assume in that the NoC used has the 2D mesh topology and jobs request the allocation of submeshes of size $a \times b$ upon arrival. As in previous works, we assume static space sharing allocation, where a job is allocated a set of cores exclusively until it terminates execution (Lo et al., 1997; Bani-Mohammad & Ababneh, 2018).

In general, two types of static space sharing allocation have been under investigation by researchers, the contiguous and non-contiguous allocation (Chuang & Tzeng, 1994; Lo et al., 1997; Seo & Kim, 2003; Bani-Mohammad & Ababneh, 2018). In contiguous allocation, the physical contiguity

among allocated cores is necessary, and the shape of the allocated submesh should be the same as that requested. This submesh restriction imposes the fragmentation problem, which has direct influence on the ability to utilize the PEs of the system. Fragmentation has two types: internal fragmentation and external fragmentation. Internal fragmentation occurs when the allocation strategy assigns to the job more cores than what it has requested. External fragmentation occurs when there are enough free cores, but the allocation algorithm is unable to allocate them because of the contiguity condition or lack of recognition by the algorithm.

Figure 2-A illustrates internal fragmentation, where a job that requests two cores is allocated four cores, causing internal fragmentation of 50%. The cores allocated are within the black frame. Figure 2-B illustrates external fragmentation, assuming that contiguous allocation is used, and a job requests eight cores as a 2×4 submesh. Eight cores are available in the mesh system, but they are not allocated because they are not contiguous.

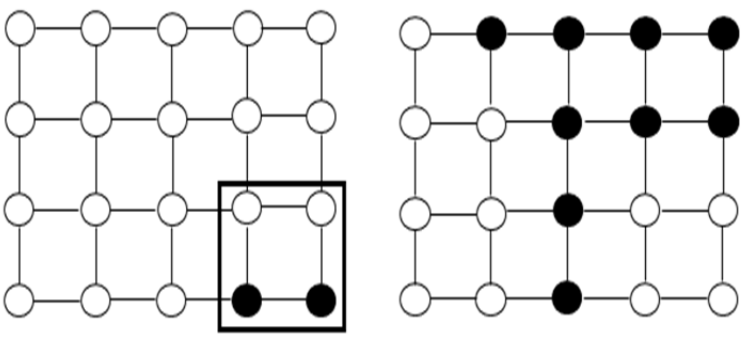
However, in non-contiguous allocation, the physical contiguity among the allocated cores, as well as having the same shape as the request, are not required; wherever cores are free, they can be allocated to the job.

Researchers have proposed non-contiguous allocation schemes that aim to improve the performance of the system in terms of average response time, which is the total time that the job spends in the computer system from arrival until departure, and in terms of system utilization, which is the percentage of cores that are utilized over time. Non-contiguous allocation can suffer from communication overhead as distances between communicating entities can be high, and interference from the messages of other co-running applications can occur. These effects can be reduced by preserving a good degree of contiguity among the cores allocated to the same job to decrease the network distances among communicating entities (Seo & Kim, 2003; Bani-Mohammad et al., 2007; Bani-Mohammad & Ababneh, 2018). In addition to improving communication delays and execution times, assigning communicating tasks to neighboring cores can reduce power consumption and chip temperatures (Mosayyebzadeh et al., 2016; Agyeman et al., 2018; Dahir et al., 2021).

Motivated by the above observations, a new allocation algorithm for 2D mesh systems is proposed in this paper. The algorithm aims to improve system performance by trying to allocate submeshes that have some contiguity degree among their cores. It starts by allocating large available rectangular submeshes, and then allocating neighbor cores. The proposed algorithm mainly aims to decrease the communication overhead in the network by preserving contiguity among the allocated cores.

The performance of the proposed algorithm is compared with that of the well-known contiguous First Fit (FF) algorithm (Zhu, 1992), and the non-contiguous L-Shaped Submesh Allocation algorithm (LSSA) (Seo & Kim, 2003) using simulations. The influence of different communication patterns on

Figure 2. Internal fragmentation and external fragmentation in 5×4 2D mesh



the performance of these allocation algorithms is studied. The communication patterns considered are one-to-all, all-to-all and near-neighbor.

The simulation results show that the performance of the proposed algorithm is better than that of the previous contiguous and non-contiguous allocation strategies considered in this work in terms of system utilization. This is because of its ability to alleviate external fragmentation. In terms of average response times, the simulation results show that the proposed algorithm has better performance than all other schemes when the one-to-all communication pattern is used. However, for all-to-all and near-neighbor communication patterns, the results show that the performance of the proposed strategy is better than that of LSSA, but FF is the best.

2. BACKGROUND AND RELATED WORKS

In this section, a brief description of some of the well-known 2D mesh contiguous and non-contiguous allocation strategies that have been proposed and investigated is presented.

2.1. 2D Buddy System

The Two-Dimensional-Buddy-System, known as 2DBS (Li & Cheng,1991), is applicable only to power-of-two square mesh systems and square submesh requests. A job that requests a submesh of size $a \times a$ is allocated a square submesh of side length 2^k, where $k = \left\lceil \log a \right\rceil$. For instance, assume a job requests 9 PEs as 3×3 submesh, then $k = 2$ and a 4×4 submesh is allocated to that request, which results in an internal fragmentation of 44%. This scheme suffers from both internal and external fragmentation, in addition to requiring that the system and requests be square (Chuang & Tzeng, 1994).

2.2. First-Fit (FF) and Best-Fit (BF) Allocation Strategies

First-Fit (FF) and Best-Fit (BF) allocation strategies (Zhu, 1992) are applicable to any mesh system regardless of its size. In these strategies, internal fragmentations are eliminated by allocating precisely the requested number of cores or processors. The nodes that can be used as base nodes for submeshes that can hold the jobs are stored in an array of the size of the mesh. When using FF, the first free suitable base node is selected as the base node for allocation. On the other hand, BF chooses the node that has the largest number of busy neighbors and smallest number of free areas to be the base node for the allocated submesh. The simulation results (Zhu, 1992) show that the performance of FF and BF are close in both average response times and system utilization. FF and BF suffer from external fragmentation as they are contiguous policies and do not support changing the orientation of the job request, which degrades system performance (Seo & Kim, 2003). Switching request orientation had been proposed in (Ding & Bhuyan, 1993), where allocation for $b \times a$ submesh is attempted if it fails for the requested $a \times b$ submsh.

2.3. Flexfold Strategy

In this strategy (Gupta & Jayendran, 1996), switching the orientation of the request and changing its shape are utilized. For a job request for a submesh $S(a,b)$, the Flex Fold strategy attempts allocation as follows: $S(a,b), S(b,a), S(a\ /\ 2, 2b)$, $S(2a, b\ /\ 2)$, $S(2b, a\ /\ 2)$ and $S(b\ /\ 2, 2a)$ using a first fit strategy. Despite improving system utilization, this strategy suffers from external fragmentation and has a constraint on the job side lengths, where both sides must be even (Seo & Kim, 2003).

2.4. ALL Shapes First-Fit Sub-Mesh Allocation Strategy (ASFF)

The All-Shapes First-Fit contiguous submesh allocation strategy (ASFF) (Ababneh et al., 2010), attempts allocation to an incoming job request by permitting all possible 2D shapes. In ASFF, given

a job request for n cores, all valid request shapes are constructed, and then these shapes are considered for allocation in a specific order. Priority is given to shapes that have smaller differences between width and height. The allocation process stops upon the first successful allocation. For example, if 12 processors are requested and the mesh system size is 6×6, then ASFF generates the following request sequence: (4,3), (3,4), (6,2), and (2,6), while if a job requests 25 processors, then only one shape (5,5) is generated. ASFF uses FF for allocation.

2.5. L-Shape Sub-Mesh Allocation Strategy (LSSA)

The L-Shape Sub-mesh Allocation strategy (LSSA) (Seo & Kim, 2003) uses a scheme that changes a rectangular submesh request *S(a,b)* into an L-shaped request. This is done by virtually removing a smaller rectangular submesh *R(c,d)* from *S(a,b)* and attaching it to a side of the remaining block in *S*. The lengths of attachment sides in *R* and the remaining block of *S* are chosen to be equal, to produce an L-shaped allocation request for $a \times b$ processors. Note that there are normally multiple choices for *R*, resulting in various L-shaped requests.

In LSSA, if the allocation of a submesh $S(a,b)$ is requested, LSSA first checks if the requested number of processors $a \times b$ is in the free state in the system. If not, allocation fails. Then, LSSA attempts allocation in the following order until allocation succeeds, or all choices are exhausted: allocation for $S(a,b)$, $S(b,a)$, $S(a/2,2b)$, $S(2b,a/2)$, $S(2a,b/2)$, then $S\left(b/2,2a\right)$. If this fails, LSSA attempts allocation for L-shaped submeshes.

2.6. Minimum Interference Paging (MIP)

The Minimum Interference Paging strategy (MIP) (Bani-Mohammad & Ababneh, 2018), attempts to reduce the distances among cores allocated using a paging variant that chooses a set of cores with the lowest distance between the first and last allocated cores, where the distance is the number of cores between the first allocated core and last on allocated. In this strategy, the valid request shapes are generated for the number of cores requested. Then, the successive shapes are first considered for contiguous allocation using the FF policy (Zhu, 1992). Contiguous allocation is given priority because it eliminates interference among the messages of different jobs, which reduces contention and communication latency. If contiguous allocation fails, the MIP algorithm (Bani-Mohammad & Ababneh, 2018) is applied, where the cores are scanned, as in Paging[0] (Lo et al., 1997), starting with the first core in the mesh system to determine all the base and end cores for all available sets of cores that can accommodate the job request. Free cores with the lowest distance between the base and end cores are allocated to the job request. The distance of a set is the number of cores visited between the base and end cores of the set during the scan.

2.7. Efficient Maximal Free Submesh Detection Scheme for Space-Sharing Allocation in Manycore Systems with 2D NoCs (PMFL)

In this strategy (Ababaneh & Bani-Mohammad, 2022), an efficient recognition-complete maximal free submesh detection scheme for 2D mesh-connected manycore systems is proposed. An advantage of this scheme over the previous recognition-complete scheme (KYMFL), proposed in (Kim & Yoon, 1998), is that its time complexity is quadratic in the number of free submeshes, while the time complexity of the previous scheme is cubic in this number. Using detailed simulations, the two schemes have been evaluated and compared. In these simulations, various previous promising policies for deciding where allocation takes place are considered. The simulation results show that when allocation and de-allocation times are considered, the proposed submesh detection scheme (PMFL) substantially outperforms the previous maximal free submesh detection scheme (KYMFL). It has achieved up to seventy percent improvement in the measured combination of these times.

3. PROPOSED NEIGHBOR ALLOCATION STRATEGY (NAS)

The main aim of the proposed Neighbor Allocation Strategy (NAS) is to decrease job communication costs and achieve good system utilization by trying to execute jobs on submeshes that have contiguity among their cores; contiguity within this context is having the network routers of cores connected via the router of at least one intermediate core.

3.1 The Neighbor Allocation Strategy (NAS)

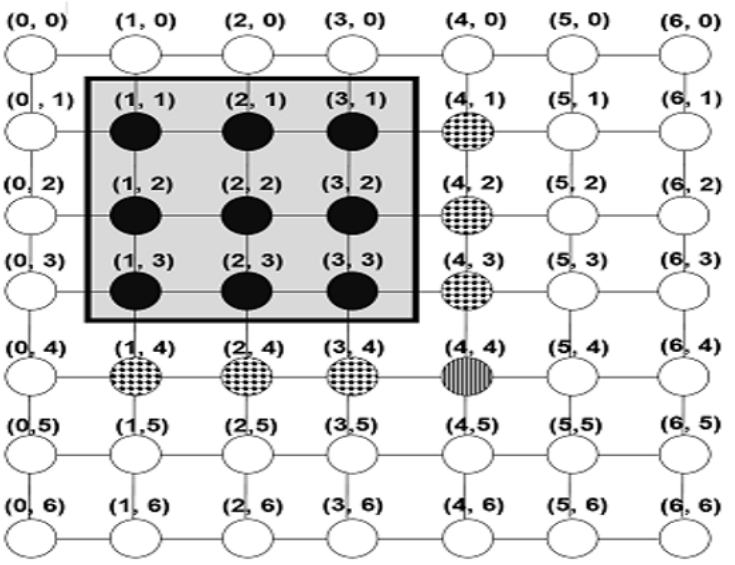
A 2D mesh is represented by $M\left(h,\ w\right)$, where h is the height of the mesh and w. is its width, each node can be identified by two coordinates $\left(x,y\right)$, where $0 \le x < w$. and $0 \le y < h$. A node *P* and a sub-mesh are said to be neighbors when one of the border nodes of the submesh $S\left(a,\ b\right)$ and P are neighbours, with their coordinates $\left(x_{11},y_{11}\right)\ and\ \left(x_{21},y_{21}\right)$ satisfying one of the following conditions:

$$x_{21} = x_{11} + 1,\ and\ y_{21} = y_{11}$$

$$y_{21} = y_{11} + 1,\ and\ x_{21} = x_{11}$$

For example, the P node (4,1) in Figure 3 is neighbor to the submesh border node (3,1), where x_{21} in $P = 3+1,\ and\ y_{21}$ in $P = y_{11} = 1$, and also the P node (1,4) is neighbor to the submesh border node (1, 3), where y_{21} in $P = 3+1,\ and\ x_{21}$in $P = x_{11} = 1$. In general, Figure 3 shows an example that clarifies the definition of neighbor relation between a node and a submesh; here the submesh $S(3,3)$ that has nodes represented as black circles, is neighbor to the set of nodes {(4, 1), (4, 2), (4, 3), (1, 4), (2, 4), (3, 4)}. Also, if (4, 1) is allocated to the same job as $S(3,3)$, then (5, 1) is considered a neighbor of the sub-mesh S. Likewise, (4, 4) is considered a neighbor of the sub-mesh S if (4, 3) or (3, 4) are allocated to the same job.

Figure 3. Neighbor relation between a node and a sub-mesh



An incoming job request is represented by $S(a,b)$, where the number of PEs requested is $a \times b$. The NAS scheme initially checks if the requested number of processors is available in the system. If not, allocation fails. Otherwise, it searches for free submeshes in the following sequence until the requested free submeshes are allocated or allocation fails. Initially NAS tries allocation to the original submesh $S(a,b)$ using FF, and if this fails then ASFF is attempted for $S(a,b)$. If this fails, LSSA is applied by constructing submeshes from $S(a,b)$ and its rotation $S(b,a)$, and attempting FF allocation for the constructed submeshes. If this fails, the following NAS scheme is applied.

The NAS largest submesh assigned to the job is called the nucleus submesh (NS). For any job that requests a submesh $S(a,b)$, the NS is constructed as follows: for even a, where $a \geq b \geq 2$, the nucleus submesh is $NS(a/2, b+k)$, where $1 \leq k \leq b$. For odd a, where $a \geq b \geq 2$, the nucleus submesh is $NS(\lceil a/2 \rceil + k, b + \lfloor a/2 \rfloor - k)$, where $1 \leq k \leq (b - 1 - \lceil a/2 \rceil)$. NAS changes k until it finds an available free submesh. If $b - 1 - \lceil a/2 \rceil$ is -1, NAS uses -1 as k, and increments k by 1 while it is less than 4 as in (Seo & Kim, 2003), otherwise k is incremented by $\lfloor b/4 \rfloor$. This is how the L-Shape (LS) submesh is constructed in LSSA. Next NAS searches the system to allocate the remaining requested processors as neighbors to the allocated processors. The goal is to maintain a degree of contiguity, while attempting to increase system utilization.

This allocation process is implemented by the algorithm in Figure 4. Note that allocation succeeds if the number of free processors is greater than or equal to $a \times b$.

Figure 4. Outline of the NAS allocation algorithm

```
Procedure NAS_Allocate (a, b):{
W: Width of the mesh; H: Height of the mesh; Job_size = axb; Allocated_processors = 0; Rotation_flag = True; Free_alocation_flag = True.
Step 1.        If (a == b) Rotation_flag=False;
Step 2.        If (number of free processors == 0 or number of free processors < job_size) Return Failure.
Step 3.        If (FF or ASFF or LSSA allocation succeeds) {
                   Assign submesh of axb processors to the job request.
                   allocated processors= axb; Add allocated processors info to APA.
                   //APA: Allocated Processors Array, which is an array that contains the job id and the coordinates of the allocated processors.
                   Return Success.}
Step 4.        If (a is even and a >= b and a >=2) Go to step 5.
               If (a is odd and a >= b and a >=2) Go to step 5.
               If (b is even and b >=a and b >=2 and Rotation_flag == True) Go to step 5.
               If (b is odd and b >=a and b>=2 and Rotation_flag == True) Go to step 5.
Step 5.        Compute k value, where k is a variable used to rebuild the job sides length (width, height), and Max_k is the maximum value for k.
               While (k <= Max_k){
                       Construct the nucleus sub-mesh (c, d) using a, b, and computed k.
Step 5.1.              Search the mesh system for the nucleus sub-mesh (c, d).
                       If (the nucleus sub-mesh is found) {
                               Assign cxd processors to the job request; Allocated_processors+= cxd.
                               Add allocated processors info to APA; Go to step 6.}
                       Else
                               Recalculate k.}
Step 6.         X= Job_size - Allocated_processors.
               If ( X > 0 ) {
               While (Job_size >=Allocated_processors){
                       Search the mesh for free neighbor processors to the nucleus sub-mesh.
                       if (free_processor is found) {
                               Assign it to the job request; Allocated_processors+= 1
                               Add allocated processors info to APA.}}
               If (Job_size ==Allocated_processors) Return Success.
               Else if (number of free processors >= job_size and Free_alocation_flag ==True) Free_alocation_flag = False; Go to step 7.
               Else Return failure;
Step 7.        If (number of free processors >= job_size) {
                       Consider nucleus sub-mesh (c, d) as (1, 1); Go to step 5.1. }

} end procedure
```

Figure 5 shows the initial state of an example 2D mesh connected multicore system, where all free cores (not allocated) are shown as white circles.

We give some examples selected carefully to clarify how NAS works. In the first example, the job requests a submesh of size 5×3 , as shown in Figure 6. The NAS strategy tries FF, ASFF and LSSA. When this fails, NAS rebuilds the job request by constructing its nucleus submesh of size 2×6 and allocates it in the mesh system, as shown in Figure 7. Next, NAS searches for free neighbor nodes. As shown in Figure 7, the neighbor cores (1, 2 and 3) are allocated to the job.

Figure 5. Initial state of a 7 × 7 2D mesh

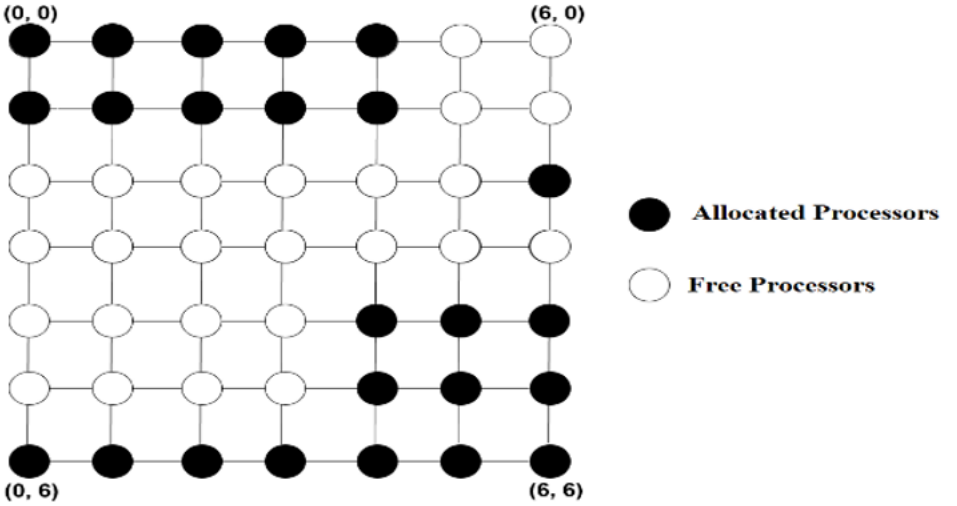


Figure 6. A job requests 5x3 submesh in 7x7 2D 2D mesh

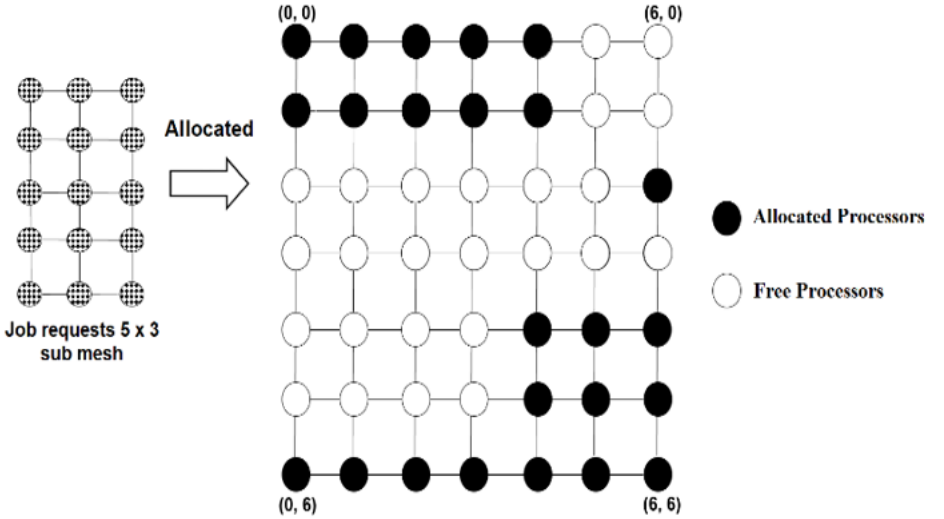
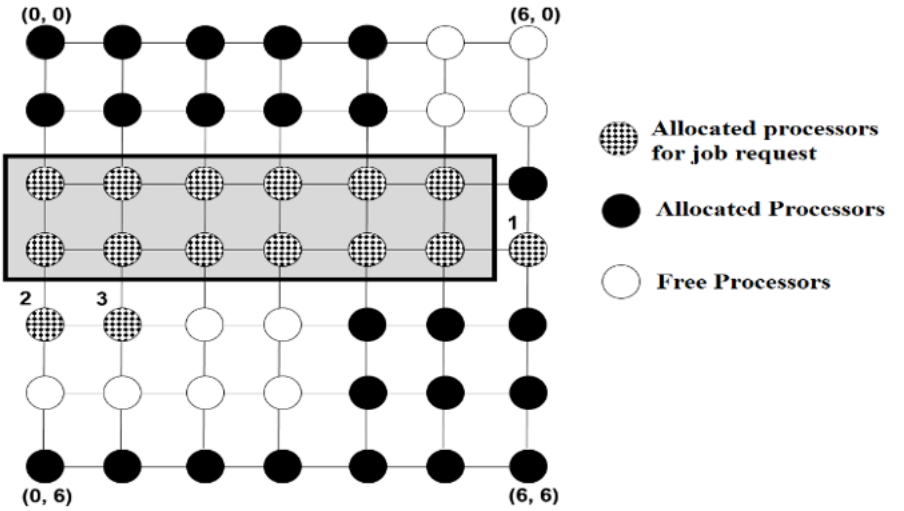


Figure 7. Allocation of 5x3 submesh in 7x7 2D mesh



Assuming the system state shown in Figure 8, the second example shows an incoming job that requests a submesh of size 6×3. Here, NAS rebuilds the request as a 6×2 nucleus submesh and allocates it in the system, then 6 cores are allocated to the job as shown in Figure 9.

In the third example, a job requests a 5×1 submesh, as shown in Figure 10. NAS tries to form a nucleus submesh, but this fails although the required number of cores is free in the mesh system. In this case, NAS considers the nucleus submesh to be 1×1 and searches the system to allocate the remaining number of cores by keeping contiguity, as shown in Figure 11.

Figure 8. A job requests 6x3 submesh in 7x7 2D mesh

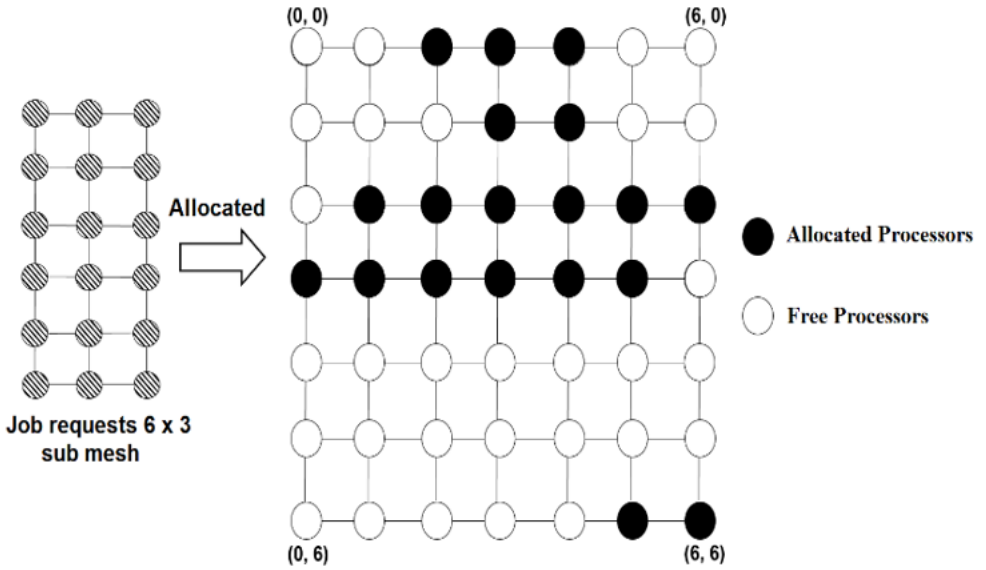


Figure 9. Allocation of 6x3 submesh

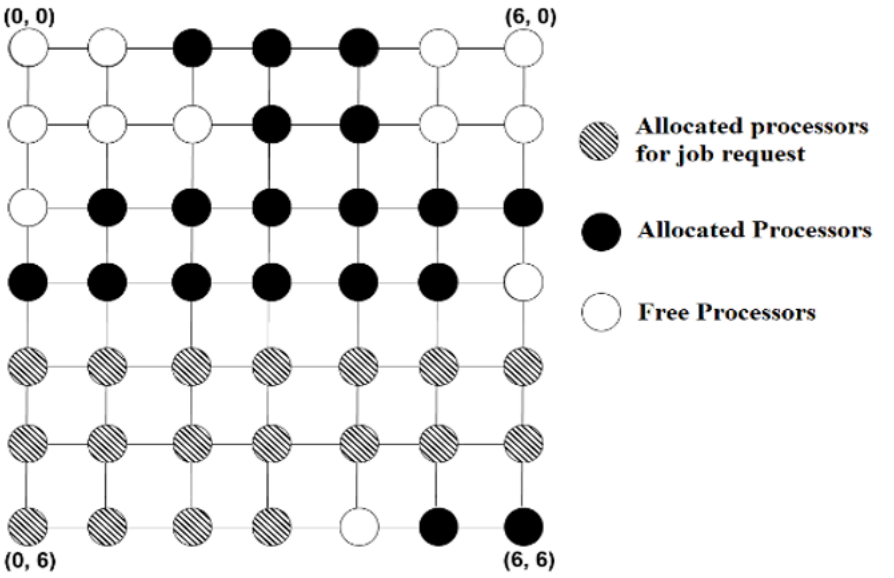
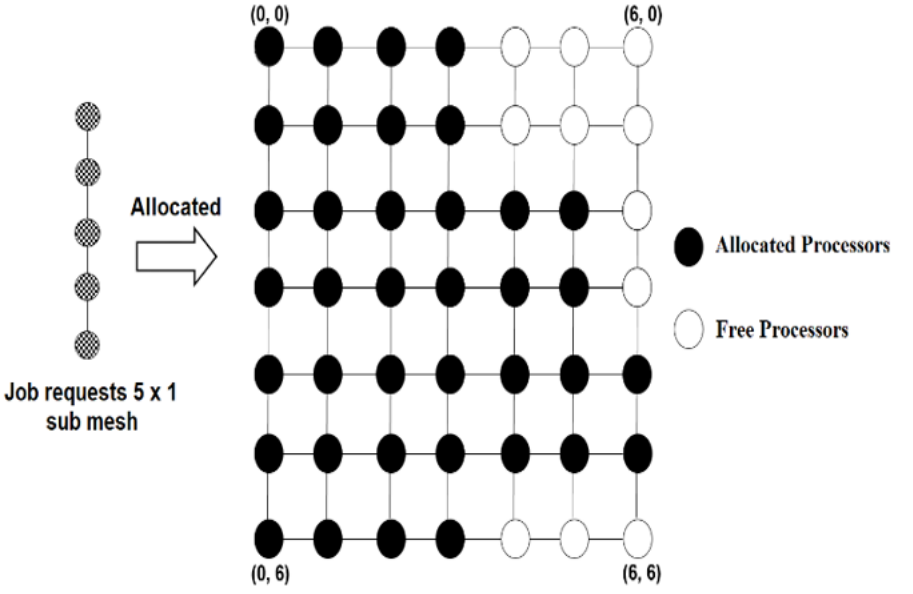


Figure 10. A job requests 5x1 submesh



In Figure 12, the request is for a submesh of size 5×2. In this example, NAS tries allocation for the nucleus submesh 4×3, but this is not available. Therefore, NAS tries again to rebuild the request by forming another nucleus submesh as 5×2. Again, this shape is not available. NAS forms another nucleus submesh as 6×1 and allocation succeeds for this submesh, and then the 4 remaining processors are allocated as shown in Figure 13.

Figure 11. Allocation of 5x1 submesh

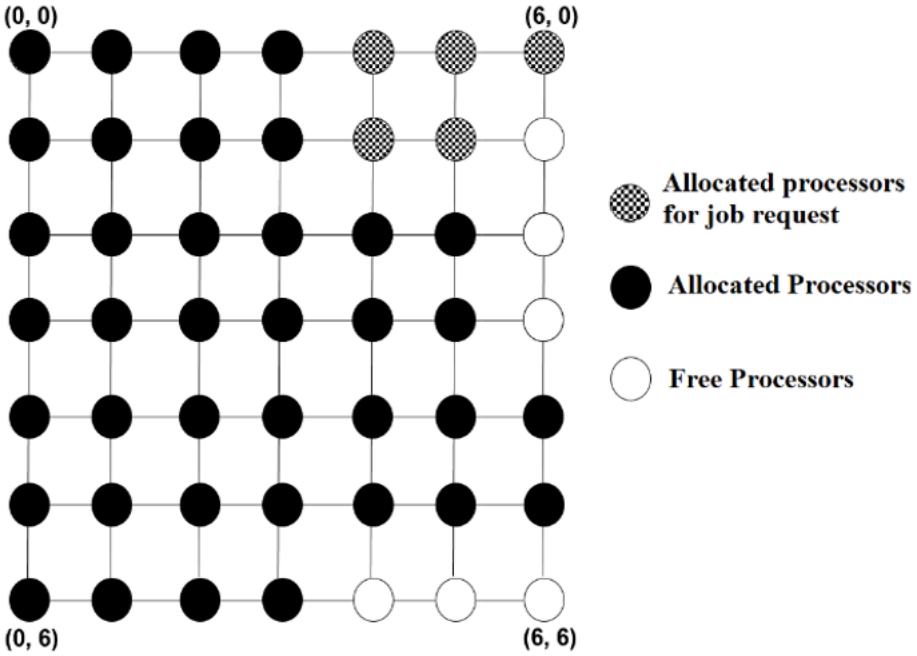


Figure 12. A job requests 5x2 submesh

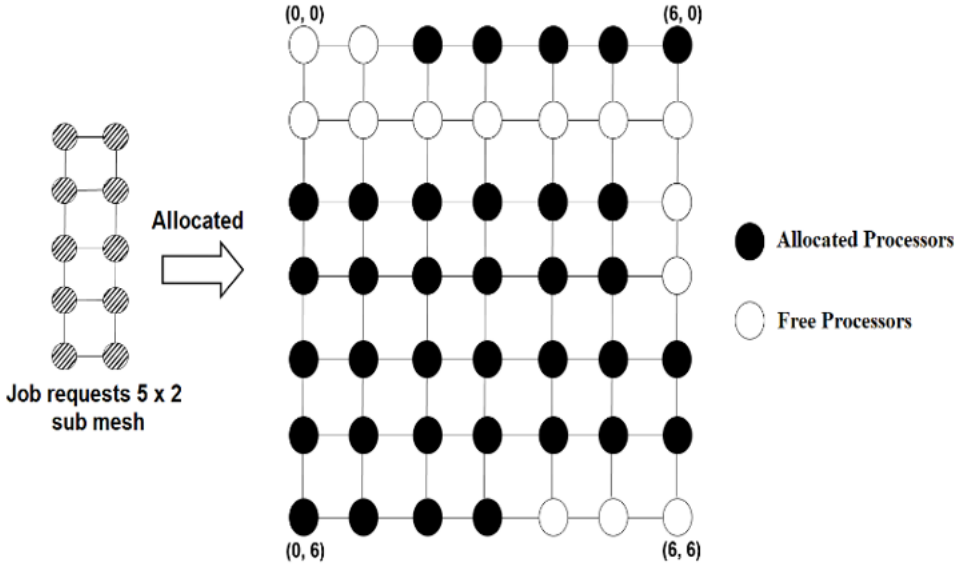
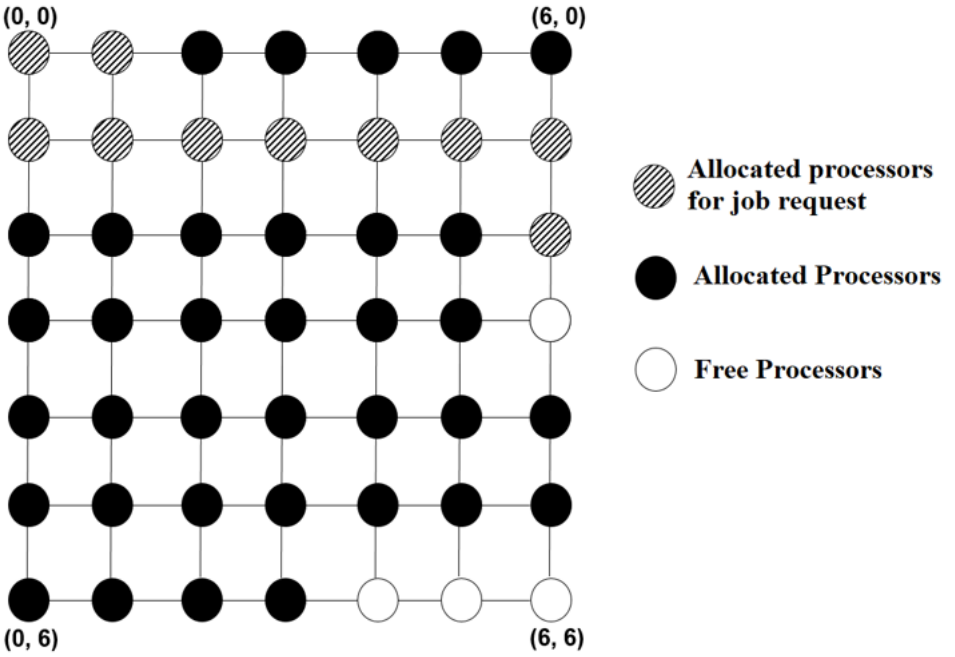


Figure 13. The allocation of job 5x2 in the system



4. SIMULATION RESULTS

In our simulation experiments carried out using the ProcSimity v4.3 simulator (ProcSimity v4.3, 1996), we assume a two-dimensional mesh-connected manycore system where cores or PEs are interconnected using a direct 2D NoC, and we use three common communication patterns for comparing performance of FF, LSSA and the proposed strategy. The communication patterns are the Near-Neighbor, One-to-All and All-to-All patterns. In Near-Neighbor communication, every core allocated to the job sends a message to each of its four neighbor cores: east, west, north, and south neighbors. This pattern has been added to ProcSimity v4.3 (Bani-Mohammad & Ababneh, 2013). In One-to-All communication, a core, randomly selected among those allocated to the same job, sends a message to all cores that are allocated to the same job (Lo et al., 1997). In All-to-All communication, every core assigned to a job sends a message to all other cores executing the same job (Lo et al., 1997). A job runs until all messages it sends are received, and each job does exactly one iteration of the given communication pattern

ProcSimity is a software simulation tool for processor allocation and job scheduling schemes in meshes and k-ary-n-cube multicomputers. It is an open source code simulator that was developed at the University of Oregon and written in the C programing language (ProcSimity v4.3, 1996). The ProcSimity simulator has been updated in (Bani-Mohammad & Ababneh, 2013; Ababneh & Bani-Mohammad, 2022; Al Abass et al., 2022) by adding new communication patterns and new allocation and scheduling strategies. ProcSimity is suitable for evaluating processor allocation and job scheduling strategies for mesh connected multicomputers and manycore systems with direct 2D NoCs.

We conduct enough runs to keep the confidence level above 95% that the relative errors are at most 5%. In each run, the values of measured metrics that include job response time, system utilization, service time, and finish time are calculated. Overall average performance values are computed at

the end of runs. The performance values considered are the job average response time and system utilization. The number of jobs per run is 1000.

The mesh system in the simulations is a 2D square mesh with side lengths of *L*. Job inter-arrival times are exponentially distributed. The job scheduling strategy used in this research work is First Come First Served (FCFS). FCFS has been used because it is fair, and the goal is to compare allocation schemes, not scheduling policies.

The job arrival rate is the inverse of the mean inter-arrival time of jobs. Two distributions are used to generate widths and heights of job requests. The first is the uniform distribution over [1, L], where the width and length of a request are generated independently. The second distribution is the uniform-decreasing distribution. It is governed by four probabilities p_1, p_2, p_3, and p_4, and three integers l_1, l_2, and l_3, where the probabilities that the width and height of a request fall in the ranges [1, l_1], [l_1+1, l_2], [l_2+1, l_3], and [l_3+1, L] are p_1, p_2, p_3, and p_4, respectively. Again, the width and height of a request are generated independently. The probability of the side lengths within the ranges are equally likely. For the simulation experiments in this research work, L=16, $p1$=0.4, $p2$ =0.2, $p3$=0.2, $p4$ =0.2, $l1$= L/8, $l2$ = L/4, $l3$ = L/2, and $l4$ = L. These distributions have often been used in the literature (Lo et al., 1997; Bani-Mohammad & Ababneh, 2018).

Wormhole switching is used as a flow control mechanism in the interconnection network. Flits are assumed to take one time unit to move between two adjacent nodes, and t_s time units to be routed through a node. Message sizes are represented by P_{len}. The performance figures in this paper are based on using $t_s = 3$ time units, and $P_{len} = 8$.

4.1. System Utilization Results

Figures 14 to 19 show the mean system utilization for One-to-All, All-to-All and Near Neighbor communication patterns, and both the uniform and uniform-decreasing job size distributions.

The results show that for heavy system loads (i.e., high job arrival rates) NAS achieves higher utilization than the remaining policies for the two job size distributions and all communication patterns considered. This means that allocation is more likely to succeed in NAS than in FF and

Figure 14. System utilization vs. arrival-rate using one-to-all communication pattern and uniform distribution for job size in a 16×16 mesh

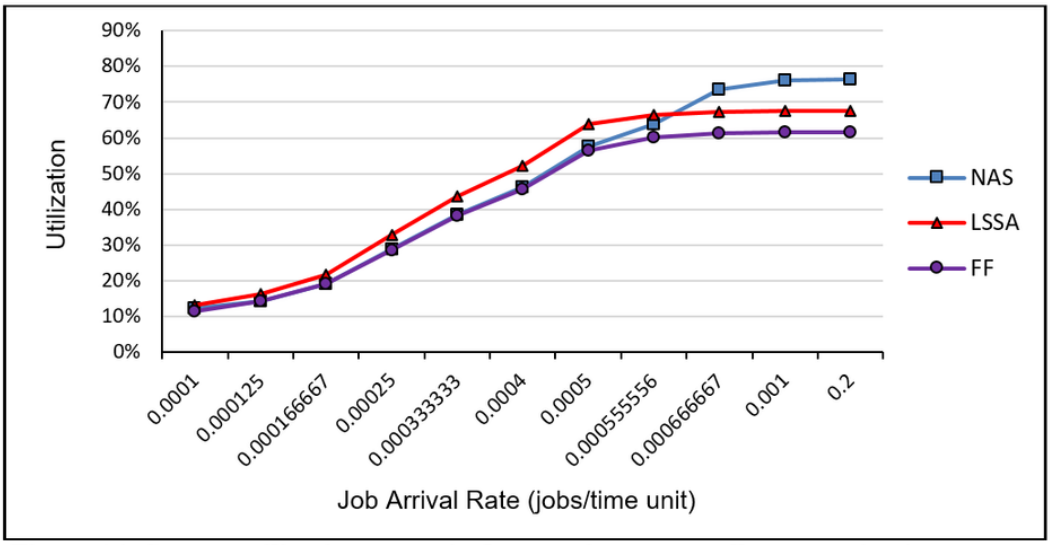


Figure 15. System utilization vs. arrival rate using the one-to-all communication pattern and uniform-decreasing distribution for job size in a 16×16 mesh

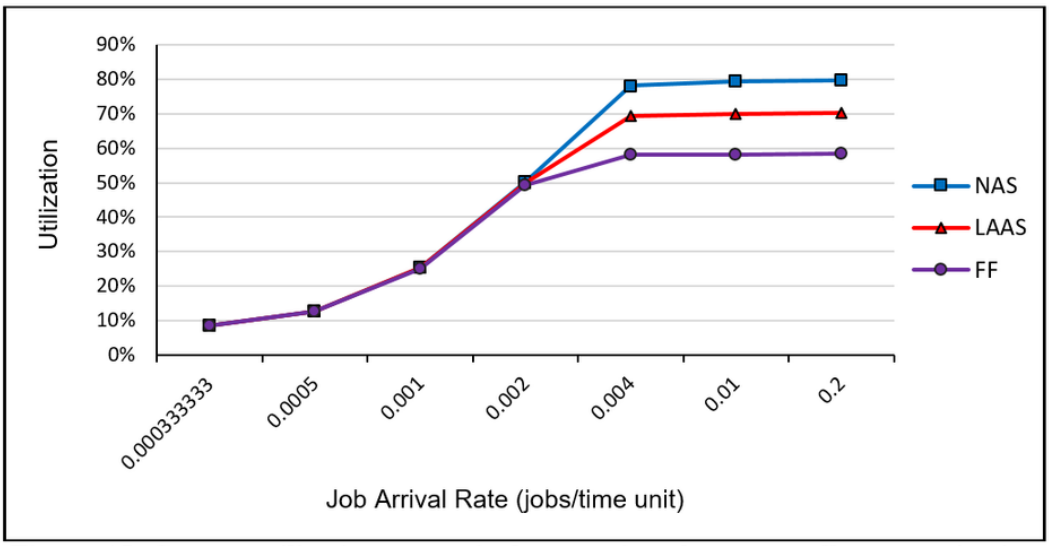
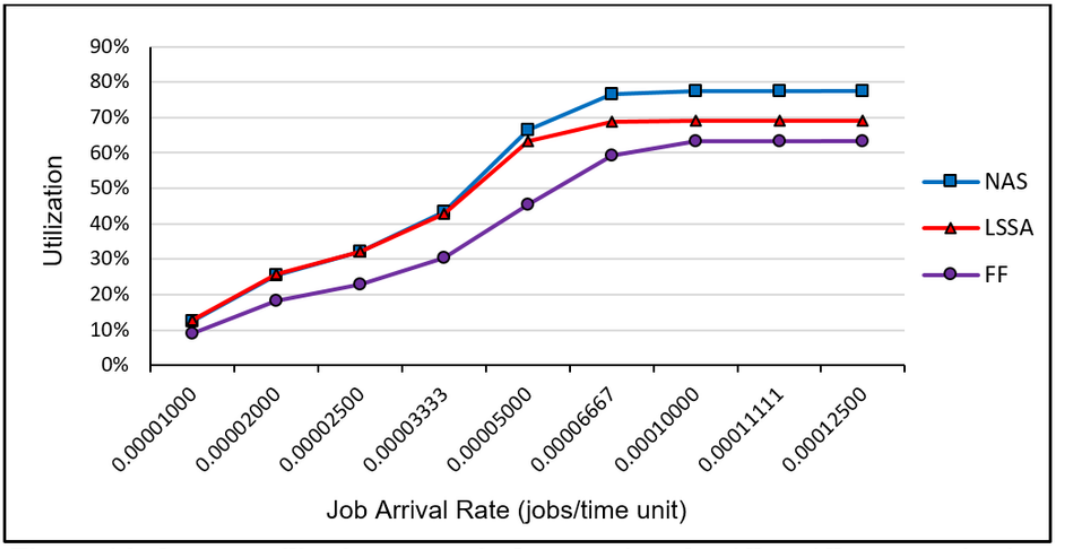


Figure 16. System utilization vs. arrival rate using the All-to-All communication pattern and uniform distribution for job size in a 16×16 mesh



LSSA. However, because higher system utilization may be accompanied with longer execution times due to longer distances and congestion that can occur in non-contiguous allocation, response times are a better metric. Moreover, they are of primary direct interest to users.

In Figure 14, for example, the mean system utilization for all considered allocation algorithms is almost the same for job arrival rates that are below the 0.0005 jobs/time units. However, for job arrival rates above 0.001 jobs/time units, the performance of NAS is better than that of LSSA and FF. This is because NAS has better ability to avoid external fragmentation by utilizing the free processors.

Figure 17. System utilization vs. arrival rate using the All-to-All communication pattern and uniform-decreasing distribution for job size in a 16×16 mesh

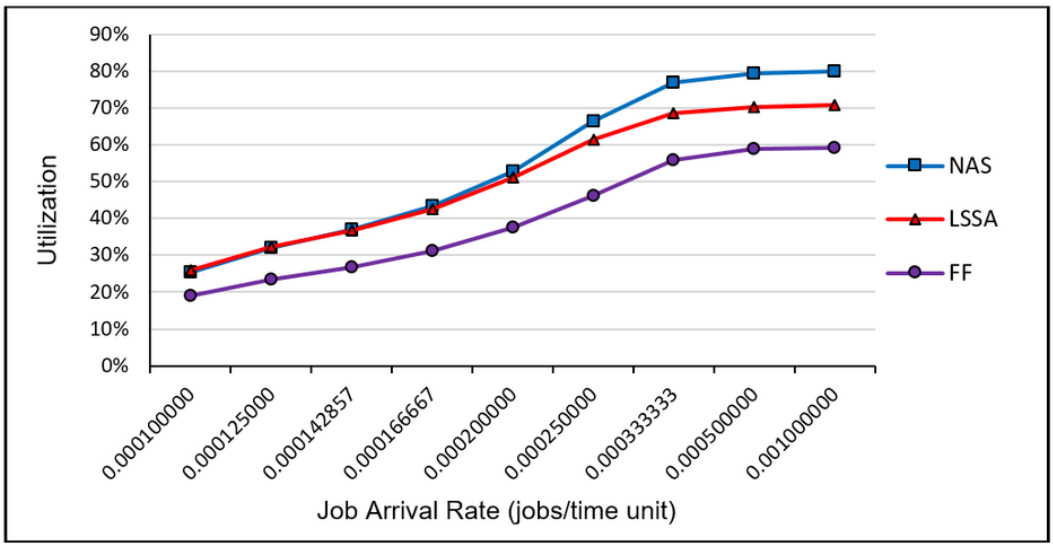
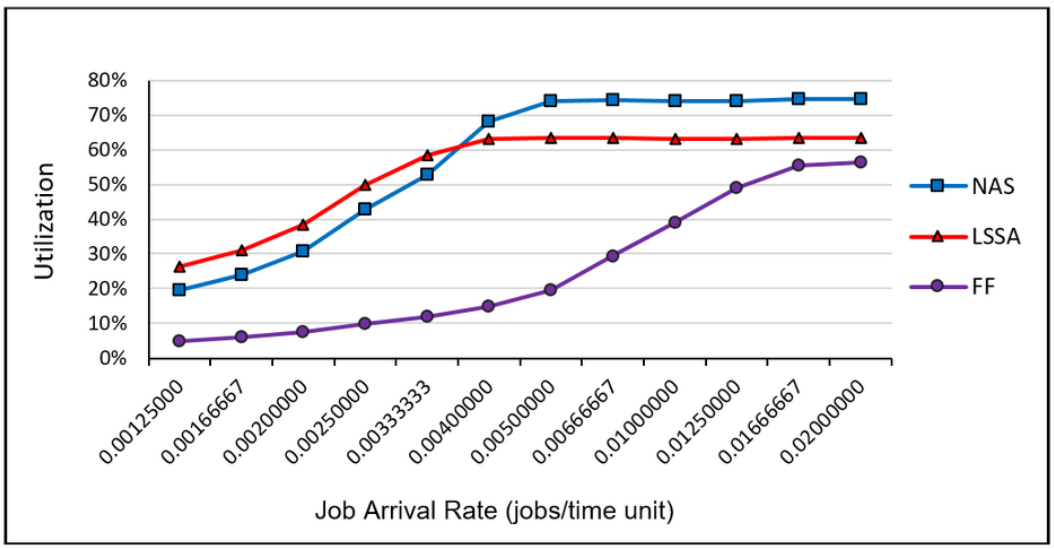


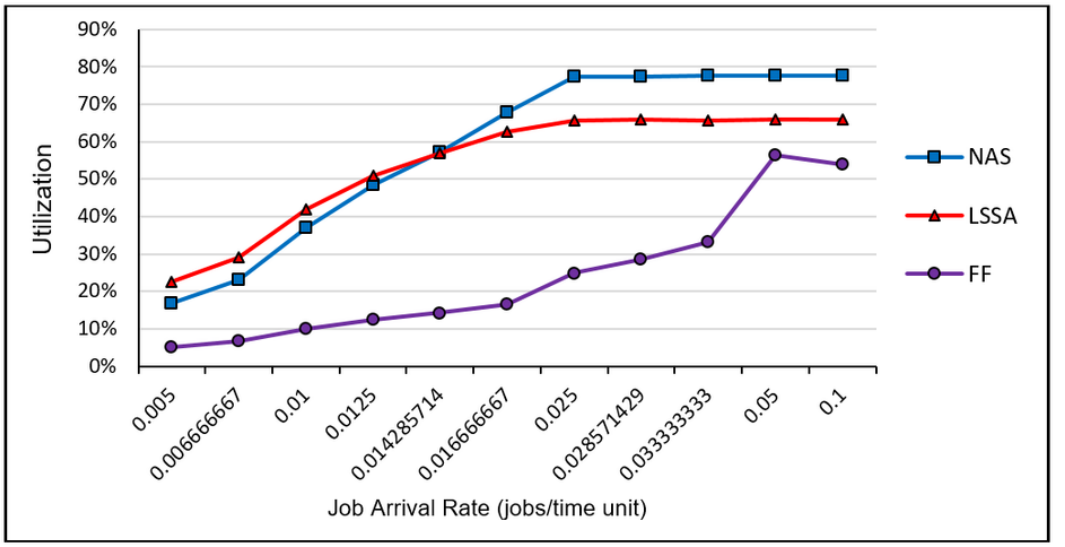
Figure 18. System utilization vs. arrival rate using the near-neighbor communication pattern and uniform distribution for job size in a 16x16 mesh



In Figure 15, it can be seen that the mean system utilization is improved for all allocation strategies when the uniform-decreasing distribution is used. This is because of the increased probability of generating small jobs compared to the size of the mesh system; successful allocation is more probable for smaller jobs. NAS also performs much better than LSSA and FF for high job arrival rates.

In Figure 16, the mean system utilization is almost the same for both of NAS and LSSA for the job arrival rates below 0.00005 jobs/time units, and it is better than that of the FF. This is again because of the ability of NAS and LSSA to better avoid external fragmentation. For high job arrival

Figure 19. Mean system utilization vs. arrival rate using the near-neighbor communication pattern and uniform-decreasing distribution for job size in a 16×16 mesh



rates (0.00006667 jobs/time units and above), NAS performs better than LSSA and FF. This is because of its ability to better avoid external fragmentation as compared to LSSA.

In Figure 17, the mean system utilization is improved for all allocation strategies when the uniform-decreasing distribution is applied. This is because of the increased probability of generating small jobs compared to the size of the mesh system, which makes successful allocation more probable. NAS performs much better than LSSA and FF.

In Figure 18, the results show that the mean system utilization for LSSA is better than that of NAS and FF when the job arrival rates are below 0.00333333 jobs/time units. However, NAS performs better than LSSA and FF when the job arrival rates are above 0.004 jobs/time units. This is because of the ability of the NAS to better avoid external fragmentation when the load is heavy.

In Figure 19, the results show that the mean system utilization for LSSA is better than that of the NAS and FF for the job arrival rates below 0.0125 jobs/time units. However, NAS performs better than LSSA and FF for the job arrival rates that are above 0.025 jobs/time units. This is because of the ability of the NAS to better avoid external fragmentation under heavy loads.

4.2. Average Response Time

Figures 20-25 show the average response times results using the One-to-All, All-to-All and Near Neighbor communication patterns when the FCFS scheduling scheme is used, and for both job size distributions considered, uniform and uniform-decreasing.

Figures 20 and 21 show that the NAS allocation strategy has superior performance over all other allocation strategies for the One-to-All communication pattern under the uniform and uniform-decreasing job size distributions considered. This is because of NAS ability to remove external fragmentation as compared to FF and LSSA, which increases the probability of successful allocation, and hence enhances the system performance in terms of job response time.

For All-to-All and Near Neighbor communication patterns results shown in Figures 22-25, FF has the best performance among the allocation strategies considered for both the uniform and uniform-decreasing job size distributions. This is because contiguous allocation strategies allocate a rectangular submesh for job requests, and this can reduce inter-job interference and communication

Figure 20. Average response time vs. arrival rate using one-to-all communication pattern and uniform distribution for job size in a 16×16 mesh

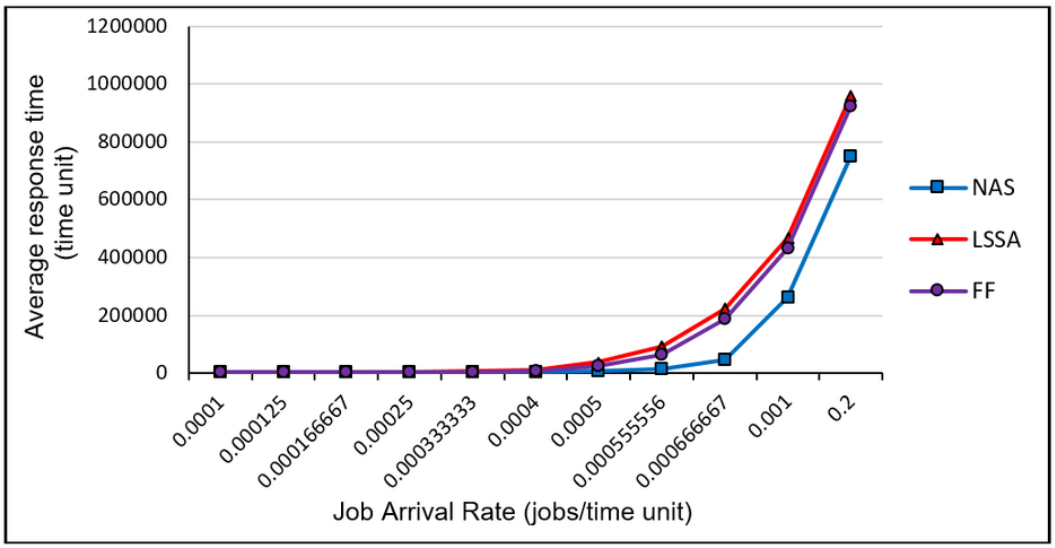
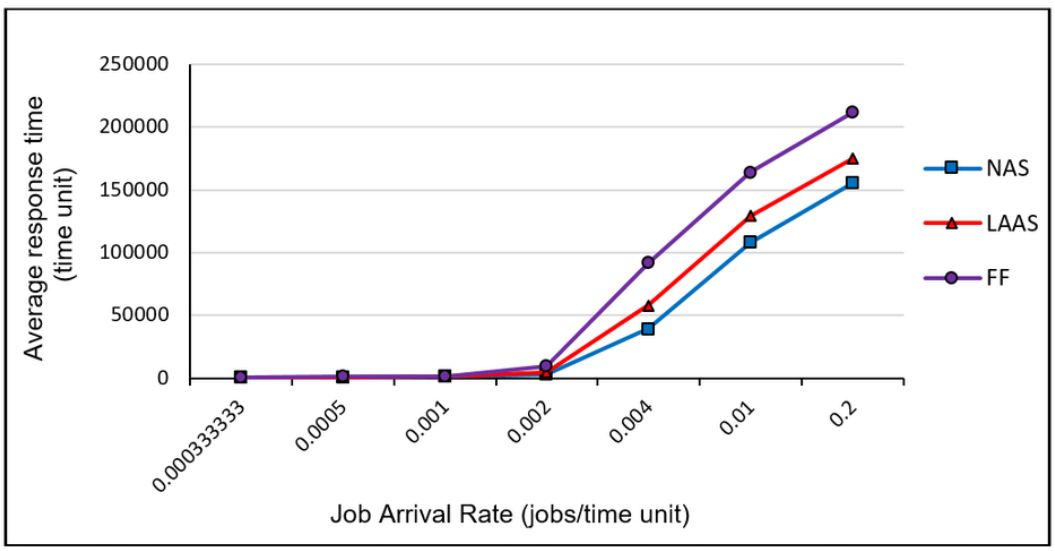


Figure 21. Average response time vs. arrival rate using the one-to-all communication pattern and uniform-decreasing distribution for job size in a 16×16 mesh



distances, which reduce the communication overhead in the system. However, NAS is better than LSSA, presumably because of its ability to achieve superior system utilization.

5. CONCLUSION AND FUTURE DIRECTIONS

Contiguous allocation schemes allocate a single submesh to each job, and they suffer from processor fragmentations. In non-contiguous allocation, the job can be executed on several small submeshes

Figure 22. Average response time vs. arrival rate using the All-to-All communication pattern and uniform distribution for job size in a 16×16 mesh

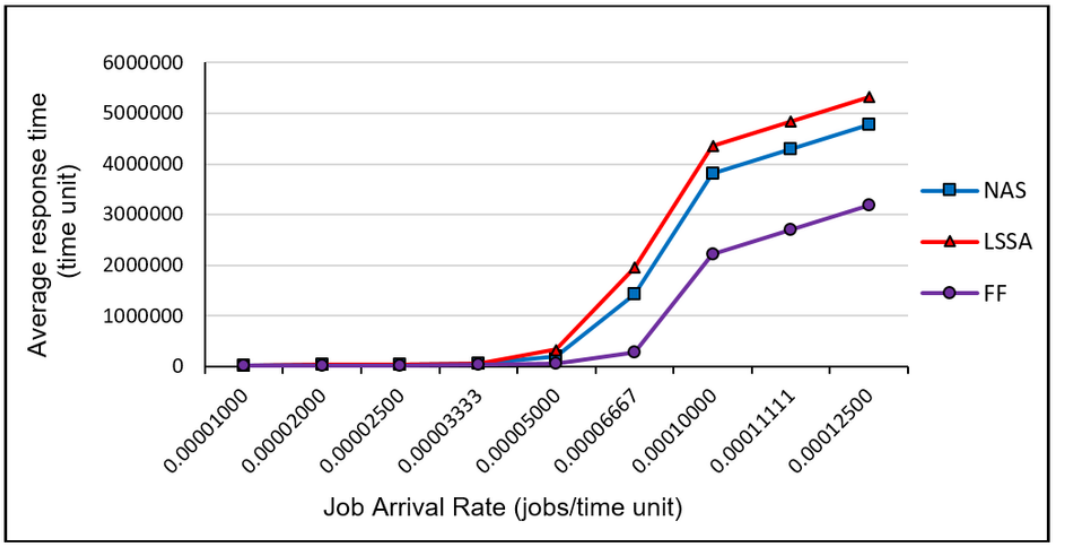
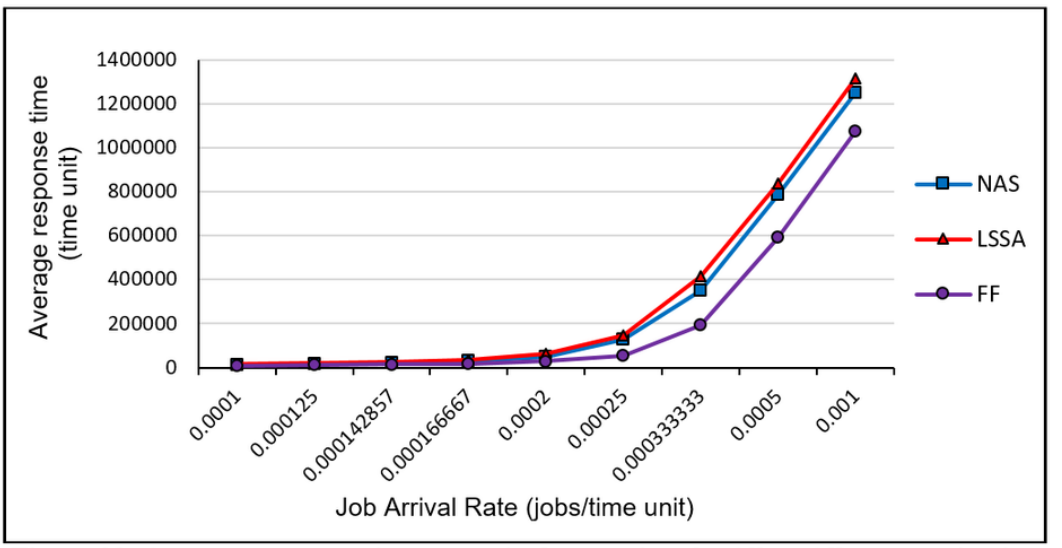


Figure 23. Average response time vs. arrival rate using the All-to-All communication pattern and uniform-decreasing distribution for job size in a 16x16 mesh



instead of waiting for one submesh of the requested size and shape to be available. The goal is to improve system performance by minimizing processor fragmentation, however non-contiguous allocation can result in high communication overhead. Generally, the aim of any allocation strategy is to improve system performance by increasing system utilization and decreasing jobs' communication overhead and response times.

Motivated by the previous observations, NAS has been proposed. Its goal is to reduce external processor fragmentation, while preserving a good degree of contiguity among the allocated processors

Figure 24. Average response time vs. arrival rate using the near-neighbor communication pattern and uniform distribution for job size in a 16×16 mesh

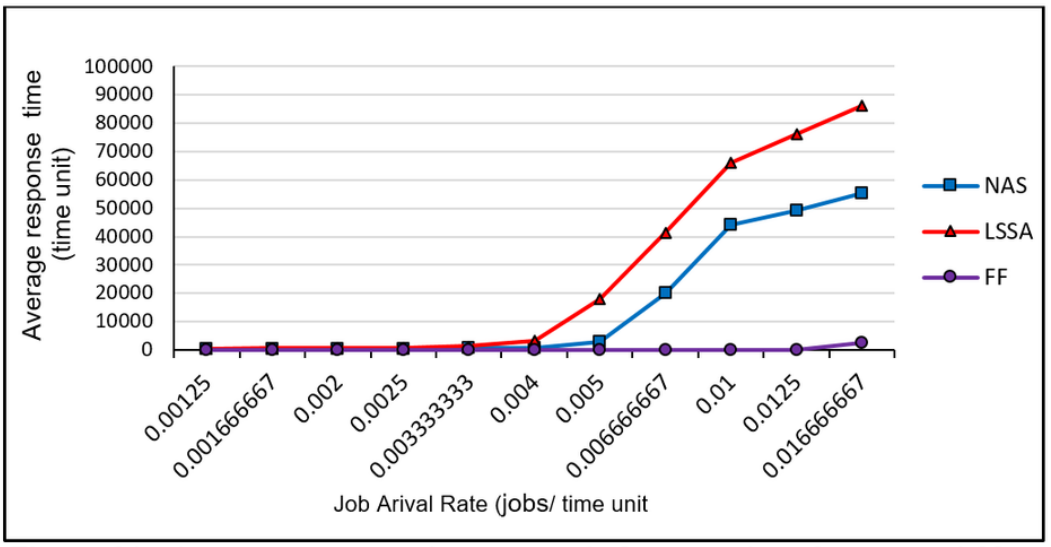
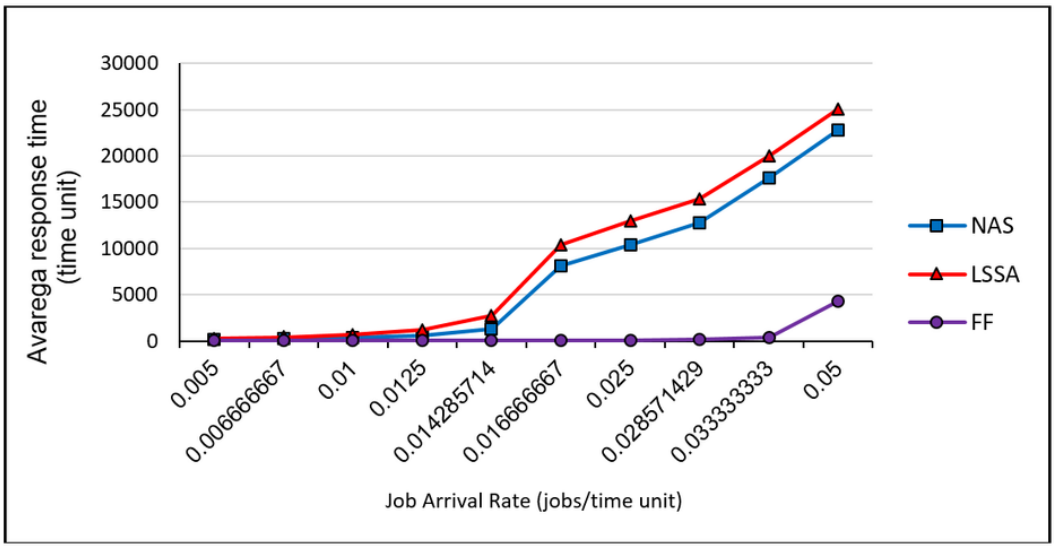


Figure 25. Average response time vs. arrival rate using the near-neighbor communication pattern and uniform-decreasing distribution for job size in a 16x16 mesh



to alleviate the communication overhead. In NAS, the job request can be allocated to several small submeshes, while having a large degree of contiguity among them. In the first step, the nucleus submesh is constructed, and then the remaining required number of processors are allocated as neighbors to the nucleus submesh or other processors allocated to the job.

Simulation experiments have been conducted for comparing FF, LSSA and NAS. The results show that the performance of NAS, when system utilization is considered, is overall better than the

other allocation schemes. This means that NAS has superior ability to overcome fragmentation and alleviate communication overhead.

Overall, the results also show that NAS, when considering average response times, is better than that of all other schemes. On the other hand, when All-to-All and Near Neighbor communication patterns are used, the contiguous FF allocation scheme has superior performance over the non-contiguous allocation schemes considered. This is because contiguous allocation assigns a single submesh to job requests, and this decreases inter-job interference and communication distances. However, NAS is better than LSSA because of its ability to maintain more contiguity.

As a continuation of this research in the future, it would be interesting to study the performance of the proposed NAS allocation strategy with different scheduling schemes, such as window-based (Ababneh & Bani-Mohammad, 2011). It would be also interesting to assess the suggested NAS allocation strategy using other performance metrics, such as power consumption.

REFERENCES

Ababneh, I., & Bani-Mohammad, S. (2011). A New Window-Based Job Scheduling Scheme for 2D Mesh Multicomputers. *Journal of Simulation Modelling Practice and Theory*, *19*(1), 482–493. doi:10.1016/j.simpat.2010.08.007

Ababneh, I., & Bani-Mohammad, S. (2022). An Efficient Maximal Free Submesh Detection Scheme for Space-Multiplexing in 2D Mesh-Connected Manycore Computers. *International Journal of Computers and Their Applications*, *29*(4), 257–268.

Ababneh, I., Bani-Mohammad, S., & Ould Khaoua, M. (2010). All Shapes Contiguous Submesh Allocation for 2D Mesh Multicomputers, *International Journal of Parallel. Emergent and Distributed Systems*, *25*(5), 411–422. doi:10.1080/17445760903546188

Agyeman, M. O., Ahmadinia, A., & Bagherzadeh, N. (2018). Energy and performance-aware application mapping for inhomogeneous 3D networks-on-chip. *Journal of Systems Architecture*, *89*, 103–117. doi:10.1016/j.sysarc.2018.08.002

Al Abass, A., Bani-Mohammad, S., & Ababneh, I. (2022). Performance Evaluation of Contiguous and Noncontiguous Processor Allocation Based on Common Communication Patterns for 2D Mesh Interconnection Network. *International Journal of Cloud Applications and Computing*, *12*(1), 1–21. doi:10.4018/IJCAC.295239

Bani-Mohammad, S., & Ababneh, I. (2013). On the performance of non-contigous allocation for common communication patterns in 2D mesh-connected multi-computers. *J. Simulation Model. Pract. Theory*, *32*, 155–165. doi:10.1016/j.simpat.2011.08.007

Bani-Mohammad, S., & Ababneh, I. (2018). Improving system performance in non-contiguous processor allocation for mesh interconnection networks. *Simulation Modelling Practice and Theory*, *80*, 19–31. doi:10.1016/j.simpat.2017.10.001

Bani-Mohammad, S., Ould Khaoua, M., Ababneh, I., & Mackhenzie, L. M. (2007). An Efficient Processor Allocation Strategy that Maintains a High Degree of Contiguity among Processors in 2D Mesh Connected Multicomputers, *ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2007)*, (pp. 934-941). IEEE. doi:10.1109/AICCSA.2007.370743

Chuang, P., & Tzeng, N. (1994). Allocating precise submeshes in mesh connected systems. *IEEE Transactions on Parallel and Distributed Systems*, *5*(2), 211–217. doi:10.1109/71.265948

Dahir, N., Karkar, A., Palesi, M., Mak, T., & Yakovlev, A. (2021). Power density aware application mapping in mesh-based network-on-chip architecture: An evolutionary multi-objective approach. *Integration (Amsterdam)*, *81*, 342–353. doi:10.1016/j.vlsi.2021.08.008

Ding, J., & Bhuyan, L. (1993). An Adaptive Submesh Allocation Strategy for Two Dimensional Mesh Connected Systems, *Proceedings of the 1993 International Conference on Parallel Processing,* (*vol. 2*, pp. 193-200). doi:10.1109/ICPP.1993.39

Gratz, P., Kim, C., Sankaralingam, K., Hanson, H., Shivakumar, P., Keckler, S., & Burger, D. (2007). On-chip interconnection networks of the TRIPS chip. *IEEE Micro*, *27*(5), 41–50. doi:10.1109/MM.2007.4378782

Gupta, V., & Jayendran, A. (1996). A flexible processor allocation strategy for mesh connected parallel systems. *In Proceedings of the International Conference on Parallel Processing*, (pp. 166–173). doi:10.1109/ICPP.1996.538572

Kim, G., & Yoon, H. (1998). 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*(2), 175–185. doi:10.1109/71.663881

Lahdhiri, H., Lorandel, J., Monteleone, S., Bourdel, E., & Palesi, M. (2020). Framework for Design Exploration and Performance Analysis of RF-NoC Manycore Architecture. *Journal of Low Power Electronics and Applications*, *10*(4), 37. doi:10.3390/jlpea10040037

Li, K., & Cheng, K.-H. (1991). A two dimensional buddy system for dynamic allocation in a partitionable mesh connected system. *Journal of Parallel and Distributed Computing*, *12*(1), 79–83. doi:10.1016/0743-7315(91)90032-5

Lo, V., Windisch, K., Liu, W., & Nitzberg, B. (1997). Non-contiguous processor allocation algorithms for mesh-connected multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, *8*(7), 712–726. doi:10.1109/71.598346

Matsutani, H., Koibuchi, M., & Amano, H. (2007). Tightly-coupled multi-layer topologies for 3-D NoCs, *In 2007 International Conference on Parallel Processing, ICPP [4343882]*.

Mosayyebzadeh, A., Amiraski, M. M., & Hessabi, S. (2016). Thermal and power aware task mapping on 3D Network on Chip. *Computers & Electrical Engineering*, *51*, 157–167. doi:10.1016/j.compeleceng.2015.12.001

ProcSimity v4.3. (1996). *ProcSimityv4.3 User's Manual. University of Oregon*.

Seo, K., & Kim, S. (2003). Improving system performance in contiguous processor allocation for mesh-connected parallel systems. *Journal of Systems and Software*, *67*(1), 45–54. doi:10.1016/S0164-1212(02)00086-9

Vangal, S., Howard, J., Ruhl, G., Dighe, S., Wilson, H., Tschanz, J., Finan, D., Iyer, P., Singh, A., Jacob, T., Jain, S., Venkataraman, S., Hoskote, Y., & Borkar, N. (2007). An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS, *IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, 98-589. doi:10.1109/ISSCC.2007.373606

Wentzlaff, D., Griffin, P., Hoffmann, H., Liewei, B., Edwards, B., Ramey, C., Mattina, M., Miao, C.-C., Brown, J. F., & Agarwal, A. (2007). On-Chip Interconnection Architecture of the Tile Processor. *IEEE Micro*, *27*(5), 15–31. doi:10.1109/MM.2007.4378780

Zhu, Y. (1992). Efficient processor allocation strategies for mesh-connected parallel computers. *Journal of Parallel and Distributed Computing*, *16*(4), 328–337. doi:10.1016/0743-7315(92)90016-G

Abeer Shdefat received the BS degree in Computer Science from the University of Jordan, Jordan in 2004, the MS degree in computer science from Al al-Bayt University, Jordan in 2019. She is presently working as a head section of the Administrative Applications Section, at the Hashemite University in Jordan. Her research interests include processor allocation and job scheduling in multicomputers.

Saad Bani-Mohammad received the BSc degree in computer science from Yarmouk University, Jordan in 1994, the MSc degree in computer science from Al al-Bayt university, Jordan in 2002, and the PhD degree in computer science from University of Glasgow, U.K., in 2008. From 2002 to 2005, he was a lecturer in the Department of Computer Science at Al al-Bayt University in Jordan. Prof. Bani-Mohammad served as a Head of Computer Science Department for 5 years (2008-2013) at Al al-Bayt University, and a Deputy Dean of the IT College at Al al-Bayt University for one year (2013-2014), and then he served as a Dean of Prince Hussein Bin Abdullah College for Information Technology at Al al-Bayt University for 6 years (2015-2020). Prof. Bani-Mohammad is presently a President's Assistant of Accreditation and Quality Assurance Commission for Higher Education Institutions (AQACHEI) in Jordan and a Professor of Computer Science in the Department of Computer Science at Al al-Bayt University, Jordan. He is a member of IEEE Computer Society. His research interests include processor allocation and job scheduling in multicomputers and E-learning. Prof. Bani-Mohammad has over 40 scientific papers and projects either presented or published. Most of his research was supported by Al al-Bayt University, Jordan and University of Glasgow, U.K. His findings were published (over 40 publications) in world leading journals and also in prestigious and top quality international conference proceedings.

Ismail Ababneh received his degree of Electro-mechanical Engineer from the National Superior School of Electronics and Electro-mechanics of Caen (The ENSI Caen at present), France in 1979. He received the MS degree in software engineering from Boston University in 1984, and the PhD degree in computer engineering from Iowa State University in 1995. From 1984 to 1989, he was a software engineer with DAS, Boston, Massachusetts. From 2007 to 2010, he was a visiting associate professor in the Department of Computer Science at Jordan University of Science and Technology. He is currently a full professor in the Department of Computer Science at Al al-Bayt University, Jordan, where he has been a faculty member since 1995. Also, he is a member of Tau Beta Pi and Eta Kappa Nu. He has worked as department head, dean of information technology college, dean of research, and vice president for administration at Al al-Bayt University. He has played a key/leading role in several large and small projects. He published thirty five research papers in journals and has over thirty papers and presentations in conferences and workshops. His main research interests are processor allocation in multicomputers and manycore systems, and routing algorithms for mobile ad hoc networks.