สำนักงานกองทุนสนับสนุนการวิจัย
THE THAILAND RESEARCH FUND

รายงานวิจัยฉบับสมบูรณ์

โครงการ

วิธีสากลในการจัดสรรหน่วยควบคุมและหน่วยคำนวณสำหรับระบบคอมพิวเตอร์งานขนานที่สามารถปรับระบบเป็นเอ็มเอสไอเอ็มดี/เอ็มไอเอ็มดี

A Universal Resource (CU and PE) Allocation for Dynamic Reconfigurable MSIMD/MIMD Parallel systems

โดย ผศ. ดร. จีรพร ศรีสวัสดิ์ และคณะ

14 กุมภาพันธ์ 2547

สัญญาเลขที่ PDF/94/2544

รายงานวิจัยฉบับสมบูรณ์

โครงการ

วิธีสากลในการจัดสรรหน่วยควบคุมและหน่วยคำนวณสำหรับระบบคอมพิวเตอร์งานขนานที่สามารถปรับระบบเป็นเอ็มเอสไอเอ็มดี/เอ็มไอเอ็มดี

A Universal Resource (CU and PE) Allocation for Dynamic Reconfigurable MSIMD/MIMD Parallel systems

คณะผู้วิจัย	สังกัด
1. ผศ. ดร. จีรพร ศรีสวัสดิ์	สถาบันเทคโนโลยีพระจอมเกล้าฯ ลาดกระบัง
2. ศ. ดร. วัลลภ สุระกำพลธร	สถาบันเทคโนโลยีพระจอมเกล้าฯ ลาดกระบัง
3. Prof. Dr. Nikitas A. Alexandridis	George Washington University, Washington DC, USA

สนับสนุนโดยสำนักงานกองทุนสนับสนุนการวิจัย

กิจกรรมประกาศ

ก่อนอื่นผู้วิจัย ขอขอบพระคุณ นักวิจัยที่ปรึกษา ศ. ดร. วันลภ สุระกำพลธร จากภาควิชาวิศวกรรมอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง และ Professor Dr. Nikitas A. Alexandridis จาก Department of Electrical and Computer Engineering, School of Engineering and Applied science, the George Washington University, Washington DC, USA สำหรับคำแนะนำ และแนวทางในการทำวิจัยหลังปริญญาเอก ตั้งแต่เริ่มโครงการจนกระทั่งงานวิจัยสำเร็จลุล่วงไปด้วยดี และสิ่งสำคัญที่ทำให้งานวิจัยนี้มีขึ้นได้ ผู้วิจัยขอขอบพระคุณ สำนักงานกองทุนสนับสนุนการวิจัย (สกว) ที่ให้การสนับสนุนด้านเงินทุน ด้วยทุนวิจัยหลังปริญญาเอก ปี 2544 – 2446 และท่านผู้อำนวยการฝ่ายวิชาการ ศ. ดร. วิชัย บุญแสง และคณะกรรมการฝ่ายวิชาการ คุณพรพิมล กิตติมศักดิ์ และเจ้าหน้าที่ประจำโครงการทุกท่าน ที่สนับสนุนให้งานวิจัยดำเนินไปด้วยดี และสำเร็จตามเป้าหมาย ท้ายสุดผู้วิจัยขอขอบพระคุณครอบครัว โดยเฉพาะคุณ วิสุทธิ์ ศรีสวัสดิ์ สำหรับความรัก ความเข้าใจ และกำลังใจ ตลอดช่วงเวลาของการทำวิจัย

ACKNOWLEDGEMENT

First of all, I would like to thank my mentors, "Professor Dr. Wanlop Surakampontorn" at the department of Electrical Engineering, Faculty of engineering, King Mongkut's Institute of Technology, Ladkrabang, Bangkok, THAILAND and "Professor Dr. Nikitas A. Alexandridis" at the department of Electrical and Computer Engineering, School of Engineering and Applied Science, the George Washington University, Washington DC, USA, for their direction and suggestion during all phases of this postdoctoral research. Also I appreciate to give gratefully thanks the Thailand Research Fund (trf) for the financial support with "the postdoctoral research fund in 2001 – 2003" to this research and the trf academic director "Professor Dr. Wichai Boonsang", the trf academic committees, Mrs. Pompimon Kittimasak and the trf official project staffs for all academic support until reaching the research goal. Lastly, I would like to thank my family especially my husband "Wisoot Srisawat" for his support, understanding, and love.

บทคัดย่อ

รหัสโครงการ PDF/94/2544

ชื่อโครงการ วิธีสากลในการจัดสรรหน่วยควบคุมและหน่วยคำนวณสำหรับระบบคอมพิวเตอร์งานขนานที่สามารถปรับระบบเป็นเอ็มเอสไอเอ็มดี/เอ็มไอเอ็มดี

ชื่อนักวิจัยและสถาบัน 1. ผศ. ดร. จีรพร ศรีสวัสดิ์ สถาบันเทคโนโลยีพระจอมเกล้าฯ ลาดกระบัง

2. ศ. ดร. วัลลภ สุระกำพลธร สถาบันเทคโนโลยีพระจอมเกล้าฯ ลาดกระบัง

3. Prof. Dr. Nikitas A. Alexandridis George Washington University, USA

E-mail Address 1. ksjeerap@kmitl.ac.th 2. kswanlop@kmitl.ac.th 3. alexan@seas.gwu.edu

ระยะเวลาของโครงการ: 2½ ปี (15 สิงหาคม 2544 - 14 กุมภาพันธ์ 2547)

ระบบคอมพิวเตอร์งานขนานที่สามารถปรับระบบเป็นเอ็มเอสไอเอ็มดีและหรือเอ็มไอเอ็มดี เป็นระบบคอมพิวเตอร์งานขนานที่มีความยืดหยุ่นและสามารถแบ่งเป็นระบบย่อยได้ เพื่อประมวลผลชิ้นงานแบบขนานหลายๆงาน ซึ่งแต่ละงานสามารถประมวลผลโหมดที่ต่างกันได้ คือ โหมดเอสไอเอ็มดีหรือโหมดเอ็มไอเอ็มดี ระบบคอมพิวเตอร์งานขนานที่สามารถปรับระบบได้ทั้งสองโหมดนี้มีบทบาทสำคัญ ในการประมวลผลแบบขนานและแบบกระจาย รวมทั้งงานที่ต้องการสมรรถนะในการประมวลผลสูง โดยในช่วงเวลาที่มีการจัดสรรทรัพยากร เช่น จัดสรรหน่วยคำนวณ งานขนานบางชนิดอาจต้องการโหมดเอสไอเอ็มดี สำหรับการทำงานหลายข้อมูลพร้อมกันโดยคำสั่งเดียว ในขณะที่งานบางชนิดอาจต้องการโหมดเอ็มไอเอ็มดี สำหรับการทำงานแบบหลายคำสั่งที่ต่างกันได้ ส่วนงานวิจัยในอดีตที่เกี่ยวกับการจัดสรรหน่วยคำนวณ โดยมากเป็นการจัดสรรให้กับหลายๆงานขนานที่เป็นอิสระกันแบบโหมดเอ็มไอเอ็มดีเท่านั้น นอกจากนั้นวิธีต่างๆดังกล่าวยังมีข้อจำกัดอีกก็คือ ถูกออกแบบมาสำหรับรูปแบบการติดต่อสื่อสารแบบเฉพาะเช่น ไฮเปอร์คิวส์หรือทูดีเมิชส์ สำหรับงานวิจัยนี้ เสนอแบบอย่างสากลเพื่อจัดสรรทรัพยากร (ซียู/พีอี) สำหรับระบบคอมพิวเตอร์งานขนานที่สามารถแบ่งเป็นระบบย่อยและปรับระบบเป็นเอ็มเอสไอเอ็มดีและหรือเอ็มไอเอ็มดีได้ วิธีนี้สามารถประยุกต์ใช้สำหรับการติดต่อสื่อสารทุกแบบที่เป็นแบบผลคูณ เช่น เมิชส์หลายมิติ ไฮเปอร์คิวส์ เอ็นอเรย์เคคิวส์ เป็นต้น และวิธีนี้ยังครอบคลุมไปถึงการประมวลผลหลายๆงานที่ต้องการโหมดเอสไอเอ็มดีและหรือโหมดเอ็มไอเอ็มดี โดยที่ขนานแบบเอ็มไอเอ็มดีต้องการเพียงระบบย่อยแต่งานขนานแบบเอสไอเอ็มดีต้องการระบบย่อยพร้อมด้วยหน่วยควบคุม ดังนั้นงานวิจัยนี้ จึงเสนอวิธีจัดสรรหน่วยคำนวณวิธีใหม่ สำหรับงานขนานแบบเอ็มไอเอ็มดี โดยใช้โครงสร้างข้อมูลแบบไบนารีทรี ซึ่งเป็นวิธีที่มีประสิทธิภาพสูงกว่าวิธีที่ใช้โครงสร้างข้อมูลแบบเคทรี และเสนอวิธีจัดสรรหน่วยควบคุม สำหรับระบบย่อยแบบเอสไอเอ็มดี เพื่อให้การจัดสรรทรัพยากรมีความสมบูรณ์สำหรับหลายๆระบบย่อยแบบเอสไอเอ็มดีและหรือเอ็มไอเอ็มดี โดยเป็นวิธีที่สามารถประมวลผลได้อย่างมีประสิทธิภาพ และในขั้นสุดท้ายเป็นการวัดประสิทธิภาพของวิธีสากลที่เสนอ โดยการเปรียบเทียบผลที่มีต่อระบบด้วยวิธีใหม่แบบไบนารีทรี กับผลด้วยวิธีแบบเคทรีที่ปรับเพิ่มสำหรับงานขนานทั้งสองโหมด ในการศึกษาด้วยการจำลองแบบพบว่า ทั้งสองวิธีให้ผลการทดลองที่ใกล้เคียงกัน นอกจากนั้นได้เสนอผลการเปรียบเทียบสำหรับระบบแบบทูดีเมิชส์ โดยวิธีใหม่แบบไบนารีทรี วิธีแบบเคทรีที่ปรับเพิ่ม และวิธีแบบทูดีเมิชส์ที่ผู้อื่นเสนอ ซึ่งให้ผลการทดลองที่ใกล้เคียงกัน โดยวิธีใหม่แบบไบนารีทรีมีประสิทธิภาพเร็วกว่า

คำหลัก: แบบวิธีสากลในการจัดสรรทรัพยากร (ซียู/พีอี) วิธีการจัดสรรหน่วยคำนวณ (พีอี) วิธีการจัดสรรหน่วยควบคุม (ซียู) สถาปัตยกรรมที่สามารถปรับระบบได้เป็นหลายเอ็มเอสไอเอ็มดีและหรือเอ็มไอเอ็มดี ระบบคอมพิวเตอร์งานขนานที่สามารถแบ่งเป็นระบบย่อยได้

ABSTRACT

Project Code: PDF/94/2544

Project Title: A Universal CU and PE Allocation for Dynamic Reconfigurable MSIMD/MIMD Parallel System

Investigator:

1. Asst. Prof. Dr. Jeeraporn Srisawat, King Mongkut's Institute of Technology Ladkrabang
2. Prof. Dr. Wanlop Surakampontorn, King Mongkut's Institute of Technology Ladkrabang
3. Prof. Dr. Nikitas A. Alexandridis, George Washington University, Washington DC, USA

E-mail Address: 1. ksjeerap@kmitl.ac.th 2. kswanlop@kmitl.ac.th 3. alexan@seas.gwu.edu

Project Period: 2½ years (August 15, 2001 - February 14, 2004)

Reconfigurable MSIMD/MIMD systems are generic and flexible partitionable parallel systems that provide sub-systems for dynamic tasks, each of which need a specific executing (SIMD or MIMD) mode. During allocation time, some tasks may call the SIMD mode for their synchronization whereas some tasks may need the MIMD mode to execute independent different instructions. Therefore, the reconfigurable MSIMD/MIMD architecture has become increasingly important in both parallel and distributed computing and high performance computing. In the past, most existing processor allocation strategies were introduced for the partitionable multicomputer to allocate independent tasks in the MIMD mode. In addition, those methods are limited in their designs for particular interconnection networks such as hypercubes or 2-D meshes. In this study, we present "a universal model" to perform dynamic resource (CU/PE) allocation decision for the reconfigurable and partitionable MSIMD/MIMD parallel systems. Our model can be applicable for all networks in the product network class, including multi-dimensional meshes, hypercubes, n-ary k-cubes, etc. Moreover, this universal model can be utilized for the reconfigurable MSIMD/MIMD systems that allow various dynamic tasks executing in the MIMD and SIMD mode in different partitions. In such special system, the MIMD task requires only the free sub-system but the SIMD task needs the free sub-system as well as the corresponding free CU. For MIMD tasks, the new generalized PE allocation method is introduced, based upon the binary tree, which improves time complexity over that of the recent k-Tree-based approach. For the SIMD partition, the generalized CU al location s trategy is introduced to complete the MSIMD/MIMD partitions in efficient time. Finally, in the system performance evaluation on the MSIMD/MIMD systems, we presented the comparative performance of our new universal binary-tree based model to the modified k-Tree-based approach to cover both MIMD and SIMD tasks. By simulation study, the results showed that our binary-tree-based approach yielded the comparable system performance to those of the k-tree-based strategy. In addition, we also compared the system performance of our binary-tree-based model, when applied on the 2-D mesh-connected systems, to those of other recent 2-D mesh-based allocation strategies. Our binary-tree-based results and modified k-tree-based results for the partitionable 2-D meshes were also comparable to those of the existing 2-D mesh-based strategies in efficient time.

Keywords: Universal resource (CU/PE) allocation model, processor (PE) allocation method, control unit (CU) allocation strategy, reconfigurable multi-SIMD/MIMD architectures, partitionable parallel systems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

A Universal CU and PE Allocation for Dynamic Reconfigurable MSIMD/MIMD Parallel Systems*

Jeeraporn Srisawat
ksjeerap@kmitl.ac.th
Department of Mathematics and Computer Science, Faculty of Science
King Mongkut's Institute of Technology Ladkrabang (KMITL), Bangkok, THAILAND

Wanlop Surakampontorn
kswanlop@kmitl.ac.th
Department of Electrical Engineering, Faculty of Engineering
King Mongkut's Institute of Technology Ladkrabang (KMITL), Bangkok, THAILAND

and
Nikitas A. Alexandridis
alexan@seas.gwu.edu
Department of Electrical and Computer Engineering
School of Engineering and Applied Science
The George Washington University (GWU), Washington DC, USA

1. INTRODUCTION

A partitionable multicomputer is a special type of parallel computers that provides at run time for executing various independent parallel/distributed applications (or tasks) on different sub-systems in parallel. For that system, each of these tasks requests an MIMD mode. The more flexible partitionable parallel system, called the reconfigurable MSIMD/MIMD system, provide independent sub-systems for the requested tasks, each of which need to process in either SIMD or MIMD mode. At execution time, some tasks may require the SIMD mode, which is good at synchronization and communication. Also some tasks may need to execute independent branching or different instructions which are suitable for the MIMD mode. Thus, the reconfigurable MSIMD/MIMD parallel system has become increasingly important for the parallel and distributed computing environment. In computer architecture research, examples of the reconfigurable MSIMD/MIMD architectures include SPP [4], PM4 [5], REPLICA [32], MAP [36], NETRA [38], and PASM [40]. Such reconfigurable MSIMD/MIMD systems perform reconfiguration or partition at the network level for a number of independent tasks and hence not at the instruction level for any task graph or specific algorithm, which reconfigure by altering the connections between processors (PEs). The SPP (single pool processor) MSIMD/MIMD architecture [4] is more generic and flexible in design than those existing reconfigurable architectures. In that study, results showed that the SPP MSIMD/MIMD architecture provided the improved system performance over existing reconfigurable MSIMD/MIMD architectures. Consequently, our study concentrates on designing the resource (CU/PE) allocation model for the new computer architecture design in the area of parallel and distributed systems.

In the reconfigurable and partitionable MSIMD/MIMD environment (see Figure 1), a number of independent smaller tasks from 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 SIMD or MIMD modes. At the front-end computer, a special designed operating system (OS) known as the resource allocator and task scheduler is introduced for this reconfigurable system. Its major responsibility is to dynamically find the appropriate location of a free sub-system in order to allocate for each incoming task, as well as to deallocate a specific sub-system and recombine partitions as soon as they become available when a task completes.

* This research was supported by the Thailand Research Fund (trf) under grant PDF/94/2544

In the past five years, most existing processor (PE) allocation methods were introduced for the partitionable multicomputer to execute in the MIMD mode. Those existing methods are also limited in their designs for particular interconnection networks that are either the hypercubes or the 2-D meshes. For the partitionable hypercubes, examples of hypercube-based PE allocation schemes include the extend BUDDY [1], the GRAY CODE [6], the fast BUDDY [9], the dynamic BINARY TREE & FREE LIST [11], the maximum SUB CUBE [20], the FREE LIST [25], the multi-queue BUDDY [34], the least overlap FREE LIST [37], the modified PRIME CUBE GRAPH [49]). For the partitionable 2-D meshes, 2-D mesh-based PE allocation strategies include the FRAME SLIDE [8], the BUSY LIST [14], the BUSY LIST with reserved task scheduling [15], the ADAPTIVE SCAN [18], the FREE SUB-LIST [26], the 2-D BUDDY [28], the FREE LIST [31], the BIT-MAP with partition [35], the QUAD TREE [45], the QUICK ALLOCATION [50], and the BIT MAP [52]. Those existing PE allocation algorithms were introduced to be applied at the front-end computer for the partitionable MIMD parallel systems. For the design of the reconfigurable SPP MSIMD/MIMD architecture, the resource (CU/PE) allocation strategy [2], called the modified bit-map BUDDY, was proposed for some network applications such as the hypercubes and the 2-D meshes. That resource (CU/PE) allocation was performed by the special OS at the back-end parallel system. Although the SPP architecture is the general design for many interconnection networks, the modified bit-map BUDDY strategy was limited and modified differently for the hypercube or 2-D mesh network since they were designed based upon existing PE allocation/deallocation strategies introduced at that time.

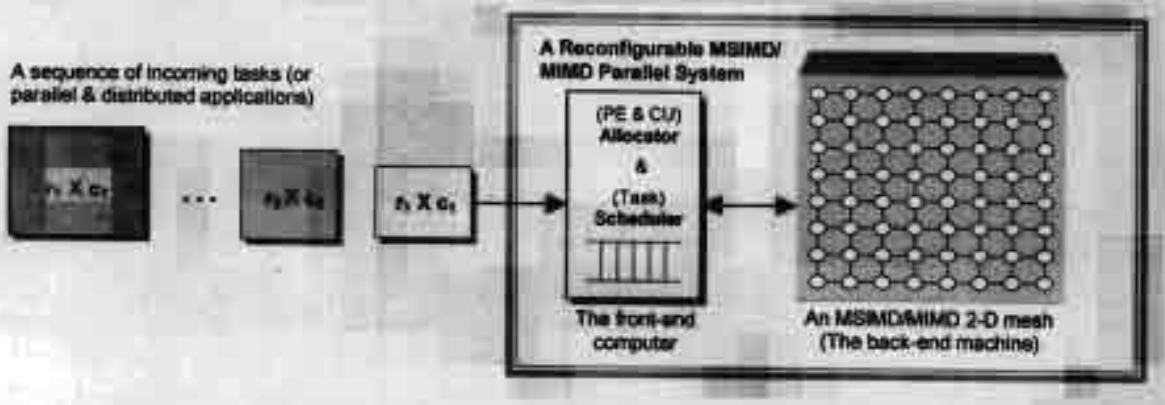


Figure 1: A reconfigurable and partitionable MSIMD/MIMD parallel and distributed computing environment.

In our study, we present "a universal binary-tree-based (CU/PE) allocation model" to perform dynamic sub-system allocation decision at the front-end computer for the reconfigurable and partitionable MSIMD/ MIMD parallel systems. Our model can be applicable for all interconnection networks that are in the product network class. In general, the product network configuration is a family of interconnection networks that includes multi-dimensional meshes, multi-dimensional tori, hypercubes, n-ary k-cubes, etc. Our universal model contains a binary-tree system state representation and allocation/deallocation algorithms as well as their time complexity analyses. For specific allocation/deallocation functions, various generalized algorithms are introduced such as network partitioning, sub-system combining, best-fit heuristic, and searching for allocation/deallocation decision. In particular, our universal (CU/PE) allocation model is designed with the new idea based upon the binary tree structure along with some modification from our previous study "the unified k-Tree-based PE allocation model [41]". Our original k-Tree-based model allows only the MIMD tasks for the partitionable parallel systems but our new binary-tree-based model can handle both MIMD and SIMD tasks for the reconfigurable and partitionable MSIMD/MIMD system. In such flexible system, each MIMD task requires only the free sub-system (or PE partition) but each SIMD task needs both the free sub-system and the corresponding free CU. Then for MIMD partitions, we introduce the new binary-based PE allocation method, which improves time complexity over that of our previous study "the k-Tree-based PE allocation strategy". For SIMD partitions, we introduce the binary-based CU allocation decision and modified the k-Tree-based PE allocation to complete the SIMD partition

in efficient time. By simulation study, we presented the system performance evaluation of our universal resource (CU/PE) allocation model on the reconfigurable MSIMD/MIMD systems. First, we presented the comparative performance of our universal binary-tree based model to the modified k-Tree-based approach. Next, we also evaluated the system performance of our model, when applied on the 2-D mesh-connected systems, to those of other recent 2-D mesh-based allocation strategies. Many experiments were performed and the system performance was presented in terms of system utilization, system fragmentation, etc.

Next section reviews some related research in the reconfigurable MSIMD/MIMD architectures and the study of existing processor allocation methods. In Section 3, we introduce the universal binary-tree-based model and the modified k-Tree-based to perform resource (CU/PE) allocation/deallocation decision for the reconfigurable MSIMD/MIMD parallel systems as well as corresponding time complexity analysis. Section 4 shows applications of our universal model for some particular interconnection networks such as 2-D meshes and hypercubes. Section 5 presents the evaluated system performance of our universal model. Finally, conclusions and future study are discussed in Section 6.

2. RELATED RESEARCH

This section presents the generic models of existing MSIMD/MIMD architectures (in Section 2.1), the resource (CU/PE) allocation strategies for the reconfigurable MSIMD/MIMD systems (in Section 2.2), and the existing processor allocation strategies for the partitionable MIMD systems (in Section 2.3).

2.1 Reconfigurable MSIMD/MIMD Architectures

The partitionable parallel system is designed for executing various independent tasks on different sub-systems in parallel. Usually, each of these tasks requests an MIMD mode. The reconfigurable MSIMD/MIMD parallel system is a flexible partitionable parallel system that supports for tasks that request to execute in SIMD and MIMD modes. In the computer architecture research, examples of early MSIMD/MIMD designs are PM4 [5], REPLICA [32], MAP[36], NETRA [38], and PASM [40]. In these MSIMD/MIMD architectures, the system composes of two major parts [4]: a base system and a partition control system. The base system is similar to that of the distributed-memory multiprocessor. The partition control system (PCS) introduces the additional hardware (CU-PE network) to handle the SIMD partition. In addition, these architectures are designed around two pools of processors, called the double-pool processor (DPP) design (see Figure 2): one for processing elements (PE) and another for control units (CU), where each processor's role is fixed for assigning as either PE or CU at their design time.

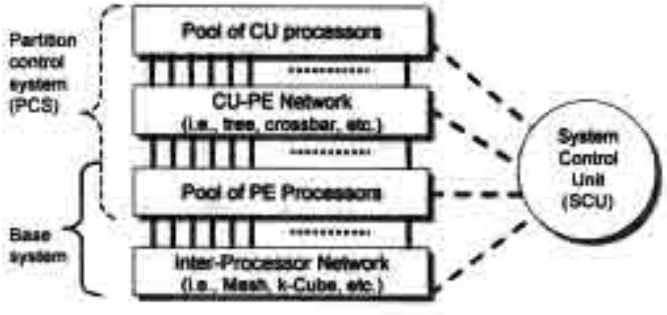


Figure 2: The DPP (Double-Pool Processor) Architecture.

The major different among those early MSIMD/MIMD architectures are the PASM and NETRA systems apply the fixed-PE partitioning whereas the REPLICA, MAP, and PM4 introduce the variable-PE partitioning. In the fixed-PE partitioning, each CU can be permanently connected to a fixed-size block of

PEs using tree networks. In such implementations, a partition is formed by combining a number of CUs for the desired PE partition. Since each partitioned size is equal to integral number of blocks, it may cause internal fragmentation for arbitrary task sizes. However, the advantage is that the CU-PE networks for these systems are inexpensive. In the variable-PE partitioning, the CU-PE network applies an MxN crossbar switch to provide flexible assignments of any CUs to any requested PE partition. Although this design could improve the performance, such implementations are generally expensive due to the cost of the crossbar. Moreover, using separate PE and CU pools still causes a performance bottleneck. For example, if the system runs out of CUs, no more SIMD tasks can be allocated. Another possibility is that if all PEs are allocated, then remaining CUs must be idle.

The new design, called the single-pool processors (SPP) architecture [4], was introduced (see Figure 3) to overcome that performance bottleneck of the early DPP design. Each of processors in the SPP pool, called a control processor element (CPE), is designed to be dynamically assigned the role of CU or PE at the allocation time. In the SPP design, the partitionable broadcasting network (PBN) is introduced to dynamically form a broadcasting bus between a selected CU and all processors in its PE partition.

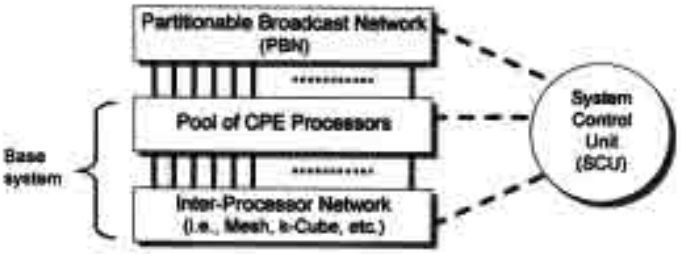


Figure 3: The SPP (Single-Pool Processor) Architecture.

The PBN network is made of N switching nodes (see Figure 4) to dynamically vary connections of the partitions by combining many features from both fixed and dynamic interconnection networks. The switch boxes are connected together into the same topology of the base fixed-interconnection network. Therefore, the number of switch I/O ports in a switching node will be equal to the degree of the network. In the resource allocation, after a set of CPEs have been configured as PEs and an additional CPE has been configured as a CU, we obtain a traditional SIMD sub-machine, as shown in Figure 5 and 6 for some examples of the 2-D mesh-connected system and the hypercube-connected system, respectively.

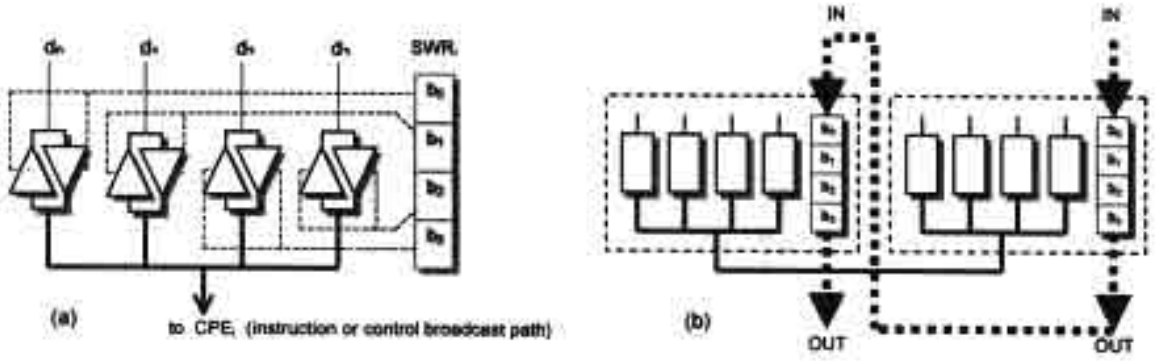


Figure 4: a) A switch node design for the PBN and b) An example of switch expansion.

Note that since the SSP architecture provide the general system base similar to the distributed-memory multiprocessors or multicomputers, except adding the PBN hardware. In our resource (CU/PE) allocation study, we adopt the reconfigurable SPP MSIMD/MIMD architecture [4], as the system base of our proposed universal resource (CU/PE) allocation model (presented in Section 3).

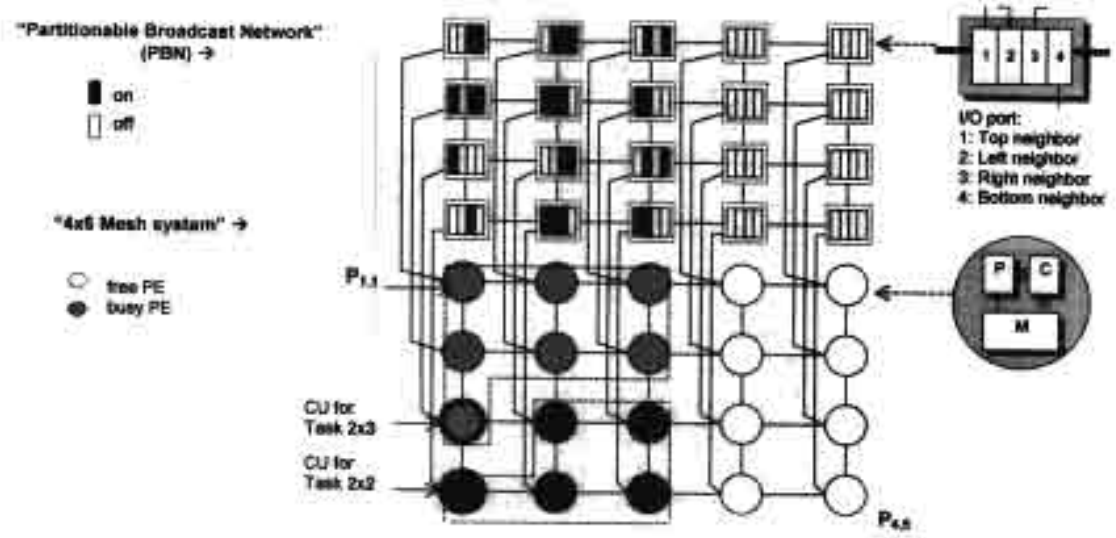


Figure 5: An example of the 2-D mesh-based MSIMD/MIMD architecture with the PBN.

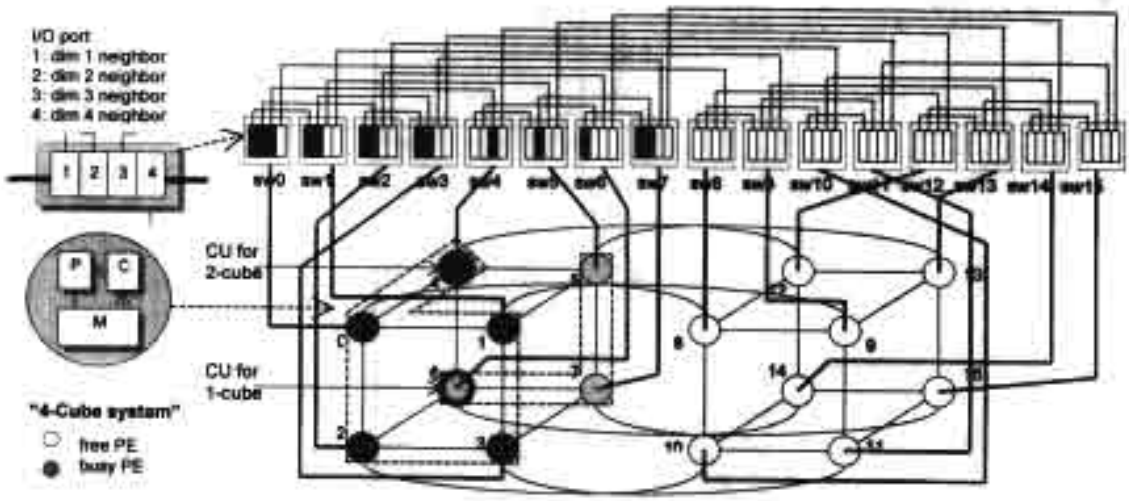


Figure 6: An example of the cube-based MSIMD/MIMD with the PBN.

2.2 Resource Allocation for Reconfigurable MSIMD/MIMD Systems

For the design of the reconfigurable SPP MSIMD/MIMD architecture [4], the resource (CU/PE) allocation strategy, called the modified bit-map BUDDY strategy [2], was introduced and performed by the special OS at the back-end parallel system. That bit-map Buddy-based approach was proposed for two specific interconnection networks that are the hypercubes and the 2-D meshes. Although the SPP architecture is the general design for many interconnection networks, the bit-map BUDDY strategy was limited and modified differently for the hypercube network or the 2-D mesh network. This is because they were modified based upon existing processor allocation/deallocation strategies introduced at that time.

The major steps in the resource (CU/PE) allocation method at the back-end parallel system are performed as follows. First, configuring an SIMD or MIMD sub-system starts with allocating a free PE partition. If the request is an SIMD mode, a corresponding free CU is allocated and then the PBN is configured into a broadcasting link between that CU and its partition. In particular for the hypercube networks, the CU algorithm is introduced in O(k) time (see Figure 7), where k is the dimension of the hypercube systems.

```
ALGORITHM: (Allocate-CUBE-MIN-ID, PE-MAX-ID)

begin
  for all PEs in the partition do in parallel
     CU-found = FALSE;
  end-for;

  /* First, all PEs in the PE partition search concurrently for a partition CO */
  for i=0 to n-1 do
      if (CU-found=FALSE) AND (neighbor[i] is NOT in the partition)
          AND (neighbor[i] is FREE)) then
            CU-id = neighbor[i];
            CU-found = TRUE;
      end-if;
  end-for;

  /* Then, this is used to select a CU-id with the smallest id in the partition */
  for i = n-1 down to 0 do
      for all PEs in the partition do in parallel
          if (ith bit in own-id is 1) then
              send (CU-id, CU-found) to neighbor[i];
      end-for;
      for all PEs in the partition do in parallel
          if (ith bit in own id=0) AND (received (CU-found) = TRUE)) then
               CU-id = min (CU-id, received(CU-id);
               CU-found = TRUE;
      end-for;
  end-for;
end.
```

Figure 7: The CU allocation algorithm of the SPP architecture for the hypercube networks.

2.3 Processor Allocation for Partitionable Parallel Systems

For allocating a PE partition (or sub-system), many processor allocation strategies are already available for the partitionable 2-D mesh-connected systems (see Section 2.3.1) and the partitionable hypercube-connected systems (see Section 2.3.2).

2.3.1 PE Allocation for Partitionable 2-D Mesh-Connected Systems

Consider an initial partitionable 2-D mesh-connected system of size N = R x C, where R is the number of rows and C is the number of columns in the system. P(i, j) will be one of its processing elements (PEs) at coordinate address (i, j) or the i^{th} row and the j^{th} column in the system, where P(1,1) denotes the PE at the Top Left (or upper leftmost position) and P(R, C) denotes the PE at the Bottom Right (or bottom rightmost position). Hence any partitionable 2D mesh-connected system will be identified by using these two addressing numbers: its base address, called the Top Left point (TLpt) and its cover address, called the Bottom Right point (BRpt), which is computed by using the base address and the system size.

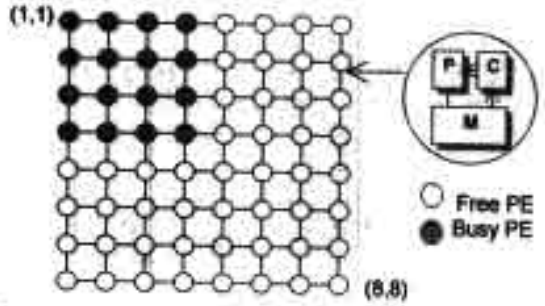


Figure 8: An example of a mesh-connected system (N = 8x8): the system status with an allocated task (4x4).

For the initial partitionable system, (TLpt) = (1, 1) and (BRpt) = (R, C) and thus the system location is denoted as <(1, 1), (R, C)>. A sub-system S(r, c) of size p = r X c will be identified and allocated to an incoming task by specifying its two addressing numbers: its base address TLpt = (x, y) and cover address BRpt = (x', y'), where $x' = x + r - 1 \leq R$ and $y' = y + c - 1 \leq C$. Figure 8 shows an example of an 8 x 8

mesh-connected system at the location <(1, 1), (8, 8)> and a (4 x 4) task that has been allocated into its subsystem at the location <(1, 1), (4, 4)>.

2.3.1.1 The Two-Dimensional Buddy Strategy

The first fit 2-D BUDDY strategy [28] [29], proposed in 1991, was an early study in processor allocation for partitionable 2-D mesh parallel systems ($N = 2^n$). In this strategy, an array of linked lists was used to store all available square sub-systems (2^p x 2^p, p = 1, 2, ..., n/2). Therefore, searching to find a free (square) sub-system for a request was done in O(log N) time. However, "sub-system combining" (executed in order to accommodate deallocation of system resources when a task completes) required O(N) time. Because this method was applied efficiently only for a square (power of 2) system and square-sized requested sub-systems; it caused high internal system fragmentation, especially for non-square requests.

2.3.1.2 The Frame Slide Strategy

Later in 1991, the first fit FRAME SLICE processor allocation method [8] [10] was proposed as a solution to the problem of internal fragmentation of the 2-D BUDDY strategy. This method allowed any mesh system size (N = R x C) and any requesting task size (r X c). A linked list was used to store only the allocated sub-systems ($N_s \leq N$). A candidate frame (starting at the left-most free PE) was created and compared with allocated frames in the list. If there was no intersection, then that frame was available; otherwise, that frame was slided horizontally by r rows (or vertically by c columns), to find a new candidate frame in $O(N_s.N/(rXc))$ time. The disadvantage however, was that sliding the frame by a factor of r (or c) might skip over some available sub-systems.

2.3.1.3 First Fit and Best Fit Bit-Map Strategies

In 1992, the first fit and best fit BIT MAP scheme [52] were introduced in order to solve the allocation miss problem in the FRAME SLICE strategy. In such bit-map approach, a 2D-array system status (N = R x C) is used to store a free/busy status-bit for every processor. For an incoming request (r X c), all N bits have to be identified at least twice to find the corresponding available sub-system; therefore, the time complexity of this bit-map method is O(N). The best-fit BIT MAP strategies yielded the better system utilization than the FRAME SLICE strategy. However, this strategy did not provide task rotation (c X r).

2.3.1.4 The Adaptive Scan Strategy

In 1993, the first fit ADAPTIVE SCAN strategy [18] was presented as another solution to the allocation miss problem of the FRAME SLICE strategy. This strategy introduced the task rotation to improve the recognition capability. It uses a busy list to store all allocated tasks ($N_s \leq N$) and the task rotation method that can allocate either S(r, c) or S(c, r) for a requested task (r X c)). Time complexity for the AS strategy to find the first free sub-system S(r, c) (or S(c, r)) for a task is $O(N_s.N)$ time. For system performance evaluation, performance results of this ADAPTIVE SCAN strategy improved over those of the FRAME SLICE method.

2.3.1.5 The First Fit with Partition Strategy

In 1996, a combining approach (i.e. FRAME SLICE with partition and BIT-MAP with partition) [35] was proposed to combine previous existing strategies (such as FRAME SLICE and BIT MAP) with predefined static partitions. This combining method improved time complexity as well as system performance by allocating requests with similar sizes close to each other (or into appropriate static partitions). This method could improve the time complexity over existing strategies by a factor of log N. For system performance study, system fragmentation of the BIT-MAP with partition strategy was almost identical to that of the BIT MAP strategy.

2.3.1.6 The Busy List Strategy

The best fit BUSY LIST strategy was proposed in 1993 [12] and 1996 [14]. This strategy improved time

complexity as well as system fragmentation over the best fit BIT MAP method. It uses "a busy list" to store allocated sub-systems and proposes the "maximum boundary value (max BV)" as the best-fit criteria. For an incoming request task (r X c), all (up to 8) candidate sub-systems of size S(r, c) or S(c, r) are created from each of the 4 corners of a particular allocated sub-system. Then, the candidate sub-mesh with the maximum BV is stored. After all N_a allocated sub-systems are identified, the candidate sub-mesh with maximum BV is selected to allocate for the task. This BUSY LIST allocation process takes $O(N_a^3)$ time, where $N_a \leq N$, but the deallocation process is only O(1) time.

2.3.1.7 The Free List Strategy

The best fit FREE LIST strategy [31] was proposed in 1995 in order to improve time complexity and system fragmentation over the best fit BIT MAP. The FREE LIST strategy used an array of linked lists to store all free sub-systems in increasing order by row (of all lists) and by column (in each list). For an incoming requested task (rXc), the first sub-system in the list number "r" is considered. If it is larger than the request, then 2-8 candidate sub-systems of size S(r, c) or S(c, r) are created and the one with the maximum boundary value (similar to the term defined in the BUSY LIST strategy) is selected to allocate to the request. The time complexity of the FREE LIST is $O(N_f^2)$ for both allocation and deallocation, where N_f is the number of free sub-systems ($N_f \leq N$).

2.3.1.8 The Quick Allocation Strategy

In 1997, the best fit QUICK ALLOCATION strategy [50] was proposed in order to improve time complexity of the existing allocation method. In that strategy, the following data structures were used: (1) a busy sub-system list (of allocated N_a tasks), (2) a coverage sub-system list, and (3) reject areas. For an incoming task (rXc), the searching process was to find an available sub-system. The allocation process began by computing the coverage sub-system list and the reject areas. Then, all coverage sub-systems were sorted in non-decreasing order. For each row (starting from 1 to R of N = R x C), find a free sub-system that did not intersect with the coverage sub-systems and the rejected areas and allocate it to the request. The time complexity of the QUICK ALLOCATION is $O(N_a\sqrt{N})$, where ($N_a \leq N$), and the system performance improved over that of the ADAPTIVE SCAN strategy.

2.3.1.9 The Free Sub-List Strategy

In early 1998, the best fit FREE SUB-LIST strategy [26], was proposed to improve the average waiting time performance by trying to minimize the amount of potential system fragmentation and preserve as many large free sub-systems as possible for subsequent tasks. A list of free sub-systems was used to store only free sub-systems (N_f), sorted in decreasing order of size values. Among all free sub-systems (N_f x (up to 8) sub-systems were computed from 4 corners of each free sub-system in the list), the one that yielded the minimum degree of fragmentation was selected for allocating to the task. The time complexity of this FREE SUB-LIST processor allocation strategy is $O(N_f^2)$ for an incoming task and the deallocation time complexity is $O(N_f^3)$ for deallocation of the finished task and also for computing all free sub-systems. Simulation results showed that the FREE SUB-LIST strategy improves system performance over those of the FREE LIST and BUSY LIST methods.

2.3.1.10 The Busy List with Reserved Scheduling Strategy

The BUSY LIST with "reserved task scheduling" strategy [15] was proposed in 1998 in order to improve average waiting time over the best fit BUSY LIST strategy with FCFS scheduling. For an incoming requested task, the regular best-fit BUSY LIST [14] is applied first. If there is no available sub-system, then a scheduler (for reserved task scheduling) will be applied by considering all possible (busy) sub-systems and finally the best sub-system is reserved, based on the maximum reserved value since it usually tends to cause the minimum system fragmentation. In the BUSY LIST with reserved task scheduling strategy, both allocation and deallocation time complexity is $O(N_a^3)$, where N_a is the number of allocated tasks ($N_a \leq N$). In simulation results, the BUSY LIST with reserved task scheduling improves the average delay time by about 20-50% over the best fit BUSY LIST with FCFS scheduling.

2.3.2 PE Allocation for Partitionable Hypercube-Connected systems

A k-dimensional partitionable hypercube- (or k-cube-)connected system (see Figure 9) is defined recursively as $Q_k = Q_{k-1} \times Q_{k-1}$, where k is the dimension of the Hypercube ($k > 1$) and k-bit addresses of any two neighbor processors are different in only one bit. Any processing element (PE) has a unique integer value ranging between 0 to $2^k - 1$.

Let Σ be a ternary symbol set (where $\Sigma = \{0, 1, *\}$ and * is a DON'T CARE symbol), which is used to represent 3^k sub-cubes in the k-Cube system. Let a sub-cube (q-cube) of all 3^k sub-cubes is represented by Q_q, where $q = 0, 1, 2, \ldots, k$, and its address represented by using a string of the ternary symbols in Σ. For example, all possible addresses of a sub-cube Q_3 in a 4-Cube system may be 0***, 1***, *0**, *1**, **0*, **1*, ***0, or ***1. Note: any sub-system is called a busy sub-system if all of its PEs are busy for executing a task and any sub-system is called a free sub-system if all of its PEs are free.

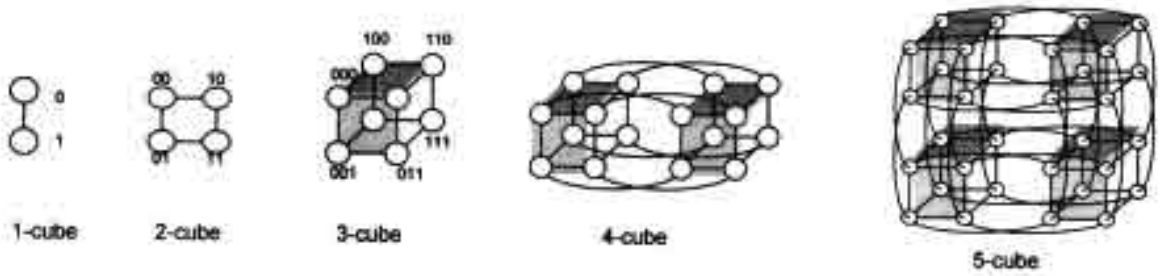


Figure 9: Examples of some hypercube-based systems.

2.3.2.1 The Gray Code Strategy

The GRAY CODE strategy [6] was proposed in 1987 for a k-cube (or hypercube Q_k) system (of size $N = 2^k$) to improve sub-cube recognition ability over the original BUDDY strategy by using a single GRAY CODE (twice sub-cube recognition) or multiple GRAY CODEs (almost complete sub-cube recognition). In the GRAY CODE allocation for an incoming task (q-cube), the least integer m is determined such that all $(i \bmod 2^k)^{th}$ allocation bit are 0's, where $i \in \#[m2^{q-1}, (m+2)2^{q-1}-1]$. The allocation time complexity of a single GRAY CODE is $O(2^{k-q+1})$ or O(N) and the deallocation time complexity (to combine sub-cubes in lower dimension(s)) is O(N).

2.3.2.2 The Extended Buddy Strategy

The Extended BUDDY strategy [1], an efficient sub-cube recognition, was proposed in 1992 to recognize more sub-cubes at different levels of the buddy tree. Two q-sub-cubes are buddies of each other if the most significant (k-q-1) bits are the same, the $(q+1)^{th}$ bit is different, and the least significant q bits are DON'T CARE (*), where $q = 0, 1, 2, \ldots, k$ and $N = 2^k$. For an incoming task (q-cube), a 2-level extended BUDDY search could recognize the same sub-cubes as the GRAY CODE strategy, and hence the all-level extended BUDDY search (from level k-q to k) performed complete sub-cubes recognition. For an incoming request q-cube, it searched from level k-q to k and at every search level, the algorithm recognized sub-cubes at that level only (to avoid check many unavailable sub-cubes).

2.3.2.3 The multi-Queue Buddy Scheduling Strategy

The multi-queue BUDDY scheduling [34] was introduced in 1995 to improve system fragmentation of the original BUDDY strategy. As the multi-queue scheduling was provided for the existing BUDDY allocation, the time complexity of this allocation algorithm was O(N). This method provided better system performance (i.e., less delay time, higher system utilization) than that of the BUDDY strategy. Advantages of multi-queue scheduling are: 1) a job requiring a small sub-cube is not blocked behind large jobs because the scheduler maintains k separate queues for each dimension of a k-Cube system ($N = 2^k$) and 2) a job waits for an occupied sub-cube of the same size instead of decomposing a new larger sub-cube.

2.3.2.4 The Fast Buddy Tree Strategy

The fast BUDDY strategy [9] was proposed in 1990 a nd 1992 to perform a fast (bit-map) complete recognition by using a collapsing binary tree to represent a hypercube system whose distant nodes (buddies) were brought close to each other for fast searching. This strategy could recognize complete sub-cubes as the (bit-map) multi GRAY CODEs in less time complexity, which is only $O(k^2 2^{k-q})$ or $O(N\log^2 N)$. In this fast BUDDY strategy, a collapsing transform (C-Transform) was a transformation, which operated on a binary tree representation of H_k and produced a collapsed tree. The C-transform at level i (ξ_i) involved collapsing T_k such that the left and right sub-trees of every node at level i were clustered together without changing their relative locations and then swapping the incoming links of the two inner nodes in every block of four nodes at level i+2 throughout level k.

2.3.2.5 The Free List Strategy

The best fit FREE LIST strategy [24] [25], a non-bit-map approach, was proposed in 1989 and 1991 in order to "improve searching time and recognition ability" over the BUDDY and GRAY CODE strategies. In the FREE LIST strategy, an array of (k+1) free lists data structure was used to maintain available sub-cube(s) for each dimension i, where i = 0, 1, 2, ..., k (of a system size $N = 2^k$). For an incoming request of q-cube, the first free sub-cube of the q^{th} dimension will be allocated, if the list[q] is not empty; otherwise a free sub-cube of dimension greater than q is decomposed. The FREE LIST processor allocation time complexity was only $O(\log_2 N)$; however, the combining complexity in the FREE LIST deallocation was $O(N \log_2 N)$. The FREE LIST allocation performance was statically optimal, similar to the bit-map approach (i.e., BUDDY, GRAY CODE, etc.). For dynamic system performance, it was better in sub-cube recognition ability, and hence improved the delay time by around 15% than those previous strategies.

2.3.2.6 The Maximum Set of Sub-Cube Strategy

The maximum set of SUB-CUBEs strategy [19] [20], another best fit (non-bit-map) FREE LIST approach, was proposed in 1988 and 1991 by using a max set of sub-cubes in order to maintain the greatest maximum set of sub-cubes after every allocation and deallocation of any sub-cube. In this approach, a set of disjoint free sub-cubes was called the max set of SUB-CUBEs (MSS) if it was equal to (or greater than) all other sets of the same free sub-cubes. However, the optimal MSS was an NP-hard problem; the "approximate MSS" was proposed in $O(N \log_2 N)$, by using a consensus graph (based on the switching theory) for quickly forming a new set of prime cubes from the current one, which was required in every recursion of the optimal MSS. Unlike the FREE LIST strategy, the combining process of this approach was done during both the allocation and deallocation processes, whereas in the FREE LIST strategy, the combining process was done at the deallocation process only.

2.3.2.7 The Dynamic Binary Tree with Free List Strategy

The dynamic BINARY TREE with FREE LIST strategy [11], another best fit approach, was proposed in 1992. It used two arrays of free lists, called 'real' and 'ext', for fast processor allocation. The 'real' list stored sub-cubes, which were the combination of sibling nodes only, whereas the 'ext' list stored sub-cubes, which were the combination of two cousin nodes. All free sub-cubes that belonged to either the 'real' list (with the higher priority) or the 'ext' list were mutually disjoint. The processor allocation/deallocation time complexity of this approach was only $O(\log_2 N)$ and the dynamic BINARY TREE & FREE LIST system performance was comparable to that of the FREE LIST strategy.

2.3.2.8 The Modified Prime Cube Graph Strategy

The modified PRIME CUBE GRAPH strategy [48] [49] was proposed in 1991 and 1993 for sub-cube and non-cubic processor allocations by using a prime cube (PC) graph. A free sub-cube was prime if there was no other free (larger) sub-cubes covering it. The objective of this strategy was to minimize processor fragmentation by keeping the redundant intersection information. In this approach, the strategy tried to avoid allocating an articulation point in the PC-graph to maintain a higher connectivity of the PC-graph. This reduced the external fragmentation, especially for non-cubic allocation in $O(L^2 \log_2 N)$ time, where L

is the number of prime-cube intersection and the number of prime-cube non-intersection. This approach improved system fragmentation by 25-50% over the FREE LIST strategy.

2.3.2.9 The Least Overlap Free List Strategy

The least overlap FREE LIST strategy [37], another best fit approach, was proposed in 1995. In this strategy, sub-cubes are allocated to requested tasks (q-cubes) in order to "minimize fragmentation" of the free sub-cubes such that the largest possible number of free sub-cubes remain after the allocation. The least overlap FREE LIST allocation time complexity was $O(\log N\ 3^{2k})$ and the deallocation procedure (to combine a released sub-cube into the free (sub-cube) list to produce a list of all maximal free cubes) was $O(\log^2 N\ 3^{2k})$, where $k = \log_2 N$. The least overlap FREE LIST strategy yielded larger available sub-cubes (in higher dimensions) than those of the maximum set of SUB-CUBE and the GRAY CODE strategies.

2.3.3 PE Allocation in Other Interconnection Networks

2.3.3.1 The Product Network-Partitioning Strategy

In 1990, product networks (see Figure 10) were proposed as a unified theory, called "theory of Cartesian product networks" [51], which included common topological analysis (i.e., degree, diameter, and average distance), routing algorithms ([21]), embedding and network partitioning strategies. As a part in that study, the PRODUCT-NETWORK partitioning was a process of dividing a network into sub-networks (or partitions) of various sizes for incoming tasks. In the PREDUCT-NETWORK partitioning allocation, the "network manager" determines the smallest partition size that can be allocated to the request if it was free. For a larger free partition, it is recursively partitioned until a partition of the appropriate size is obtained. If there is no free partition, then the request is queued according to a certain scheduling policy. When a partition is released, it is added into the pool (i.e., a FREE LIST structure) of free partitions and then a merging process is applied (if it is possible) to form larger free partitions to minimize fragmentation. In this approach, the generalized idea was introduced for the network partitioning and sub-system allocation at the network level and for a specific interconnection network (i.e., hypercubes, 2-D meshes), this approach suggested to apply any efficient existing strategy.

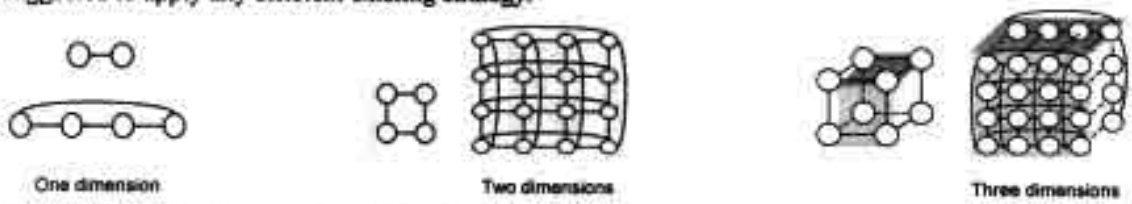


Figure 10: Examples of some product-network-based systems.

2.3.3.2 The Bit-Map Hypercycle-Based Strategy

In 1995, first fit BIT-MAP hypercycle-based processor allocation [17] was proposed for hypercycles [16], a sub-class of product networks (see Figure 11), which include hypercubes (binary k-cubes), multi-dimensional toruses, etc. In this approach, all processors in the hypercycle system of size ($N = m_k \times m_{k-1} \times m_1$) were assigned unique numbers from 0 to N-1. The first fit BIT-MAP hypercycle-based allocation strategy utilized a list of allocation bits (numbered from 0 to N-1), which was set to 1 if it was allocated; otherwise it was set to 0 (or free). The time complexity of a processor allocation was O(N) (or O(PN) if P permutations (or several lists) were used). Note that this approach was similar to the bit-map GRAY CODE when applied to the hypercube networks.

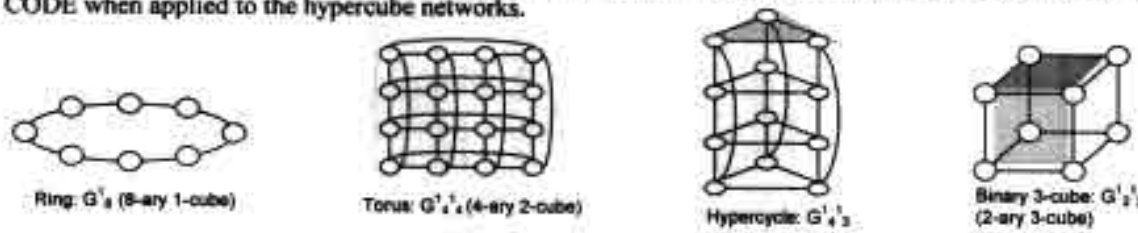


Figure 11: Examples of some hypercycle-based systems.

3. UNIVERSAL CU/PE ALLOCATION FOR RECONFIGURABLE MSIMD/MIMD SYSTEMS

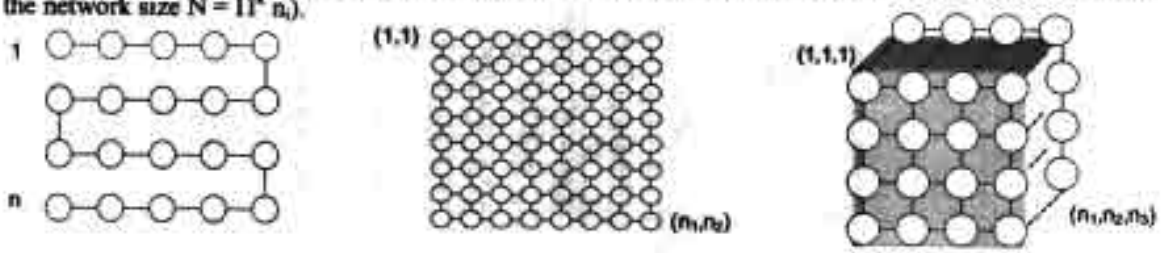
In this study, "the universal CU/PE allocation model" is designed to perform resource (CU/PE) allocation/ deallocation decision for the reconfigurable and partitionable MSIMD/MIMD parallel architectures. Interconnection networks that cover in our model are all networks in the product network class including multi-dimensional meshes, multi-dimensional tori, hypercubes, n-ary k-cubes, etc. In this universal resource (CU/PE) allocation model, we use the tree data structure to represent system states of the reconfigurable and partitionable parallel systems. In addition, we present not only a universal model but also a generalized method for all networks in the product network class. Term "generalized" implies that for each particular interconnection network that belong to the product networks (such as 2-D meshes), we also uses the same tree structure and the same allocation/deallocation methodology, as applied at any k-D network level (such as k-D meshes). Next in our universal tree-based model, we present the tree system states representation (in Section 3.1) and a number of generalized algorithms such as the network partitioning (in Section 3.2), the sub-system combining (in Section 3.3), the best-fit heuristic for sub-system (PE) a nd c ontrol unit (CU) a llocation de cision (in S ection 3. 4), a nd t he r esource (CU/PE) allocation/ deallocation decision (in Section 3.5) as well as time complexity analysis (in Section 3.6).

3.1 System States Representation

3.1.1 Product Networks and Parallel Systems

DEFINITION 1: A Cartesian product of k basic networks is defined in [51] as $G(V, E) = G_1 \times G_2 \times \ldots \times G_k$, where $V = \{\alpha = (a_1, a_2, \ldots, a_k)$ be an address of any PE (processing element) in $G \mid a_1 \in V_1, a_2 \in V_2, \ldots, a_k \in V_k\}$ and $E = \{\langle \alpha, \beta \rangle$ be a link between any two PEs in G, $\alpha = (a_1, a_2, \ldots, a_k)$ and $\beta = (b_1, b_2, \ldots, b_k) \mid$ there exists an i such that $\langle a_i, b_i \rangle \in E_i$ (or $|a_i - b_i| = 1$) and for $\forall j \neq i$, $a_j = b_j\}$. The size of the product network is represented as $N = n_1 \times n_2 \times \ldots \times n_k$, where $n_i = |V_i|$ or size of each sub-network G_i, $i = 1, 2, \ldots, k$.

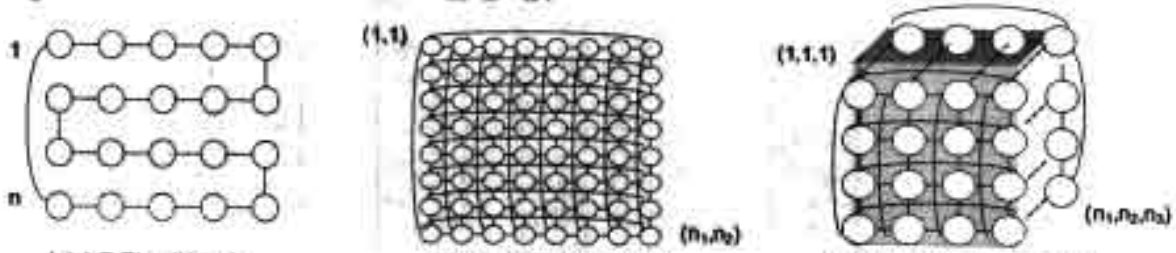
For examples, a k-dimensional (k-D) mesh parallel system (see Figure 12) is a product network of k linear arrays $G_i(V_i, E_i)$ of n_i nodes, $i = 1, 2, \ldots, k$ (where $V_i = \{1, 2, \ldots, n_i\}$, $E_i = \{\langle j, j+1 \rangle \mid j = 1, 2, \ldots, n_i - 1\}$, and the network size $N = \Pi^k n_i$).



(a) 1-D Linear Array (N = n) (b) 2-D Mesh ($N = n_1 x n_2$) (c) 3-D Mesh ($N = n_1 x n_2 x n_3$)

Figure 12: Some examples of the k-D meshes: a) a linear array (k=1); b) a 2-D mesh (k=2); and c) a 3-D mesh (k=3).

As another example, a k-dimensional (k-D) torus parallel system (see Figure 13) is a product network of k rings $G_i(V_i, E_i)$ of n_i nodes, $i = 1, 2, \ldots, k$ (where $V_i = \{1, 2, \ldots, n_i\}$, $E_i = \{\langle j, j+1 \bmod n_i \rangle \mid j = 1, 2, \ldots, n_i\}$, and the network size $N = \Pi^k n_i$). Note: an n-ary k-cube is a special case of the k-D torus, where each G_i is a ring of n nodes and the network size $N = \Pi^k n = n^k$.



(a) 1-D Ring (N = n) (b) 2-D Torus ($N = n_1 x n_2$) (c) 3-D Torus ($N = n_1 x n_2 x n_3$)

Figure 13: Some examples of the k-D tori: a) a ring (k=1); b) a 2-D torus (k=2); and c) a 3-D torus (k=3).

As another example, hypercube (or binary k-cube) parallel system (see Figure 14) is a product network of k linear arrays (or rings) $G_i(V_i, E_i)$ of exactly 2 nodes, i = 1, 2, ..., k, (where $V_i = \{0, 1\}$, $E_i = \{ <0, 1> \}$, and the network size $N = \Pi^k 2 = 2^k$).

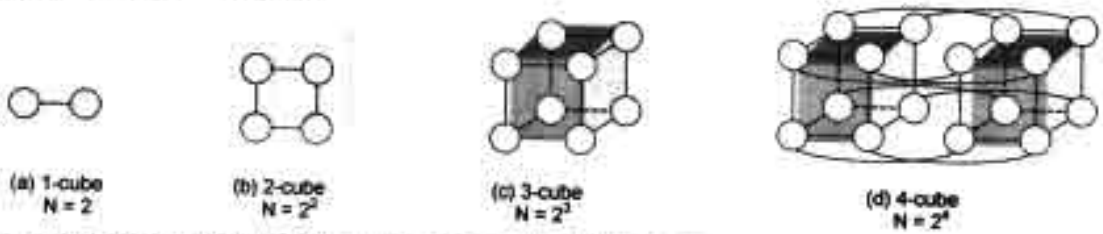


Figure 14: Some examples of the Hypercubes: a) a 1-cube (k=1); b) a 2-cube (k=2); c) a 3-cube (k=3); and d) a 4-cube (k=4).

Note: it is assumed without loss generality that each sub-network $G_i(V_i, E_i)$ in the product network refers to a basic network (i.e., a linear array, a ring, etc.), where a set of vertices $V_i = \{1, 2, \ldots, |V_i|\}$ and a set of edges $E_i = \{e = <x, y> \mid e \neq \emptyset;\ x, y \in V_i\}$, i = 1, 2, ..., k. However, the more complicated product networks can be represented by using parenthesis(s), where the inner most parenthesis will be constructed first. For example, $G = (G_1 \times G_2) \times G_3$ is a product network of two networks: a product network ($G_1 \times G_2$) and a basic network G_3. Finally, it is assumed that the product network $G_1 \times G_2 \times \ldots \times G_k$ is constructed or reorganized such that G_1 provides the maximum number of links.

3.1.2 Tree System States Representation

The tree data structure is used to represent system states or to store allocation information of the partitionable MSIMD/MIMD parallel system. Such a parallel system is constructed with any k-product network configuration, where k is a number of basic networks (i.e., linear arrays, rings, etc.) in the product networks (PN) class. In our study, a "system" or a "host network" refers to a given partitionable parallel system with a k-product network ($G_1 \times G_2 \times \ldots \times G_k$) of size $N = n_1 \times n_2 \times \ldots \times n_k$ at a system address, represented by using two k-coordinates $<(1, 1, \ldots, 1), (n_1, n_2, \ldots, n_k)>$, the base- and cover-addresses. For an incoming task, a "sub-system" or a "sub-network" at the address $<(a_1, a_2, \ldots, a_k), (b_1, b_2, \ldots, b_k)>$ can be partitioned to have either

1) a smaller dimensions or network degrees (see Figure 15a) than those of the k-product network such as $G_1 \times G_2 \times \ldots \times G_j$ of size $N = n_1 \times n_2 \times \ldots \times n_j$, where $j < k$ or
2) a smaller system size with the same network degrees $N' = n_1' \times n_2' \times \ldots \times n_k'$ (see Figure 15b) than those of the k-product network, where $n_i' \leq n_i$ and i = 1, 2, ..., k or
3) a combined partitioning of network degree and size (see Figure 15c), which consists of two sub-systems (i.e., $N' = n_1 \times n_2 \times \ldots \times p$ and $N'' = n_1 \times n_2 \times \ldots \times n_{k-1} \times n_k - p$), where $p \geq 1$. If p = 1, then the first sub-system is reduced the network degree from k to k – 1, whereas the network degree of another sub-system is still the same (k).

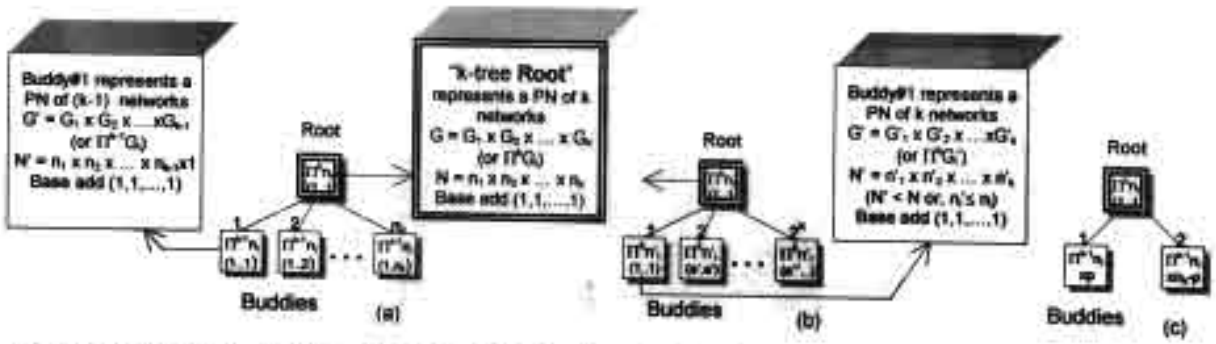


Figure 15: Various trees system state representation: a) after partitioning by network degree (that reduces the number of PNs but sizes for dimension 1 to k-1 are the same); b) after partitioning by network size (that reduces sizes for all k dimensions but the number of PNs are the same); and c) after partitioning by network degree and size.

In our universal tree-based model, the number of nodes in the tree becomes dynamic, corresponding to the number of tasks allocated, and hence our model is classified as a non-processor-bit-map approach. At beginning, the tree consists of only one node, called the "root". That root node is used to store the system information (such as the system's size, base-address, status, type, etc.). During execution time when many tasks are allocated, each leaf node in the tree, representing a sub-system, may be available (free) for incoming task(s) or unavailable (busy) for executing task(s) and each internal node is partially available, which are represented by using status = 0, 1, or x, respectively. In order to allocate an incoming task, each larger free node can be partitioned into a number of children/node, called "buddies". The number of buddies of a system network is derived by using partitioning criteria (described in detail later in Section 3.2), according to each incoming task which request a specified k'-product network and a particular size:

1) Partitioning by network degree (see Figure 15a) creates the number of buddies = n_k since the size of the k^{th} basic network (n_k) is decomposed to have one processor (see another example in Figure 16),
2) Partitioning by network size (see Figure 15b) creates the number of buddies = 2^k since all basic network sizes are partitioned (see also an example in Figure 17), and
3) Partitioning by network degree and size (see Figure 15c) creates the number of buddies = 2 since the k^{th} basic network is partitioned only once to have p and $n_k - p$ processors (see Figure 18), where $p \geq 1$.

Figure 16 illustrates the particular example of the binary tree and the corresponding system status that stores 3 incoming tasks (3-, 2-, and 4-cubes) with the MIMD mode on a given partitionable 5-Cube system.

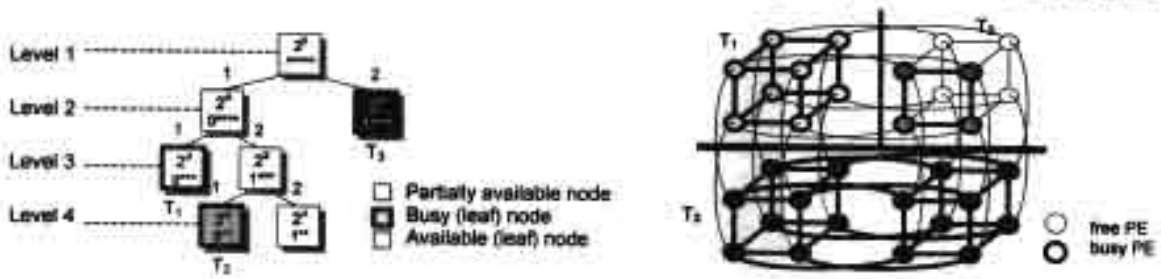


Figure 16: The binary tree system state representation for a 5-Cube system with 3 MIMD tasks allocated.

Figure 17 illustrates another specific example of the quad tree and corresponding system status that stores three incoming tasks (4x4, 2x3, and 2x4) with the MIMD mode on a given 8x10 Mesh-connected system.

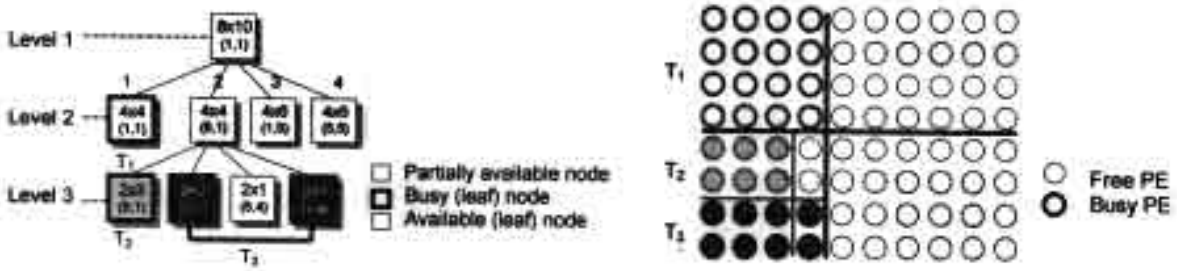


Figure 17: The quad tree system state representation for a 2-D (8x10) Mesh system with 3 MIMD tasks allocated.

Figure 18 illustrates the specific example of the binary tree and corresponding system status that stores an incoming tasks (8x8) with the MIMD mode on a given 8x8x8 Mesh-connected system.

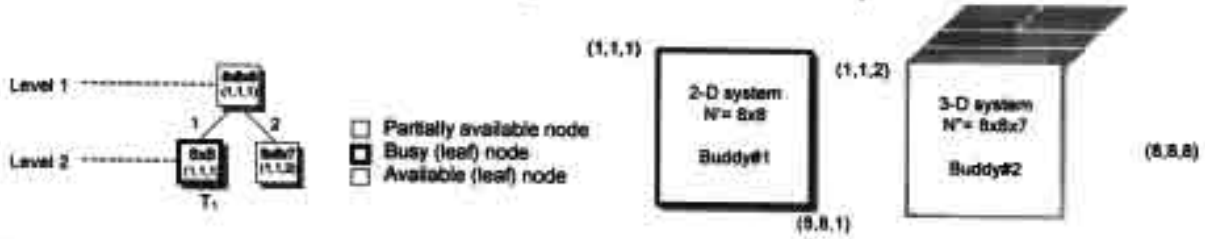


Figure 18: The binary tree system state representation for a 3-D (8x8x8) Mesh system with one MIMD task allocated.

For the reconfigurable and partitionable MSIMD/MIMD parallel system, in order to distinguish between MIMD and SIMD partitions in the allocation process, we use two simple examples (see Figure 19 and 20) for the partitionable 2-D meshes. Figure 19 illustrates the quad tree structure with some information about five MIMD tasks (4x4, 2x3, 3x4, 2x2, and 1x5) allocated into an 8x10 system.

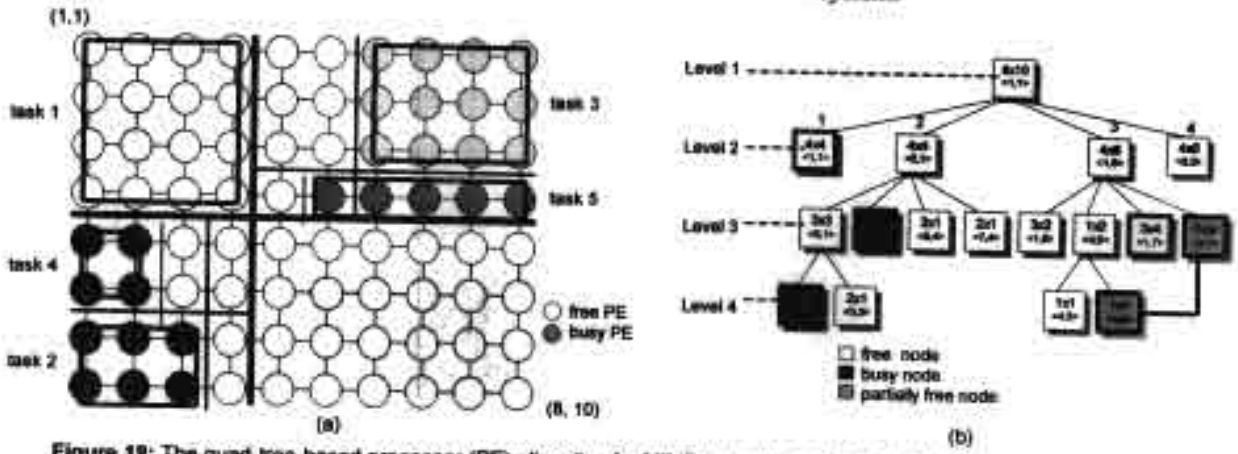


Figure 19: The quad-tree-based processor (PE) allocation for MIMD tasks (4x4, 2x3, 3x4, 2x2, and 1x5) on an 8x10 Mesh system: a) the allocated system status and b) the corresponding quad tree structure.

Figure 20 illustrates the system status of applying the modified quad tree structure with CU allocation for allocating five SIMD tasks (2x3, 2x2, 1x5, 4x4, and 3x6) on an 8x10 mesh system.

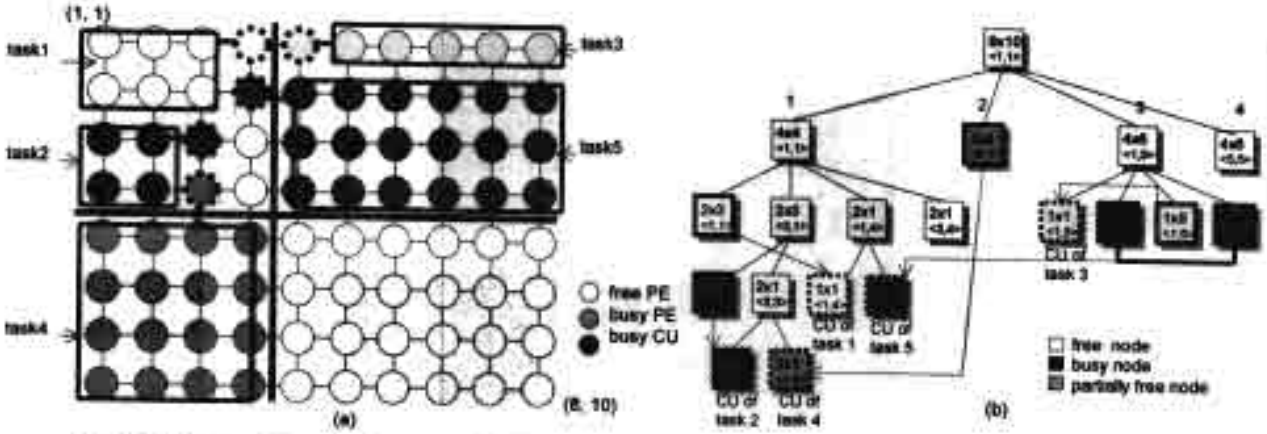


Figure 20: The quad-tree-based resource (CU/PE) allocation for SIMD tasks (2x3, 2x2, 1x5, 4x4, and 3x6) on an 8x10 mesh system: a) the allocated system status and b) the corresponding quad tree structure.

Figure 21 shows the tree node information and the node degree for any tree node that represents an MIMD task and a special node structure that represents an MSIMD/MIMD task, which includes an additional link to the corresponding CU for SIMD tasks.

Figure 21: The tree node structure: node's information and node's degree for a) any MIMD task and b) any SIMD task.

In our universal tree-based model, the generalized methodology for resource (CU/PE) allocation/ deallocation and task scheduling consists of four main procedures:

(1) Network partitioning procedure (Section 3.2): this function is used to partition a system network into sub-systems for the smaller requested task,
(2) Sub-system combining procedure (Section 3.3): this function is used to recombine a number corresponding free sub-partitions into a larger free sub-system,
(3) Allocation/deallocation searching procedure (Section 3.5): the allocation function is defined to find a free sub-system for each incoming task, applied in allocation process by using the best-fit heuristic criteria (Section 3.4). The deallocation function is defined to find the location of a finished task in deallocation process, and
(4) Task scheduling procedure: this function is used to perform scheduling for all tasks in the waiting queue (i.e., a priority FCFS scheduling).

The diagram in Figure 22 illustrates the dynamic tree-based sub-system allocation/deallocation for a reconfigurable and partitionable MSIMD/MIMD parallel system.

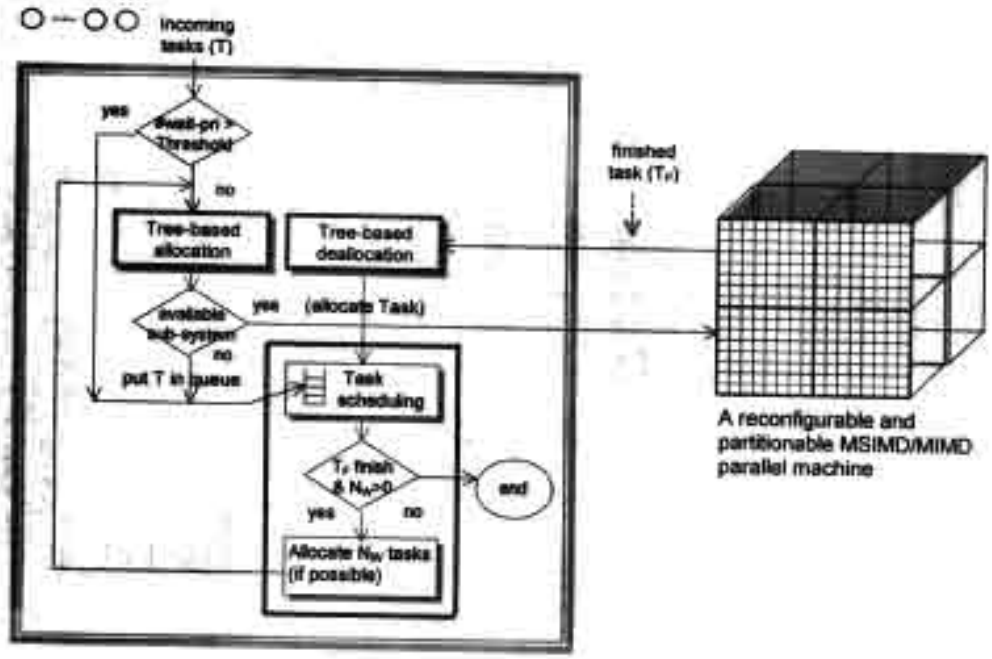


Figure 22: A Processor allocation/deallocation and task scheduling diagram.

During execution time, when there is an incoming task if the wait priority of the first task in the waiting queue is more than the threshold value, that task will be put in the waiting queue. Otherwise, the processor "allocation" procedure will find an appropriate free sub-system by searching into the tree. If there exists an available sub-system, the requested task will be allocated on the system. If no available sub-system, the request will be put in the waiting queue with priority FCFS scheduling.

When a task is finished, the processor "deallocation" procedure will find the allocated position of that finished task by using sub-set path searching into the tree. After finishing free (or deallocate) the node that stores information, the recombining process is applied to combine all corresponding (free) sub-partitions (or Buddy nodes) as soon as they become available. The recombining process starts form the new free node (of the completed task) to the root of the tree. This process will stop when there is at least one buddy of a corresponding node (along the combining path) is not available. At this time, if there are task(s) in the waiting queue, the priority FCFS scheduling will be applied to perform scheduling and allocation for these waiting tasks.

3.2 Network Partitioning

In our universal tree-based model, the network partitioning is applied when an incoming task requires a smaller sub-system than the larger available node. To allocate the fit sub-system to the task, the selected tree node is dynamically created and partitioned into a number of children/node, called "buddies". The number of buddies of any tree node are derived by using one of the following methods, namely 1) the partitioning by network degree, 2) the partitioning by network size, and 3) the partitioning by network degree and size. All partitioning methods except the last one are applied from our previous study [41].

3.2.1 Partitioning by Network Degree

This network partitioning is performed by partitioning the last basic network (G_k) that was included in the product network (or partitioning along the k^{th} dimension). Hence, this partitioning method reduces the number of basic networks from k to k-1 per partitioning process.

Let a requested network of an incoming task be a j-product network, where $j < k$. If $j = k-1$, then only one partitioning step is applied to the last (k^{th}) dimension of the system. Therefore, the number of buddies/node is equal to the size of the G_k network ($|V_k| = n_k$). Each buddy (or sub-network), whose ID = 1, 2, ..., n_k, represents a (k-1)-product network $G_1xG_2x...xG_{k-1}$. The size of each buddy ID is $N_{ID} = n_1 \times n_2 \times ... \times n_{k-1}$ (x 1). The base address of each buddy is $\alpha_{ID} = (a_1, a_2, ..., a_{k-1}, ID)$ and its last cover address is $\beta_{ID} = (b_1, b_2, ..., b_{k-1}, ID)$, where $b_i = a_i + n_i - 1$ and $i = 1, 2, ..., k$. Time complexity of each partitioning process is $O(k n_k)$ since there are n_k buddies and each buddy's information is computed for all k basic networks. Hence the time complexity for the recursive partitioning in L steps (or levels) is $O(k^2n)$ which includes identifying the number of buddies, computing buddies' base-addresses and sizes, where $L \leq k$ and $n = \max(n_i)$.

Figure 23 illustrates an example of the partitioning by network degree on a 3-D mesh (of size 64 x 64 x 64) in to sixty-four 2-D meshes (of size 64x64 (x1)).

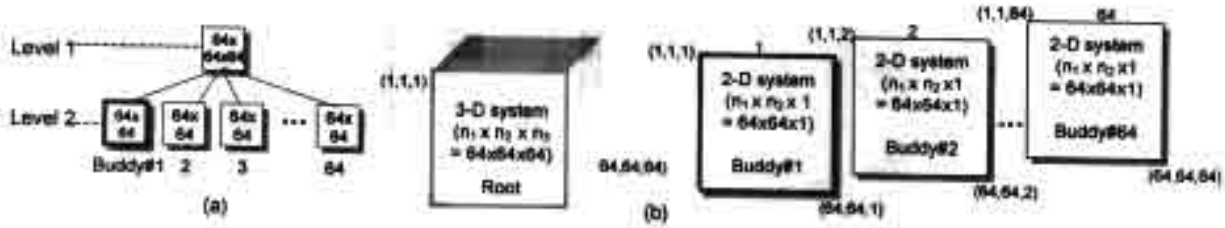


Figure 23: An example of the partitioning by network degree on a 3-D System (N = 64 x 64 x 64): a) the tree structure and b) the system and sub-systems after partitioning from a 3-D system to 64 of 2-D systems (N = 64 x 64).

Usually, this partitioning method is applied for the system with the large k (dimensions) and the small n (processors/dimension). Figure 24 depicts another example of the partitioning by network degree on a hypercube or 5-cube (k = 5 and n = 2).

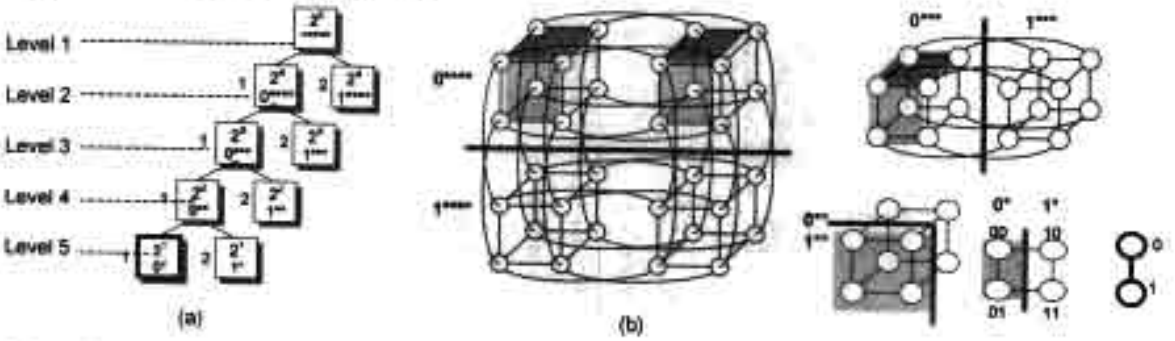


Figure 24: An example of the partitioning by network degree on a hypercube (5-cube) system: a) the tree structure and b) the system and sub-systems after partitioning from a 5-cube to the smallest 1-sub-cubes.

3.2.2 Partitioning by Network Size

This network partitioning is the partitioning process that partitions all k dimensions of the k-D system ($N = n_1 \times n_2 \times \ldots \times n_k$) into smaller sub-systems and allocates one for the request $p_1 \times p_2 \times \ldots \times p_k$, where $p_i \le n_i$, $i = 1, 2, \ldots, k$. This partitioning method reduces the network sizes but the number of basic networks (k) are still the same. The relationship of the corresponding buddy's ID, base-address, size, and status are performed by using "sub-system bit-map", defined as follows:

Let a requested network is defined as $G' = G_1' \times G_2' \times \ldots \times G_k'$(of size $p_1 \times p_2 \times \ldots \times p_k$). Therefore, the number of buddies/node in each partitioning is equal to 2^k since each of all k dimensions are partitioned into two sizes, which are not necessary equal. Then buddies' assigned IDs are 1, 2, 3, ..., 2^k that correspond to the k-bit-map $b_{k-1} \ldots b_2 b_1 b_0$, where $b_i = 0$ or 1, $i = 1, 2, \ldots, k$. For example if $k = 2$, then $2^k = 4$ and ID = 1, 2, 3, 4 (or 00, 01, 10, 11), respectively. Next, we design the "Buddy-ID-Address-Size conversion" algorithm (see all steps in Figure 25). This algorithm provides efficient time complexity for consequent algorithms in the model such as the network partitioning, the sub-system combining (Section 3.3), and the best fit heuristic for allocation decision (Section 3.4). This network partitioning process (i.e., identifying #buddies = 2^k, their base-addresses and sizes for all k dimensions) is computed in $O(k2^k)$ time. Note that this partitioning method is usually applied when k is small.

Let R be a considering root node (of size $N = n_1 \times n_2 \times \ldots \times n_k$).
B be a system bit-map of each Buddy ID ($B = b_{k-1} \ldots b_1 b_0$).
J be a requested task or job (of size = $p_1 \times p_2 \times \ldots \times p_k$).
$\alpha = (a_1, a_2, \ldots, a_k)$ be a base address of R and also the base address of the Buddy#1 of R.
$S = p_1 \times p_2 \times \ldots \times p_k$ be a requested task's size and also the size of the Buddy#1 of R.

Conversion Method
1. Convert a set of all buddy-IDs (1, 2, 3, ..., k) to their corresponding k-bit-IDs; [computed in $O(k2^k)$]
2. Compute a buddy's base-address = $(a_1', a_2', \ldots, a_k')$ and its size = $(n_1' \times n_2' \times \ldots \times n_k')$; [computed in $O(k2^k)$]
   for $j = 1, 2, \ldots, k$,
   $a_j' = a_j$; $n_j' = p_j$; if $b_{j-1} = 0$ (or the j^{th} bit (or dimension) is not partitioned.)
   $a_j' = a_j + p_j$; $n_j' = n_j - p_j$; if $b_{j-1} = 1$ (since that dimension is partitioned, corresponding to Buddy#1's size.)

Figure 25: The "Buddy-ID-Address-Size conversion" algorithm for the partitioning by network size.

Figure 26 illustrates two examples of the partitioning by network size on the 2-D and 3-D mesh systems. In Figure 26a, the 2-D mesh system (of size $N = n_1 \times n_2 = 64 \times 64$) is stored in the tree's root with a base address $\alpha = (a_1, a_2) = (1, 1)$. Suppose the first incoming task requests a sub-system of size 20x20. For this task, the root is partitioned into $2^k = 4$ buddies and the request will be allocated to the Buddy#1, where its base-address is $(a_1, a_2) = (1, 1)$ and its size is $(p_1 \times p_2) = (20 \times 20)$.

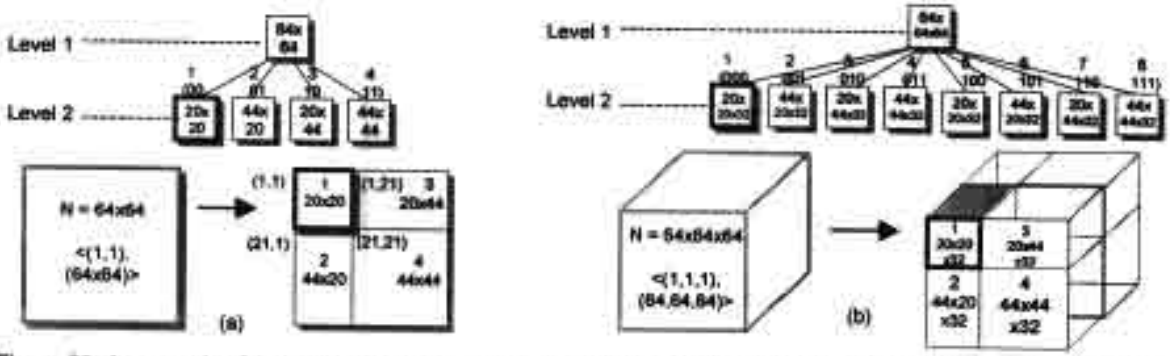


Figure 26: An example of the partitioning by network size: a) for the task (20x20) on the 2-D mesh system (N = 64 x 64); and b) for the task (20x20x32) on the 3-D mesh system (N = 64x64x64).

By applying the "Buddy-ID-Address-Size conversion" algorithm, first the set of 2-bit-map of each buddy i, $\{i / i = 1, 2, 3, 4\}$ is converted to $\{(b_1\ b_0) \mid b_i = 0 \text{ or } 1\} = \{00, 01, 10, 11\}$. Then, the set of corresponding base-addresses is $\{(a_1', a_2') \mid a_j' = a_j + (p_j * b_{j-1})\} = \{(1,1), (21,1), (1,21), (21,21)\}$ and the set of corresponding sizes is $\{(n_1', n_2')\} = \{(20x20), (44x20), (20x44), (44x44)\}$. In Figure 26b, the similar process is applied for the first incoming task (20x20x32) on the 3-D system (N = 64x64x64).

3.2.3 Partitioning by Network Degree and Size

The partitioning by network degree and size is introduced as a flexible partitioning method that combine the first two partitioning methods, which is either to decompose a network degree (k) or to partition a dimension's size (n_i). It is performed by partitioning the i^{th} dimension of the basic network G_i (of size n_i, $1 \le i \le k$) into 2 partitions, corresponding to p and n_i - p processors, where $p \ge 1$. This partitioning method always creates exactly 2 buddies per node, and hence it is called the binary tree. If i = k and p = 1, the network degree of the first partition is reduced from k to k-1 and the size of the k^{th} dimension of the second partition is reduced from n_k to n_k-1. In a special case, if i =k and n_i=2, the network degree of both partitions are reduced from k to k-1 and the size of the k^{th} dimension of both partitions are reduced from 2 to 1.

In general, suppose an incoming task needs a j-product network, where $j < k$. If j = k-1, the next level partitioning is applied to create 2 buddies/node. Each buddy (ID = 1, 2) represents a k-1 or k-product network $G_1 \times G_2 \times \ldots \times G'_k$ (of size $n_1 \times n_2 \times \ldots \times n_{k-1} \times p$) and another k-product network $G_1 \times G_2 \times \ldots \times G'_k$ (of size $n_1 \times n_2 \times \ldots \times n_{k-1} \times n_k$-p). We may need to perform k partitioning steps if all k dimensions of the requested task ($p_1 \times p_2 \times \ldots \times p_k$) are less than those of the system ($n_1 \times n_2 \times \ldots \times n_k$) at address ($a_1, a_2, \ldots, a_k$), where $p_i < n_i$, i =1, 2, ..., k. Time complexity of each partitioning process is O(k) since there are 2 buddies (ID = 1, 2) and their information is computed (i.e., base-addresses $a_j' = a_j$ or a_j+p_j and sizes $n_j' = p_j$ or n_j-p_j for buddy#1 or 2 of any dimension j (whereas other dimensions are the same as their root). For all k basic networks or dimensions, time complexity of the recursive partitioning is $O(k^2)$, which is improved over those ($O(k^2n)$ and $O(k2^k)$) of the first two partitioning methods. Therefore, this partitioning method is very useful for both high k and large n. Figure 27 illustrates an example of the partitioning by network degree and size on a 3-D mesh (N = 64x64x64) into a 2-D mesh (N' = 64x64) and a 3-D mesh (N'' = 64x64x63), compared to those (by the partitioning by network degree) in Figure 23.

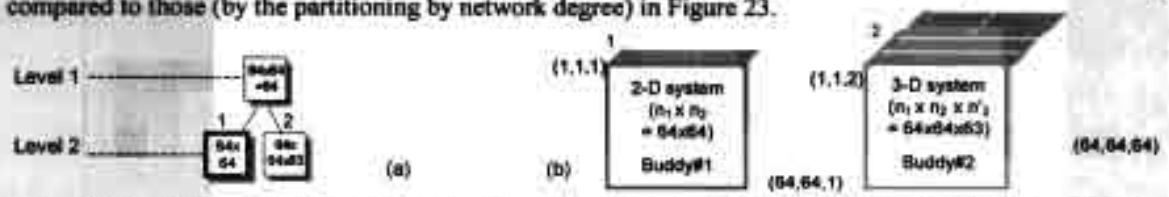


Figure 27: An example of the partitioning by network degree and size on a 3-D System (N = 64 x 64 x 64): a) the binary tree and b) the system after partitioning a 3-D system into 2 sub-systems.

Figure 28 illustrates another example of the partitioning by network degree and size on the 2-D and 3-D meshes, compared to those (by the partitioning by network size) in Figure 26. In Figure 28a for the 2-D mesh (N = 64 x 64), suppose the first incoming task requests a sub-system of size 20x20. For this task, the root, is partitioned into 2 buddies and then the Buddy#1 (64x20) is partitioned into 2 sub-buddies and the request will be allocated to the sub-Buddy#1 (20x20). For the 2-D system, we need two partitioning steps to reach the requested size and hence for the 3-D system, we need three partitioning steps to reach the fit partition (see Figure 28b). In general, for the k-D system, we have to perform k partitioning steps to partition all k dimensions.

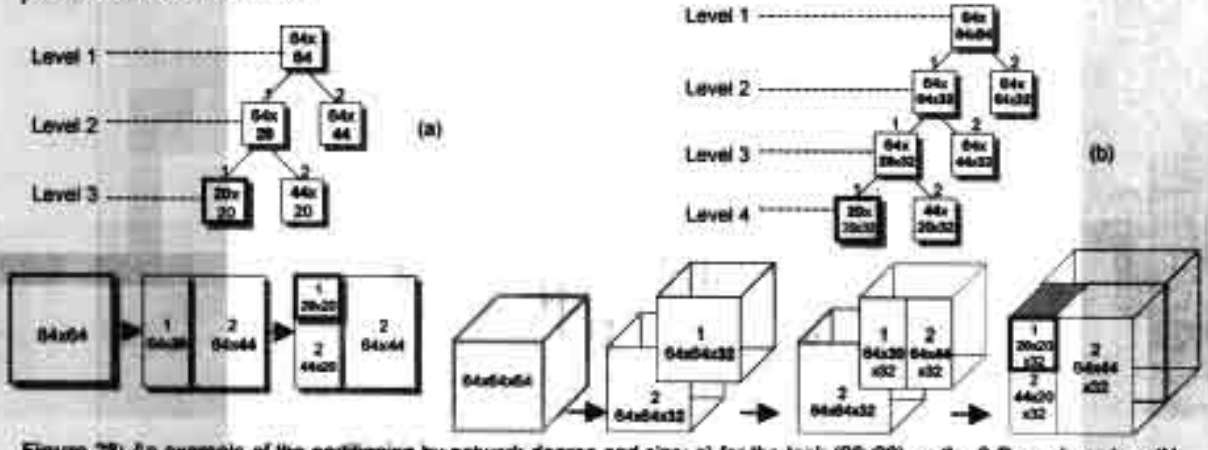


Figure 28: An example of the partitioning by network degree and size: a) for the task (20x20) on the 2-D mesh system (N = 64x64) and b) for the task (20x20x32) on the 3-D mesh system (N = 64x64x64).

Note that we apply each of three partitioning methods with different conditions and assumptions. First, the partitioning by network degree can be applied with an assumption that the degree of the task must less than or equal to that of the system and its size in each dimension should be the same as that of the system. For example, given a hypercube system of size $N = 2^{10}$. Only hypercube tasks with a requested degree $k \leq 10$ are allowed to execute on this system. Next, the partitioning by network size can be applied with an assumption that the degree of the task must be the same as that of the system and its size for each of dimension should be less than or equal to that of the system. For example, given a 2-D mesh system of size ($N = 10x10$). Only 2-D task allowed to execute on this system ($k = 2$) with a required size $n_1 \times n_2 \leq 10x10$.

Lastly, the partitioning by network degree and size is introduced in this study to combine those two partitioning methods in order to support the more flexible tasks and partitions. Usually, it is useful for the system with high k and large n (i.e., as $k = 10$ and $n = 64$) but incoming tasks require small k (i.e., $k = 2$ for 2-D meshes) or small n (i.e. $n = 2$ for hypercubes). For example, given a system of size $N = 10^{10}$ ($k = 10$ and $n = 10$) and a task requests a hypercube (or 9-cube) of size $N = 2^9$ ($k = 9$, $n = 2$). Here, we illustrate two possible ways to create the fit sub-partition of size 2^9 that are

Apply the partitioning by network degree first and then apply the partitioning by network size (Figure 29a): first we will create a tree with 10 buddies (each of size 10^9) by applying the partitioning by network degree in $O(k^2n)$ time, where $k = n = 10$. Next, the 1st buddy at level 2 is partitioned by applied the partitioning by network size to have 2^9 sub-buddies. The first sub-buddy's size is 2^9 and other sizes are $8x2x2^7$, $2x8x2^7$, $8x8x2^7$,..., $2x2x8^7$, $8x2x8^7$, $2x8x8^7$, 8^9, respectively. So we have $9 + 2^9$-1 free nodes (or fragments) and later we have to combine them. Time complexity of the partitioning is $O(k2^k)$ and that of the combining is $O(k^2 2^{2k})$, where $k = 9$.

Apply the partitioning by network degree and size (Figure 29b): we need 10 partitioning steps to reach the requested size 2^9. Hence we have only 10 fragments of size $10^9 \times 9$, $10^8 \times 8$, $10^7 \times 8 \times 2$, $10^6 \times 8 \times 2^2$, $10^5 \times 8 \times 2^3$, $10^4 \times 8 \times 2^4$, $10^3 \times 8 \times 2^5$, $10^2 \times 8 \times 2^6$, $10 \times 8 \times 2^7$, and 8×2^8. Then, time complexity is $O(k^2)$, where $k = 10$ and clearly it improves time complexity over the above method.

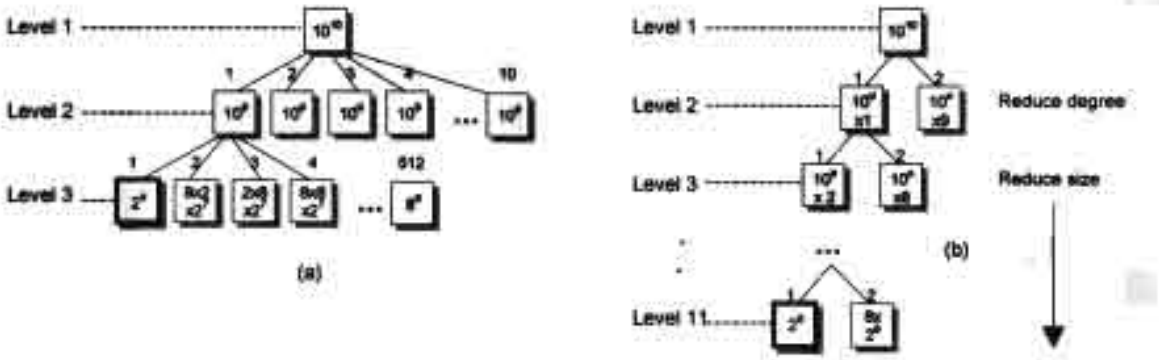


Figure 29: An example of applying 3 partitioning methods on a 10-D system ($N = 10^{10}$) for the task 2^9. a) apply the partitioning by network degree first and then apply the partitioning by network size and b) apply the partitioning by network degree and size.

In another practical example, given a 3-D system of size $N = 50x50x50$ ($k = 3$) and a task needs a 2-D mesh (10x10). Again we show two possible ways to create the fit sub-partition of size 10x10 that are

Apply the partitioning by network size: we create a tree with 2^3 buddies. The first buddy size is 10x10x1 for the request and others are 40x10x1, 10x40x1, 40x40x1, 10x10x49, 40x10x49, 10x40x49, and 40x40x49, respectively. Now we have 7 fragments and sometime later we need to combine them. Time complexity of the partitioning is $O(k2^k)$, where $k = 3$.

Apply the partitioning by network degree and size: we need 3 steps to get the requested size 10x10 and hence there are only 3 fragments of size 40x10x1, 50x40x1, and 50x50x49 respectively. Time complexity of the partitioning is $O(k^3)$, where $k = 3$. Therefore, it improves time complexity and it also provides the maximum free size (50x50x49) over to the partitioning by network size.

3.3 Sub-System Combining

In our universal tree-based model, the sub-system combining procedure is a process for any internal (partially free) node in the tree, which is used to recombine the smaller free nodes into the larger free sub-system. This combining method will be applied during allocation for the subsequent incoming tasks or deallocation processes for the finished tasks in order to maintain the maximum free node (described later in Section 3.5).

For the MSIMD/MIMD partitions, we apply one of the following methods namely: the combining by network degree, the combining by network size, and the combining by network degree and size. All combining methods except the last one are applied from our previous study [41].

3.3.1 Combining by Network Degree

This combining process is used to recombine some free nodes, corresponding to the partitioning by network degree. For this type of combining, two buddy nodes of the same root are adjacent if the different of their IDs is 1. In the combining by network degree, there are n_k buddies/node. Assume that some buddies (at any level L) are allocated for executing tasks and some buddies are free. Hence for a larger available sub-system, a number of combinations to combine other free buddies are identified.

Next, in order to simplify the complexity of the combining process, we classify this tree-based sub-system combining algorithms into three groups, which will be applied later in the allocation and deallocation procedures:

ALGORITHM CD.1 "All n_k-Buddy Combining": This procedure is used to combine all n_k free buddies (of size $n_1 \times n_2 \times \ldots \times n_{k-1} \times 1$) at level L into a larger k-Tree's node, at lower level L-1, whose base address $\alpha = (a_1, a_2, \ldots a_k)$ and size = $n_1 \times n_2 \times \ldots \times n_{k-1} \times n_k$. This combining process (i.e., identifying a sub-partition's status, according to #buddies) is computed in $O(n_k)$ time and it is applied after finishing the deallocation of a finished task in order to maintain the maximum size(s) of free node(s) in the tree as mush as possible.

ALGORITHM CD.2 "Some n-Buddy Combining": This combining procedure is used to combine a number (2, 3, ... , or n_k-1) of adjacent free buddies (each of size $n_1 \times n_2 \times \ldots \times n_{k-1} \times 1$) at level L+1 into a larger free sub-network (of size $n_1 \times n_2 \times \ldots \times n_{k-1} \times n$, where $1 < n < n_k$) in order to allocate for an incoming request that is larger than a buddy but equal to (or less than) the combined sub-network.

Let n be the number of combined buddies.

- For a non-wraparound network (i.e., a linear array, a mesh, etc.), there are (n_k-n+1) possible results of each combining size ($n_1 \times n_2 \times \ldots \times n_{k-1} \times n$) whose base address is at the first free buddy node of the combined sub-system.
- For a wraparound network (i.e., a ring, a torus, etc.), there are n_k possible results of each combining size ($n_1 \times n_2 \times \ldots \times n_{k-1} \times n$).

Time complexity of this type of combining is $O(n_k+k)$ for a specific n, where $1 < n < n_k$ and k required for computing the combined size and base address of the combined sub-system. This combining process can be applied during allocation or predefined after an allocation of an incoming task or deallocation of a finished task.

For instance, there are 2 possible results for combining (n_k-1) buddies such as {1, 2, 3, ..., n_k-1} and {2, 3, 4, ..., n_k}) of size $n_1 \times n_2 \times \ldots \times n_{k-1} \times n_k$-1 (of a non-wraparound network). And also, there are (n_k-1) possible results for combining 2 buddies (i.e., {1, 2}, {2, 3}, {3, 4}, ..., {n_{k-1}, n_k}) into a sub-system of size $n_1 \times n_2 \times \ldots \times n_{k-1} \times 2$, and so on (see Figure 30a).

In the practical example (Figure 30b), we can combine the last 63 free buddies from overall 64 buddies. This is a combined sub-system of two possible results, illustrated in Figure 30a.

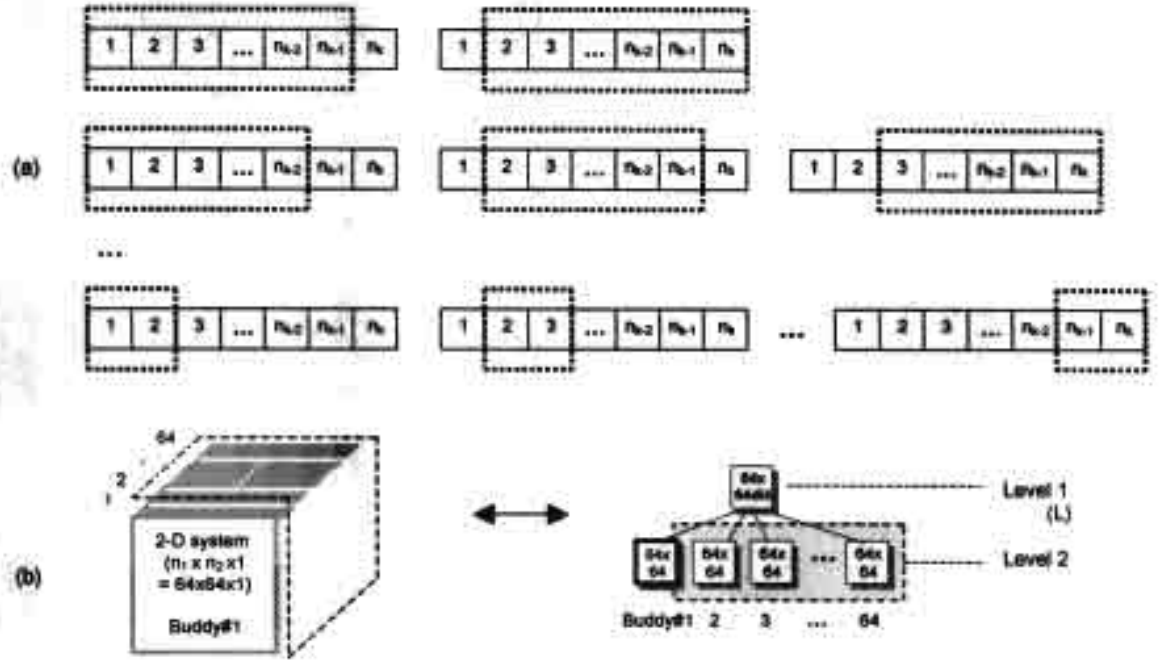


Figure 30: a) All possible combined results of some n-Buddy Combining and b) An example of identifying a combined sub-system form 63 buddies {2, 3, ..., 64} of size 64x64x63 on a 3-D system.

ALGORITHM CD.3 "Some Sub-buddies Combining": This combining procedure is used to combine some free sub-buddy nodes from other higher levels $L + i \le k+1$, where $1 < i < k$, to yield more sub-systems recognition than those obtained from the combining in Algorithm CD.2.

For more sub-system recognition, a number of adjacent free nodes in other dimensions ($1 \le i < k$) or at higher level (L+1, L+2, ..., k+1) can be combined into a larger sub-system "(k-1)-product network", corresponding to the top root at level L-1.

- At level L, each buddy node represents a (k-1)-product network (of size $n_1 \times n_2 x \ldots x n_{k-1} \times 1$) with partitioning along the k^{th} basic network (or dimension).
- Each of other (k-1)-product networks (or $n_i = 1$, $1 \le i < k$) can be recognized by combining free nodes at level $L+k-i \le k$. There are n_i possible combined sub-systems (whose combined sizes = $n_1 x \ldots n_{i-1} \times 1 \times n_{i+1} x \ldots x n_k$) by combining ($n_{i+1} \times n_{i+2} \times \ldots \times n_k$) free nodes (of size $n_1 \times n_2 \times \ldots \times n_{i-1}$), where $n_0=1$, and $i = k-1, k-2, \ldots, 1$, for combining at level L+1, L+2, ..., k+1.

Time complexity of this combining algorithm is $O(n_k)$ for each combined sub-system since it is combined from n_k nodes at level L+1.

- For combining n_k nodes at level L+1, time complexity is $O(n^2)$ for all (n_{k-1}) possible combined results, where $n = \max(n_{k-1}, n_k)$.
- For combining $n_{k-1} \times n_k$ nodes at level L+2, time complexity is $O(n^3)$ for all (n_{k-2}) possible combined results, where $n = \max(n_{k-2}, n_{k-1}, n_k)$.
- Finally, for combining $n_1 \times n_2 x \ldots x n_{k-1} \times n_k$ nodes at level k+1, time complexity is $O(kn^k)$ for all (n_1) possible combined results, where $n = \max(n_1, n_2, \ldots, n_k)$.

Therefore, the total time complexity for all possible combining results from all levels is $O(kn^{k+1})$ since $2n^2 + 3n^3 + \ldots + kn^k < kn^2 (1+n+n^2+\ldots+n^{k-2}) = kn^2(n^{k-1}-1)/(n-1) < kn^{k+1}$. Note: the base address of the combined sub-system is temporary stored at the first free buddy of that combined sub-system.

For instance, at level L+1 (see Figure 31a) there are n_{k-1} possible combined sub-systems (i.e., {1.1, 2.1, ..., n_k.1}, {1.2, 2.2, ..., n_k.2} ,..., {1.n_k, 2.n_k, ..., n_k.n_k}, where "i.j" represent buddy-ID i at level L and sub-buddy-ID j at level L+1) of size $n_1 \times n_2 \times \ldots \times n_{k-2} \times 1 \times n_k$ each of which consists of a number of n_k free

nodes (of size $n_1 \times \ldots \times n_{k-2} \times 1 \times 1$). See also two practical examples in Figure 31b (for a combing result at level L+1 on a 3-D mesh network) and Figure 32a (for a combining result at level L+1 on a hypercube network). Similarly at level L+2, there exists n_{k-2} possible combined sub-systems (of size $n_1 \times n_2 \times \ldots \times n_{k-3} \times 1 \times n_{k-1} \times n_k$) each of which consists of a number of $(n_{k-1} \times n_k)$ free nodes (of size $n_1 \times \ldots \times n_{k-3} \times 1 \times 1 \times 1$), etc. Figure 32b illustrates the practical example of the combining at L+2 on a hypercube network.

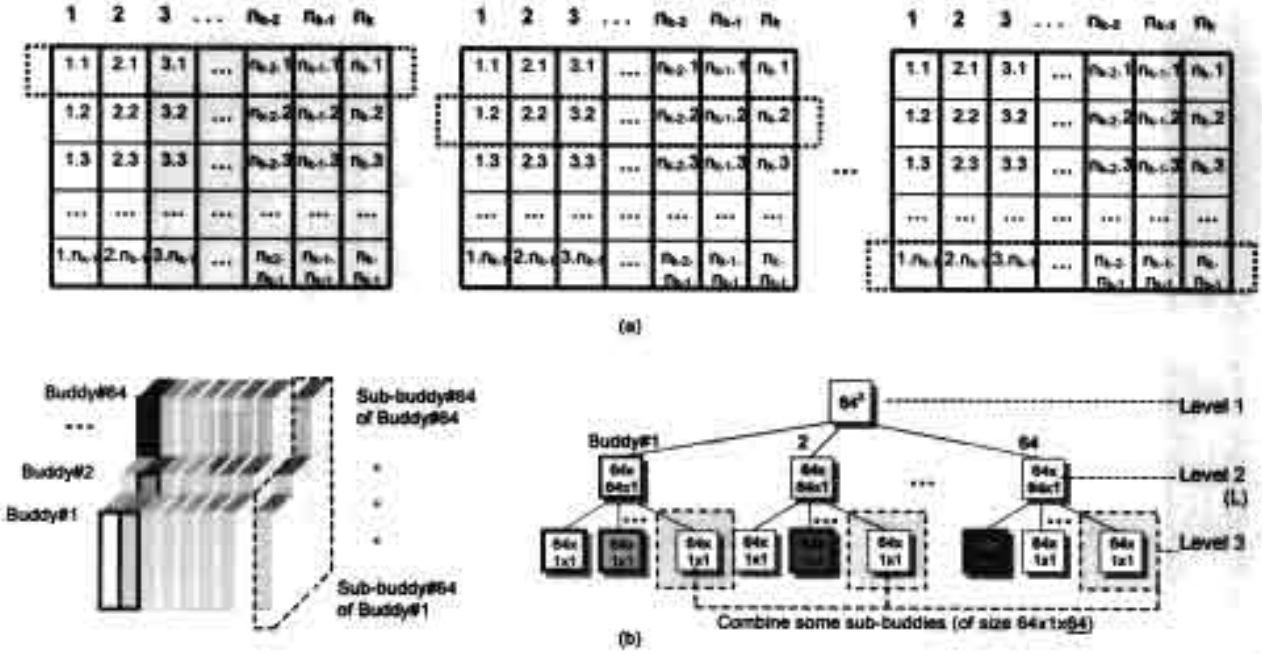


Figure 31: a) All possible combined results (of size $n_1 \times n_2 \times \ldots \times n_{k-2} \times 1 \times n_k$) at level L+1 and b) An example of some Sub-Buddy Combining on a 3-D system: identifying a combined sub-system form 64 sub-buddies {1.64, 2.64, 3.64, ..., 64.64) of size 64x1x64.

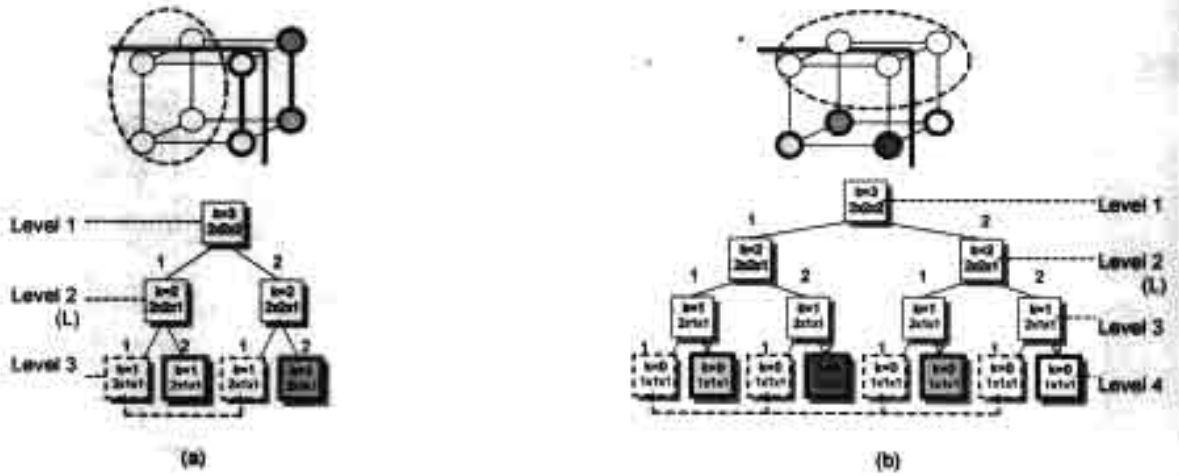


Figure 32: An example of some Sub-Buddy Combining on a hypercube (or 3-cube): a) One ({1,1}) of two 2-cubes, combined at level 3 and b) One ({1,1,1,1}) of two 2-cubes combined at level 4.

Note that in any practical k-D system, we usually apply this combining algorithm for a k-D system with any k dimension and n = 2, namely hypercube networks. However, for n > 2, we may not apply this combining algorithm because the nature of the partitioning process may not base upon the partitioning by network degree, as we will discuss next.

3.3.2 Combining by Network Size

The sub-system combining by network size corresponds to the partitioning that allows all dimensions to be partitioned. Thus, the definition of adjacent buddies is identified differently from the partitioning by network degree. For any k-D system, we introduce the "Combinations of 2^j Adjacent Buddies" algorithm in order to provide a generalized method for combining 2^j buddies (where j = 1, 2, ..., k-1) into the larger free sub-systems. This algorithm (see all steps in Figure 33) is computed in $O(k2^{k-j})$ time for the j^{th} dimension and hence $O(k2^k)$ time for all values of the variable j (since $\sum_{i}^{k-1} k2^{k-i} = 2k\,[2^{k-1}-1]$).

Let R be a considering root node (of size $N = n_1 \times n_2 \times \ldots \times n_k$).
B be a system bit-map of each Buddy ID ($B = b_{k-1} \ldots b_1 b_0$).

(2^{k-j}) Combinations (combine 2^j free buddies in the j^{th} dimension, j=1, 2, ..., k-1): for each j,
1. Create a set (2^{k-j} elements) of (k-j)-bit binary strings; [computed in $O((k-j)2^{k-j})$]
2. Append 2^{k-j} elements with j "*"s to create a set of some (adjacent) k-bit ternary strings; [computed in $O(k2^{k-j})$]
3. Shift left (k-1) times for each ternary string ($T = t_{k-1} \ldots t_1 t_0$) of all 2^{k-j} strings, where $t_i \in \{0,1,*\}$.
(Note: Each ternary string T compounds of (k-j) 0's or 1's and j "*"s.) [computed in $O(k2^{k-j})$]

Figure 33: The "Combinations of 2^j Adjacent Buddies" algorithm for the sub-system combining.

For instance, Figure 34 shows all 2^j adjacent buddies on a 3-D mesh, where j = 1, 2.

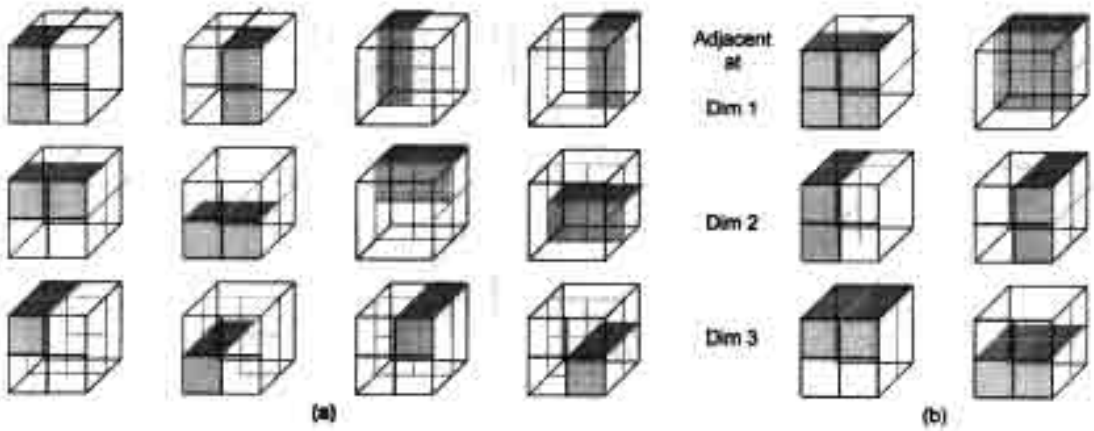


Figure 34: An example of all 2^j Adjacent Buddies for the 3-D mesh: a) j = 1 (2 adjacent buddies) and b) j = 2 (4 adjacent buddies)

As the practical example, consider a 2-D (64x64)-mesh with 4-buddy partitioning (see Figure 26.a), where k = 2 and $2^k = 4$. In order to combine 2^j Buddies, if j = 1, there are $k2^{k-1} = 2x2^{2-1} = 4$ possible combinations (see Figure 35), which are recognized as follows: first, a 4-element set of "a compound of 2^j adjacent buddies" is a set of binary strings {0, 1}. Next, each of these two elements is appended with j * 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. Finally, each ternary string is interpreted. For example, a ternary string 1* = {10, 11} (or Buddy#3 and Buddy#4) and hence the combined system's size is equal to 64 x 44.

Next, in order to simplify the complexity of the combining process, we classify this tree-based sub-system-combining algorithm into three groups, which will be applied later in the allocation and deallocation procedures:

ALGORITHM CS.1 "All 2^k-Buddy Combining": This combining procedure is used to combine all free 2^k buddies (at level L) into a larger free 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 tree as much as possible.

ALGORITHM CS.2 "Some 2^j-Buddy Combining": This combining procedure is used to combine a number (i.e., 2, 4, ..., or 2^{k-1} or $j = 1, 2, ..., k-1$) of adjacent free buddies (of the same root sub-tree) at level L into a larger free sub-system.

Our purpose is to allocate an appropriate sub-system for an incoming task, whose requested size is larger than that of each of 2^k buddies but is less than or equal to that of the combined sub-system.

- First, applying the combining of 2^j adjacent buddies algorithm (in Figure 33), for a ternary string T.
- Then, 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 all steps in Figure 35).

This combining algorithm 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 $O(k^3 2^k)$ for all j.

Let R be a considering root node (Size $N = n_1 \times n_2 \times \ldots \times n_k$ & Base address $\alpha = (a_1, a_2, \ldots, a_k)$).
T be a ternary string $(t_{k-1} \ldots t_1 t_0)$, obtained from the Algorithm in Figure 33.

Buddy-ID-Address-Size Conversion Method

1. Convert the ternary string $T=(t_{k-1} \ldots t_1 t_0)$ to a set (2^j elements) combinable binary-IDs $(b_{k-1} \ldots b_1 b_0)$ [*computed in $O(k2^j+k)$*]
   (e., $T = 1^* \rightarrow \{10, 11\} = \{Buddy\#3, Buddy\#4\}$
   1.1 Create 2^j binaries $(r_{j-1} \ldots r_1 r_0)$ and replace them in location(s) of j-*s in T;
   1.2 Convert binary-IDs back to a set of 2^j integer-IDs;
2. Compute a base-address & a size of each combined sub-system of 2^j sub-systems [*computed in $O(2k2^j)$*]
   2.1 Base address = the base address of the minimum buddy-ID (BD-ID);
   2.2 Combined size = $(n_1' \times n_2' \times \ldots \times n_k')$, for $\forall i$, $i = 1, 2, \ldots, k$;
   $n_i' = n_i$ (of the min BD-ID) + n_i (of the max BD-ID) if (b_{i-1} (of the min BD-ID) = b_{i-1} (of the max BD-ID));
   $n_i' = n_i$ (of the min BD-ID) otherwise.

Figure 35: The "2^j Buddy-ID-Address-Size conversion" algorithm (Algorithm CS.2).

Figure 34 and 36 illustrate all possible combining sub-systems for 3-D and 2-D meshes, when we apply the algorithms, described in Figure 35.

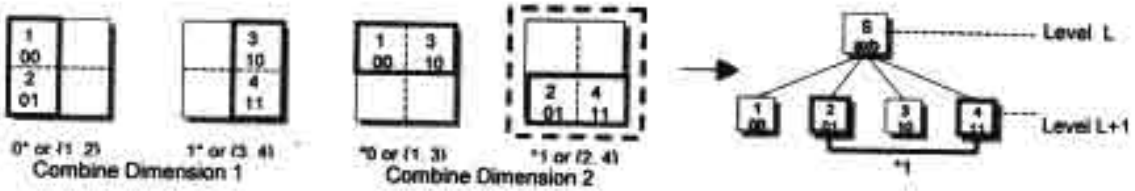


Figure 36: An example of all possible combining sub-systems in Algorithm CS.2 for the 2-D mesh

ALGORITHM CS.3 "1-Buddy and Some Sub-Buddy Combining" and "Some Sub-Buddy 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-systems recognition (from any partitioning size) than those obtained from the combining in Algorithm CS.2. To accomplish these combining subsystems,

- We apply the "Buddy-SubBuddy combining" algorithm (see all steps in Figure 37a) 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.
- We also apply the "SubBuddy-SubBuddy combining" algorithm (see all steps in Figure 37b) for recognizing $k2^{k-1}$ combined sub-systems of some adjacent free nodes at level L+1.

Each of these conversion processes can be computed in $O(k^2 2^k)$ time for each combined node (represented by a ternary string) and $O(k^2 2^{2k})$ time for all possible 2^k strings.

Figure 38 and 39 illustrate two practical examples of all possible combining sub-systems for the 2-D meshes and tori.

Let $B = (b_{k-1} \ldots b_1 b_0)$ be a free buddy (represented in the system bit-map).
$T = (t_{k-1} \ldots t_1 t_0)$ be a ternary string (one of k dimensions to be combined);

Buddy&SubBuddy-ID-Address-Size Conversion Method

1. Compute k combinable buddies $C = (c_{k-1} \ldots c_1 c_0)$ of B from all possible k dimensions;
   i.e., for the j^{th} dim of 2-D mesh, if j = 1, then T = 0* → C = 01 or {Buddy#2} [*computed in* $O(k^2)$]
2. Identify combinable sub-buddies (for each C) as a set of ternary string $S = (s_{k-1} \ldots s_1 s_0)$
   by replacing (k-j) 0's/1's in T with *; and replacing j *'s in T with j b's; [*computed in* $O(k^2)$ *for k C's*]
   for ∀ bit i = 0,1,2, ..., k-1, $c_i = t_i$; $s_i = *$; if $t_i = 0/1$
   or $c_i = b_i'$ $s_i = b_i$; if $t_i = *$
   i.e., for above 2-D mesh, S = *0 → {00, 10} or {SubBuddy#1, SubBuddy#3}
3. Convert set S by using "2^j combinable-Buddy-ID-Address-Size conversion" algorithm (in Figure 33.),
   including #combined sub-systems $(n_1' x n_2' \ldots x n_k')$ and their base addresses; [*computed in* $O((k-j)(3k2^{j-1}+k))$]
4. Compute base-address & size of each combined S from 2 Buddies in the set [*computed in* $O(k^2)$ *for k C's*]
   4.1 Final base address = Base address of the min buddy-ID (of the combined S);
   4.2 Final combined size = $(n_1^* x n_2^* \ldots x n_k^*)$, for ∀i, i = 1, 2, ..., k;
   $n_i^* = n_i'$(of the min BD-ID) + n_i' (of the max BD-ID) if $(b_{i-1}$ (of the min BD-ID) ≠ b_{i-1} (of the max BD-ID));
   $n_i^* = n_i'$(of the min BD-ID) otherwise.

Figure 37a: The "Buddy-SubBuddy-ID-Address-Size conversion" algorithm (Algorithm CS.3-1).

Let $T = (t_{k-1} \ldots t_1 t_0)$ be a ternary string of a pair of combinable 2 buddies.
$B_i = (b_{k-1} \ldots b_1 b_0)$ be one of 2 free sub-buddies (represented in the system bit-map), where i = 1, 2.

SubBuddy&SubBuddy-ID-Address-Size Conversion Method (in this case j = 1)

1. Convert T to a set of (2 elements) buddy-ID $\{B_1, B_2 / B_i = (b_{k-1} \ldots b_1 b_0)\}$ of 2 partially free buddies at level L
   by creating 2^j binary-numbers $(r_{j-1} \ldots r_1 r_0)$, replacing them into j *'s of $(t_{k-1} \ldots t_1 t_0)$; [*computed in* $O(k2^j)$]
2. Identify combinable sub-buddies of each of 2 buddies as a set of ternary string $S (s_{k-1} \ldots s_1 s_0)$
   for ∀ bit i = 0, 1, 2, ..., k-1, [*computed in* $O(k^2)$]
   $b1_i = t_i$; $b2_i = t_i$; and $s1_i = *$; $s2_i = *$; if $t_i = 0$ or 1,
   or $b1_i = 0$; $b2_i = 1$; and $s1_i = b1_i'$; $s2_i = b2_i'$; if $t_i = *$
3. Similar to that defined in Algorithm CS.3-1 (in Figue 37).
4. Similar to that defined in Algorithm CS.3-1 (in Figure 37).

Figure 37b: The "some SubBuddy-ID-Address-Size conversion" algorithm (Algorithm CS.3-2).

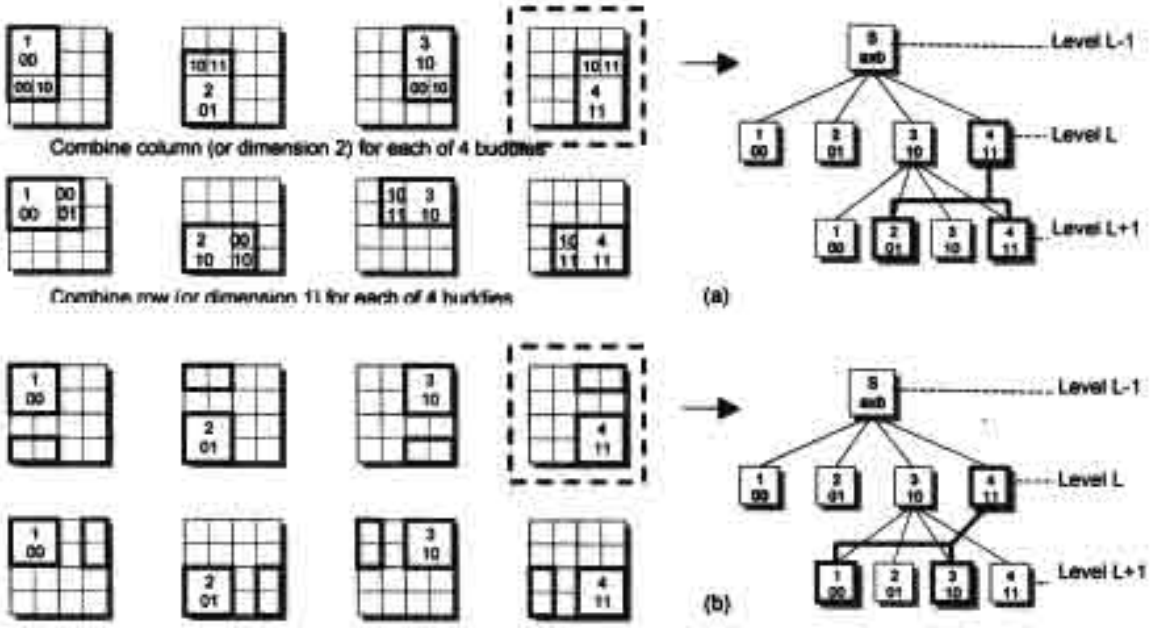


Figure 38: All possible combining of Algorithm CS.3.1: a) for 2-D meshes and b) 8 more for 2-D Tori.

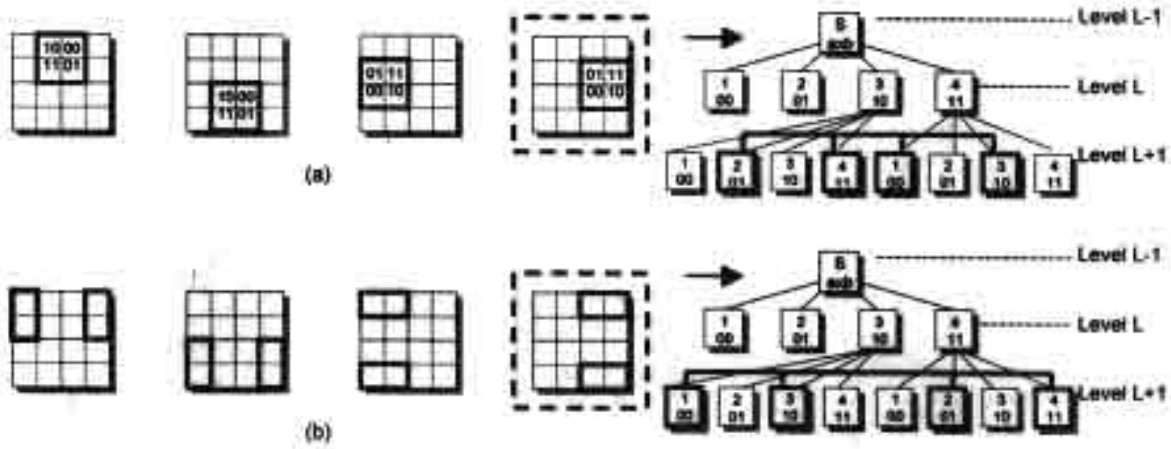


Figure 39: All possible combining of Algorithm CS.3.2: a) for 2-D meshes and b) 4 more for 2-D Tori.

Note that we apply only 2-adjacent-level combining methods in Algorithm CS.3 since others (more than 2 levels) are rarely utilized in practical and are not recognized as a regular non-overlap node and size in the tree. Figure 40 shows the particular example of 4 rectangles that cannot be simply combined on a 2-D mesh (of size N = 64x64), where k = 2. However, in the system performance evaluation of our previous study [41], those combined sub-systems cause very little effect and hence can be ignored.

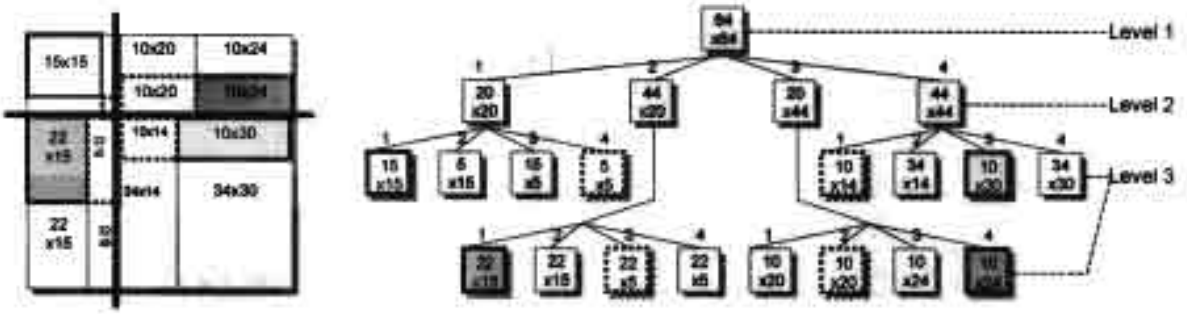


Figure 40: An example of non-combinable sub-system for the case of 3-adjacent-level combining on a 2-D mesh.

3.3.3 Combining by Network Degree and Size

The combining by network degree and size is introduced, corresponding to the partitioning by network degree and size, in order to combine two buddies and some sub-buddies. This combining process (*i.e.*, identifying a sub-partition's status, according to #buddies) is computed in O(1) time for two buddies of the same root and $O(k^4)$ time for other combining of the different roots. Next, in order to simplify the complexity of the combining process, we classify this binary-Tree-based sub-system combining algorithms into five groups, which will be applied later in the allocation and deallocation procedures:

<u>ALGORITHM CDS.1</u> "2-Buddy Combining": This combining procedure is used to combine two free buddies ($n_1 \times n_2 \times \ldots \times p \times \ldots \times n_k$ and $n_1 \times n_2 \times \ldots \times n_i - p \times \ldots \times n_k$) at level L into a larger free tree's node ($n_1 \times n_2 \times \ldots \times n_k$) at level L-1. This combining process is computed in O(1) 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 tree as much as possible.

As compare to the combining by network size, Figure 41a illustrates all possible combined results for 2-D meshes. These are similar to some combining results of ALGORITHM CS.2.

ALGORITHM CDS.2.1 "1-Buddy and 1-SubBuddy Combining (different sizes at dimension i)": This combining procedure is used to combine adjacent free buddy and corresponding sub-buddy at level L-1 and L into a larger free sub-system. This combining process is computed in $O(k^2)$ time for each dimension and $O(k^3)$ for k dimensions. Note: the buddy must be partitioned before combining (see Figure 41b).

ALGORITHM CDS.2.2 "Some 2-SubBuddy Combining (same sizes at dimension i)": This combining procedure is used to combine two adjacent free sub-buddies (from different roots) at level L into a larger free sub-system similar to some combining results of ALGORITHM CS.2 (of the combining by network size). This combining process is computed in $O(k^2)$ time for each dimension and $O(k^3)$ for k dimensions. Note that we can combine one dimension i at of two sub-buddies if their have the same size at dimension i, where i = 1, 2, .., or k. Figure 41c illustrates all possible combined results for 2-D meshes.

ALGORITHM CDS.3 "Some 2-SubBuddy Combining (different sizes at dimension i)": This combining procedure is used to combine two adjacent free sub-buddies (from different roots) at level L and L+1 into a larger free sub-system in case that they have the different size at dimension i. This combining process is computed in $O(k^2)$ time for each dimension and $O(k^4)$ for k dimensions. Figure 41d illustrates all possible combined results for 2-D meshes.

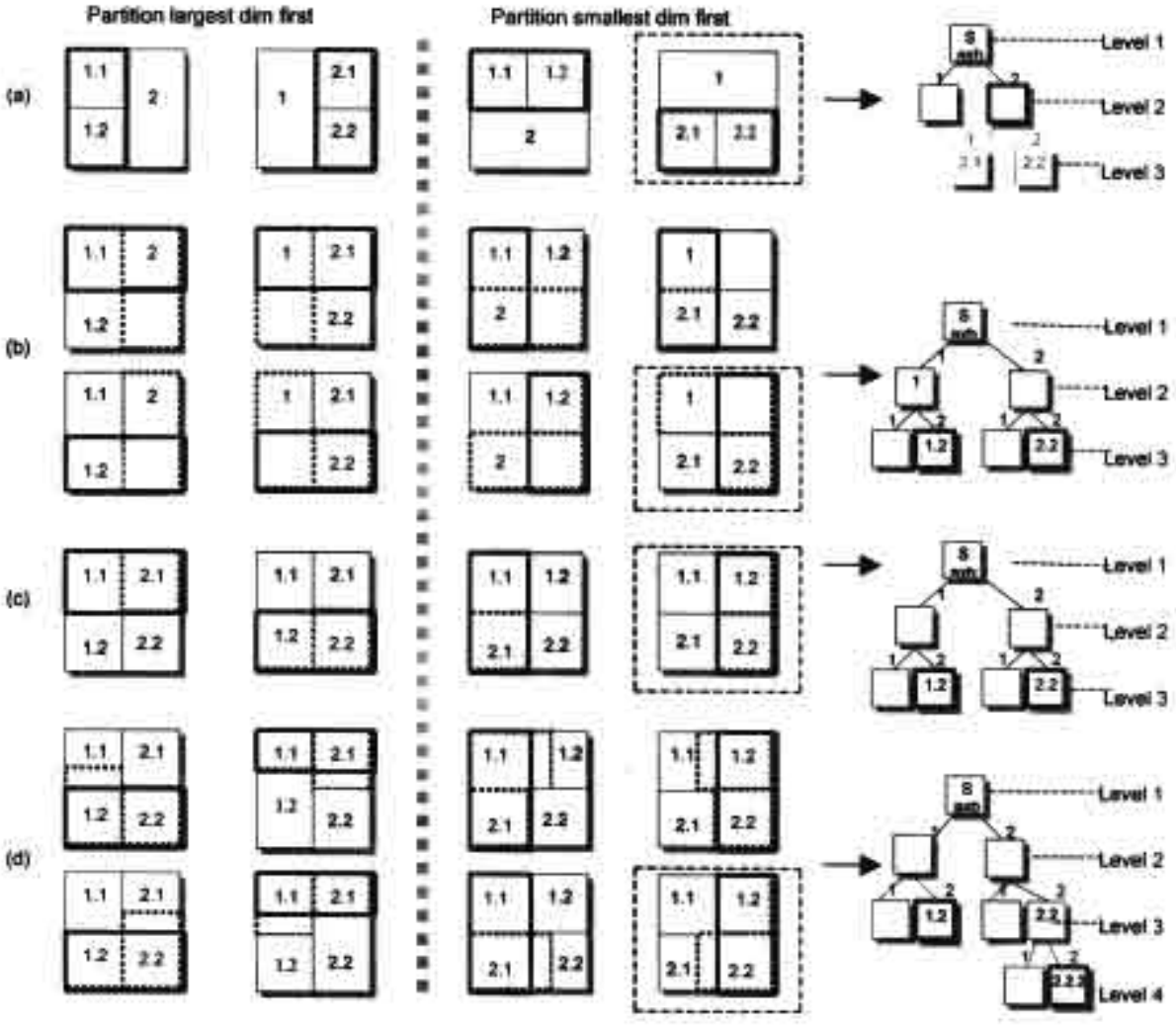


Figure 41: An example of all combined sub-systems in the partitioning and combining by network degree and size on a 2-D mesh : a) results of ALGORITHM CDS.1; b) results of ALGORITHM CDS.2.1; c) results of ALGORITHM CDS.2.2; and d) results of ALGORIHTM CDS.3.

So far, there are two advantages of this partitioning and combining method over the previous two methods:
1. It improves time complexity of the partitioning and combining process and
2. It provides larger free nodes (or fragments) and hence yields less the number of fragments in the system.
However, this combining method yields less recognition capability (in the rare cases) than those obtained from the combining by network size. In practical, such rarely combined sub-systems cause very little effect to the system performance (see results in Section 5).

In particular, ALGORITHM CDS.1-3 are applied directly for the k-D mesh-like systems (i.e., k-D meshes, k-D tori, n-ary k-cubes, etc.) More advantage of this partitioning and combining method is it provides a flexible way of partitioning and covers all networks in the product network class, as describe next.

ALGORITHM CDS.4 "Two Sub-buddies Combining if they are partitioned along the same dimensions": This combining procedure is used to combine two free sub-buddy nodes (i.e. subBuddy#2 of the left sub-tree and the corresponding subBuddy#1 of the right sub-tree) at level L+1 if 2 sub-buddy nodes in L to L+1 levels are partitioned based upon the same dimensions in O(1) time. For levels L+i ≤ n, more sub-systems recognition are found if 2 sub-buddy nodes in L to L+i levels are partitioned based upon the same dimensions. Time complexity of this combining process is O(n+k), where $n = \max(n_1, n_2, n_3, \ldots, n_k)$.

Figure 42 illustrates an example of the combining by network degree and size on a 3-D mesh (N = 64 x 64 x 64) at level 3 (see Figure 42.a) for N' = 64x64x24 and at level 4 (see Figure 42.b) for N" = 64x64x12.

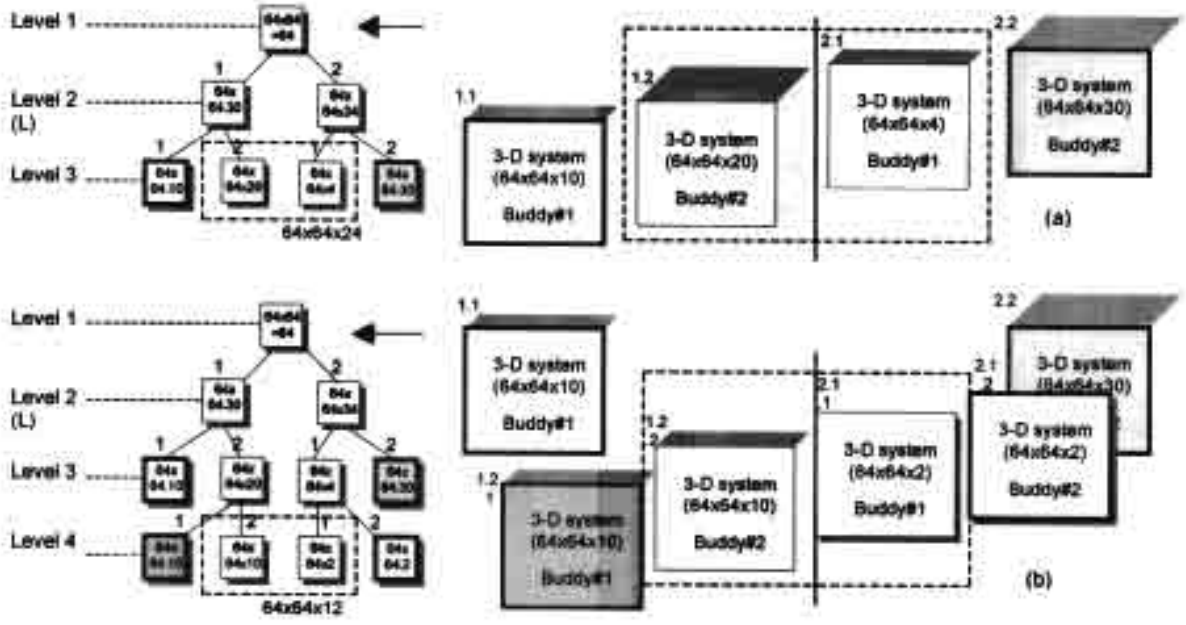


Figure 42: An example of two Sub-Buddy combining (partitioned on the k^{th} dimension) on a 3-D System (N = 64x64x64): a) a combining result at L+1 (N' = 64x64x24) and b) another combining result at L+2 (N" = 64x64x12).

ALGORITHM CDS.5 "Some Sub-buddies Combining if they are partitioned into 2 equal sizes": The combining procedure is special for a particular system (of size $N = n^k$, $n = 2, 2^2, 2^3 \ldots, 2^d$). In this case, it is partitioned each time (or each dimension) into two equal sizes (n/2). This combining procedure is used to combine some free sub-buddy nodes from the same level L (identified from 2 or more-adjacent levels) to yield more sub-systems recognition in $O(j2^j)$ time for a particular j that combines 2^j buddies and hence $O(kd\ 2^{kd})$ time (since $\sum_j j2^j < 2kd[1+2+2^2+\ldots+2^{kd-2}] = kd\ 2^{kd}$) for all js (or all levels) to get the same combined size, where $j = 1, 2, \ldots, kd-1$.

Figure 43 illustrates an example of the combining by network degree and size on a 3-D mesh (N = 64 x 64 x 64, k = 2) at level 3 (see Figure 42.a) for N' = 32x64, at level 4 (see Figure 42.b) for N" = 64x32, and at level 5 (see Figure 42.c) for N"' = 32x64.

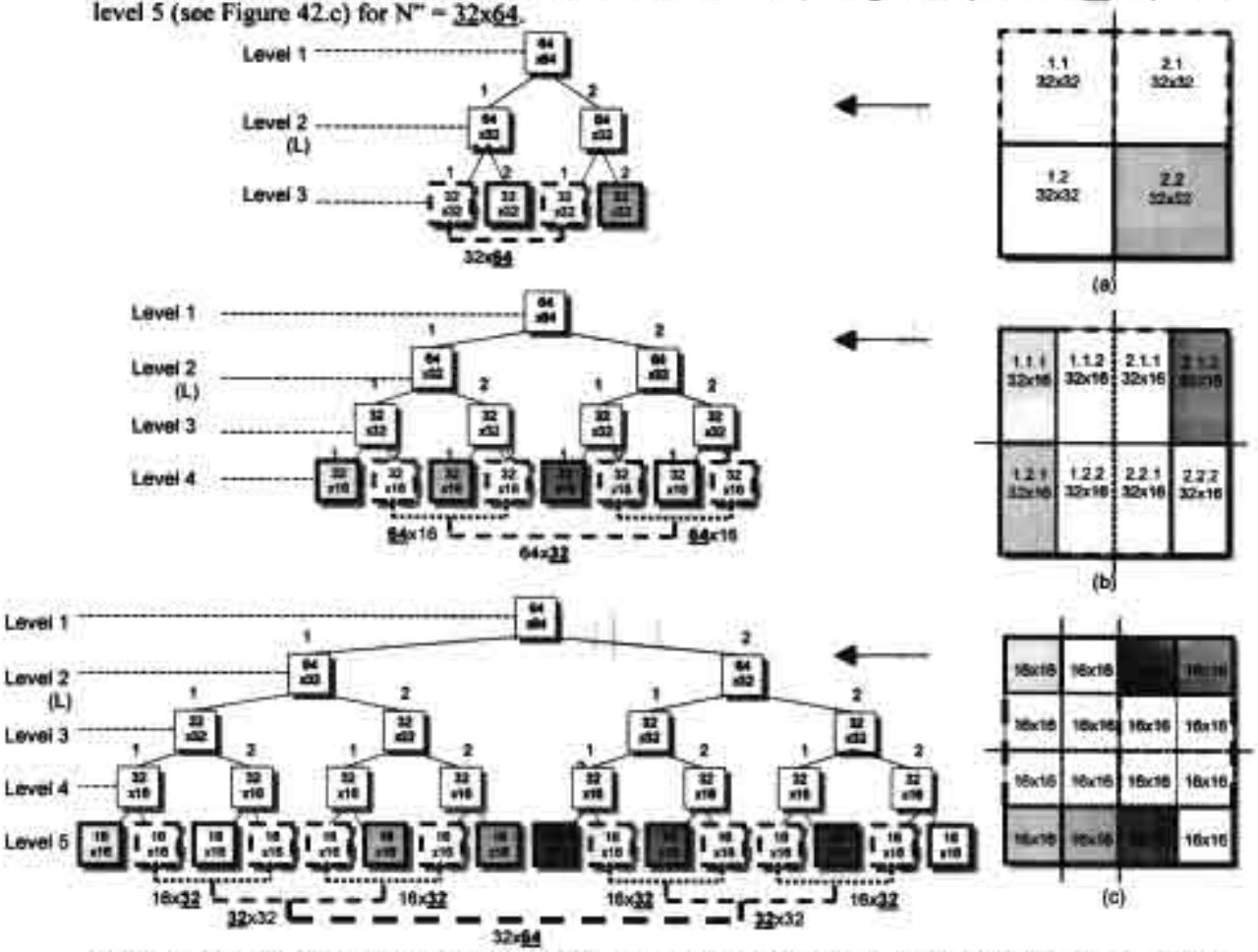


Figure 43: An example of some Sub-Buddy Combining (each of the same size and level) on a 2-D mesh (64x64): a) One of two (32x64) from 2 nodes combined at level 3; b) One (64x32) from 2^2 nodes combined at level 4; and c) one (32x64) from 2^3 nodes combined at level 5.

The last combining method (ALGORITHM CDS.5) can be applied to the hypercube networks (where n = 2, d = 1) with the same time complexity $O(k2^k)$ as that of the combining by network degree (see also Figure 32). Therefore, this partitioning and combining method provides the more generality than that of Method 1 and Method 2 since it can be applied for both of the following cases: 1) for the partitionable systems with small n and high k (i.e., Hypercube networks) and 2) for the partitionable systems with small k and high n (i.e., 2-D Meshes)). Whereas the partitioning and combining by network degree (Method 1) is suitable for the partitionable systems with small n and high k and the partitioning and combining by network size (Method 2) is efficient for the partitionable systems with small k and high n.

Note that "the expanded node size of the combined node" is introduced to improve the disadvantage (many small fragments in the tree) especially for Method 1 and Method 2. Also more benefit of this idea is to limit the number of free nodes (N_F) in the tree and provide the same methodology to update and partition as a regular (free) leaf node. For example, in Figure 23 (for Method 1) the expanded node size is 64x64x63 stored in Buddy#2 (after the allocation of Buddy#1(64x64x1)). In Figure 26 (for Method 2) the expanded node size is 64x44 stored in Buddy#3 (after the allocation of Buddy#1 (20x20).) This expanding process will be applied after the allocation of any task by combining the corresponding buddies and sub-buddies of that allocated node (not all nodes in the tree) and only the maximum free size will be stored in the tree as the expanded free node size and others are masked as busy. This idea will be applied in Section 3.4 (in the CU of the combined node) and Section 3.5 (in the expanded node after the allocation of a task).

3.4 Best-Fit Heuristic

The best-fit heuristic is to find the likely (or approximately) best free sub-system among all available sub-systems for any task. For the partitionable MSIMD/MIMD k-D system, we always apply the best-fit heuristic for processor (PE) or sub-system allocation (in Section 3.4.1) for all tasks requesting SIMD or MIMD mode. In addition, for a particular task with SIMD mode, we have to add the best-fit heuristic to find the corresponding free CU (in Section 3.4.2) for the predefined sub-system or partition.

3.4.1 Best-fit Heuristic for PE Allocation

In our model, for sub-system or processor (PE) allocation decision we apply our previous study [41] to find the likely best sub-system for the k-Tree-based model, based upon the first two partitioning and combining methods. In addition, we introduce the new binary-based method, based upon the partitioning and combining by network degree and size. In particular, there are four effectively criteria that are applied in our resource (CU/PE) allocation model by using the following priority:

Best-Fit Criteria

For each visited node during performing DFS (depth first search) in the tree,
- Criterion 1: Find all free sub-systems (sizes ≥ request) that can preserve the "maximum free size" as possible.
- Criterion 2: If there are many candidates that have the same property in (1), then the candidate that gives the "minimum different size factor (diffSF)" after k-rotations is selected.
- Criterion 3: If there are many candidates that have the same property in (1) & (2), then the "smallest size" candidate that yields the "minimum combining factor (CF)" is selected.

Otherwise, select by random.

Finally after searching to visit all nodes in the tree then for the best free sub-system
- If it is "equal to" the request, then it is directly allocated to the request.
- Criterion 4: Perform the partitioning process. If it is "larger than" the request, then apply partitioning and one of its buddies which yields properties similar to that given in Step 1 – Step 3 plus being the "best buddy node" will be selected.

Note: the criterion 1 to criterion 3 are applied for every free leaf node and every free combined sub-system, whereas the criterion 4 is utilized only once for the best free sub-system, obtained from Steps 1-3. For time complexity, the criterion 1 is applied in O(k) time for all partitioning and combining methods. The criterion 2 is applied in $O(k^2)$ time for all three partitioning and combining methods. The criterion 3 is applied in O(k) time for the first and last partitioning and combining methods and $O(k^2)$ time for the second partitioning and combining method. Finally, the criterion 4 is applied in O(kn), $O(k2^k)$, and O(k) time for three partitioning and combining methods, respectively.

In the practical example of applying the PE best-fit heuristic for the partitioning and combining by network size, consider a 64x64-system with two tasks allocated (see Figure 44). Suppose there are two new incoming tasks (22x5 and 15x10). For the first task T(22x5), searching process starts from the root and then performs DFS to visit all k-Tree nodes. After applying criteria 1-3, the best free node that can accommodate the request is S_1(22, 5) (see Figure 44.a). This free size is equal to the requested task 22x5 and hence we do not have to apply the criterion 4. For another task T(15x10) (see Figure 44.b), after applying the criteria 1-3, the best sub-system at that time is S(22, 15) found at level 3. Next the criterion 4 is applied since it is larger than the requested

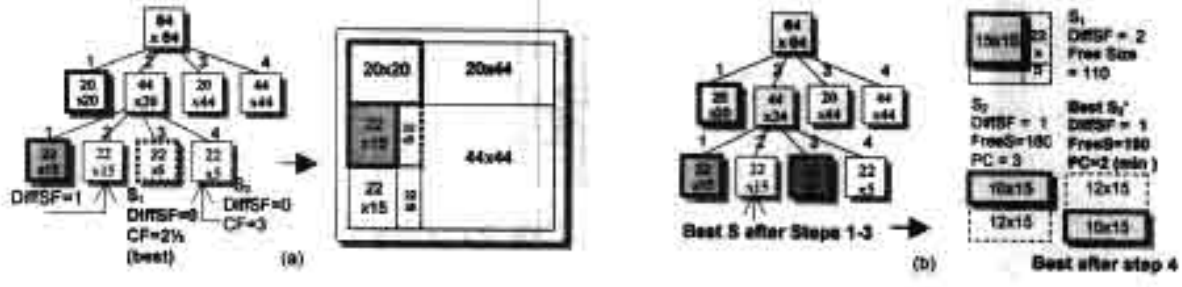


Figure 44: An example of the "best fit" k-Tree-based allocation: a) the best S after step 1-3 for the task (22x5) and b) the best S after step 4 for another task (15x10).

T(15x10), and hence S(22, 15) is partitioned. After partitioning, the rotated size 10x15 (S_2) provides the better best-fit value than that of the regular size 15x10 (S_1). Finally, among S_2 and S_2', the S_2'(10,15) yielding the best-fit value is selected and allocated to the request task (15x10)).

Figure 45 illustrates another application of our binary-Tree-based model when we apply the partitioning and combining by network degree and size to the previous example (described in Figure 44). For the first task T(22x5), searching process starts from the root and then performs DFS to visit all tree nodes. After applying criteria 1-3, the best free node that can accommodate the request is S(44, 5) (see Figure 45.a). Although S(22, 15) and S(44, 5) yield the same different size factor (diffSF = 1), the smaller S(44, 5) is selected . Next the criterion 4 is applied since it is larger than the requested T(22x5), and hence S(44, 5) is partitioned. After partitioning, among S_1 and S_2, the S_1 is selected and allocated to the request task (22x5)). For the second task T(15x10) (see Figure 45.b), this method yields the similar result (applied step 1-4) to that of the above example. Therefore, the S_2'(10,15) is selected and allocated to the request task (15x10)).

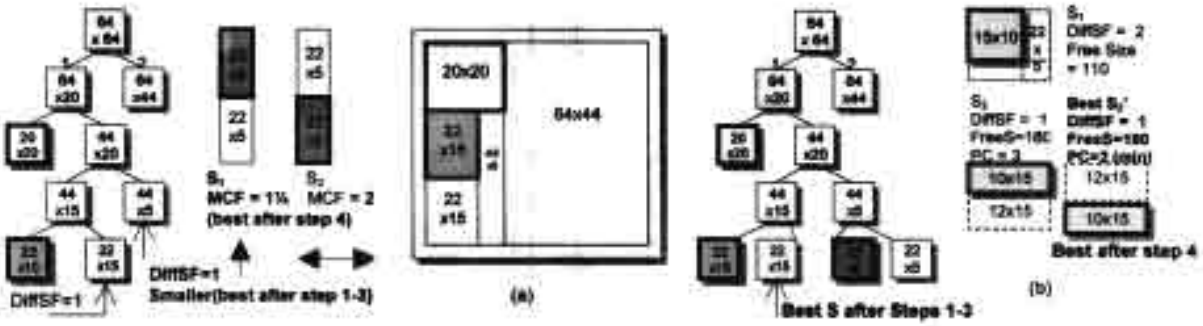


Figure 45: An example of the "best fit" binary-Tree-based allocation: a) the best S after step 4 for the task (22x5) and b) the best S after step 4 for another task (15x10).

3.4.1.1 Algorithm for Criterion 1 (Maintain the Maximum Free Size)

Before applying the criterion 1, we have to find the maximum free size (described later in Section 3.5). Now let's assume that it already exists. During searching into the tree under criterion 1, the maintaining of the maximum free size in our study is identified in terms of the overlap status (disjoint, intersect, or subset) between each node and the maximum free size (see Algorithm A.1). This algorithm can be applied to all three partitioning and combining methods in O(k) time.

ALGORITHM A.1: "Overlap Status" of two available sub-systems S_1 (the maximum free size) and S_2 (any free sub-system) in the tree is identified as follows: Let two sub-systems (S_i, i = 1, 2) represent in term of base-address $\alpha_i = (a_{i1}, a_{i2}, \ldots, a_{ik})$, last-cover address $\beta_i = (b_{i1}, b_{i2}, \ldots, b_{ik})$, and size $|S_i| = (n_{i1} \times n_{i2} \times \ldots \times n_{ik})$, where $b_{ij} = a_{ij} + n_{ij} - 1$; i = 1, 2; and j = 1,2,..., k. Then, S_2 is a "subset" of S_1 if ($a_{2j} \geq a_{1j}$ and $b_{2j} \leq b_{1j}$) for $\forall j$, 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 $\exists j$, j = 1, 2,...,k. Otherwise (neither subset nor disjoint) S_1 "intersects" S_2. Time complexity of identifying each status (disjoint, intersect, or subset) can be computed in O(k) time since at most k dimensions are compared. See some examples of the overlap statuses (disjoint, intersect, or subset) in Figure 46.

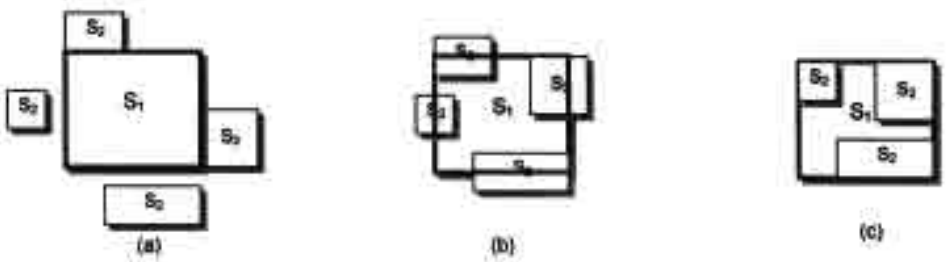


Figure 46: Some possible examples of the overlap statuses: a) disjoint; b) intersect; and c) subset.

Figure 47 illustrates the practical example of "overlap status" for the partitioning & combining by network size. This example shows various overlap statuses between S_1 "the maximum free size" [$n_1 = 6 \times 6$ at <(1, 5), (6, 10)>] and other free sub-systems such as S_2 [$n_2 = 4 \times 4$ at <(5,1), (8,4)>], S'_2 [$n'_2 = 2 \times 4$ at <(5, 7), (6, 10)>], and S''_2 [$n''_2 = 4 \times 2$ at <(5, 5), (8, 6)>]. S_1 and S_2 are disjoint since for i = 2 there exists (a_{12} - a_{22}) = (5 – 1 = 4) ≥ n_{22} (= 4). S'_2 is subset of S_1 since S'_2 = <(5,7), (6, 10)> ⊂ S_1 = <(1, 5), (6, 10)> (or ∀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.

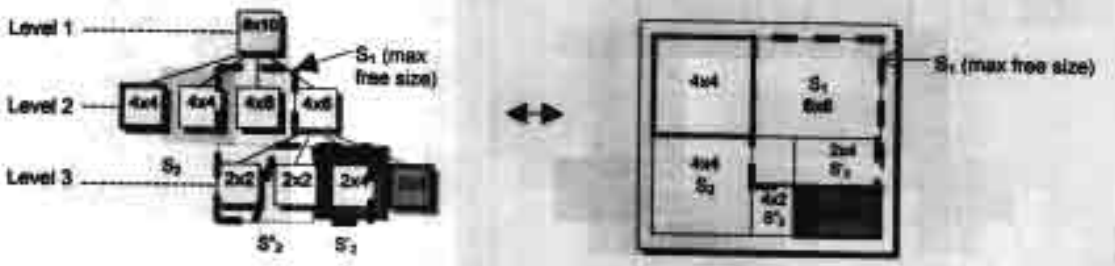


Figure 47: An example of identifying "overlap status" for the partitioning and combining by network size.

3.4.1.2 Algorithm for Criterion 2 (Minimum Different Size Factor)

After applying the criterion 1, we may have to apply the criterion 2 since usually there are many candidates that can preserve the maximum free size. In our study, the minimum different size factor is identified among all rotated sizes (see Algorithm A.2). The minimum different size factor is to find the most fit size (or yielding minimum different size ≤ k) for any free node. Among many rotated sizes of a task that yield the same minimum different size factor (≤ k), the one with maximum remaining size (if partitioned) is selected. This algorithm is computed in $O(k^2)$ time and can be applied to the all three partitioning and combining methods.

ALGORITHM A.2: "Task Rotation" in this study is a process obtained by shifting a task size k-1 times to find the suitable allocate into the free sub-system (S) for the requested (k-D) task of size $p_1 \times p_2 \times \ldots \times p_k$. Thus, there are k possible rotated sizes that can be allocated for the task which are ($p_1 \times p_2 \times \ldots \times p_k$), ($p_2 \times p_3 \times \ldots \times p_k \times p_1$), ..., and ($p_k \times p_1 \times \ldots \times p_{k-1}$), respectively. If that S is partitioned for each rotated size, the different size factor ($0 \leq diffSF \leq k$) and the maximum (remaining) free size (FS) (0 < | FS after partition | < | FS before partition |) are computed in O(k) time and hence in $O(k^2)$ time for all k possible rotated sizes.

See some examples of the task rotation (in Figure 48) for a 3-D system. Suppose we are now considering a free sub-system of size 64 x 64 x 64 for 3 tasks (64 x 32 x 64, 64 x 32 x 32, and 32 x 32 x 32). At a time, each task is rotated and compared to the free sub-system (64 x 64 x 64). For the task 64 x 32 x 64 (Figure 48.a), there are 3 rotated sizes (64 x 32 x 64, 32 x 32 x 64, and 64 x 64 x 32); each of which yield the same diffSF (1) so any one can be selected, except 64 x 64 x 32 is selected for the partitioning and combining by network degree. For the task 64 x 32 x 32 (Figure 48.b), there are 3 rotated sizes (64 x 32 x 32, 32 x 32 x 64, and 32 x 64 x 32); each of which yields the same diffSF (2) then 64 x 32 x 32 is selected. Finally, for the task 32 x 32 x 32 (Figure 48.c), there exists only one rotated size; each of which yields the same diffSF (3). Note: if any rotated size can be selected, we try to accept the one that allows us to partition the highest dimension (i.e., the k dimension) as much as possible because our policy is try to allocate a regular buddy node first; otherwise the combined node is selected.

Figure 48: Some possible examples of the task rotation for 3-D system: a) the free system; b) 3 rotated sizes for task 64x32x64; c) 3 rotated sizes for task 64x32x32; and d) one rotated size for task 32x32x32.

Two practical examples of applying the criterion 2 (the minimum different size factor) are illustrated before in Figure 44.b and Figure 45.b for the last two partitioning and combining methods, respectively. Figure 49 illustrates another practical example of applying "task rotation" for the partitioning and combining by network degree and size. Suppose given a 2-D system (N = 64x64) and there exists a task (20x20) allocated. If the next task requests for a sub-system of size 20x10. After rotated, there are two sizes (20x10 and 10x20) that can be allocated for the task 20x10. First for the size 20x10 (see Figure 49.a), the different size factor is 2 and so we have to apply two partitioning steps and hence its maximum free size after partitioning is 480 processor. Then for the rotated size 10x20 (see Figure 49.b), the different size factor is 1 (min) and then we can apply only one partitioning step and hence its maximum free size after partitioning is 680 processor (max). Therefore, we selected the rotated size 10x20 for the task 20x10.

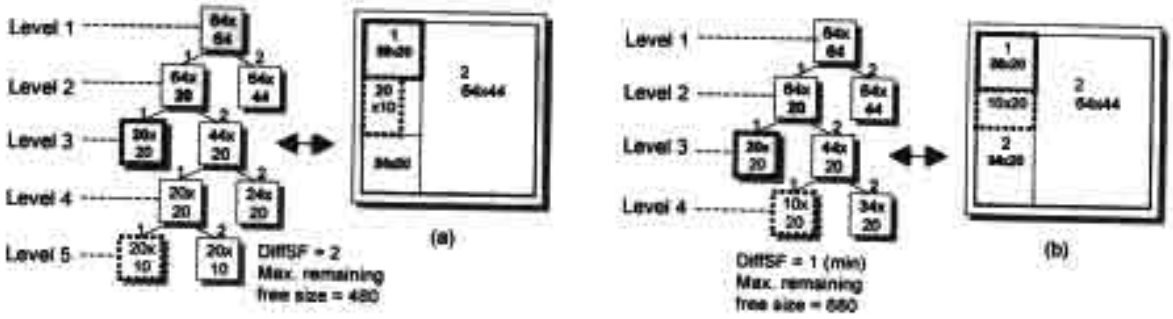


Figure 49: An example of identifying "task rotation" for the partitioning and combining by network degree and size.

3.4.1.3 Algorithm for Criterion 3 (Smallest Size as well as Minimum Combining Factor)

After applying the criterion 1 and 2, consequently we also may have to apply the criterion 3 since sometime we still have some candidates that satisfy both the criteria 1 and 2. To identify the minimum combining factor (minCF), we apply Algorithm A.3 for each node to find the most fit one (or the smallest size and the minimum CF computed from k dimensions). Then among many nodes that yield the same smallest size, the one with minimum combining factor is selected. This algorithm can be applied to all partitioning and combining methods. For time complexity, the criterion 3 is applied in O(k) for the partitioning and combining by network degree, $O(k^2)$ for partitioning and combining by network size, and O(k) for the partitioning and combining by network degree and size.

ALGORITHM A.3-1: "Combining Factor" of any free sub-system S or α (at level L) is computed from its adjacent neighbor nodes (β). In our study, the combining factor is an approximate value (or probability of combining (PC)), which is defined as

$$PC = \begin{cases} 0 & \text{if that particular combined side is one of k system boundaries (since it cannot be combined);} \\ \frac{1}{4} & \text{if its adjacent node of that particular side is busy (since it can be combined after it becomes free);} \\ \frac{1}{2} & \text{if its adjacent node is partially available (and some free sub-buddies may be combined); or} \\ 1 & \text{if its adjacent node is free (or it can be immediately combined).} \end{cases}$$

Then, the combining factor for the first adjacent level is defined in term of a summation of PC of each of combinable buddies (C) from all B buddies (β_{1j}) of the same root sub-tree (R) of the free sub-system S (or α_1), where $B = n_k$, 2^k, or 2 for the three partitioning and combining methods, respectively.

Let $CF_1(\alpha_1, \beta_{1j}) = \sum_{j=1}^{C} PC_j(\alpha_1, \beta_{1j})$ is the combining factor of α_1 and its adjacent nodes β_{1j}

Then, the combining factor computed from k adjacent levels (CF (α)) is identified and hence it is computed as the summation of the CF_i at each level $L - i + 1$ and corresponding dimension j ($CF_i(\alpha_i, \beta_{ij})$), where i = 1, 2,..., k and j = 1, 2,..., C (combinable buddies of α). Note: we use the corresponding root of α (R(α)) in the adjacent level.

$$CF(\alpha) = \sum_{i=1}^{k} CF_i(\alpha_i = R(\alpha_{i-1}), \beta_{ij}) \qquad \text{where } \alpha_1 = \alpha.$$

Finally, we handle the effect of system boundary as follows: for i =1,2,...,k (dimensions) ; if (a_i=1 or b_i=n_i) then increment system boundary (SB) by 1, where at beginning the value of SB is set to zero. Then, we add the effect of combining with the system boundary (k - SB) into the CF(α) that is CF(α) = CF(α) + (k - SB). Since combinable buddies of α (or C) = 2, k, or 1 for three partitioning and combining methods (see Algorithm A.3-2), then their time complexities are O(k), O(k^2), and O(k), respectively.

ALGORITHM A.3-2: "Combinable nodes (C)" for each partitioning and combining method is identified based upon its corresponding partitioning and combining method, as follows:

- For the partitioning and combining by network degree, time complexity of the CF(α) is O(1) and the number of combinable buddies (C) are C = 1 (if n_i = 2 i.e., a hypercube network) or C = 2 (if n_i > 2) and their IDs are defined as follows: See some corresponding examples in Figure 50 and 51.

  If n_i = 2 and if ID of α = id, then C = 1 (β) and ID of β_i = 1 (if id = 2) or 2 (if id = 1).
  If n_i > 2 and if ID of α = id, then C = 2 (β_1, β_2) and

  $$\text{ID of } \beta_{i1} = \begin{cases} \text{id -1} & \text{if id = 2, 3, ..., } n_i \text{ for both networks and} \\ \text{0 (SB)} & \text{if id = 1 for a non-wraparound network or} \\ n_i & \text{if id = 1 for a wraparound network.} \end{cases}$$

  $$\text{and ID of } \beta_{i2} = \begin{cases} \text{id+1} & \text{if id+1} \le n_i\text{; otherwise 0 (SB) for a non-wraparound network (a buddy).} \\ \text{id+c} & \text{if id+c} \le n_i\text{; otherwise 0 (SB) for a non-wraparound network (a combine).} \\ \text{(id+1) mod } n_i & \text{for a wraparound network (a buddy).} \\ \text{(id+c) mod } n_i & \text{for a wraparound network (a combine).} \end{cases}$$

  where SB = System Boundary, c = #combined nodes.

- For the partitioning and combining by network size, the number of combinable buddies (C = k) and time complexity of the CF(α) is O(k^2) and each combinable node j at level L – i (β_{ij}, where i, j = 1, 2, ..., k) is defined by using the k-bit-map as follows: See also some corresponding examples in Figure 52 and 53.

  $\beta_{ij} = (b_{k-1} \ldots \underline{b_j} \ldots b_1 b_0)$ (or negate the j^{th} bit of α = $(b_{k-1} \ldots b_1 b_0)$; b_j = 0 or 1, for a buddy.
  $\beta_{ij} = (t_{k-1} \ldots \underline{t_j} \ldots t_1 t_0)$ (or negate the j^{th} bit (non*) of α=$(t_{k-1} \ldots t_1 t_0)$; t_j = 0,1,or * for a combine.

- For the partitioning and combining by network degree and size, there is only one combinable buddy (C = 1) at each level and time complexity of the CF(α) is O(k) and the combinable node at level L – i (β_i, where i = 1, 2, ..., k) is defined by negating ID of α (id), then ID of β_i = 1 (if id = 2) or 2 (if id = 1). See also a corresponding example in Figure 54.

Some examples of the combinable nodes for the partitioning and combining by network degree (method 1) are illustrated in Figure 50 for a 2-D system (i.e., mesh (non-wraparound network) or torus (wraparound network).) First, Figure 50.a illustrates two combinable buddies of a buddy (α = 2) for a 2-D mesh, which are β_{1j} = 1, 3 (indicated in dash blocks) for j = 1, 2, respectively. Figure 50.b illustrates two combinable buddies of a buddy (α = n_2) for a 2-D torus, which are β_{1j} = n_2–1, 1. Figure 50.c shows only one combinable buddy of a combine node (α = 1-2) for another 2-D mesh, which are β_1 = 3. Finally, Figure 50.d illustrates two combinable buddies of a combine node (α = $n_{k-1} - n_k$) for another 2-D torus, which are β_{1j} = n_2–2, 1 for j = 1, 2, respectively.

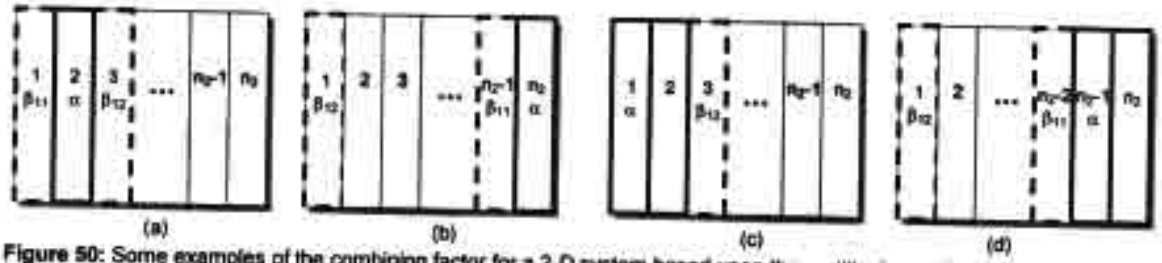


Figure 50: Some examples of the combining factor for a 2-D system based upon the partitioning and combining by network degree: a) a buddy 2-D mesh; b) a buddy 2-D torus; c) a combine 2-D mesh; and d) a combine 2-D torus.

Figure 51 illustrates the practical example of applying the partitioning and combining by network degree. Given a 2-D mesh (or non-wraparound network) and a considering node α at level 3 (L). The combinable β_{11} and β_{12} (at level 3) and β_{22} (at level 2) are illustrated in the k-Tree and the system status. Note: at level 2 (L-1), there is not β_{21} since it is a boundary of the system. Then, the combining factor (CF) of α (where k = 2) in Figure 51 is computed as follows:

$$CF(\alpha) = CF_1(\alpha_1 = \alpha, \beta_{1j}) + CF_2(\alpha_2 = R(\alpha_1), \beta_{2j})$$

$$= [PC_1(\alpha_1, \beta_{11}) + PC_2(\alpha_1, \beta_{12})] + [PC_1(\alpha_2, \beta_{21} + PC_2(\alpha_2, \beta_{22}]$$

$$= [1 + \tfrac{1}{4}] + [0 + 1] = 2\tfrac{1}{4}.$$

Finally, when considering the system boundary effect (where k = 2 and SB = 1), then CF(α) is 2¼ + (k - SB) = 3¼ since one size of this sub-system is a system boundary.

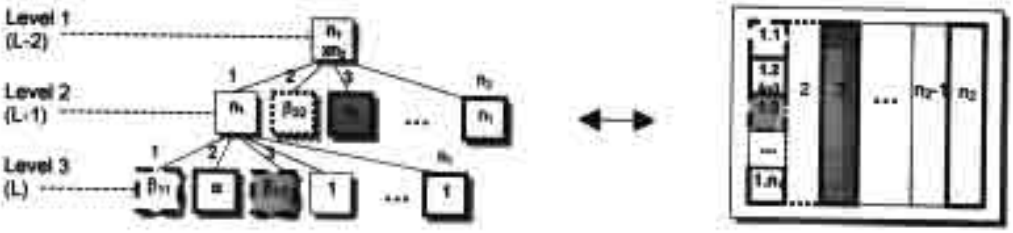


Figure 51: An example of computing CF(α) for the partitioning and combining by network degree.

Some examples of the combinable nodes for the partitioning and combining by network size are also illustrated in Figure 52 for a 2-D mesh system. Figure 52.a illustrates two combinable buddies of a buddy (α), where ID of α = 1 (00), 2 (01), 3 (10), or 4 (11), respectively. In the first figure, where ID of α = 1 (or 00), then IDs of two combinable nodes are $\beta_{11} = 0\underline{1}$ (or ID = 2) and $\beta_{12} = \underline{1}0$ (or ID = 3). Figure 52.b illustrates two combinable buddies of a combine node (α = *0 or {00, 10}). Therefore, the two combinable nodes are $\beta_{11} = 0\underline{1}$ (or ID = 2) and $\beta_{12} = 1\underline{1}$ (or ID = 4) respectively.

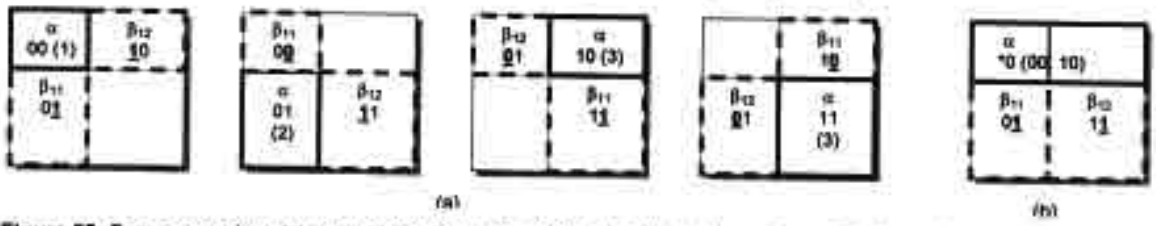


Figure 52: Some examples of the combining factor for a 2-D system based upon the partitioning and combining by network size: a) 4 possible case of a buddy and b) a combine node.

Figure 53 illustrates another practical example of applying the partitioning and combining by network size. Given a 2-D mesh (N = 64 x 64). Let α (or α_1) = b_1b_0 = 11 (or ID = 4), residing at level 3 (L) and 2 combinable buddies of α are $\beta_{11} = 1\underline{0}$ (or ID = 3) ; $\beta_{12} = \underline{0}1$ (or ID = 2). Assume the new considering node at level 2 (L-1) is the root of node α_1 at level 2 is $\alpha_2 = R(\alpha_1) = 10$ (or ID = 3) and then 2 adjacent nodes of α are $\beta_{21} = 1\underline{1}$ (or ID = 4) and $\beta_{22} = \underline{0}0$ (or ID = 1).

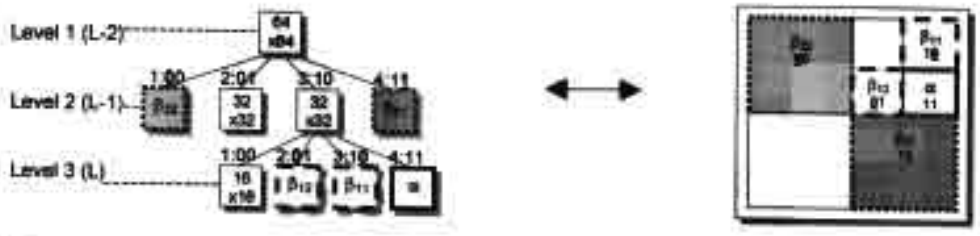


Figure 53: An example of computing CF(α) for the partitioning and combining by network size.

Then, the combining factor (CF) of α (where k = 2) in Figure 53 is computed as follows:

$$CF(\alpha) = CF_1(\alpha_1 = \alpha, \beta_{1j}) + CF_2(\alpha_2 = R(\alpha_1), \beta_{2j})$$

$$= [PC_1(\alpha_1, \beta_{11}) + PC_2(\alpha_1, \beta_{12})] + [PC_1(\alpha_2, \beta_{21} + PC_2(\alpha_2, \beta_{22}]$$

$$= [1 + \frac{1}{4}] + [\frac{1}{4} + \frac{1}{4}] = 1\frac{3}{4}.$$

Finally, when considering the system boundary effect (where k = 2 and SB = 2), then CF(α) is $1\frac{3}{4} + (k - SB) = 1\frac{3}{4}$

Figure 54 illustrates the practical example of applying the partitioning and combining by network degree and size. Given a 2-D mesh (N = 64x64). Let α is a Buddy#2 at level 5. A combinable buddy of α is β_1 = Buddy#1 (at the same level). For another level, β_2 = Buddy#1 (at level 4) since the root of α is Buddy#2 at level 4. Then, the combining factor (CF) of α (where k = 2) in Figure 54 is computed as follows:

$$CF(\alpha) = CF_1(\alpha_1 = \alpha, \beta_1) + CF_2(\alpha_2 = R(\alpha_1), \beta_2) = \frac{1}{4} + 1 = 1\frac{1}{4}.$$

Finally, when considering the system boundary effect (where k = 2 and SB = 1), then CF(α) is $1\frac{1}{4} + (k - SB) = 2\frac{1}{4}$.

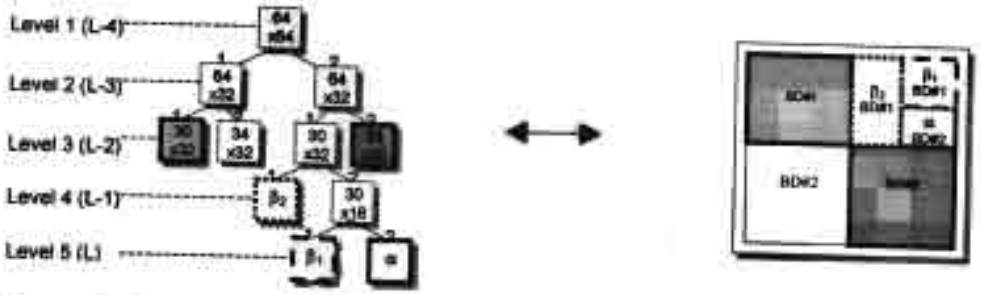


Figure 54: An example of computing CF(α) for the partitioning and combining by network degree and size.

3.4.1.4 Algorithm for Criterion 4 (Best Buddy Location after Partitioning)

After searching to visit all nodes in the tree, we obtain the best sub-system, whose size may equal or larger than the requested task. If the sub-system size already fits to the task, we do not have to apply the partitioning process and this criterion. Otherwise, we apply the partitioning process and Algorithm A.4 for each buddy node to find the most fit one (or the like best buddy node among B nodes, where the number of buddies B = n_k, 2^k, or 2 for three partitioning and combining methods in $O(kn_k)$, $O(k2^k)$, O(k), respectively.

ALGORITHM A.4: "The Best Buddy (or Best Sub-partition)" is applied for partitioning process (Step 4 in the best-fit heuristic). In this case, assume the best free sub-system from Steps 1-3 is the node represented as S, whose size is larger than the requested task. Then, the node S will be partitioned into B buddies and the best sub-partition (or one of B buddies will be allocated to the request), where B = n_k, 2^k, or 2 for the three partitioning and combining methods, respectively. In this case, we apply an approximate probability of combining (PC = 0, 1/4 , 1/2 , or 1) similar to that of the criterion 3.

However, the combining factor for the k-1 adjacent levels is similar for all possible B buddies since they always have the same root. Therefore, first we just apply the system boundary effect for each buddy, as follows: for i = 1, 2, ..., k ; if (a_i = 1 or b_i = n_i) then increment system boundary (SB) by 1, where at beginning the value of SB is set to zero. Then, the effect of combining with the system boundary is defined in terms of the probability of combining (PC = k – SB), which is computed for all buddy nodes in O(kn), $O(k2^k)$, O(k) time for three partitioning and combining methods, respectively. The buddy (α) that yields the minimum probability of combining (min PC), corresponding to the system boundary is selected.

If all buddy nodes provide the same PC, then we justify our decision in order to select the likely best buddy (α), based upon the local combining capability in terms of the probability of combining (PC), which is defined as follows:

- For the partitioning and combining by network degree, the best buddy (one of all n nodes) will be identified as follows: for j = 1, 2, ..., n; PC_j = PC (sub-buddies j) of the left combinable buddy (β_1) (or PC (β_1) if its sub-buddies do not exist) + PC (sub-buddies j) of the right combinable buddy (β_2) (or PC (β_2) if its sub-buddies do not exist). Then, the ID of α is set equal to the ID of the sub-buddy or j (if it yields the minimum PC). This process is computed in O(n) time since we have n buddy nodes and only two combinable nodes for each buddy. See some corresponding examples in Figure 55 and 56.

- For the partitioning and combining by network size, the best buddy (one of all 2^k nodes) can be identified directly from the ID of the best sub-system S (from step 1-3 or the one that needs to be partitioned). That is the ID of α is set equal to the ID of S (1, 2, ..., or 2^k). This can be computed in $O(2^k)$ time since we have 2^k buddy nodes. See some corresponding examples in Figure 57 and 58.

- For the partitioning and combining by network degree and size, the best buddy (one of two nodes) will be identified as follows: for j = 1, 2; PC_j = PC (sub-buddies j) of the combinable buddy (β) (or PC (β) if its sub-buddies do not exist). Then, the ID of α is set equal to the ID of the sub-buddy j (if it yields the minimum PC). This process is computed in O(1) time since we have only two buddy nodes. See some corresponding examples in Figure 59 and 60.

Some examples of identifying the best buddy nodes for the partitioning and combining by network degree are illustrated in Figure 55. Figure 55.a shows the computing of PC_j (1¼), which is equal for j = 1, 2, 3, and 4. Therefore, any buddy can be selected. Then, in this case the minimum address is selected. Figure 55.b illustrates another different result of PC_j (1¼ or ½) which is equal for all js, except PC_2 = ½.

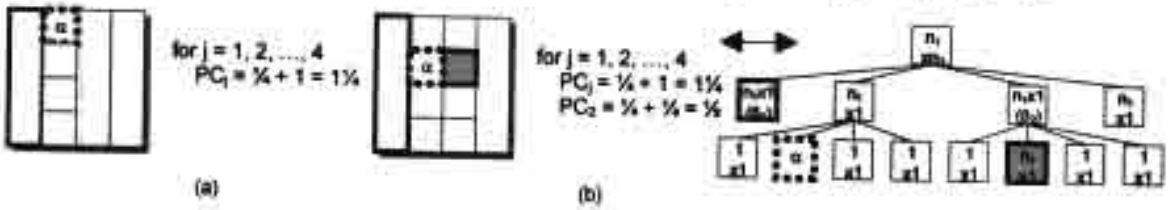


Figure 55: Some examples of the best buddy for the partitioning and combining by network degree: a) any of n buddies can be the best node; so the first one is selected and b) the second buddy is selected since it yields the minimum PC.

Figure 56 illustrates the practical example of applying the partitioning and combining by network degree. Given a 5-cube (or hypercube) system and four tasks (two 3-cubes, one 2-cube, and one 1-cubes) allocated. In this case, the 3-cube node is partitioned into two buddies. The PC of Buddy#1 is ¼ since its corresponding sub-buddy at the same level is busy. The PC of Buddy#2 is ½ since its corresponding sub-buddy at the same level is partially free. Then the Buddy#1 yielding the minimum PC is selected.

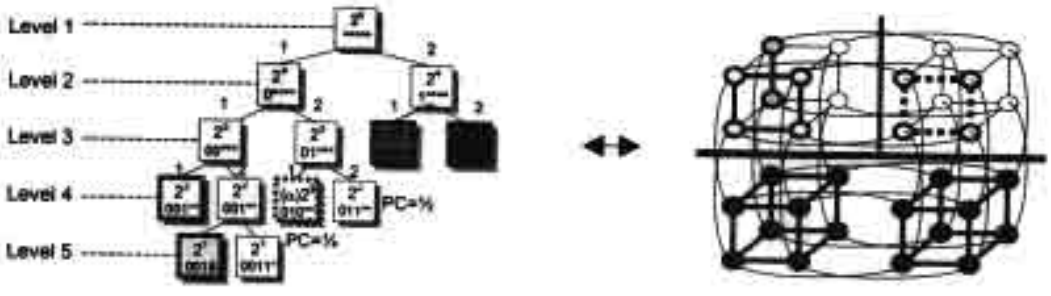


Figure 56: An example of identifying the best buddy for the partitioning and combining by network degree.

Some examples of identifying the best buddy nodes for the partitioning and combining by network size are illustrated in Figure 57. In the first figure, the ID α (or the selected buddy) is 1, which is the same as the ID of S. If the ID of buddy is 4 (the last figure), the ID of α (or the selected buddy) is Buddy#4.

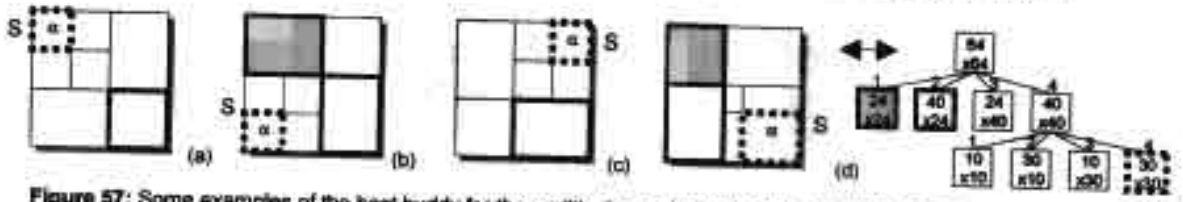


Figure 57: Some examples of the best buddy for the partitioning and combining by network degree.

Figure 58 illustrates the practical example of applying the partitioning and combining by network size. Given a 2-D mesh system (N = 8x10) and two tasks (4x4 and 2x4) allocated. For the next incoming task (3x3), suppose after applying the best fit heuristic, the best free node is Buddy#2 (4x4) at level 2 is selected. The PC of buddy 1, 2, 3, and 4 are 1, 0, 2, 1, respectively. Therefore, the sub-Buddy#2 is select since it yields the minimum PC = 0 (i.e., k = 2, SB = 2).

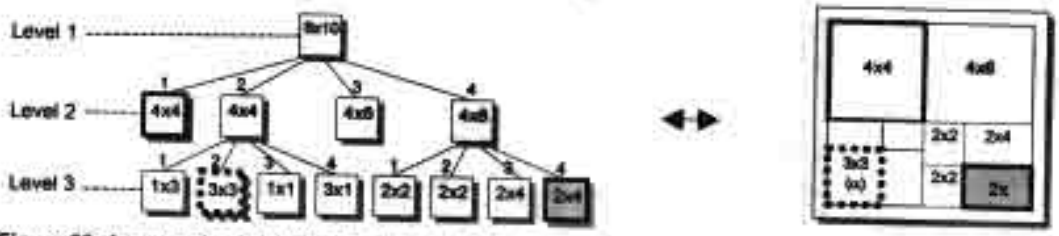


Figure 58: An example of identifying the best buddy for the partitioning and combining by network size.

Some examples of identifying the best buddy nodes for the partitioning and combining by network degree and size are illustrated in Figure 59. Figure 59.a shows the same PC of the first and second buddy, which is 0 (since k = 2 and SB = 2). Then the local combining factor is applied. Now the PC of the first buddy is ¼ and the PC of the second buddy is 1 and hence the first buddy is selected as the best buddy. Similar result is illustrated in Figure 59.b.

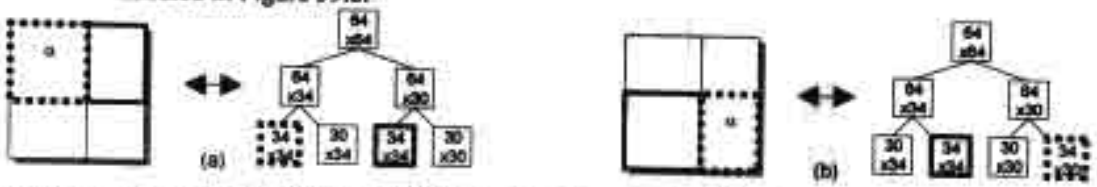


Figure 59: Some examples of the best buddy for the partitioning and combining by network degree and size: a) the first buddy is the best node and b) the second buddy is selected.

Figure 60 illustrates the practical example of applying the partitioning and combining by network degree and size. Given a 2-D mesh system (N = 64x64) and three tasks (20x20, 22x15, and 22x5) allocated. For the next incoming task (10x15), the best sub-system after step 1-3 is the second buddy (22x15) at level 5. After partitioning, the second buddy that yields the minimum PC (=0) is selected.

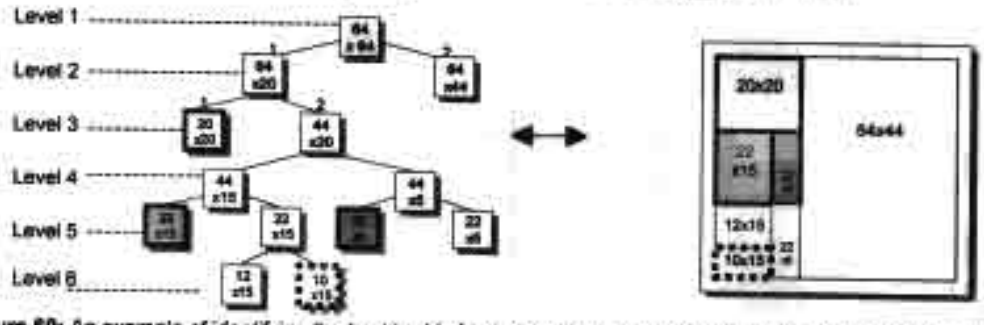


Figure 60: An example of identifying the best buddy for the partitioning and combining by network degree and size.

3.4.2 Best-Fit Heuristic for CU Allocation

In the reconfigurable SPP MSIMD/MIMD system design [2] that we use as our system-base for the resource (CU/PE) allocation, all processors are specially designed, called CPEs (control processor elements) since their roles (CU or PE) are assigned at run time. Therefore, in our study the CU for the selected sub-system (S) can be any CPE that is directly connected to that sub-system. For the CU allocation decision, first we introduce the general CU searching (in Section 3.4.2.1) and then the application of the general CU searching to our tree-based CU allocation method (in Section 3.4.2.2).

3.4.2.1 The General CU Scanning Methods

In general, we introduce two main strategies to find the appropriate CU for a selected sub-system, namely: 1) the processor-bit CU-scanning strategy in O(kN) time and 2) the k-sub-system CU-scanning strategy in $O(k^2)$ time, where N represents the system size ($N = n_1 \times n_2 \times \ldots \times n_k$).

3.4.2.1.1 The Processor-Bit CU-Scanning Strategy

Given a selected sub-system (S) of size $N' = m_1 \times m_2 \times \ldots \times m_k$ at address $\langle(a_1, a_2, \ldots, a_k), (b_1, b_2, \ldots, b_k)\rangle$, where the first k-coordinate $(a_1, a_2, \ldots, a_k)$ represents the base address and the second k-coordinate $(b_1, b_2, \ldots, b_k)$ represents the last cover address of the sub-system. Suppose the system size is $N = n_1 \times n_2 \times \ldots \times n_k$ and $m_k \le n_k$. First, we illustrate all possible processors (or CPEs); each of which can be assigned the CU role for the sub-system, by using two simple examples for 2-D and 3-D mesh systems (see Figure 61).

For the 2-D system (see Figure 61.a), it is not too difficult to find all possible processors being around the sub-system (S). Next in order to find an appropriate free CU at boundary of S, we can start scanning from the minimum address processor and go through all of them (one by one) from left to right and also from top to bottom. In the scanning from left to right (along dimension 1), we have to do the scanning twice for the top and the bottom. In the scanning from top to bottom (along dimension 2), we also have to do the scanning twice for the left and the right. However, it is more difficult for the 3-D system (see Figure 61.b) to perform such scanning for all possible CUs, and hence it is the most difficult for any k-D system.

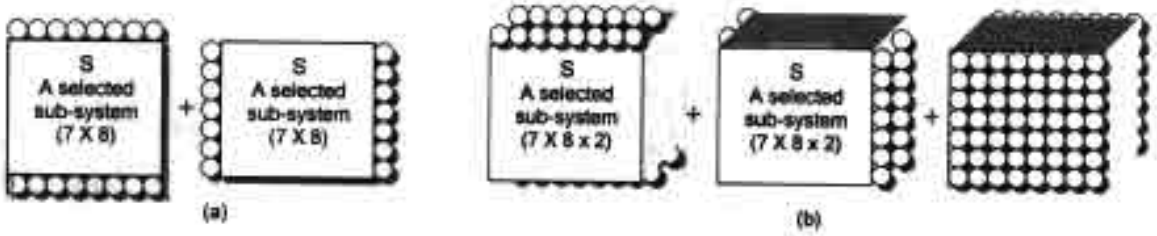


Figure 61: Some examples of a selected sub-system (S) and all of its corresponding CUs at boundary of S: a) all CUs for a 2-D mesh sub-system ($N' = 7 \times 8$) and b) all CUs for a 3-D mesh sub-system ($N' = 7 \times 8 \times 2$).

In general for a k-D system, a number of all possible CPEs that can be the CU (or candidate CUs) of the subsystem S are scanned from the first dimension to the k^{th} dimension and are identified as follows:

A number of all candidate CUs of S

$$= 2\sum_{i=1}^{k} m_1 \times m_2 \times \ldots \times m_{i-1} \times 1 \times m_{i+1} \times \ldots \times m_k$$

$$= 2[(1 \times m_2 \times \ldots \times m_{k-1} \times m_k) + (m_1 \times 1 \times \ldots \times m_{k-1} \times m_k) + \ldots + (m_1 \times m_2 \times \ldots \times 1 \times m_k) + (m_1 \times m_2 \times \ldots \times m_{k-1} \times 1)]$$

For example (see Figure 61 for the 2-D and 3-D systems), if $k = 2$ and the size of a considered sub-system (S) is $N' = m_1 \times m_2$ (7 x 8), all candidate CUs for this sub-system is $2(1 \times 8 + 7 \times 1) = 30$ processors (Figure 61.a). For $k = 3$ and the size of another sub-system (S) is $N'' = m_1 \times m_2 \times m_3$ (7 x 8 x 2), all candidate CUs for this sub-system is $2(1 \times 8 \times 2 + 7 \times 1 \times 2 + 7 \times 8 \times 1) = 172$ processors (Figure 61.b)

Next, we have to find the address of each of those candidate CUs and indicate its status (i.e., free (0) or busy (1)). In the k-D system, a processor address is represented in terms of a k-coordinate such as the base address $(a_1, a_2, \ldots, a_k)$, the cover address $(b_1, b_2, \ldots, b_k)$, etc. For the sub-system (S) of size $(N' = m_1 \times m_2 \times \ldots \times m_k)$ at address $\langle (a_1, a_2, \ldots, a_k), (b_1, b_2, \ldots, b_k) \rangle$, the minimum address of the first possible CU is $(a_1\text{-}1, a_2, \ldots, a_k)$. The scanning process starts from the first dimension to the k^{th} dimension. For each dimension i (i = 1, 2, ..., k), there are two consecutive groups of $(m_1 \times m_2 \times \ldots \times m_{i-1} \times 1 \times m_{i+1} \times \ldots \times m_k)$ processors, where each of their addresses is identified as follows:

CUs at dimension i = 1, 2, ... k: addressing of $2(m_1 \times m_2 \times \ldots \times m_{i-1} \times 1 \times m_{i+1} \times \ldots \times m_{k-1} \times m_k)$ CUs (where $m_i = 1$) are

Initialize Group 1 $(a_1, a_2, \ldots, a_i\text{-}1, \ldots, a_k)$

Group 2 $(a_1, a_2, \ldots, b_i\text{+}1, \ldots, a_k)$, where $b_i = a_i + m_i - 1$

Computation all dimensions, except i (since $m_i = 1$)

for $d_1 = 0, 1, 2, \ldots, m_1\text{-}1$ a_1+d_1

for $d_2 = 0, 1, 2, \ldots, m_2\text{-}1$ a_2+d_2

...

for $d_k = 0, 1, 2, \ldots, m_k\text{-}1$ a_k+d_k

Note: for each dimension i if $(a_i\text{-}1 < 0)$, it is a system boundary and no CUs in group 1 of that dimension.

For example (see Figure 62), if k = 2 and the size of a selected sub-system (S) is $N' = 7 \times 8$, addressing of all 30 candidate CUs, where $m_1 \times m_2 = 7 \times 8$ at $\langle (a_1, a_2), (b_1, b_2) \rangle = \langle (5, 5), (11, 12) \rangle$ are

- Candidate CUs at dimension 1: for $d_2 = 0, 1, 2, \ldots, 7$

  Group 1 (8 PEs): $(\mathbf{a_1\text{-}1}, a_2+d_2) = (\mathbf{4}, 5), (\mathbf{4}, 6), \ldots, (\mathbf{4}, 12)$

  Group 2 (8 PEs): $(\mathbf{b_1+1}, a_2+d_2) = (\mathbf{12}, 5), (\mathbf{12}, 6), \ldots, (\mathbf{12}, 12)$

- Candidate CUs at dimension 2: for $d_1 = 0, 1, 2, \ldots, 6$

  Group 1 (7 PEs): $(a_1+d_1, \mathbf{a_2\text{-}1}) = (5, \mathbf{4}), (6, \mathbf{4}), \ldots, (11, \mathbf{4})$

  Group 2 (7 PEs): $(a_1+d_1, \mathbf{b_2+1}) = (5, \mathbf{13}), (6, \mathbf{13}), \ldots (11, \mathbf{13})$

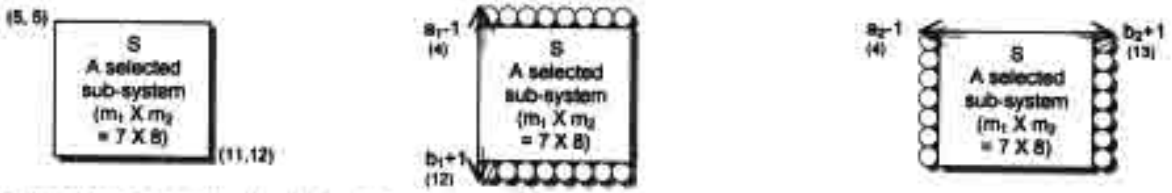


Figure 62: An example of a selected 2-D sub-system (S): all corresponding candidate CUs and their addressing.

In Figure 63, if k = 3 and the size of a selected sub-system (S) is $N' = 7 \times 8 \times 2$, addressing of all 172 candidate CUs, where $m_1 \times m_2 \times m_3 = 7 \times 8 \times 2$ at $\langle (a_1, a_2, a_3), (b_1, b_2, b_3) \rangle = \langle (5,5,5), (11,12,6) \rangle$ are

- Candidate CUs at dimension 1: $d_2 = 0, 1, 2, \ldots, 7$ and $d_3 = 0, 1$

  Group 1 (16 PEs): $(\mathbf{a_1\text{-}1}, a_2+d_2, a_3+d_3) = (\mathbf{4}, 5, 5), (\mathbf{4}, 5, 6), \ldots, (\mathbf{4}, 12, 6)$

  Group 2 (16 PEs): $(\mathbf{b_1+1}, a_2+d_2, a_3+d_3) = (\mathbf{12}, 5, 5), (\mathbf{12}, 5, 6), \ldots, (\mathbf{12}, 12, 6)$

- Candidate CUs at dimension 2: for $d_1 = 0, 1, 2, \ldots, 6$ and $d_3 = 0, 1$

  Group 1 (14 PEs): $(a_1+d_1, \mathbf{a_2\text{-}1}, a_3+d_3) = (5, \mathbf{4}, 5), (5, \mathbf{4}, 6), \ldots, (11, \mathbf{4}, 6)$

  Group 2 (14 PEs): $(a_1+d_1, \mathbf{b_2+1}, a_3+d_3) = (5, \mathbf{13}, 5), (5, \mathbf{13}, 6), \ldots, (11, \mathbf{13}, 6)$

- Candidate CUs at dimension 3: for $d_1 = 0, 1, 2, \ldots, 6$ and $d_2 = 0, 1, 2, \ldots, 7$

  Group 1 (56 PEs): $(a_1+d_1, a_2+d_2, \mathbf{a_3\text{-}1}) = (5, 5, \mathbf{4}), (5, 6, \mathbf{4}), \ldots, (11, 12, \mathbf{4})$

  Group 2 (56 PEs): $(a_1+d_1, a_2+d_2, \mathbf{b_3+1}) = (5, 5, 7), (5, 6, 7), \ldots, (11, 12, 7)$

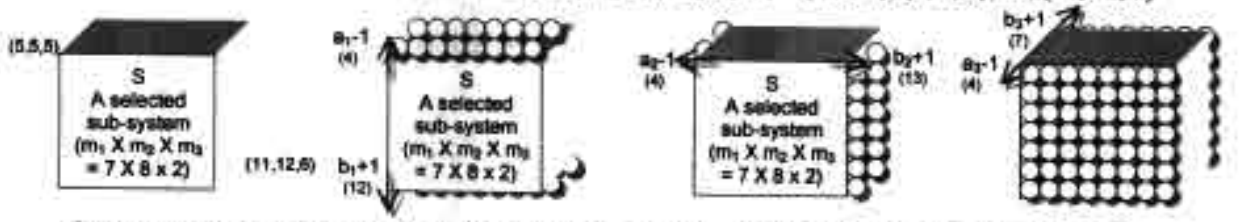


Figure 63: An example of a selected 3-D sub-system (S): all corresponding candidate CUs and their addressing.

3.4.2.1.2 The k-Sub-System CU-Scanning Strategy

Scanning for all possible CUs by using the processor-bit CU scanning strategy takes O(kN) time for any free sub-system (S), which is very time consuming. Then, the k-sub-system CU-scanning strategy is introduced to improve that time complexity. Given a selected sub-system (S) of size $N' = m_1 \times m_2 \times \ldots \times m_k$ at address $\langle(a_1, a_2, \ldots, a_k), (b_1, b_2, \ldots, b_k)\rangle$. In this case, for k dimensions there are 2k (non overlap) sub-systems of all possible candidate CUs (or CUSs), whose addressing are defined in $O(k^2)$ as follows:

> For dimension i =1, 2,..., k, addressing of two CUSs (each of size = $m_1 \times m_2 \times \ldots \times m_{i-1} \times 1 \times \ldots \times m_k$, where $m_i = 1$) are
>
> Min CUS address = $\langle(a_1, a_2, \ldots, a_i\text{-}1, \ldots, a_k), (b_1, b_2, \ldots, a_i\text{-}1, \ldots, b_k)\rangle$
> Max CUS address = $\langle(a_1, a_2, \ldots, b_i\text{+}1, \ldots, a_k), (b_1, b_2, \ldots, b_i\text{+}1, \ldots, b_k)\rangle$

For example (see Figure 62), if k = 2 and the size of a selected sub-system (S) is N'= 7 x 8, addressing of 4 CUSs (of 30 candidate CUs), where $m_1 \times m_2 = 7 \times 8$ at $\langle(a_1, a_2), (b_1, b_2)\rangle = \langle(5, 5), (11, 12)\rangle$ are

- At dimension 1:
  min CUS (8 PEs): $\langle(a_1\text{-}1, a_2), (a_1\text{-}1, b_2)\rangle = \langle(4, 5), (4, 12)\rangle$
  max CUS (8 PEs): $\langle(b_1\text{+}1, a_2), (b_1\text{+}1, b_2)\rangle = \langle(12, 5), (12, 12)\rangle$
- At dimension 2:
  min CUS (7 PEs): $\langle(a_1, a_2\text{-}1), (b_1, a_2\text{-}1)\rangle = \langle(5, 4), (11, 4)\rangle$
  max CUS (7 PEs): $\langle(a_1, b_2\text{+}1), (b_1, b_2\text{+}1)\rangle = \langle(5, 13), (11, 13)\rangle$

In another example (see Figure 63), if k = 3 and the size of a selected sub-system (S) is N" = 7 x 8 x 2, addressing of 6 CUSs (of 172 candidate CUs), where $m_1 \times m_2 \times m_3 = 7 \times 8 \times 2$ at $\langle(a_1, a_2, a_3), (b_1, b_2, b_3)\rangle = \langle(5, 5, 5), (11, 12, 6)\rangle$ are

- At dimension 1:
  min CUS (16 PEs): $\langle(a_1\text{-}1, a_2, a_3), (a_1\text{-}1, b_2, b_3)\rangle = \langle(4, 5, 5), (4, 12, 6)\rangle$
  max CUS (16 PEs): $\langle(b_1\text{+}1, a_2, a_3), (b_1\text{+}1, b_2, b_3)\rangle = \langle(12, 5, 5), (12, 12, 6)\rangle$
- At dimension 2:
  min CUS (14 PEs): $\langle(a_1, a_2\text{-}1, a_3), (b_1, a_2\text{-}1, b_3)\rangle = \langle(5, 4, 5), (11, 4, 6)\rangle$
  max CUS (14 PEs): $\langle(a_1, b_2\text{+}1, a_3), (b_1, b_2\text{+}1, b_3)\rangle = \langle(5, 13, 5), (11, 13, 6)\rangle$
- At dimension 3:
  min CUS (56 PEs): $\langle(a_1, a_2, a_3\text{-}1), (b_1, b_2, a_3\text{-}1)\rangle = \langle(5, 5, 4), (11, 12, 4)\rangle$
  max CUS (56 PEs): $\langle(a_1, a_2, b_3\text{+}1), (b_1, b_2, b_3\text{+}1)\rangle = \langle(5, 5, 7), (11, 12, 7)\rangle$

3.4.2.2 The Tree-Based CU Searching Methods

For the tree-based CU allocation approach, we introduce three best-fit heuristics to find the appropriate tree-node containing some candidate CUs for a selected sub-system in different time complexity: 1) the CU depth first search (CU-DFS) strategy ($O(N_A + kN_F + k^2)$); 2) the CU adjacent search (CU-AS) strategy ($O(k^2)$); and 3) the CU inside search (CU-IS) strategy (O(1)) for all partitioning and combining methods.

3.4.2.2.1 The CU Depth First Search (CU-DFS) Strategy

The CU depth first search (CU-DFS) strategy is used to find any free node (represented as R) in the tree that is adjacent to the selected sub-system (S). The searching starts from the root and goes to the left most (leaf) node, which is the first node R. If that node R is free, then it will check whether R is adjacent to S or not (see Definition 2), identified in O(k) time. If so, the best-fit value of that node R is computed for the candidate CU of size one processor (p = 1x1x...x1). During the DFS search, we also apply the best-fit value in Section 3.4.1 (Step 1 - 3 without task rotation) to the adjacent node only in O(k) time. Then, the new adjacent node R will be updated if it yields the better the best-fit value than the current adjacent node R in the record. The searching process is repeated for the next free node (if there exists) until all nodes in the tree are visited. So far, time complexity of the CU-DFS searching to select the best node R for the sub-system S is $O(N_A + kN_F + k^2)$ since there are N_A busy nodes (just visiting), N_F free nodes (visiting with adjacency checking), and k adjacent nodes (visiting with best-fit computing). Finally, for the best adjacent node R after complete the DFS search, first we identified all 2k CUSs of S (in $O(k^2)$ time) and select the best CUS and hence the best CU corresponding to the best adjacent node R (in $O(k^2)$ time). Therefore, total time complexity is $O(N_A + kN_F + k^2 + k^2 + k^2)$ or $O(N_A + kN_F + k^2)$. This CU-DFS strategy can be applied directly to all three partitioning and combining methods in $O(N_A + kN_F + k^2)$ time.

DEFINITION 2: Any two sub-systems (S and R) are adjacent along the dimension i if either $|a_{si} - b_{ri}| = 1$ or $|b_{si} - a_{ri}| = 1$, where $i = 1, 2, \ldots, k$; $(\alpha_s, \beta_s) = \langle(a_{s1}, a_{s2}, \ldots, a_{sk}), (b_{s1}, b_{s2}, \ldots, b_{sk})\rangle$ represents the address of S; and $(\alpha_r, \beta_r) = \langle(a_{r1}, a_{r2}, \ldots, a_{rk}), (b_{r1}, b_{r2}, \ldots, b_{rk})\rangle$ represents the address of R.

For example (see Figure 64), suppose $k = 2$ and a selected sub-system is S (of size $N' = m_1 \times m_2$ at the address $\langle(a_{s1}, a_{s2}), (b_{s1}, b_{s2})\rangle$) and a considering free node is R (of size $N'' = m'_1 \times m'_2$ at the address $\langle(a_{r1}, a_{r2}), (b_{r1}, b_{r2})\rangle$)). Figure 64.a illustrates that R and S are not adjacent since $|a_{si} - b_{ri}| \neq 1$ and $|b_{si} - a_{ri}| \neq 1$ for all $i = 1, 2$. Figure 64.b illustrates that R is adjacent to S since for dim $i = 2$, $|b_{s2} - a_{r2}| = 1$. Figure 64.c also illustrates that R is adjacent to S since for dim $i = 1$, $|a_{s1} - b_{r1}| = 1$, respectively.

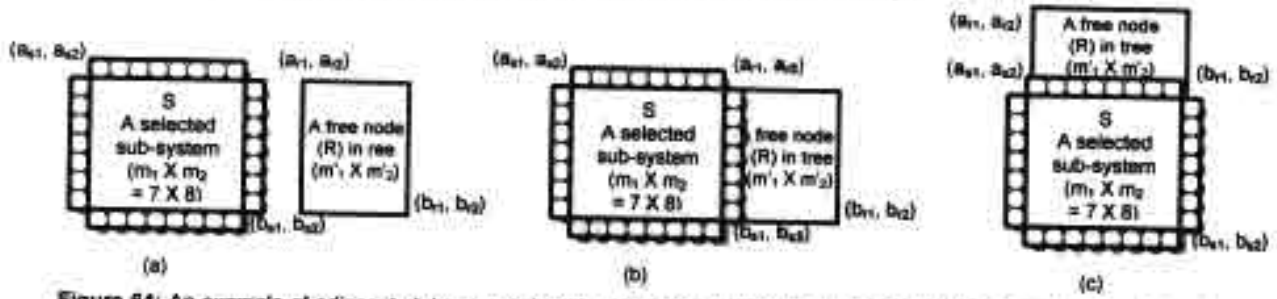


Figure 64: An example of adjacent statuses a selected sub-system S and a free node R: a) Non-adjacent status; b) and c) Two adjacent statuses.

3.4.2.2.2 The CU Adjacent Search (CU-AS) Strategy

Searching for all candidate CUs for S by using the CU-DFS strategy takes $O(N_A + kN_F + k^2)$ time for a current selected sub-system (S) and hence it is very time consuming for all candidate sub-systems (S). Then, the CU adjacent search (CU-AS) strategy is introduced to improve that time complexity.

The CU-AS strategy is introduced in order to find the free nodes R that are directly adjacent to the selected sub-system S. The searching starts from the selected S and goes to the first or next adjacent buddy node (R) in O(1) time. If that adjacent node R is free, its best-fit value (see Section 3.4.1 (only Step 1 - 3 without task rotation) is applied for the CU of size one processor ($p = 1x1x\ldots x1$) in O(k) time. Then, the new node R will be updated if it yields the better the best-fit value than the current node R in the record. This searching process is repeated for the next adjacent node for at most k nodes in $O(k^2)$ time including the best-fit computing. Note that the CU-AS strategy yields the similar system performance to that of the CU-DFS strategy (see Section 5), although it does not perform searching for all candidate CUs as the CU-DFS strategy do. Time complexity of the CU-AS strategy is only $O(k^2)$, described for each method as follows:

- For the partitioning and combining by network degree (Method 1), there are at most two adjacent buddy nodes (in the left and right) identified in O(1) time, where all processors in those two buddy nodes can be candidate CUs if they are free. Searching may need k levels for the minimum free node's size as must as possible, and hence time complexity of the CU-AS strategy for this partitioning and combining method is $O(k^2)$ including the O(k) time for best-fit computing. See the corresponding example in Figure 65.
- For the partitioning and combining by network size (Method 2), the searching starts from S and its adjacent free nodes (at most k nodes) can be identified directly (see Algorithm CU.1 and the corresponding example) in O(k) time. Searching for all k adjacent nodes of S and computing their best-fit values is $O(k^2)$ time. Therefore, time complexity of the CU-AS strategy is $O(k^2)$ time.
- For the partitioning and combining by network degree and size (Method 3), the searching starts from S. The adjacent node of S is directly identified in O(1) and its best-fit value can be computed in O(k) time since there is only one adjacent buddy (see Algorithm CU.2 and the corresponding example). Therefore, time complexity of the CU-AS is O(k) the next level searching and hence $O(k^2)$ for the k levels searching for the minimum free node's size as much as possible.

Note: In this CU-AS searching strategy, we illustrate time complexity based upon the idea of the combining and expanding, which is stored in the expanded node size (see more detail in Section 3.3 and Section 3.5).

Figure 65 illustrates the practical example of applying the CU-AS strategy for the partitioning and combining by network degree. Suppose the selected sub-system (S) is the buddy #2 at level 2 and hence its two adjacent buddy nodes are #1 and #3 (at level 2), respectively. All free nodes of the buddy #1 and 3 can be CUs for S, but the one providing best-fit value (i.e., the sub-buddy #1 at level 4) is selected.

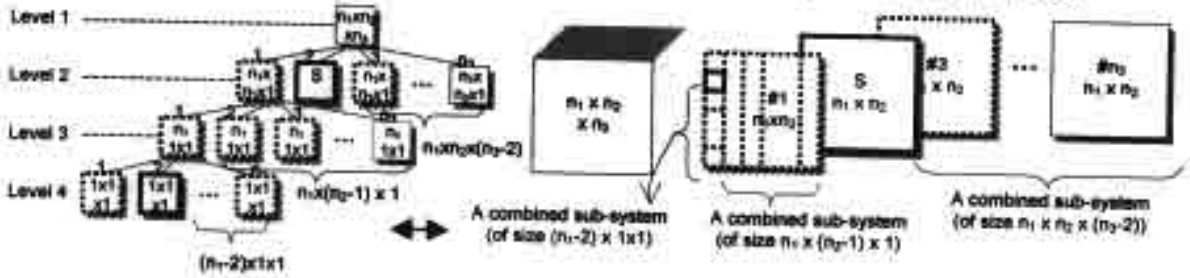
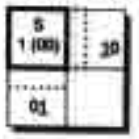


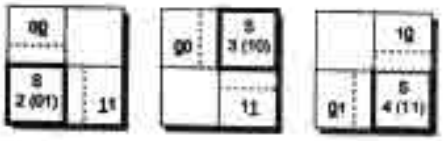
Figure 65: An example of applying the CU-AS for the partitioning and combining by network degree.

ALGORITHM CU.1: "Adjacent buddy nodes of a selected sub-system S" for the partitioning and combining by network size. There are three possible cases, concerning S as a buddy node or a combined sub-system (see Section 3.3.2):

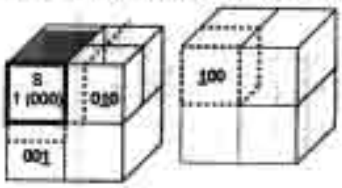
Case 1: if S is any buddy node (ID = 1, 2, ..., or 2^k in an integer format or (b_{k-1} ... b_1 b_0) in a conversion sub-system bit-map format), there are k adjacent buddy nodes and we compute the best-fit value (Step 1-3 without task rotation) of each adjacent node in O(k) time. For the same level as S, there exist k adjacent buddies, which are identified by negating b_j for one dimension at a time, where j = 1, 2, ..., k. Therefore, time complexity for finding k buddies including best-fit value is $O(k^2)$. For example, if k = 2, k adjacent buddies of any S (where ID = 1, 2, 3, or 4) are



2 adjacent buddies of S (b_1b_0 = 00) are 01 and 10.



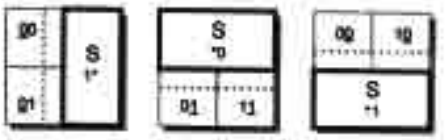
If k = 3, k adjacent buddies and k adjacent sub-buddies of an S where ID = 1 is



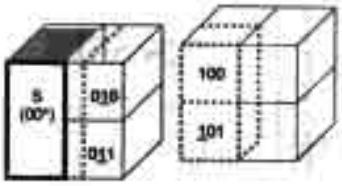
3 adjacent buddies of S ($b_2b_1b_0$ = 000) are 001, 010, and 100.

Case 2: if S is any combined node from 2^j buddies, represented in a ternary string format (t_{k-1} ... t_1 t_0) of S and stored in a special expand node size, there exist js' * and k-j bits (b_i) in that string, where j = 1, 2, ..., k-1. For each adjacent buddy, we compute the best-fit value (Step 1-3 without task rotation) in O(k) time. For the same level as S, there exist k adjacent buddies, which are identified by negating a non* (b_i) for one dimension at a time and then expanding all*s, where i = 1, 2, ..., k. Therefore, time complexity for finding k buddies including best-fit value is $O(k^2)$. For example, if k = 2, k adjacent buddies and k adjacent sub-buddies of any S (where j = 1 and a ternary string is 0*, 1*, *0, or *1) are

2 adjacent buddies of S (t_1t_0 = 0*) are 1* (10 and 11).



If k = 3, k adjacent buddies of an S where ID = 1 and either j = 1 or 2 (since only the maximum combined node at any level is stored in the expanded node) are

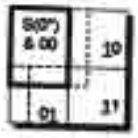


← For j = 1,
3 adjacent buddies of S ($b_2t_1t_0$ = 00*) are 01* (010, 011) and 10* (100,101).

→ or For j = 2,
3 adjacent buddies of S ($b_2t_1t_0$ = 00*) are 1** (100, 101, 110, 111).

Case 3: if S is any combined 1-buddy and 2^{k-1}-sub-buddies node, represented in a ternary string (t_{k-1} ... t_i t_0) and a binary string (b_{k-1} ... b_i b_0) of S, all adjacent buddies can be identified by applying Case 2. Note: there exists one * and k-1 bits (b_i) in that ternary string. For each adjacent buddy, we compute the best-fit value (Step 1-3 without task rotation) in O(k) time. For the same level as S, there exist 2(k-1) adjacent buddies, which are identified by negating a non* (b_i) for one dimension at a time and then expanding *, where i = 1, 2, ..., k. Hence, time complexity for finding 2(k-1) buddies including best-fit value is $O(k^2)$.

For example, if k = 2, 2(k-1) adjacent buddies of any S (where a ternary string is 0* and 00 or 01, *0 and 00 or 01, 1* and 10 or 11, or *1 and 10 or 11) are



2 (2(k-1)) adjacent buddies of S (t_1t_0=0*, b_1b_0=00) are 10, 11.

2 (2(k-1)) adjacent buddies of S (t_1t_0=*0, b_1b_0=00) are 01, 11.

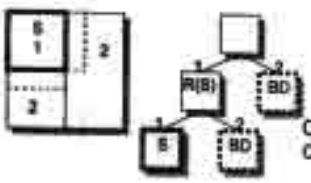
ALGORITHM CU.2: "Adjacent buddy of a selected sub-system S" for the partitioning and combining by network degree and size.

If S is any buddy node (ID = 1, 2), there is only one adjacent buddy nodes and we compute the best-fit value (Step 1-3 without task rotation) of each adjacent node in O(k) time. In order to find CUs from k sides as identified by the partitioning and combining by network size, we consider the following two cases:

Case 1: Beginning at the same level as S. For the same level as S, there exists only one adjacent buddy, which is identified by negating the ID of S (i.e. if ID of S = 1 it is 2 and if ID of S = 2 it is 1). Therefore, time complexity for finding a buddy including best-fit value is O(k).

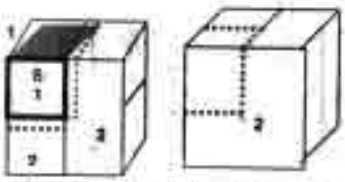
Case 2: Beginning at the upper level or the sub-sequence root(s) of S (R(S)) up to k-1 levels. In each upper level, find an adjacent buddy of the R(S), which is identified by negating the ID of the R(S). Time complexity for finding a buddy including best-fit value is O(k) and $O(k^2)$ for at most k levels

For example, if k = 2, an adjacent buddy of any S and an adjacent buddy of S's root (where ID = 1 or 2) are



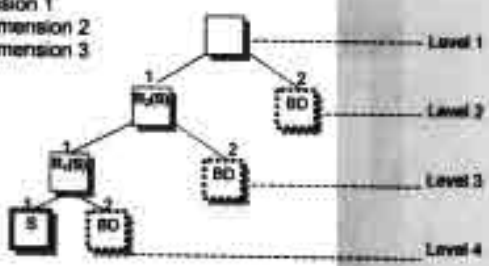
Case 1: 1 adjacent buddy of S (ID = 1) is 2.
Case 2: 1 adjacent buddy of R(S) (ID = 1) is 2.

If k = 3, an adjacent buddy of S (where ID = 1) and adjacent buddies of (k-1)-level S's roots are



Case 1: 1 adjacent buddy of S (ID = 1) is 2 at level 4
Case 2: For $R_1(S)$,
1 adjacent buddy of $R_1(S)$ (ID = 1) is 2 at level 3
For $R_2(S)$,
1 adjacent buddy of $R_2(S)$ (ID = 1) is 2 at level 2

Note:
S is partitioned along dimension 1
$R_1(S)$ is partitioned along dimension 2
$R_2(S)$ is partitioned along dimension 3



After DFS searching of both searching methods (the CU-DFS strategy and the CU-AS strategy) to visit all nodes in the tree for the SIMD task, we obtain the best sub-system S and the best node for CU, whose size may equal or larger than the requested task. The final step (applied only once) for both strategies is selecting the best-fit PE and CU after partitioning. Let's consider the following criterion:

- If the sub-system size already fits to the task and the CU node size is equal to one PE, we do not have to apply the partitioning method.
- Otherwise, we apply Algorithm A.4 for each buddy node to find the most fit one (or the like best buddy node among B nodes, where B = n_k, 2^k, or 2 for three partitioning and combining methods in $O(kn_k)$, $O(k2^k)$, O(k) time, respectively. Note that
  - After partitioning, we will have more candidate CUs inside S, where adjacent sub-buddy nodes = 2, k, or 1 for Method 1, 2, or 3, respectively. Then, we have to compute the best-fit value (Step 4 in Section 3.4.1) for each adjacent node of the best buddy (from S) in O(k), $O(k^2)$, O(k) time, for Method 1, 2, 3.
  - Then, between the best adjacent node inside S and the best adjacent node outside S, the one yields the better best-fit value is selected.
  - Finally, if the CU node size is equal to 1, we do not have to perform the partitioning. Otherwise, we partition that CU node and select the best CU for that best buddy for the sub-system (PE) for the requested tasks.

3.4.2.2.3 The CU Inside Search (CU-IS) Strategy

Searching for all possible candidate CUs by using the CU-DFS strategy takes $O(N_A + kN_P + k^2)$ time and searching for some candidate CUs by using the CU-AS needs $O(k^2)$ time. The later strategy improves time complexity over the previous one for a current sub-system (S). However, it is still time consuming for all candidate sub-systems. Then, the CU inside search (CU-IS) strategy is introduced to improve that time complexity, which is O(1) time for a current sub-system (S). In this case, the searching is similar to that for the sub-system in PE allocation, except now we are always looking for the sub-system that is larger than the requested and hence it always includes CUs inside that sub-system. Thus, we do not need extra time to search for CUs outside the sub-system, as the above two strategies. Although this strategy does not search for outside CUs as those two strategies do, they yield the comparable system performance as those of two previous strategies (see Section 5). Therefore, time complexity of the CU-IS strategy is O(1).

3.5 Searching for Allocation/Deallocation

This section integrates all computing functions (in Section 3.2 – 3.4) to form the searching for resource (CU/PE) allocation/deallocation decision. As an introduction in Section 3.1, Figure 2.2 illustrates the diagram of the dynamic tree-based resource (CU/PE) allocation/deallocation computing flow for a reconfigurable and partitionable MSIMD/MIMD parallel system.

When there is an incoming task, the dynamic resource allocation process will check in the waiting queue first. If the wait priority of the first task in the waiting queue is more than the threshold value, that task will be put in the waiting queue. Otherwise, the processor "allocation" procedure will find an appropriate free sub-system for that task by searching into the tree. If there is a free sub-system, the requested task will be allocated on the system. If no available sub-system, the request will be put in the waiting queue with FCFS scheduling. In particular, in the tree-based resource (CU/PE) allocation procedure, searching starts from the root and performing depth first search (DFS) to visit all free nodes in the tree by visiting the left most (leaf) node first. If that node is free and its size can accommodate the request, its best-fit value (see Section 3.4.1) is computed. For an SIMD task, the CU-searching strategy is also applied (see Section 3.4.2.2). 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 free leaf node in the tree. After all nodes are visited, the final process is applied, which is either to 1) allocate the best sub-system directly to the request (if its size is equal to that of the request) or 2) partition (see Section 3.2 and 3.4) to find the best sub-partition of the corresponding node for the request (if its size is larger than that of the request). Next step is applying the sub-system combining process (see Section 3.3) to recombine sub-systems by starting from the

partitioned node. Note that only the maximum combined size among all possible combinations is stored in the tree since we always keep the non-overlap sub-systems in the tree. Then the expanded node size (see an example in Figure 66) is stored in a selected node in the combined group and updated as free and other corresponding nodes are updated as busy. Finally, the maximum free size (applied in the best-fit heuristic) is updated.

Table 1 summarizes main functions of our resource (CU/PE) allocation model for the three partitioning and combining methods.

Table 1. Main functions of the universal resource (CU/PE) allocation process.

Main Functions	Method 1	Method 2	Method 3
(1) Leaf node operation (for N_f free nodes by using DFS) - Compute best-fit value for each node (for SIMD/MIMD) - Compute best-fit value for CUs of each node (for SIMD)	Section 3.4.1 Section 3.4.2	Section 3.4.1 Section 3.4.2	Section 3.4.1 Section 3.4.2
(2) Partitioning after finding the best free node - Best sub-partition - for PE (SIMD/MIMD) - for CU (SIMD) - Network partitioning - Allocate (update status of corresponding nodes)	Section 3.4.1.4 Section 3.4.2.2 Section 3.2.1	Section 3.4.1.4 Section 3.4.2.2 Section 3.2.2	Section 3.4.1.4 Section 3.4.2.2 Section 3.2.3
(3) Combining after allocation the best sub-partition - Sub-system combining (Algorithm C.2-C.3) - Expand size & update status of corresponding nodes	Section 3.3.1	Section 3.3.2	Section 3.3.3
(4) Update the maximum free size	Section 3.4.1	Section 3.4.1	Section 3.4.1

Note: Method 1 is based upon the partitioning and combining by network degree. Method 2 is based upon the partitioning and combining by network size. Method 3 is based upon the partitioning and combining by network degree and size.

When a task is finished, the processor "deallocation" procedure will find the allocated position of that finished task by using sub-set path searching into the tree. After finishing free (or deallocate) the node that stores information, the recombining process is applied to combine all corresponding (free) sub-partitions (or Buddy nodes) as soon as they become available. The recombining process starts form the new free node (of the completed task) to the root of the tree. This process will stop when there is at least one buddy of a corresponding node (along the combining path) is not available. Finally, the maximum free size (applied in the best-fit heuristic) is updated. At this time, if there are task(s) in the waiting queue, the priority FCFS scheduling will be applied to perform scheduling and allocation for these waiting tasks.

Table 2 summarizes main functions of our resource (CU/PE) deallocation model for three partitioning and combining methods.

Table 2. Main functions of the universal resource (CU/PE) deallocation process.

Main Functions	Method 1	Method 2	Method 3
(1) Sub-set-path searching to the finished node			
(2) Deallocation status - Deallocate (update status of corresponding nodes) - Unexpanded size & update status of corresponding nodes			
(3) Combining after deallocation - Sub-system combining (Algorithm C.1) up to root	Section 3.3.1	Section 3.3.2	Section 3.3.3
(4) Update the maximum free size	Section 3.4.1	Section 3.4.1	Section 3.4.1

Note that in the searching for allocation/deallocation algorithm, especially in the implementation part, we introduce the "expanded free node size" function (in step 3) for a combined sub-system, which is selected as the best free sub-system for the request. This idea is so important for the general tree-based allocation for the partitionable k-D systems in order to limit the number of nodes in the tree and provide the same methodology to update and partition as a regular (free) leaf node. For any combined sub-system, a sub-buddy node (such as the one with either the minimum buddy-ID or the largest buddy size) is selected to be the expanded node. That node is used to store the combined information (i.e., a combined size, a new base address, a free status), then other nodes in the combined sub-system are updated as busy. These busy nodes will be free whenever the expanded node is free (in step 2 of the deallocation process).

Figure 66 illustrates an example of the combining and expanding for given a 2-D mesh system with three tasks allocated (buddy#1 and #2 at level 2, buddy#1 at level 3: see Figure 66.a). Suppose that the last allocated node is the buddy 1 at level 3. Then, the expanded node processing of a combined (6x6) sub-system is illustrated in Figure 66.b. The expanded node's information (i.e. expand status, new size, new base address, old size, old base address) will be stored on a selected node (buddy#4 at level 2).

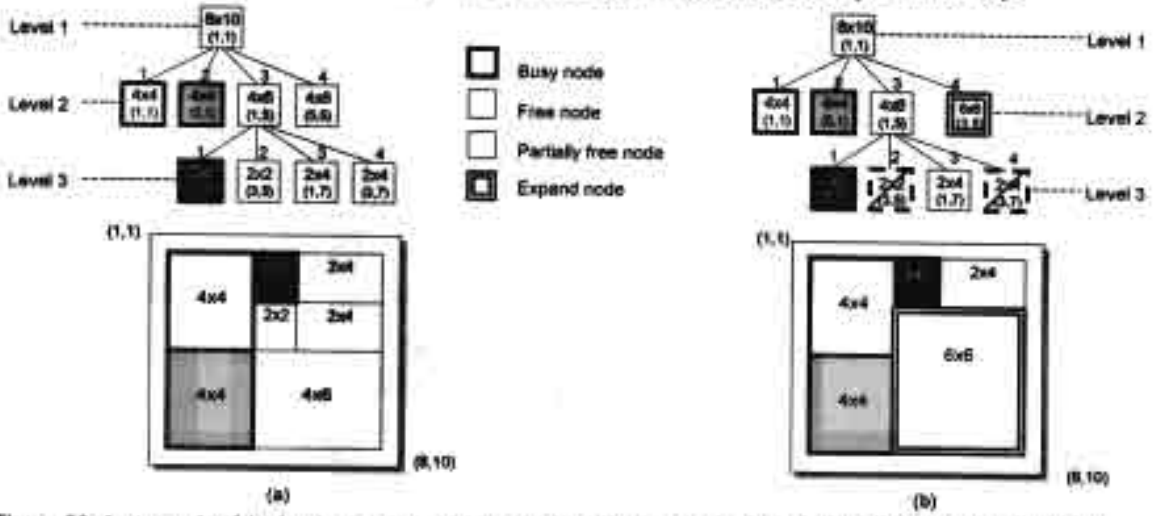


Figure 66: An example of the "node size expanding": a) the current tree with 3 task allocated and b) the corresponding tree with "side expanding".

3.6 Time Complexity of the Universal Resource (CU/PE) Allocation Model

Before we start deriving the total time complexity of the universal resource allocation/deallocation model, we show the summary of time complexity for all major processes (in Section 3.2 – Section 3.4) of the allocation model in Table 3.

Table 3. The number of buddies and time complexity of each process in the universal resource (CU/PE) allocation model.

Functions	Method 1	Method 2	Method 3
Number of Buddies	n	2^k	2
	Time Complexity		
Network Partitioning	$O(k^2n)$	$O(k2^k)$	$O(k^2)$
Sub-system Combining - in Allocation process - in Deallocation process	$O(kn^{k+1})$ $O(n)$	$O(k^2 2^{2k})$ $O(2^k)$	$O(k^4)$ $O(k)$
Best-Fit Heuristic for PE Allocation (for any SIMD / MIMD task) - for N_F external nodes	$O(k^2)$		
Best-Fit Heuristic for CU Allocation (for any SIMD task) - CU-DFS for any S - CU-AS for any S - CU-IS for any S	$O(N_A + kN_F + k^2)$ $O(k^2)$ $O(1)$		

Note: Method 1 is based upon the partitioning and combining by network degree. Method 2 is based upon the partitioning and combining by network size. Method 3 is based upon the partitioning and combining by network degree and size.

3.6.1 Time Complexity of Searching for Allocation/Deallocation for any MIMD task

Let N be the system size ($N = n_1 \times n_2 \times \ldots \times n_k$),
N_A be the maximum number of allocated tasks ($N_A \le N$),
N_F be the corresponding number of free nodes in the k-Tree ($N_A + N_F \le N$),
M be the maximum number of nodes in the k-Tree, where
M = external (leaf) nodes + internal (non-leaf) nodes < 2N) and $n = \max(n_1, n_2, \ldots, n_k)$.
$(\le (N_A + N_F) + (N_A + N_F - 1) / (b - 1)$, where b = n, 2^k, or 2 for Method 1, 2, or 3.)

Table 4 illustrates the summary of each function time complexity and total time complexity of the universal PE (or sub-system) allocation for any MIMD task.

Table 4. Time complexity of main functions of the universal resource (CU/PE) allocation process for any MIMD task.

Main Functions	Method 1	Method 2	Method 3
(1) Leaf node operation (for N_F nodes by DFS) Compute best-fit value for each node	$O(N_A + k^2N_F)$	$O(N_A + k^2N_F)$	$O(N_A + k^2N_F)$
(2) Partitioning after visit all nodes - Best sub-partition - Allocate (update status)	$O(kn)$ $O(k)$	$O(k2^k)$ $O(k)$	$O(k)$ $O(k)$
(3) Combining after allocation - Combining (Algorithm C.2-C.3) - Expand size & update status - Update the maximum free size	$O(kn^{k+1})$ $O(k)$ $O(N_A + kN_F)$	$O(k^2 2^{2k})$ $O(k)$ $O(N_A + kN_F)$	$O(k^4)$ $O(k)$ $O(N_A + kN_F)$
Total time complexity = (1) + (2) + (3)	$O(N_A + k^2N_F + kn^{k+1})$	$O(N_A + k^2N_F + k^2 2^{2k})$	$O(N_A + k^2N_F + k^4)$

Note: Method 1 is based upon the partitioning and combining by network degree. Method 2 is based upon the partitioning and combining by network size. Method 3 is based upon the partitioning and combining by network degree and size.

Table 5. Time complexity of main functions of the universal resource (CU/PE) deallocation process for any MIMD task.

Main Functions	Method 1	Method 2	Method 3
(1) Subset-path searching (from the root to the finished node)	$O(kn)$	$O(n2^k)$	$O(kn)$
(2) Deallocation status - Deallocate (update status) - Unexpanded size & update status	$O(1)$ $O(kn)$	$O(1)$ $O(k^2 2^k)$	$O(1)$ $O(k^4)$
(3) Combining after deallocation - Combining (Algorithm C.1) up to root - Update the maximum free size	$O(kn)$ $O(N_A + kN_F)$	$O(n2^k)$ $O(N_A + kN_F)$	$O(kn)$ $O(N_A + kN_F)$
Total time complexity = (1) + (2) + (3)	$O(N_A + kN_F + kn)$	$O(N_A + kN_F + n2^k)$	$O(N_A + kN_F + kn)$

In order to simplify the time complexity analysis, we assume that for a particular system network, requested tasks always require sub-systems that have the same interconnection network as that provided by the system. Therefore, we derive total time complexity for each partitioning and combining method in the following sub-section.

3.6.1.1 Time Complexity when applying the partitioning and combining by network degree

THEOREM 1: Time complexity of the tree-based allocation to find the best free sub-system for each incoming task on a product network-based systems (of size $N = n_1 \times n_2 \times \ldots \times n_k$) that partitioning and combining by network degree is $O(N_A + k^2N_F + kn^{k+1})$.

PROOF: In the allocation algorithm, a number of recursive iterations of the DFS (depth first search) are at most a number of nodes in the tree and only nodes whose sizes are larger than (or equal to) the request are visited. In this (sub-system bit-map) approach, the number of nodes in the tree is proportional to the number of busy nodes (N_A) and the number of free nodes (N_F), where $N_A + N_F \leq N$ and $N_F \leq (n-1)N_A$. Since the number of external (or leaf) nodes are at most $N_A + N_F \leq N$ and the number of internal nodes are at most (#leaf nodes-1) divided by (n-1). Therefore, the total number of nodes in the tree is at most M nodes, where $M = (N_A + N_F) + (N_A + N_F - 1) / (n-1)$. For each (free) leaf node (of N_F nodes), the best-fit value is computed in $O(k^2)$ time and hence $O(k^2N_F)$ for all leaf nodes. 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(kn) time. And the corresponding node(s) in the tree is updated in O(k) time. Next, the combining process is applied in $O(kn^{k+1})$ time. Then, the expanding size and update status takes O(k) time. Finally, the maximum free size is updated by applying the DFS in $O(N_A + kN_F)$. Thus, total time complexity to visit all nodes in the tree is approximately $O(N_A + k^2N_F + kn^{k+1})$. In addition, this method when applied to the hypercube (or k-cube) provides time complexity $O(N_A + k^2N_F + k2^{k+1})$ since $n = 2$.

THEOREM 2: Time complexity of the tree-based deallocation to free the particular tree node that stores the finished task and to combine the free buddy nodes of the root sub-tree to the root of the tree on the partitionable product network-based systems ($N = n_1 \times n_2 \times \ldots \times n_k$) that partitioning and combining by network degree is $O(N_A + kN_F + kn)$.

PROOF: In the deallocation procedure, searching for the location of a finished sub-system from the root is at most kn steps. Combining all n buddy nodes from the finished sub-system to the root (if possible) takes another n(kn) steps. Then, the resume-node-size process of the expand-node-size process (if any) may be required in O(kn) time. Finally, the maximum free size is updated by applying the DFS in $O(N_A + kN_F)$. Therefore, total time complexity of the tree-based deallocation is $O(N_A + kN_F + kn)$. Note: Our model, when applied to the hypercube (or k-cube) systems, provides time complexity $O(N_A + kN_F)$ since n = 2.

3.6.1.2 Time Complexity when applying the partitioning and combining by network size

THEOREM 3: Time complexity of the tree-based allocation to find the best free sub-system (PE) for each incoming task on a product network-based systems (of size $N = n_1 \times n_2 \times \ldots \times n_k$) that partitioning and combining by network size is $O(N_A + k^2N_F + k^2 2^k)$.

PROOF: In the allocation algorithm, a number of recursive iterations of the DFS (depth first search) are at most a number of nodes in the tree and only nodes whose sizes are larger than (or equal to) the request are visited. In this (sub-system bit-map) approach, the number of nodes in the tree is proportional to the number of allocated tasks or busy nodes (N_A) and the number of free nodes (N_F) in the 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 are at most $N_A + N_F \leq N$ and the number of internal nodes are at most (#leaf nodes-1) divided by $(2^k - 1)$. Therefore, the total number of nodes in 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 $O(k^2N_F)$ for all leaf nodes. 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 $O(k2^k)$ time. And, the corresponding node(s) in the tree is updated in O(k) time. Next, the combining process is applied in $O(k^2 2^{2k})$ time. Then, the expanding size and update status takes O(k) time. Finally, the maximum free size is updated by applying the DFS in $O(N_A + kN_F)$. Thus, total time complexity to visit all nodes in the tree is approximately $O(N_A + k^2N_F + k^2 2^k)$. In addition, this tree-based model, when applied to the 2-D/3-D mesh or 2-D/3-D torus systems, provides a linear time complexity $O(N_A + N_F)$ since k = 2 for the 2-D systems and k = 3 for the 3-D systems.

THEOREM 4: Time complexity of the tree-based deallocation to free the particular tree node that stores the finished task and to combine the free buddy nodes of the root sub-tree to the root of the tree on the partitionable product network-based systems ($N = n_1 \times n_2 \times \ldots \times n_k$) that partitioning and combining by network size is $O(N_A + kN_F + nk^2)$.

PROOF: In the deallocation procedure, searching for the location of a finished sub-system from the root is at most $n(2^k)$ steps. Combining all 2^k buddy nodes from the finished sub-system to the root (if possible) takes another $n(2^k)$ steps. Then, the resume-node-size process of the expand-node-size process (if any) may be required in $O(k^2 2^k)$ time. Finally, the maximum free size is updated by applying the DFS in $O(N_A + kN_F)$. Therefore, total time complexity of the tree-based deallocation is $O(N_A + kN_F + n2^k)$. Note: Our best-fit tree-based model, when applied to the 2-D/3-D mesh or torus systems, provides a linear time complexity $(N_A + N_F + n)$ since since k = 2 for the 2-D systems and k = 3 for the 3-D systems.

3.6.1.3 Time Complexity when applying the partitioning and combining by network degree and size

THEOREM 5: Time complexity of the tree-based allocation to find the best free sub-system (PE) for each incoming task on a product network-based systems (of size $N = n_1 \times n_2 \times \ldots \times n_k$) that partitioning and combining by network degree and size is $O(N_A + k^2N_F + k^4)$.

PROOF: In the allocation algorithm, a number of recursive iterations of the DFS (depth first search) are at most a number of nodes in the tree and only nodes whose sizes are larger than (or equal to) the request are visited. In this (sub-system bit-map) approach, the number of nodes in the tree is proportional to the number of allocated tasks or busy nodes (N_A) and the number of free nodes (N_F) in the tree, where $N_A + N_F \leq N$ and $N_F \leq N_A$. Since the number of external (or leaf) nodes are at most $N_A + N_F \leq N$ and the number of internal nodes are at most (#leaf nodes-1) divided by (2 – 1). Therefore, the total number of nodes in the tree is at most M nodes, where $M = (N_A + N_F) + (N_A + N_F - 1) / (2 - 1)$. For each (free) leaf node (of N_F

nodes), the best-fit value is computed in $O(k^2)$ time and hence $O(k^2N_F)$ for all leaf nodes. 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) time. And, the corresponding node(s) in the tree is updated in O(k) time. Next, the combining process is applied in $O(k^4)$ time. Then, the expanding size and update status takes O(k) time. Finally, the maximum free size is updated by applying the DFS in $O(N_A + kN_F)$. Thus, total time complexity to visit all nodes in the tree is approximately $O(N_A + k^2N_F + k^4)$. In addition, our tree-based model, when applied to the 2-D/3-D mesh or 2-D/3-D torus systems, provides a linear time complexity $O(N_A + N_F)$ since $k = 2$ for the 2-D systems and $k = 3$ for the 3-D systems.

THEOREM 6: Time complexity of the tree-based deallocation to free the particular tree node that stores the finished task and to combine the free buddy nodes of the root sub-tree to the root of the tree on the partitionable product network-based systems ($N = n_1 \times n_2 \times \ldots \times n_k$) that partitioning and combining by network degree and size is $O(N_A + kN_F + kn)$.

PROOF: In the deallocation procedure, searching for the location of a finished sub-system from the root is at most kn steps. Then, combining all 2 buddy nodes from the finished sub-system to the root (if it is possible) takes another kn steps. Then, the resume-node-size process of the expand-node-size process (if any) may be required in $O(k^4)$ time. Finally, the maximum free size is updated by applying the DFS in $O(N_A + k N_F)$. Therefore, total time complexity of the tree-based deallocation is $O(N_A + kN_F + kn)$. Note: Our best-fit tree-based model, when applied to the 2-D/3-D mesh or torus systems, provides a linear time complexity $O(N_A + N_F + n)$ since $k = 2$ for the 2-D systems and $k = 3$ for the 3-D systems.

3.6.2 Time Complexity of Searching for Allocation/Deallocation for any SIMD task

Time complexity of searching for allocation for any SIMD task is similar to that for the MIMD task, except we have to add the CU allocation for each node (or each selected sub-system) in both Step 1 and 2.

3.6.2.1 Time Complexity when applying the partitioning and combining by network degree

Table 6 illustrates each function time complexity and total time complexity of the universal CU/PE allocation approach for any SIMD task, based upon the partitioning and combining by network degree.

Table 6. Time complexity of main functions of the universal resource (CU/PE) allocation process (by Method 1) for any SIMD task.

Main Functions	Method 1 (partitioning and combining by network degree)		
	CU-DFS	CU-AS	CU-IS
(1) Leaf node operation (for N_F nodes)	$O(N_A+(k^2+N_A+kN_F+k^2)N_F)$	$O(N_A+(k^2+k^2)N_F)$	$O(N_A+(k^2+1)N_F)$
Compute best-fit value			
- for each free node (SIMD/MIMD)	$O(k^2)$	$O(k^2)$	$O(k^2)$
- for CUs of each selected node (SIMD)	$O(N_A + kN_F + k^2)$	$O(k^2)$	$O(1)$
(2) Partitioning after visit all nodes			
- Best sub-partition			
- for PE (SIMD/MIMD)		$O(kn)$	
- for CU (SIMD)		$O(kn)$	
- Allocate (update status)		$O(k)$	
(3) Combining after allocation			
- Combining (Algorithm C.2-C.3)		$O(kn^{k+1})$	
- Expand size & update status		$O(k)$	
- Update the maximum free size		$O(N_A + kN_F)$	
Total time complexity	$O(k^2N_F + N_AN_F + k(N_F)^2 + kn^{k+1})$	$O(N_A + k^2N_F + kn^{k+1})$	$O(N_A + k^2N_F + kn^{k+1})$

THEOREM 7: Time complexity of the tree-based allocation approach including the CU-DFS strategy to find the best free sub-system and the best CU for each incoming task on a product network-based systems that partitioning and combining by network degree is $O(k^2N_F + N_AN_F + k(N_F)^2 + kn^{k+1})$.

PROOF: Similar to Theorem 1, except that for each (free) leaf node (of N_F nodes), the best-fit value is computed in $O(k^2)$ time for finding PE (or sub-system) and $O(N_A+kN_F+k^2)$ time for finding CU by applying the CU-DFS strategy. Therefore, it takes $O(k^2 + N_A + kN_F + k^2)$ or $O(k^2 + N_A + kN_F)$ time in finding both

PE and CU for each node and hence $O(N_A + (k^2 + N_A + kN_F) N_F)$ for all leaf nodes. Thus, total time complexity to visit all nodes in the tree is approximately $O(k^2N_F + N_AN_F + k(N_F)^2 + kn^{k+1})$, where $O(kn^{k+1})$ is time complexity of the combining process by network degree.

THEOREM 8: Time complexity of the tree-based allocation approach including the CU-AS strategy to find the best free sub-system and the best CU for each incoming task on a product network-based systems that partitioning and combining by network degree is $O(N_A + k^2N_F + kn^{k+1})$. [*PROOF:* Similar to that illustrated in Theorem 7, except time complexity of the CU-AS strategy is $O(k^2)$.]

THEOREM 9: Time complexity of the tree-based allocation approach including the CU-IS strategy to find the best free sub-system and the best CU for each incoming task on a product network-based systems that partitioning and combining by network degree is $O(N_A+k^2N_F+kn^{k+1})$. [*PROOF:* Similar to that illustrated in Theorem 7, except time complexity of the CU-IS strategy is $O(1)$.]

THEOREM 10: Time complexity of the tree-based deallocation approach to free the particular tree PE node and CU node that stores the finished task and to combine the free buddy nodes of the root sub-tree to the root of the tree on the partitionable product network-based systems that partitioning and combining by network degree is $O(N_A + kN_F + kn)$. [*PROOF:* Similar to that illustrated in Theorem 2.]

3.6.2.2 Time Complexity when applying the partitioning and combining by network size

Table 7 illustrates each function time complexity and total time complexity of the universal resource (CU/PE) allocation for any SIMD task, based upon the partitioning and combining by network size.

Table 7. Time complexity of main functions of the universal resource (CU/PE) allocation process (by Method 2) for any SIMD task.

Main Functions	Method 2 (partitioning and combining by network size)		
	CU-DFS	CU-AS	CU-IS
(1) Leaf node operation (for N_F nodes) Compute best-fit value - for each free node (SIMD/MIMD) - for CUs of each selected node (SIMD)	$O(N_A+(k^2+N_A+kN_F+k^2)N_F)$ $O(k^2)$ $O(N_A + kN_F + k^2)$	$O(N_A+(k^2+k^2)N_F)$ $O(k^2)$ $O(k^2)$	$O(N_A+(k^2+1)N_F)$ $O(k^2)$ $O(1)$
(2) Partitioning after visit all nodes - Best sub-partition - for PE (SIMD/MIMD) - for CU (SIMD) - Allocate (update status)	$O(k2^k)$ $O(k2^k)$ $O(k)$		
(3) Combining after allocation - Combining (Algorithm C.2-C.3) - Expand size & update status - Update the maximum free size	$O(k^22^{2k})$ $O(k)$ $O(N_A + kN_F)$		
Total time complexity	$O(k^2N_F + N_AN_F + k(N_F)^2 + k^22^{2k})$	$O(N_A + k^2N_F + k^22^{2k})$	$O(N_A + k^2N_F + k^22^{2k})$

THEOREM 11: Time complexity of the tree-based allocation approach including the CU-DFS strategy to find the best free sub-system (PE) for each incoming task on a product network-based systems that partitioning and combining by network size is $O(k^2N_F + N_AN_F$ or $k(N_F)^2 + k^22^{2k})$.

PROOF: Similar to Theorem 3, except that for each (free) leaf node (of N_F nodes), the best-fit value is computed in $O(k^2)$ time for finding PE (or sub-system) and $O(N_A+kN_F+k^2)$ time for finding CU by applying the CU-DFS strategy. Therefore, it takes $O(k^2+N_A+kN_F+ k^2)$ or $O(k^2+N_A+kN_F)$ time in finding both PE and CU for each node and hence $O(N_A + (k^2 + N_A + kN_F) N_F)$ for all leaf nodes. Thus, total time complexity to visit all nodes in the tree is approximately $O(k^2N_F + N_AN_F + k(N_F)^2 + k^22^{2k})$, where $O(k^22^{2k})$ is time complexity of the combining process by network size.

THEOREM 12: Time complexity of the tree-based allocation approach including the CU-AS strategy to find the best free sub-system (PE) for each incoming task on a product network-based systems that partitioning and combining by network size is $O(N_A + k^2N_F + k^22^{2k})$. [*PROOF:* Similar to that illustrated in Theorem 11, except time complexity of the CU-AS strategy is $O(k^2)$.]

THEOREM 13: Time complexity of the tree-based allocation approach including the CU-IS strategy to find the best free sub-system (PE) for each incoming task on a product network-based systems that partitioning and combining by network size is $O(N_A + k^2N_F + k^2 2^{2k})$. [*PROOF:* Similar to that illustrated in Theorem 11, except time complexity of the CU-IS strategy is O(1).]

THEOREM 14: Time complexity of the tree-based deallocation approach to free the particular tree PE node and CU node that stores the finished task and to combine the free buddy nodes of the root sub-tree to the root of the tree on the partitionable product network-based systems that partitioning and combining by network size is $O(N_A + kN_F + k^2 2^{2k})$. [*PROOF:* Similar to that illustrated in Theorem 4.]

3.6.2.3 Time Complexity when applying the partitioning and combining by network degree and size

Table 8 illustrates each function time complexity and total time complexity of the universal CU/PE allocation approach for any SIMD task, based on the partitioning and combining by network degree & size.

Table 8. Time complexity of main functions of the universal resource (CU/PE) allocation process (by Method 3) for any SIMD task.

Main Functions	Method 3 (partitioning and combining by network degree and size)		
	CU-DFS	CU-AS	CU-IS
(1) Leaf node operation (for N_F nodes) Compute best-fit value - for each free node (SIMD/MIMD) - for CUs of each selected node (SIMD)	$O(N_A+(k^2+N_A+kN_F+k^2)N_F)$ $O(k^2)$ $O(N_A + kN_F + k^2)$	$O(N_A+(k^2+k^2)N_F)$ $O(k^2)$ $O(k^2)$	$O(N_A+(k^2+1)N_F)$ $O(k^2)$ $O(1)$
(2) Partitioning after visit all nodes - Best sub-partition - for PE (SIMD/MIMD) - for CU (SIMD) - Allocate (update status)	$O(k)$ $O(k)$ $O(k)$		
(3) Combining after allocation - Combining (Algorithm C.2-C.3) - Expand size & update status - Update the maximum free size	$O(k^4)$ $O(k)$ $O(N_A + kN_F)$		
Total time complexity	$O(k^2N_F + N_AN_F + k(N_F)^2 + k^4)$	$O(N_A + k^2N_F + k^4)$	$O(N_A + k^2N_F + k^4)$

THEOREM 15: Time complexity of the tree-based allocation approach including the CU-DFS strategy to find the best free sub-system (PE) for each incoming task on a product network-based systems that partitioning and combining by network degree and size is $O(k^2N_F + N_AN_F + k(N_F)^2 + k^4)$

PROOF: Similar to Theorem 5, except that for each (free) leaf node (of N_F nodes), the best-fit value is computed in $O(k^2)$ time for finding PE (or sub-system) and $O(N_A + kN_F + k^2)$ time for finding CU by applying the CU-DFS strategy. Therefore, it takes $O(k^2 + N_A + kN_F + k^2)$ or $O(k^2+N_A+kN_F)$ time in finding both PE and CU for each node and hence $O(N_A + (k^2 + N_A + kN_F) N_F)$ for all leaf nodes. Thus, total time complexity to visit all nodes in the tree is approximately $O(k^2N_F + N_AN_F + k(N_F)^2 + k^4)$, where $O(k^4)$ is time complexity of the combining process by network degree and size.

THEOREM 16: Time complexity of the tree-based allocation approach including the CU-AS strategy to find the best free sub-system (PE) for each incoming task on a product network-based systems that partitioning and combining by network degree and size is $O(N_A + k^2N_F + k^4)$. [*PROOF:* Similar to that illustrate in Theorem 15, except time complexity of the CU-AS strategy is $O(k^2)$.]

THEOREM 17: Time complexity of the tree-based allocation approach including the CU-IS strategy to find the best free sub-system (PE) for each incoming task on a product network-based systems that partitioning and combining by network degree and size is $O(N_A + k^2N_F + k^4)$. [*PROOF:* Similar to that illustrate in Theorem 15, except time complexity of the CU-IS strategy is O(1).]

THEOREM 18: Time complexity of the tree-based deallocation to free the particular tree node that stores the finished task and to combine the free buddy nodes of the root sub-tree to the root of the tree on the partitionable product network-based systems that partitioning and combining by network degree and size is $O(N_A + kN_F + kn)$ [*PROOF:* Similar to that illustrated in Theorem 6.]

4. APPLICATION OF THE UNIVERSAL CU/PE ALLOCATION MODEL

The universal tree-based resource (CU/PE) allocation model can be applied to all interconnection networks that belong to the product network class such as multi-dimensional (k-D) meshes, multi-dimensional (k-D) tori, n-ary k-cubes, hypercubes, hypercycles, etc. In this section, we show some applications of the universal resource allocation model to two popular interconnection networks, which are the 2-D mesh networks and the hypercube (or k-cube) networks.

4.1 The Universal Resource (CU/PE) Allocation Model for 2-D Meshes

In the reconfigurable MSIMD/MIMD system, there are two different modes providing for incoming tasks: SIMD (single instruction, multiple data) and MIMD (multiple instructions, multiple data). The MIMD task requires only a free sub-system (or partition), consisting of processing elements (PEs) for distributed computing of many instructions and data. The SIMD task needs both a free partition (PEs) and a control unit (CU) for parallel computing of a single instruction with multiple data. Next, in order to simplify our explanation, we present the application on the 2-D mesh for all MIMD tasks first (in Section 4.1.1) and then the application on the 2-D mesh for all SIMD tasks (in Section 4.1.2). However in practical (see Section 5), mixing modes are utilized for parallel and distributed computing in the reconfigurable and partitionable MSIMD/MIMD parallel system.

4.1.1 Sub-system (PEs) Allocation for MIMD Tasks

Suppose we have a 2-D mesh system of size 16x16 and a sequence of 5 incoming tasks (4x7, 2x2, 3x4, 8x8, and 3x3), which come in one at a time. Before we apply the universal resource (CU/PE) allocation model on this system, let's show how the product network ($G = G_1 \times G_2$) of the 2-D mesh-connected system of size $N = n_1 \times n_2$ (16x16) is constructed. Figure 67 illustrates the product network G, a product of two basic networks (or linear arrays) G_i of size $n_i = 16$, where $k = 2$ and $i = 1, 2$.

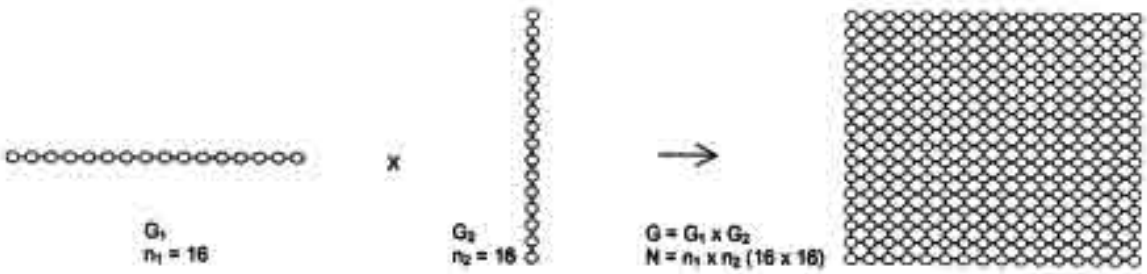


Figure 67: An example of a 2-D mesh-connected system, a product network of two linear array networks.

For the 2-D mesh-connected system ($k = 2$), the value of k is very small and hence we can apply either Method 2 (the partitioning and combining by network size) or Method 3 (the partitioning and combining by network degree and size) of the universal resource (CU/PE) allocation model.

4.1.1.1 Apply Method 2: the Partitioning and Combining by Network Size

Figure 68 illustrates the system status and the corresponding k-Tree that shows the allocation of the first incoming task (4x7). For this task, the root node (or the first node) of the k-Tree is created (starting at level 1) to store the system information (i.e., size = 16x16, based address = <1, 1>, status = 0). For the initial system, we have only one free node in the tree and hence it is the best one (when applying the best-fit heuristic (Step 1 - 3)). For the final step (Step 4), the partitioning process to select the best buddy node for allocating to the task is applied. Usually for the first task, we select the first buddy node (see Figure 68.a). After the allocation, we apply the combining process to the corresponding nodes of the current partitioning. Now, we have two possible combined sub-systems of sizes 12x16 and 16x9. Then, the expand-node size is applied to the larger size (12x16), the combining of the buddy#2 and the byddy#4 (at level 2). We store the new expand size into the free buddy#2 and mark the buddy#4 as a busy node (see Figure 68.b). Then the maximum free size is updated, which is the buddy#2 (12x16) at level 2.

Figure 69 illustrates the system status and the corresponding k-Tree that shows the allocation of the second task (2x2) and the third task (3x4), respectively. For the task (2x2), the searching starts from the root and goes to the left most free node (see Figure 68.b), which is the buddy#2 (12x16) at level 2. Then its best-fit value (Step 1-3) is computed (i.e., preserve maxFS = F, diffSF = 2, size = 192, CF = 2¼) and it is recorded as the first best node. The searching then visits the next free node, which is the buddy#3 (4x9) at level 2. The best-fit value of this node (Step 1-3) is computed (i.e., preserve maxFS = T, diffSF = 2, size = 36, CF = 3¼). Since this node (4x9) performs the better best-fit value (i.e. it can preserve the maximum free size (maxFS), the first criterion in the best-fit heuristic), it is updated as the current best node. At this time, all free nodes are visited and then Step 4 of the best-fit heuristic is applied to the best node (from Step 1-3). After the partitioning process, the best sub-partition for the second task (2x2) is the buddy#3 at level 3 (see Figure 69.a). Finally after the combining process of the current partitioning, there are two possible combined sub-systems of sizes 4x7 and 2x9. Then, the expand-node size is applied to the larger size (4x7), the combining of the buddy#1 and the byddy#2 (at level 3). We store the new expand size into the free buddy#1 and mark the buddy#2 as a busy node. Next since the maximum free size (12x16), the buddy#2 at level 2, is not partitioned, it is not necessary to be updated at this time.

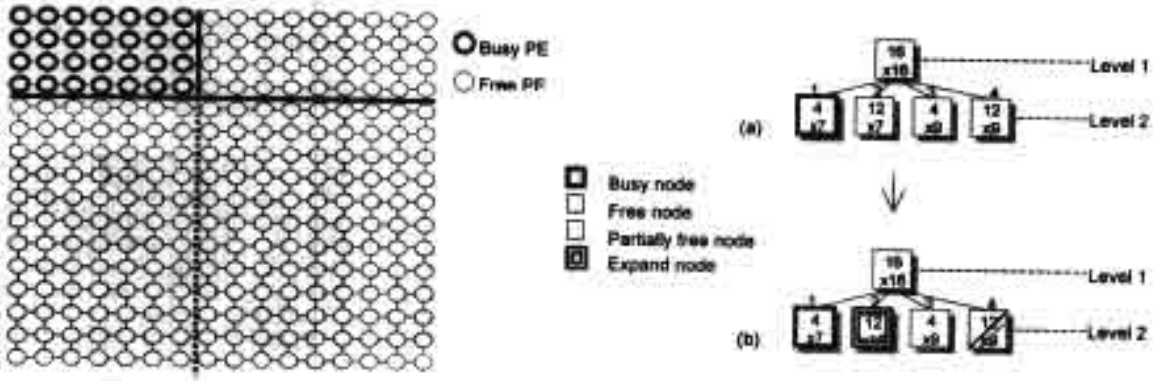


Figure 68: The system status and the corresponding k-Tree of the allocation of the first task (4x7), based on the partitioning and combining by network size, for a 2-D mesh.

For the third task (3x4), the first free node (see Figure 69.a) is the buddy#2 (12x16) at level 2 with computed best-fit value (i.e., preserve maxFS = F, diffSF = 2, size = 192, CF = 2¼). Then, it is recorded as the first best node. The next free node is the buddy#1 (4x7) at level 3 with the computed best-fit value (i.e., preserve maxFS = T, diffSF = 1 (for the rotated size 4x3), size = 28, CF = 5½) and hence it is updated as the new best node. Next, there is the last free node (the buddy#4 at level 3) to visit but it size (2x2) can not accommodate to the request (3x4). Now, all free nodes are visited and then Step 4 is applied to the best node (from Step 1-3) in order to select the best sub-partition. After the partitioning process, the buddy#1 at level 4 is selected for the task (3x4) with the rotated size 4x3 (see Figure 69.b) since it yields the better best-fit value. Finally for the maximum free size (12x16), in this case it is not necessary to be updated.

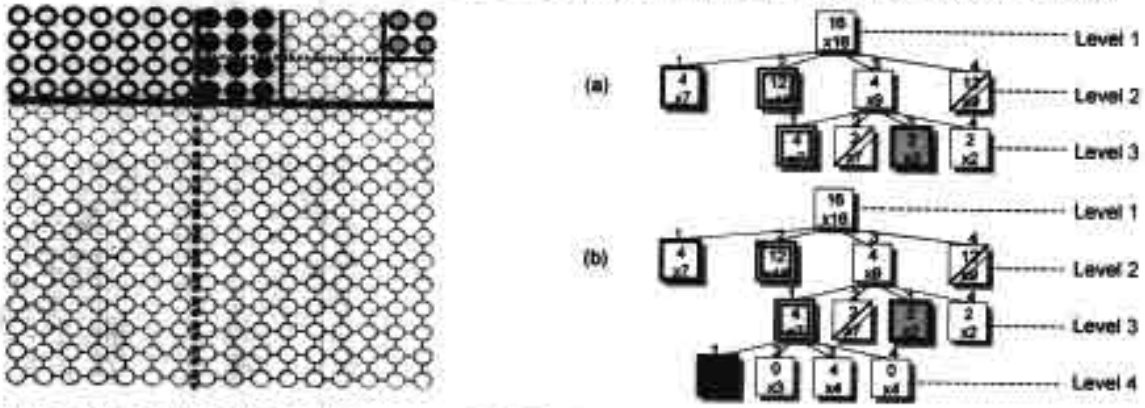


Figure 69: The system status and the corresponding k-Tree of the allocation of a) the second task (2x2) and b) the third task (3x4), based on the partitioning and combining by network size, for a 2-D mesh.

Figure 70 illustrates the system status and the corresponding k-Tree of the allocation of the fourth task (8x8) and the fifth task (3x3), respectively. For the task (8x8), the first free node (see Figure 69.b) is the buddy#2 (12x16) at level 2 with the computed best-fit value (i.e., preserve maxFS = F, diffSF = 2, size = 192, CF = 2¼) and it is recorded as the first best node. Next, two sub-sequence free nodes are the buddy#3 (4x4) at level 4 and the buddy#4 (2x2) at level 3 but their sizes can not accommodate to the request (8x8). So far, all free nodes are visited and then Step 4 of the best-fit heuristic is applied to partition the best node (from Step 1-3), in order to select the best sub-partition (the buddy#2 (8x8) at level 3) for the fourth task (see Figure 70.a). After combining the corresponding nodes of the current partitioning, we have two possible combined sub-systems (4x16 and 12x8). The larger size (12x8), the combining of the buddy#3 and the byddy#4 (at level 3), is selected. We store the expand size in the free buddy#3 and mark the buddy#4 as a busy node. Finally, since the buddy#2 at level 2, the maximum free size (12x16), is partitioned, we have to compute the new maximum free size. At beginning, we select the new expanded node (12x8), the buddy#3 at level 3, as the temporary maximum free size. Then, we have to update the maximum free size by performing DFS to visit all free nodes in the tree if the larger node exists.

For the last incoming task (3x3), the first free node (see Figure 70.a) is the buddy#1 (4x8) at level 3 with the computed best-fit value (i.e., preserve maxFS = T, diffSF = 2, size = 32, CF = 5) and it is recorded as the first best node. The next free node is the buddy#3 (12x8) at level 3, but it cannot preserve the maximum free size, the first best-fit criterion. Then, the searching goes to the next free node, the buddy#3 (4x4) at level 4. Its best-fit value is computed (i.e., preserve maxFS = T, diffSF = 2, size = 16, CF = 5½) and updated as the current best node since it performs the better best-fit value, according to the criterion 3 (the smaller size). The last free node is the buddy#4 at level 3, but its size (2x2) can not accommodate to the request (3x3). Now all free nodes are visited and then Step 4 of the best-fit heuristic is applied to the best node (from Step 1-3) in order to select the best sub-partition. After the partitioning process, the buddy#3 at level 5 is selected for the fifth task (see Figure 70.b). After the combining process of the current partitioning, we have two possible combined sub-systems (4x1 and 1x4). The expand-node size is applied to the first size (4x1), the combining of the buddy#1 and the byddy#2 (at level 5). The new expand size is stored into the free buddy#1 and the buddy#2 is marked as a busy node. For the maximum free size computing, since the current maximum free size (12x8), is not partitioned, it is not necessary to be updated.

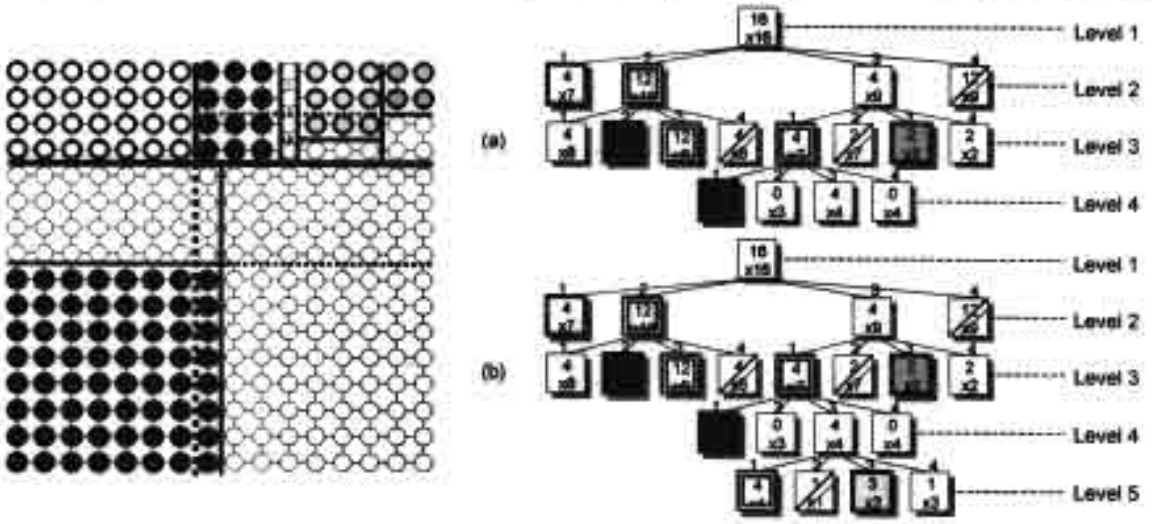


Figure 70: The system status and the corresponding k-Tree of the allocation of a) the fourth task (8x8) and b) the fifth task (3x3), based on the partitioning and combining by network size, for a 2-D mesh.

4.1.1.2 Apply Method 3: the Partitioning and Combining by Network Degree and Size

In order to see the compared results to the previous application, we use the same 2-D mesh system and incoming tasks, defined in Section 4.1.1.1. Figure 71 illustrates the system status and the corresponding binary-Tree that illustrates the allocation of the first incoming task (4x7). For this task, the root of the tree is created (at level 1) to store the system information (i.e., size = 16x16, based address = <1, 1>, status = 0). Initially, we have only one free node, which is the best one (when applying the best-fit heuristic (Step 1-3)).

The final step (Step 4), the partitioning process to select the best buddy node for allocating to the task, is applied. Usually for the first task, we select the first buddy node. Finally, the maximum free size is updated, which is the buddy#2 (12x16) at level 2.

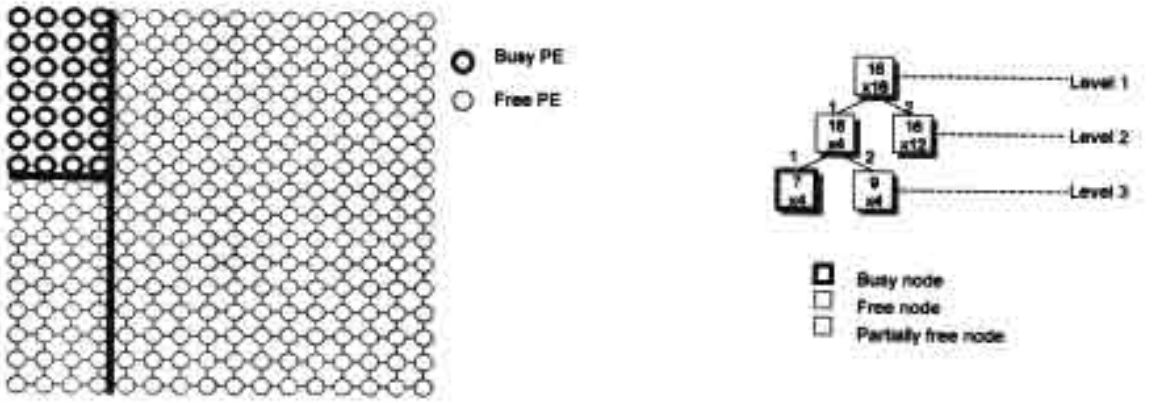


Figure 71: The system status and the corresponding binary-Tree of the allocation of the first task (4x7), based on the partitioning and combining by network degree and size, for a 2-D mesh.

Figure 72 illustrates the system status and the corresponding binary-Tree that shows the allocation of the second task (2x2) and the third task (3x4), respectively. For the task (2x2), the searching starts from the root and the first free node is the buddy#2 (9x4) at level 3 (see Figure 71). Then its best-fit value is computed (i.e., preserve maxFS = T, diffSF = 2, size = 36, CF = 3¼) and it is recorded as the first best node. The next free node is the buddy#2 (16x12) at level 2. According to the best-fit criterion 1, it cannot preserve the maximum free size, which is not better than the current best-fit node. Then, searching goes to the next free node but now all free nodes are visited. Then, Step 4 of the best-fit heuristic is applied to the best node (from Step 1-3). After applying the partitioning process twice, the best sub-partition for the second task (2x2) is the buddy#2 (2x4) at level 4, partitioned along the 1th dimension, and then the buddy#1 (2x2) at level, partitioned alone the 2nd dimension (see Figure 72.a). Next since the maximum free size (16x12), the buddy#2 at level 2, is not partitioned, it is not necessary to be updated at this time.

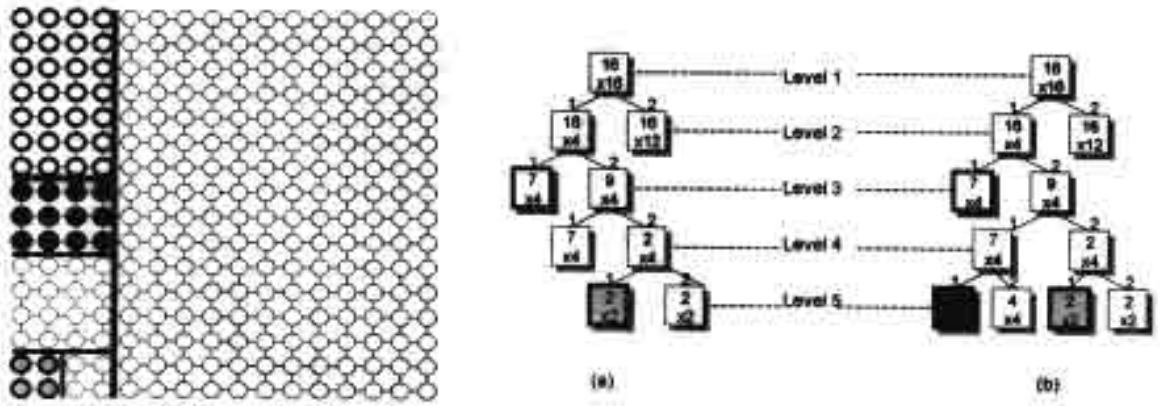


Figure 72: The system status and the corresponding binary-Tree of the allocation of a) the second task (2x2) and b) the third task (3x4), based on the partitioning and combining by network degree and size, for a 2-D mesh.

For the third task (3x4), the first free node (see Figure 72.a) is the buddy#1 (7x4) at level 4 with computed best-fit value (i.e., preserve maxFS = T, diffSF = 1, size = 28, CF = 3¼) and it is the first best node. The next free node is the buddy#2 (2x2) at level 5 but it size (2x2) can not accommodate to the request (3x4). The last free node is the buddy#2 (16x12) at level 2 but it cannot preserve the maximum free size, the criterion 1 of the best-fit heuristic. Then the partitioning process (Step 4) is applied to the best node (from Step 1-3) in order to select the best sub-partition, the buddy#1 (3x4) at level 5 for the fifth task (see Figure 72.b). Finally for the maximum free size (16x12), in this case it is not necessary to be updated.

Figure 73 illustrates the system status and the corresponding binary-Tree of the allocation of the fourth task (8x8) and the fifth task (3x3). For the task (8x8), the first free node (see Figure 72.b) is the buddy#2 (4x4) at level 5 but its size is less than the task's size. The next free node is the buddy#2 (2x2) at level 5 but its size is too small for the task. The last free node is the buddy#2 (16x12) at level 2 with the computed best-fit value (i.e., preserve maxFS = F, diffSF = 2, size = 192, CF = 2¼) and it is recorded as the first free node. So far, all free nodes are visited and then Step 4 of the best-fit heuristic is applied to partition the best node (from Step 1-3), in order to select the best sub-partition for the fourth task (see Figure 73.a). In this case, the partitioning process is applied twice. According to the criterion 4 of the best-fit heuristic, we select the buddy#1 (8x12) at level 3, partitioned along the 1st dimension, and then the buddy#2 (8x8) at level 4, partitioned along the 2nd dimension. Finally, since the buddy#2 at level 2, the maximum free size (16x12), is partitioned, we have to compute the new maximum free size. At beginning, we select the new maximum node (the buddy#2 (8x12) at level 3) after the current partitioning. Then, we update the maximum free size by performing DFS to visit all free nodes in the tree if the larger node exists.

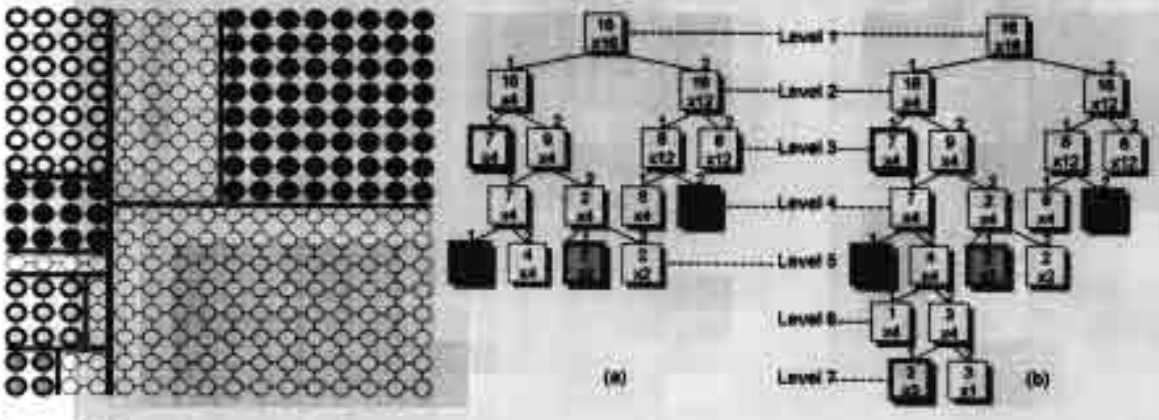


Figure 73: The system status and the corresponding binary-Tree of the allocation of a) the fourth task (8x8) and b) the fifth task (3x4), based on the partitioning and combining by network degree and size, for a 2-D mesh.

For the last incoming task (3x3), the first free node (see Figure 73.a) is the buddy#2 (4x4) at level 5 with the computed best-fit value (i.e., preserve maxFS = T, diffSF = 2, size = 16, CF = 3¾) and it is recorded as the first best node. The next free node is the buddy#2 (2x2) at level 5, but it cannot accommodate to the task (3x3). Then, the searching goes to the next free node, the buddy#1 (8x4) at level 4 with the computed best-fit value (preserve maxFS = T, diffSF = 2, size = 32). This node does not perform the better best-fit value, according to the criterion 3 of the best-fit heuristic. The last free node is the buddy#2 (8x12) at level 3. However, it cannot preserve the maximum free size. Now all free nodes are visited and then Step 4 of the best-fit heuristic is applied to the best node (from Step 1-3) in order to select the best sub-partition. The partitioning process is applied twice. According to the criterion 4 of the best-fit heuristic, we select the buddy#2 (3x4) at level 6, partitioned along the 1st dimension and then the buddy#1 (3x3) at level 7, partitioned along the 2nd dimension for the fifth task (see Figure 73.b). In this case, we do not have to update the maximum free size since the current maximum free size (8x12) is not partitioned.

4.1.2 Sub-System (PEs) and Control Unit (CU) Allocation for SIMD tasks

Suppose we have a 2-D mesh system of size 16x16 and a sequence of 5 incoming SIMD tasks (4x7, 2x2, 3x4, 8x8, and 3x3), which come in one at a time. For the 2-D mesh system (k = 2), the value of k is very small and thus we can apply either Method 2 (the partitioning and combining by network size) or Method 3 (the partitioning and combining by network degree and size) of the universal CU/PE allocation model.

4.1.2.1 Apply Method 2: the Partitioning and Combining by Network Size

Figure 74 illustrates the system status and the corresponding k-Tree that shows the allocation of the first incoming task (4x7). The sub-system (or PEs) allocation is similar to the allocation illustrated in previous section (see Figure 68) for the MIMD task. For the SIMD task in this section, we add the allocation for the

corresponding CU. Usually for the first task, we select the first buddy node for the PE allocation and perform another partitioning process to the smallest adjacent sub-partition for the CU allocation. After the CU/PE allocation, we apply the combining process to the corresponding nodes of the partitioning for PEs and the partitioning for CU. At this time, the maximum free size is the buddy#2 (12x16) at level 2.

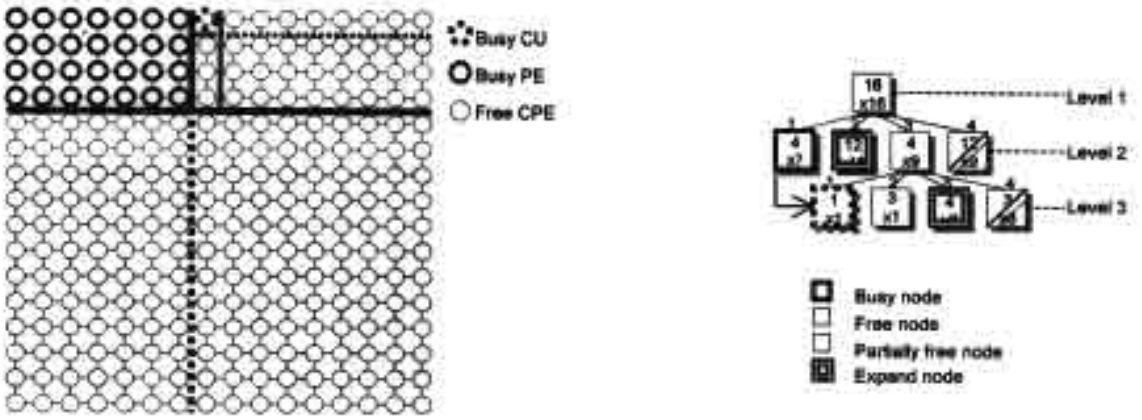


Figure 74: The system status and the corresponding k-Tree of the allocation of the first SIMD task (4x7), based on the partitioning and combining by network size, for a 2-D mesh.

Figure 75 illustrates the system status and the corresponding k-Tree that shows the allocation of the second task (2x2). The sub-system (or PEs) allocation is similar to the allocation, illustrated in previous section (Figure 69.a) for the MIMD task. For the SIMD task, we have to apply the CU searching (i.e., the CU-DFS, CU-AS, or CU-IS strategy) to find the adjacent node (containing some CUs) for each visiting node. For example, the last free node, the buddy#3 (4x8) at level 3 (see Figure 74) is updated as the best node since it yields the better best-fit value (i.e., preserve maxFS = T, diffSF = 2, size = 32, CF = 4½). For CU searching of this node, the best adjacent node is the buddy#2 at level 3 if we apply the CU-DFS or CU-AS strategy. If we apply the CU-IS strategy, we do not have to find adjacent node because the corresponding CU can be inside the best sub-system, the buddy#3 at level 3. After all free nodes are visited, Step 4 of the best-fit heuristic (the partitioning process) is applied to the best node for PEs and its adjacent node for CU (from Step 1-3). After the partitioning process, the best sub-partition for the second task (2x2) is the buddy#1 at level 4 and the corresponding CU is selected from the outside node (from applying the CU-DFS or CU-AS), the buddy#2 (3x1) at level 2.

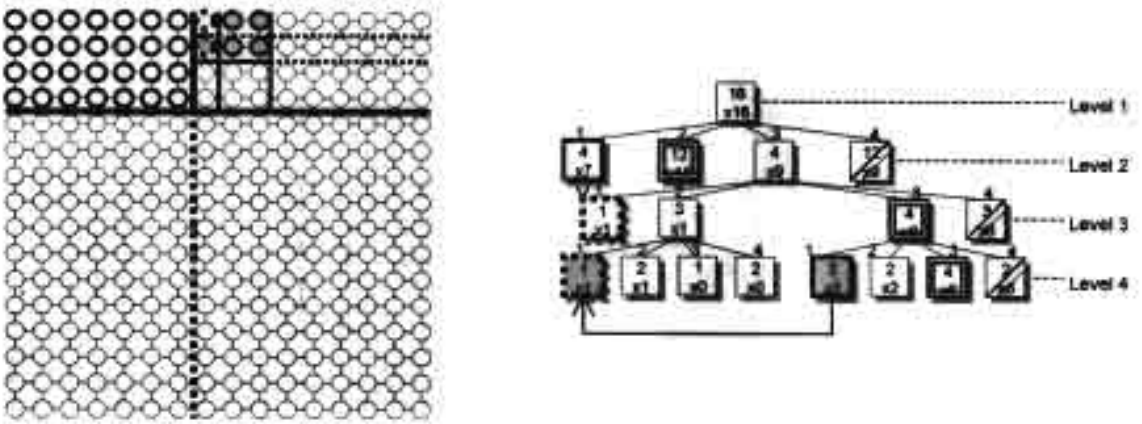


Figure 75: The system status and the corresponding k-Tree of the allocation of the second SIMD task (2x2), based on the partitioning and combining by network size, for a 2-D mesh.

Figure 76 illustrates the system status and the corresponding k-Tree that shows the allocation of the third task (3x4), the fourth task (8x8), and the fifth task (3x3), respectively. The searching for the best free node for the sub-system (PEs) allocation and the corresponding CU for the CU allocation for each of these tasks is similar to that applied for the second task.

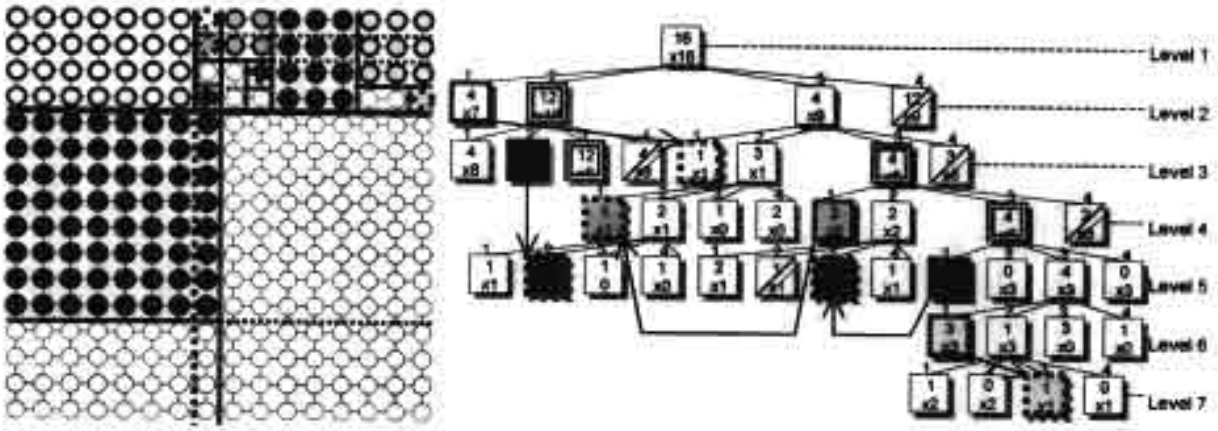


Figure 76: The system status and the corresponding k-Tree of the allocation of the 3rd, 4th, and 5th SIMD tasks (3x4, 8x8, 3x3), based on the partitioning and combining by network size, for a 2-D mesh.

4.1.2.2 Apply Method 3: the Partitioning and Combining by Network Degree and Size

Figure 77 illustrates the system status and the corresponding binary-Tree that shows the allocation of the first incoming task (4x7). The sub-system (or PEs) allocation is similar to the allocation illustrated in previous section (Figure 71) for the MIMD task. Then, for the SIMD task we have to add the allocation for the corresponding CU. Usually for the first task, we select the first buddy node for the PE allocation and perform another partitioning process to the smallest adjacent sub-partition for the CU allocation. After the CU/PE allocation, we apply the combining process to the corresponding nodes of the partitioning for PEs and the partitioning for CU. The first maximum free size is the buddy#2 (12x16) at level 2.

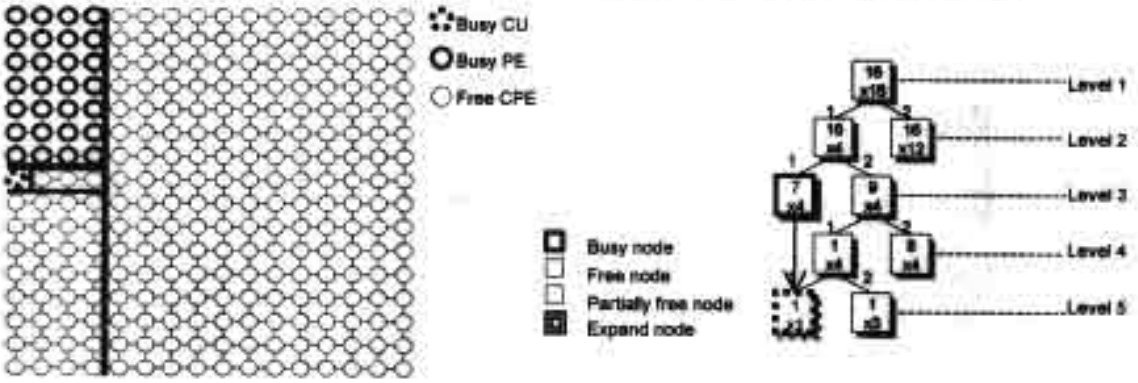


Figure 77: The system status and the corresponding k-Tree of the allocation of the first SIMD task (4x7), based on the partitioning and combining by network degree and size, for a 2-D mesh.

Figure 78 illustrates the system status and the corresponding binary-Tree that shows the allocation of the second task (2x2). The sub-system (or PEs) allocation is similar to the allocation illustrated in previous section (see Figure 72.a) for the MIMD task. For the SIMD task we have to apply the CU searching (i.e., the CU-DFS, CU-AS, or CU-IS strategy) to find the adjacent node (containing some CUs) for each visiting node. After visiting all free nodes, the best node (from Step 1-3) is the buddy#2 (8x4) at level 4 and its adjacent node is the buddy#2 (1x3) at level 5. After the partitioning process, the best sub-partition for the second task (2x2) is the buddy#1 at level 6 and the corresponding CU is selected from the outside node (from applying the CU-DFS or CU-AS strategy), the buddy#2 (1x3) at level 5.

Figure 79 illustrates the system status and the corresponding k-Tree that shows the allocation of the third task (3x4), the fourth task (8x8), and the fifth task (3x3), respectively. The searching for the best free node for the sub-system (PEs) allocation and the corresponding CU for the CU allocation for each of these tasks is similar to that applied for the second task.

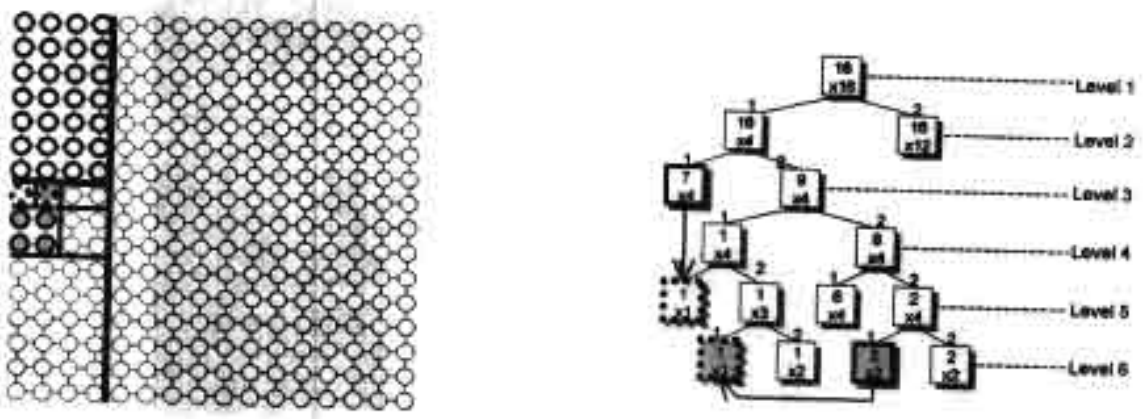


Figure 78: The system status and the corresponding k-Tree of the allocation of the second SIMD task (2x2), based on the partitioning and combining by network degree and size, for a 2-D mesh.

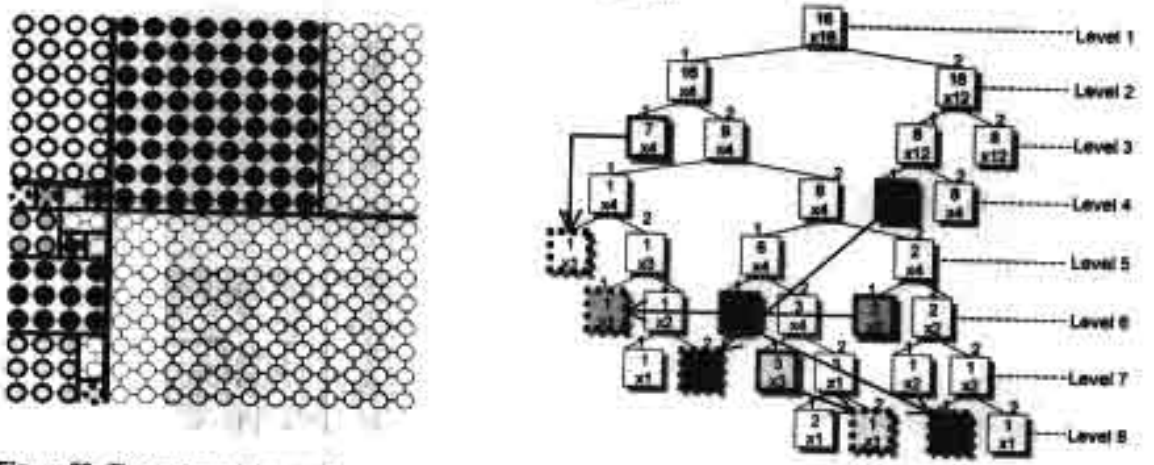


Figure 79: The system status and the corresponding k-Tree of the allocation of the 3rd, 4th, and 5th SIMD tasks (3x4, 8x8, 3x3), based on the partitioning and combining by network degree and size, for a 2-D mesh.

4.2 The Universal Resource (CU/PE) Allocation Model for Hypercubes

For the reconfigurable MSIMD/MIMD system with the hypercube network, two different modes providing for incoming tasks: SIMD (single instruction, multiple data) and MIMD (multiple instructions, multiple data). The MIMD task requires only a free sub-system (or partition), consisting of processing elements (PEs) for distributed computing of many instructions and data. The SIMD task needs both a free partition (PEs) and a control unit (CU) for parallel computing of a single instruction with multiple data. Next, in order to simplify our explanation, we present the application on the hypercube for all MIMD tasks first (in Section 4.2.1) and then the application on the hypercube for all SIMD tasks (in Section 4.2.2). However in practical (see Section 5), mixing modes are allowed for parallel and distributed computing in the reconfigurable and partitionable MSIMD/MIMD parallel system.

4.2.1 Sub-system (PEs) Allocation for MIMD Tasks

Suppose we have a hypercube (or 5-cube) system of size $N = 2^5$ and a sequence of three incoming tasks (3-cube, 4-cube, 2-cube), coming in one at a time. Before we apply the universal resource (CU/PE) allocation model on this system, let's show how the product network ($G = G_1xG_2xG_3xG_4xG_5$) of the 5-cube-connected system of size $N = n_1xn_2xn_3xn_4xn_5$ (2^5) is constructed.

Figure 80 illustrates the product network G, a product of five basic networks (or linear arrays) G_i of size $n_i = 2$, where $k = 5$ and $i = 1, 2, 3, 4, 5$. For the hypercube-connected system, the value of n is 2 and hence we apply the partitioning and combining by network degree (Method 1).

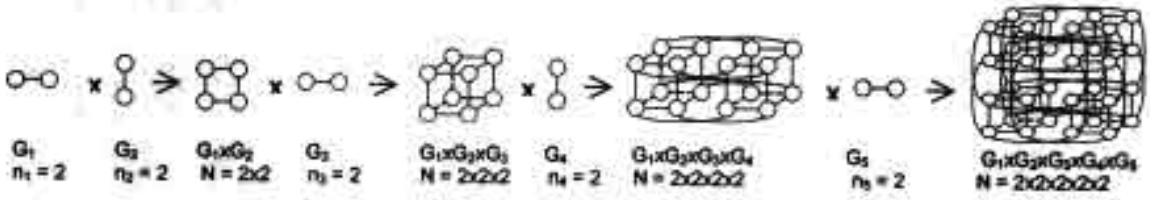


Figure 80: An example of a hypercube (5-cube)-connected system, a product network of five linear array networks.

Figure 81 illustrates the system status and the corresponding binary tree that shows the allocation of the first incoming task (3-cube). For this task, the root node (or the first node) of the tree is created (starting at level 1) to store the system information (i.e., size = 2^5, based address = <1,1,1,1,1>, status = 0). For the initial system, we have only one free node in the tree and hence it is the best one (when applying the best-fit heuristic (Step 1 - 3)). For the final step (Step 4), the partitioning process is applied (twice for this task size) to select the best buddy node for allocating to the task. Usually for the first task, we always select the first buddy node. After the allocation, we will apply the combining process (i.e., combining for a 4-cube) to the corresponding nodes of the sub-sequence partitioning if the buddy of its root is partially free (or some of its buddy nodes are busy). In this case, we do not have to apply the combining process since the buddy#2 (4-cube) is free. Then the first maximum free size is the buddy#2 (4-cube) at level 2.

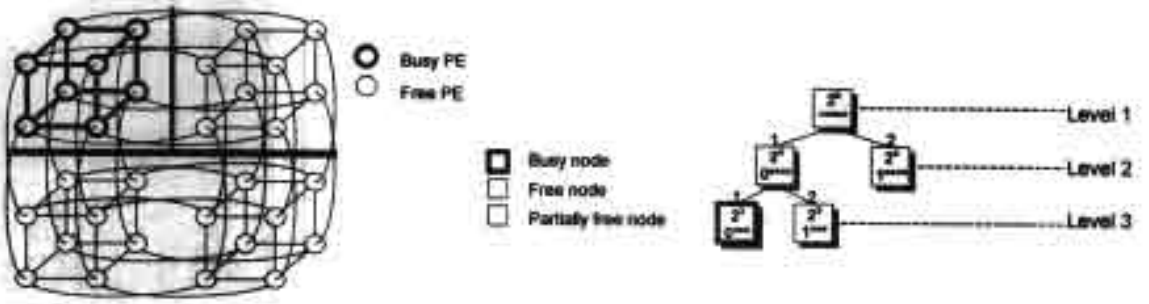


Figure 81: The system status and the corresponding binary-Tree of the allocation of the first task (3-cube), based on the partitioning and combining by network degree, for a hypercube.

Figure 82 illustrates the system status and the corresponding binary tree that shows the allocation of the second task (4-cube) and the third task (2-cube). For the task (4-cube), the searching starts from the root and goes to the left most free node (see Figure 81), which is the buddy#2 (3-cube) at level 3 but its size cannot accommodate to the task. The searching then visits the next free node, which is the buddy#2 (4-cube) at level 2. The best-fit value of this node (Step 1-3) is computed (i.e., preserve maxFS = F, diffSF = 0, size = 16, CF = ½) and it is recorded as the first best node. At this time, all free nodes are visited and then Step 4 of the best-fit heuristic is applied to the best node (from Step 1-3) if its size is larger than the task. In this case, we do not apply the partitioning process since the node's size is equal to the task's size (4-cube) and hence no need for the combining process (see Figure 82.a). Next since the maximum free size (4-cube), the buddy#2 at level 2, is allocated, we have to find the new maximum free size by using DFS to visit all free nodes in the tree. Now, the maximum free size is the buddy#2 (3-cube) at level 3.

For the third task (2-cube), the searching starts from the root and goes to the left most free node (see Figure 82.a), which is the buddy#2 (3-cube) at level 3. The best-fit value of this node (Step 1-3) is computed (i.e., preserve maxFS = F, diffSF = 0, size = 8, CF = ½) and it is recorded as the first best node. The searching is continuing to the next free node. So far, all free nodes are visited and then Step 4 of the best-fit heuristic is applied to the best node (from Step 1-3), the buddy#2 (3-cube) at level 3. After applying the partitioning process once, we allocate the buddy#1 (at level 4) for the third task (2-cube). After the allocation, we will apply the combining process (i.e., combining for a 4-cube) to the corresponding nodes of the sub-sequence partitioning if the buddy of its root is partially free (or some of its buddy nodes are busy). In this case, we do not have to apply the combining process since the buddy#2 (4-cube) is busy. Next since the maximum free size (3-cube), the buddy#2 at level 3, is partitioned, we have to find the new maximum free size. At beginning, we set the temporary maximum free size to the buddy#2 (2-cube) at level 4 and then using DFS to visit all free nodes in the tree to update if the larger node exists.

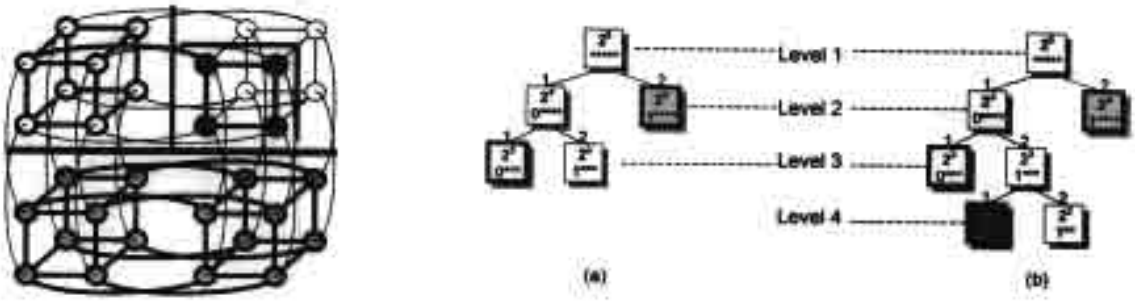


Figure 82: The system status and the corresponding binary-Tree of the allocation of (a) the second task (4-cube) and (b) the third task (2-cube), based on the partitioning and combining by network degree, for a hypercube.

4.2.2 Sub-system (PEs) Allocation and Control Unit (CU) for SIMD Tasks

For the application for SIMD tasks, suppose we have a hypercube (or 5-cube) system of size $N = 2^5$ and a sequence of three incoming SIMD tasks (3-cube, 4-cube, and 2-cube), which come in one at a time. For the hypercube-connected system, the value of n is 2 and thus we apply Method 1 (the partitioning and combining by network degree).

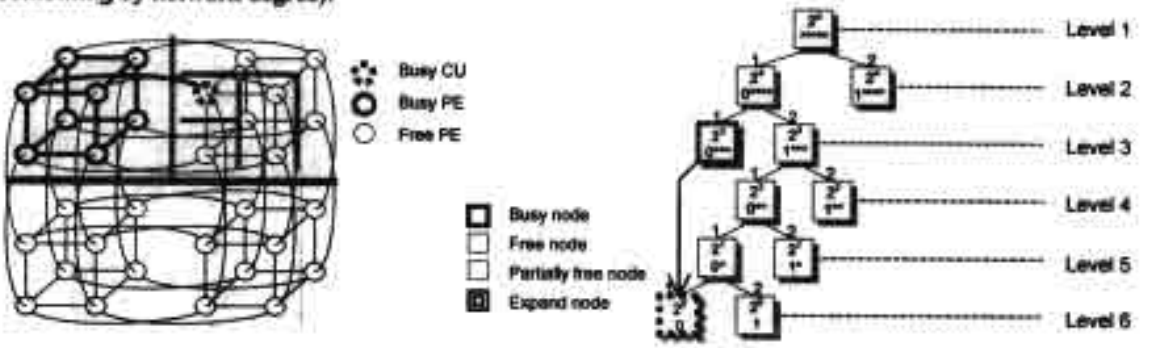


Figure 83: The system status and the corresponding binary-Tree of the allocation of the first SIMD task (3-cube), based on the partitioning and combining by network degree, for a hypercube.

Figure 83 illustrates the system status and the corresponding binary tree that shows the allocation of the first incoming SIMD task (3-cube). For this task, the root node of the tree is created (starting at level 1) to store the system information (i.e., size = 2^5, based address = <1,1,1,1,1>, status = 0). Initially, we have only one free node in the tree and hence it is the best one (from applying the best-fit heuristic (Step 1 - 3)). For the final step (Step 4), the partitioning process is applied (twice for the 3-cube) and for the first task, we always select the first buddy node (at level 3) for the sub-system (PEs) allocation. Then we perform another partition process on its buddy, the buddy#2 (at level 3), for the CU allocation. After the allocation, we will apply the combining process to the corresponding nodes of the sub-sequence partitioning if the buddy of its root is partially free (or some of its buddy nodes are busy). In this case, we do not have to apply the combining process since the buddy#2 is free. Finally, the first maximum free size is the buddy#2 (4-cube) at level 2.

Figure 84 illustrates the system status and the corresponding binary tree that shows the allocation of the second SIMD task (4-cube) and the third SIMD task (2-cube), respectively. For the task (4-cube), the searching starts from the root and goes to first free node (see Figure 83), which is the buddy#2 (1 PE) at level 6 but its size cannot accommodate to the task. The searching then visits the next free nodes, which are the buddy#2 (1-cube) at level 5, the buddy#2 (2-cube) at level 4, the buddy#2 (3-Cube) at level 5 but their sizes cannot accommodate to the task (4-cube). Then, the last free node is the buddy#2 at level 2. The best-fit value of this node (Step 1-3) is computed (i.e., preserve maxFS = F, diffSF = 0, size = 32, CF = ½) and it is recorded as the first best node. For CU searching for this node, the best adjacent node is the buddy#2 at level 6 if we apply the CU-DFS or the CU-AS strategy. After all free nodes are visited, then Step 4 of the best-fit heuristic is applied to the best node (from Step 1-3) if its size is larger than the task.

In this case, we do not apply the partitioning process since the node's size is equal to the task's size (4-cube) and hence no need for the combining process (see Figure 84.a). Next since the maximum free size (4-cube), the buddy#2 at level 2, is allocated, we have to find the new maximum free size by using DFS to visit all free nodes in the tree. Now, the maximum free size is the buddy#2 (2-cube) at level 4. For the third task (2-cube), the searching for the best node for the sub-system (PEs) allocation and the corresponding CU for the CU allocation is similar to that applied for the second task (see Figure 84.b).

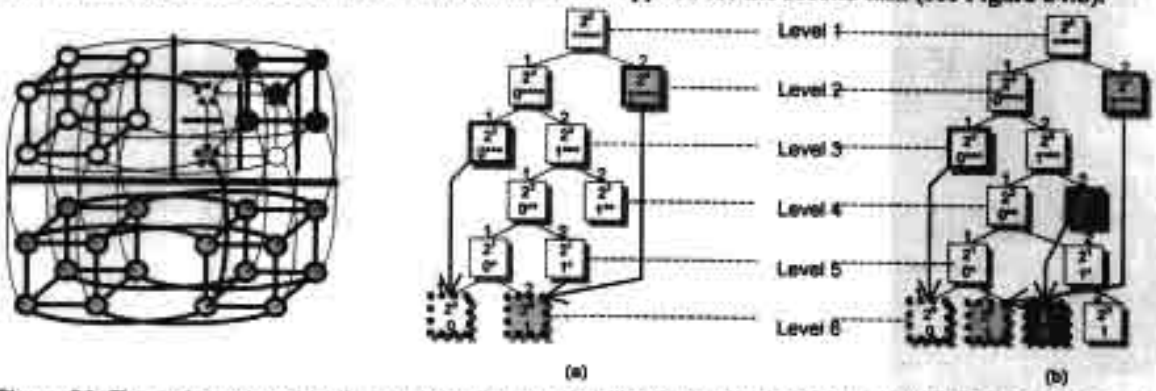


Figure 84: The system status and the corresponding binary-Tree of the allocation of a) the second SIMD task (4-cube) and b) the third SIMD task (2-cube), based on the partitioning and combining by network degree, for a hypercube.

5. PERFORMANCE EVALUATION

In order to evaluate the system performance, the universal tree-based resource (CU/PE) allocation approach was developed and applied for two network applications, which are the 2-D meshes and the 3-D meshes. These two interconnection networks are efficient for the reconfigurable and partitionable systems since they provide small node degrees (or links), low cost per node, and hence low system cost of links. Therefore, in the partitioning process we have not to cut many links in order to form a partition. For example, there are three values of the node degrees of the 2-D meshes: 1) node degree = 4 for each internal node, 2) node degree = 3 for each border node, and 3) node degree = 2 for each corner node (see Figure 85.a). For the 3-D meshes, there are four values of the node degrees, which are 6, 5, 4, and 3 for each inner node, each side node, each border node, and each corner node, respectively (see Figure 85.b).

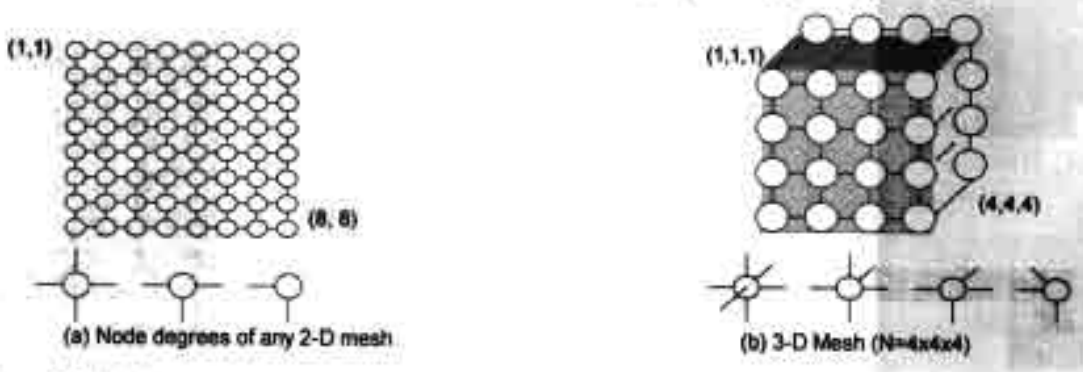


Figure 85: Node degrees: a) at most 4 for the 2-D meshes and b) at most 6 for the 3-D meshes.

By simulation study, a number of experiments are performed to investigate the effect of applying the "universal tree-based resource (CU/PE) allocation model" for performing processor allocation/deallocation for those reconfigurable and partitionable mesh-connected systems. The system performance is investigated in terms of system utilization, system fragmentation, etc. Note that the system utilization (U_{sys}) is measured as a summation of the busy processors (allocated for tasks) over the number of processors in the system, computed when the system reaches the steady state. Similarly, the system fragmentation (F_{sys}) is measured as a summation of the free processors over the total number of processor in the system, also computed when the system reaches the steady state, where $F_{sys} = 1 - U_{sys}$ (if there is no internal fragmentation.)

For those 2-D and 3-D mesh-connected systems (where k = 2 and 3) the value of k (or the number of dimensions) is very small and hence we can apply either Method 2 (the partitioning and combining by network size) and sometime later is called "the modified k-Tree-based resource allocation strategy", or Method 3 (the partitioning and combining by network degree and size), called "the b inary-tree-based resource allocation strategy". In particular, for the reconfigurable and partitionable 2-D and 3-D mesh-connected systems (for both MIMD and SIMD tasks), we introduces the comparison results between the modified k-Tree-based resource allocation strategy and the binary-tree-based resource allocation strategy. Also for the partitionable 2-D mesh-connected systems (for only MIMD tasks) we presents the comparison results of our universal tree-based resource (PE) allocation model (when applied to the 2-D meshes (in $O(N_A + N_F)$ time)) to recently 2-D mesh-based strategies, which are the BUSY LIST strategy ($O(N_s^3)$) [14], the FREE SUB-LIST strategy ($O(N_f^3)$) [26], and the QUICK ALLOCATION strategy ($O(N_s\sqrt{N})$) [50]. When considering time complexity, our tree-based approach perform the processor allocation decision in linear time (of the number of allocated tasks (N_A) and corresponding free nodes (N_F)), which is efficient, compared to those ($O(N_s^3)$, $O(N_f^3)$, and , $O(N_s\sqrt{N})$ of the above existing methods, where $N_A \approx N_s$ (see system performance results) and $N_F < N_f$ (since our model stores only non-overlap free nodes while N_f includes overlap free sub-systems in the free lists.)

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 universal resource (CU/PE) allocation model are represented for the static system performance (with concerning processor allocation for incoming tasks (or jobs) only (or assumed that no task finishes during the considering time)) and the 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 ($N = n_1 \times n_2$) are varied and the task sizes [1x1 - $n_1 \times n_2$] are generated, where $\alpha = 1$, $\beta = \max(n_1, n_2)$ for the Uniform distribution $U(\alpha, \beta)$ and $\mu = \sigma = \max(n_1, n_2)/2$ for the Normal distribution $N(\mu, \sigma)$. Other parameters are fixed such as task arrival rate ~ Poisson (λ) (or inter-arrival time ~ Exp($1/\lambda$=5)), and service time ~ Exp(μ=10), etc.

5.1 System Performance Evaluation for the Partitionable 2-D Meshes (only MIMD Tasks)

In order to set the same incoming tasks and environment to all allocation strategies for the comparison purpose, we assumed that no task finishes during the considering time. Therefore, in this study N_A represents the maximum number of allocated tasks in the system and N_F represents the corresponding free nodes in the tree.

5.1.1 Investigate the Effect of System Sizes to the System Performance

In the first experiment, we investigated "the effect of system sizes (N) to the system utilization (U_{sys}) and the system fragmentation (F_{sys})" for the 2-D meshes, executing only MIMD tasks. In this experiment, the system sizes ($N = n_1 x n_2$) were varied and the task sizes (1x1 - $n_1 x n_2$) were generated and fixed by using the Uniform distribution (see Table 9 and Table 11) or the Normal distribution (see Table 10 and Table 12)).

Table 9: Effect of "the system sizes" to the system utilization (%) for the Uniform distribution.

System Sizes ($N = n_1 x n_2$)	Binary-Tree	k-Tree	Free Sub-List	Busy List	Quick Allocation
64x64	60.75	60.97	56.06	57.27	55.64
128x128	61.63	60.11	56.98	56.67	57.84
256x256	60.30	59.77	55.77	54.61	56.87
512x512	59.36	58.86	56.11	55.91	56.53

Table 9 illustrates the results of the system utilization for the Uniform distribution (see also Figure 86). For all test cases our binary-tree strategy performed approximately 60 - 61% system utilization which was comparable to those of the k-tree strategy and was improved over those of the recently 2-D mesh-based strategies, which were at most 57% system utilization.)

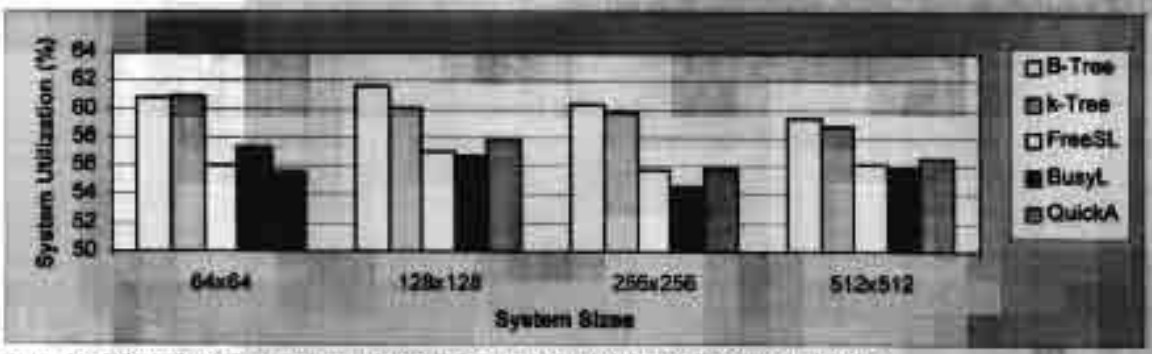


Figure 86: Effect of "the system sizes" to the system utilization (%) for the Uniform distribution.

Table 10 illustrates the results of the system utilization for the Normal distribution (see also Figure 87). For the system size N = 64x64, the binary-tree, the k-tree, and the free sub-list strategies yielded the comparable results (~61% system utilization) and were improved over those (~58% system utilization) of the busy list and the quick allocation strategies.

For the system size N = 512x512, our binary-tree strategy performed 62.2% system utilization which was improved over that (61.3%) of the k-tree strategy and those (57.0%, 57.2%, and 55.9%) of the recently 2-D mesh-based strategies (the free sub-list, the busy list, and the quick allocation strategies.)

For other test cases (N = 128x128 and 256x256), our binary-tree strategy performed approximately 59 - 60% system utilization, which was comparable to those of the k-tree strategy and improved over those of the recently 2-D mesh-based strategies, which were at most 58% system utilization.)

Table 10: Effect of "the system sizes" to the system utilization (%) for the Normal distribution.

System Sizes ($N = n_1 x n_2$)	Binary-Tree	k-Tree	Free Sub-List	Busy List	Quick Allocation
64x64	61.05	61.18	61.42	58.46	57.53
128x128	59.34	59.16	56.89	58.02	54.79
256x256	59.41	60.10	58.02	55.12	55.09
512x512	62.20	61.28	57.04	57.22	55.89

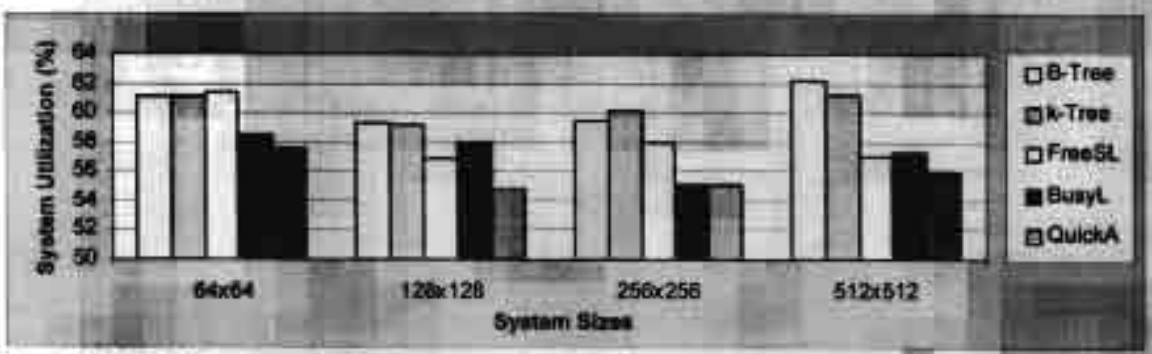


Figure 87: Effect of "the system sizes" to the system utilization (%) for the Normal distribution.

Table 11 and Figure 88 illustrate the results of the system fragmentation for the Uniform distribution. For all test cases our binary-tree strategy performed approximately 39 - 40% system fragmentation which was comparable to those of the k-tree strategy and was improved over those of the recently 2-D mesh-based strategies, which were approximately 42 - 44% system fragmentation.

Table 11: Effect of "the system sizes" to the system fragmentation (%) for the Uniform distribution.

System Sizes ($N = n_1 x n_2$)	Binary-Tree	k-Tree	Free Sub-List	Busy List	Quick Allocation
64x64	39.25	39.03	43.94	42.73	44.36
128x128	38.37	39.89	43.02	43.33	42.16
256x256	39.70	40.23	44.23	45.39	44.13
512x512	40.64	41.14	43.89	44.09	43.47

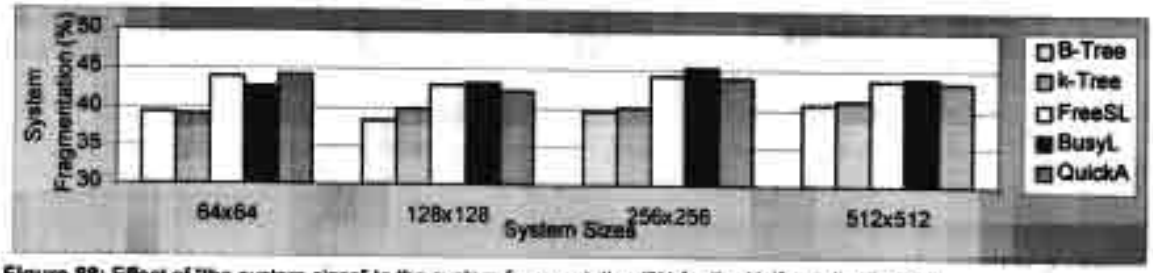


Figure 88: Effect of "the system sizes" to the system fragmentation (%) for the Uniform distribution.

Table 12 and Figure 89 illustrate the results of the system fragmentation for the Normal distribution. For N = 64x64, the binary-tree strategy performed approximately 38.9% system fragmentation, which was comparable to those (38.8% and 38.6%) of the k-tree and the free sub-list strategies and was improved over those (41.5% and 42.5%) of the busy list and the quick allocation strategies. For other test cases (N = 128x128 and 256x256), our binary-tree strategy performed approximately 40 - 41% system fragmentation, which was comparable to those of the k-tree strategy and improved over those (42 - 45%) of the recently 2-D mesh-based strategies (the free sub-list, the busy list, and the quick allocation strategies.)

Table 12: Effect of "the system sizes" to the system fragmentation (%) for the Normal distribution.

System Sizes ($N = n_1 x n_2$)	Binary-Tree	k-Tree	Free Sub-List	Busy List	Quick Allocation
64x64	38.95	38.82	38.58	41.54	42.47
128x128	40.66	40.84	43.11	41.98	45.21
256x256	40.59	39.90	41.98	44.86	44.91
512x512	37.80	38.72	42.96	42.78	44.11

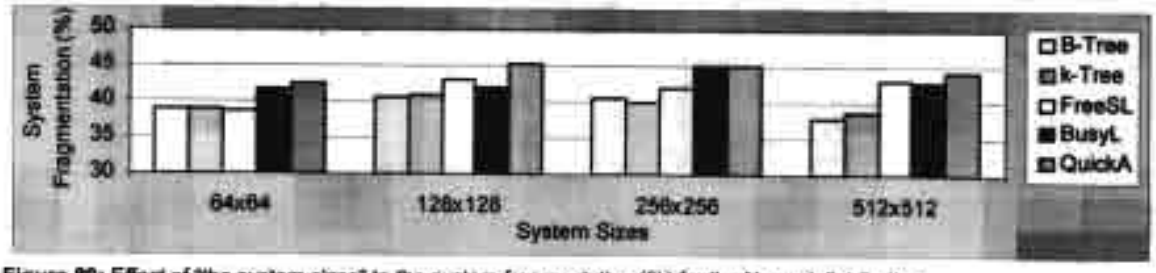


Figure 89: Effect of "the system sizes" to the system fragmentation (%) for the Normal distribution.

In summary, the varying the system sizes (and the generating the task sizes in $1x1 - n_1 \times n_2$) do not effect to the system performance (i.e., U_{sys} and F_{sys}) for all allocation methods since each strategy tends to yield the results, effected by the method itself for all system sizes. In addition, the effect of task sizes generated by using the Uniform or Normal distributions tends to be the same. Therefore, in the next investigation, we will show the results of the Uniform distribution only.

5.1.2 Investigate the Effect of Task Sizes to the System Performance

In the second experiment, we investigated "the effect of task sizes to the system utilization (see Table 13) and the system fragmentation (see Table 14)". In this experiment, the system size was fixed ($N = n_1 \times n_2 = 512x512$) and the task sizes were generated and varied (by using the Uniform distribution) in various ranges (i.e., the "large" range ($1x1 - n_1 \times n_2$), the "medium" range ($1x1 - {}^{n1}/_2 \times {}^{n2}/_2$), and the "small" range

(1x1 - $^{n1}/_4$ x $^{n2}/_4$)). Table 13 and Figure 90 illustrate the results of the system utilization. For the small range of task sizes (1x1 - 128x128 or 1x1 - $^{n1}/_4$ x $^{n2}/_4$), the binary-tree, the free sub-list, and the busy list allocation strategies performed the comparable system utilization, which were 81.1%, 82.6%, and 82.6%, respectively and were improved over those (79.1%, 80.1%) of the k-tree and the quick allocation strategies. For the medium range of task sizes (1x1 - 256x256 or 1x1 - $^{n1}/_2$ x $^{n2}/_2$), the top three strategies that performs the best system utilization were are the busy list (73.4%), free sub-list (71.1%), and the binary-tree (69.2%), which were improved over those (68.3% and 66.3%) of the k-tree and quick allocation strategies. For the large range of task sizes (1x1 - 512x512 or 1x1 - n_1 x n_2), the binary-tree and k-tree strategies yielded the best comparable system utilization (59.3% and 58.6%), which were improved over those (56.11%, 55.9%, and 56.5%) of the free sub-list, the busy list, and the quick allocation strategies, respectively.

Table 13: Effect of "the task sizes" to the system utilization (%) for the Uniform distribution.

Task Sizes (1x1 - rxc)	Binary-Tree	k-Tree	Free Sub-List	Busy List	Quick Allocation
1x1 - 128x128	81.08	79.12	82.57	82.62	80.15
1x1 - 256x256	69.20	66.29	71.11	73.38	66.31
1x1 - 512x512	59.36	58.86	58.11	55.91	56.53

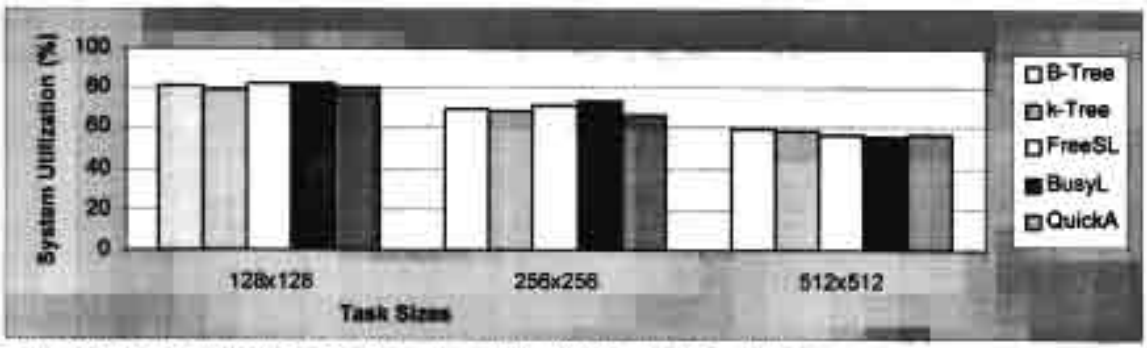


Figure 90: Effect of "the task sizes" to the system utilization (%) for the Uniform distribution.

Table 14 and Figure 91 illustrate the results of the system fragmentation, the tree-based approach and those existing 2-D mesh-based strategies also tended to perform the same effecting results similar to the system utilization since $U_{sys} = 1\text{-}F_{sys}$ (or since there was no internal system fragmentation).

Table 14: Effect of "the task sizes" to the system fragmentation (%) for the Uniform distribution.

Task Sizes (1x1 - rxc)	Binary-Tree	k-Tree	Free Sub-List	Busy List	Quick Allocation
1x1 - 128x128	18.92	20.88	17.43	17.38	19.85
1x1 - 256x256	30.80	31.71	28.89	26.62	33.69
1x1 – 512x512	40.64	41.14	43.89	44.09	43.37

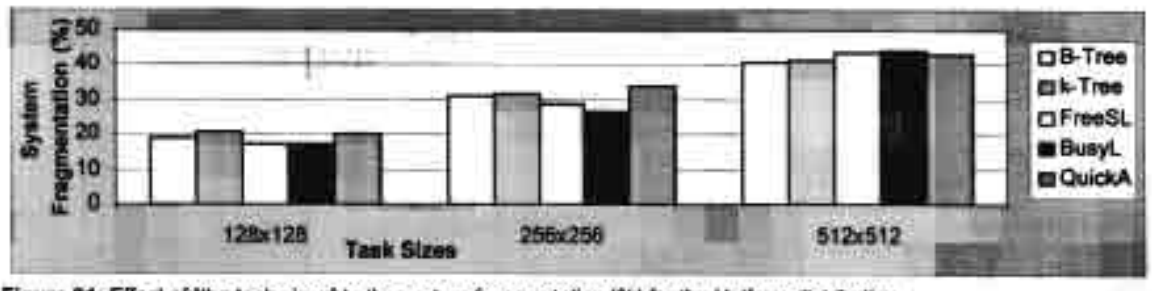


Figure 91: Effect of "the task sizes" to the system fragmentation (%) for the Uniform distribution.

In summary, the varying of task sizes on the fixed system size (N = n_1 x n_2) is effecting to the system performance (i.e., U_{sys} and F_{sys}). When generating range of task sizes increases (small - large), the system utilization increases while the system fragmentation decreases. For the small and medium ranges, the top ranking are the busy list, the free sub-list, and the binary-tree strategies respectively, whereas for the large range the top ranking are the binary-tree, the k-tree, and the busy list strategies, respectively.

5.2 System Performance Evaluation for the Reconfigurable and Partitionable 2-D Meshes and 3-D Meshes (both MIMD / SIMD Tasks)

In order to set the same incoming tasks and environment to the CU allocation strategies (the CU-DFS and CU-AS) for the comparison purpose, we assumed that no task finishes during the considering time.

In the last experiment, we investigated the effect of system sizes (N) to the system utilization (U_{sys}), where the system sizes ($N = n_1 \times n_2$) were varied and the task sizes ($1x1 - {}^{n1}/_2 \times {}^{n2}/_2$) were generated and fixed. For all test cases the CU-AS and CU-DFS strategies performed the same system utilization. The reason is that the system performance results of these two methods were different only when sub-system (S) and task (T) sizes were equal which rarely occurred.

Table 15 and Figure 92 illustrate the results of the system utilization of the modified k-tree strategy for the 2-D and 3-D mesh-connected systems when the percentage of the SIMD tasks were increased (such as 0%, 10%, 20%). The results showed that increasing the percentage of SIMD tasks did not effect the system utilization for the 2-D and the 3-D meshes, except when N = 64x64.

Table 15. Effect of the "system sizes" to the system utilization (%) for the 2-D and 3-D Meshes.

2-D Mesh				3-D Mesh			
% of SIMD Tasks mixed with MIMD Tasks							
System Sizes ($N = n_1 x n_2$)	0%	10%	20%	System Sizes ($N = n_1 x n_2 x n_3$)	0%	10%	20%
64x64	68.76	68.46	71.91	32x32x32	58.14	58.55	58.60
128x128	68.25	67.82	67.62	64x64x64	49.73	54.32	54.33
256x256	70.16	70.16	70.16	128x128x128	50.52	50.52	50.52

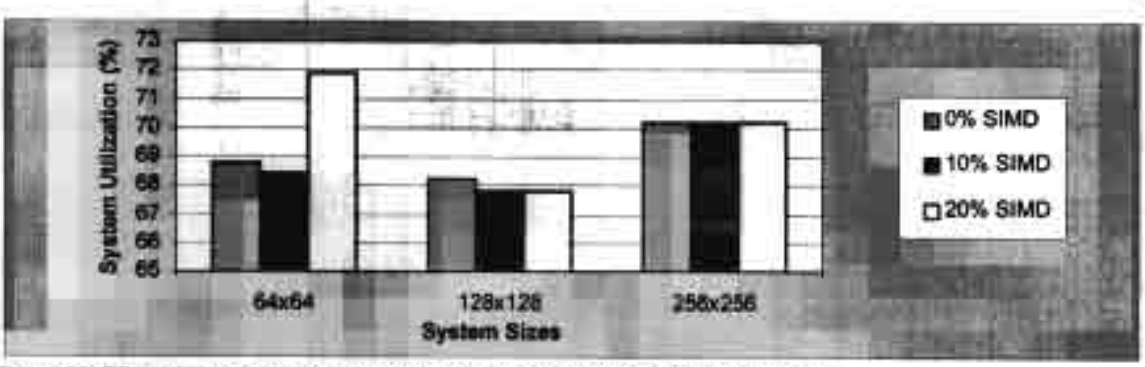


Figure 92: Effect of "the system sizes" to the system utilization (%): a) for the 2-D Meshes.

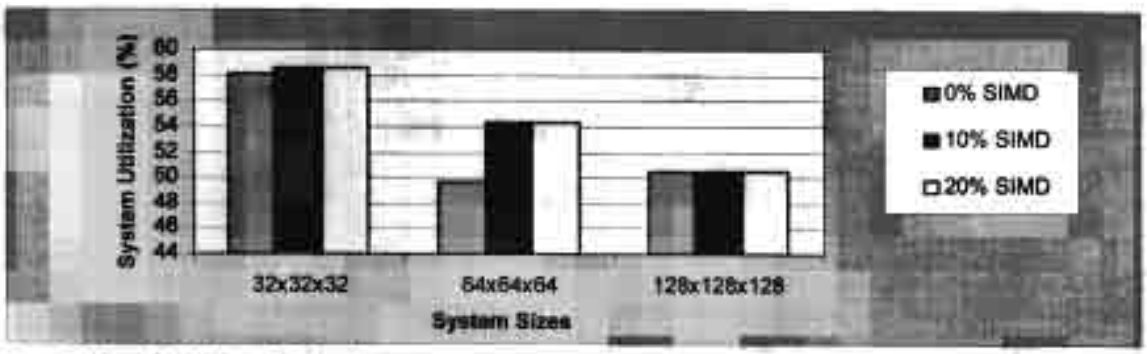


Figure 92 (cont.): b) for the 3-D Meshes.

Table 16 and Figure 93 illustrate the results of the system utilization of the binary-tree and the k-tree strategies for the 2-D and 3-D meshes. For the 2-D meshes, the results showed that the binary-tree strategy yielded the improved system utilization over that the k-tree strategy, except when N = 64x64. For the 3-D meshes, both binary-tree and k-tree strategies yielded the comparable system utilization in all test cases.

Table 16: Effect of "the system sizes" to the system utilization (%) for the 2-D and 3-D Meshes.

2-D Mesh			3-D Mesh		
System Sizes ($N = n_1 x n_2$)	Binary-Tree	k-Tree	System Sizes ($N = n_1 x n_2 x n_3$)	Binary-Tree	k-Tree
64x64	60.75	60.97	32x32x32	47.24	47.71
128x128	61.63	60.11	64x64x64	44.46	44.28
256x256	60.30	59.77	128x128x128	43.30	42.53

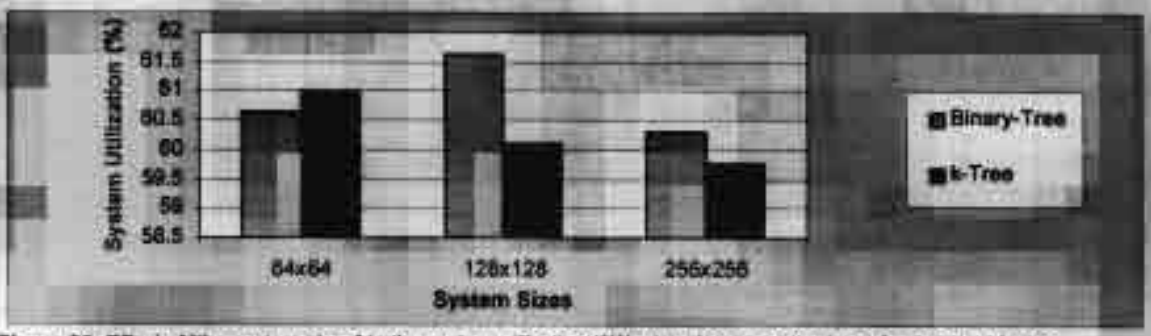


Figure 93: Effect of "the system sizes" to the system utilization (%)by tree-based methods: a) for the 2-D Meshes.

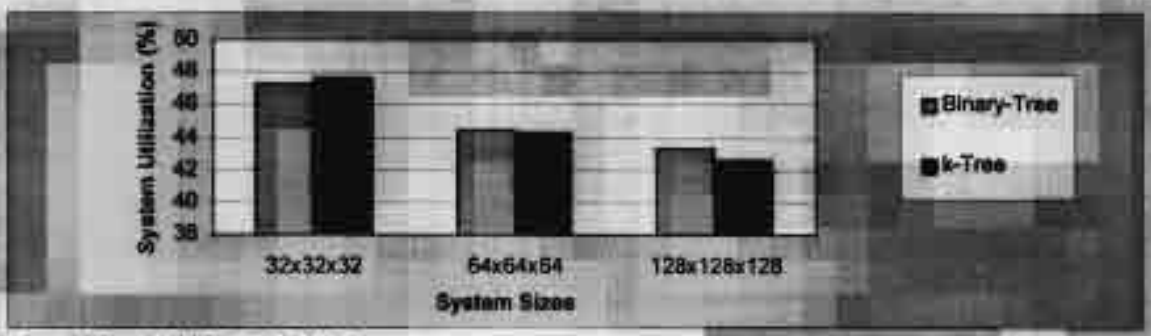


Figure 93 (cont.): b) for the 3-D Meshes.

Table 17 and Figure 94 illustrate the results of the system fragmentation of the modified k-tree strategy for the 2-D and 3-D meshes when the percentage of the SIMD tasks were increased (such as 0%, 10%, 20%). The results showed that increasing the percentage of having SIMD tasks in the system did not effect the system fragmentation for the 2-D and the 3-D meshes, except when N = 64x64. The results were similar to the system utilization since $U_{sys} = 1\text{-}F_{sys}$ (or since there was no internal system fragmentation).

Table 17. Effect of the "system sizes" to the system fragmentation (%) for the 2-D and 3-D Meshes.

2-D Mesh				3-D Mesh			
	% of SIMD Tasks mixed with MIMD Tasks						
System Sizes ($N = n_1 x n_2$)	0%	10%	20%	System Sizes ($N = n_1 x n_2 x n_3$)	0%	10%	20%
64x64	31.24	31.54	28.09	32x32x32	41.86	41.45	41.40
128x128	31.75	32.18	32.18	64x64x64	50.27	45.68	45.67
256x256	29.84	29.84	29.84	128x128x128	49.48	49.48	49.48

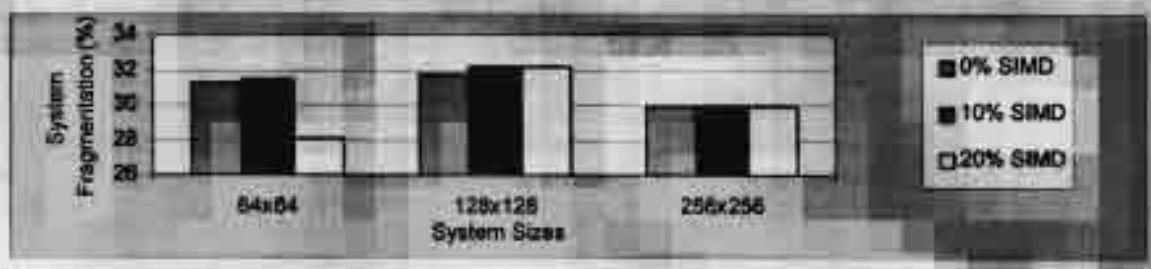


Figure 94: Effect of "the system sizes" to the system fragmentation (%): a) for the 2-D Meshes.

Figure 94 (cont.): b) for the 3-D Meshes.

Table 18 and Figure 95 illustrate the results of the system fragmentation of the binary-tree and the k-tree strategies for the 2-D and 3-D meshes. The results showed that the binary-tree strategy yielded the improved system utilization over that the k-tree strategy, except when N = 64x64 they are comparable.

Table 18: Effect of "the system sizes" to the system fragmentation (%) for the 2-D and 3-D Meshes.

2-D Mesh			3-D Mesh		
System Sizes ($N = n_1 x n_2$)	Binary-Tree	k-Tree	System Sizes ($N = n_1 x n_2 x n_3$)	Binary-Tree	k-Tree
64x64	39.25	39.03	32x32x32	52.76	52.39
128x128	38.37	39.89	64x64x64	55.54	55.72
256x256	39.70	40.23	128x128x128	56.70	57.47

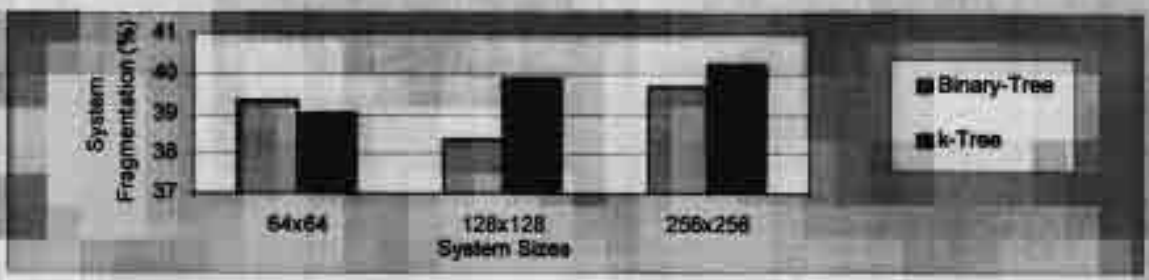


Figure 95: Effect of "the system sizes" to the system fragmentation (%) by tree-based methods: a) for the 2-D Meshes.

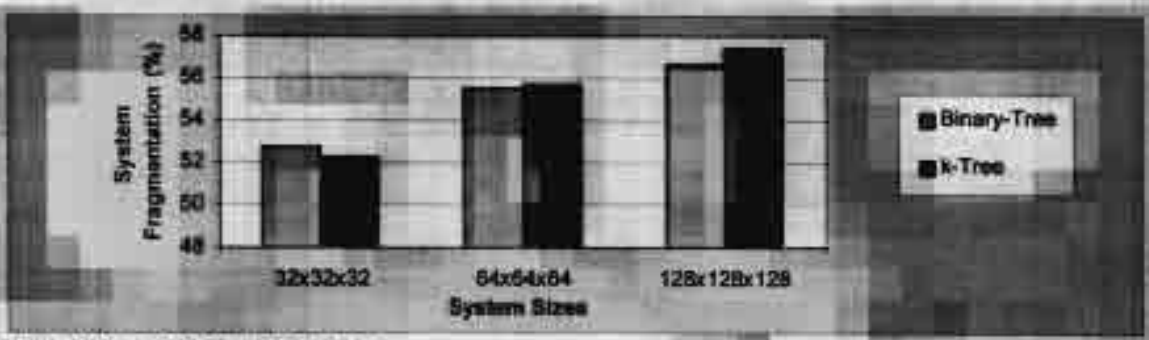


Figure 95 (cont.): b) for the 3-D Meshes.

6. CONCLUSION AND FUTURE STUDY

This study introduces "a universal tree-based model" to perform dynamic resource (CU/PE) allocation decision for the reconfigurable and partitionable MSIMD/MIMD parallel systems. Our universal resource (CU/PE) allocation model can be applied to all interconnection networks in the product-network class such as multi-dimensional meshes, multi-dimensional tori, hypercubes, n-ary k-cubes, etc. Since these reconfigurable systems can execute various dynamic tasks (with MIMD and SIMD modes) in different partitions during run time, we present the new binary-tree-based approach for MIMD and SIMD tasks in efficient time. Moreover, we provide the modified k-tree-based approach to be more useful by adding the CU allocation to cover SIMD partitions for the reconfigurable and partitionable MSIMD/MIMD parallel systems. Time complexity of the tree-based universal model (for MIMD tasks) is efficient for any k-D product-network systems, compared to the original k-tree-based approach and also efficient when applied to the 2-D mesh systems, compared to the recently 2-D mesh-based processor allocation strategies (i.e., the free sub-list strategy, the busy list strategy, and the quick allocation strategy). The total time complexity (for CU/PE allocation) depends on the time complexity of integrating the CU allocation method into the system. Therefore, we also introduces three best-fit heuristics for the tree-based CU allocation: 1) the CU-DFS strategy in $O(N_A + kN_F + k^2)$ time and 2) the CU-AS strategy in $O(k^2)$ time and 3) the CU-IS strategy in $O(1)$ time. Finally, we perform many experiments to investigate system performance of applying our new binary tree-based (CU/PE) allocation model for the reconfigurable and partitionable 2-D and 3-D meshes. By simulation study, the results showed that our binary-tree strategy yielded the comparable system utilization and system fragmentation to those by the k-tree strategy and improved over those by the k-tree strategy in some cases. For the 2-D partitionable meshes, our binary-tree-based results and modified k-tree-based results were also comparable to those of the recently 2-D mesh-based strategies (i.e., the free sub-list strategy, the busy list strategy, and the quick allocation strategy) and improved over those of the recently strategies for some test cases in more efficient time.

In the future study, we will apply our universal and general resource allocation model to some practical applications in parallel and distributed computing, high performance computing, and super computing.

7. REFERENCES

[1] S. Al-Bassam and et al., Processor Allocation for Hypercubes, Journal of Parallel and Distributed Computing, v.16, pp. 394-401, 1992.

[2] M. S. Baig, Dynamic Reconfiguration of Partitionable MSIMD/MIMD Parallel Systems, Doctoral Dissertation, The George Washington University, December 1991.

[3] M. S. Baig, N. A. Alexandridis, and T. El-Ghazawi, A Highly Reconfigurable Multiple SIMD/MIMD Architecture, In proceeding of the fourth ISMM International Conference on Parallel and Distributed Computing and Systems, pp. 35-36, October 1991.

[4] 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.

[5] F.A. Briggs et al., PM4- A Reconfigurable Multiprocessor System for Pattern Recognition and Image Processing, In Proceeding of National Computer Conference, pp. 255-265, 1979.

[6] M. Chen and K. G. Shin, Processor Allocation in an N-Cube Multiprocessor Using Gray Codes, IEEE Transactions on Computers, v.C-36(12), pp.1396-1407, 1987.

[7] P. J. Chuang and N. F. Tzeng, Dynamic Processor Allocation in Hypercube Computers, In Proceeding of the 17th International Symposium on Computer Architecture, May 1990.

[8] P. J. Chuang and N.F. Tzeng, An Efficient Submesh Allocation Strategy for Mesh Computer Systems, in Proceeding of International Conference on Distributed Computing Systems, pp. 256-263, May 1991.

[9] P. J. Chuang and N.F. Tzeng, A Fast Recognition-Complete Processor Allocation Strategy for Hypercube Computers, IEEE Transactions on Computers, v.41(4), pp. 467-479, 1992.

[10] 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.

[11] D. Das Sharma and D. K. Pradhan, A Novel Approach for Subcube Allocation in Hypercube Multiprocessors, in Proceeding of the Fourth IEEE Symposium on Parallel and Distributed Processing, pp. 336-345, December, 1992.

[12] D. Das Sharma and D.K. Pradhan, A Fast and Efficient Strategy for Submesh Allocation in Mesh-Connected Parallel Computers, in Proceeding of the fifth IEEE Symposium on Parallel and Distributed Processing, pp. 682-689, 1993.

[13] D. Das Sharma, Space and Time Scheduling in Multicomputers, Ph.D. Dissertation, University of Massachusetts, 1995.

[14] D. Das Sharma and D.K. Pradhan, Submesh Allocation in Mesh Multicomputers Using Busy-List: A Best-Fit Approach with Complete Recognition Capability, Journal of Parallel and Distributed Computing, v.36, pp. 106-118, 1996.

[15] 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.

[16] N. J. Dimopoulos, and et al, Routing and processor allocation on a hypercycle-based multiprocessor, in Proceeding of International Conference on Supercomputing., ACM, pp. 106-114, 1991.

[17] N. J. Dimopoulos and V. V. Dimakopoulos, Optimal and Suboptimal Processor Allocation for Hypercycle-based Multiprocessors, IEEE Transactions on Parallel and Distributed Systems, v.6(2), pp. 175-184, 1995.

[18] 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.

[19] S, Dutt and J. P. Hayes, On Allocating Subcubes in a Hypercube Multiprocessor, In Proceeding of the third Conference on Hypercube Computers and Applications, pp. 801-810, January 1988.

[20] S. Dutt and J. P. Hayes, Subcube Allocation in Hypercube Computers, IEEE Transactions on Computers, v.40(3), pp. 341-352, 1991.

[21] T. El-Ghazawi and A. Youssef, A General Framework for Developing Adaptive Fault-Tolerant Routing Algorithms, IEEE Transactions on Reliability, v. 42(2), pp. 250-258, 1993.

[22] Intel, A Touchstone DELTA System Description, Supercomputer Systems Division, Intel Corporation, Beaverton, OR 97006, 1991.

[23] Intel, Paragon XP/S Overview, Supercomputer Systems Division, Intel Corporation, Beaverton, OR 97006 (1991)

[24] J. Kim, C. R. Das, and W. Lin, A Processor Allocation Scheme for Hypercube Computers, In Proceeding of International Conference on Parallel Processing, pp. 231-238, August 1989.

[25] J. Kim, C.R. Das, and W. Lin, A Top-Down Processor Allocation Scheme for Hypercube Computers, IEEE Transactions on Parallel and Distributed Systems, v.2(1), pp. 20-30, 1991.

[26] 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.

[27] R. Krishnamurti, Reconfigurable Parallel Architectures for Special Purpose Computing, Ph. D. Thesis, Department of Computer and Information Sciences, University of Pennsylvania, Philadelphia, PA 1987.

[28] 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.

[29] K. Li and K. H. Cheng, A Two-Dimensional Buddy System for Dynamic Resource Allocation in a Partitionable Mesh ConnecteSystem, Journal of Parallel and Distributed Computing, v.12, pp. 79-83, 1991.

[30] H. Li and M. Maresca, Polymorphic-Torus Network, IEEE Transactions on Computers, v. C-30, pp. 1345-1351, March 1981.

[31] 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.

[32] Y. W. Ma and R. Krishnamurti, The Architecture of REPLICA- A Special Purpose Computer System for Active Multi-Sensory Perception for 3-Dimensional Objects, In Proceeding of International Conference on Parallel Processing, pp. 30-37, August 1984.

[33] P. Mattson and et al., Intel/Sandia ASCI system, in Proceeding of International Parallel Processing Symposium, 1996.

[34] P. Mohapatra and et al., A Lazy Scheduling Scheme for Hypercube Computers, Journal of Parallel and Distribute Computing, v.27, pp. 26-37, 1995.

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

[36] G. J. Nutt, Microprocessor Implementation of a Parallel Processor, The 4th Annual Symposium on Computer Architecture, pp. 147-152, March 1977.

[37] S. Rai, J. L. Trahan, and T. Smailus, Processor Allocation in Hypercube Multiprocessors, IEEE Transactions on Parallel and Distributed Systems, v.6(6), pp. 606-616, 1995.

[38] M. Sharma et al., NETRA: An Architecture for a Large Multi-Processor Vision System, Parallel Computer Vision, L. Uhr, editor, Academic Press Inc., Florida, 1987.

[39] H. J. Siegel et al., A Survey of Interconnection Methods for Reconfigurable Parallel Processing Systems, In Proceeding of National Computing Conference, pp. 529-541, 1979.

[40] H. J. Siegel et al., PASM: A Partitionable SIMD/MIMD System for Image Processing and Pattern Recognition, IEEE Transactions on Computers, v.C-30, pp. 934-947, 1981.

[41] J. Srisawat, A Unified Approach to Processor Allocation and Task Scheduling for Partitionable Parallel Architectures, Doctoral Dissertation, The George Washington University, Washington DC, USA, August 31, 1999.

[42] J. Srisawat and N.A. Alexandridis, Efficient Processor Allocation Scheme with Task Embedding for Partitionable Mesh Architectures, in Proceeding of International Conference on Computer Applications in Industry and Engineering, pp. 305-308, Las Vegas, USA, November, 1998.

[43] J. Srisawat and N.A. Alexandridis, A Quad-Tree Based Dynamic Processor Allocation Scheme for Mesh-Connected Parallel Machines, in Proceeding of International Conference on Computer Applications in Industry and Engineering, pp. 309-312, Las Vegas, USA, November, 1998.

[44] J. Srisawat and N.A. Alexandridis, Reducing System Fragmentation in Dynamically Partitionable Mesh-Connected Architectures, in Proceeding of International Conference on Parallel and Distributed Computing and Networks, pp. 241-244, Brisbane, Australia, December, 1998.

[45] 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.

[46] 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.

[47] L. Snyder, Introduction to the Configurable Highly Parallel Computer, Computer, pp. 47-56, Jan. 1982.

[48] H. Wang and Q. Yang, Prime Cube Graph Approach for Processor Allocation in Hypercube Multicomputers, In Proceeding of International Conference on Parallel Processing, vol. I, pp. 25-32, September 1991.

[49] O. Yang and H. Wang, A New Graph Approach to Minimizing Processor Fragmentation in Hypercube Multiprocessors, IEEE Transactions on Parallel and Distributed Systems, v.4(10), pp. 1165-1171, 1993.

[50] 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.

[51] A. Youssef, Design and Analysis of Product Networks, In Proceeding of Frontiers' 95 Fifth Symposium of Massively Parallel Computation, pp. 521-528, 1995.

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

APPENDIX A

OUTPUT

1. ผลงานที่คาดว่าตีพิมพ์

1. J. Srisawat, W. Surakampontorn, and N.A. Alexandridis, **"A New Binary-Tree-Based Subsystem Allocation for Partitionable k-D Mesh Multicomputers,"** (expected to be published in Journal of Parallel and Distributed Computing in 2004 - 2005.)

2. J. Srisawat, W. Surakampontorn, and N.A. Alexandridis, **"A Universal CU and PE Allocation for Dynamic Reconfigurable MSIMD/MIMD Parallel Systems,"** (expected to be published in IEEE Transactions on Parallel and Distributed Systems in 2004 - 2005.)

2. การนำผลงานวิจัยไปใช้ประโยชน์

ผลงานวิจัยในขั้นนี้ สามารถนำไปใช้ประโยชน์ในเชิงวิชาการ ในด้านการพัฒนาการเรียนการสอนในระดับปริญญาโท และปริญญาโทเอก

3. การเสนอผลงานในที่ประชุมวิชาการ

J. Srisawat, W. Surakampontorn, and N.A. Alexandridis, **"A k-Tree-Based Resource (CU/PE) Allocation for Reconfigurable MSIMD/MIMD Multi-dimensional Mesh-connected Architectures,"** *in Proceeding of the 2002 International Technical Conference on circuits/systems, Computers and Communications*, v. 1, pp. 58-62, Phuket, Thailand, July 2002.

APPENDIX B

REPRINT PUBLICATION

Title: A k-Tree-Based Resource (CU/PE) Allocation for Reconfigurable MSIMD/MIMD Multi-dimensional Mesh-connected Architectures

Authors: Jeeraporn Srisawat, Wanlop Surakampontorn, and Nikitas A. Alexandridis

Proceeding: In Proceeding of the 2002 International Technical Conference on Circuits/Systems, Computers and Communications, v. 1, pp. 58-62, Phuket, Thailand, July 2002.

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).

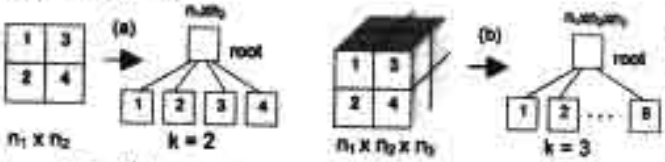


Figure 1. The 2^k buddies of the k-Trees: (a) 2-D mesh; and (b) 3-D mesh

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.

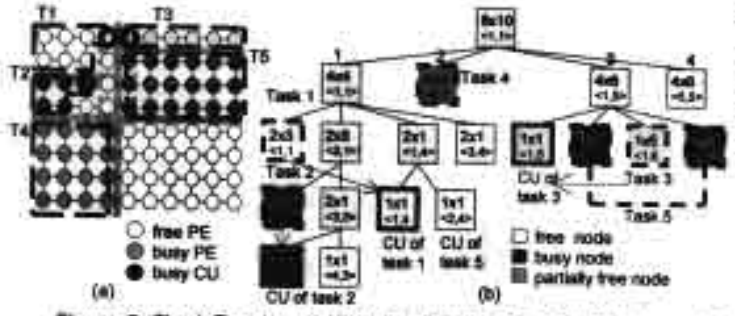


Figure 2. The k-Tree-based (CU/PE) allocation for 3 SIMD tasks and 2 MIMD tasks on an 8x10 mesh system: (a) the allocated system status and (b) the corresponding k-Tree system state representation.

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 $j = 1, 2, \ldots, k-1$) in 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 $\langle(a_1, a_2, \ldots, a_k), (b_1, b_2, \ldots, b_k)\rangle$, then the number of all possible CUs are $2 \sum_{i=1}^{k} m_1 \times m_2 \times \ldots \times m_{i-1} \times 1 \times m_{i+1} \times \ldots \times m_k$. 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 $= m_1 x m_2 x \ldots x 1 x m_{i+1} x \ldots x m_k$), where i=1,2,...,k.
Min CUS address = $\langle(a_1, a_2, \ldots, a_i\text{-}1, \ldots, a_k), (b_1, b_2, \ldots, a_i\text{-}1, \ldots, b_k)\rangle$
Max CUS address = $\langle(a_1, a_2, \ldots, b_i\text{+}1, \ldots, a_k), (b_1, b_2, \ldots, b_i\text{+}1, \ldots, b_k)\rangle$
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 $\langle(a_1, a_2), (b_1, b_2)\rangle = \langle(5, 5), (11, 12)\rangle$) are

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

For any free node R (of size $d_1 \times d_2 \times \ldots \times d_k$) 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 $[(d_1\text{-}2) x (d_2\text{-}2) x \ldots x (d_{i-1}\text{-}2) x 1 x d_{i+1} x \ldots x d_k]$), where i = 1, 2, ..., k.
Min BS addr=$\langle(a_1\text{+}1, a_2\text{+}1, \ldots, a_i, a_{i+1}, \ldots, a_k), (b_1\text{-}1, b_2\text{-}1, \ldots, a_i, b_{i+1}, \ldots, b_k)\rangle$
Max BS addr=$\langle(a_1\text{+}1, a_2\text{+}1, \ldots, b_i, a_{i+1}, \ldots, a_k), (b_1\text{-}1, b_2\text{-}1, \ldots, b_i, b_{i+1}, \ldots, b_k)\rangle$
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 $d_1 \times d_2$ = 7 x 8) (at $\langle(a_1, a_2), (b_1, b_2)\rangle = \langle(5, 5), (11, 12)\rangle$) are

- For a fixed dim 1,
  - a min BS (8 PEs): $\langle(a_1, a_2), (a_1, b_2)\rangle = \langle(5, 5), (5, 12)\rangle$
  - a max BS (8 PEs): $\langle(b_1, a_2), (b_1, b_2)\rangle = \langle(11\_5), (11,12)\rangle$
- For a fixed dim 2,
  - a min BS (5 PEs): $\langle(a_1\text{+}1, \mathbf{a_2}), (b_1\text{-}1, \mathbf{a_2})\rangle = \langle(6, 5), (10, 5)\rangle$
  - a max BS (5 PEs): $\langle(a_1\text{+}1, \mathbf{b_2}), (b_1\text{-}1, \mathbf{b_2})\rangle = \langle(6,12), (10,12)\rangle$

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, $j = 1, 2$.
$|S_i|$ is $n_1 \times n_2 \times \ldots \times n_k$ at $<(a_1, a_2, \ldots, a_k), (b_1, b_2, \ldots, b_k)>$, $j=1,2, \ldots, k$.
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 $\exists j$, $j = 1,2,\ldots, k$
S_1 and S_2 are disjoint either if $(a_1 + a_2 \geq n_2)$ or if $(a_2 + a_1 \geq n_1)$

For example, suppose k = 2 and a selected sub-system is S ($m_1 \times m_2$) and a free node is R ($d_1 \times d_2$). 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.

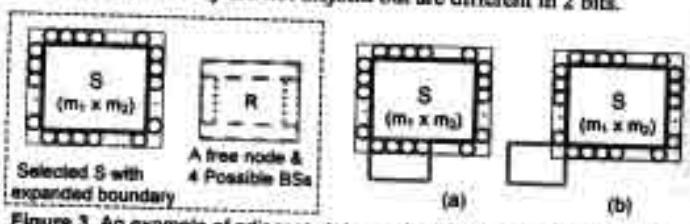


Figure 3. An example of adjacent statuses: (a) adjacent; (b) not adjacent.

2.4.2.2 The CU Adjacent Search (CU-AS)

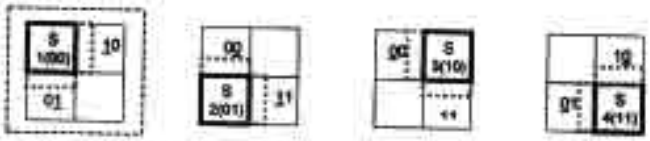
The CU adjacent search (CU-AS) is another approach used to find any free node R that is adjacent to the selected sub-system S. In this method, the searching starts from S and its adjacent nodes can be identified directly (see the following 4 cases). If the node R is free, its corresponding BSs are identified 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 $I = 1, 2, \ldots,$ 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, $j = 1, 2, \ldots, k$
negate b_{j-1} $(b_{k-1} \ldots \underline{b_{j-1}} \ldots b_1 b_0)$.
if $n_{j-1} \neq 1$, 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

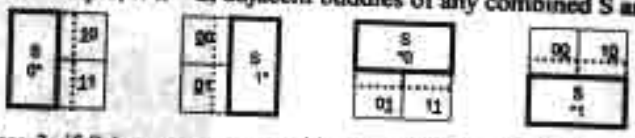


† - Negate dim 1 ($b_1\underline{b_0}$ => 01). Compute minBS $<(a_1, a_2), (a_1, b_2)>$
- Negate dim 2 ($\underline{b_1}b_0$ => 10). Compute minBS $<(a_1, a_2), (b_1, a_2)>$

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}, $j = 1, 2, \ldots, k-1$)
T is a ternary string of $S = (b_{k-1} \ldots b_j \ldots b_0)$
There exist ja' * in T [i.e., k Ts are $(b_{k-1} \ldots b_j \ldots *), (b_{k-1} \ldots b_{j-1} \ldots *b_0), \ldots,$ $(*b_{k-2} \ldots b_{j-1} \ldots *)$ 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 = $(b_{k-1} * \ldots **)$, 2^{k-1} adjacent nodes are
negate b_{k-1} $(n_{k-1} * \ldots **)$, 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).

THEOREM 1: 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 $O(k^3 2^{2k}(N_A+N_F))$. 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 $O(k^32^{2k})$ for all combined sub-systems and $O(k^32^k(N_A+N_F))$ 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 $O(k^32^{2k})$. Thus, total time complexity of the k-Tree-based (CU/PE) allocation with applying the CU-AS heuristic is $O(k^32^k(N_A+N_F))$, 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 $O(k^42^k(N_A+N_F)\ (kN_F))$ 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(λ) (or inter-arrival time – Exp($1/\lambda$=5)), and service time – Exp(μ=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 ($1x1 - n_1/2 \times n_2/2$) 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			
$N=n_1xn_2$	0%	10%	20%	N	0%	10%	20%
64x64	68.76	68.46	71.91	n=32	58.14	58.55	59.80
128x128	66.25	67.62	67.62	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	78.565	78.268	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.