

# A “Generalized k-Tree-Based Model to Sub-System Allocation” for Partitionable Multi-Dimensional Mesh-Connected Architectures

Jeeraporn Srisawat<sup>1</sup> and Nikitas A. Alexandridis<sup>2</sup>

<sup>1</sup> King Mongkut’s Institute of Technology  
Ladkrabang, Bangkok 10520, THAILAND  
ksjeerap@kmitl.ac.th

<sup>2</sup>The George Washington University,  
Washington, DC 20052, U.S.A.  
alexan@seas.gwu.edu

**Abstract.** This paper presents a new processor allocation approach called “a generalized k-Tree-based model” to perform dynamic sub-system allocation/deallocation decision for partitionable multi-dimensional mesh-connected architectures. Time complexity of our generalized k-tree-based sub-system allocation algorithm is  $O(k^4 2^k (N_A + N_F) + k^2 2^{2k})$  for the partitionable k-D meshes and  $O(N_A + N_F)$  for the partitionable 2-D meshes, where  $N_A$  is the maximum number of allocated tasks,  $N_F$  is the corresponding number of free sub-meshes,  $N$  is the system size, and  $N_A + N_F \leq N$ . Most existing processor allocation strategies have been proposed for the partitionable 2-D meshes with various degrees of time complexity and system performance. In order to evaluate the system performance, the generalized k-Tree-based model was developed and by simulation studies the results of applying our k-Tree-based approach for the partitionable 2-D meshes were presented and compared to existing 2-D mesh-based strategies. Our results showed that the k-Tree-based approach (when it was applied for the partitionable 2-D meshes) yielded the comparable system performance to those recently 2-D mesh-based strategies.

## 1 Introduction

A multi-dimensional (k-D) mesh-connected parallel architecture is a useful network type of parallel systems for high performance parallel applications that may require different degrees of processor interconnection (such as 1-D text processing, 2-D image processing, 3-D graphic processing, etc.). The partitionable k-D mesh system is provided (at run time) for executing various independent applications (or tasks) in parallel. Examples of prototypes and commercial parallel systems (which support a multi-user environment) include the Intel Touchstone system [5], the Intel Paragon XP/S [6], the Intel/Sandia ASCI System [10], etc. In the partitionable parallel systems, a number of independent smaller tasks (from the same or different applications) come in, each requiring at run time a separate sub-system (or partition) to execute. In order to provide appropriate free sub-systems for

new tasks, a special designed operating system (known as the processor allocator) has to dynamically partition the computer system to allocate a sub-system for each incoming task, as well as to deallocate a sub-system and recombine partitions as soon as they become available when a task completes. For the partitionable 2-D mesh-connected systems, a number of processor allocation/deallocation (decision) techniques have been proposed in the past such as bit-map approaches [11], [16] and non-bit-map approaches [1], [2], [3], [4], [7], [8], [9], [12], [13], [14], [15]. Among those existing 2-D mesh-based strategies, time complexities of recently sub-system allocation methods (that yield the comparable system performance) are  $O(N_a^3)$  of the Busy List (BL) [3],  $O(N_a\sqrt{N})$  of the Quick Allocation (QA) [15], and  $O(N_f^2)$  for the 2-D meshes and  $O(N_f^k)$  for the k-D meshes of the Free Sub-List (FSL) [7], where  $N$  is the system size,  $N_a$  is the number of allocated tasks ( $N_a \leq N$ ), and  $N_f$  is the number of free sub-systems ( $N_f \leq N$ ).

In this paper, we propose the designing of “a generalized k-tree-based model” in order to perform dynamic sub-system allocation/deallocation decision for partitionable multi-dimensional mesh-connected architectures. Our generalized model includes a k-Tree system state representation and a number of generalized algorithms (such as the network partitioning algorithm, the sub-system combining algorithm, the best-fit heuristic for allocation decision, the sub-system allocation/deallocation decision algorithm). Time complexity of the generalized k-Tree-based approach for the partitionable k-D meshes is  $O(k^4 2^k (N_A + N_F) + k^2 2^{2k})$ , where  $N_A$  is the maximum number of allocated tasks,  $N_F$  is the corresponding number of free sub-meshes,  $N$  is the system size, and  $N_A + N_F \leq N$ . Therefore, time complexity of our approach for the partitionable 2-D meshes is only  $O(N_A + N_F)$ . For the performance evaluation (by simulation study), the generalized k-Tree-based model was developed and the system performance of our application for the partitionable 2-D meshes was presented and compared to the recently 2-D mesh-based strategies such as the Busy List (BL) [3], the Free-Sub List (FSL) [7], and the Quick Allocation (QA) [15]. Our simulation results showed that the k-Tree-based approach yielded the comparable system performance to those recently 2-D mesh-based strategies.

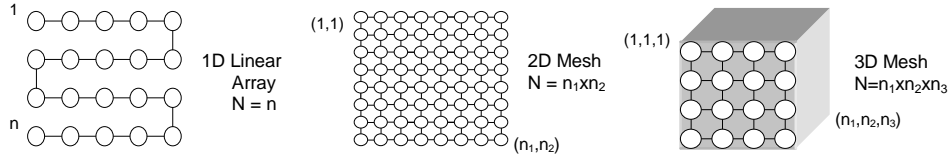
In the next section, we present the generalized k-Tree-based model to perform processor allocation/deallocation decision for the partitionable k-D mesh parallel machines as well as the corresponding time complexity. Section 3 presents the evaluated system performance of the generalized k-Tree-based model. Then, the performance results (by simulation study) for the 2-D mesh networks are presented and compared to existing 2-D mesh-based strategies. Finally, conclusions are discussed in Section 4.

## 2 k-Tree-Based Model to Sub-System Allocation for k-D Meshes

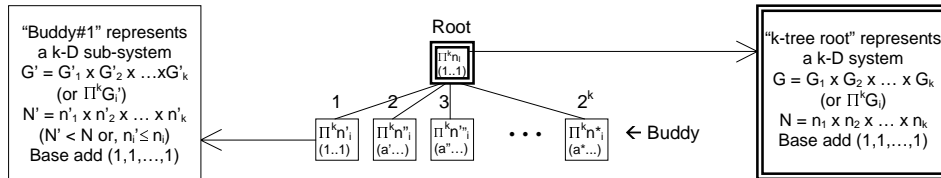
In this paper, “the generalized k-Tree-based model” is proposed to perform sub-system allocation/deallocation decision for the partitionable multi-dimensional (k-D) mesh-connected architectures. In the generalized k-Tree-based model, we use a data structure, called a “k-Tree” to represent the system states of the partitionable k-D mesh-connected systems. Our generalized k-Tree-based model includes a k-Tree system states representation (Section 2.1) and a number of generalized algorithms for each sub-system allocation/deallocation decision such as the network partitioning algorithm (Section 2.2), the sub-system combining algorithm (Section 2.3), the best-fit heuristic for allocation decision (Section 2.4), and the sub-system allocation/deallocation decision algorithm (Section 2.5).

## 2.1 k-Dimensional Mesh Systems and k-Tree System Stage Representation

**DEFINITION 1:** A *k-Dimensional (k-D) Mesh Network* (of size  $N = n_1 \times n_2 \times \dots \times n_k$ ) is defined as a network graph  $G(V, E) = G_1 \times G_2 \times \dots \times G_k$  [or a product of  $k$  linear arrays  $G_i(V_i, E_i)$  of  $n_i$  nodes;  $V_i = \{1, 2, \dots, n_i\}$  and  $E_i = \{\langle j, j+1 \rangle \mid j = 1, 2, \dots, n_i-1\}$ ], where  $V = \{\alpha = (a_1, a_2, \dots, a_k) \mid a_i \in V_i, a_2 \in V_2, \dots, a_k \in V_k\}$  and  $E = \{\langle \alpha, \beta \rangle \mid \text{there exists an } i \text{ such that } \langle a_i, b_i \rangle \in E_i, \text{ where } i = 1, 2, \dots, k.\}$

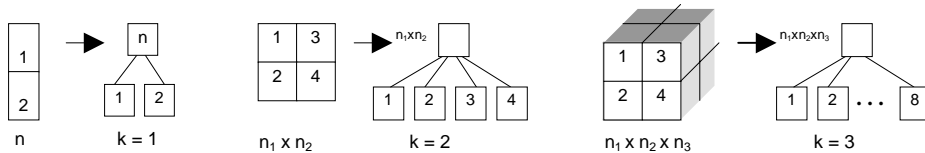


The “*k-Tree*” data structure is used to represent system states (or store allocation information) of the partitionable  $k$ -D mesh-connected parallel system, where  $k$  is a number of dimensions of the multi-dimensional ( $k$ -D) mesh-connected system. This is a special  $k$ -Tree since sub-systems’ sizes after partitioning are not necessary equal, unlike a balance  $k$ -Tree of the same sub-system sizes after partitioning. In this paper, a “system” refers to a given partitionable  $k$ -D system, represented as a  $k$ -product network  $G_1 \times G_2 \times \dots \times G_k$  of size  $N = n_1 \times n_2 \times \dots \times n_k$  and a “sub-system” refers to a smaller  $k$ -D system of size  $N' = n_1' \times n_2' \times \dots \times n_k'$ , where  $n_i' \leq n_i$  and  $i = 1, 2, \dots, k$ , which is provided for incoming task(s).



**Fig. 1.** The  $k$ -tree’s root and its  $2^k$  buddies.

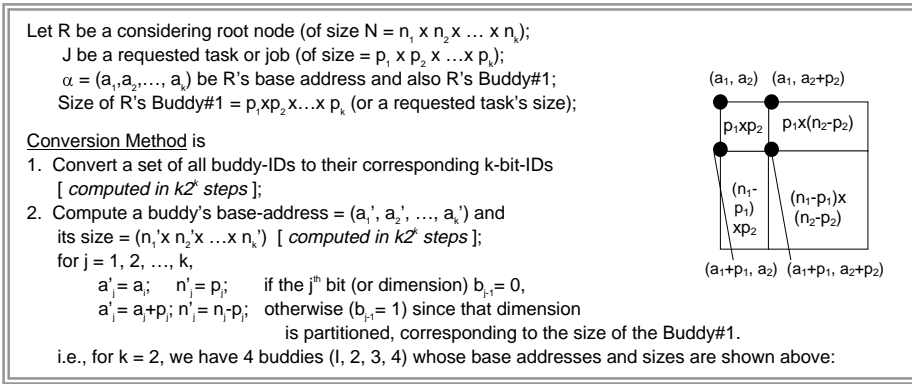
In our  $k$ -Tree-based approach, the number of nodes in the  $k$ -Tree is dynamic, corresponding to the number of tasks allocated. At the start (see Fig. 1.), the  $k$ -Tree consists of only one node (called the root), used to store the system information (i.e., a size, a base-address, a status, etc.) of the initial system (which is the  $k$ -D network  $G_1 \times G_2 \times \dots \times G_k$  (or  $\Pi^k G_i$ ) of size  $N = n_1 \times \dots \times n_k$  (or  $\Pi^k n_i$ ) with a base address  $\alpha = (1, 1, \dots, 1)$ , where  $i = 1, 2, \dots, k$ . During execution (run) time when many jobs (or tasks) are allocated, each leaf node (representing a sub-system) may be available or free (status = 0) or unavailable or busy (status = 1) and each internal node is partially available (status =  $x$ ). In order to allocate an incoming task, each larger free node in the  $k$ -Tree can be dynamically created and partitioned into a number of children/node, called buddies. Assume that the incoming tasks always request the same value “ $k$ ” as provided by the system. Therefore, the number of buddies =  $2^k$  (described later in Section 2.2).



## 2.2 Network Partitioning Algorithm

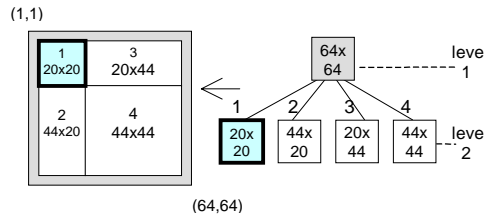
The network partitioning is a partitioning process that partitions all  $k$  dimensions (or  $k$  linear array networks) of the  $k$ -D mesh system (of size  $N = n_1 \times n_2 \times \dots \times n_k$ ) into smaller sub-systems and allocates one for the request ( $k$ -D mesh of size  $p_1 \times p_2 \times \dots \times p_k$ , where  $p_i \leq n_i, i = 1, 2, \dots, k$ ). Under this network partitioning, the relationship of the corresponding buddy's ID, buddy's base-address, and buddy's size are defined as follows:

Let a requested network is defined as a network graph  $G' = G_1' \times G_2' \times \dots \times G_k'$  (of size  $p_1 \times p_2 \times \dots \times p_k$ , where  $p_i \leq n_i, i = 1, 2, \dots, k$ ). Therefore, the number of buddies/node in each partitioning is equal to  $2^k$  (with the assigned IDs = 1, 2, 3, ...,  $2^k$ ) since all  $k$  dimensions are partitioned into two sizes, which are not necessary equal. Next, we introduce the corresponding buddy's base-address and buddy's size which can be defined in the "Buddy-ID-Address-Size conversion" algorithm (see Fig. 2.). This conversion process is introduced to provide the small number of steps for the network partitioning algorithm (described next) and the sub-system combining algorithm (described later in Section 2.3). This process (i.e., identifying #buddies =  $2^k$ , their base-addresses and sizes) is computed in  $k2^k$  steps and hence the total time complexity is  $O(k2^k)$ .



**Fig. 2.** The "Buddy-ID-Address-Size conversion" algorithm for the network partitioning.

For example (see Fig. 3.), consider a 2-D mesh system (of size  $N = n_1 \times n_2 = 64 \times 64$ ) which is stored in the  $k$ -Tree's root with a base address  $\alpha = (a_1, a_2) = (1, 1)$ , where  $k = 2$ . Suppose the first incoming task requests a sub-system of size  $20 \times 20$ . For this task ( $20 \times 20$ ), the root (2-D mesh system of size  $64 \times 64$ ), is partitioned into  $2^k = 4$  buddies. Let's assume that the request will be allocated to the sub-Buddy#1 (at level 2).



By applying "Buddy-ID-Address-Size conversion" algorithm (in Fig. 2.), the sub-Buddy#1's base-address is  $(a_1, a_2) = (1, 1)$  and its size is  $(p_1 \times p_2) = (20 \times 20)$ . First the set of bit-address of each buddy  $i, \{i / i=1, 2, 3, 4\}$  is  $\{(b_1 b_0) \mid b_i = 0 \text{ or } 1\} = \{00, 01, 10, 11\}$ , and hence its corresponding base-address is  $\{(a_1', a_2') \mid a_j' = a_j + (p_j * b_{j-1})\} = \{(1,1), (21,1), (1,21), (21,21)\}$  and its corresponding size is  $\{(n_1', n_2')\} = \{(20 \times 20), (44 \times 20), (20 \times 44), (44 \times 44)\}$ .

**Fig. 3.** An example of a 2-D mesh of size  $N = 64 \times 64$  with an allocated task of size  $20 \times 20$ .

### 2.3 Sub-System Combining Algorithm

The sub-system combining is applied during processor allocation or deallocation. First, the “Combinations of  $2^j$  Adjacent Buddies” algorithm is introduced (see Fig. 4.) in order to combine  $2^j$  buddies (where  $j = 1, 2, \dots, k-1$ ) into the larger free sub-systems. This algorithm is computed in  $O(k2^{k-j})$  time for each  $j$  and hence  $O(k2^k)$  time for all values of the variable  $j$ , where  $j = 1, 2, 3, \dots, k-1$  (since  $k2^1 + k2^2 + k2^3 + \dots + k2^{k-1} = 2k[2^{k-1}-1]$ ).

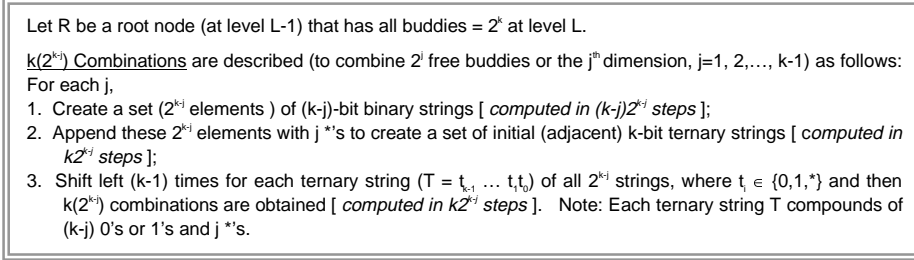


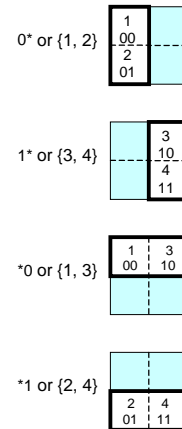
Fig. 4. The “Combinations of  $2^j$  adjacent buddies” algorithm for the sub-system combining.

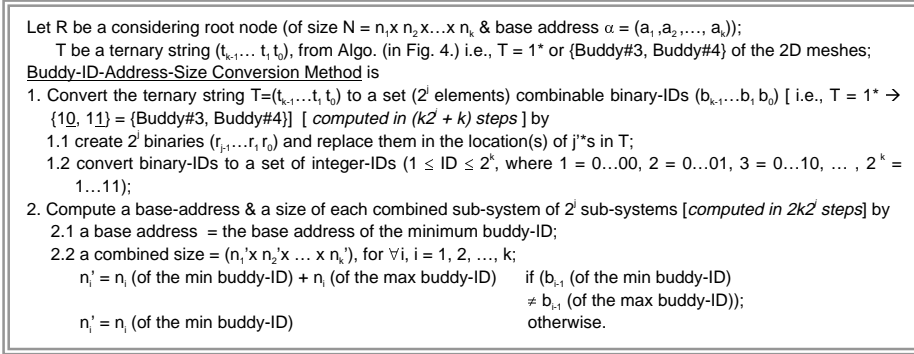
For example, consider a (64x64)-mesh system with 4-buddy partitioning, where  $k = 2$  and  $2^k = 4$  (see Fig. 3.). In order to combine one of  $k$  dimensions or  $2^j$  Buddies (such as  $j = 1$ ), there are  $k2^{k-j}$  ( $= 2 \times 2^{2-1} = 4$ ) possible combinations, recognized as follows: First, a 4-element set of “a compound of  $2^j$  adjacent buddies” is computed as a set of binary strings {0, 1}. Next, each of these two elements is appended with  $j$  (or 1) \* to create the corresponding set of ternary string {0\*, 1\*}. Then, each ternary string is shifted left  $(k-1)$  times to create a 4-element set of ternary string {0\*, \*0, 1\*, \*1} which represents a set of combinable strings (i.e., a ternary string 1\* is interpreted as a set of 2-adjacent Buddies {10, 11} (or {Buddy#3, Buddy#4}) for a combined system's size = 64x44.)

Next, we classify the  $k$ -Tree-based sub-system combining algorithms into three groups, which will be applied later in the allocation and deallocation procedures (Section 2.5):

**ALGORITHM C.1 “Combine All Buddies”:** This combining procedure is used to combine all free  $2^k$  buddies (at level L) into a larger free  $k$ -Tree's node at level L-1. This combining process is computed in  $O(2^k)$  time and it is applied after finishing the deallocation of any finished task in order to maintain the minimum number of nodes and the maximum free nodes' sizes in the  $k$ -Tree as much as possible.

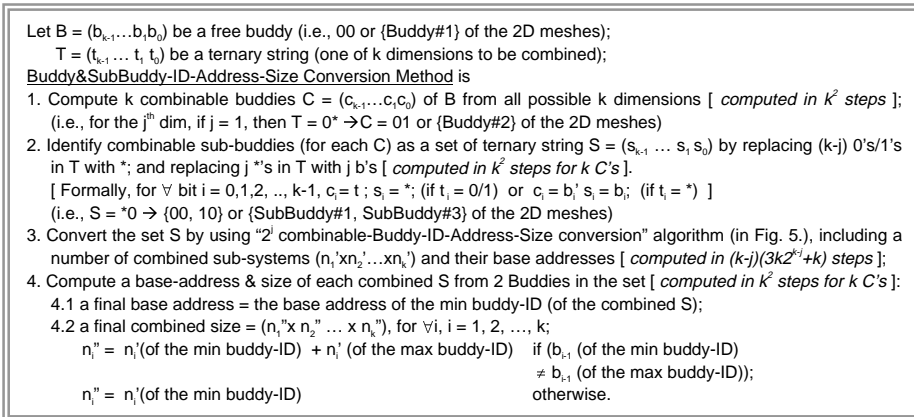
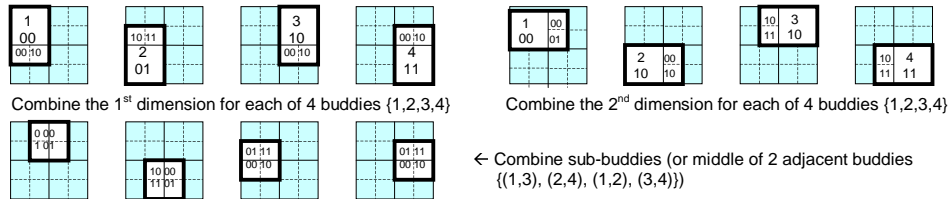
**ALGORITHM C.2 “Combine Some Buddies”** (or called “Buddy-Buddy combining”): This combining procedure is used to combine a number (i.e., 2, 4, ..., or  $2^{k-1}$  or  $j = 1, 2, \dots, k-1$ ) of adjacent free buddies (of the same root sub-tree) at level L into a larger free sub-system in order to allocate for an incoming task (whose requested a size which is larger than that of each of  $2^k$  buddies but is less than or equal to that of the combined sub-system.) After applying the algorithm (in Fig. 4.: see also the following figures for the combining of 2-D meshes), for a ternary string T, the combined sub-system's size and its base-address are computed by using the “ $2^j$  Combinable Buddy-ID-Address-Size conversion” algorithm (see Fig. 5.) which can be computed in  $(3k2^j + k)$  steps. Therefore, time complexity of this combining for each  $j$  in order to combine all possible  $\sum_{j=1}^{k-1} k2^{k-j}$  combined sub-systems (from  $2^j$  adjacent free buddies) is  $O(k^2 2^k)$  and hence  $O(k^2 2^k)$  for all value of  $j$ , where  $j = 1, 2, \dots, k-1$ .





**Fig. 5.** The “ $2^k$  Combinable Buddy-ID-Address-Size conversion” algorithm (Algorithm C.2).

**ALGORITHM C.3** “Combine a Buddy and Corresponding Sub-Buddies” (or called “Buddy-SubBuddy combining”) and “Combine Some SubBuddies” (or called “SubBuddy-SubBuddy combining”): These combining procedures are used to combine some free buddies (at level L) and their corresponding sub-buddy nodes (at Level L+1) to yield more sub-system recognition (from any partitioning size) than those obtained from the combining in Algorithm C.2. Then, we introduce the “Buddy-SubBuddy combining” algorithm (see Fig. 6.) for recognizing  $k2^k$  combined sub-systems of a free buddy at level L and its adjacent free nodes (or sub-buddies) at level L+1 and also the “SubBuddy-SubBuddy combining” algorithm (see Fig. 7.) for recognizing  $k2^{k-1}$  combined sub-systems of some adjacent free nodes at level L+1 [see also following figures for all possible combining for the 2-D meshes]. Each of these conversion processes can be computed in  $O(k^2 2^k)$  time for each combined node (or string) and  $O(k^2 2^{2k})$  time for all possible  $2^k$  strings.



**Fig. 6.** The “Buddy&SubBuddy-ID-Address-Size conversion” algorithm (Algorithm C.3-1).

Let  $T = (t_{k-1} \dots t_1 t_0)$  be a ternary string of a pair of combinable 2 buddies  
 SubBuddy&SubBuddy-ID-Address-Size Conversion Method is (in this case  $j = 1$ )

1. Convert  $T$  to a set of (2 elements) buddy-ID  $\{B_1, B_2/B_i = (b_{i,k-1} \dots b_{i,1} b_{i,0})\}$  of 2 partially free buddies at level  $L$  by creating 2 binary-numbers  $(r_{i,1} \dots r_{i,0})$ ; replacing them into  $j$ 's of  $(t_{k-1} \dots t_1 t_0)$  [computed in  $k^2$  steps].
2. Identify combinable sub-buddies of each of 2 buddies as a set of ternary string  $S = (s_{k-1} \dots s_1 s_0)$  [ computed in  $k^2$  steps]: for  $\forall$  bit  $i = 0, 1, 2, \dots, k-1, b_{1i} = t_i; b_{2i} = t_i$ ; and  $s_{1i} = *; s_{2i} = *$ ; if  $t_i = 0$  or  $1$ ,  
 or  $b_{1i} = 0_i; b_{2i} = 1_i$ ; and  $s_{1i} = b_{1i}; s_{2i} = b_{2i}$ ; if  $t_i = *$
3. Similar to that of algorithm in Fig. 6.
4. Similar to that of algorithm in Fig. 6.

Fig. 7. The “SubBuddy&SubBuddy-ID-Address-Size conversion” algorithm (Algorithm C.3-2)

## 2.4 Best-Fit Heuristic for Allocation Decision

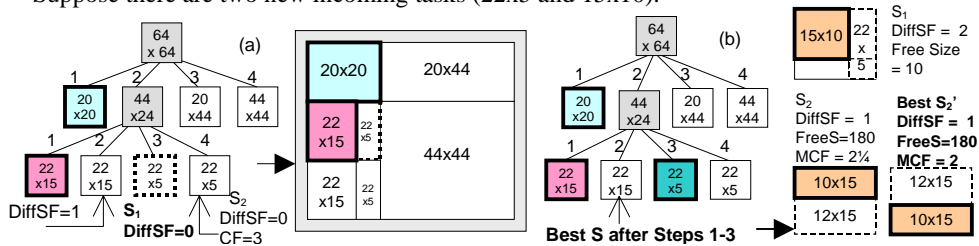
In this sub-section, we present a *generalized best-fit heuristic* for the partitionable k-D mesh-connected systems. The best-fit heuristic is to find the best free sub-system (of all possible available sub-systems) for an incoming task by introducing a number of criteria that tend to cause the minimum system fragmentation(s), which are:

**Best-Fit Criteria:**

1. Find all free sub-systems that can preserve the “maximum free size” as possible [ see *Algorithm A.1* in  $O(k)$  time ].
2. If there are many candidates (sizes  $\geq$  the request) that have the same property in (1), then the candidate that gives the “minimum different size factor (diffSF)” is selected [ see *Algorithm A.2* in  $O(k^2)$  time ].
3. If there are many candidates (sizes  $\geq$  the request) that have the same property in (1) & (2), then the “smallest size” candidate that yields the “minimum combining factor (CF)” [ see *Algorithm A.3* in  $O(k)$  time ] is selected. Otherwise, select by random.
4. After searching on all nodes in the k-Tree,
  - If the best free sub-system is “equal to” the request, then it is directly allocated to the request.
  - Otherwise (it is “larger than” the request), it is partitioned and one of its buddies which yields properties similar to that given in Step 1 – Step 3 plus being the “best buddy location” or providing the “minimum modified CF (MCF)” will be selected [ see *Algorithm A.4* in  $O(k^2)$  time ].

**Note:** Criteria 1-3 are applied for every free node or combined sub-system; however, a criterion 4 is computed only once for the best free sub-system, obtained from Steps 1-3.

For example, given a 64x64-system with two tasks (20x20, 22x15) allocated (see Fig. 8). Suppose there are two new incoming tasks (22x5 and 15x10).

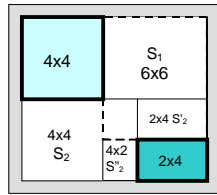


For the new task (22x5), searching process starts from the root and then visits all nodes in the k-Tree (in order to find the best free S). After applying Steps 1-3 of the best-fit heuristic, the best free node that can accommodate the request is  $S_1(22, 5)$  (with min diffSF, min CF) at level 3 (see Fig. 8.a). Therefore, it is allocated to the requested task 22x5. For the next task (15x10) (see Fig. 8.b), after applying the best-fit criteria (Steps 1-3), the best sub-system is  $S(22, 15)$  at level 3 since it can preserve the maximum free size (64x44). Since the sub-system  $S(22, 15)$  is larger than the request 15x10, Step 4 is applied that is the  $S(22, 15)$  is partitioned. After partitioning, the rotated size 10x15 ( $S_2$ ) provides the better best-fit value (diffSF = 1, free size = 180) than that of the regular size 15x10 ( $S_1$ ) (diffSF = 2, free size = 110). Finally, among  $S_2$  (the top buddy) and  $S_2'$  (the bottom buddy), the  $S_2'(10,15)$  is selected and allocated to the request task (15x10) since it yields the minimum modified CF (MCF) (see Algorithm A.4).

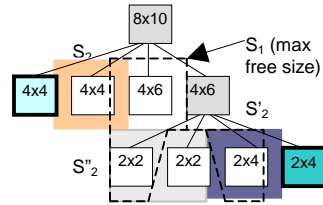
Fig. 8. An example of the “best fit” k-Tree-based allocation.

**ALGORITHM A.1:** “Overlap Status” of two available sub-systems  $S_1$  (the maximum free size) and  $S_2$  (any free sub-system) is identified as follows: If these two sub-systems ( $S_i$ ,  $i = 1, 2$ ) in the k-Tree have base-address  $\alpha_i = (a_{i1}, a_{i2}, \dots, a_{ik})$ , last-cover address  $\beta_i = (b_{i1}, b_{i2}, \dots, b_{ik})$ , and size  $|S_i| = (n_{i1} \times n_{i2} \times \dots \times n_{ik})$ , where  $b_{ij} = a_{ij} + n_{ij} - 1$ ,  $i = 1, 2$ , and  $j = 1, 2, \dots, k$ , then Case 1: the sub-system  $S_2$  is a *subset* of the sub-system  $S_1$  if  $(a_{2j} \geq a_{1j} \text{ and } b_{2j} \leq b_{1j})$  for  $\forall j, j = 1, 2, \dots, k$ . Case 2: the sub-systems  $S_1$  and  $S_2$  are *disjoint* either if  $(a_{1j} - a_{2j} \geq n_{2j})$  or if  $(a_{2j} - a_{1j} \geq n_{1j})$  for  $\exists j, j = 1, 2, \dots, k$ . Otherwise (neither Case 1 nor Case 2) the sub-system  $S_1$  *intersects* the sub-system  $S_2$ . Each of these three statuses (subset, disjoint, intersect) can be computed in  $O(k)$  time.

For instance, the following figure illustrates various overlap statuses between  $S_1$  “the maximum free size” [ $n_1 = 6 \times 6$  at  $\langle (1, 5), (6, 10) \rangle$ ] and other free sub-systems such as  $S_2$  [ $n_2 = 4 \times 4$  at  $\langle (5, 1), (8, 4) \rangle$ ],  $S'_2$  [ $n'_2 = 2 \times 4$  at  $\langle (5, 7), (6, 10) \rangle$ ], and  $S''_2$  [ $n''_2 = 4 \times 2$  at  $\langle (5, 5), (8, 6) \rangle$ ].



$S_1$  and  $S_2$  are *disjoint* since for  $i = 2$  there exists  $(a_{12} - a_{22}) = (5 - 1) = 4 \geq n_{22}$  ( $= 4$ ).  $S'_2$  is *subset* of  $S_1$  since  $S'_2 = \langle (5, 7), (6, 10) \rangle \subset S_1 = \langle (1, 5), (6, 10) \rangle$  (or  $\forall i = 1, 2; a_{2i} \geq a_{1i}$ ; and  $b_{2i} \leq b_{1i}$ ).  $S''_2$  *intersects* to  $S_1$  since they are not disjoint and also neither is a subset of the other.



**ALGORITHM A.2:** “Task Rotation” in this paper is a process of shifting a given task size  $k-1$  times to find the suitable location of the free sub-system for the requested (k-D) task of size  $p_1 \times p_2 \times \dots \times p_k$ . Thus, there are  $k$  possible rotated sizes that can be allocated for the task:  $(p_1 \times p_2 \times \dots \times p_k)$ ,  $(p_2 \times p_3 \times \dots \times p_1 \times p_k)$ , ..., and  $(p_k \times p_1 \times \dots \times p_{k-1})$ . For the partitioning of each rotated task size against a given free  $S$ , the different size factor ( $0 \leq \text{diffSF} \leq k$ ) and the maximum (remaining) free size (FS) ( $0 < |\text{FS after partition}| < |\text{FS before partition}|$ ) are computed in  $O(k)$  time and hence in  $O(k^2)$  time for all  $k$  possible rotated sizes. See an example of the task rotation in Fig. 8.b that is the rotated  $S_2(10 \times 15)$  is selected rather than the regular  $S_1(15 \times 10)$ .

**ALGORITHM A.3:** “Combining Factor (CF)” of any free sub-system  $S$  (at level  $L$ ) is computed from its adjacent neighbor nodes as a summation of the probability of combining (PC) of each of  $2^k$  combinable nodes of the same root sub-tree.

In this study, for an adjacent side we define  $\text{PC}=0$  if that particular combined side is one of  $2^k$  system boundaries (since that side cannot be combined),  $\text{PC}=1/4$  if its adjacent node of that particular side is busy (since it can be combined after it becomes free);  $\text{PC}=1/2$  if its adjacent node is partially available (and some free sub-buddies may be combined); or  $\text{PC}=1$  if its adjacent node is free (or it can be immediately combined). Then,  $\text{CF}(\alpha)$  is the combining factor of  $\alpha$  is  $\text{CF}(\alpha) = \text{CF}_1(\alpha, \beta) + \text{CF}_2(\alpha, \gamma)$ , where  $\text{CF}_1(\alpha, \beta) = \text{PC}(\alpha, \beta_1) + \text{PC}(\alpha, \beta_2) + \dots + \text{PC}(\alpha, \beta_k)$  is the combining factor of  $\alpha$  at  $L-1$  and  $\text{CF}_2(\alpha, \gamma) = \text{PC}(\alpha, \gamma_1) + \text{PC}(\alpha, \gamma_2) + \dots + \text{PC}(\alpha, \gamma_k)$  is the combining factor of  $\alpha$  at  $L-2$ :

Let  $\alpha$  denotes a binary-ID ( $b_{k-1} \dots b_0$ ) of a considering node at level  $L$   
 $k$  is a number of combinable buddies of the root sub-tree (of  $\alpha$ ) at level  $L-1$  or  $L-2$ .  
 $\beta_1, \beta_2, \dots, \beta_k$  denote binary-IDs of combinable node(s) of the considering node  $\alpha$  with the same root sub-tree at level  $L-1$   
 $\gamma_1, \gamma_2, \dots, \gamma_k$  denote binary-IDs of adjacent node(s) of the considering node  $\alpha$  with the same root sub-tree at level  $L-2$

**Identify adjacent nodes** by using the following rules:

- 1) For a k-Tree node, each  $\beta_i$  or  $\gamma_i$  is identified by negating the  $i^{\text{th}}$  bit of that node ( $\alpha$ ) which are  $\beta_1 = (b_{k-1} \dots b_1 b_0)$ ,  $\beta_2 = (b_{k-1} \dots b_1 b_0)$ , ..., and  $\beta_k = (b_{k-1} \dots b_0 b_1)$ . Therefore, for a root( $\alpha$ ) =  $(r_{k-1} \dots r_1 r_0)$ , its combinable buddies are  $\gamma_1 = (r_{k-1} \dots r_1 r_0)$ ,  $\gamma_2 = (r_{k-1} \dots r_1 r_0)$ , ..., and  $\gamma_k = (r_{k-1} \dots r_1 r_0)$ , respectively (See the following example).
- 2) For a combined sub-system  $S = (t_{k-1} \dots t_1 t_0)$  of  $2^j$  free (or partially free) buddies,  $1 \leq j < k$ , (or  $2^j$  combined nodes) and  $\beta_i =$  negate the  $i^{\text{th}}$  0/1 (or non \*) bit, represented  $S$  as a binary number, where  $i=1, 2, \dots, j$ .

Given an 8x8-mesh. Let  $\alpha = b_1 b_0 = 11$  (or 4), residing at level 3 and 2 adjacent buddies of  $\alpha$  are  $\beta_1 = 10$  (or 3);  $\beta_2 = 01$  (or 2). Assume the root of node  $\alpha$  at level 2 = 10 (or 3) and then 2 adjacent nodes of the root ( $\alpha$ ) are  $\gamma_1 = 11$  (or 4) and  $\gamma_2 = 00$  (or 1).

**ALGORITHM A.4:** “*The Best Buddy (or Best Sub-partition)*” is applied after partitioning (for Step 4 in the best-fit heuristic). In this case, assume the best free sub-system from Steps 1-3 is the node  $S$ , whose size is larger than the requested task. Then, the node  $S$  will be partitioned into  $2^k$  buddies and the best sub-partition (or one of  $2^k$  buddies will be allocated to the request. Let  $\beta_i = (\beta_{i1}, \beta_{i2}, \dots, \beta_{ik})$  be a set of combinable buddies of the considering node ( $\alpha_i$ ), where  $i = 1, 2, \dots, 2^k$ . Since the combining factor (see Algo. A.3)  $CF(\alpha, \beta)$  is the same for all  $\alpha_i$ ,  $i = 1, 2, \dots, 2^k$ , the modified combining factor  $MCF(\alpha_i, \beta_j)$  for each  $\alpha_i$  is computed as  $MCF(\alpha_i, \beta_j) = \sum_1^k PC(\alpha_i, \beta_j)$  in  $O(k2^k)$  time and the one (yielding min MCF) is selected as the best buddy.

See an example of finding the best buddy node in Fig. 8.b: the MCF of  $S_2$  ( $10 \times 15$ ) =  $1+1+\frac{1}{4}+0 = 2\frac{1}{4}$  (since its four combinable boundaries are two free node, one busy node, and one system boundary) and the MCF of  $S'_2 = 1+1+0+0 = 2$  (since its four combinable boundaries are two free nodes and two system boundaries); and hence the  $S'_2$  yields the minimum MCF and it is selected.

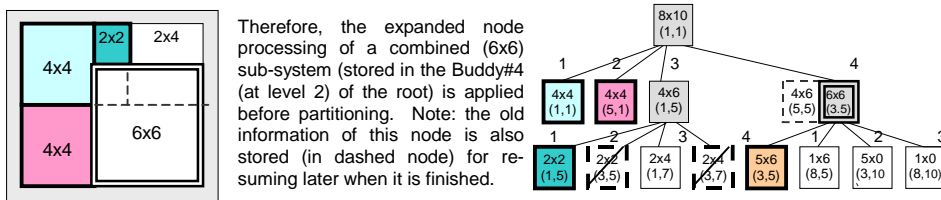
## 2.5 Allocation/Deallocation Decision Algorithm

In “*the best-fit k-Tree-based processor allocation*” procedure, the searching starts from the k-Tree’s root and goes to the left most (leaf) node. If that node is free and its size can accommodate the request, then its best-fit value (see Section 2.4) is computed. Then, the best sub-system is updated if the new free sub-system yields the better best-fit value (since it tends to cause the minimum system fragmentation). The above process is repeated for the next node in the k-Tree (if there exists). After all nodes (including each leaf node and each internal node (for a number of combined sub-systems: see Section 2.3)) are visited, the final process is applied, which is either 1) to allocate the best sub-system directly to the request (if its size is equal to that of the request) or 2) to partition the corresponding node (see Section 2.2) for the request (since its size is larger than that of the request).

Finally, whenever a task is finished, “*the k-Tree-based processor deallocation*” procedure is applied by searching for the location of the finished sub-system starts from the k-Tree’s root and goes to the subset path until reaching the leaf node that stores information of the finished task. After finding the corresponding k-Tree’s node of the finished task, its status is updated (or removed from the k-tree). Finally, the combining process is recursively applied from the finished node(s) to the root (if it is possible).

Note: the expand-node-size function (in the allocation process) will be applied if a combined sub-system is selected as the best sub-system for the current incoming task in order to limit the number of nodes in the k-Tree and provide the same methodology to update and partition as a regular (free) leaf node. Then, other corresponding nodes in the combined sub-system have to be updated as busy. Finally (in the deallocation process), these corresponding busy nodes will be free whenever that expanded node is free (in the resume-node-size function).

For example, assume that in the current system, there are three tasks allocated: Buddy#1 and Buddy#2 at level 2, Buddy#1 at level 3 and suppose that a new incoming task requests a sub-system of size  $5 \times 6$  (which is larger than each buddy node but less than a combined sub-system of size  $6 \times 6$ ).



## 2.6 Time Complexity Analysis

Let  $N$  be the system size ( $N = n_1 \times n_2 \times \dots \times n_k$ ),  $N_A$  be the maximum number of allocated tasks ( $N_A \leq N$ ),  $N_F$  be the corresponding number of free nodes in the  $k$ -Tree ( $N_A + N_F \leq N$ ), and  $M$  be the maximum number of nodes in the  $k$ -Tree (where  $M = \text{external (leaf) nodes} + \text{internal (non-leaf) nodes} < 2N$ ).

**THEOREM 1:** Time complexity of the  $k$ -Tree-based allocation to find the best free sub-system for each incoming task on a  $k$ -D mesh (of size  $N = n_1 \times n_2 \times \dots \times n_k$ ) is  $O(k^4 2^k (N_A + N_F) + k^2 2^{2k})$ .

*PROOF:* In the allocation algorithm (see Section 2.5), a number of recursive iterations of the DFS (depth first search) are at most a number of nodes in the  $k$ -Tree and only nodes whose sizes are larger than (or equal to) the request are visited. In this (non-bit map) approach, the number of nodes in the  $k$ -Tree is proportional to the number of allocated tasks or busy nodes ( $N_A$ ) and the number of free nodes ( $N_F$ ) in the  $k$ -Tree, where  $N_A + N_F \leq N$  and  $N_F \leq (2^k - 1)N_A$ . Since the number of external (or leaf) nodes in the  $k$ -Tree are at most  $N_A + N_F \leq N$  and the number of internal nodes are at most  $(\text{\#leaf nodes} - 1) / (2^k - 1)$ ; therefore the total number of nodes in the  $k$ -Tree is at most  $M$  nodes, where  $M = (N_A + N_F) + (N_A + N_F - 1) / (2^k - 1)$ . For each (free) leaf node (of  $N_F$  nodes), the best-fit value is computed in  $O(k^2)$  time and hence  $O(k^2 N_F)$  for all leaf nodes. For each internal node (of  $(N_A + N_F - 1) / (2^k - 1)$  nodes), the best-fit value is computed in  $O(k^2)$  time for each node and hence  $O(k^4 2^{2k})$  time for  $k^3 2^k + k^2 2^{2k}$  combined sub-systems, as summarized in Table 1. (Note that the maximum free size (in Step 0) is computed only once for each task by using DFS in  $O(k^3 2^k (N_A + N_F))$  time.) Finally, after finding the best free sub-system (Step 3), if its size is equal to the request, then it is directly allocated to the request. Otherwise, the network partitioning and the best sub-partition will be applied, which can be computed in  $O(k 2^k)$  time. Then, the corresponding node(s) in the  $k$ -Tree is updated in  $O(k)$  time. Note: for the combined sub-system that is larger than the request, before partitioning, expand-node-size process is applied in  $O(k^2 2^{2k})$  time. Thus, total time complexity to visit all nodes in the  $k$ -Tree is approximately  $[N_A + k N_F + k^3 2^k (N_A + N_F - 1) / (2^k - 1)] + [N_A + k^2 N_F] + [(k^4 2^{2k} + k^3 2^k) (N_A + N_F - 1) / (2^k - 1)] + [k^2 2^{2k} + 2k 2^k + k] = O(k^4 2^k (N_A + N_F) + k^2 2^{2k})$ , where  $N_A + N_F \leq N$ .

**Table 1.** Time complexity of the  $k$ -Tree-based approach for the  $k$ -D meshes.

Functions in each $k$ -Tree's node for sub-system Allocation (see Section 2.5)	Time complexity
(0) Before searching to find the best free sub-system	
- Compute a maximum free size (from all $M$ nodes in the $k$ -Tree)	
Busy Leaf + Free Leaf + Internal Node operations $[\cong N_A + k N_F + k^3 2^{2k} (N_A + N_F - 1) / (2^k - 1)]$	$O(k^3 2^k (N_A + N_F))$
(1) Leaf node operation (for $N_F$ nodes) : - Compute best-fit value $O(k^2)$ $[\cong N_A + k^2 N_F]$	$O(N_A + k^2 N_F)$
(2) Internal node operation (for $(N_A + N_F - 1) / (2^k - 1)$ nodes) $[\cong (k^4 2^{2k} + k^3 2^k) (N_A + N_F - 1) / (2^k - 1)]$	$O(k^4 2^{2k} (N_A + N_F))$
Sub-system combining: Algo.C.2 $O(k^2 2^k)$ and Algo.C.3 (next level) $O(k^2 2^{2k})$	
- Compute best-fit value (of all combined subsystems) $O(k^4 2^{2k})$	
(3) Partitioning after finding the best free node $[\cong k^2 2^{2k} + 2k 2^k + k]$	$O(k^2 2^{2k})$
- Expanding node (for a combined sub-system) $O(k^2 2^{2k})$	
- Best sub-partition $O(k 2^k)$	
- Network partitioning $O(k 2^k)$	
- Allocate (update $k$ -Tree) $O(k)$	
Total time (for $M$ nodes) = (0) + (1) + (2) + (3)	$O(k^4 2^k (N_A + N_F) + k^2 2^{2k})$

Note: Our  $k$ -Tree-based model, when applied to the 2-D/3-D meshes, provides a linear time complexity  $O(N_A + N_F)$

**THEOREM 2:** Time complexity of the  $k$ -Tree-based deallocation to free the particular  $k$ -Tree node that stores the finished task and to combine the free buddy nodes of the root sub-tree to the root of the  $k$ -Tree on the partitionable  $k$ -D mesh ( $N = n_1 \times n_2 \times \dots \times n_k$ ) is  $O(n 2^k + k^2 2^{2k})$ , where  $n = \max(n_1, n_2, \dots, n_k)$ .

*PROOF:* Let  $n$  be the maximum depth of the  $k$ -Tree ( $n = \max(n_1, n_2, \dots, n_k)$ ). Searching for the location of a finished sub-system from the root is at most  $n(2^k)$  steps. Then, combining all  $2^k$  buddy nodes from the finished sub-system to the root (if it is possible) takes another  $n(2^k)$  steps. Finally, the resume-node-size process of the expand-node-size process (if any) may be required in  $O(k^2 2^{2k})$  time. Therefore, total time complexity of the  $k$ -Tree-based deallocation is  $O(n 2^k + k^2 2^{2k} < M)$ .

### 3 Performance of the k-Tree-Based Approach on the 2-D Meshes

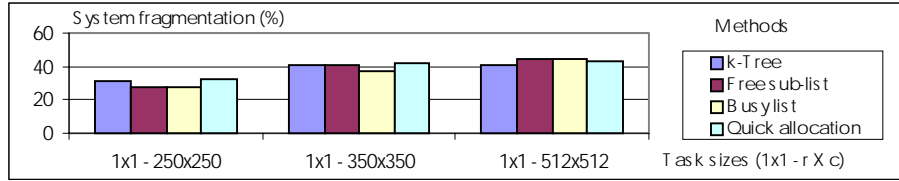
In order to evaluate the system performance, the generalized k-Tree-based approach was developed. By simulation study, a number of experiments are performed to investigate the effect of applying the “k-tree-based model” for performing processor allocation/deallocation for the partitionable 2-D meshes and compare to recently 2-D mesh-based strategies ([3], [15], [7]). The investigated system performance includes system utilization, system fragmentation, average allocation time, etc. For each experiment, a number of simulation time units are iterated around 5,000-50,000 time units and a number of incoming tasks are generated approximately 1,000-10,000 tasks, according to the setting of the system size parameter, the task size (i.e., row, column) parameter and the task size’s distribution. For each evaluated result, a number of different data sets are generated and the algorithm is repeated until an average system performance does not change (or at least 100 iterations). Experimental results of applying the k-Tree-based strategy are represented for both *static system performance* (with concerning processor allocation for incoming tasks (or jobs) only (or it is assumed that no task finishes during the considering time)) and *dynamic system performance* (with taking into account of deallocation for some finished tasks). In this study, in order to set the same incoming tasks and environment to all strategies for the comparison purpose, the static system performance is concerned (i.e., when we measure the system utilization and system fragmentation); otherwise the dynamic system performance is concerned. In each experiment, two task-size distributions are considered: the Uniform distribution  $U(\alpha, \beta)$  and the Normal distribution  $N(\mu, \sigma)$ . For each of these distributions, the system sizes ( $N = R \times C$ ) are varied and the task sizes [ $1 \times 1 - R \times C$ ] are generated, where  $\alpha = 1$ ,  $\beta = \max(R, C)$  for the Uniform distribution  $U(\alpha, \beta)$  and  $\mu = \sigma = \max(R, C)/2$  for the Normal distribution  $N(\mu, \sigma)$ . Other parameters are fixed such as task arrival rate  $\sim$  Poisson ( $\lambda$ ) (or inter-arrival time  $\sim \text{Exp}(1/\lambda=5)$ ), and service time  $\sim \text{Exp}(\mu=10)$ , etc.

In Experiment 1, we investigated “*the effect of system sizes to the system utilization ( $U_{\text{sys}}$ ) and the system fragmentation ( $F_{\text{sys}}$ )*”. In this experiment, the system sizes ( $N = R \times C$ ) were varied and the task sizes ( $1 \times 1 - R \times C$ ) were generated and fixed. In Table 2 (the system utilization result), for all test cases the k-Tree strategy performed  $\sim 60\%$  system utilization which was comparable to those of the recently 2-D mesh-based strategies (i.e., for the uniform distribution, the FSL, the BL, and the QA strategies yielded  $\sim 56\%$ ,  $\sim 57\%$ , and  $\sim 56\%$  system utilization, etc.). For the system fragmentation, the k-Tree approach and these existing strategies also performed the same results for the system fragmentation since there was no internal system fragmentation ( $F_{\text{sys}} = 1 - U_{\text{sys}}$ ). In Experiment 2, we investigated “*the effect of task sizes to the system utilization and the system fragmentation*”. In this case, the system size was fixed ( $N = 512 \times 512$ ) and the task sizes were generated and varied. In Fig. 9. (the system fragmentation result), for all test cases the k-Tree strategy performed the comparable system fragmentation to the FSL, BL, and QA strategies, which were  $\sim 30\%$ ,  $\sim 40\%$ , and  $\sim 41\%$  system fragmentation for task sizes [ $1 \times 1 - 250$ ], [ $1 \times 1 - 350 \times 350$ ], and [ $1 \times 1 - 512 \times 512$ ], respectively. For the system utilization, the k-Tree approach and these existing 2-D mesh-based strategies also performed the same results since  $U_{\text{sys}} = 1 - F_{\text{sys}}$  (or no effect of the internal system fragmentation). In Experiment 3, we investigated “*the effect of allocation time*” of the k-Tree and existing 2-D mesh-based strategies when the system sizes were increased. In Fig. 10. (the average allocation time), our k-Tree approach yielded the improved average allocation time, compared to the existing strategies for all tested cases, except when the system size was small ( $N = 64 \times 64$ ). In this case, the average allocation time of the k-Tree, FSL, and BL strategies were approximately constant since they depended on the number of allocated tasks ( $N_a$ ). However, the average allocation time of the QA strategy was increase linearly which depended on the number of allocated tasks ( $N_a$ ) and the system size ( $N$ ). In Experiment 4, we investigated “*the effect of*

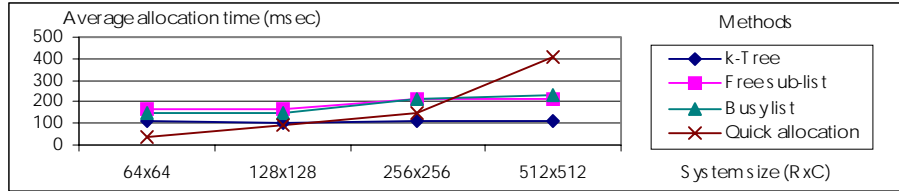
allocation and deallocation time” when the system sizes were increased. In Fig. 11.. (the average allocation and deallocation time), our k-Tree approach yielded the improved average allocation and deallocation time, compared to the existing strategies when system sizes were increased. In this case, the average allocation and deallocation time of the k-Tree was approximately constant while those of existing 2-D mesh-based strategies were increased linearly.

**Table 2.** Effect of “the system sizes” to the system utilization (%).

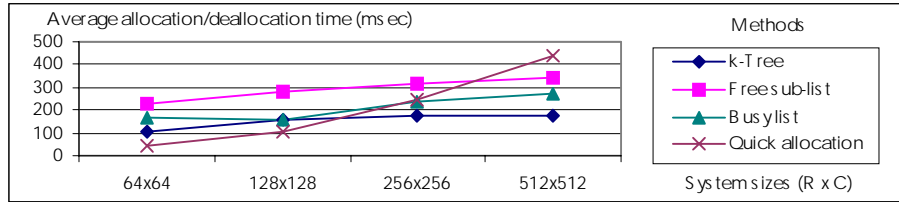
Task size Distributions	System sizes (N = RxC)	k-Tree	Free Sub-List (FSL)	Busy List (BL)	Quick Allocation (QA)
Uniform ( $\alpha, \beta$ ) $\alpha=1, \beta=\min(R,C)$ For [1x1 - RxC]	64 x 64	60.97	56.06	57.27	55.64
	128 x 128	60.11	56.98	56.67	57.84
	256 x 256	59.77	55.77	54.61	55.87
	512 x 512	58.86	56.11	55.91	56.53
Normal ( $\mu, \sigma^2$ ) $\mu = \min(R,C)/2$ For [1x1 - RxC]	64 x 64	61.18	61.42	58.46	57.53
	128 x 128	59.16	56.89	58.02	54.79
	256 x 256	60.10	58.02	55.12	55.09
	512 x 512	61.28	57.04	57.22	55.89



**Fig. 9.** Effect of “the task sizes” to the system fragmentation (%).



**Fig. 10.** Effect of “the system sizes” to the average allocation time.



**Fig. 11.** Effect of “the system sizes” to the average allocation and deallocation time.

## 4 Conclusion

In this paper, we present the design and the development of the “generalized k-tree-based sub-mesh allocation” model for the partitionable multi-dimensional mesh-connected systems. Time complexity of the k-Tree-based approach is  $O(N_A + N_F)$  for the partitionable 2-D and 3-D meshes and  $O(k^2 2^k (N_A + N_F) + k^2 2^{2k})$  for the partitionable k-D meshes, where  $N_A$  is the maximum number of allocated tasks and  $N_F$  is the corresponding number of free sub-meshes. By simula-

tion studies, a number of experiments were performed to investigate the system performance of applying the k-Tree-based model for the partitionable 2-D meshes. In the experimental results, the system performance (i.e., the system utilization and system fragmentation) of the k-Tree-based model for the partitionable 2-D meshes was comparable to existing 2-D mesh-based strategies. In addition, the k-tree-based approach yielded the improved allocation/deallocation decision time (i.e., the average allocation time, the average allocation and deallocation time), compared to those 2-D mesh-based strategies when the system sizes (N) are very large.

## 5 References

- [1] Chuang, P.J., Tzeng, N.F.: An Efficient Submesh Allocation Strategy for Mesh Computer Systems. In *Procs. of Int'l Conf. on Distributed Computing Systems*, May (1991) 256-263
- [2] Das Sharma, D., Pradhan, D. K.: A Fast and Efficient Strategy for Submesh Allocation in Mesh-Connected Parallel Computers. In *Procs. of the 5<sup>th</sup> IEEE Symp. on Parallel and Distributed Processing* (1993) 682-689
- [3] Das Sharma, D., Pradhan, D. K.: Submesh Allocation in Mesh Multicomputers Using Busy-List: A Best-Fit Approach with Complete Recognition Capability. *Journal of Parallel and Distributed Computing*, Vol. 36 (1996) 106-118
- [4] Ding, J., Bhuyan, L.N.: An Adaptive Submesh Allocation Strategy for Two-Dimensional Mesh Connected Systems. In *Procs. of Int'l Conf. on Parallel Processing*, Vol. II (1993) 193-200
- [5] Intel: A Touchstone DELTA System Description. Supercomputer Systems Division, Intel Corporation, Beaverton, OR 97006 (1991)
- [6] Intel: Paragon XP/S Overview. Supercomputer Systems Division, Intel Corporation, Beaverton, OR 97006 (1991)
- [7] Kim, G., Yoon, H.: 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*, Vol. 9(2) (1998) 175-185
- [8] Li, K., Cheng, K.H.: Job Scheduling in a Partitionable Mesh Using a Two-Dimensional Buddy System Partitioning Scheme. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2(4) (1991) 413-422
- [9] Liu, T., et. al.: A Submesh Allocation Scheme for Mesh-Connected Multiprocessor Systems. In *Proceedings of 1995 Int'l Conf. on Parallel Processing*, Vol. II (1995) 159-163
- [10] Mattson, et al.: Intel/Sandia ASCI system. In *Procs. of Int'l Parallel Processing Symposium* (1996)
- [11] Mohapatra, P.: Processor Allocation Using Partitioning in Mesh Connected Parallel Computers.: *Journal of Parallel and Distributed Computing*, Vol. 39 (1996) 181-190
- [12] Srisawat, J., Alexandridis, N.A.: Efficient Processor Allocation Scheme with Task Embedding for Partitionable Mesh Architectures. In *Procs. of Int'l Conf. on Computer Applications in Industry and Engineering*, Las Vegas, November (1998) 305-308
- [13] Srisawat, J., Alexandridis, N.A.: Reducing System Fragmentation in Dynamically Partitionable Mesh-Connected Architectures. In *Procs. of Int'l Conf. on Parallel and Distributed Computing and Networks*, Australia, December (1998) 241-244
- [14] Srisawat, J., Alexandridis, N.A.: A New Quad-Tree-Based Sub-System Allocation Technique for Mesh-Connected Parallel Machines. In *Procs. of the 13<sup>th</sup> ACM-SIGARCH Int'l Conf. on Supercomputing*, Greece, June (1999) 60-67
- [15] Yoo, S., et. al.: An Efficient Task Allocation Scheme for 2D Mesh Architectures. *IEEE Transactions on Parallel and Distributed systems*, Vol. 8(9) (1997) 934-942
- [16] Zhu, Y.: Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers. *Journal of Parallel and Distributed Computing*, Vol.16 (1992) 328-337