An Efficient Maximal Free Submesh Detection Scheme for Space-Multiplexing in 2D Mesh-Connected Manycore Computers

Ismail Ababneh* and Saad Bani-Mohammad*
Al al-Bayt University, Mafraq 25113, JORDAN

Abstract

For multicore systems, space-sharing (space-multiplexing) is a promising core allocation strategy as the number of cores grows into the hundreds and thousands because it can achieve scalability and good system performance. In space-multiplexing, an application is allocated its own set of cores for the duration of its execution. In this paper, we propose a new efficient maximal free submesh detection scheme for two-dimensional mesh-connected manycore systems. Free submeshes that are not contained in other free submeshes are detected and placed in a free-list for direct support of space-multiplexing. An advantage of the proposed scheme is that its time complexity is quadratic in the number of free submeshes, whereas the time complexity of the previous such scheme is cubic in this number. In addition to complexity analysis, detailed simulations are carried out to evaluate the proposed scheme. In the simulations, we consider several approaches to selecting allocation submeshes from the free-list. The approaches range from a promising first-fit variant to a scheme that aims to keep large free submeshes for future allocation requests. The results show that the proposed scheme is substantially more time-efficient than the previous cubic recognition-complete maximal free submesh scheme. It achieved up to seventy percent reduction in the average combined allocation and de-allocation times in these simulations.

Key Words: Manycore systems, mesh interconnection network, space-sharing (space-multiplexing), maximal free submesh, contiguous submesh allocation.

1 Introduction

Over the past two decades, there has been, mainly because of the power wall, a shift from increasing the clock frequency of single-thread microprocessors to multicore processors, where several cores or processing units are built on a single chip. As per Moore's law, it has also been possible to incorporate many cores and build manycore processors, which are multicore systems with many relatively simple cores that can support high explicit parallelism.

For communication among the cores in manycore systems, a Network-on-Chip (NoC) architecture is used. This is to avoid the bottleneck problem that the bus interconnection architecture suffers from when the number of cores becomes large. The mesh is a popular NoC interconnection topology, in both its two-dimensional (2D) and three-dimensional (3D) forms [16, 17, 24]. An example is the 2D 8×10 mesh network of an 80-core Intel manycore research chip, where each core or Processing Element (PE) is associated with a 5-port router for communication [23]. Four ports are used for communicating with the four neighbors of internal cores. Cores on mesh corners and on its edges have fewer neighbors. The fifth port is for NoC-PE communication. Another example is the TRIPS OCN that has a 4x10 wormhole-routed 2D mesh interconnection network [11]. A recent 1024-node manycore system has the mesh topology, and comprises 32 clusters, where a cluster is a 4×8 mesh. To decrease the system's diameter, each cluster has in its middle a Radio Hub (RH) that communicates with the four 2D routers of the middle cluster node. Communication among clusters takes place via the RHs. Within clusters, communication uses the mesh interconnection network [13]. In [17], a NOC that combines ring and 2D mesh interconnections and adapts to application requirements is proposed for improved scalability and energy efficiency up to 1024 processing elements.

Several studies indicate that space-multiplexing (space-sharing) is a promising core allocation strategy for manycore systems, as it can achieve scalability and good performance for large core numbers [22, 25]. By allocating resources spatially, traditional time multiplexing scheduling is transformed into a layout and partitioning problem, where jobs or applications, including possibly the OS, run on their own sets of cores. In space-sharing, a job can be executed on a submesh of the manycore system, which can reduce communication distances, interference among jobs, message delays, energy consumption and chip temperatures. Supporting this, previous studies have shown that mapping the communicating tasks of a parallel job to neighboring cores, in particular the cores of a single submesh, can reduce communication delays and power consumption, and improve throughput and job execution times [5, 8, 18].

Based on this, we assume that allocation is contiguous, where a job requests and is allocated a single submesh of a width and a height that the job specifies. By running on a single submesh, the job can achieve reduced distances among the cores it is allocated. Also, a space-sharing allocation policy is required to achieve high system utilization. It must be capable of detecting all free core submeshes and should aim to maximize the number of allocated cores (i.e., minimize core fragmentation) [9, 15, 26]. In addition, it must be time-efficient, especially as the system size (i.e., number of cores) of the manycore system grows. Migratory space-sharing policies that carry out defragmentation of scattered submeshes that result when jobs exit the system have been considered. An issue in these policies is when to carry out

* Computer Science Department, Prince Hussein Bin Abdullah Faculty of Information Technology. Emails: ismael@aabu.edu.jo and bani@aabu.edu.jo.

ISCA Copyright© 2022

defragmentation and the associated job migrations, the cost of migration, and application size constraints imposed by some of these policies to simplify defragmentation [19, 20]. Such constraints can result in internal fragmentation.

Several contiguous space-sharing allocation policies were proposed for multicomputers with 2D mesh interconnection topologies. Several of these policies do not scale well. They have time complexities that grow with the system size (its number of PEs) [7, 10, 26, 27]. However, more efficient policies that build lists of free submeshes and/or allocated submeshes were proposed. The allocation decisions they make are based on the elements in these lists, and their time complexities are functions of list sizes [1, 6, 9, 12, 14]. A major advantage of such policies is that list sizes can be much smaller than system sizes [1, 3].

An advantage of submesh allocation that is based on *free* submeshes is its flexible submesh selection. When multiple large-enough free submeshes exist, they all can readily and simply be considered for allocation to the current request, which can lead to superior allocation. A free-list allocation policy that can detect all *maximal* free submeshes in a 2D mesh system has been proposed [12]. However, the submesh detection algorithm it uses is not efficient. It has time complexity that is cubic in the number of free submeshes. A free submesh is *maximal* if it is not contained in another free submesh. More efficient algorithms for building the free-list have been proposed [1, 14]. However, they are not recognition-complete. They fail to detect some available submeshes, which can result in unsuccessful allocation to the current allocation request although there is in fact at least one large-enough free submesh for satisfying this request.

In this paper, we propose an efficient recognition-complete maximal free submesh detection scheme for 2D mesh-connected manycore systems. An advantage of this scheme over the previous recognition-complete scheme, proposed in [12], 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, we evaluated and compared these two detection schemes. In these simulations, various previous promising policies for deciding where allocation will take place are considered. They range from a promising first-fit variant [10] to a policy that aims to keep large free submeshes for future allocation using a reservation function [12], and a policy that gives priority to mesh corner then peripheral allocation [3]. Corner and peripheral placements have for goal leaving large free submeshes for future allocation. The simulation results show that when allocation and de-allocation times are considered, the proposed submesh detection scheme substantially outperforms the previous maximal free submesh detection scheme. It has achieved up to seventy percent improvement in the measured combination of these times.

We limit our attention here to 2D meshes; however, this work can be adapted for 3D meshes. This paper is organized as six sections. Section 2 below contains a few preliminaries. Section 3 has a review of related schemes. The proposed detection scheme is defined and analyzed in Section 4. The system model, and simulation parameters and results are in Section 5. Finally, Section 6 contains the research conclusions.

2 Preliminaries

The target manycore system, M(W, H), is of width W and height H. A core is denoted by the coordinates (i, j), where $1 \leq i \leq W$ and $1 \leq j \leq H$. Cores are interconnected by bidirectional communication links as shown in Figure 1. The size of the mesh is the number of its cores, N, where N=W*H.

A w × h submesh is represented by a 4-tuple (i1, j1, i2, j2), where (i1, j1) represent its base node, and (i2, j2) its end node. We have that w = i2 – i1 + 1 and h = j2 – j1 + 1. In Figure 1, (1, 3, 3, 4) represents the 3 × 2 submesh S_1.

A submesh is said to be free if none of its cores is allocated. As indicated previously, a free submesh is said to be *maximal* if it is not contained in another free submesh. Also, a submesh is said to be busy or allocated if all its cores are allocated to the same job. In Figure 1, the maximal free submeshes are (1, 1, 1, 4), (1, 3, 5, 4), and (3, 1, 5, 4). For example, (3, 1, 5, 2) is not maximal as it is contained in (3, 1, 5, 4). If the cores (2, 1) and (2, 2) in Figure 1 are allocated to the same job, then (2, 1, 2, 2) is a busy or allocated submesh.

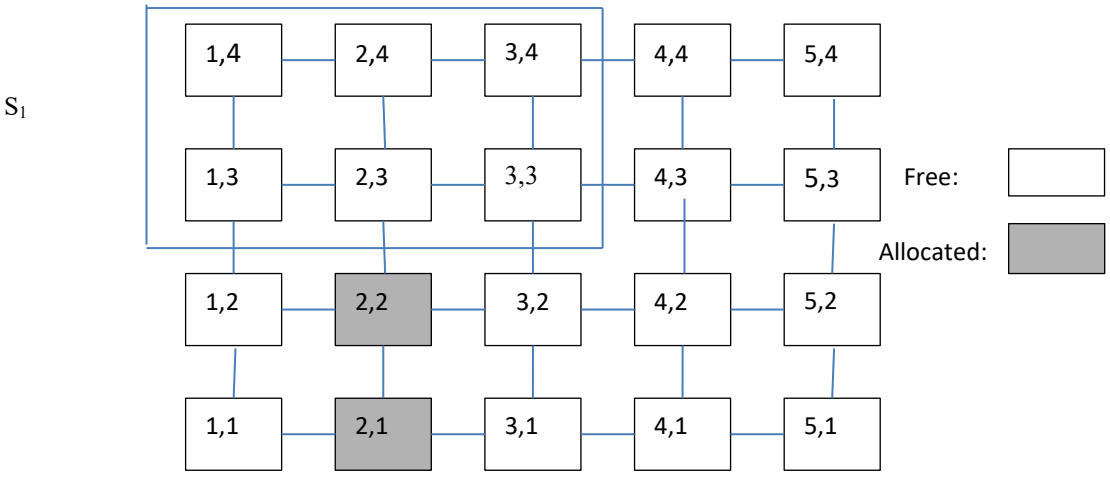

Figure 1. M(5, 4) 2D mesh

3 Previous Works

Several contiguous allocation policies based on free and/or busy submeshes have been proposed for 2D meshes. They differ in how they detect free submeshes, how they select a free submesh for allocation, and where the allocated submesh is located within the free submesh selected. A major aim of a contiguous allocation policy should include reducing external fragmentation. That is, reducing the number of free cores that remain idle although they are sufficient in number to satisfy the allocation request of the job that the scheduling algorithm has selected for execution. For instance, a policy may fail to detect some of the free submeshes, and it may make some poor placement choices that result in a relatively large number of small free submeshes, rather than fewer large ones. In what follows, N is the size of the target manycore system, and f and b are the numbers of free and busy submeshes, respectively. We also assume that a job's allocation request upon arrival is for an $\alpha \times \beta$ free submesh, where the width and height of the submesh requested are α and β.

3.1 Busy-List with Global Adjacency (BLGA)

This policy [9] uses a list of allocated processors. It attempts to satisfy the allocation request in its $\alpha \times \beta$ or $\beta \times \alpha$ orientations using a submesh that has the largest number of adjacent busy cores and mesh boundary cores. The rationale is that this can reduce fragmentation. The time complexity of allocation in BLGA is in $O(b^3)$, and that of de-allocation is in $O(1)$. In [28], a BLGA improvement that aims to allocate adjacent submeshes to requests served close in time is proposed. However, such heuristic assumes that jobs that start execution close in time tend to have close exit times.

3.2 Free-List with First-Fit Adjacency (FLFFA)

This scheme [14] builds a free-list that approximates the maximal free list (MFL), and a busy-list. The free-list elements are sorted in the non-decreasing order of their shorter edge. The first free submesh that can accommodate the current allocation request as $(\alpha \times \beta)$ or $(\beta \times \alpha)$ is the allocation candidate. Allocation for both orientations is considered in the corners of the candidate submesh, and a placement with the maximum number of adjacent busy cores and mesh peripheral cores is the allocation submesh. The number of adjacent busy processors is computed using the busy-list. FLFFA was compared to BLGA using simulations, where it achieved superior waiting delays. Also, FLFFA outperformed BLGA overall in another simulation study [12].

The allocation and de-allocation operations have $O(f^2)$ time complexities. In [1], an $O(f)$ heuristic for approximating MFL has been proposed. In both heuristics, building the free-list starts with expanding, if possible, the released submesh into the current elements of the free-list when a job terminates. This step produces new free submeshes. Then, expansions of the current elements into the new submeshses are attempted. The expanded and new submeshes are used in forming the new free-list. A problem is that this approach is not recognition-complete, as can be seen in Example 1 below.

Example 1. In Figure 2, if the job running on (2, 1, 2, 4) terminates, the released submesh cannot be expanded into the existing free submeshes ({(1, 1, 1, 2), (3, 1, 5, 2)}), and the new free submesh is {(2, 1, 2, 4))}. Then, (1, 1, 1, 2) is expanded into (2, 1, 2, 4) to produce the free submesh (1, 1, 2, 2). Likewise, expanding (3, 1, 5, 2) into (2, 1, 2, 4) produces (2, 1, 5, 2). Thus, (1, 1, 5, 2) is not detected, and the detected submeshes (1, 1, 2, 2) and (2, 1, 5, 2) are not maximal as they are proper submeshes of the undetected maximal free submesh (1, 1, 5, 2).

3.3 Reservation Best-Fit (RBF)

This policy [12] builds the MFL and sorts its elements in their non-increasing size order. The submesh size is the number of cores it contains. The policy also builds a busy list. The MFL is scanned for large enough free submeshes. In these submeshes, all possible $\alpha \times \beta$ and $\beta \times \alpha$ submesh corner placements are considered candidate submeshes. A reservation function is employed for computing leftover free submeshes with the aim of preserving large free submeshes for later use. Using simulations, RBF outperformed FLFFA

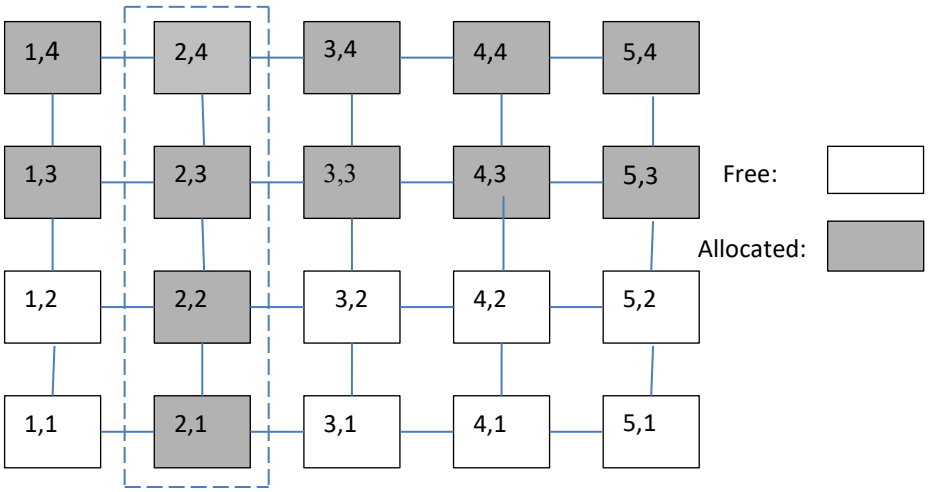

Figure 2. An example illustrating incomplete recognition

and BLGA in terms of average job waiting delays [12]. The time complexity for building MFL upon a job exit is in $O(f^3)$, and that needed upon an allocation operation is in $O(f^2)$. Both use submesh subtraction operations, instead of expansions.

3.4 Right Border Line (RBL)

This strategy [6] keeps a busy-list and uses it to determine the allocation right border lines. These lines consist of nodes that can serve as bases for the current allocation request. By construction, the lines either have to their left an allocated submesh or they are the left boundary of the mesh itself. A policy that re-builds the RBLs and looks for an allocation RBL for $\beta \times \alpha$ when none is found for $\alpha \times \beta$ was also proposed. In both cases, the upper end core of the first allocation RBL to be found is chosen as base node for the allocated submesh. The allocation time of RBL is in $O(b^2)$, and that of de-allocation is in O(1). Using simulations, the performance of the orientation-switching RBL policy was evaluated and compared to the performance of BLGA and several other allocation policies. The policies considered produced almost similar average job waiting delays, with BLGA performing slightly better than the others [6].

4 Proposed Submesh Detection Scheme

In the free submesh detection scheme proposed in this paper, the maximal free submeshes are found and form an unordered *MFL*. Initially, this list contains the entire mesh. It is then reconstructed after each job departure (i.e., submesh release), and each allocation.

4.1 Detection of Maximal Free Submeshes upon De-allocation

When a job terminates and the submesh it is allocated is released, an attempt is made first to expand the released submesh into the current elements in *MFL* using two patterns. The first is a horizontal-vertical expansion, and the second is a vertical-horizontal expansion. In horizontal expansion, a submesh may expand into a free submesh located to its left or right. In vertical expansion, the expanded into submesh may be above or below the expanding submesh. At most two new different submeshes can result from these expansions. The first is generated by the horizontal-vertical expansion, and the second is generated by the vertical-horizontal expansion. If expansion is not possible, the only new submesh is the released submesh itself. In all cases, if an expanding submesh covers an expanded into submesh after the expansion, the latter is removed from *MFL* because its free cores are covered. The expansions of the released submesh into the elements of *MFL* are of the complete type, as defined below [1].

Definition 1. A free submesh (i1, j1, i2, j2) is *completely* expandable right into an adjacent free submesh (i3, j3, i4, j4) if $i2 \geq i3 - 1$, $i2 < i4$, $i1 < i3$, $j2 \leq j4$, and $j1 \geq j3$. This expansion turns (i1, j1, i2, j2) into (i1, j1, i4, j2). Complete expansions down, up, and left are defined similarly. If j2 = j4 and j1 = j3, we have that (i3, j3, i4, j4) $\subset$ (i1, j1, i4, j2). When (i3, j3, i4, j4) is an element of *MFL*, it is removed because its free cores are in (i1, j1, i4, j2). Expansions in the remaining directions are handled similarly.

Example 2. In Figure 3, *MFL* = {(4, 3, 5, 4), (1, 3, 5, 3), (3, 1, 3, 3)}. Assume S = (1, 4, 3, 4) is released. The horizontal step of the horizontal-vertical complete expansion of S into the elements of MFL produces (1, 4, 5, 4) by the complete expansion of S into (4, 3, 5, 4), then (1, 4, 5, 4) becomes (1, 3, 5, 4) by its complete expansion into (1, 3, 5, 3) in the vertical step and (1, 3, 5, 3) is removed.

As a result of the release of a submesh and its expansions, it may be possible to expand *MFL* elements completely or partially (see Definition 2 below) into the new submeshes to produce larger or additional free submeshes. This expansion is conducted next. If an edge of a submesh in *MFL* is partially or completely covered by or borders a new submesh, the submesh or parts of it are extended to the other end of the new submesh. An element in *MFL* that is covered by one of the new submeshes before or after expansion is removed from this list.

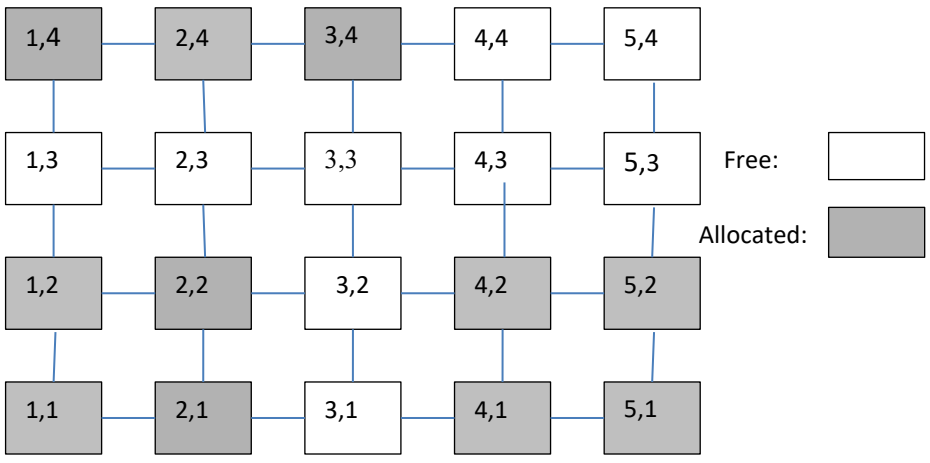

Figure 3: Submesh release example

Example 3. Continuing with Example 2. The new free submesh is (1, 3, 5, 4), and the submeshes that remain in *MFL* are (3, 1, 3, 3) and (4, 3, 5, 4). The submesh (4, 3, 5, 4) is covered by (1, 3, 5, 4), therefore it is removed. The submesh (3, 1, 3, 3) is expanded completely into (1, 3, 5, 4), producing the maximal free submesh (3, 1, 3, 4).

Definition 2. The partial expansion of a free submesh into a second one is the horizontal and/or vertical expansion of the largest subpart(s) of this free submesh into the second one.

Example 4. If the submesh (5, 4, 6, 5) in Figure 4 is released, the part (5, 1, 5, 3) of (4, 1, 5, 3) is expanded partially into (5, 4, 6, 5), producing (5, 1, 5, 5). Also, (5, 6, 5, 6) is completely expanded into (5, 4, 6, 5) to produce (5, 4, 5, 6).

It can be noticed that the free submesh (5, 1, 5, 6) in Example 4 is not detected by the expansions considered so far. Partial and complete expansions across the released submesh are needed. To carry out such expansions, the free submeshes must be processed further for possible inter-expansions. For example, (5, 1, 5, 5) can be expanded completely into (5, 4, 5, 6) to produce the maximal free submesh (5, 1, 5, 6). In this expansion, the expanded into submesh is removed because it is covered by the result of the expansion. In all cases, a free submesh is removed and does not appear in the final *MFL* if it is covered by any other free submesh.

Another reason for missing maximal free submeshes by the expansions considered so far is the existence of free submeshes around a corner of the released submesh or its expansions. A maximal free submesh is missed if the sides of corner free submeshes are shorter than the sides they face in the released submesh or its expansion. For example, if the submesh (2, 2, 3, 4) is released in Figure 4, then the partial expansions of (1, 1, 1, 3) and (1, 1, 2, 1) into this submesh will produce (1, 2, 3, 3) and (2, 1, 2, 4). However, (1, 1, 2, 3) is not detected. To detect this maximal free submesh, (1, 1, 2, 1) should be expanded into (1, 2, 3, 3). Finally, once these additional expansions are carried out the detected free submeshes should be processed for coverage. In the current example, (1, 1, 2, 3) covers (1, 1, 1, 3), therefore the latter submesh is removed. The complete de-allocation algorithm is in Figure 5.

For analyzing this de-allocation algorithm, it can be seen that the number of possible complete expansions in Step 2 is in O(f). Also, because of the generation of $S1$ and $S2$ the number of free submeshes at the end of Step 2 is at most $f + 2$. As a result of the partial and complete expansions in Steps 3 and 4, the number of known free submeshes at the end of these steps is also in O(f). The number of tests needed for coverage detection and processing in Step 6 is in O(f^2). After this coverage detection, the number of free submeshes detected is in O(f) because only a subset of maximal free submeshes has been detected so far. Consequently, the number of operations needed in Step 7 is in O(f^2).

Upon completion of Step 7, the maximal free submeshes that are generated by the partial and complete expansions across the released submesh are detected. Also, are detected the additional maximal free submeshes associated with the corner free submeshes. Intuitively, the number of additional maximal free submeshes detected in Step 7 is in O(f), and the number of tests needed for coverage detection and processing in Step 8 is in O(f^2). It is easy to see that Steps 1 and 5 take constant time. Hence, the complexity of this algorithm is in O(f^2).

4.2 Detection of Maximal Free Submeshes upon Allocation

Upon allocation, a selection scheme that determines the submesh where allocation will take place, such as first-fit, is employed. Then, where allocation takes place in this submesh is determined. This placement could, for example,

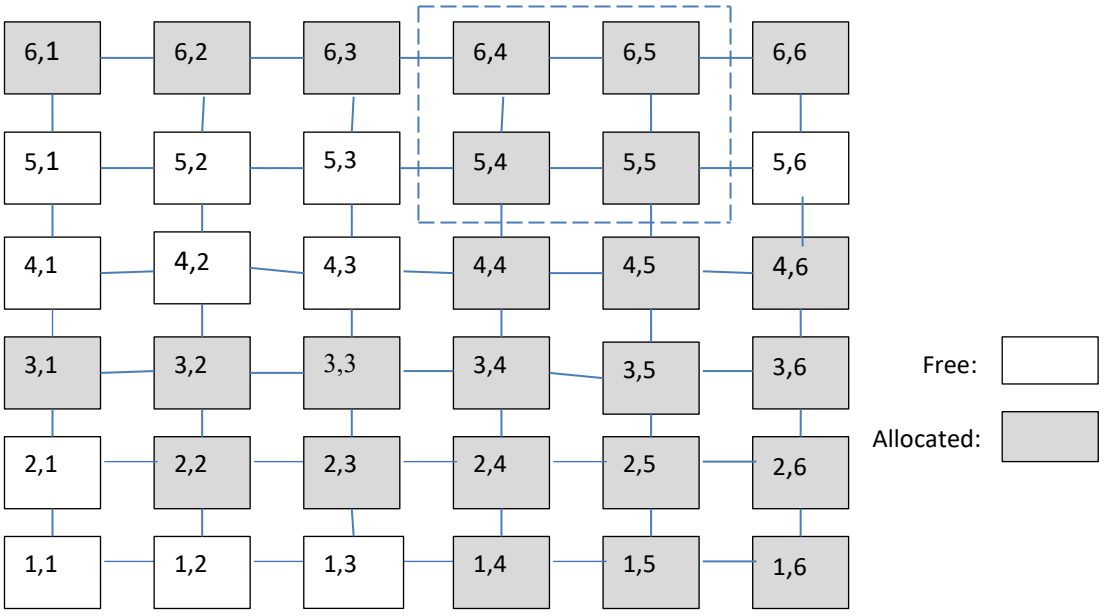

Figure 4: Submesh release example

Procedure de-allocate(S) /* An allocated submesh S is released */
Step 1) *num_free_cores* += size(S) /* update the number of free cores */
S_1=S; S_2=S
Step 2) Completely expand S_1 horizontally into the elements of *MFL*
Completely expand S_2 vertically into the elements of *MFL*
Completely expand S_1 vertically into the elements of *MFL*
Completely expand S_2 horizontally into the elements of *MFL*
$R = S_1$
Step 3) for each submesh F in *MFL* {
if F is outside R and they are not adjacent go to next F
else if $F \subseteq R$ remove F from *MFL*
else if no node in F is adjacent to a node in S go to next iteration of this loop else if complete expansion of F into R is possible {
completely expand F into R (down, right, up, or left)
if $R \subseteq F$ then $R = F$ and remove F from *MFL*
}
else if partial expansion from F into R is possible
form resulting fragments and add them at the head of a temporary list *TL*
else completely expand R into F (up, right, left, or down) if possible
}
Step 4) repeat Step 3) for $R = S_2$ if $S_2 \neq S_1$
Step 5) Append *TL* at the head of *MFL* to form the list *FL*: *FL* = *TL* + *MFL*
TL = ∅
Step 6) /* Remove *FL* elements that are non-maximal: */
for each element S_i in *FL*
for each element S_j that is after S_i in *FL*
if ($S_j \subseteq S_i$) remove S_j from *FL*
else if ($S_i \subseteq S_j$) mark S_i for removal before going on to the next S_i
Step 7) /* Carry out expansions across the released submesh and around corners */
Carry out all additional complete and partial expansions among *FL* elements
Add the fragments that result from partial expansions at the head of *TL*
Step 8) *FL* = *TL*+*FL*; Remove non-maximal *FL* elements; *MFL* = *FL*; *TL* = ∅
} /* end of procedure de-allocate */

Figure 5: The de-allocation algorithm

be in the lower-left or lower-right corner of the allocation submesh. Then, *MFL* is rebuilt. The allocation submesh is removed from *MFL* and the fragments that result from the subtraction of the allocated submesh from it are added at the head of a temporary list, *TL*. Also, the allocated submesh is subtracted from overlapping *MFL* elements, and the results are added at the beginning of *TL*. Finally, *TL* is appended at the head of *MFL*. The list that results is scanned, and a submesh in this list is removed if it is covered by another element in the list. Thus, the elements that remain in the list are maximal, and they constitute the new *MFL*. The allocation algorithm is given in Figure 6.

/* Current job requests the allocation of an $\alpha \times \beta$ submesh */
Procedure allocate (α, β){
Step 1) if *num_free_cor* < $\alpha\beta$ return Failure
Step 2) Select an allocation submesh S from *MFL*, and position the allocated submesh A within S
if no S is found return Failure
Step 3) Remove S from *MFL*
Step 4) Subtract A from S
Step 5) Add fragments that result from the subtraction at the head of a temporary list *TL*
Step 6) for each submesh S_i in *MFL*
if S_i overlaps with A{
Remove S_i from *MFL*
Subtract the overlapping part $A \cap S_i$ from S_i
Add the resulting fragments at the head of *TL*
}
Step 7) Append *TL* at the head of *MFL* producing a list *FL*
Step 8) Remove *FL* elements that are non-maximal
Step 9) *num_free_cores* = *num_free_cores* - $\alpha\beta$; *MFL* = *FL*; return Success
} /* end of procedure allocate */

Figure 6: Allocation algorithm

The subtraction operation used in the allocation algorithm is one that produces maximal difference submeshes. For example, subtracting (1, 1, 2, 2) from (1, 1, 5, 4) in Figure 7 yields the fragments (3, 1, 5, 4) and (1, 3, 5, 4). The subtraction of (3, 2, 4, 3) from (1, 1, 5, 4) produces the four difference submeshes (1, 1, 5, 1), (1, 1, 2, 4), (1, 4, 5, 4), and (5, 1, 5, 4), as another example.

Example 5. This example illustrates how allocation works. Assume a free system, and a request for a 2 × 2 submesh arrives. Initially, *MFL* consists of the whole mesh (1, 1, 5, 4). If the request is allocated (1, 1, 2, 2), then (1, 1, 5, 4) is removed from *MFL*, and the subtraction of (1, 1, 2, 2) from (1, 1, 5, 4) yields the fragments (3, 1, 5, 4) and (1, 3, 5, 4), which are added at the head of *TL*. Then, *TL* is appended at the head of *MFL* to produce *FL* = {(3, 1, 5, 4), (1, 3, 5, 4)}. This is the final *MFL* because all its elements are maximal. If a 4 × 2 allocation request arrives, the allocation selection algorithm may choose the allocation submesh S = (1, 3, 5, 4) and allocate A = (1, 3, 4, 4). In this case, (1, 3, 5, 4) is removed from *MFL*, and the subtraction of A from S produces the fragment (5, 3, 5, 4), which is added to a new *TL*. Then A *is* subtracted from (3, 1, 5, 4), yielding the fragments (3, 1, 5, 2) and (5, 1, 5, 4), which are added to *TL*. The submesh (3, 1, 5, 4) is removed from *MFL*. Finally, (5, 3, 5, 4) is removed because it is covered by (5, 1, 5, 4). The final *MFL* is {(3, 1, 5, 2), (5, 1, 5, 4)}.

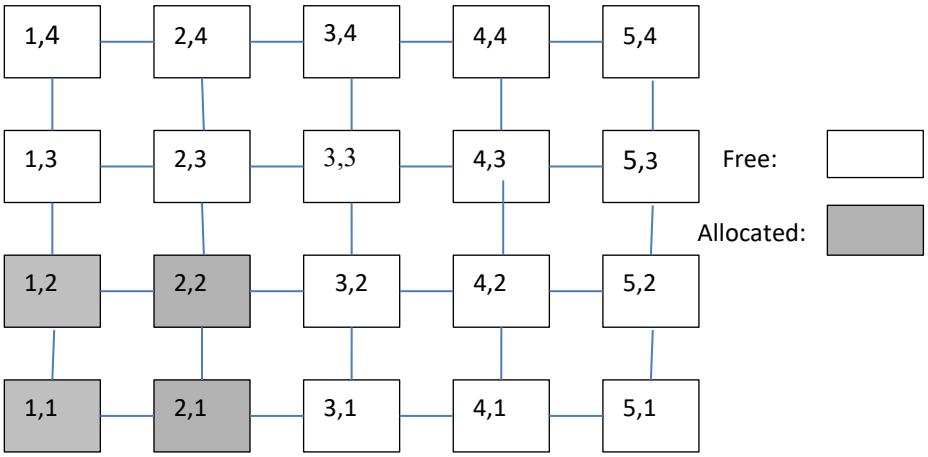

Figure 7: Subtraction and allocation example

Analyzing the allocation algorithm, we assume that a scheme that can select an allocation submesh in $O(f^2)$ time is used. The first-fit is an example of such schemes as it requires $O(f)$ steps for this selection. The number of fragments that results from subtracting the allocated submesh from the free submeshes in Step 6 is in $O(f)$; their number is at most $4f$ as the subtraction operation of a 2D submesh from another 2D submesh results in at most four submeshes. Therefore, the number of operations in Step 8 and the complexity of the algorithm are in $O(f^2)$.

4.3 Selection of Allocated Submeshes

A comparison of several policies for selecting where allocation takes place when the maximal free submesh detection scheme proposed in [12] is used can be found in an earlier work [2]. For the comparison of the maximal free submesh detection scheme that we propose to that proposed in [12], the following promising schemes for determining where allocation takes are considered:

4.3.1 Switching First-Fit (SFF). The first *MFL* element that is large enough for the current $\alpha \times \beta$ request is the allocation submesh, and the $\alpha \times \beta$ submesh in its lower-left corner is allocated for the request. If this fails, first-fit allocation is re-attempted for the $\beta \times \alpha$ orientation. Switching request sides was first proposed in [10], and it has been used in many studies [1, 3, 6, 9, 14, 26].

4.3.2 Maximum Mesh Peripheral Length (MMPL). This policy gives priority to allocating mesh corner submeshes because they have the most peripheral cores. In scanning *MFL*, if there is a corner submesh that is large enough for $\alpha \times \beta$ or $\beta \times \alpha$ request shapes, the requesting job is placed in this mesh corner in the right orientation and scanning is terminated. Any corner placement will have the most peripheral cores. If a large enough submesh in *MFL* has a side aligned with a mesh edge, the peripheral lengths associated with possible $\alpha \times \beta$ and $\beta \times \alpha$ placements are computed. When there is no corner allocation, the first placement with the most peripheral cores is assigned to the request. If no corner or peripheral placement is possible, the request is placed at the base of the first large-enough internal submesh in *MFL* [3]. A generalization of the orientation switching transformation that permits all viable request shapes has also been proposed; when combined with giving preference to allocating peripheral submeshes it resulted in significant system performance improvements [4].

4.3.3 Reservation Best-Fit (RBF). In this scheme, proposed in [12], switching the orientation of requests is also allowed, and the goal of the allocation submesh selection scheme is to leave large free submeshes for future allocation, as was discussed earlier. Also, because our simulations have shown that the system performance of RBF depends on the order of *MFL* elements, we have ordered them as in [12] in the proposed *MFL* detection scheme when it was used with RBF so as to have the same performance as the original proposal.

5 Simulation Results

Simulation was employed for evaluating and comparing the maximal free submesh detection schemes when they were used with the three allocation submesh selection schemes considered. To this end, we implemented the detection and selection schemes in the ProcSimity simulator that we have been adding our proposed scheduling and allocation algorithms to for the last two decades. The original ProcSimity is a C-language tool that was developed initially at the University of Oregon for research in processor allocation and job scheduling for distributed memory multicomputers [21].

As in many previous related works, the 2D mesh system has equal sides of length L [1, 3, 6, 9, 12, 14, 26]. Job interarrival times follow an exponential distribution, and the scheduling algorithm assumed is first-come-first-served. Job execution times follow an exponential distribution with a mean of one time-unit. The side-lengths of allocation requests are generated using two distributions: the uniform over the interval $[1, L]$, and a uniform-decreasing distribution

that uses four probabilities pr_1, pr_2, pr_3 and pr_4, and four side lengths sl_1, sl_2, sl_3, and sl_4. These probabilities are for the α and β of a request to fall within [1, sl_1], [sl_1+1, sl_2], [sl_2+1, sl_3] and [sl_3+1, $s\ l_4$]. The side lengths within a range are distributed uniformly. In this paper, we use pr_1 = 0.4, $pr_2 = pr_3 = pr_4$ = 0.2, $sl_1 = L/8$, $sl_2 = L/4$, $sl_3 = L/2$, and $sl_4 = L$. The distributions adopted here were used in several previous research works [1, 3, 6, 9, 14, 15]. Independent simulation runs are repeated so as to have a 95% confidence level that relative errors do not exceed 5% of the means. In each simulation run, 1000 jobs are executed.

The system performance parameter measured in this study is the *average turnaround time* for all jobs, where a job's turnaround time is the time the job spends in the system. The efficiency of the detection schemes is evaluated using the time taken allocating and de-allocating. This second performance parameter is the main parameter because the two detection schemes are expected to produce *similar* system performance since they are both based on detecting the set of maximal free submeshes and are recognition-complete. In what follows, we denote the policies as <*D*>(<*S*>), where *D* is the detection scheme and *S* is the allocation submesh selection scheme. The proposed MFL detection scheme is denoted as PMFL, and that proposed by Kim and Yoon in [12] is denoted as KYMFL.

We first compare the system performance of the schemes for the workload models assumed. In Figure 8, the average turnaround times are plotted against average job arrival rates for the detection and allocation schemes and the uniform-decreasing size distribution in a 32 × 32 system. It can be seen in this figure that PMFL and KYMFL have, as expected, similar system performance. The results for the uniform distribution lead also to a similar conclusion, however they are not shown to conserve space. Also, simulations for other system sizes that grow to thousands of cores (16 × 16, 64 × 64, 128 × 128 and 256 × 256) do not modify this system performance conclusion for PMFL and KYMFL. The detection schemes PMFL and KYMFL have similar system performance because they both detect the unique set of maximal free submeshes. Also, MMPL and RBF have similar performance, and they outperform SFF substantially. Note that MMPL is a simpler scheme when compared with RBF.

To compare the policies in terms of allocation and de-allocation times, we measured the average actual times taken by the combination of these operations for five hundred runs of the simulator. In Figures 9 and 10, we show the. combined measured times against the job arrival rates under

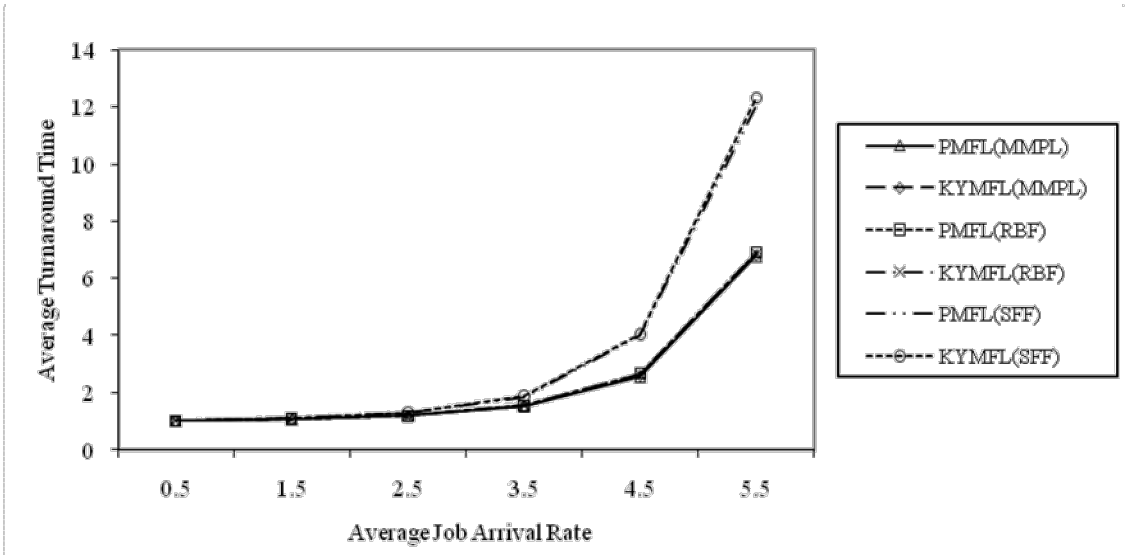

Figure 8: Average job turnaround times in a 32 × 32 system for the uniform-decreasing size distribution

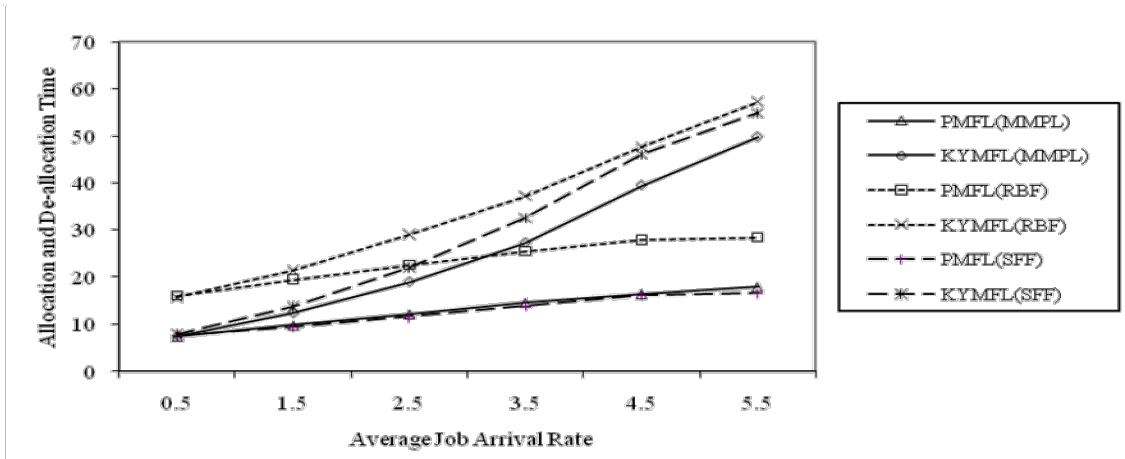

Figure 9: Measured combined times in a 32 × 32 system for the uniform-decreasing size distribution

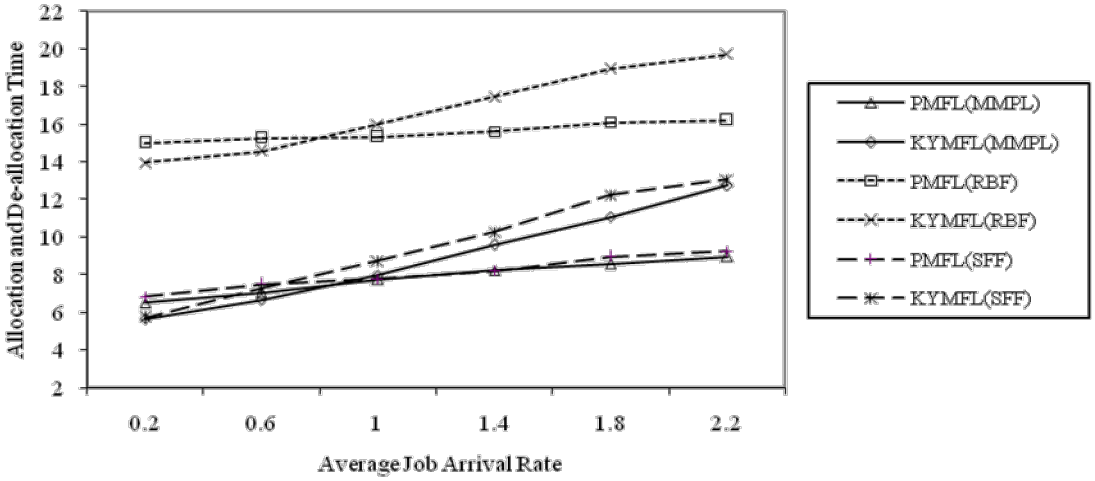

Figure 10: Measured combined times in a 32 × 32 system for the uniform size distribution

the size distributions considered in a 32 × 32 system. In these figures PMFL outperforms KYMFL substantially. Moreover, the advantage of PMFL is superior when the size distribution is uniform-decreasing. The reduction in the combined times for PMFL reaches 70% in Figure 9, and30% in Figure 10. Under the uniform-decreasing distribution, the average job size is smaller than under the uniform distribution, leading to a larger number of allocated (and free submeshes). This results in superior advantage for PMFL.

The average number of maximal free submeshes was computed for the simulations. This number increases with the system load and depends on the allocation scheme. As expected, it is comparatively small and varied from 1.16 to 3.22 for the uniform distribution. For the uniform-decreasing distribution, it varied from 1.5 to 9.85.

To illustrate the efficiency advantage of PMFL more clearly, we plot, in Figures 11 and 12, the relative measured times for PMFL with respect to KYMFL. In these figures, we have R(*S*) =T(PMFL(*S*))/T(KYMFL(*S*)), where T(PMFL(*S*)) is the measured simulation allocation and de-allocation time for PMFL when the selection algorithm is *S*, and T(KYMFL(*S*)) is this time for KYMFL and the same selection algorithm. Figure 11 shows that the efficiency advantage of PMFL over KYMFL is substantial under most loads. It increases with the load because the number of free submeshes, *f*, also increase with the load. The reduction in the combined times reaches about 50% for RBF, and it reaches about 70% for SFF and MMPL.

In Figure 12, the performance advantage of PMFL for medium to heavy loads is less substantial because *f* is smaller when the size distribution is uniform. The reduction in the combined times reaches about 10% for RBF, and it reaches about 30% for SFF and MMPL under heavy loads.

In summary, PMFL and KYMFL have similar system performance as they have identical submesh recognition capability, however PMFL can be much more time efficient than KYMFL, especially when the number of free submeshes is large. The numbers of allocated and free submeshes are larger when the core allocation requirements of jobs are small.

In Figure 13, we show the combined allocation and de-allocation times of the detection and selection policies for various side lengths under the system load of 4.5 jobs/time unit and the uniform-decreasing side-length distribution. Figure 14 is for a load of 1.8 jobs/time unit and the uniform

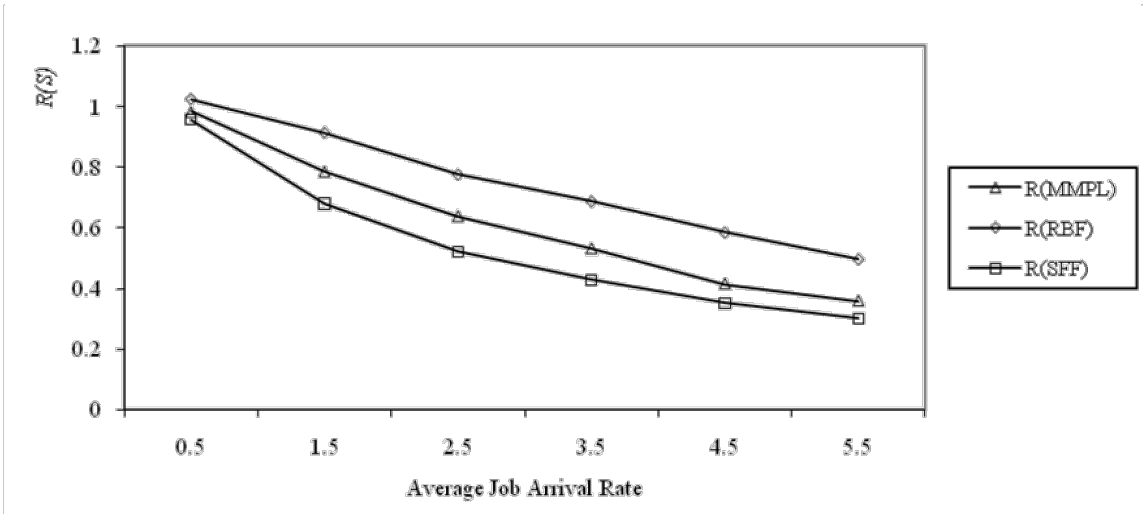

Figure 11: Ratio of the measured times for the allocation submesh selection policies and the uniform-decreasing size distribution in a 32 × 32 system

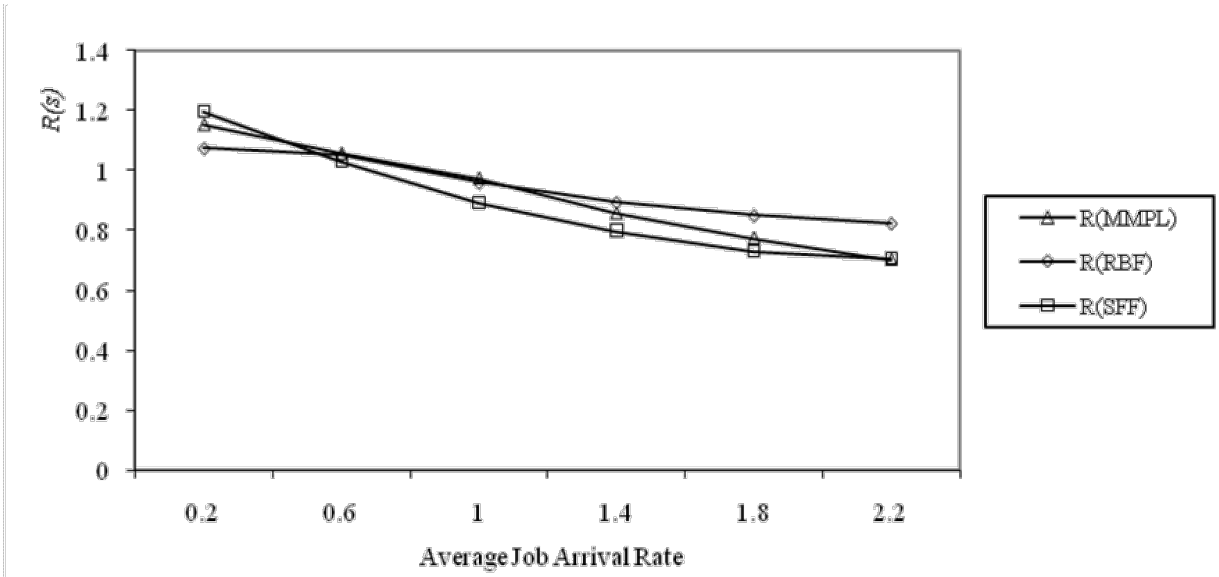

Figure 12: Ratio of the measured times for the allocation submesh selection policies and the uniform size distribution in a 32 × 32 system

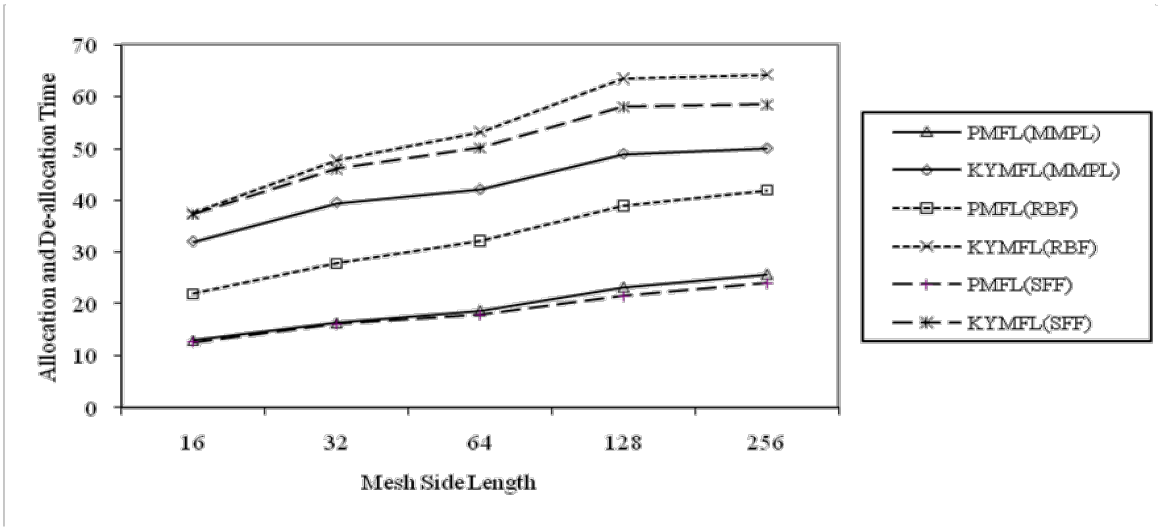

Figure 13: Measured combined times for doubled side lengths under the uniform-decreasing size distribution and a system load of 4.5 jobs/time unit

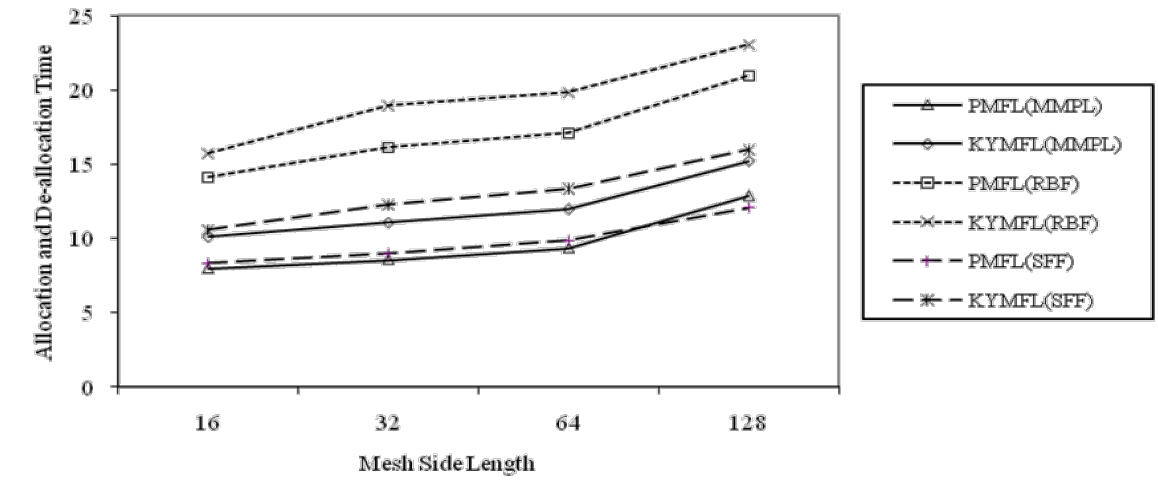

Figure 14: Measured combined times for doubled side lengths under the uniform size distribution and a system load of 1.8 jobs/time unit

performance advantage of PMFL can remain substantial as the size of the computer system grows to tens of thousands of cores.

6 Conclusions

In this paper, we have proposed an efficient maximal free submesh detection scheme for space-sharing allocation in manycore systems with 2D NoCs. Several studies indicate that space-sharing is a promising core allocation strategy in manycore systems, as it can achieve scalability and good performance for large core numbers [22, 25]. Parallel jobs or applications, including the OS, run on their own sets of cores, which can reduce interference among jobs, message delays, energy consumption and chip temperatures. Studies have shown that mapping the communicating tasks of a parallel job to neighboring cores, in particular those forming a submesh, can reduce communication delays and power consumption, and improve throughput and job execution times [5, 8, 18]. In this research, maximal free submeshes that are not contained in other free submeshes are detected and placed in a free-list. An advantage of this scheme over that proposed previously is that its time complexity is quadratic in *f*, whereas that of the previous scheme is cubic in this number. In addition to this theoretical comparison, the two recognition-complete detection schemes were evaluated and compared using detailed simulations when three promising allocation submesh selection schemes were used in combination with these detection schemes. The results show that the detection schemes have similar free submesh recognition-capability and average turnaround times, however the proposed scheme is overall substantially more efficient than the previous scheme in terms of the combined allocation and de-allocation times. Also, the simulated time performance advantage increases with the number of free submeshses, which is compatible with the time complexity advantage. It is to be noted that detecting maximal free submeshes is suitable for achieving simple, flexible, and efficient selection of allocation submeshes as the largest free submeshes are readily available in a list. The results also show that the simple scheme MMPL achieves good system performance. It outperforms SFF and achieves similar performance to the more complicated RBF scheme. As extensions to this work, more general defragmentation algorithms that make use of the efficient MFL detection mechanisms proposed in this work could be investigated.

Acknowledgment

We would like to thank Al al-Bayt University as this research has been carried out during a sabbatical leave for Ismail Ababneh.

References

[1] I. Ababneh, "An Efficient Free-List Submesh Allocation Scheme for Two-Dimensional Mesh-Connected Multicomputers," *Journal of Systems and Software*, 79:1168-1179, 2006.

[2] I. Ababneh, "A Performance Comparison of Contiguous Allocation Placement Schemes for 2D Mesh-Connected Multicomputers," 2007 ACS/IEEE International Conference on Computer Systems and Applications, (AICCSA 2007), Amman, Jordan, May 13-16, 2007.

[3] I. Ababneh, "On Submesh Allocation for 2D Mesh Multicomputers using the Free-List Approach: Global Placement Schemes," *Performance Evaluation*, 66(2):105-120, 2009.

[4] I. Ababneh, S. Bani-Mohammad, and M. Ould-Khaoua, "All Shapes Contiguous Submesh Allocation for 2D Mesh Multicomputers," *International Journal of Parallel, Emergent, and Distributed Systems,* 25(5):411-421, 2010.

[5] M. O. Agyeman, A. Ahmadinia, and N. Bagherzadeh, "Energy and Performance-Aware Application Mapping for Inhomogeneous 3D Networks-on-Chip," *Journal of Systems Architecture*, 89:103-117, September 2018.

[6] G.-M. Chiu and S.-K. Chen, "An Efficient Submesh Allocation Scheme for Two-Dimensional Meshes with Little Overhead," *IEEE Trans. on Parallel and Distributed Systems*, 10(5):471-486, 1999.

[7] P.-J. Chuang and N.-F. Tzeng, "Allocating Precise Submeshes in Mesh Connected Systems," *IEEE Trans. on Parallel and Distributed Systems*, 5(2):211-217, 1994.

[8] N. Dahir, A. Karkar, M. Palesi, T. Mak, and A. Yakovlev, "Power Density Aware Application Mapping in Mesh-Based Network-on-Chip Architecture: An Evolutionary Multi-Objective Approach," *Integration*, 81:342-353, November 2021.

[9] D. Das Sharma and D. K. Pradhan, "Submesh Allocation in Mesh Multicomputers using Busy-List: A Best-Fit Approach with Complete Recognition Capability," *J. of Parallel and Distributed Computing*, 36:106-118, 1996.

[10] J. Ding and L.-N. Bhuyan, "An Adaptive Submesh Allocation Strategy for Two-Dimensional Mesh Connected Systems," *Proc. Int'l Conf. Parallel Processing II*, pp. 193-200,1993.

[11] P. Gratz, C. Kim, K. Sankaralingam, H. Hanson, P. Shivakumar, S. W. Keckler, and D. Burger, "On-Chip Interconnection Networks of the TRIPS Chip," *IEEE Micro*, 27(5):41-50, November 2007.

[12] G. Kim and H. Yoon, "On Submesh Allocation for Mesh Multicomputers: A Best-Fit Allocation and a Virtual Submesh Allocation for Faulty Meshes," *IEEE Trans. on Parallel and Distributed Systems*, 9(2):175-185, 1998.

[13] H. Lahdhiri, J. Lorandel, S. Monteleone, E. Bourdel, and M. Palesi, "Framework for Design Exploration and Performance Analysis of RF-NoC Manycore Architecture," *Journal of Low Power Electronics and Applications*, 10(4):37, MDPI 2020.

[14] T. Liu, W.-K. Huang, F. Lombardi and L. N. Bhuyan, "A Submesh Allocation Scheme for Mesh-Connected Multiprocessor Systems," *Proc. Int'l Conf. Parallel Processing II*, pp. 159-163,1995.

[15] V. Lo, K. J. Windisch, W. Liu, and B. Nitzberg, "Noncontiguous Processor Allocation Algorithms for Mesh-Connected Mutlicomputers," *IEEE Trans.*

on Parallel and Distributed Systems, 8(7):712-725, 1997.

[16] H. Matsutani, M. Koibuchi, and H. Amano, "Tightly-Coupled Multi-Layer Topologies for 3-D NoCs," 2007 International Conference on Parallel Processing, ICPP [4343882], *Proceedings of the International Conference on Parallel Processing*, pp. 75-85, https://doi.org/10.1109/ICPP.2007.79, 2007

[17] S. Mazumdar and A. Scionti, "Ring-Mesh: A Scalable and High-Performance Approach for Manycore Accelerators," *The Journal of Supercomputing*, 76:6720-6752, 2020.

[18] A. Mosayyebzadeh, M. M. Amiraski, and S. Hessabi, "Thermal and Power Aware Task Mapping on 3D Network on Chip," *Computers and Electrical Engineering*, 51:157-167, April 2016

[19] J. Ng, X. Wang, A. Singh, and T. Mak, "Defragmentation for Efficient Runtime Resource Management in NoC-Based Many-Core Systems," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 24(11):3359-3372, 2016

[20] A. Pathania, V. Venkataramani, M. Shafique, T. Mitra, and J. Henkel, "Defragmentation of Tasks in Many-Core Architecture," *ACM Trans. on Architecture and Code Optimization*, 14(1):1-21, 2017

[21] ProcSimity, *ProcSimity v4.3 User's Manual*, University of Oregon, May 17, 1996.

[22] H. Sasaki, T. Tanimoto, K. Inoue, and H. Nakamura, "Scalability-Based Manycore Partitioning," PACT'12, Minneapolis, Minnesota, USA, September 19-23, 2012.

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

[24] D. Wentzlaff, P. Griffin, H. Hoffmann, B. Liewei, B. Edwards, C. Ramey, M. Mattina, M. Chyi-Chang, J.F. Brown, and A. Agarwal, "On-Chip Interconnection Architecture of the Tile Processor," *Micro, IEEE*, 27(5):15-31, November 2007

[25] D. Wentzlaff, C. Gruenwald, N. Beckmann, K. Modzelewski, A. Belay, L. Youseff, J. Miller, and A. Agrawal, "A Unified Operating System for Clouds and Manycore: fos," Computer Science and Artificial Intelligence Lab, MIT, Tech. Rep. MIT-CSAIL-TR-2009-059, Nov. 2009

[26] S.-M. Yoo, H. Y. Youn, and B. Shirazi, "An Efficient Task Allocation Scheme for 2D Mesh Architectures," *IEEE Trans. on Parallel and Distributed Systems*, 8(9):934-942, 1997.

[27] Y. Zhu, "Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers," *J. Parallel and Distributed Computing*, 16:328-337, 1992.

[28] X. Zhu and W.-M. Lin, "Allocation-Time-Based Processor Allocation Scheme for 2D Mesh Architecture," *J. of Information Science and Engineering*, 16:301-311, 2000.

Ismail Ababneh received a BS degree in Electromechanical Engineering from the National Superior School of Electronics and Electro-mechanics of Caen, France, in 1979, the MS degree in Software Engineering from Boston University in 1984, and the Ph.D. degree in Computer Engineering from Iowa State University in 1995. From 1984 to 1989, he was a Software Engineer with Data Acquisition Systems, Boston, Massachusetts. He is presently a professor in the Department of Computer Science at Al al-Bayt University in Jordan. 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 a member of Tau Beta Pi and Eta Kappa Nu. He held several administative positions at Al al-Bayt University, including Head of Computer Science Department, Dean of IT College, Director of Computer Center, Dean of Research, and Vice President for Administration and Student Affairs. His current research interests include processor allocation in multicomputers, and ad hoc routing algorithms. He has published about 70 papers in journals, conferences and workshops.

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.