# A k-Tree-Based Resource (CU/PE) Allocation for Reconfigurable MSIMD/MIMD Multi-Dimensional Mesh-Connected Architectures*

Jeeraporn Srisawat[1] Wanlop Surakampontorn[2] and Nikitas A. Alexandridis[3]
[1]Faculty of Science, King Mongkut's Institute of Technology Ladkrabang
Bangkok 10520, THAILAND
[2]Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang
Bangkok 10520, THAILAND
[3]Department of Electrical and Computer Engineering
School of Engineering and Applied Science
The George Washington University
Washington, DC 20052, USA
e-mail: ksjeerap@kmitl.ac.th, kswanlop@kmitl.ac.th, alexan@seas.gwu.edu

**Abstract:** In this paper, we present a new generalized k-Tree-based (CU/PE) allocation model to perform dynamic resource (CU/PE) allocation/deallocation decision for the reconfigurable MSIMD/MIMD multi-dimensional (k-D) mesh-connected architectures. Those reconfigurable multi-SIMD/MIMD systems allow dynamic modes of executing tasks, which are SIMD and MIMD. The MIMD task requires only the free sub-system; however the SIMD task needs not only the free sub-system but also the corresponding free CU. In our new k-Tree-based (CU/PE) allocation model, we introduce two best-fit heuristics for the CU allocation decision: 1) the CU depth first search (CU-DFS) in $O(kN_F)$ time and 2) the CU adjacent search (CU-AS) in $O(k2^k)$ time. By the simulation study, the system performance of these two CU allocation strategies was also investigated. Our simulation results showed that the CU-AS and CU-DFS strategies performed the same system performance when applied for the reconfigurable MSIMD/MIMD 2-D and 3-D mesh-connected architectures.

## 1. Introduction

A partitionable multicomputer is a special type of parallel systems that provides (at run time) for executing various independent parallel/distributed applications (or tasks) on different sub-systems in parallel. For this system, each of these tasks requests an MIMD mode. The more flexible partitionable parallel system, called the reconfigurable multi-SIMD/MIMD system, provide independent sub-systems for the requested tasks, executing in the SIMD and MIMD modes. At run time some tasks may call the SIMD mode (which is good at synchronization and communication) whereas some tasks may need to execute independent branching or different instructions (which are suitable for the MIMD mode). Therefore, the dynamic reconfigurable MSIMD/MIMD parallel architecture has become increasingly important for the parallel and distributed computing environment. The SPP MSIMD/MIMD architecture [1] is the flexible design of reconfigurable MSIMD/MIMD systems since the roles of its processors, called CPEs (control processor elements), will be assigned to be either PE or CU at run time. This architecture performs dynamic reconfiguration at the network level (for any independent SIMD/MIMD task) not at the instruction level (that reconfiguring by altering the connections between the PEs in order to match the task graph or particular algorithm.)

In such a reconfigurable MSIMD/MIMD environment, a number of independent tasks (of the same or different applications) come in. Each of these tasks requires (at run time) a separate sub-system (or partition) to execute in either the SIMD mode or the MIMD mode. At the front-end computer, a special designed OS (known as the resource allocator and task scheduler) will provide appropriate free sub-systems for the new incoming task(s). In particular, that OS has to dynamically find the location of a free sub-system in order to allocate for each incoming task, as well as to deallocate a busy sub-system and recombine partitions as soon as they become available when a task completes. In the reconfigurable MSIMD/MIMD systems, the requested MIMD mode requires only the free sub-systems but the requested SIMD mode needs both the free sub-system and the corresponding free CU.

In the past five years, most existing processor (PE) allocation methods were introduced for partitionable multicomputers to allocate independent tasks, (executing in the MIMD mode) and for specific interconnection networks such as 2-D meshes. Those PE allocation strategies includes FRAME SLIDE [2], BUSY LIST with Scheduling [3], ADAPTIVE SCAN [4], FREE SUB-LIST [5], 2-D BUDDY [6], FREE LIST [7], BIT-MAP with Partition [8], QUAD TREE [9], QUICK ALLOCATION [11], and BIT MAP [12]). All of them were introduced (at the front-end computer) for the partitionable MIMD 2-D mesh-connected multicomputers. For the reconfigurable SPP MSIMD/MIMD architecture [1], the resource (CU/PE) allocation strategy, called bit-map BUDDY, for hypercube networks was also introduced. However, the bit-map BUDDY strategy was handled by the special OS at the back-end MSIMD/MIMD parallel system.

In this paper, we present a new generalized k-Tree-based (CU/PE) allocation model to perform dynamic resource (CU/PE) allocation decision (at the front-end computer) for the reconfigurable MSIMD/MIMD parallel systems, which utilize the multi-dimensional (k-D) mesh interconnection networks. This new generalized k-Tree-based (CU/PE) allocation model is extended from our previous study [10]. That k-Tree-based model was introduced for performing (PE) allocation for the partitionable MIMD k-D mesh-connected systems. Our new model covers the resource (CU/PE) allocation for the reconfigurable MSIMD/MIMD k-D mesh-connected architectures, which allows independent tasks, executing in the MIMD and SIMD modes. In addition, in order to complete the SIMD partition, we introduce two best-fit heuristics for the CU allocation decision: 1) the CU depth first search (CU-DFS) strategy in $O(kN_F)$ time and 2) the CU adjacent search (CU-AS) strategy in $O(k2^k)$ time. With the CU-AS strategy, our k-Tree-based (CU/PE) allocation model yields the same time complexity as that of the MIMD sub-system (PE) allocation in our previous study (when applied to 2-D and 3-D meshes). Finally, the system performance of these two strategies was also investigated and compared (in terms of system utilization and system fragmentation) by the simulation study. In particular, the results of applying our model to the reconfigurable MSIMD/MIMD 2-D and 3-D mesh-connected systems are presented.

Next section illustrates our new generalized k-Tree-based (CU/PE) allocation model to perform the resource allocation/deallocation decision for the reconfigurable multi-SIMD/MIMD k-D mesh-connected architectures. Section 3 presents the evaluated system performance of applying the new k-Tree-based (CU/PE) allocation model for some interconnection networks such as 2-D meshes and 3-D meshes. Finally, conclusion and future study are discussed in Section 4.

* This research was supported by the Thailand Research Fund under Grant PDF44-Jeeraporn Srisawat.

## 2. k-Tree-Based (CU/PE) Allocation Model for Reconfigurable MSIMD/MIMD k-D Meshes

Our generalized k-Tree-based (CU/PE) allocation model includes a k-Tree system state representation (Section 2.1) and algorithms for network partitioning (Section 2.2), sub-system combining (Section 2.3), best-fit heuristic (Section 2.4), searching for allocation/ deallocation decision (Section 2.5). This new generalized k-Tree-based (CU/PE) allocation model is extended from our previous study [10], applied only for the partitionable MIMD k-D meshes, to cover the reconfigurable multi-SIMD/MIMD k-D mesh-connected architectures. In particular, in this paper we introduce two best-fit heuristics for the CU allocation decision for the SIMD task (in section 2.4.2) to complete the SIMD partition in efficient time.

### 2.1 k-Tree System State Representation

We use a data structure, called a k-Tree to represent system states of the reconfigurable MSIMD/MIMD k-D mesh-connected system. In our k-Tree-based (CU/PE) allocation model, the number of nodes in the k-Tree are dynamic, corresponding to the number allocated tasks. At start, the k-Tree consists of only one (root) node, used to store the system information (i.e., a size, a base-address, a status, etc.) of the initialized system. During run time when many tasks are executing, each leaf node (or a sub-system) may be free (for incoming task(s)) or busy (for executing task(s)) and each internal node is partially available. In order to allocate an incoming task, each larger free node can be dynamically created and partitioned into $2^k$ buddies/node (see Figure 1). Note: in this new k-Tree-based model, any k-Tree node is modified to include a link to a CU for the SIMD task (see Figure 2).

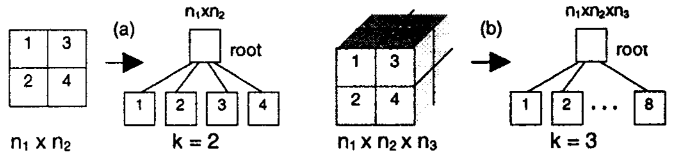


**Figure 1.** The $2^k$ buddies of the k-Trees: (a) 2-D mesh; and (b) 3-D mesh.

Figure 2 illustrates an example of the system state representation of applying the k-Tree-based (CU/PE) allocation model for allocating three SIMD tasks (of sizes 2x3, 2x2, and 1x5) and two MIMD tasks (of sizes 4x4 and 3x6) on an 8x10-mesh system.

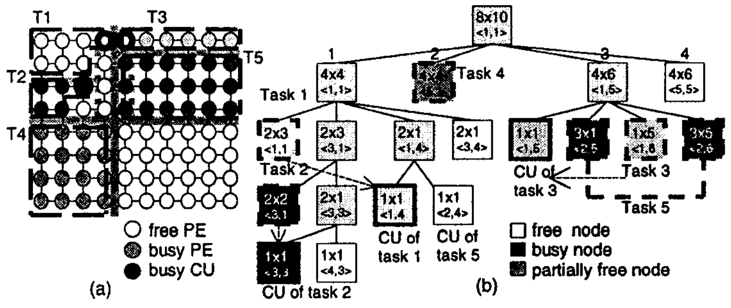


**Figure 2.** The k-Tree-based (CU/PE) allocation for 3 SIMD tasks and 2 MIMD tasks on an 8x10 mesh system: (a) the allocated system status and (b) the corresponding k-Tree system state representation.

### 2.2 Network Partitioning

The k-Tree-based network partitioning is the partitioning process that partitions all k dimensions of the k-D system ($N = n_1 x n_2 x \ldots x n_k$) into smaller $2^k$ sub-systems and allocates an appropriate one for the request (of size $p_1 \times p_2 \times \ldots \times p_k$, where $p_i \le n_i$, $i = 1, 2, \ldots, k$). In this paper, we utilize Buddy-ID-Address-Size-Conversion algorithm of our previous study. This network partitioning process (i.e., identifying #buddies = $2^k$, base-addresses, and sizes) is computed in $O(k2^k)$ time (see more detail in [10]). Note: the network partitioning will be applied and modified later (in Section 2.4.2) in order to handle the CU partitioning for a selected sub-system.

### 2.3 Sub-System Combining

The sub-system combining is applied during processor allocation or deallocation. We also utilize the combinations of $2^j$-Adjacent-Buddies algorithm of our previous study in order to combine $2^j$ buddies (where $j = 1, 2, \ldots, k-1$) 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 js (since $k2^1 + k2^2 + k2^3 + \ldots + k2^{k-1} = 2k[2^{k-1}-1]$). The k-Tree-based combining process is classified into four main groups: 1) Combining all buddies, 2) Combining some adjacent buddies, 3) Combining (adjacent) buddy(s) and corresponding adjacent sub-buddies, and 4) Combining some adjacent sub-buddies (see more details in [10]).

### 2.4 Best-Fit Heuristic

#### 2.4.1 Best-fit Heuristic for PE Allocation

The best-fit heuristic is to find the likely best free sub-system for an incoming task. For PE allocation decision, we also utilize the generalized best-fit heuristic [10] for the partitionable k-D meshes.

Best-Fit Criteria:

1. Find all free Ss that can preserve the "max free size" [ *O(k) time* ].
2. If many Ss have property (1), S that gives "min different size factor (diffSF)" is selected [ *$O(k^2)$ time* ].
3. If many Ss have (1)&(2), the "smallest size" S yielding "min combining factor (CF)" is selected [ *O(k) time* ]. Otherwise, select by random.
4. After all nodes are visited,
   - If the best S = request, then it is directly allocated to the request.
   - Otherwise it is partitioned and one of its buddies that yields "min modified CF (MCF)" will be selected [ *$O(k2^k)$ time* ].

Note: Criteria 1-3 are applied for every free node and combined S. But criterion 4 is computed only once for the best free S of Steps 1-3.

#### 2.4.2 Best-Fit Heuristic for CU Allocation

In the reconfigurable SPP MSIMD/MIMD design [1], CPEs (control processor elements) were added in the system and their roles (CU or PE) are assigned at run time. Therefore, a CU for a selected sub-system (S) can be any CPE that is directly connected to S. First, we introduce a generalized method to identify all possible CUs and their addressing. If the size of the selected sub-system (S) is $m_1 x\ m_2 x \ldots x\ m_k$ at address $<(a_1, a_2, \ldots, a_k), (b_1, b_2, \ldots, b_k)>$, then the number of all possible CUs are $2 \sum^k_{i=1} m_1 \text{ x } m_2 \text{ x} \ldots \text{x } m_{i-1} \text{ x1x } m_{i+1} \text{ x} \ldots \text{x } m_k$. For example, if k = 2 and S = 7 x 8, # possible CUs is 2(1x8 + 7x1) = 30.

In general, for k dimensions of S there are 2k (outside) sub-systems of CUs (CUSs). CUSs' addressing are defined as follows: for each dim i (of size = $m_1 x m_2 x \ldots x\mathbf{1}x m_{i+1} x \ldots x m_k$), where i=1,2,...,k.

Min CUS address = $<(a_1, a_2, \ldots, \mathbf{a_i\text{-}1}, \ldots, a_k), (b_1, b_2, \ldots, \mathbf{a_i\text{-}1}, \ldots, b_k)>$

Max CUS address = $<(a_1, a_2, \ldots, \mathbf{b_i+1}, \ldots, a_k), (b_1, b_2, \ldots, \mathbf{b_i+1}, \ldots, b_k)>$

For example, if k = 2 and S = 7 x 8, addressing of all 30 CUs (or 4 CUSs) for a selected sub-system ($S = m_1 x\ m_2 = 7x\ 8$) (at $<(a_1, a_2), (b_1, b_2)> = <(5, 5), (11, 12)>$) are

- For a fixed dim 1,
  - a min CUS (8 PEs): $<(\mathbf{a_1\text{-}1}, a_2), (\mathbf{a_1\text{-}1}, b_2)> = <(4, 5), (4, 12)>$
  - a max CUS (8 PEs): $<(\mathbf{b_1+1}, a_2), (\mathbf{b_1+1}, b_2)> = <(12,5),(12,12)>$
- For a fixed dim 2,
  - a min CUS (7 PEs): $<(a_1, \mathbf{a_2\text{-}1}), (b_1, \mathbf{a_2\text{-}1})> = <(5, 4), (11, 4)>$
  - a max CUS (7 PEs): $<(a_1, \mathbf{b_2+1}), (b_1, \mathbf{b_2+1})> = <(5,13),(11,13)>$

For any free node R (of size $d_1 \text{ x } d_2 \text{ x} \ldots \text{x } d_k$) in the k-Tree, there are 2k inside sub-systems at boundary (BSs). BSs' addressing are defined as follows: for each dimension i (each of size $[(d_1\text{-}2)x\ (d_2\text{-}2)\ x \ldots x\ (d_{i-1}\text{-}2)\ x\ \mathbf{1}\ x\ d_{i+1} x \ldots x\ d_k]$), where i = 1, 2, ..., k.

Min BS addr=$<(a_1+1, a_2+1, \ldots, \mathbf{a_i}, a_{i+1}, \ldots, a_k), (b_1\text{-}1, b_2\text{-}1, \ldots, \mathbf{a_i}, b_{i+1}, \ldots, b_k)>$

Max BS addr=$<(a_1+1, a_2+1, \ldots, \mathbf{b_i}, a_{i+1}, \ldots, a_k), (b_1\text{-}1, b_2\text{-}1, \ldots, \mathbf{b_i}, b_{i+1}, \ldots, b_k)>$

For example, if k = 2 and R = 7 x 8, addressing of all 26 PEs (or 4 BSs) for a free node (R) of size $d_1 \text{ x } d_2 = 7 \text{ x } 8$) (at $<(a_1, a_2), (b_1, b_2)> = <(5, 5), (11, 12)>$) are

- For a fixed dim 1,
  - a min BS (8 PEs): $<(\mathbf{a_1}, a_2), (\mathbf{a_1}, b_2)> = <(5, 5), (5, 12)>$
  - a max BS (8 PEs): $<(\mathbf{b_1}, a_2), (\mathbf{b_1}, b_2)> = <(11,5), (11,12)>$
- For a fixed dim 2,
  - a min BS (5 PEs): $<(a_1+1, \mathbf{a_2}), (b_1\text{-}1, \mathbf{a_2})> = <(6, 5), (10, 5)>$
  - a max BS (5 PEs): $<(a_1+1, \mathbf{b_2}), (b_1\text{-}1, \mathbf{b_2})> = <(6,12), (10,12)>$

### 2.4.2.1 The CU Depth First Search (CU-DFS)

The CU depth first search (CU-DFS) is used to find any free node R that is adjacent to the selected sub-system S. The searching starts from the root and goes to the left most (leaf) node. If it is free, it then will be checked whether it is adjacent to S. If so, its best-fit value (Section 2.4.1) is computed. Then, new S and R will be updated if they yield the better best-fit value. The above process is repeated for the next free node (if there exists). Time complexity of the CU-DFS is $O(kN_F)$. In order to identify any boundary free PEs of a free node whether or not it is adjacent to the selected sub-system S, we define the adjacent status in O(k), as follows:

```
Let $S_1$ is an selected sub-system S with expanded boundary (Figure 3)
    $S_2$ is a free node, i = 1, 2.
    $|S_i|$ is $n_{i1} \times n_{i2} \times \ldots \times n_{ik}$ at $<(a_{i1}, a_{i2},\ldots,a_{ik}), (b_{i1}, b_{i2}, \ldots, b_{ik})>$, j=1,2,..., k.
Then, $S_1$ is adjacent to $S_2$ if they are not disjoint and are different only
one bit. "Disjoint status" can be identified (O(k)) as for ∃j, j = 1,2,..., k
    $S_1$ and $S_2$ are *disjoint* either if $(a_{1j} - a_{2j} \geq n_{2j})$ or if $(a_{2j} - a_{1j} \geq n_{1j})$
```

For example, suppose k = 2 and a selected sub-system is S ($m_1$ x $m_2$) and a free node is R ($d_1$ x $d_2$). Figure 3.a illustrates the adjacent status of S (10) and R (11), which are satisfied both not-disjoint and one bit different. Figure 3.b shows the non-adjacent status of S (10) and R (01) since they are not disjoint but are different in 2 bits.

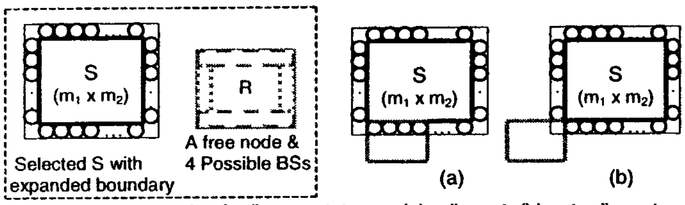


Figure 3. An example of adjacent statuses: (a) adjacent; (b) not adjacent.

### 2.4.2.2 The CU Adjacent Search (CU-AS)

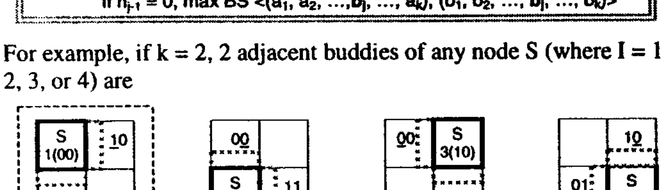
The CU adjacent search (CU-AS) is another approach used to find any free node R that is adjacent to the selected sub-system S. In this method, the searching starts from S and its adjacent nodes can be identified directly (see the following 4 cases). If the node R is free, its corresponding BSs are identified and its best-fit value (Section 2.4.1) is computed. New S and R will be updated if they have the better best-fit value. Time complexity of the CU-DFS is $O(k2^k)$.

In order to identify any boundary free PEs of a free node (R) whether or not it is adjacent to the selected sub-system S, we define the adjacent buddies in $O(k^2)$ for any non-combined sub-system (S) or $O(k2^k)$ for any combined sub-system (S).

Case 1: if S is any buddy node (1, 2, ..., or $2^k$), we compute its BS(s) in $O(k^2)$ time.

```
Let S is a selected sub-system ($S = m_1 \times m_2 \times \ldots \times m_k$)
    I is a Buddy ID of S, where I = 1, 2, ..., or $2^k$.
Then k Adjacent Buddies can be identified by
    1. Convert I to a binary strings ($b_{k-1} \ldots b_1 b_0$)
    2. Compute k adjacent Buddies: for dim j, j = 1, 2, ..., k
       negate $b_{j-1}$ ($b_{k-1} \ldots \underline{n_{j-1}} \ldots b_1 b_0$),
       if $n_{j-1}$ =1, min BS $<(a_1, a_2,\ldots,a_j, \ldots, a_k), (b_1, b_2, \ldots, a_j, \ldots, b_k)>$
       if $n_{j-1}$ = 0, max BS $<(a_1, a_2, \ldots,b_j, \ldots, a_k), (b_1, b_2, \ldots, b_j, \ldots, b_k)>$
```

For example, if k = 2, 2 adjacent buddies of any node S (where I = 1, 2, 3, or 4) are

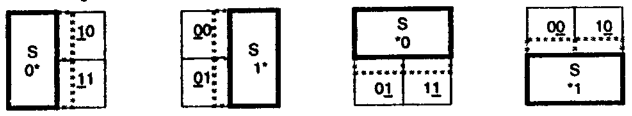


↑ - Negate dim 1 ($b_1\underline{n_0}$ => 0$\underline{1}$), Compute minBS <($a_1$,$a_2$), ($a_1$,$b_2$)>
- Negate dim 2 ($\underline{n_1}b_0$ => $\underline{1}$0), Compute minBS <($a_1$,$a_2$), ($b_1$,$a_2$)>

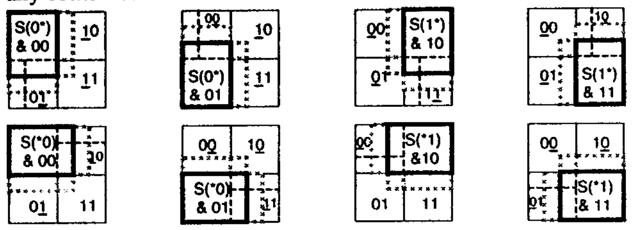
Case 2: if S is any combined ($2^{k-j}$) buddy node (j = 1, 2, ..., k-1), we compute its BS(s) in $O(k2^k)$.

```
Let S is a selected sub-system (or a combined S of $2^{k-j}$, j = 1, 2, ..., k-1)
    T is a ternary string of S = ($t_{k-1} \ldots t_1 t_0$)
There exist js' * in T [i.e., k Ts are ($b_{k-1} \ldots b_j$ *..*), ($b_{k-1} \ldots b_{j+1}$ *..*$b_0$),...,
(*$b_{k-2} \ldots b_{j-1}$*..*] and $(k-j)2^j$ adjacent Buddies of each T are identified by
  - Negate a non* dim (from min to max Nos.), set each of **..* = 0/1 &
    compute minBS (n=1) or maxBS (n=0)
    Example, for j = 1 & T = ($b_{k-1} \ldots b_2 b_1$ *), $(k-1)2^1$ adjacent nodes are
      negate $b_1$ ($b_{k-1} \ldots b_2 n_1$ *), set dim 1 = 0/1,
            if n=1, compute min BS $<(a_1, a_2, a_3,\ldots, a_k), (b_1, a_2, b_3,\ldots, b_k)>$
            If n=0,compute max BS $<(a_1, b_2, a_3,\ldots, a_k), (b_1, b_2, b_3,\ldots, b_k)>$
 , . . . , negate $b_{k-1}$ ($n_{k-1} \ldots b_2 b_1$ *), set dim 1 = 0/1,
            if n=1, compute min BS $<(a_1, a_2, a_3,\ldots, a_k), (b_1, b_2, b_3,\ldots, a_k)>$
            if n=0,compute max BS $<(a_1, a_2, a_3,\ldots, b_k), (b_1, b_2, b_3,\ldots, b_k)>$
    for j = k-1 &T = ($b_{k-1}$ *...**), $2^{k-1}$ adjacent nodes are
      negate $b_{k-1}$ ($n_{k-1}$ *...**), set dim 1, 2,...,,k-1= 0/1(min->max).
```

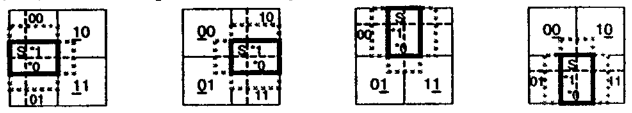
For example, if k = 2, adjacent buddies of any combined S are



Case 3: if S is any combined ($2^{k-j}$) buddy(s)&($2^j$) sub-buddies node (j=1,2,...,k-1), all adjacent buddies can be identified by applying Case 1 and Case 2 in $O(k2^k)$. For example, if k=2, adjacent buddies of any combined S are



Case 4: if S is any combined ($k2^{k-1}$) sub-buddies node (j=1,2,...,k-1), all adjacent buddies can be identified by applying Case 1 and 2 in $O(k2^k)$. For example, if k=2, adjacent buddies of any combined S are



## 2.5 Searching for Allocation/Deallocation

In the k-Tree-based (CU/PE) allocation procedure, searching starts from the root and perform DFS (depth first search) by visiting the left most (leaf) node. If that node is free and its size can accommodate the request, its best-fit value is computed. For an SIMD task, either the CU-DFS or CU-AS strategy is applied. Then, the best (SIMD/MIMD) sub-system (S) is updated if the new free S yields the better best-fit value. The above process is repeated for the next node in the k-Tree including all external (leaf) nodes and internal nodes. After all nodes are visited, the final process is applied to either 1) allocate the best sub-system directly to the request (if its size is equal to that of the request) or 2) partition the corresponding node for the request (if its size is larger than that of the request).

Whenever a task is finished, the k-Tree-based (CU/PE) deallocation procedure is applied by searching for the location of the finished sub-system starts from the root and goes to the subset path until reaching the leaf node that stores information of the finished task. After finding that k-Tree's node of the finished task, its status is updated. Finally, the combining process is recursively applied (to remove free internal nodes(s)) from both PE and CU partitions to the root (if it is possible).

<u>**THEOREM 1**</u>: Time complexity of the k-Tree-based (CU/PE) allocation with applying the CU-AS strategy (to find the best free sub-system for each incoming task) on a k-D mesh is $O(k^5 2^{2k}(N_A+N_F))$. $N_A$ is the max allocated tasks ($N_A \leq N$), $N_F$ is the corresponding free nodes in k-Tree ($N_A+N_F \leq N$ and $N_F \leq (2^k-1)N_A$).

PROOF: Since #external (or leaf) nodes are at most $N_A + N_F \leq N$ and #internal nodes are at most (#leaf nodes-1) divided by $(2^k-1)$, therefore all nodes (M) = $(N_A+N_F) + (N_A+N_F-1) / (2^k-1) < 2N$. [For each (free) leaf node, the best-fit value is computed in $O(k^3)$ and $O(k^3N_F)$ for all free nodes. For each internal node, the best-fit value is computed in $O(k^5 2^{2k})$ for all combined sub-systems and $O(k^5 2^k(N_A+N_F))$ for all internal nodes.] Finally, after all nodes are visited, if the best S's 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 in $O(k^3 2^{2k})$. Thus, total time complexity of the k-Tree-based (CU/PE) allocation with applying the CU-AS heuristic is $O(k^5 2^k(N_A+N_F))$, where $N_A+N_F \le N$ and hence $O(N_A+N_F)$ when applied to the 2-D/3-D meshes.

Note: time complexity of the k-Tree-based (CU/PE) allocation with applying the CU-DFS heuristic is $O(k^4 2^k(N_A+N_F)\ (kN_F))$ and hence $O(N_F(N_A+N_F))$ when applied to the 2-D/3-D meshes.

THEOREM 2: Time complexity of the k-Tree-based (CU/PE) deallocation (to free the particular k-Tree node that stores the finished task and to combine the free internal nodes to the root of the k-Tree) on a k-D mesh is $O(n2^k)$, where $n=\max(n_1,n_2,\ldots,n_k)$.

PROOF: 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. Therefore, total time complexity is $O(n2^k < M)$.

## 3. System Performance Evaluation

By simulation study, a number of experiments were performed to investigate the system performance effect (i.e., system utilization and fragmentation) of applying our k-tree-based (CU/PE) allocation model for the reconfigurable MSIMD/MIMD 2-D and 3-D meshes. For each experiment, (simulation) time units were iterated around 5,000-50,000 units and incoming tasks were generated around 1,000-10,000 tasks, according to the system parameter(s) setting. For each evaluated result, different data sets were generated and the algorithm was repeated until an average system performance does not change. The Uniform distribution $U(\alpha, \beta)$ was considered for the task-size distribution. Task arrival rate ~ Poisson($\lambda$) (or inter-arrival time ~ Exp($1/\lambda=5$)), and service time ~ Exp($\mu=10$). Note: in order to set the same incoming tasks and environment to both CU allocation strategies for the comparison purpose, we assumed that no task finishes during the considering time.

In Experiment 1, we investigated the effect of system sizes to the system utilization ($U_{sys}$), where the system sizes ($N=n_1 x n_2$) were varied and the task sizes ($1x1 - {}^{n1}/_2 x {}^{n2}/_2$) were generated and fixed. For all test cases the CU-AS and CU-DFS strategies performed the same system utilization (since these methods were different only when sub-system (S) and task (T) sizes were equal which hardly occurred.). Table 1 showed the results ($\%U_{sys}$) of applying the k-Tree-based (CU/PE) allocation for 2-D and 3-D meshes, which yielded the same results when increasing percentage of SIMD tasks.

**Table 1.** Effect of the system sizes to the system utilization (%).

| 2-D Mesh | | | | 3-D Mesh | | | |
|---|---|---|---|---|---|---|---|
| $N=n_1 x n_2$ | 0% | 10% | 20% | N | 0% | 10% | 20% |
| 64x64 | 68.76 | 68.46 | 71.91 | n=32 | 58.14 | 58.55 | 58.60 |
| 128x128 | 68.25 | 67.82 | 67.82 | n=64 | 49.73 | 54.32 | 54.33 |
| 256x256 | 70.16 | 70.16 | 70.16 | n=128 | 50.52 | 50.52 | 50.52 |

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

| Task size | 0% | 10% | 20% | 50% |
|---|---|---|---|---|
| 1x1-64x64 | 81.403 | 78.585 | 78.265 | 78.200 |
| 1x1-128x128 | 70.158 | 70.160 | 70.162 | 69.007 |
| 1x1-256x256 | 59.995 | 59.995 | 59.995 | 59.997 |

In Experiment 2, we investigated the effect of task sizes to the system utilization, where the system size was fixed (N = 256x256) and the task sizes were varied. Table 2 showed that the system utilization increased when the maximum task-size parameter was reduced since a number of small tasks could be allocated. For the system utilization, these strategies performed the same results since $U_{sys} = 1 - F_{sys}$ (or no effect of internal system fragmentation).

## 4. Conclusion and Future Study

This paper introduces two best-fit heuristics for the k-Tree-based CU allocation: 1) the CU-DFS strategy in $O(kN_F)$ and 2) the CU-AS strategy in $O(k2^k)$. The CU allocation is added to complete the design of the new generalized k-tree-based (CU/PE) allocation model for the reconfigurable MSIMD/MIMD k-D mesh-connected architectures. By simulation study, a number of experiments were performed to investigate system performance of applying our new k-Tree-based (CU/PE) allocation model for reconfigurable 2-D and 3-D meshes. System performance results (i.e., system utilization & fragmentation) of applying our model with including the CU-AS strategy showed the same results to those of the CU-DFS strategy. However, for the 2-D or 3-D meshes the CU-AS strategy yields O(1) time which is better than $O(N_F)$ time performed by the CU-DFS strategy.

In the future study, we will modify and add the CU searching to some existing 2-D mesh-based PE allocation methods. Those modified strategies can support SIMD tasks for the reconfigurable MSIMD/MIMD 2-D meshes. Therefore, the system performance of those (CU/PE) allocation methods will be investigated and compared to our k-Tree-based (CU/PE) allocation approach.

## References

[1] M. S. Baig, T. El-Ghazawi, and N. A. Alexandridis, "Single Processor-Pool MSIMD/MIMD Architecture," *in proceeding of Fourth IEEE Symposium on Parallel and Distributed Processing*, pp. 460-467, Texas, December 1992.

[2] P. J. Chuang and N. F. Tzeng, "Allocating Precise Submesh in Mesh Connected Systems," *IEEE Transaction on Parallel and Distributed Systems*, v.5(2), pp. 211-217, 1994.

[3] D. Das Sharma and D. K. Pradhan, "Job Scheduling in Mesh Multicomputers," *IEEE Transactions on Parallel and Distributed Systems*, v.9(1), pp. 57-70, 1998.

[4] J. Ding and L.N. Bhuyan, "An Adaptive Submesh Allocation Strategy for Two-Dimensional Mesh Connected Systems", *in Proceeding of International Conference on Parallel Processing*, vol. II, pp.193-200, 1993.

[5] G. Kim and H. Yoon, "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*, v.9(2), pp. 175-185, 1998.

[6] K. Li and K.H. Cheng, "Job Scheduling in a Partitionable Mesh Using a Two-Dimensional Buddy System Partitioning Scheme," *IEEE Transactions on Parallel and Distributed Systems*, v.2(4), pp. 413-422, 1991.

[7] T. Liu and et. al., "A Submesh Allocation Scheme for Mesh-Connected Multiprocessor Systems," *in Proceeding of International Conference on Parallel Processing*, pp. 159-163, vol. II, 1995.

[8] P. Mohapatra, "Processor Allocation Using Partitioning in Mesh Connected Parallel Computers," *Journal of Parallel and Distributed Computing*, pp. 181-190, v. 39, 1996.

[9] J. Srisawat and N.A. Alexandridis, "A New Quad-Tree-Based Sub-System Allocation Technique for Mesh-Connected Parallel Machines," *in Proceeding of the 13th ACM-SIGARCH International Conference on Supercomputing*, pp. 60-67, Rhodes, Greece, June 1999.

[10] J. Srisawat and N.A. Alexandridis, "A Generalized k-Tree-Based Model to Sub-System Allocation for Partitionable Multi-Dimensional Mesh-Connected Architectures," *in Proceeding of the 3rd International Symposium on High Performance Computing*, pp. 205-217, Springer publisher, Tokyo, Japan, October 2000.

[11] S. Yoo and et. al., "An Efficient Task Allocation Scheme for 2D Mesh Architectures," *IEEE Transactions on Parallel and Distributed systems*, v. 8(9), pp. 934-942, 1997.

[12] Y. Zhu, "Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers," *Journal of Parallel and Distributed Computing*, v.16, pp. 328-337, 1992.