ขั้นตอนวิธีการหาเส้นทางที่สั้นที่สุด สำหรับการแลกเปลี่ยนแบบเพอร์ซันนัลไลซ์ ออล-ทู-ออล ที่เหมาะสมที่สุด บนเครือข่ายการเชื่อมต่อไฮเปอร์คิวบ์แบบชั้น

SHORTEST-PATH ROUTING ALGORITHMS FOR OPTIMAL ALL-TO-ALL PERSONALIZED EXCHANGE ON HIERARCHICAL HYPERCUBE NETWORKS

นันทิพัฒน์ พิศุทธางกูร

NUNTIPAT PHISUTTHANGKOON

วิทยานิพนธ์นี้สำหรับการศึกษาตามหลักสูตร

ปริญญาปรัชญาดุษฎีบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์

ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2565

KMITL-2022-SC-D-002-114

SHORTEST-PATH ROUTING ALGORITHMS FOR OPTIMAL ALL-TO-ALL PERSONALIZED EXCHANGE ON HIERARCHICAL HYPERCUBE NETWORKS

NUNTIPAT PHISUTTHANGKOON

A THESIS SUBMITTED IN FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF DOCTOR OF PHILOSOPHY PROGRAM IN COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE SCHOOL OF SCIENCE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2022

KMITL-2022-SC-D-002-114

หัวข้อวิทยานิพนธ์	ขั้นตอนวิธีการหาเส้นทางที่สั้นที่สุด สำหรับการแลกเปลี่ยนแบบเพอร์ซันนัลไลซ์ ออล-ทู-ออล ที่เหมาะสมที่สุด บนเครือข่ายการเชื่อมต่อไฮเปอร์คิวบ์แบบชั้น
ชื่อนักศึกษา	เรืออากาศตรี นันทิพัฒน์ พิศุทธางกูร
รหัสประจำตัว	62605119
ปริญญา	ปรัชญาดุษฎีบัณฑิต (วิทยาการคอมพิวเตอร์)
ภาควิชา	วิทยาการคอมพิวเตอร์
พ.ศ.	2565
อาจารย์ที่ปรึกษาวิทยานิพนธ์	รองศาสตราจารย์ ดร. จีรพร วีระพันธุ์

บทคัดย่อ

ในระบบการประมวลผลแบบขนานและแบบกระจาย เครือข่ายไฮเปอร์คิวบ์ (Hypercube Networks : HC) คือ หนึ่งในเครือข่ายที่มีประสิทธิภาพในการสื่อสารเป็นอย่างมาก แต่การนำเครือข่าย HC มาสร้างลงบนชิปของระบบมัลติโพเซสเซอร์ (Multiprocessor : MP) จะมีต้นทุนการสร้างที่สูง เนื่องจากมีเส้นเชื่อมเพิ่มขึ้นตามขนาดเครือข่ายที่ใหญ่ขึ้น ขณะที่เครือข่ายไฮเปอร์คิวบ์แบบชั้น (Hierarchical Hypercube : HHC) เป็นเครือข่ายที่มีราคาถูก และมีความยืดหยุ่นมากกว่าสำหรับการสร้างระบบ MP แต่เครือข่าย HHC มีข้อจำกัดจากปัญหาการชนกันของข้อมูลในระหว่างการติดต่อสื่อสาร เนื่องจากความซับซ้อนของเครือข่ายและเส้นเชื่อมที่ถูกลดลง ด้วยเหตุนี้ ในปี 2007 มีการศึกษาและออกแบบการสื่อสารระหว่างต้นทาง S และปลายทาง D แต่ยังมีข้อจำกัดเนื่องจากต้องเริ่มที่ $S = 0$ เท่านั้น ซึ่งในทางปฏิบัติเครือข่าย HHC-MP จะต้องเริ่มต้นจากต้นทาง S ใดๆ โดยที่ไม่ขัดแย้งกับเส้นทางอื่นระหว่างสื่อสารพร้อมกันแบบขนาน และในปี 2013 ได้มีการนำเสนอการหาเส้นทางแบบหนึ่งต่อหนึ่ง (Node to Node) แบบขนานระหว่างหน่วยประมวลผลจำนวน $k \leq \lceil (m+1)/2 \rceil$ ซึ่งในทางปฏิบัติแล้วควรจะรองรับหน่วยประมวลผลได้มากกว่า $\lceil (m+1)/2 \rceil$ ดังนั้นวิทยานิพนธ์ฉบับนี้จึงนำเสนอ 1. การหาเส้นทางการสื่อสารแบบหนึ่งต่อหนึ่งที่สั้นที่สุดจาก S และ D ใดๆ 2. การหาเส้นทางการสื่อสารที่สั้นที่สุดแบบขนานจาก S และ D ใดๆ โดยใช้วิธีการจัดลำดับบิตไปข้างหน้าและถอยกลับแบบวนรอบ 3. การฝังฟังก์ชัน ATAPE (All-to-all Personalized Exchange) แบบขนานลงบนชิปด้วยเส้นทางที่สั้นที่สุด สำหรับการแบ่งกลุ่มของดูอัล-คิวบ์แบบไขว้ (Grouping of Cross Dual-cube : GCD) และการรวมกลุ่มคิวบ์ย่อยแบบไขว้ (Grouping of Cross Sub-cube : GCS) ขนาด $k \leq 2^n/2^{M-1}$ เมื่อ $n = 2^m+m$ และ $M = 2^{m-1}$ ที่ทำให้ได้ความถูกต้อง 3 ปัจจัย (Triple Right Assignment) คือ งานที่ถูกต้อง (Right Task), กลุ่มที่ถูกต้อง (Right Partition) และเวลาที่ถูกต้อง (Right Time) และสุดท้าย 4. การจัดสรรหน่วยประมวลบนโครงสร้างต้นไม้แบบทวิภาคที่ปรับเปลี่ยนได้ (Reconfigurable Binary Tree) เพื่อรองรับการจัดกลุ่มของหน่วยประมวลผลที่เหมาะสม บนเครือข่าย HHC-MP จากการทดลองพบว่าการแบ่งกลุ่มแบบ GCD และ GCS สำหรับหน่วยประมวลผล $k \leq 2^n/2^{M-1}$ สามารถสื่อสารเพื่อแลกเปลี่ยนข้อมูลแบบ ATAPE ด้วยเส้นทางที่สั้นที่สุดและไม่ขัดแย้งกัน โดยแต่ละกลุ่มย่อยสามารถสื่อสารพร้อมกันแบบขนานอย่างอิสระหรือใช้ประโยชน์ (Utilization) ของระบบได้ 100%

คำสำคัญ : การหาเส้นทางการสื่อสารแบบขนานที่สั้นที่สุด การแลกเปลี่ยนแบบเพอร์ซันนัลไลซ์ออล-ทู-ออลที่ดีที่สุด โครงสร้างต้นไม้แบบทวิภาคที่ปรับเปลี่ยนได้เพื่อการจัดสรรงาน เครือข่ายไฮเปอร์คิวบ์แบบชั้น

Thesis Title	Shortest-path Routing Algorithms for Optimal All-to-All Personalized Exchange on Hierarchical Hypercube Networks
Student Name	Pilot Officer Nuntipat Phisutthangkoon
Student ID	62605119
Degree	Doctor of Philosophy (Computer Science)
Department	Computer Science
Year	2022
Thesis Advisor	Associate Professor Dr. Jeeraporn Werapun

Abstract

Hypercube (HC) networks ($N = 2^n$) provide efficient communication for parallel and distributed computing but the HC-based multi–processor (MP) system is costly and not scalable, while hierarchical hypercube (HHC) networks ($N = 2^n$, $n = 2^m + m$) are less expensive and more scalable for the MP systems but the HHC-routing easily conflicts. Recently, one effective solution for the conflict on the HHC–MPs was the node-disjoint path routing between (S, D) nodes but that solution was available for reliable routing with S (source) = 0. However, the realistic HHC-MP systems require the arbitrary-S routing without conflicts for the efficient parallel processing. Later 2013, parallel reliable N2N (node-to-node) routing was proposed for $k \leq \lceil (m+1)/2 \rceil$ nodes, while the HHC-MPs should support $k > \lceil (m+1)/2 \rceil$. Thus, the first innovation-and-contribution of this study is a new generalized node-to-node (N2N) shortest-path routing from source nodes S to destination nodes D as well as the original parallel N2N SP-routing, proposed with forward and backward reordering algorithms in $O(|\mu|^2)$ for embedding on the HHC-MPs, where $|\mu|$ is Hamming-distance between S-and-D. Next contribution is the optimal ATAPE (all–to–all personalized exchange) communication, which is embedded for parallel $N2N$-SP functions (k) on the HHC-MPs, where $k \leq 2^n/2^{M-1}$ where $n = 2^m+m$ and $M = 2^{m-1}$. Next contribution, GCD (Grouping of Cross Dual-cube) partitioning and GCS (Grouping of Cross Sub-cube) combining are presented for multiple-task mapping without conflicts on 2^M independent groups in order to achieve the triple-right assignment (right task, right partition, right time). Finally, a reconfigurable tree-structure to recognize the GCD and GCS patterns systematically for efficient processor allocation and deallocation on the HHC-MPs. The experiment was conducted to evaluate our conflict solution by applying the ATAPE communication. The result confirmed that there were no conflicts on the GCD-and-GCS mapping for the 100% HHC-utilization.

Keywords: Generalized parallel shortest-path routing; Optimal ATAPE (All-to-All personalized exchange); Reconfigurable Binary Tree for task scheduling on HHC; Hierarchical hypercube networks (HHC).

กิตติกรรมประกาศ

วิทยานิพนธ์นี้มิอาจจะสำเร็จลุล่วงไปได้ด้วยดี หากมิได้รับคำแนะนำ คำชี้แจง ความรู้ และความเอาใจใส่เป็นอย่างดี จาก รศ.ดร.จีรพร วีระพันธุ์ ผู้เป็นอาจารย์ที่ปรึกษา ซึ่งท่านได้สละเวลาให้กับข้าพเจ้าอย่างเต็มที่ จึงใคร่ขอขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณ ผศ.ดร.นวลสวาท หิรัญสกลวงศ์ ผศ.ดร.อนันตพร หรรษคุณาฒัย ผศ.ดร.วรางคณา กิ้มปาน และ รศ.ดร.ศุภกานต์ พิมลธเรศ คณะกรรมการสอบหัวข้อ และโครงร่างวิทยานิพนธ์ ที่กรุณาให้คำแนะนำตลอดจนชี้แนะจนในที่สุดทำให้วิทยานิพนธ์ฉบับนี้สำเร็จลงได้

ขอขอบพระคุณคณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ที่สนับสนุนทุนการศึกษา

ขอขอบพระคุณกองทัพอากาศ และโรงเรียนนายเรืออากาศนวมินทกษัตริยาธิราช ที่อนุญาตให้ได้รับการลาศึกษา และสนับสนุนในทุกๆด้าน

ขอขอบพระคุณบิดา มารดา ที่สนับสนุนให้ได้เรียนในระดับที่ตั้งใจ และขอขอบคุณภริยา ที่เป็นกำลังใจในระหว่างการศึกษาเป็นอย่างดียิ่ง

ขอขอบคุณพี่ๆ และเพื่อนๆ ที่ได้เรียน ได้ศึกษางานวิจัยร่วมกัน ช่วยเป็นกำลังใจส่งเสริมสนับสนุนในทุกๆด้าน

ขอขอบพระคุณบรรณาธิการวารสารวิชาการระดับนานาชาติ Parallel and Distributed Computing (JPDC), Parallel Computing Systems & Applications ผู้อ่านงานวิจัยทุกๆท่าน และผู้ที่มีส่วนร่วมให้ความคิดเห็น วิพากษ์วิจารณ์ เปิดโอกาส และให้การยอมรับแนวคิดให้ได้รับการตีพิมพ์ในวารสาร

สำหรับคุณงามความดี และประโยชน์อันใดที่เกิดขึ้นจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้กับบิดา มารดา อาจารย์ทุกท่านซึ่งเป็นที่เคารพรักยิ่ง ตลอดจนญาติพี่น้อง และผองเพื่อนทุกคน

นันทิพัฒน์ พิศุทธางกูร

สารบัญ

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

คำย่อ/สัญลักษณ์

ATAPE	All-to-All Personalized Exchange
HC	Hypercube
HHC	Hierarchical Hypercube
MP	Multiprocessor
HD	Hamming Distance
HHC-MP	Hierarchical Hypercube Multiprocessor
PDS	Parallel and Distributed System
(S, D)	Source and Destination
C	Control Units
GC	Gray-Code
GCD	Grouping of Cross Dual-cube
GCS	Grouping of Cross Sub-cube
Cross	Cross of Dual-cubes
SP	Shortest Path
MIN	Multistage Interconnection Network
HIN	Hierarchical Interconnection Network
THIN	Triple-based HIN
dBCube	DeBruijn-Cube Networks
HFCube	Hierarchical Folded Cube
EES	External Edge Sequence
XOR	Exclusive OR Operation
LS	Latin-Square
sc	Sub-cube
dc	Dual-cube
fs	First Sub-cube
ss	Second Sub-cube
N	Total number of Processors ($N = 2^n$, $n = 2^m+m$ and $m \geq 2$)
k	Groups of processors
α	2^m-bit String of Main-net (b_{n-1}, ..., b_{m+1}, b_m)
β	m-bit String of Sub-net (b_{m-1}, ..., b_1, b_0)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในเครื่องประมวลผลสมรรถนะสูง (High Performance Computing Machines) และระบบการประมวลผลแบบขนานและแบบกระจาย (Parallel and Distributed Systems : PDSs) มีรูปแบบเครือข่ายการเชื่อมต่อ (Interconnection Networks) เป็นองค์ประกอบสำคัญของระบบ เพราะเครือข่ายการเชื่อมต่อที่มีประสิทธิภาพ มีบทบาทอย่างมากในการพัฒนาเครื่องคอมพิวเตอร์ยุคใหม่ในอนาคต ซึ่งจะต้องมีหน่วยประมวลผลจำนวนมาก (Massive Multiprocessors) เพื่อรองรับการประมวลผลข้อมูลขนาดใหญ่ (Big Data) โดยรูปแบบเครือข่ายการเชื่อมต่อ ถูกออกแบบมาเพื่อติดต่อสื่อสาร และแลกเปลี่ยนข้อมูล ระหว่างหน่วยประมวลผล และอุปกรณ์อื่นๆ ที่เกี่ยวข้องในระบบ ซึ่งจะต้องดำเนินการติดต่อสื่อสารได้อย่างรวดเร็ว และสามารถเพิ่มขยายขนาดของระบบ (Scalable Systems) ได้โดยไม่ยุ่งยากซับซ้อนในราคาที่เหมาะสม

จากอดีตจนถึงปัจจุบันได้มีการศึกษาค้นคว้าเกี่ยวกับรูปแบบเครือข่ายการเชื่อมต่ออย่างหลากหลาย [1, 2, 8, 15, 19, 20, 21, 22, 25] และหนึ่งในนั้น คือ เครือข่ายไฮเปอร์คิวบ์ (Hypercube Network : HC, $N = 2^n$) ไฮเปอร์คิวบ์มีคุณสมบัติเด่น คือ มีค่าเส้นผ่านศูนย์กลางที่สั้น (Short Diameter) มีค่าโหนดดีกรี (Node Degree) ที่สูง มีการติดต่อสื่อสารที่มีประสิทธิภาพ (Efficient communication) และมีความสมมาตรในตัวเอง (Symmetry) การนำเครือข่ายไฮเปอร์คิวบ์มาสร้างลงบนชิปของระบบมัลติโพรเซสเซอร์ (Multiprocessors : MPs) ที่มีหน่วยประมวลผลจำนวนไม่มาก จะเป็นระบบประมวลผลแบบขนาน (Parallel System) ที่มีประสิทธิภาพมาก อย่างไรก็ตามในการสร้างระบบที่ต้องปรับขยายขนาดเพิ่มได้ตามยุคสมัย โดยเฉพาะยุคที่มีข้อมูลมากมหาศาล (Big-data Era) จะพบว่า ข้อเสียของเครือข่ายไฮเปอร์คิวบ์ คือ ยังไม่มีความยืดหยุ่นที่พอเหมาะพอดี เนื่องจากเมื่อมีการเพิ่มขนาดของเครือข่าย (จำนวนหน่วยประมวลผล) ให้ใหญ่ขึ้น เครือข่ายไฮเปอร์คิวบ์จะมีสัดส่วนด้านราคาที่สูงเพิ่มขึ้นมาก (ซึ่งวัดจากจำนวนเส้นเชื่อมที่มีมากขึ้น)

เครือข่ายไฮเปอร์คิวบ์แบบชั้น (Hierarchical Hypercube Network : HHC, $N = 2^n$, $n = 2^m+m$) เป็นเครือข่ายอีกชนิดหนึ่ง ซึ่งเหมาะสมที่จะนำมาสร้างลงบนชิปของระบบ MPs ที่รองรับการเพิ่มขยายขนาดได้ในราคาที่เหมาะสมกว่าเครือข่าย HC เนื่องจากเครือข่าย HHC มีโครงสร้างเป็นชั้นและในแต่ละชั้นจะประกอบด้วยโครงสร้างพื้นฐานของเครือข่าย HC ดังนั้นจุดเด่นของเครือข่าย HHC คือ เป็นเครือข่ายที่มีราคาถูกกว่าเครือข่าย HC เนื่องจากใช้เส้นเชื่อมที่น้อยลง มีความสมมาตรในตัวเอง อีกทั้งยังมีขนาดของเส้นผ่านศูนย์กลางเครือข่าย และโหนดดีกรีที่เหมาะสมคือไม่ต่างไปจากเครือข่าย HC มากนัก ในขณะที่รองรับความสามารถในการเชื่อมต่อหน่วยประมวลผลได้จำนวนมาก ถึงแม้ว่าเครือข่าย HHC จะสืบทอดคุณสมบัติเด่นของไฮเปอร์คิวบ์ไว้ได้มาก

และสร้างได้ในราคาที่ถูกกว่า แต่เครือข่าย HHC มีข้อจำกัดจากปัญหาการชนกันของข้อมูล (Data Collision) ในระหว่างการติดต่อสื่อสารเพื่อแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผล โดยเฉพาะอย่างยิ่ง ปัญหาจะเกิดขึ้นบ่อย เมื่อต้องแลกเปลี่ยนข้อมูลระหว่างหลายๆ หน่วยประมวลผลพร้อมๆ กัน หมายเหตุ ในงานวิจัยนี้จะเรียก “การชนกันของข้อมูล” ว่า “การขัดแย้งกัน” (Conflict)

จากปัญหาการขัดแย้งที่เกิดขึ้นได้ง่าย ทำให้เครือข่าย HHC ที่นำเสนอไว้เพื่อเป็นต้นแบบ ในปี ค.ศ. 1994 ไม่สามารถนำมาใช้งานได้ ดังนั้นในปี ค.ศ. 2007 ได้มีการศึกษาวิจัยและออกแบบเส้นทางการสื่อสารหลายช่องทาง จากต้นทางหนึ่ง S (Source) สู่อีกปลายทางหนึ่ง D (Destination) เพื่อเพิ่มความน่าเชื่อถือในการแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผลบนเครือข่าย HHC ผลการศึกษาในครั้งนั้น สามารถแก้ปัญหาขัดแย้งดังกล่าวได้ แต่วิธีการที่นำเสนอยังไม่สมบูรณ์ เพราะมีข้อจำกัดที่ต้องเริ่มที่ต้นทาง $S = 0$ เท่านั้น ต่อมาในปี ค.ศ. 2013 ได้มีการศึกษาและออกแบบเส้นทางการสื่อสาร (S, D) แบบขนาน (หรือ (S, D) หลายคู่) จำนวน k เส้นทางที่แตกต่างกันเพื่อเพิ่มความน่าเชื่อถือในการสื่อสารข้อมูลแบบขนาน เริ่มต้นจาก k ต้นทาง ไปยัง k ปลายทางแบบขนาน โดยที่ $k \leq \lceil (m+1)/2 \rceil$ อย่างไรก็ตามในทางปฏิบัติ เครือข่าย HHC จะสามารถใช้งานได้จริง ถ้าสามารถแก้ปัญหาการขัดแย้งต่างๆ ที่อาจเกิดขึ้นได้ในระหว่างการติดต่อสื่อสารแลกเปลี่ยนข้อมูลแบบขนาน และควรรองรับระบบย่อยขนาดที่ใหญ่ขึ้นได้ คือ $k > \lceil (m+1)/2 \rceil$ โดยไม่มีการขัดแย้งกัน รวมถึงการเลือกใช้เส้นทางที่สั้นที่สุด เพื่อการติดต่อสื่อสารที่มีประสิทธิภาพที่สุดด้วย

งานวิจัยในวิทยานิพนธ์ฉบับนี้ จึงนำเสนอการศึกษาเพื่อแก้ปัญหาการขัดแย้งบนเครือข่าย HHC เพื่อให้สามารถนำเครือข่าย HHC มาใช้ได้จริงในระบบ MPs โดยที่การติดต่อสื่อสารเพื่อแลกเปลี่ยนข้อมูลจะต้องสามารถเลือกใช้เส้นทางที่สั้นที่สุด และประยุกต์ใช้ได้จริงในการดำเนินการแลกเปลี่ยนข้อมูลแบบออล-ทู-ออลเพอร์ซันนัลไลซ์ (All-to-All Personalized Exchange : ATAPE) ซึ่งเป็นฟังก์ชันการติดต่อสื่อสารที่ครอบคลุมเงื่อนไขหลากหลาย ที่สามารถประยุกต์ใช้เพื่อทดสอบการสื่อสารแบบขนานที่ไม่ขัดแย้งในวิทยานิพนธ์ฉบับนี้ ฟังก์ชัน ATAPE คือหนึ่งในการดำเนินการแบบขนานที่มีประสิทธิภาพที่สุดในการสื่อสารแลกเปลี่ยนข้อมูล และถูกนำมาประยุกต์ใช้อย่างกว้างขวางในงานแบบขนานชนิดต่างๆ โดย ATAPE เป็นการติดต่อสื่อสารระหว่างหน่วยประมวลผล (N) ที่ทุกๆ หน่วยประมวลผล สามารถส่งข้อความที่แตกต่างกันไปยังทุกๆหน่วยประมวลผลอื่นๆ ได้พร้อมกัน เป็นจำนวน N รอบ แอพพลิเคชันที่สามารถประยุกต์ใช้ ATAPE ได้โดยตรง เช่น การทรานส์โพสเมตริกซ์แบบขนาน (Parallel Matrix Transposition) การแปลงฟูรีเยร์อย่างเร็วแบบขนาน (Parallel Fast Fourier Transformation) เป็นต้น

ดังนั้นเพื่อจัดการกับการหาเส้นทางการติดต่อสื่อสารแบบ ATAPE ที่ถูกต้องเหมาะสม และเป็นเส้นทางที่สั้นที่สุด ที่จะถูกฝังลงบนชิปของเครือข่าย HHC วิทยานิพนธ์ฉบับนี้ จึงนำเสนอผลการศึกษาวิจัยดังนี้

1. การหาเส้นทางการสื่อสารระหว่างหน่วยประมวลผลแบบหนึ่งต่อหนึ่งที่สั้นที่สุด จากต้นทาง S สู่ปลายทาง D ใดๆ โดยใช้การเข้าคู่แบบเกรย์-โค๊ด (Gray-code Mapping) และการจัดลำดับบิต (Bit-Reordering) บนเครือข่าย HHC

2. การฝังฟังก์ชัน ATAPE แบบขนานลงบนชิปด้วยเส้นทางที่สั้นที่สุด และไม่มีการขัดแย้ง บนเครือข่าย HHC โดยใช้ฟังก์ชัน $D_j^{C_i} = d_{i,j} = S_{C_i \oplus j}$ สำหรับหน่วยประมวลผล $k \leq 2^n/2^{M-1}$ เมื่อ $M = 2^{m-1}$ และ $n = 2^m+m$

3. การพาร์ทิชันเครือข่าย HHC โดยใช้วิธีที่เรียกว่า การจัดกลุ่มดูอัล-คิวบ์แบบไขว้ (Grouping of Cross Dual-cube : GCD) ที่สามารถหลีกเลี่ยงการขัดแย้ง ในขณะสื่อสารระหว่างประมวลผลแบบขนาน ยิ่งไปกว่านั้นภายใต้การพาร์ทิชันด้วยวิธี GCD แต่ละกลุ่มย่อย (Cross Dual-cube : Cross) ที่มีหน่วยประมวลผลจำนวน $k = 2^{m+1}$ สามารถดำเนินการ ATAPE บนเครือข่าย HHC ได้พร้อมกันทั้งหมด โดยไม่มีการขัดแย้งกัน

4. การรวมกลุ่มของหน่วยประมวลผลที่มีขนาด $2^{m+1} < k \leq 2^n/2^{M-1}$ โดยใช้วิธีที่เรียกว่า การรวมกลุ่มคิวบ์ย่อยแบบไขว้ (Grouping of Cross Sub-cube : GCS) เพื่อสื่อสารแบบขนานด้วยหน่วยประมวลผลที่มีขนาดใหญ่ขึ้น

5. การจัดสรรหน่วยประมวลผลลงบนโครงสร้างต้นไม้ที่ปรับเปลี่ยนได้ (Reconfigurable Tree-structure)

1.2 วัตถุประสงค์ของงานวิจัย

วิทยานิพนธ์นี้ มีวัตถุประสงค์เพื่อศึกษาและออกแบบเส้นทางการติดต่อสื่อสารระหว่างหน่วยประมวลผล บนเครือข่าย HHC ให้สื่อสารได้โดยไม่มีการขัดแย้งกัน (หรือไม่มีการชนกันของข้อมูล) และเป็นเส้นทางที่สั้นที่สุด โดยมีรายละเอียด ดังนี้

1) เพื่อออกแบบการสื่อสารด้วยเส้นทางที่สั้นที่สุด (Shortest Path Routing) ระหว่างหน่วยประมวลผลจากต้นทาง S ไปยังปลายทาง D แบบยืดหยุ่น คือ มีค่า S เป็นโหนดใดก็ได้ (Arbitrary Source)

2) เพื่อออกแบบการสื่อสารด้วยเส้นทางที่สั้นที่สุดแบบขนาน (Parallel Shortest Path Routing) ที่ไม่มีการขัดแย้งกันระหว่างหน่วยประมวลผล (S, D) หลายคู่ ขนาด $k = 2^m$

3) เพื่อออกแบบการจัดกลุ่มของหน่วยประมวล (Group of Partitioning) ที่มีขนาดใหญ่ขึ้น คือ $k \leq 2^{m+1}$ เพื่อใช้สำหรับการแลกเปลี่ยนข้อมูลแบบ ATAPE โดยไม่มีการขัดแย้งกัน

4) เพื่อออกแบบฟังก์ชันสำหรับการแลกเปลี่ยนข้อมูลแบบ ATAPE และฝังลงบนชิปของเครือข่าย HHC

5) เพื่อจัดสรรหน่วยประมวลผลลงบนโครงสร้างต้นไม้ที่ปรับเปลี่ยนได้ (Reconfigurable Tree Structure)

1.3 ขอบเขตของงานวิจัย

1) ศึกษาฟังก์ชันการสื่อสารระหว่างหน่วยประมวลผล บนเครือข่าย HHC ได้แก่ ฟังก์ชันการหา k เส้นที่แตกต่างกัน จากหนึ่งต้นทางและปลายทาง (Node Disjoint Paths), การหาเส้นทาง k เส้นที่ต่างกันของหนึ่งต้นทางสู่หลายปลายทาง (Node-to-set Disjoint Paths) และ การหาเส้นทาง k เส้นที่ต่างกันของหลายต้นทางสู่หลายปลายทาง (Set-to-Set Disjoint Paths) มาเป็นต้นแบบ

2) ออกแบบการจัดกลุ่มของหน่วยประมวลจำนวน $k = 2^{m+1}$ หน่วยประมวลผล และการรวมกลุ่ม $2^{m+2} \leq k \leq 2^n/2^{M-1}$ สำหรับการแลกเปลี่ยนข้อมูล ATAPE แบบขนาน ที่ไม่มีการขัดแย้งกัน โดยหน่วยประมวลผล (k) ที่อยู่ในกลุ่ม (Group) เดียวกัน ต้องเริ่มประมวลผลพร้อมกัน และหน่วยประมวลผลที่อยู่คนละกลุ่ม สามารถเริ่มประมวลผลได้อย่างอิสระ (ไม่ต้องเริ่มพร้อมกัน)

3) ศึกษาฟังก์ชัน ATAPE แบบฝังบนชิปของเครือข่ายต่างๆ ได้แก่ เครือข่ายไฮเปอร์คิวบ์, เครือข่ายดราก้อนฟราย (Dragonfly Networks) และ เครือข่ายการเชื่อมต่อแบบมัลติสเตจ (Multistage Interconnection Networks : MINs) เพื่อเป็นต้นแบบในการออกแบบ ATAPE แบบใหม่ที่ประยุกต์ใช้ได้จริงบนเครือข่าย HHC

1.4 ประโยชน์ที่คาดว่าจะได้รับ

ประโยชน์ที่คาดว่าจะได้รับจากงานวิจัยนี้มีดังนี้

1) ได้วิธีการสื่อสารเพื่อแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผลที่เป็นเส้นทางที่สั้นที่สุดแบบแบบหนึ่งต่อหนึ่ง บนเครือข่าย HHC โดยสามารถกำหนดหน่วยประมวลผลต้นทาง และหน่วยประมวลผลปลายทางได้อย่างอิสระ

2) ได้การสื่อสารระหว่างหน่วยประมวลผลที่สั้นที่สุดแบบขนาน ขนาด $k \leq 2^m$ บนเครือข่าย HHC ที่ไม่มีการขัดแย้งกัน เมื่อ $0 \leq S, D \leq N-1$

3) ได้ฟังก์ชันสำหรับการแลกเปลี่ยนข้อมูลแบบ ATAPE ที่สามารถฝังลงบนชิปของเครือข่าย HHC ได้อย่างเต็มประสิทธิภาพและไม่มีการขัดแย้งกัน โดยจัดกลุ่มของหน่วยประมวลผลจำนวน $k \leq 2^n/2^{M-1}$ หน่วยประมวลผล และหน่วยประมวลผลในกลุ่มเดียวกันต้องเริ่มประมวลผลพร้อมกัน ขณะที่หน่วยประมวลผลคนละกลุ่ม สามารถเริ่มประมวลผลได้อย่างอิสระ

4) ได้ต้นแบบของระบบคอมพิวเตอร์แบบขนานในอนาคต ด้วยการฝังฟังก์ชัน ATAPE ที่สามารถทำการสื่อสารระหว่างหน่วยประมวลผล ด้วยเส้นทางที่สั้นที่สุด และไม่มีการขัดแย้งกัน

บทที่ 2
ทฤษฎี และงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงทฤษฎีที่เกี่ยวข้องกับชนิดของเครือข่ายการเชื่อมต่อสำหรับคอมพิวเตอร์แบบขนาน โดยในปัจจุบันสามารถแบ่งออกเป็น 2 ชนิด คือ เครือข่ายการเชื่อมต่อแบบสแตติก (Static Interconnection Networks) และเครือข่ายการเชื่อมต่อแบบไดนามิค (Dynamic Interconnection Networks) โดยเครือข่ายการเชื่อมต่อแบบสแตติก คือ เครือข่ายที่มีโครงสร้างถาวร ซึ่งจะไม่สามารถปรับเปลี่ยนเส้นเชื่อมระหว่างหน่วยประมวลผลได้ (ในวิทยานิพนธ์ฉบับนี้จะเรียกหน่วยประมวลผลว่า “โหนด”) ดังนั้นการเชื่อมต่อแบบสแตติก จะมีต้นทุนในการสร้างเครือข่ายที่ค่อนข้างถูก และมีโครงสร้างที่หลากหลาย เช่น โครงสร้างแบบเส้นตรง (Linear Structure), โครงสร้างแบบเมตริกซ์ (Matrix Structure), โครงสร้างแบบวงแหวน (Ring Structure), โครงสร้างแบบไฮเปอร์คิวบ์ (Hypercube Structure) และ โครงสร้างแบบต้นไม้ (Tree Structure) ในขณะที่เครือข่ายการเชื่อมต่อแบบไดนามิค คือ เครือข่ายที่มีความยืดหยุ่น เนื่องจากสามารถปรับเปลี่ยนรูปแบบของโครงสร้างในการติดต่อสื่อสารได้ ดังนั้นเครือข่ายแบบไดนามิค จะมีต้นทุนการสร้างที่สูงและมีความซับซ้อน ตัวอย่างเช่น เครือข่ายที่ใช้เส้นทางแบบดิจิทัล (Digital Bus Networks) และเครือข่ายการเชื่อมต่อแบบมัลติสเตจ (Multistage Interconnection Networks) ในวิทยานิพนธ์ฉบับนี้ นำเสนอเครือข่ายการเชื่อมต่อแบบสแตติกชนิดหนึ่ง ซึ่งก็คือเครือข่ายการเชื่อมต่อแบบชั้น โดยนำเสนอในหัวข้อที่ 2.1 และ ในหัวข้อที่ 2.2 จะกล่าวถึงขั้นตอนวิธีการสื่อสารเพื่อเปลี่ยนแบบเพอร์ซันนัลไลซ์ออล-ทู-ออล (All-to-All Personalized Exchange : ATAPE)

2.1 เครือข่ายการเชื่อมต่อแบบชั้น (Hierarchical Interconnection Networks: HINs)

เครือข่าย HINs จัดอยู่ในประเภทการเชื่อมต่อแบบสแตติก ซึ่งได้รับความสนใจจากนักวิจัยเป็นอย่างมากในปัจจุบัน จุดเด่นของเครือข่าย HINs คือ การลดต้นทุนในการสร้างให้ถูกลงโดยการออกแบบเครือข่ายที่ลดจำนวนเส้นเชื่อมลง และยังคงมีความเหมาะสมกับการนำมาประยุกต์ใช้ในการสื่อสารแบบขนาน เนื่องจากโครงสร้างของเครือข่าย HINs มีลักษณะเป็นชั้นๆ โดยที่ชั้นล่างสุดจะเป็นการสื่อสารระหว่าง “โหนดชั้นใน” (Internal Nodes) ขณะที่ชั้นบนสุดจะเป็นการจัดกลุ่มของ “โหนดชั้นนอก” (External Nodes) เพื่อเชื่อมต่อโหนดชั้นในเข้าไว้ด้วยกัน โดยใช้การทำซ้ำ (Replication) ของโหนดและเส้นเชื่อม ต่อไปจะเป็นนิยามที่จะถูกใช้ในวิทยานิพนธ์ฉบับนี้

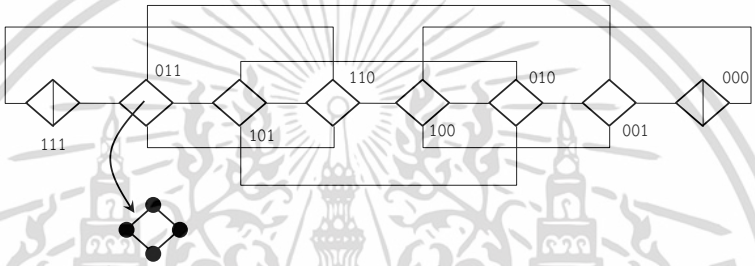
นิยามที่ 1 *ดีกรีของระบบ* (Network Degree) หมายถึง จำนวนเส้นเชื่อมที่มากที่สุดต่อหนึ่งโหนดเมื่อเทียบกับทุกๆ โหนดในระบบ

นิยามที่ 2 *เส้นผ่านศูนย์กลางของระบบ* (Network Diameter) หมายถึง ระยะทางที่สั้นที่สุดระหว่างสองโหนดใดๆ ที่มีระยะห่างไกลกันมากที่สุดในระบบ

การออกแบบเครือข่าย HINs โดยการใช้หลักการทำซ้ำ (Replication Techniques) จะถูกนำเสนอในหัวข้อต่อไปนี้ในหัวข้อ 2.1.1 – 2.1.5

2.1.1 เครือข่าย deBruijn-Cube (DeBruijn-Cube Networks : dBCube)

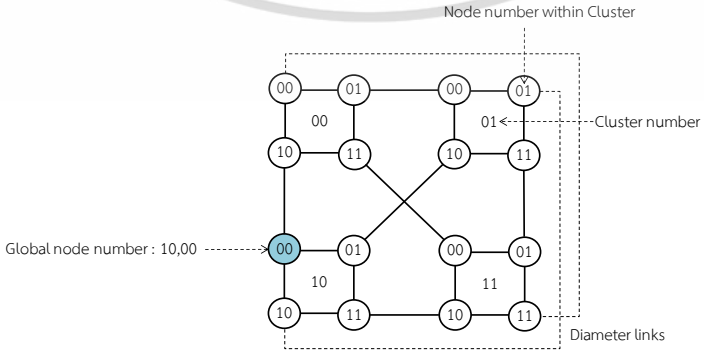
dBCube(*c*, *d*) คือ การออกแบบโครงสร้างเครือข่ายที่เป็นกราฟผสม (Compound Graph) ซึ่งประกอบด้วยกราฟ deBruijn ที่แต่ละโหนดถูกแทนด้วยไฮเปอร์คิวบ์ [6] โดยการมี *c* คิวบ์ ต่อ 1 คลัสเตอร์ (Cluster) แต่ละคิวบ์จะมี *d* มิติ รูปที่ 2.1 แสดง dBCube(8, 2) ที่แต่ละโหนดถูกสร้างด้วย 2 คิวบ์ หรือ $d = 2$ และจะมีจำนวนคิวบ์ทั้งหมด 8 คิวบ์ ต่อ คลัสเตอร์ หรือ $c = 8$ ในขณะที่ dBCube(*c*, *d*) มีดีกรีของระบบ คือ $d + 1$ และเส้นผ่านศูนย์กลางของระบบ คือ $1 + \log(c)$



รูปที่ 2.1 ตัวอย่างเครือข่ายการเชื่อมต่อ dBCube (8, 2)

2.1.2 เครือข่ายลูกบาศก์แบบชั้น (Hierarchical Cubic Networks :HCN)

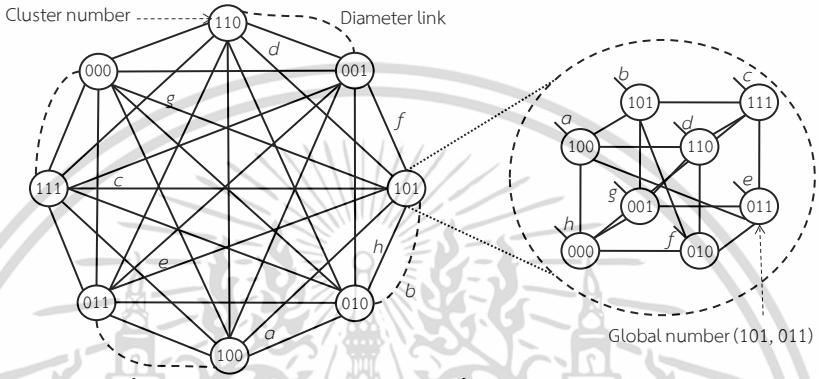
เครือข่ายลูกบาศก์แบบชั้น หรือ HCN(*n*, *n*) [7] คือ เครือข่ายแบบชั้นที่ประกอบด้วย 2^n คลัสเตอร์ ที่แต่ละคลัสเตอร์ประกอบด้วย *n* มิติของไฮเปอร์คิวบ์ แต่ละโหนดบนเครือข่าย HCN จะมีดีกรีของระบบ คือ $n + 1$ และมีเส้นผ่านศูนย์กลางของระบบ คือ $n + \lfloor (n+1)/3 \rfloor + 1$ สังเกตว่า HCN จะมีเส้นเชื่อมน้อยกว่าเกือบครึ่งหนึ่งของไฮเพอร์คิวบ์ ขณะที่มีขนาดของเส้นผ่านศูนย์กลางที่เล็กกว่า ตัวอย่างของ HCN(2, 2) แสดงในรูปที่ 2.2 โดยแต่ละโหนดจะถูกแทนด้วย n บิต และ ถูกอ้างถึงโดยการใช้บิตของคลัสเตอร์ตามด้วยบิตของโหนดที่ต้องการอ้างถึง เช่น 10, 00 จะหมายถึงโหนดสีฟ้าในรูปที่ 2.2



รูปที่ 2.2 ตัวอย่างเครือข่ายการเชื่อมต่อ HCN (2, 2)

2.1.3 เครือข่ายโฟลดคิวบ์แบบชั้น (Hierarchical Folded Cube : HFCube)

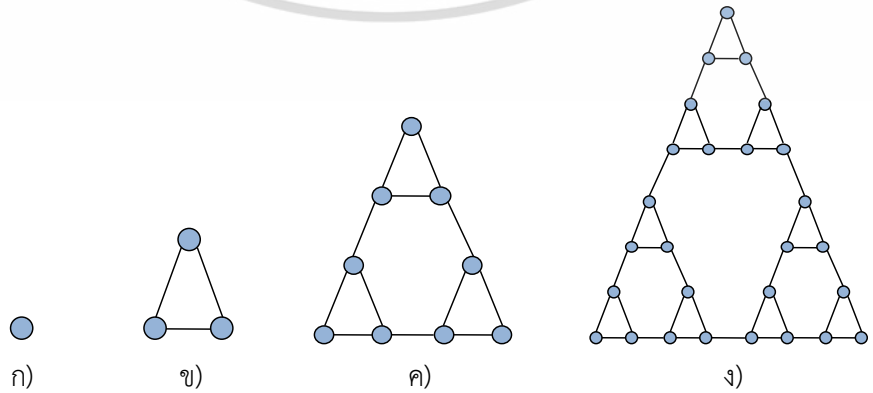
HFCube คือ เครือข่ายการเชื่อมต่อแบบชั้นที่ใช้โฟลดไฮเปอร์คิวบ์เป็นคลัสเตอร์ และถูกแทนด้วยโครงสร้างโฟลดคิวบ์อีกครั้งหนึ่ง HFCube(*n*, *n*) ถูกเสนอโดย Y.Shi และคณะ [19] ในปี 2000 โครงสร้างของ HFCube ประกอบด้วย 2^n คลัสเตอร์ แต่ละคลัสเตอร์จะประกอบด้วยโฟลดไฮเปอร์คิวบ์ (Folded Hypercube : FHC) หรือ FHC(*n*) แต่ละโหนดใน HFCube(*n*, *n*) มีจำนวนดีกรีของระบบ คือ *n* + 2 ขณะที่เส้นผ่านศูนย์กลางของระบบ คือ *n* + 1 ตัวอย่างของ HFCube(3, 3) แสดงให้เห็นในรูปที่ 2.3



รูปที่ 2.3 ตัวอย่างเครือข่ายการเชื่อมต่อ HFCube (3, 3)

2.1.4 เครือข่ายทริปเปิล-เบสแบบชั้น (Triple-based HINs :THIN)

เครือข่าย THIN ถูกเสนอโดย B. Qiao และคณะ [15] ในปี 2007 รูปที่ 2.4 แสดงตัวอย่างของเครือข่าย THIN ในระดับต่างๆ โดยที่ ก) แสดงระดับที่ 0 THIN, ข) แสดงระดับที่ 1 THIN, ค) แสดงระดับที่ 2 THIN และ ง) แสดงระดับที่ 3 THIN โดยในระดับที่ 1 THIN คือ ส่วนประกอบพื้นฐานที่สามารถนำมาสร้างเป็นระดับอื่นๆ ต่อไป ดังนั้น THIN สามารถขยายขนาดได้ง่าย โดยนำระดับที่ 1 ไปเป็นส่วนประกอบของระดับที่ *k*-1 เพื่อที่จะสร้างเป็นระดับที่ *k* จากรูปจะเห็นได้ว่าโครงสร้างของ THIN เป็นโครงสร้างที่ไม่มีความซับซ้อน เนื่องจากมี ดีกรีของระบบ คือ ค่าคงที่, มีความสมมาตร และมีความยืดหยุ่นที่ดี โดยดีกรีและเส้นผ่านศูนย์กลางของระบบ THIN คือ 3 และ $2^{\log_3 N-1}$ ตามลำดับ เมื่อ *N* คือ จำนวนโหนดทั้งหมดในระบบ



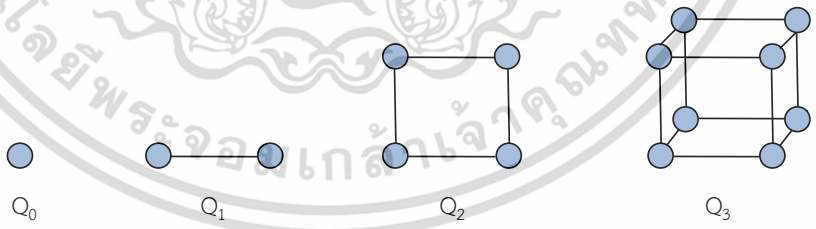
รูปที่ 2.4 โครงสร้างของเครือข่าย THIN ก) ระดับที่ 0, ข) ระดับที่ 1 ถึง ง) ระดับที่ 3

2.1.5 เครือข่ายไฮเปอร์คิวบ์แบบชั้น (Hierarchical Hypercube Networks : HHCs)

เครือข่าย HHC ถูกเสนอขึ้นครั้งแรกในปี 1994 โดย Malluhi และ Bayoumi [11] โดยถูกเสนอขึ้นมาเพื่อเป้าหมายในการลดต้นทุน (Cost) ในการสร้างเครือข่าย โดยการลดจำนวนเส้นเชื่อมของระบบ โครงสร้างของเครือข่าย HHC ประกอบด้วย 2 ระดับ ซึ่งระดับบนสุด หรือคลัสเตอร์ จะใช้โครงสร้างของไฮเปอร์คิวบ์ และจะแทนที่โหนดระดับบนสุดด้วยโครงสร้างของไฮเปอร์คิวบ์อีกครั้งหนึ่ง ดังนั้นเครือข่าย HHC นอกจากจะถูกออกแบบมาเพื่อลดต้นทุนในการสร้างแล้ว ยังสืบทอดคุณสมบัติสำคัญๆของเครือข่ายไฮเปอร์คิวบ์ (Hypercube Network : HC) เช่น ประสิทธิภาพในการสื่อสารและความสมมาตรของเครือข่าย เพื่อทำให้รู้โครงสร้างและหลักการทำงานของเครือข่าย HHC การอธิบายต่อไปจะเป็นการอธิบายนิยามของเครือข่าย HC และ เครือข่ายHHC ในนิยามที่ 3 และ 4 ตามลำดับ

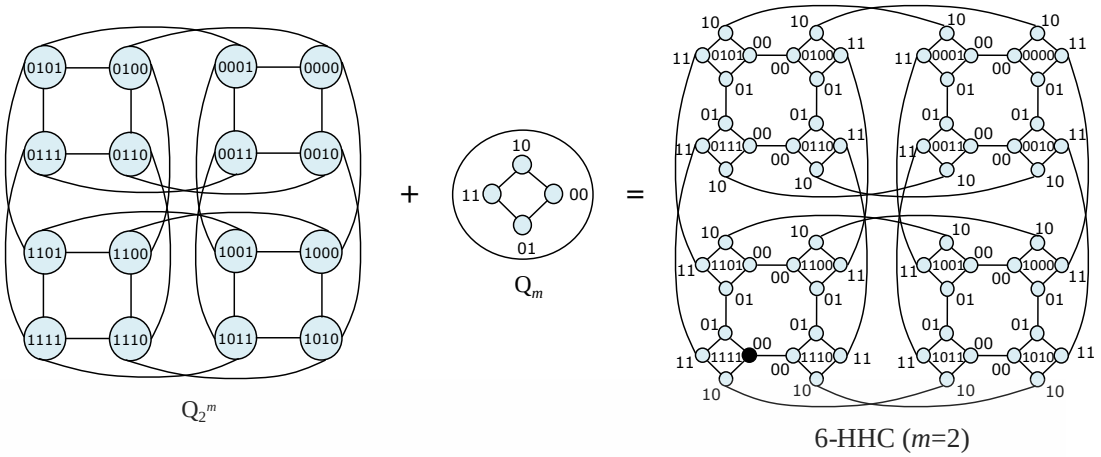
นิยามที่ 3 เครือข่าย n–ไฮเปอร์คิวบ์ (Hypercube : HC) หรือ Q_n [17] จะประกอบด้วย 2^n โหนด ($N = 2^n$) และแต่ละโหนดจะถูกแทนด้วยเลขฐานสองที่มีขนาด n บิต ดังนี้ $(b_{n-1}\ b_{n-2}\ \ldots\ b_1\ b_0)$ เมื่อ $b_i \in \{0, 1\}$ และ $0 \le i \le n-1$ และกำหนดให้สองโหนดใดๆ จะอยู่ติดกันก็ต่อเมื่อทั้งสองโหนดมีบิตที่ต่าง 1 บิต หรือ มีระยะทางแฮมมิ่ง (Hamming Distance) ระหว่างสองโหนด คือ 1

ในทางทฤษฎีแล้วประสิทธิภาพในการสื่อสารคือจุดเด่นอย่างหนึ่งที่สำคัญของเครือข่ายไฮเปอร์คิวบ์ แต่เมื่อมีการขยายขนาดของเครือข่ายให้มีขนาดใหญ่ขึ้น (Larger Multi-Processors : MPs) จะทำให้มีข้อจำกัดเกิดขึ้น เช่น ไม่มีความยืดหยุ่น และมีต้นทุนในการสร้างสูง ดังนั้นเครือข่าย HHC ($N = 2^n$, $n = 2^m+m$) จึงถูกเสนอมา โดยยังคงคุณสมบัติที่ดีของเครือข่าย HC ไว้ ขณะที่ยังสามารถลดต้นทุนในการสร้าง และรองรับความยืดหยุ่นในการขยายขนาดได้อย่างมีประสิทธิภาพ



รูปที่ 2.5 โครงสร้างของเครือข่าย HC หรือ Q_n เมื่อ n = 0 - 3

นิยามที่ 4 เครือข่ายไฮเปอร์คิวบ์แบบชั้น (n-HHC) ที่มีขนาด $N = 2^n$ เมื่อ $n = 2^m+m$ [11] จะเริ่มต้นโครงสร้าง โดยการสร้างคลัสเตอร์ (Clusters) ด้วยโครงสร้างของ 2^m–ไฮเปอร์คิวบ์ (Q_{2m}) และจะถูกเรียกว่า เครือข่ายหลัก (Main–Network) ขั้นตอนต่อไปทำการแทนที่แต่ละโหนดของ Q_{2m} ด้วย m–ไฮเปอร์คิวบ์ โดยจะเรียกว่าเครือข่ายรอง (Sub-Network) ดังนั้นจำนวนหน่วยประมวลผลของเครือข่าย HHC คือ $2^{2^m} \times 2^m = 2^{2^m+m}$ โหนด โดยแต่ละโหนดจะถูกแทนด้วย n บิตที่ไม่ซ้ำกัน (Unique) ที่ภายใน n บิต จะประกอบด้วย 2^m บิต ของเครือข่ายหลัก และ m บิต ของเครือข่ายรอง



รูปที่ 2.6 โครงสร้างของเครือข่าย 6-HHC ($N = 2^n = 64$, $n = 2^m+m = 6$ และ $m = 2$) จากเครือข่ายหลัก Q_{2^m} และเครือข่ายรอง Q_m

ดังนี้ $\{(\alpha, \beta) \mid \alpha = b_{n-1}\, b_{n-2} \ldots b_m, \beta = b_{m-1}\, b_{m-2} \ldots b_0\}$, เมื่อ $b_i \in \{0, 1\}$, $0 \le i \le n-1$, $n = 2^m+m$ และ $m \ge 2$

รูปที่ 2.6 แสดงโครงสร้างของเครือข่าย 6-HHC เมื่อ $m = 2$ ซึ่งจะได้ว่า $N = 64$, $n = 6$ จากรูปสังเกตว่า โครงสร้างของเครือข่ายหลัก ประกอบด้วย Q_{2^m} หรือ Q_4 และโครงสร้างของเครือข่ายรอง คือ Q_m หรือ Q_2 เมื่อนำเครือข่ายรองเข้าไปแทนที่แต่ละโหนดบนเครือข่ายหลัก จะทำให้ได้โครงสร้างของเครือข่าย HHC

ในการสื่อสารระหว่างสองโหนดใดๆบนเครือข่าย HHC จะถูกเรียกว่าโหนดต้นทาง S (Source) และโหนดปลายทาง D (Destination) โดยกำหนดให้ $S = (\alpha_S, \beta_S)$ และ $D = (\alpha_D, \beta_D)$ เมื่อ α_S และ β_S หมายถึง รหัสเครือข่ายหลัก และเครือข่ายรองของโหนดต้นทาง S ตามลำดับ ซึ่งสำหรับปลายทาง D จะมีลักษณะเดียวกัน เช่น โหนดสีดำ ในรูปที่ 2.6 จะอ้างถึงโดย $(\alpha, \beta) = (1111, 00)$ นอกจากนี้ ระหว่างโหนด S และ D ใดๆบนเครือข่าย HHC จะมีเส้นเชื่อมระหว่างโหนดก็ต่อเมื่อ เข้าเงื่อนไขใดเงื่อนไขหนึ่งจากสองเงื่อนไขที่จะอธิบายต่อไปนี้

เงื่อนไขที่ 1 : $\alpha_S = \alpha_D$ และระยะทางแฮมมิ่ง (Hamming Distance) ของ β_S และ β_D คือ 1 โดย จะถูกเรียกว่า เส้นเชื่อมภายใน (Internal Edge)

เงื่อนไขที่ 2 : $\alpha_S = \alpha_D \oplus 2^{\beta_S}$ และ $\beta_S = \beta_D$ เมื่อ $\oplus$ หมายถึง การดำเนินการ XOR โดยจะถูกเรียกว่า เส้นเชื่อมภายนอก (External Edge)

ตัวอย่างเส้นเชื่อมภายใน เช่น โหนด $S = (\alpha_S, \beta_S) = (0000, 11)$ และโหนด $D = (\alpha_D, \beta_D) = (0000, 01)$ จะเชื่อมกันด้วยเส้นเชื่อมภายใน เนื่องจาก $\alpha_S = \alpha_D = 0000$ และระยะทางแฮมมิ่งของ 11 และ 01 คือ 1

ตัวอย่างเส้นเชื่อมภายนอก เช่น โหนด $S = (\alpha_S, \beta_S) = (0000, 11)$ และโหนด $D = (\alpha_D, \beta_D) = (1000, 11)$ จะเชื่อมกันโดยเส้นเชื่อมภายนอก

เนื่องจาก $\alpha_S = 0000 = 1000 \oplus 2^{11_2}$

$= 1000 \oplus 8_{10}$

$= 1000 \oplus 1000_2$

ดังนั้น $\alpha_S = 0000$

และ $\beta_S = \beta_D = 11$

นิยามที่ 5 กำหนดให้ *m* บิต–เกรย์โค๊ด (*m*-bit Gray Code) คือ ชุดของตัวเลขฐานสองที่แต่ละค่า จะประกอบด้วย *m* หลัก และสองค่าใดๆ ที่อยู่ลำดับติดกันจะมีบิตต่างกันเพียงหนึ่งบิต โดย 1 บิต–เกรย์โค๊ด คือ $G_1 = (0, 1)$ และ G_1^* จะได้จากการเรียงลำดับแบบตรงกันข้ามของ G_m โดยที่ G_m สามารถได้จากการทำซ้ำของรูปแบบ $(0G_{m-1}, 1G_{m-1}^*)$ เพื่อที่จะทำให้สมาชิกที่ติดกันมีบิตต่างเพียง 1 บิต

ตัวอย่างเช่น $G_2 = (\mathbf{0}G_1, \mathbf{1}G_1^*) = (\mathbf{0}0, \mathbf{0}1, \mathbf{1}1, \mathbf{1}0)$

$G_3 = (\mathbf{0}G_2, \mathbf{1}G_2^*) = (\mathbf{0}00, \mathbf{0}01, \mathbf{0}11, \mathbf{0}10, \mathbf{1}10, \mathbf{1}11, \mathbf{1}00)$

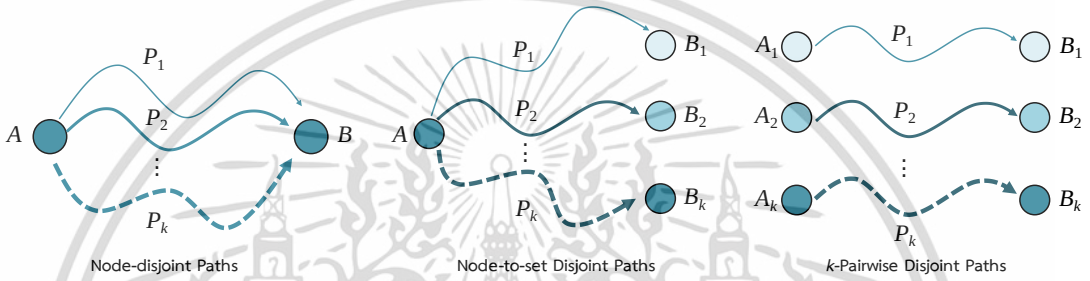
นอกจากนี้ [11] ได้เสนอการหาเส้นทางแบบหนึ่งต่อหนึ่ง ดังแสดงในขั้นตอนวิธีที่ 2.1 ที่เริ่มต้นทางโหนดปัจจุบัน *C* (Current Node) ไปยังโหนดปลายทาง *D* (Destination Node) โดยใช้การหาเส้นทางที่เรียกว่า Scube ร่วมกันกับ *m* บิต–เกรย์โค๊ด แต่เส้นทางที่ได้จะไม่การันตีว่าเป็นเส้นทางที่สั้นที่สุด และเมื่อนำไปใช้ในการหาเส้นทางที่รับส่งข้อมูลพร้อมกันแบบขนานในขนาดงานที่เล็ก เช่น $k \le 2^m$ สามารถดำเนินการได้โดยไม่มีการขัดแย้ง แต่เมื่องานมีขนาดใหญ่ขึ้นเช่น $k \ge 2^m$ จะมีความขัดแย้งเกิดขึ้น

ขั้นตอนวิธี 2.1 ขั้นตอนการหาเส้นทางแบบหนึ่งต่อหนึ่งบนเครือข่าย HHC

```
Node-to-node Routing
1.  if (C ≠ D) then
2.      if (α_C = α_D) then
3.          Scube-routing(β_C, β_D);
4.      else
5.      T = C ⊕ D;
6.       I = {indeces of 1's in α_T};
7.      if (β_C ∈ I ) then
8.          Routing on external edge;
9.      else
10.         Scube-routing(β_C, β_D);
11. end
```

2.1.5.1 การหาเส้นทางเพื่อรับส่งข้อมูลบนเครือข่าย HHC

ในปัจจุบันมีหลายขั้นตอนวิธี (Algorithms) [3, 4, 5, 22] ถูกเสนอเพื่อทำการหาเส้นทางในการติดต่อสื่อสารระหว่างหน่วยประมวลผลบนเครือข่าย HHC โดยจะแตกต่างกันตามวัตถุประสงค์ของเส้นทาง เช่น การหาเส้นทางจำนวน k เส้น จากต้นทาง A ไปยังปลายทาง B (Node-disjoint Paths) ดังรูปที่ 2.7 ก), การหาเส้นทาง k เส้นจากต้นทาง A ไปยังหลายปลายทาง B_i เมื่อ $1 \le i \le k$ (Node-to-set Disjoint Paths) ดังรูปที่ 2.7 ข) และการหาเส้นทาง k เส้นทางจาก k ต้นทางและปลายทาง (k-Pairwise Disjoint Paths) ดังแสดงในรูปที่ 2.7 ค) โดยรายละเอียดของเนื้อหาจะถูกแสดงในหัวข้อต่อไปนี้



ก) ข) ค)

รูปที่ 2.7 รูปแบบของการสื่อสารระหว่างหน่วยประมวลผลในงานวิจัยก่อนหน้า

2.1.5.1.1 การหาเส้นทางจำนวน k เส้น จากต้นทาง A สู่ปลายทาง B (The Node-disjoint Paths)

ในปี 2007 Wu และคณะ [22] ได้เสนอการหาเส้นทาง k เส้นทางที่แตกต่างกันบนเครือข่าย n-HHC ($N = 2^n$ และ $n = 2^m+m$) ดังรูปที่ 2.7 ก) โดยการหาเส้นทางจากต้นทาง $A = (A_{ex}, A_{in})$ ไปสู่ปลายทาง $B = (B_{ex}, B_{in})$ จะประกอบด้วย 2 ขั้นตอน โดยขั้นแรก คือ การหาเส้นทางที่สั้นที่สุดด้านนอก (Shortest External-Path) และขั้นตอนที่สอง คือ การนำผลลัพธ์ที่ได้จากขั้นตอนแรกมาพิจารณาจาก 4 กรณี เพื่อที่จะได้ผลลัพธ์ k เส้นทางที่แตกต่างกันเมื่อ $k \le m+1$ ซึ่งมีขั้นตอนตามขั้นตอนวิธีที่ 2.2

ขั้นตอนวิธี 2.2 แสดงขั้นตอนการหาเส้นทางจำนวน k เส้น จากต้นทาง A ไปยังปลายทาง B ด้วยระยะทางที่ไม่เกินค่าที่มากที่สุดระหว่าง $2^{m+1}+2m+1$ และ $2^{m+1}+m+4$ ซึ่งเส้นทาง k เส้นทางที่ได้จะไม่มีการซ้อนทับเส้นทางกัน หรือ ไม่มีทิศทางที่ไปทางเดียวกัน เนื่องจากต้องการมีเส้นทางอื่นเพื่อรองรับในกรณีที่เส้นทางเดิมไม่สามารถใช้งานได้

นิยามที่ 6 กำหนดให้ โหนด A และ B คือ สองโหนดใดๆ ที่แตกต่างกันบนเครือข่าย HHC และกำหนดให้ลำดับเส้นเชื่อมด้านนอก (External Edge Sequence : EES) คือ เส้นทางด้านนอก หรือเส้นทางบนเครือข่ายหลักของเครือข่าย HHC

ขั้นตอนวิธี 2.2 วิธีการหาเส้นทาง *k* เส้นที่แตกต่างกันจากโหนด A และ B บนเครือข่าย HHC

Node-Disjoint Paths

1. Assume $A = (0^{2^m}, 2^m)$
2. Collect indices i of B_{ex} (External bits of B) with $b_i = 1$;
3. Construct a set π or EES by arranging all indices i into an m-bit Gray-Code;
4. Construction 2 of m+1 paths, depending on four cases;
5. **Case 1:** $A_{in} \in \pi$ and $B_{in} \in \pi$
6. $P_1 = \pi^{c_0}$;
7. $P_2 = \pi^{c_{(z+1) \bmod r}}$;
8. **Case 2:** $A_{in} \notin \pi$ and $B_{in} \in \pi$
9. $P_1 = (0^m, \pi, 0^m)$;
10. $P_2 = \pi^{c_{(z+1) \bmod r}}$;
11. **Case 3:** $A_{in} \in \pi$ and $B_{in} \notin \pi$
12. $P_1 = (\theta^{B_{in}}, B_{in})$;
13. $P_2 = \pi^{c_0}$;
14. **Case 4:** $A_{in} \notin \pi$ and $B_{in} \notin \pi$
15. $P_1 = (0^m, \pi, 0^m)$;
16. $P_2 = (\theta^{B_{in}}, B_{in})$;
17. Other m-1 paths are obtained according to the first m-1 unused EES of
18. $\pi^{c_0}, \pi^{c_1}, \ldots, \pi^{c_{r-1}}$;

รูปที่ 2.8 แสดงตัวอย่างเส้นทางบนเครือข่าย 6–HHC เมื่อ $N = 2^n = 64$ และ $n = 2^m+m = 6$ และ $m = 2$ จากต้นทาง $A = 0$ หรือ $(A_{ex}, A_{in}) = (0000, 00)$ ไปยังปลายทาง $B = 49$ หรือ $(B_{ex}, B_{in}) = (1100, 01)$

กำหนดให้ $\longrightarrow$ คือ เส้นทางด้านนอก หรือเส้นทางบนเครือข่ายหลัก (External Edge)

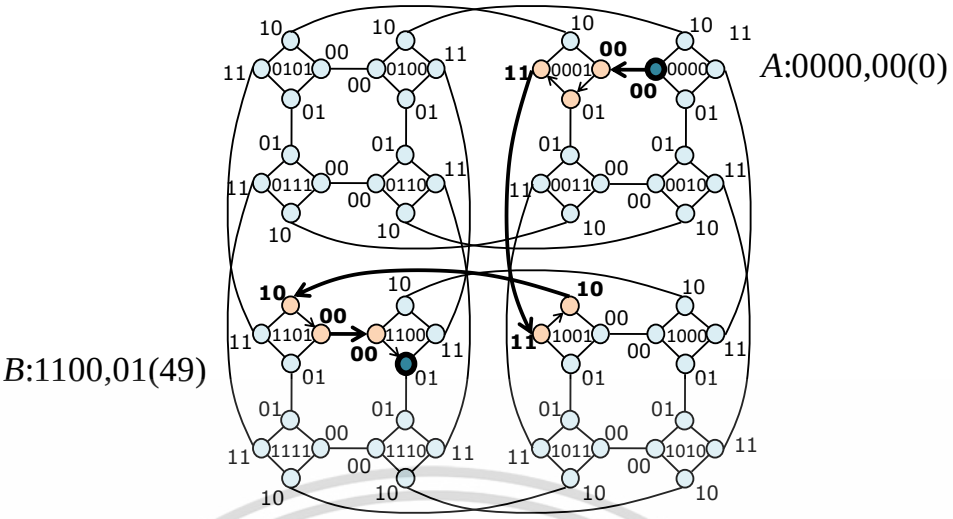
$\overset{*}{\longrightarrow}$ คือ เส้นทางด้านใน หรือ เส้นทางบนเครือข่ายรอง (Internal Edge)

และจากต้นทาง และปลายทางดังกล่าว การหาเส้นทางด้วยขั้นตอนวิธีที่ 2.2 จะได้ EES = (11, 10) เนื่องจาก B_{ex} = 1100 มีบิต 1 ในอินเด็กซ์ที่ 2 และ 3 และจะได้เส้นทางหนึ่งจากทั้งหมด k เส้นทาง โดยใช้เส้นทาง P_1 ที่ตกอยู่ในกรณีที่ 4 (Case 4) ซึ่งจะได้เส้นทางดังต่อไปนี้

$$
\begin{aligned}
A\text{: } (0000, 00) &\longrightarrow (0001, 00) \overset{*}{\longrightarrow} (0001, 01) \\
&\overset{*}{\longrightarrow} (0001, 11) \longrightarrow (1001, 11) \\
&\overset{*}{\longrightarrow} (1001, 10) \longrightarrow (1101, 10) \\
&\overset{*}{\longrightarrow} (1101, 00) \longrightarrow (1100, 00) \overset{*}{\longrightarrow} (1100, 01)\text{: } B
\end{aligned}
$$

ซึ่งมีระยะทางเท่ากับ 9 (Lengths) และผ่านโหนดทั้งหมด 10 โหนด ดังแสดงให้เห็นรูปที่ 2.8

อย่างไรก็ตาม ขั้นตอนวิธีที่ 2.2 ใช้งานได้ก็ต่อเมื่อ ต้นทาง $A = 0$ เท่านั้น จึงไม่เหมาะสมในการนำไปสร้างเครือข่ายที่ใช้ได้จริง



รูปที่ 2.8 ตัวอย่างของการหาเส้นทาง ด้วยขั้นตอนวิธี 2.2 จากต้นทาง A = 0000,00 สู่ปลายทาง B = 1100,01 บนเครือข่าย 6–HHC

2.1.5.1.2 การหาเส้นทางที่แตกต่างกันแบบหนึ่งต้นทาง ไปยัง k ปลายทาง (Node-to-set Disjoint Paths)

ในปี 2011 โดย Bossard และคณะ [5] นำเสนอวิธีการหาเส้นทาง k เส้นจากต้นทาง A ไปยังหลายปลายทาง B_i บนเครือข่าย HHC เมื่อ $1 \le i \le k$ หรือเรียกอีกอย่างว่า การกระจายไปยังโหนดย่อย (Subset Broadcasting) ขั้นตอนที่มีประสิทธิภาพจากขั้นตอนวิธีนี้ คือ การลดขนาดของปัญหาจากหนึ่งโหนดไปยัง k ปลายทางของระยะทางทั้งหมด ไปยังการมองปัญหาให้เล็กลง โดยทำให้เป็นปัญหาระดับเครือข่ายย่อย ($N' = 2^m$) โดยขั้นตอนวิธี 2.3 อธิบายขั้นตอนวิธีหลัก ที่ประกอบด้วย 4 ขั้นตอนเพื่อหาเส้นทาง k เส้นทาง โดยที่แต่ละเส้นทางจะมีความยาวไม่เกิน $m2^m+2^m+2m+4$ ภายในเวลา $O(km2^m)$ เมื่อ $1 \le i \le k$ และ $k \le m+1$

ขั้นตอนวิธี 2.3 วิธีการหาเส้นทาง k เส้นที่แตกต่างกันจากโหนด A และ B บนเครือข่าย HHC

```
HHC-N2S
1. if ( |D∩Q_m(s_0)| ≥ k − 1) then
2.         Solving the N2S routing in a Q_m by Cube-N2S;
3. else
4.         Step1. “Preprocessing” distribute all D to distinct subcubes;
5.         Step2. “Hypercube routing” apply Cube-N2S onto Hypercube;
6.         Step3. “Path discarding” remove the unnecessary paths from Step2;
7.         Step4. “Subcube routing” convert cube-level paths back to
8.                 HHC –level paths;
```

2.1.5.1.3 การหาเส้นทางที่แตกต่างกัน k เส้นทาง จาก k ต้นทาง และปลายทาง (Node-to-set Disjoint Paths)

ในปี 2011 Bossard และคณะ [3] ได้เสนอการหาเส้นทาง k เส้นทางจากต้นทาง S_i สู่ปลายทาง D_i บนเครือข่าย HHC เมื่อ $0 \le i \le k$ และ $k \le \lceil (m+1)/2 \rceil$ แสดงให้เห็นในรูปที่ 2.7 ค) โดยขั้นตอนวิธี 2.4 อธิบายถึงขั้นตอนหลักในการหาเส้นทาง k เส้น จากต้นทาง S_i สู่ปลายทาง D_i โดยระยะทางของแต่ละเส้นทางจะถูกคำนวณภายใน $O(2^{5m})$ และมีความยาวไม่เกิน $2^{m+1}+m(2^{m+1}+1)+4$ อย่างไรก็ตามในการดำเนินการแบบขนานแต่ละ k เส้นทางอาจจะไม่การันตีว่าเป็นเส้นทางที่สั้นที่สุดและเส้นทาง จำนวน k เส้น ยังมีจำนวนน้อยเกินไปเมื่อนำมาสร้างเครือข่าย

ขั้นตอนวิธี 2.4 วิธีการหาเส้นทางที่แตกต่างกัน k เส้นทาง จาก k ต้นทาง และปลายทาง บนเครือข่าย HHC

***k*-pairewise**

1. **For all** external nodes with $|T(\sigma)| \ge 2$ **do**
2. Spreading all nodes of T;
3. P = Generates disjoint-paths in Q_{2m};
4. **For all** $p \in P$ **do**
5. R = Generates disjoint-paths in HHC;
6. **return** the path R;

2.2 การแลกเปลี่ยนข้อมูลแบบเพอร์ซันนัลไลซ์ ออล-ทู-ออล (All-to-All Personalized Exchange)

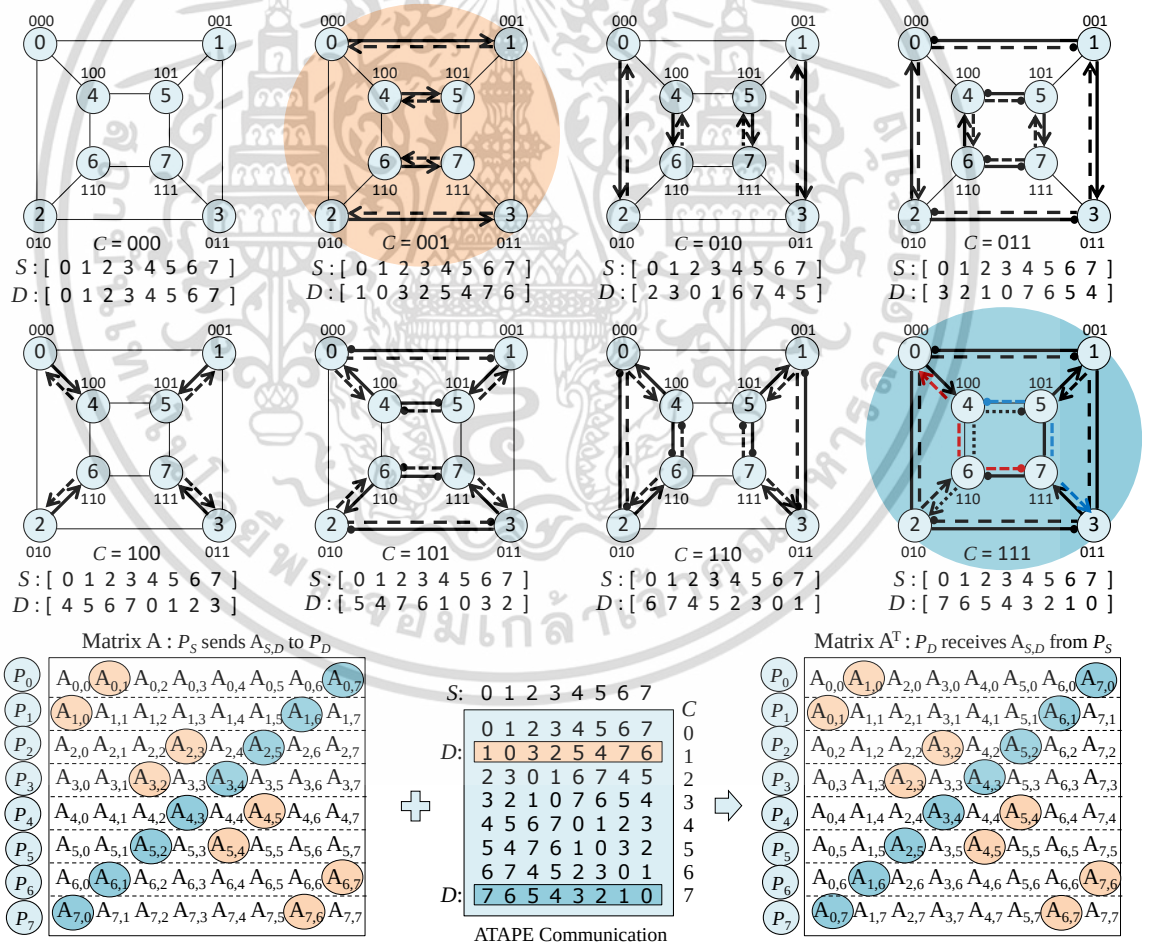
ในระบบคอมพิวเตอร์แบบขนาน และแบบกระจาย (Parallel and Distributed Computers) มีรูปแบบของการติดต่อสื่อสารเพื่อแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผลหลายรูปแบบ เช่น การติดต่อสื่อสารแบบหนึ่งต่อหนึ่ง (One–to–One), การติดต่อสื่อสารแบบหนึ่งต่อกลุ่ม (One–to–Many) และการติดต่อแบบทุกหน่วยประมวลผลกับทุกหน่วยประมวลผล หรือ ออล-ทู-ออล (All–to–All) ซึ่งในวิทยานิพนธ์ฉบับนี้จะใช้การติดต่อแบบออล-ทู-ออล เพื่อแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผล

การติดต่อแบบออล–ทู–ออล สามารถแบ่งตามลักษณะการสื่อสารออกเป็น 2 ประเภท คือ การกระจายข้อมูลแบบออล–ทู–ออล (All–to–All Broadcast : ATAB) และการแลกเปลี่ยนแบบเพอร์ซันนัลไลซ์ ออล–ทู–ออล (All–to–All Personalized Exchange : ATAPE) โดย ATAB คือ การติดต่อสื่อสารโดยส่งข้อความเดียวกันระหว่างหน่วยประมวลผล N หน่วยประมวลผล ขณะที่ ATAPE คือ ทุกๆหน่วยประมวลผลส่งข้อความที่แตกต่างกัน N ข้อความไปยังหน่วยประมวลผลอื่นๆ จำนวน N รอบ ซึ่งในปัจจุบันมีการนำ ATAPE ไปประยุกต์ใช้กับการติดต่อเพื่อแลกเปลี่ยนข้อมูล เช่น การแปลงฟูเรียร์แบบขนาน และการทรานสโพสต์เมตริกซ์แบบขนาน ดังแสดงรูปที่ 2.9

D. Scott ได้เสนอการติดต่อสื่อสารแบบ XOR-ATAPE ในปี 1991 [18] ดังแสดงในสมการ 2.1 โดยเป็นการติดต่อสื่อสารระหว่างต้นทาง S (Source) จำนวน N หน่วย ไปยังปลายทาง D (Destination) จำนวน N หน่วย บนเครือข่าย HC โดยแต่ละรอบจะถูกควบคุมด้วยตัวควบคุม C (Control Units)

$$D = S \oplus C \tag{2.1}$$

รูปที่ 2.9 แสดงตัวอย่างของการติดต่อสื่อสารแบบ ATAPE พร้อมทั้งแสดงการประยุกต์ใช้กับการทรานสโพสเมตริกซ์แบบขนาน บนเครือข่าย HC ภายในเวลา $O(N+\log_2 N)$ จากรูปกำหนดให้ต้นทาง S = 0, 1, ..., 7 และเมื่อใช้สมการที่ (2.1) ในแต่ละรอบของตัวควบคุม (C) จะทำให้ได้ตารางลาติน ซึ่งตารางลาติน คือ ตารางที่เก็บปลายทาง (D) ของการติดต่อสื่อสารในแต่ละรอบ จากต้นทาง S ไปยังปลายทาง D เช่น เมื่อ C = 7 จะมีการแลกเปลี่ยนของ S และ D ดังต่อไปนี้ $P_0 \longrightarrow P_7$, $P_1 \longrightarrow P_6$, $P_2 \longrightarrow P_5$, $P_3 \longrightarrow P_4$, $P_4 \longrightarrow P_3$, $P_5 \longrightarrow P_2$, $P_6 \longrightarrow P_1$, $P_7 \longrightarrow P_0$



รูปที่ 2.9 ตัวอย่าง ATAPE โดยใช้สมการ $D = S \oplus C$ สำหรับเมตริกซ์ทรานสโพส บนเครือข่ายไฮเปอร์คิวบ์ ที่มีขนาน 8 หน่วยประมวลผล

บทที่ 3

การหาเส้นทางที่สั้นที่สุดเพื่อการฝังฟังก์ชัน ATAPE แบบขนานบนเครือข่าย HHC

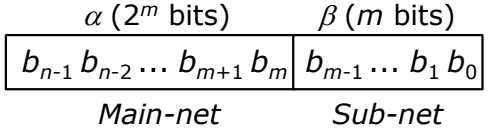
เนื่องจากการหาเส้นทางการสื่อสารเพื่อรับส่งข้อมูลระหว่างหน่วยประมวลผลของงานวิจัยก่อนหน้า [22] ได้นำเสนอกระบวนการสื่อสารเพื่อเพิ่มความน่าเชื่อถือ และสามารถแก้ปัญหาขัดแย้งได้ แต่วิธีการที่นำเสนอยังไม่สมบูรณ์ เพราะมีข้อจำกัดที่ต้องเริ่มที่โหนดต้นทาง $S = 0$ เท่านั้น ซึ่งทำให้ในกรณีที่โหนดต้นทางเปลี่ยนไปเป็นโหนดอื่น เช่น $S = 1, 2, 3, ..., N-1$ จะไม่สามารถหาเส้นทางที่ถูกต้องได้ ดังนั้นในหัวข้อนี้จึงเสนอ การหาเส้นทางสื่อสารที่สั้นที่สุดระหว่างต้นทาง S ไปยังปลายทาง D แบบยืดหยุ่นบนเครือข่าย HHC ที่จะเสนอในหัวข้อ 3.1 และการหาเส้นทางที่สั้นที่สุดแบบขนานบนเครือข่าย HHC เมื่อ $k = 2^{m+1}$ ในหัวข้อ 3.2 และการเพิ่มขนาดของหน่วยประมวลผล $2^{m+1} < k \leq 2^n/2^{M-1}$ ในหัวข้อที่ 3.3 และสุดท้ายในหัวข้อ 3.4 นำเสนอการจัดสรรงานที่ดีที่สุดด้วยโครงสร้างต้นไม้แบบทวิภาค (Binary Tree) บนเครือข่าย HHC

3.1 การหาเส้นทางสื่อสารที่สั้นที่สุดแบบหนึ่งต่อหนึ่งบนเครือข่าย HHC

ขั้นตอนวิธีการแก้ปัญหาการหาเส้นทางการสื่อสารเพื่อรับส่งข้อมูลระหว่างหน่วยประมวลผลแบบหนึ่งต่อหนึ่งระหว่างต้นทาง (Source) S ใดๆสู่ปลายทาง (Destination) D ใดๆ จะประกอบด้วย 2 ขั้นตอนหลัก ประกอบด้วยขั้นตอนแรก คือ การหาเส้นทางภายนอก (External-Path) หรือ μ เพื่อเป็นการกำหนดเส้นทางเบื้องต้นสำหรับการสื่อสาร โดยนำขั้นตอนวิธีที่ 2.1 และ 2.2 มาประยุกต์เข้าด้วยกันและถูกนำเสนอในหัวข้อที่ 3.1.1 ขั้นตอนที่สองถูกนำเสนอในหัวข้อที่ 3.1.2 คือ การจัดเส้นทางภายใน (Internal-Path) ที่สั้นที่สุดเพื่อเป็นการเติมเต็มเส้นทางภายนอก โดยปรับปรุงขั้นตอนจากขั้นตอนวิธี 2.2 และเสนอวิธีการจัดลำดับบิตใหม่ (Reordering) เพื่อให้ได้เส้นทางที่สั้นที่สุด

3.1.1 การหาเส้นทางภายนอกที่สั้นที่สุดบนเครือข่ายหลัก

กำหนดให้คู่ใดๆของโหนดต้นทาง $S = (\alpha_S, \beta_S)$ และโหนดปลายทาง $D = (\alpha_D, \beta_D)$ คือ โหนดบนเครือข่ายการเชื่อมต่อ HHC ($N = 2^n$, $n = 2^m+m$) และแต่ละโหนดถูกแทนด้วย n บิต ที่ประกอบไปด้วย 2 ส่วนโดยส่วนแรกมีจำนวน 2^m บิต ประกอบด้วย $\alpha = b_{n-1}, ..., b_{m+1}, b_m$ และถูกเรียกว่าเครือข่ายหลัก ส่วนที่สองมีจำนวน m บิต ประกอบด้วย $\beta = b_{m-1}, ..., b_1, b_0$ และถูกเรียกว่าเครือข่ายรอง ดังประกอบในรูปที่ 3.1



รูปที่ 3.1 การระบุอินเด็กซ์ระดับบิต ของหน่วยประมวลผลบนเครือข่าย HHC

ขั้นตอนวิธีที่ 3.1 แสดงการหาเส้นทางภายนอกที่สั้นที่สุดบนเครือข่ายหลัก ระหว่างโหนดต้นทาง $S = (\alpha_S, \beta_S)$ และปลายทาง $D = (\alpha_D, \beta_D)$ ใดๆ โดยการใช้การเข้าคู่แบบเกรย์โค๊ด (Gray-code Mapping) เพื่อให้ได้เส้นทางภายนอกที่สั้นที่สุดและมีเส้นทางที่ถูกต้อง ขั้นตอนวิธีที่ 3.1 เริ่มดำเนินการโดย การหาบิตที่ต่างกันบนเครือข่ายหลักระหว่างโหนด S และ D โดยกำหนดให้ $\alpha^* = \alpha_S \oplus \alpha_D = \{b_{2^{m+m-1}}, b_{2^{m+m-2}}, ..., b_m\}$ เมื่อ $\oplus$ หมายถึง การดำเนินการ XOR (Exclusive OR operation)

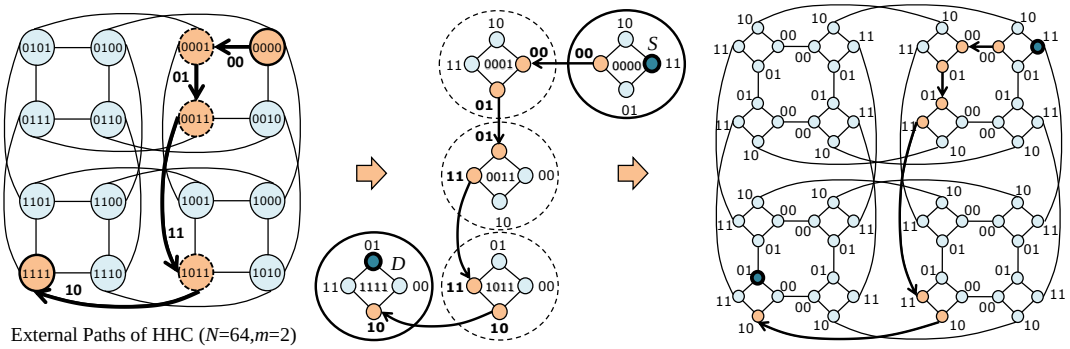
ขั้นตอนต่อมา คือ การหาบิต 1 โดยเรียงจากขวามือไปซ้ายมือในผลลัพธ์ของ α^* เพื่อนำไปเปรียบเทียบหาเส้นทางบนเครือข่ายหลัก โดยการใช้การเข้าคู่แบบเกรย์-โค๊ด ตัวอย่างเช่น กรณี $m = 2$ จะได้ว่า 2-บิต เกรย์-โค๊ด = {00, 01, 11, 10} และถ้าผลลัพธ์ของ $\alpha^* = 1101$ จะได้ว่าตำแหน่งของบิต 1 โดยเรียงจากขวามือไปซ้ายมือ คือตำแหน่งที่ 0, 2 และ 3 ตามลำดับ

ขั้นตอนวิธีที่ 3.1 ขั้นตอนการหาเส้นทางบนเครือข่ายหลัก HHC โดยการใช้ตัวดำเนินการ XOR (Exclusive-OR Operation) และการเข้าคู่แบบเกรย์โค๊ด (Gray-code Mapping)

GetExPath

1. $index = \emptyset$; $\mu = \emptyset$
2. $\alpha^* = \alpha_S \oplus \alpha_D = \{b_{2^{m+m-1}}, b_{2^{m+m-2}}, ..., b_m\}$;
3. **for** α^* with $b_i = 1$ **do**
4. $index = index \cup i$;
5. **end for**
6. **for** $index_i$ **do** // Gray-code mapping for a shortest ex-path
7. $\mu = \mu \cup$ GC[index];
8. **end for**
9. **return** μ;

รูปที่ 3.2 แสดงการหาเส้นทางบนเครือข่ายหลัก (μ) ของเครือข่าย 6-HHC ($N = 64$, $n = 6$ และ m = 2) โดยกำหนดให้ ต้นทาง $S = (\alpha_S, \beta_S) = (0000,11) = 3$ และโหนดปลายทาง $D = (\alpha_D, \beta_D) = (1111,01) = 61$ ขั้นตอนแรก คือ การหา $\alpha^* = \alpha_S \oplus \alpha_D$ จะได้ว่า $\alpha^* = \alpha_S \oplus \alpha_D = 0000 \oplus 1111 = 1111$ ขั้นตอนต่อไป คือ การหาบิต 1 ในผลลัพธ์ของ α^* จะได้ว่าตำแหน่งของ บิต 1 โดยเรียงจากขวามือไปซ้ายมือ คือ ตำแหน่ง (Index) ที่ 0, 1, 2 และ 3 ตามลำดับ ขั้นตอนต่อไป คือ การเปรียบเทียบตำแหน่งที่ได้ กับการเข้าคู่เกรย์โค๊ด ที่มี $m = 2$ (2-bit Gray-code) จะได้ผลลัพธ์ $\mu = \{ 00, 01, 11, 10\}$ จากการเปรียบเทียบกับ 2-บิต เกรย์-โค๊ด = {00, 01, 11, 10} และจากผลลัพธ์ที่ได้จากโหนดต้นทาง $S = 3$ สู่ปลายทาง $D = 61$ จะมีเส้นทางการสื่อสารบนเครือข่ายหลัก ดังแสดงในรูป 3.2

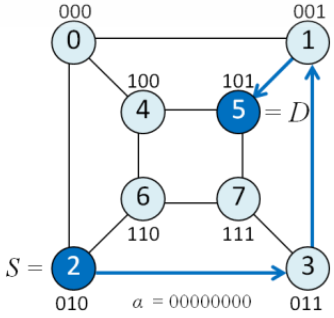


รูปที่ 3.2 ตัวอย่างเส้นทางการสื่อสารบนเครือข่ายหลัก (μ) จากโหนดต้นทาง S = (0000,11) = 3 สู่ปลายทาง D = (1111,01) = 61 เมื่อ μ = { 00, 01, 11, 10}

3.1.2 การจัดเส้นทางการสื่อสารภายในที่สั้นที่สุดบนเครือข่าย HHC

การหาเส้นทางการสื่อสารเพื่อรับส่งข้อมูลที่สั้นที่สุด แบบหนึ่งต่อหนึ่ง (Node to Node) จะเป็นการออกแบบขั้นตอนวิธีที่ต่อเนื่องจากหัวข้อ 3.1.1 โดยใช้ค่า μ มาทำการพิจารณาเพื่อหาเส้นทางภายในบนเครือข่ายรอง (Sub-Network) โดย การหาเส้นทางการสื่อสารที่สั้นที่สุดระหว่างต้นทาง S ไปยังปลายทาง D จะถูกแบ่งเป็น 2 กรณี คือ กรณีแรกคือ กรณีที่ต้นทาง S และปลายทาง D อยู่ในโหนดเดียวกันของเครือข่ายหลัก หรือ $\alpha_S = \alpha_D$ กรณีที่สองคือ ต้นทาง S และปลายทาง D ไม่อยู่ในโหนดเดียวกันของเครือข่ายหลัก หรือ $\alpha_S \neq \alpha_D$

สำหรับกรณีแรก $\alpha_S = \alpha_D$ การหาเส้นทางจะใช้วิธี ไฮเปอร์คิวบ์-บิตไวส์ (Hypercube-bitwise Routing) ซึ่งก็คือ การกำหนดเส้นทางจากบิตที่แตกต่างกัน จากบิตขวาสุดมาซ้ายสุด ระหว่าง S และ D ทีละบิต เช่น บนเครือข่าย 11-HHC เมื่อ m = 3 ดังนั้นจะมีบิตของ α จำนวน $2^m = 2^3$ = 8 บิต และบิตของ β จำนวน m = 3 บิต กำหนดให้ ต้นทาง $S = (\alpha_S, \beta_S)$ = (00000000, 010) = 2 และปลายทาง $D = (\alpha_D, \beta_D)$ = (00000000, 101) = 5 สังเกตว่าทั้งโหนดต้นทาง S และปลายทาง D มีค่า $\alpha_S = \alpha_D$ = 00000000 นั่นคือ ทั้งต้นทาง S และปลายทาง D อยู่บนโหนดเดียวกันบนเครือข่ายหลัก ดังนั้นเส้นทางการสื่อสารจะเป็นเส้นทางที่อยู่ใน 3-ไฮเปอร์คิวบ์ และเมื่อใช้วิธี ไฮเปอร์คิวบ์-บิตไวส์ จะได้เส้นทางที่เริ่มต้นจาก S = (00000000, 010) → (00000000, 011) → (00000000, 001) → (00000000, 101) = D ที่มีระยะทางเท่ากับจำนวนบิตที่แตกต่างกันของ S และ D ซึ่งก็คือ 3 ดังแสดงในรูปที่ 3.3



รูปที่ 3.3 ตัวอย่างการหาเส้นทางด้วยวิธีไฮเปอร์คิวบ์-บิตไวส์ จาก S ไปยัง D บน 3-HC ของ 11-HHC

กรณีที่สอง คือ กรณีที่ต้นทาง S และปลายทาง D ไม่ได้อยู่ในโหนดเดียวกันของเครือข่ายหลัก หรือ $\alpha_S \neq \alpha_D$ กรณีนี้เป็นกรณีที่มีความซับซ้อนกว่ากรณีแรก เพราะจะมีการนำค่า β_S และ β_D มาพิจารณาเพื่อให้ได้เส้นทางการสื่อสารที่ถูกต้องและเป็นเส้นทางที่สั้นที่สุด โดยจะนำค่า $\mu = \{c_0, c_1, ..., c_{r-1}\}$ ที่ได้จากขั้นตอนวิธีที่ 3.1 มาพิจารณาหาเส้นทางที่สั้นที่สุดตาม β_S และ β_D ที่อยู่ใน μ โดยวิธีการหาเส้นทางจะนำเสนอในขั้นตอนวิธีที่ 3.2

ขั้นตอนวิธีที่ 3.2 ขั้นตอนการจัดลำดับบิตใหม่ (Reordering) ของเส้นทางบนเครือข่ายหลัก (μ)

GetSP-PathHHC

1. $P = \emptyset$; μ = GetExPath (Algorithm 3.1)
2. **if** ($\alpha_S = \alpha_D$) **then**
3. set P_{in} (HC internal-path routing);
4. **else** ($\alpha_S \neq \alpha_D$)
5. **Case** 1: $\beta_S \in \mu$ and $\beta_S = \beta_D$
6. $P = P \cup \{ \beta_S$, SPordering (μ - { β_S })};
7. **Case** 2: β_S , $\beta_D \in \mu$ and $\beta_S \neq \beta_D$
8. $P = P \cup \{ \beta_S$, SPordering (μ - { β_S, β_D}), β_D};
9. **Case** 3: $\beta_S \in \mu$ and $\beta_D \notin \mu$
10. $P = P \cup \{ \beta_S$, SPordering (μ - { β_S })};
11. **Case** 4: $\beta_S \notin \mu$ and $\beta_D \in \mu$
12. $P = P \cup \{$ SPordering (μ - { β_S }), β_D };
13. **Case** 5: $\beta_S \notin \mu$ and $\beta_D \notin \mu$
14. $P = P \cup \{$ SPordering (μ)};
15. **return** P;

SPordering in O($|\mu|^2$), $|\mu| = r$ (Hamming distance between main-nets of S & D)

1. $temp = \beta_S$; $r' = r$;
2. **while** ($|p|$) < r **do**
3. **for** (i = 0; i< r'; i++) **do**
4. find $\mu[\, i\,]$ that is closest to $temp$;
5. $temp = \mu[\, i\,]$;
6. $p = p \cup \mu[\, i\,]$;
7. update $\mu[\, i\,]$ - 1;
8. $r' = r'$ - 1;
9. **break**;
10. **end for**
11. **end while**

ขั้นตอนวิธีที่ 3.2 แสดงขั้นตอนการหาเส้นทางเพื่อรับส่งข้อมูลที่สั้นที่สุดที่ปรับปรุงขั้นตอนวิธีมาจากขั้นตอนวิธีที่ 2.2 โดยกรณีที่ $\alpha_S \neq \alpha_D$ จะแบ่งเป็นกรณีย่อย 5 กรณี พร้อมด้วยขั้นตอนวิธีย่อย SPordering เพื่อที่จะได้เส้นทางที่สั้นที่สุด ภายใน O($|\mu|^2$)

ตัวอย่างเช่น บนเครือข่าย 6-HHC เมื่อ $N = 2^n = 64$, $n = 2^m+m = 6$ และ $m = 2$ กำหนดต้นทาง $S = (\alpha_S, \beta_S) = (0101, 11) = 23$ และปลายทาง $D = (\alpha_D, \beta_D) = (1010, 11) = 43$ การหาเส้นทางที่สั้นที่สุดเพื่อรับส่งข้อมูล ด้วยขั้นตอนวิธีที่ 3.2 จะเริ่มต้นที่ การหาเส้นทางบนเครือข่ายหลัก หรือ μ โดยใช้ขั้นตอนวิธีที่ 3.1 ซึ่งจะได้ $\alpha^* = \alpha_S \oplus \alpha_D = 0101 \oplus 1010 = 1111$ และเมื่อเปรียบเทียบตำแหน่งที่ได้กับการเข้าคู่เกรย์โค๊ด ที่มี $m = 2$ (2-bit Gray-code) จะได้ผลลัพธ์ $\mu = \{ 00, 01, 11, 10\}$ ขั้นตอนต่อมา สังเกตว่า $\alpha_S = 0101 \neq 1010 = \alpha_D$ ดังนั้นแสดงว่า ทั้งต้นทาง S และปลายทาง D ไม่ได้อยู่ในโหนดเดียวกันของเครือข่ายหลัก จึงต้องพิจารณาว่าเมื่อ $\beta_S = 11$ และ $\beta_D = 11$ ตกอยู่ในกรณีใด จากทั้งหมด 5 กรณี เมื่อพิจารณาจาก $\mu = \{ 00, 01, 11, 10\}$ จะได้ว่าในกรณีนี้ จะเข้าเงื่อนไขกรณีที่ 1 (Case 1) ซึ่งก็คือ $\beta_S \in \mu$ และ $\beta_S = \beta_D$ และเส้นทางการสื่อสาร (P) จะได้จากสมการ

$$\begin{aligned} P &= \{ \beta_S , \text{SPordering} (\mu - \{ \beta_S \})\} \\ &= \{ 11, \text{SPordering} (\mu - \{11\})\} \\ &= \{ 11, \text{SPordering} (00, 01, 10)\} \end{aligned}$$

ขั้นตอนต่อไปเป็นการดำเนินการเรียงลำดับบิตใหม่ด้วยฟังก์ชัน SPordering โดยวิธีการของ SPordering เริ่มต้นที่การเปรียบเทียบค่า $temp = \beta_S$ กับอินเด็กซ์ที่ 0 ใน μ ที่มีบิตใกล้กันมากที่สุด (แตกต่างกันน้อยที่สุด) และทำการทำซ้ำจนกว่าทุกค่าใน μ เป็นค่าที่ถูกเรียงลำดับความต่างบิตที่เหมาะสมที่สุด ดังนั้นจาก SPordering (00, 01, 10) จะได้ว่า $temp = 11$ และ ค่าแรกของ μ (อินเด็กซ์ 0) คือ 00 จะเห็นได้ว่า 11 และ 00 มีความต่างบิต 2 บิต ดังนั้นไม่ใช่บิตที่ดีที่สุด ต่อมาเทียบ 11 และ 01 จะเห็นได้ว่า มีความต่าง 1 บิต ดังนั้น จะได้ค่าแรกของ SPordering คือ 01 และต่อมา เทียบ 01 กับ 00 จะได้ค่าต่อมาคือ 00 และสุดท้ายเทียบ 00 กับ 10 จะได้ค่าสุดท้ายคือ 10 ดังนั้น ผลลัพธ์ของฟังก์ชัน SPordering(00, 01, 10) = 01, 00, 10 และจะได้เส้นทางที่สั้นที่สุด (P) ดังนี้

$$\begin{aligned} P &= \{ \beta_S , \text{SPordering} (\mu - \{ \beta_S \})\} \\ &= \{ 11, \text{SPordering} (00, 01, 10)\} \\ &= \{ 11, 01, 00, 10\} \end{aligned}$$

และท้ายที่สุดผลลัพธ์ของเส้นทางจากต้นทาง $S = 23$ ไปสู่ปลายทาง $D = 43$ โดยใช้ขั้นตอนวิธีที่ 3.2 คือ $P = \{ 11, 01, 00, 10\}$ และจะได้เส้นทางคือ $S = 23 = (0101, 11) \longrightarrow (1101, 11) \xrightarrow{*} (1101, 01) \xrightarrow{*} (1111, 01) \longrightarrow (1111, 00) \xrightarrow{*} (1110, 00) \longrightarrow (1110, 10) \xrightarrow{*} (1010, 10) \longrightarrow (1010, 11) = 43 = D$ ที่มีระยะทาง 8 ประกอบด้วย 4 เส้นทางภายนอก ถูกแทนด้วย $\longrightarrow$ และ มี 4 เส้นทางภายใน ถูกแทนด้วย $\xrightarrow{*}$ ดังแสดงให้เห็นในรูปที่ 3.4

รูปที่ 3.5 แสดงตัวอย่างการหาเส้นทางที่สั้นที่สุด โดยใช้ขั้นตอนวิธีที่ 3.2 กำหนดจุดเริ่มต้นคือ $S = (\alpha_S, \beta_S) = (0000, 11) = 3$ และปลายทาง $D = (\alpha_D, \beta_D) = (1111, 01) = 61$ การหาเส้นทางที่สั้นที่สุดจะเริ่มต้นที่การหาเส้นทางบนเครือข่ายหลัก หรือ μ โดยใช้ขั้นตอนวิธีที่ 3.1 ซึ่งจะได้

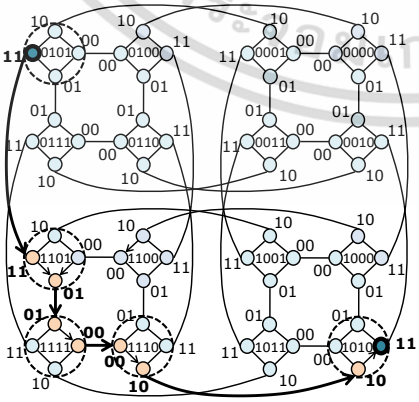
$\alpha^* = \alpha_S \oplus \alpha_D$ = 0000 ⊕ 1111 = 1111 และเมื่อเปรียบเทียบตำแหน่งที่ได้กับการเข้าคู่เกรย์โค๊ด ที่มี m = 2 (2-bit Gray-code) จะได้ผลลัพธ์ μ = { 00, 01, 11, 10} ขั้นตอนต่อมา สังเกตว่า α_S = 0000 ≠ 1111 = α_D ดังนั้นแสดงว่าทั้งต้นทาง S และปลายทาง D ไม่ได้อยู่ในโหนดเดียวกันบนเครือข่ายหลัก จึงต้องพิจารณาว่าเมื่อ β_S = 11 และ β_D = 01 ตกอยู่ในกรณีใดจากทั้งหมด 5 กรณี เมื่อพิจารณาจาก μ = { 00, 01, 11, 10} จะได้ว่าในกรณีนี้ จะเข้าเงื่อนไขกรณีที่ 2 (Case 2) ซึ่งก็คือ $\beta_S \in \mu$ และ $\beta_D \in \mu$ และเส้นทางการสื่อสาร (P) จะได้จากสมการ

$$\begin{aligned} P &= \{ \beta_S , \text{SPordering } (\mu - \{ \beta_S, \beta_D\}), \beta_D\} \\ &= \{ 11 , \text{SPordering } (\mu - \{ 11, 01\}), 01\} \\ &= \{ 11 , \text{SPordering } (00, 10), 01\} \end{aligned}$$

ขั้นตอนต่อไปเป็นการดำเนินการเรียงลำดับบิตใหม่ด้วยฟังก์ชัน SPordering จะได้ว่า *temp* = 11 และ ค่าแรกของ μ (อินเด็กซ์ 0) คือ 00 จะเห็นได้ว่า 11 และ 00 มีความต่างบิต 2 บิต ดังนั้นไม่ใช่บิตที่ดีที่สุด ต่อมาเทียบ 11 และ 10 จะเห็นได้ว่า มีความต่าง 1 บิต ดังนั้นจะได้ค่าแรกของ SPordering คือ 10 และต่อมา เทียบ 10 กับ 00 จะได้ค่าสุดท้ายคือ 00 ดังนั้น ผลลัพธ์ของฟังก์ชัน SPordering (00, 10) =10, 00 และจะได้เส้นทางที่สั้นที่สุด (P) ดังนี้

$$\begin{aligned} P &= \{ \beta_S , \text{SPordering } (\mu - \{ \beta_S, \beta_D\}), \beta_D\} \\ &= \{ 11 , \text{SPordering } (00, 10), 01\} \\ &= \{ 11, 10, 00, 01\} \end{aligned}$$

และท้ายที่สุดผลลัพธ์ของเส้นทางจากต้นทาง S = 3 ไปสู่ปลายทาง D = 61 โดยใช้ขั้นตอนวิธีที่ 3.2 คือ P = { 11, 10, 00, 01} และจะได้เส้นทางคือ S = 23 = (0000, 11) $\longrightarrow$ (1000, 11) $\xrightarrow{*}$ (1000, 10) $\longrightarrow$ (1100, 10) $\xrightarrow{*}$ (1100, 00) $\longrightarrow$ (1101, 00) $\xrightarrow{*}$ (1101, 01) $\longrightarrow$ (1111, 01) = 43 = D ที่มีระยะทาง 7 ประกอบด้วย 4 เส้นทางภายนอก และมี 3 เส้นทางภายใน ดังแสดงให้เห็นในรูปที่ 3.5



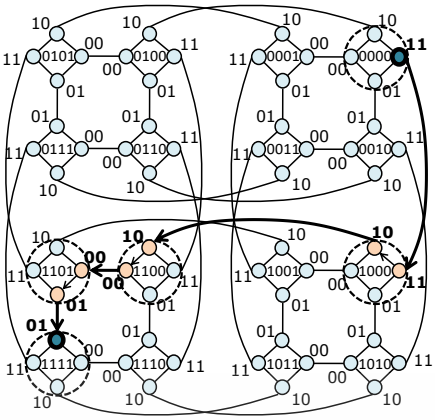
HHC Routing (N=64, m=2)
From S = (α_S , β_S) = 0101,11 = 23
To D = (α_D , β_D) = 1010,11 = 43

α^*= $\alpha_S \oplus \alpha_D$ = (1111)
μ = {c_0, c_1, c_2, c_3} = {00,01,11,10}
Case 1: P = {β_S, SPorderingt(μ-{β_S})}
= {11, SPordering(μ-{11})}
SPordering(μ-{11})=SPordering(00,01,~~11~~,10)
={01,00,10}
then P = {11,01,00,10}

HHC routing-path from S to D

Main-Net	Sub-Net
0101	S: 0101, 11
1101	1101, **11** 1101, 01
11**1**1	1111, **01** 1111, 00
111**0**	1110, **00** 1110, 10
10**1**0	1010, **10** D: 1010, 11
4 steps	4+4 steps

รูปที่ 3.4 ตัวอย่างเส้นทางการสื่อสาร ต้นทาง S = 23 และปลายทาง D = 43 บน 6-HHC



HHC Routing (N=64, m=2)
From $S = (\alpha_S, \beta_S) = 0000,11 = 3$
To $D = (\alpha_D, \beta_D) = 1111,01 = 61$

$\alpha^* = \alpha_S \oplus \alpha_D = (1111)$
$\mu = \{c_0, c_1, c_2, c_3\} = \{00,01,11,10\}$
Case 2: $P = \{\beta_S, \text{SPordering}(\mu\text{-}\{\beta_S,\beta_D\}), \beta_D\}$
$= \{11, \text{SPordering}(\mu\text{-}\{11,01\}), 01\}$
SPordering(μ-{11,01})=SPordering{00,~~01~~,~~11~~,10}
$= \{10,00\}$
then $P = \{11,10,00,01\}$

HHC routing-path from S to D

Main-Net	Sub-Net
0000	S: 0000, 11
1000	1000, **11** 1000, 10
1**1**00	1100, **10** 1100, 00
110**1**	1101, **00** 1101, 01
111**1**	D: 1111, **01**
4 steps	4+3 steps

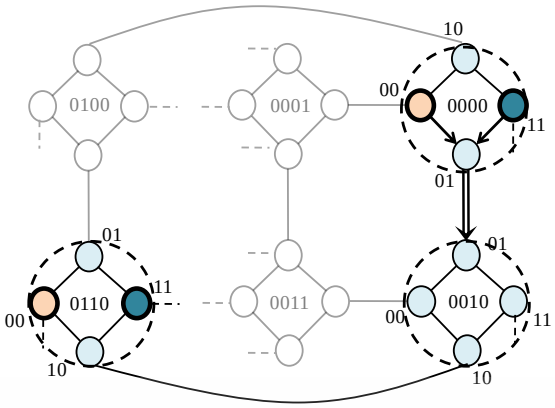
รูปที่ 3.5 ตัวอย่างเส้นทางการสื่อสาร ต้นทาง S = 3 และปลายทาง D = 61 บน 6-HHC

3.2 การหาเส้นทางที่สั้นที่สุดแบบขนาน ของหน่วยประมวลผล $k = 2^{m+1}$ บนเครือข่าย HHC

แม้ว่าขั้นตอนวิธีที่ 3.2 ในการนำเสนอในหัวข้อที่ผ่านมา เป็นขั้นตอนที่ดีและเหมาะสมสำหรับการหาเส้นทางที่สั้นที่สุดแบบหนึ่งต่อหนึ่งระหว่างต้นทาง S และปลายทาง D แต่เมื่อนำมาประยุกต์ใช้เพื่อการฝังฟังก์ชัน ATAPE ลงบนเครือข่าย HHC ในลักษณะของการสื่อสารแบบขนาน (Parallel Communication) ขั้นตอนวิธีที่ 3.2 ยังไม่เหมาะสม เนื่องจากยังมีการขัดแย้งกันของเส้นทางการสื่อสารขณะรับส่งข้อมูล ด้วยเหตุนี้ในหัวข้อนี้จึงนำเสนอให้เห็นถึงการขัดแย้งกันของเส้นทางการสื่อสารแบบขนาน จากต้นทาง S สู่ปลายทาง D และนำเสนอวิธีการแก้ไขโดยการปรับปรุงขั้นตอนวิธีที่ 3.2 เพื่อหลีกเลี่ยงการขัดแย้งกันเมื่อสื่อสารแบบขนาน ซึ่งจะถูกนำเสนอในหัวข้อ 3.2.1 และต่อมาคือการฝังฟังก์ชัน ATAPE ขนาด $k = 2^{m+1}$ บนเครือข่าย HHC โดยการใช้เทคนิคการจัดกลุ่มของหน่วยประมวลผล ที่มีชื่อว่า “กลุ่มของดูอัล-คิวบ์แบบไขว้” (Grouping of Cross Dual-Cubes : GCD) และถูกนำเสนอในหัวข้อ 3.2.2 ตามลำดับ

3.2.1 การปรับปรุงขั้นตอนวิธีเพื่อรองรับการสื่อสารแบบขนาน

ก่อนหน้าที่จะนำเสนอวิธีการหาเส้นทางที่สั้นที่สุดแบบขนานบนเครือข่าย HHC ในหัวข้อนี้จะแสดงให้เห็นถึงปัญหาการขัดแย้งกันขณะสื่อสารแบบขนานเพื่อรับส่งข้อมูลระหว่างต้นทาง S_i และปลายทาง D_i เมื่อใช้ขั้นตอนวิธีที่ 3.2 โดยตัวอย่างที่แสดงให้เห็นถึงการขัดแย้งแสดงในรูปที่ 3.6 จากรูปเป็นการหาเส้นทางการสื่อสารแบบขนานระหว่างสองเส้นทาง บนเครือข่าย HHC เมื่อ m = 2 โดยเส้นทางแรก เริ่มจาก ต้นทาง S_1 = (0000, 00) = 0 สู่ปลายทาง D_1 = (0110, 00) = 24 และเส้นทางที่สอง เริ่มต้นจาก ต้นทาง S_2 = (0000, 11) = 3 สู่ปลายทาง D_2 = (0110, 11) = 27 ในกรณีที่ใช้ขั้นตอนวิธีที่ 3.2 จะแสดงขั้นตอนการหาเส้นทางดังนี้



S_1 = (0000 , 00) = 0; D_1 = (0110,00) = 24
S_2 = (0000 , 11) = 3; D_2 = (0110,11) = 27
$\alpha^* = \alpha_S \oplus \alpha_D$ = 0110; Thus μ = {01, 10}
Case 5: P = {SPordering(μ)}
2-bit-GC = {00, 01, 11, 10}
SPordering compares β_{S1} = 00 with μ
then (path $S_1 \rightarrow D_1$) P_1 = {01, 10}
SPordering compares β_{S2} = 11 with μ
then (path $S_2 \rightarrow D_2$) P_2 = {01, 10}

รูปที่ 3.6 ตัวอย่างของการสื่อสารแบบขนาน ระหว่าง S_1 = 0000,00 = 0 สู่ D_1 = 0110,00 = 24 และ S_2 = 0000,11 = 3 สู่ D_2 = 0110,11 = 27 โดยใช้ขั้นตอนวิธีที่ 3.2

เส้นทางที่ 1 (P_1)

ต้นทาง $S_1 = (\alpha_{1S}, \beta_{1S}) = (0000, 00) = 0$

ปลายทาง $D_1 = (\alpha_{1D}, \beta_{1D}) = (0110, 00) = 24$

$\alpha^* = \alpha_S \oplus \alpha_D = 0000 \oplus 0110 = 0110$

เมื่อเทียบกับ 2-บิต เกรย์โค๊ด (00, 01, 11, 10)

จะได้ μ = { 01, 10}

จาก β_{1S} = 00 และ β_{1D} = 00

จะตกอยู่ใน กรณี 5 คือ $\beta_S \notin \mu$ และ $\beta_D \notin \mu$

หาเส้นทางได้จาก P = { SPordering (μ)}

กำหนดให้ $temp = \beta_{1S}$ = 00

ดังนั้น เมื่อนำ 00 เทียบกับ อินเด็กซ์แรกของ μ คือ 01

จะได้ อินเด็กซ์แรก ของ P_1 คือ 01

ต่อมา เมื่อนำ 01 เทียบกับ อินเด็กซ์ต่อมาของ μ คือ 10

จะได้ อินเด็กซ์สุดท้าย ของ P_1 คือ 10

ดังนั้น สรุปผลลัพธ์ได้ว่า P_1 = {01, 10}

เส้นทางที่ 2 (P_2)

ต้นทาง $S_2 = (\alpha_{2S}, \beta_{2S}) = (0000, 11) = 3$

ปลายทาง $D_2 = (\alpha_{2D}, \beta_{2D}) = (0110, 11) = 27$

$\alpha^* = \alpha_S \oplus \alpha_D = 0000 \oplus 0110 = 0110$

เมื่อเทียบกับ 2-บิต เกรย์โค๊ด (00, 01, 11, 10)

จะได้ μ = { 01, 10}

จาก β_{2S} = 11 และ β_{2D} = 11

จะตกอยู่ใน กรณี 5 คือ $\beta_S \notin \mu$ และ $\beta_D \notin \mu$

หาเส้นทางได้จาก P = { SPordering (μ)}

กำหนดให้ $temp = \beta_{2S} = 11$

ดังนั้น เมื่อนำ 11 เทียบกับ อินเด็กซ์แรกของ μ คือ 01

จะได้ อินเด็กซ์แรก ของ P_2 คือ 01

ต่อมา เมื่อนำ 01 เทียบกับ อินเด็กซ์ต่อมาของ μ คือ 10

จะได้ อินเด็กซ์สุดท้าย ของ P_2 คือ 10

ดังนั้น สรุปผลลัพธ์ได้ว่า $P_2 = \{01, 10\}$

และจากรูปที่ 3.6 จะเห็นได้ว่า เส้นทางการสื่อสารของ $P_1 = \{01, 10\}$ และ $P_2 = \{01, 10\}$ มีเส้นทางดังต่อไปนี้

$S_1 = 0 = (0000, 00) \xrightarrow{*} (0000, 01) \longrightarrow (0010, 01) \xrightarrow{*} (0010, 00) \xrightarrow{*} (0010, 10) \longrightarrow (0110, 10) \xrightarrow{*} (0110, 00) = 24 = D_1$

$S_2 = 3 = (0000, 11) \xrightarrow{*} (0000, 01) \longrightarrow (0010, 01) \xrightarrow{*} (0010, 00) \xrightarrow{*} (0010, 10) \longrightarrow (0110, 10) \xrightarrow{*} (0110, 11) = 27 = D_2$

จะเห็นได้ว่า เมื่อทั้งสองเส้นทาง เริ่มรับส่งข้อมูลพร้อมกันแบบขนาน จะมีการขัดแย้งกันบนเส้น $e = (u, v)$ เมื่อ $u = (0000, 01)$, $v = (0010, 01)$ ในสัญญาณนาฬิกา (Clock) ที่ 2

ด้วยเหตุนี้ ขั้นตอนวิธีที่ 3.2 จึงไม่เหมาะที่จะนำมาประยุกต์ใช้กับฟังก์ชัน ATAPE แบบขนาน ดังนั้นเพื่อแก้ปัญหาการขัดแย้งกันเมื่อดำเนินการแบบขนานดังกล่าว วิทยานิพนธ์ฉบับนี้จึงนำขั้นตอนวิธี 3.2 มาปรับปรุง โดยการลดจำนวนเงื่อนไขกรณีลง (Case-reduction) เพื่อลดความซับซ้อน และนำเสนอฟังก์ชันการจัดลำดับบิตใหม่แบบพลวัต (Dynamic Reordering) เรียกว่าฟังก์ชัน dSPordering ที่มีทั้งการจัดลำดับบิตไปข้างหน้า (Forward) และถอยกลับแบบวนรอบ (Backward) เพื่อที่จะหลีกเลี่ยงการชนโดยการใช้เส้นทางที่สมมาตรกัน ซึ่งถูกอธิบายในขั้นตอนวิธีที่ 3.3

ขั้นตอนวิธีที่ 3.3 ขั้นตอนการหาเส้นทางเพื่อรับส่งข้อมูลแบบขนาน โดยการลดเงื่อนไข (Case-Reduction) และการจัดลำดับบิตใหม่แบบพลวัต (Dynamic Reordering)

GetParallelSPathHHC

1. $P = \emptyset$; μ = GetExPath (Algorithm 3.1)
2. **if** ($\alpha_S = \alpha_D$) **then**
3. set P_{in} (HC internal-path routing);
4. **else** ($\alpha_S \neq \alpha_D$)
5. **if** ($\beta_S \in \mu$) then
6. $P = P \cup \{ \beta_S$, dSPordering (μ - { β_S }) };
7. **else**
8. $P = P \cup \{$ dSPordering (μ) };
9. **return** P;

dSPordering(μ) // F: Forward

1. $temp = \beta_S$; $r' = r$;
2. d = (next index of β_S in μ) % r;

```
3.  while (|p|) < r do
4.        for ( i = 0; i< r′; i++) do
5.            id = (i + d) % r′;
6.              find μ[ id ] that is closest to temp;
7.                    temp = μ[ id ];
8.                    p = p ∪ μ[ id ];
9.                    update |μ| – 1;
10.                   r′  = r′ – 1;
11.                   d = (id + 1) % r′;       break;
12.           end for
13.  end while
14.  return p
```

```
dSPordering(μ) // B: Backward
1.  temp = β_S; r′ = r;
2.  d = (back index of β_S in μ+r) % r;
3.  while (|p|) < r do
4.        for ( i = 0; i< r′; i++) do
5.            id = (d – i + r′) % r′;
6.            find μ[ id ] that is closest to temp;
7.                    temp = μ[ id ];
8.                    p = p ∪ μ[ id ];
9.                    update |μ| – 1;
10.                   r′  = r′ – 1;
11.                   d = (id – 1 + r′) % r′;      break;
12.       end for
13. end while
14. return p
```

ขั้นตอนวิธี 3.3 คือขั้นตอนวิธีในการหาเส้นทางที่เหมาะสมที่สุด สำหรับการฝังฟังก์ชัน ATAPE แบบขนาน ลงบนเครือข่าย HHC ซึ่งถูกประยุกต์มาจากขั้นตอนวิธีที่ 3.2 โดยที่จะลดเงื่อนไขกรณีลง และใช้ประโยชน์จากคุณสมบัติที่มีความโดดเด่นอย่างหนึ่งของเครือข่าย HHC คือ ความสมมาตร (Symmetrical) เพื่อการหลีกเลี่ยงการขัดแย้งกันและทำให้เกิดความสอดคล้องในระหว่างการสื่อสารแบบขนาน โดยใช้ฟังก์ชัน dSPordering (Forward) และ dSPordering (Backward) ซึ่งทั้งสองฟังก์ชันเป็นวิธีการจัดลำดับบิตใหม่แบบพลวัตด้วยทิศทางตรงกันข้าม คือ เป็นการจัดลำดับบิตใหม่ไปข้างหน้าแบบวนรอบ (Forward) และ เป็นการจัดลำดับบิตใหม่ถอยกลับแบบวนรอบ (Backward)

ดังนั้นเมื่อใช้ขั้นตอนวิธีที่ 3.3 และ dSPordering (Forward) ซึ่งเป็นฟังก์ชันเริ่มต้น (Default function) ในการหาเส้นทางการสื่อสารแบบขนานระหว่างสองเส้นทางเดิมที่ได้อธิบายด้วยขั้นตอนที่ 3.2 และรูปที่ 3.6 บนเครือข่าย HHC เมื่อ $m = 2$ โดยเส้นทางแรก เริ่มจาก ต้นทาง S_1 = (0000, 00) = 0 สู่ปลายทาง D_1 = (0110, 00) = 24 และเส้นทางที่สอง เริ่มต้นจาก ต้นทาง S_2 = (0000, 11) = 3 สู่ปลายทาง D_2 = (0110, 11) = 27 จะแสดงขั้นตอนการหาเส้นทางดังนี้

<u>เส้นทางที่ 1 (P_1)</u> ต้นทาง $S_1 = (\alpha_{1S}, \beta_{1S}) = (0000, 00) = 0$

ปลายทาง $D_1 = (\alpha_{1D}, \beta_{1D}) = (0110, 00) = 24$

$\alpha^* = \alpha_S \oplus \alpha_D = 0000 \oplus 0110 = 0110$

เมื่อเทียบกับ 2- บิต เกรย์โค๊ด (00, 01, 11, 10)

จะได้ $\mu = \{ 01, 10\}$

จาก $\beta_{1S} = 00$

จะตกอยู่ใน กรณี $\beta_S \notin \mu$

หาเส้นทางได้จาก P = {dSPordering (μ)}

กำหนดให้ $temp = \beta_{1S} = 00$

ดังนั้น เมื่อนำ 00 เทียบกับ อินเด็กซ์แรกแบบไปข้างหน้า ของ μ คือ 01

จะได้ อินเด็กซ์แรก ของ P_1 คือ 01

ต่อมา เมื่อนำ 01 เทียบกับ อินเด็กซ์ต่อมาของ μ คือ 10

จะได้ อินเด็กซ์สุดท้าย ของ P_1 คือ 10

ดังนั้น สรุปผลลัพธ์ได้ว่า P_1 = {01, 10}

<u>เส้นทางที่ 2 (P_2)</u> ต้นทาง $S_2 = (\alpha_{2S}, \beta_{2S}) = (0000, 11) = 3$

ปลายทาง $D_2 = (\alpha_{2D}, \beta_{2D}) = (0110, 11) = 27$

$\alpha^* = \alpha_S \oplus \alpha_D = 0000 \oplus 0110 = 0110$

เมื่อเทียบกับ 2 - บิต เกรย์โค๊ด (00, 01, 11, 10)

จะได้ $\mu = \{ 01, 10\}$

จาก β_{2S}= 11

จะตกอยู่ใน กรณี $\beta_S \notin \mu$ and $\beta_D \notin \mu$

หาเส้นทางได้จาก P = { SPordering (μ)}

กำหนดให้ $temp = \beta_{2S} = 11$

ดังนั้น เมื่อนำ 11 เทียบกับ อินเด็กซ์แรกแบบไปข้างหน้า ของ μ คือ 10

จะได้ อินเด็กซ์แรก ของ P_2 คือ 10

ต่อมา เมื่อนำ 10 เทียบกับ อินเด็กซ์ต่อมาของ μ คือ 01

จะได้ อินเด็กซ์สุดท้าย ของ P_2 คือ 01

ดังนั้น สรุปผลลัพธ์ได้ว่า P_2 = {10, 01}

และจากรูปที่ 3.7 จะเห็นได้ว่า เส้นทางการสื่อสารของ P_1 = {01, 10} และ P_2 = {10, 01} มีเส้นทางดังต่อไปนี้

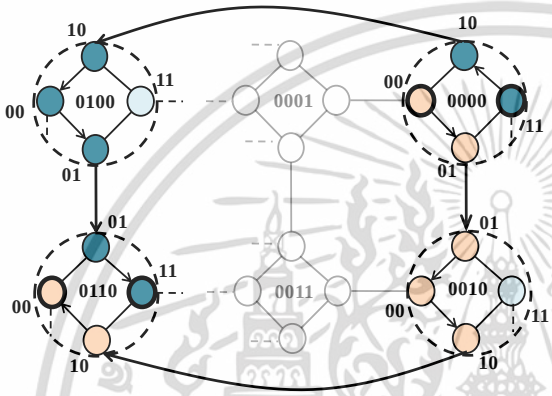
$P_1 : S_1 = 0 \quad =$ (0000, 00) $\xrightarrow{*}$ (0000, 01) $\longrightarrow$ (0010, 01) $\xrightarrow{*}$

(0010, 00) $\xrightarrow{*}$ (0010, 10) $\longrightarrow$ (0110, 10) $\xrightarrow{*}$

(0110, 00) = 24 = D_1

$$P_2 : S_2 = 3 = (0000, 11) \xrightarrow{*} (0000, 10) \longrightarrow (0010, 10) \xrightarrow{*}$$
$$(0010, 11) \xrightarrow{*} (0010, 01) \longrightarrow (0110, 01) \xrightarrow{*}$$
$$(0110, 11) = 27 = D_2$$

จากรูป สังเกตว่าการใช้ฟังก์ชัน dSPordering (Forward) ซึ่งเป็นการจัดลำดับบิตใหม่ ไปข้างหน้าแบบวนรอบ สามารถทำให้เส้นทางทั้งสองเส้นทางของ P_1 (เส้นทางสีส้ม) และ P_2 (เส้นทางสีน้ำเงิน) เป็นเส้นทางที่มีความสมมาตรกันและยังคงเป็นเส้นทางที่สั้นที่สุด โดยจะมีระยะทาง (Length) คือ 6 เท่ากัน และไม่มีการขัดแย้งกันระหว่างการรับส่งข้อมูล



S_1 = (0000 , 00) = 0; D_1 = (0110,00) = 24
S_2 = (0000 , 11) = 3; D_2 = (0110,11) = 27
$\alpha^* = \alpha_S \oplus \alpha_D$ = 0110; Thus μ = {01, 10}
Case $\beta_S \notin \mu$: P = {dSPordering(μ)}
2-bit-GC = {00, 01, 11, 10}
dSPordering compares β_{S1} = 00 with 01
then path ($S_1 \rightarrow D_1$) P_1 = {01, 10}
dSPordering compares β_{S2} = 11 with 10
then path ($S_2 \rightarrow D_2$) P_2 = {10, 01}

HHC routing-path from S_1 to D_1

Main-Net	Sub-Net
0000	S_1 : 0000, 00 0000, 01
00**1**0	0010, **01** 0010, 00 0010, 10
0**1**10	0110, **10** D_1 : 0110, 00
2 steps	2+4 steps

HHC routing-path from S_2 to D_2

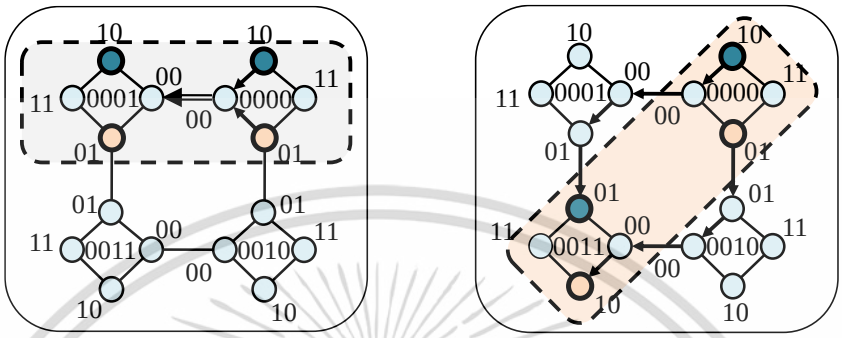
Main-Net	Sub-Net
0000	S_2 : 0000, 11 0000, 10
0**1**00	0100, **10** 0100, 11 0100, 01
0**1**10	0110, **01** D_2 : 0110, 11
2 steps	2+4 steps

รูปที่ 3.7 ตัวอย่างของการสื่อสารแบบขนาน ระหว่าง S_1 = 0000,00 = 0 สู่ D_1 = 0110,00 = 24 และ S_2 = 0000,11 = 3 สู่ D_2 = 0110,11 = 27 โดยใช้ขั้นตอนวิธีที่ 3.3

3.2.2 การฝังฟังก์ชัน ATAPE แบบขนานด้วยเส้นทางที่สั้นที่สุดบนเครือข่าย HHC

ในหัวข้อนี้จะเสนอการติดต่อสื่อสารเพื่อแลกเปลี่ยนข้อมูลด้วยฟังก์ชัน ATAPE แบบขนาน ที่สามารถแลกเปลี่ยนข้อมูลเฉพาะสำหรับแต่ละหน่วยประมวลผลที่ถูกจัดให้อยู่ในกลุ่มเดียวกันเพื่อที่จะติดต่อสื่อสารด้วยเส้นทางที่สั้นที่สุดบนเครือข่าย HHC ซึ่งแต่ละกลุ่มจะมีจำนวนหน่วยประมวลผลขนาด $k \le 2^{m+1}$ และจะใช้ขั้นตอนวิธีที่ 3.3 เป็นขั้นตอนวิธีในการหาเส้นทางการสื่อสารเพื่อรับส่งข้อมูล สังเกตว่า ในกรณีที่ $k \le 2^m$ การจัดกลุ่มของหน่วยประมวลผลที่เหมาะสมที่สุดบนเครือข่าย HHC คือ การจัดให้หน่วยประมวลผล k อยู่ในเครือข่ายรองที่มีขนาด 2^m หรือจัดให้อยู่ในไฮเปอร์คิวบ์ (Hypercube) ดังนั้นการรับส่งข้อมูลเพื่อติดต่อสื่อสารแบบ ATAPE จะไม่มีการส่งข้อมูลออกจาก

เครือข่ายรอง ในกรณีที่ $k = 2^{m+1}$ การติดต่อสื่อสารแบบ ATAPE จะมีความซับซ้อนมากขึ้น เนื่องจากจะมีการขัดแย้งกันในขณะสื่อสารระหว่างหน่วยประมวลผลที่ถูกจัดกลุ่มให้อยู่ในโหนดชั้นนอกที่อยู่ติดกัน (Adjacent External Node) ดังนั้นการจัดกลุ่มในลักษณะนี้จะไม่สามารถดำเนินการแบบ ATAPE ได้ ดังตัวอย่างที่แสดงในรูปที่ 3.8 ก)



รูปที่ 3.8 ก) การขัดแย้งกันของสองเส้นทางที่อยู่ในโหนดชั้นนอกที่ติดกัน และ ข) การจัดกลุ่มของดูอัล-ควิบ์แบบไขว้ที่ไม่มีการขัดแย้ง บนเครือข่าย 6-HHC

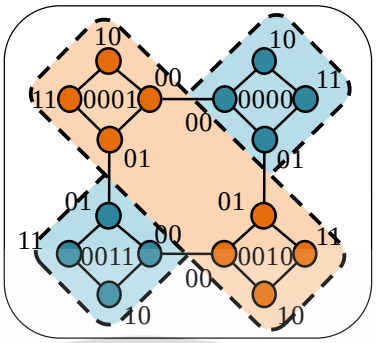
รูปที่ 3.8 ก) แสดงการขัดแย้งกันขณะสื่อสารของหน่วยประมวลผล ที่ถูกจัดกลุ่มอยู่ในโหนดชั้นนอกที่อยู่ติดกันบนเครือข่าย 6-HHC กำหนดให้เส้นทางแรก เริ่มต้นที่ ต้นทาง $S_1 = 1 = (0000, 01)$ ส่งข้อมูลไปยังปลายทาง $D_1 = 5 = (0001, 01)$ (โหนดสีส้ม) และเส้นทางที่สอง ต้นทาง $S_2 = 2 = (0000, 10)$ ส่งข้อมูลไปยังปลายทาง $D_2 = 6 = (0001, 10)$ (โหนดสีน้ำเงิน) จากรูปจะเห็นได้ว่า มีการขัดแย้งกันบนเส้น (edge) $e = (u, v)$ เมื่อ $u = 0000, 00$ และ $v = 0001, 00$

จากปัญหาของการขัดแย้งกันที่เกิดขึ้น หัวข้อนี้จึงเสนอวิธีการแก้ปัญหา โดยใช้วิธีการจัดกลุ่มหน่วยประมวลผลแบบไขว้ของหน่วยประมวลผลจำนวน $k = 2^{m+1}$ (ดังรูปที่ 3.8 ข)) เพื่อให้ได้กลุ่มของหน่วยประมวลผลที่เหมาะสมในการดำเนินการแลกเปลี่ยนข้อมูลแบบ ATAPE ด้วยเส้นทางที่สั้นที่สุด และไม่มีการขัดแย้งกันระหว่างสื่อสาร โดยหัวข้อ 3.2.2.1 นำเสนอการจัดกลุ่มของหน่วยประมวลผลดังกล่าว โดยเรียกว่า การจัดกลุ่มของดูอัล-ควิบ์แบบไขว้ (Grouping of Cross Dual-Cube : GCD) และหัวข้อ 3.2.2.2 นำเสนอการสร้างตารางลาติน (Latin Square) จากตัวดำเนินการเอ็กซ์ออล (XOR Operation : XOR) เพื่อกำหนดลำดับในการสื่อสารแบบ ATAPE ซึ่งเมื่อใช้ GCD และขั้นตอนวิธีที่ 3.3 พร้อมด้วยการดำเนินการแบบ XOR (XOR Operation) ทำให้การสื่อสารเพื่อแลกเปลี่ยนข้อมูลแบบ ATAPE ของหน่วยประมวลผลขนาด $k \leq 2^{m+1}$ บนเครือข่าย HHC จะได้ประโยชน์สูงสุด

3.2.2.1 การจัดกลุ่มของดูอัล-ควิบ์แบบไขว้ (Grouping of Cross Dual-Cubes)

เพื่อเป็นการจัดเตรียมกลุ่มของหน่วยประมวลผลที่เหมาะสมในการสื่อสารพร้อมกันแบบ ATAPE โดยไม่มีการขัดแย้งกันในระหว่างการสื่อสาร ในหัวข้อนี้เสนอวิธีการจัดกลุ่มของดูอัล-ควิบ์แบบไขว้ โดยในแต่ละกลุ่มจะมีดูอัล-ควิบ์แบบไขว้ (Cross of Dual-Cubes : Cross) จำนวน $2^{2^{m-1}-1}$ คู่

และในแต่ละคู่จะมีจำนวนหน่วยประมวลผล จำนวน $k \le 2^{m+1}$ ดังแสดงในรูปที่ 3.9 และแสดงการพิสูจน์ในทฤษฎีบท ที่ 3.1 - 3.3



รูปที่ 3.9 การจัดกลุ่มของดูอัล-คิวบ์แบบไขว้ เพื่อหลีกเลี่ยงการชนกัน เมื่อ $k = 2^{m+1}$ บน 6-HHC

<u>ทฤษฎีบท 3.1</u> ในการแบ่งกลุ่ม (Partitioning) บนเครือข่าย HHC ($N = 2^n$, $n = 2^m+m$) ขนาดของกลุ่มที่เล็กที่สุดที่จะสื่อสารพร้อมกันแบบ ATAPE โดยไม่มีการขัดแย้งด้วยหน่วยประมวลผลขนาด $k = 2^{m+1}$ คือ $N' = 2^{n'}$ เมื่อ $n' = 2^{m-1}+m$ และจำนวนของ Cross ในแต่ละกลุ่ม คือ $2^{2^{m-1}-1}$

พิสูจน์ จากโครงสร้างของเครือข่าย HHC จำนวนของหน่วยประมวลผลที่มีขนาดเล็กที่สุดในแต่ละกลุ่ม คือ 2^{M+m} เมื่อ $M = 2^{m-1}$ ดังนั้นบนเครือข่าย HHC-MP ที่ $m \ge 2$ จะมีค่า M อย่างน้อยคือ 2^{m-1} ตัวอย่างเช่น ถ้าจำนวนหน่วยประมวลผลในกลุ่ม คือ 2^{M+m} เมื่อ $M = 2^{m-2}$ และ $m = 2$ จะได้ว่า $M = 1$ และ $2^{M+m} = 2^{m+1}$ ซึ่งจะเกิดความขัดแย้ง ดังรูปที่ 3.8 ดังนั้นชัดเจนว่า ขนาดของหน่วยประมวลผลที่มีขนาดเล็กที่สุดในกลุ่ม คือ 2^{M+m} เมื่อ $M = 2^{m-1}$ ดังรูปที่ 3.9 และจำนวน Cross ในแต่ละกลุ่ม คือ $2^{n'}/2^{m-1} = 2^{2^{m-1}-1}$ □

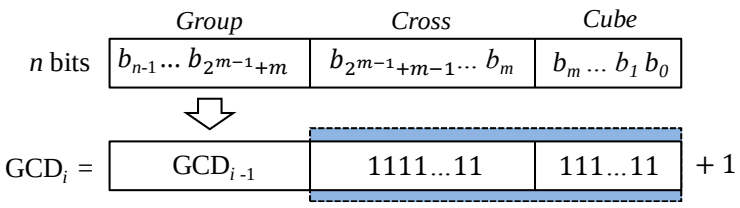
<u>ทฤษฎีบท 3.2</u> ภายใต้การแบ่งกลุ่มแบบ GCD จำนวนของกลุ่มที่สามารถสื่อสารกันได้อย่างอิสระบนเครือข่าย HHC ($N = 2^n$, $n = 2^m+m$) คือ $N/2^{n'}$ เมื่อ $n' = 2^{m-1}+m$ และแต่ละกลุ่มจะประกอบด้วยโหนดเริ่มต้น (Starting Node) ที่แตกต่างกันจำนวน $N' = 2^n$ โหนด และกำหนดให้ GCD_i คือ กลุ่มของโหนดเริ่มต้นกลุ่มที่ i เมื่อ $1 \le i < N/2^{n'}$ และสามารถสร้างได้จากสมการ 3.1 ภายในเวลา O(1)

$$GCD_0 = 0$$

และ

$$GCD_i = GCD_{i-1} + 2^{2^{m-1}+m} \qquad (3.1)$$

พิสูจน์ จากทฤษฎีที่ 3.1 ขนาดของกลุ่มบนเครือข่าย HHC คือ $N' = 2^{n'}$, จำนวนของกลุ่ม คือ $N/2^{n'}$ และแต่ละกลุ่ม มีจำนวน Cross คือ $2^{n'}/ 2^{m+1} = 2^{2^{m-1}-1}$ ดังนั้น ภายใต้การแบ่งกลุ่มแบบ GCD จะมีโหนดเริ่มต้น (Starting Nodes) ของสอง GCD ที่อยู่ติดกันสามารถหาได้โดย โหนดแรกของ GCD_i = โหนดสุดท้ายของ GCD_{i-1} + 1 เมื่อกำหนดให้ $GCD_0 = 0$



ดังนั้น $GCD_i = GCD_{i-1} + \sum_{i=0}^{2^{m-1}+m-1} 2^i + 1$

$= GCD_{i-1} + (2^{2^{m-1}+m-1+1} - 1) + 1$

$= GCD_{i-1} + 2^{2^{m-1}+m}$

เมื่อ $\sum_{i=0}^{2^{m-1}+m-1} 2^i$ คือ เลขฐาน 10 ของ $b_{2^{m-1}+m-1} \dots b_1 b_0$ = 1111...11 (ค่ามากที่สุด) ของ GCD_{i-1} ซึ่งจะมีทั้งหมด $2^{m+1}+m$ บิต □

หลังจากที่ได้วิธีการหา GCD_i จากสมการที่ 3.1 ขั้นตอนต่อไป คือ การหาโหนดเริ่มต้น (S_p) ที่อยู่ในแต่ละ GCD_i เมื่อ i = 0, 1, ..., $2^{2^{m-1}+1} - 1$ และ p = 0, 1, ..., $2^m - 1$ โดยใช้ขั้นตอนวิธีที่ 3.4 ซึ่งคือการหาดูอัล-คิวบ์ที่อยู่ภายใน $Cross_0$ ของกลุ่มนั้นๆก่อนเป็นอันดับแรก สำหรับ $Cross_0$ ของกลุ่มแรกจะประกอบด้วย dCube1 = GCD_0 = 0 และ dCube2 = GCD_0 + โหนดสุดท้ายของกลุ่ม - |Cube| = $2^{2^{m-1}+m} - 2^m$ = 16 – 4 = 12 ดังแสดงในรูปที่ 3.12 ข)

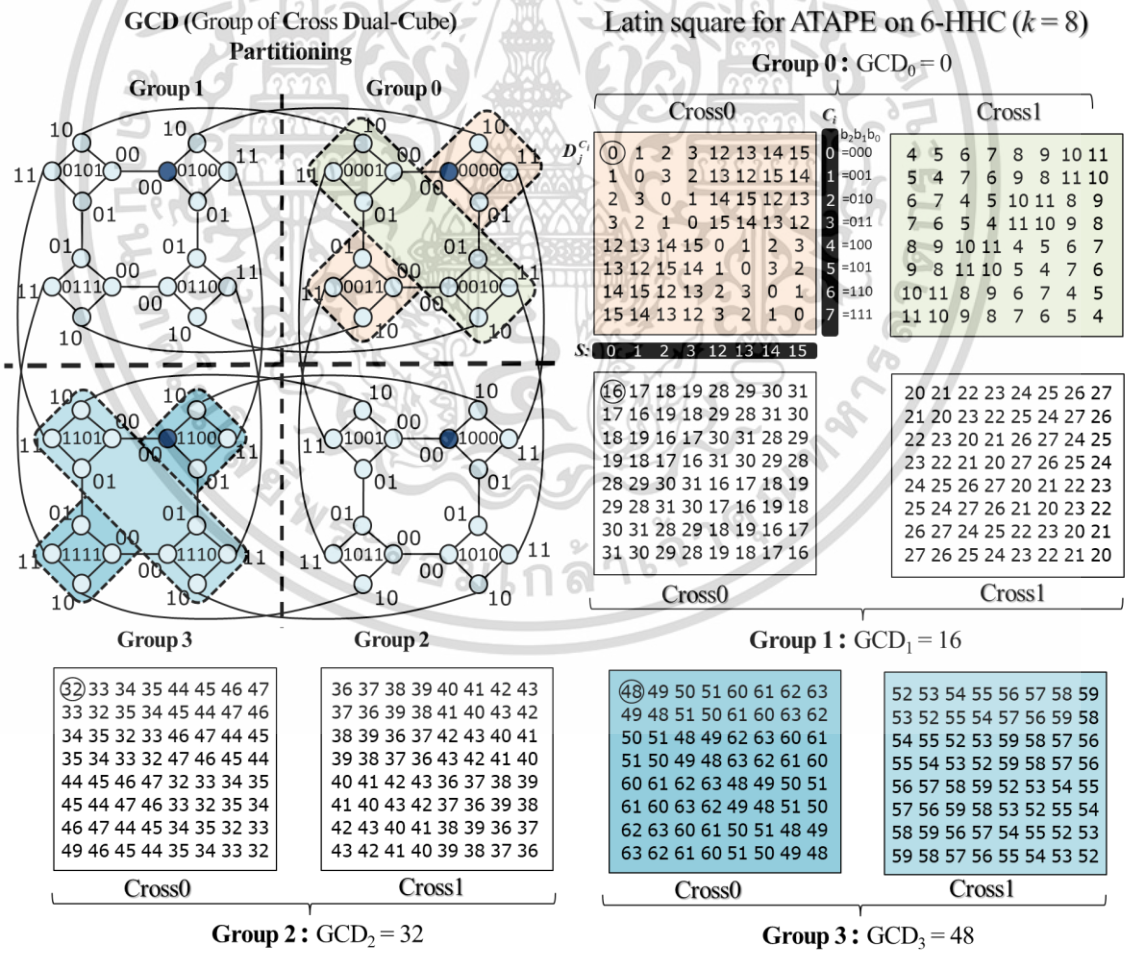
ขั้นตอนวิธี 3.4 ขั้นตอนการหาโหนดเริ่มต้น (Start Nodes) ที่อยู่ในแต่ละ GCD_i

```
GetSourceOfSubSystem
1. dCube1 = GCD_i; dCube2 = GCD_i + 2^(2^(m-1)+m) – 2^m;
2. for ( cross j = 0; j< #crosses; j++) do          // 2 cubes per cross
3.       for ( processor p=0; p < 2^m; p++) do // 2 × 2^m nodes
4.           S[ p ] = dCube1 + p;  // S on dual cube 1 (of cross_j)
5.           S[ p+ 2^m] = dCube2 + p; // S on dual cube 2 (of cross_j)
6.       end for
7.       dCube1 = dCube1 + 2^m;
8.       dCube2 = dCube2 – 2^m;
9. end for
```

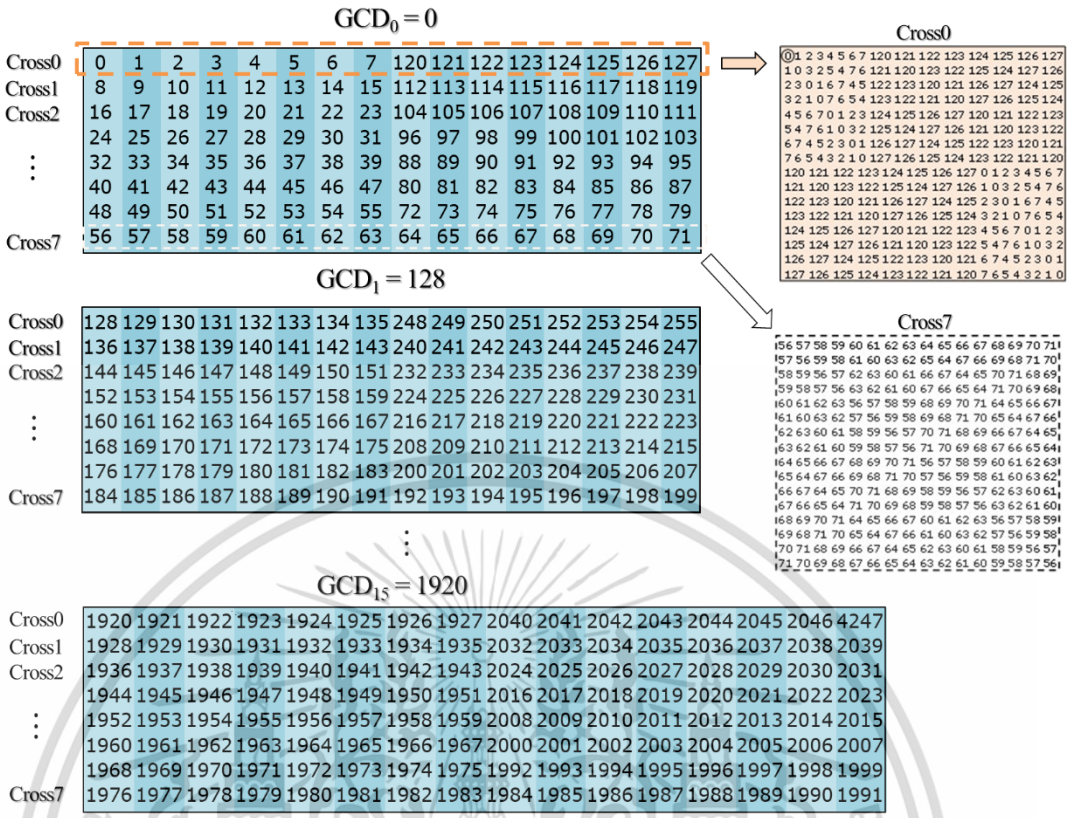
ตัวอย่างการหาโหนดเริ่มต้น (S_{cj}) บน 6-HHC ($N = 2^n$, $n = 2^m+m$) เมื่อ m = 2 และ $k = 2^{m+1}$ = 8 จากสมการที่ 3.1 จะได้ว่า GCD_0 = 0 และ GCD_i ($1 \le i < 4$) = 16, 32 และ 48 ตามลำดับ สำหรับ GCD_0 = 0 จะมีจำนวน Cross คือ $2^{2^{m-1}-1}$ = 2 ซึ่งประกอบด้วย $Cross_0$ และ $Cross_1$ ขั้นตอนต่อไปคือการหาโหนดเริ่มต้น ที่อยู่ในแต่ละ Cross จากขั้นตอนวิธีที่ 3.4 เมื่อดำเนินการวนรอบของ j และ cj จะได้ผลลัพธ์ของ $Cross_0$ = {**0**, 1, 2, 3, **12**, 13, 14, 15} (*dCube*1 = 0, *dCube*2 = 12) และ $Cross_1$ = {**4**, 5, 6, 7, **8**, 9, 10, 11} (*dCube*1 = 4, *dCube*2 = 8)

รูปที่ 3.10 แสดงภาพรวมการแบ่งกลุ่มของ $GCD_0 = 0$, $GCD_1 = 16$, $GCD_2 = 32$ และ $GCD_3 = 48$ เมื่อ $m = 2$ พร้อมแสดงจำนวน Cross และ โหนดเริ่มต้นของแต่ละ Cross โดยใช้วิธีการอธิบายมาข้างต้นทั้งหมด และถูกจัดให้อยู่ในแถวแรกของตารางลาติน ซึ่งรายละเอียดของการสร้างตารางลาตินจะถูกอธิบายในหัวข้อ 3.2.2.2

รูปที่ 3.11 แสดงการจัดกลุ่มของ GCD_i เมื่อ $0 \leq i < 16$ และ $m = 3$ ซึ่งประกอบด้วย $GCD_0 = 0$, $GCD_1 = 128$, $GCD_2 = 256$, $GCD_3 = 384$, $GCD_4 = 512$, $GCD_5 = 640$, $GCD_6 = 768$, $GCD_7 = 896$, $GCD_8 = 1024$, $GCD_9 = 1152$, $GCD_{10} = 1280$, $GCD_{11} = 1408$, $GCD_{12} = 1536$, $GCD_{13} = 1664$, $GCD_{14} = 1792$ และ $GCD_{15} = 1920$ ยกตัวอย่างเช่น $GCD_0 = 0$ จะประกอบด้วย $Cross_0$ – $Cross_7$ และในแต่ละ Cross จะมีจำนวนโหนดเริ่มต้นจำนวน 16 โหนด เช่น $Cross_0$ = {0, 1, 2, 3, 4, 5, 6, 7, 120, 121, 122, 123, 124, 125, 126, 127}, $Cross_1$ = {8, 9, 10, 11, 12, 13, 14, 15, 112, 113, 114, 115, 116, 117, 118, 119}, ..., $Cross_7$ = {56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71}



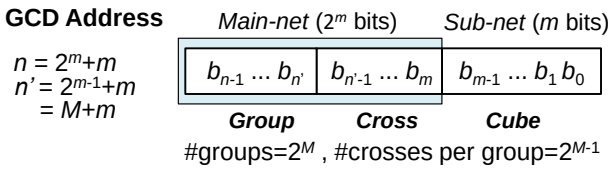
รูปที่ 3.10 ตัวอย่างการจัดกลุ่ม GCD บนเครือข่าย 6-HHC ($n = 6$, $m = 2$) และ $k = 2^{m+1} = 8$



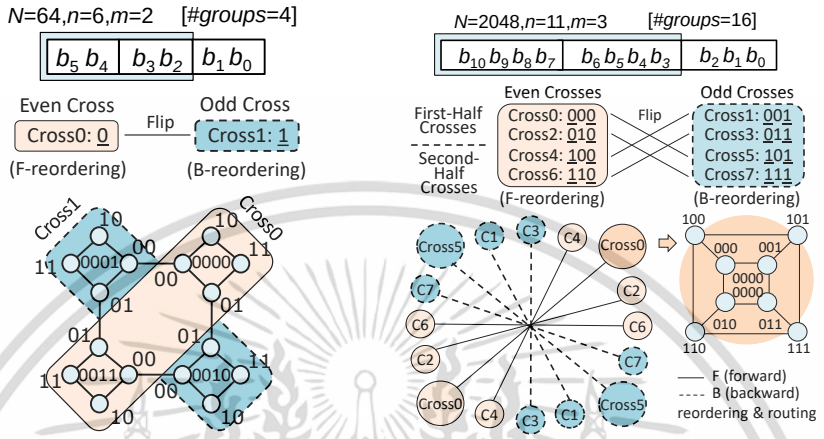
รูปที่ 3.11 ตัวอย่างการจัดกลุ่ม GCD บนเครือข่าย 11-HHC (n = 11, m = 3) และ k = 16

<u>ทฤษฎีบท 3.3</u> ภายใต้การจัดกลุ่มแบบ GCD ที่มีจำนวนกลุ่ม คือ $2^{2^{m-1}}$ จะมี Cross ที่สามารถดำเนินการแลกเปลี่ยนข้อมูลแบบ ATAPE ด้วยหน่วยประมวลผลจำนวน $k = 2^{m+1}$ ได้พร้อมกัน โดยไม่มีความขัดแย้งระหว่างสื่อสาร เมื่อใช้หลักการไขว้เลขคู่ (Even Cross) และการไขว้เลขคี่ (Odd Cross) โดย การไขว้เลขคู่ คือ การหาเส้นทางไปข้างหน้าแบบวนรอบ (Forward Routing) และการไขว้เลขคี่ คือ การหาเส้นทางถอยกลับแบบวนรอบ (Backward Routing)

พิสูจน์ รูปที่ 3.12 ก) แสดงโครงสร้าง และการระบุอินเด็กซ์ระดับบิตของ GCD (GCD Addressing) โดยที่จำนวนของกลุ่มคือ $2^{2^{m-1}}$ และจำนวน Cross ในแต่ละกลุ่ม คือ $2^{2^{m-1}-1}$ การหาเส้นทางที่สั้นที่สุดแบบขนานของการไขว้เลขคู่ สามารถหาเส้นทางโดยใช้ขั้นตอนวิธีที่ 3.3 และฟังก์ชัน SPordering แบบไปข้างหน้าวนรอบ ซึ่งเป็นฟังก์ชันเริ่มต้น (Default function) สำหรับการไขว้เลขคี่ จะใช้ฟังก์ชัน SPordering แบบถอยกลับวนรอบ โดยทั้งสองฟังก์ชันคือฟังก์ชันที่จะช่วยให้สามารถหลีกเลี่ยงการขัดแย้งกันขณะรับส่งข้อมูลแบบ ATAPE เนื่องจากทั้งสองฟังก์ชันจะรับส่งข้อมูลในทิศทางที่ตรงกันข้ามกัน (Symmetrical Path) ดังนั้นทุกๆ Cross ในแต่ละกลุ่ม สามารถสื่อสารแบบ ATAPE ได้พร้อมกัน โดยที่ไม่มีการขัดแย้งกัน รูปที่ 3.12 ข) แสดงคู่ของ Cross ที่มีทิศทางตรงข้ามกัน เมื่อ m = 2 คือ $Cross_0$ และ $Cross_1$ หรือรูปที่ 3.12 ค) เมื่อ m = 3 จะมีคู่ของ Cross ที่มีทิศทางตรงกันข้ามกัน คือ $Cross_0$ และ $Cross_5$, $Cross_2$ และ $Cross_1$, $Cross_4$ และ $Cross_7$, $Cross_6$ และ $Cross_3$ โดยทุกๆคู่จะสามารถใช้ประโยชน์จากการรับส่งข้อมูลแบบสองทิศทางได้อย่างเต็มประสิทธิภาพ □



ก)

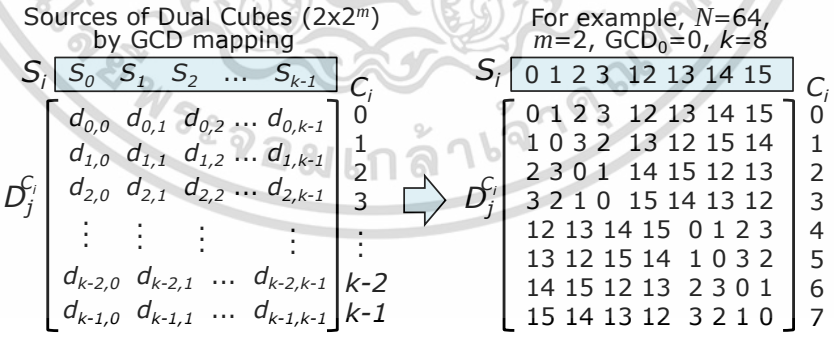


ข) ค)

รูปที่ 3.12 ก) โครงสร้างระดับบิตของ GCD และ ข) ตัวอย่างการ Cross เมื่อ m = 2 และ ค) m = 3

3.2.2.2 การสร้างตารางลาตินเพื่อการสื่อสารแบบ ATAPE (A Latin-Square for ATAPE)

ในหัวข้อนี้จะนำเสนอการสร้างตารางลาติน (Latin-Square : LS) โดยตารางลาตินจะแสดงวิธีการเรียงสับเปลี่ยนที่เป็นไปได้ของโหนดปลายทาง (Destination Node : D) ซึ่งถูกคำนวณมาจากโหนดเริ่มต้น จำนวน $k = 2^{m+1}$ จากขั้นตอนวิธีที่ 3.4 โดยตารางลาตินจะมีโครงสร้างดังรูปที่ 3.13



รูปที่ 3.13 โครงสร้างของตารางลาตินของโหนดปลายทาง $D_j^{C_i}$

ตารางลาตินจะแสดงการเรียงสับเปลี่ยน เพื่อให้ได้รอบการติดต่อสื่อสารแบบ ATAPE ระหว่างโหนดต้นทาง S และโหนดปลายทาง $D_j^{C_i}$ ในแต่ละรอบตัวควบคุม (Control Unit : C_i) ที่มีจำนวน k รอบ ($0 \le i \le k$-1) ดังนั้นตารางลาตินของการ ATAPE บนเครือข่าย HHC จะมีขนาด

เท่ากับ $k \times k$ เมื่อ $k = 2^{m+1}$ และค่าโหนดปลายทาง $D_j^{C_i} = d_{i,j}$ คำนวณได้จากสมการที่ 3.3 ซึ่งเป็นสมการที่ถูกปรับปรุงมาจากฟังก์ชัน $D = S \oplus C$ ที่ถูกเสนอไว้ใน [18]

$$D_j^{C_i} = d_{i,j} = S_{(C_i \oplus j)} \tag{3.3}$$

เมื่อ $d_{i,j}$ คือ โหนดปลายทาง (Destination Node) ที่ i และ j เมื่อ ($0 \le i, j \le k$-1)

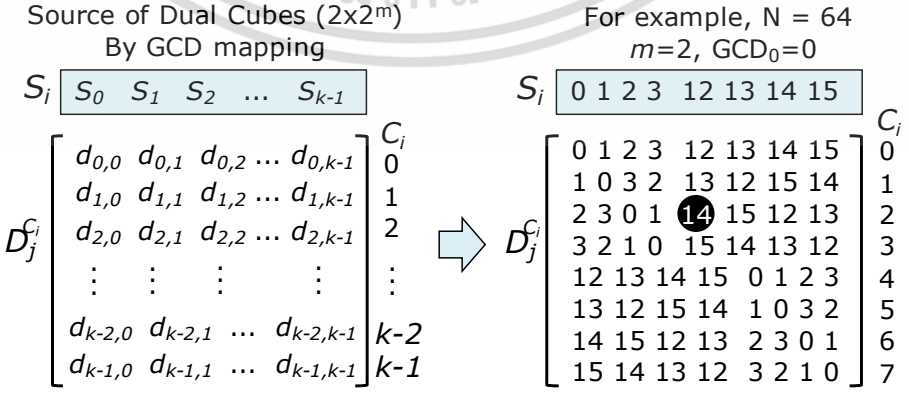
S_i คือ โหนดต้นทาง (Source Node) ที่ i

C_i คือ ตัวควบคุม ที่ i

และ $\oplus$ คือ ตัวดำเนินการ XOR (XOR Operation)

เนื่องจากการใช้ฟังก์ชัน XOR โดยตรงที่เหมาะสมกับการสื่อสารบนเครือข่าย HC จะไม่สามารถนำมาสร้างตารางลาตินสำหรับการ ATAPE บนเครือข่าย HHC ได้ เนื่องจากเครือข่าย HHC มีโครงสร้างที่มีความซับซ้อนมากกว่า ดังนั้นการสร้างตารางลาตินที่เสนอในวิทยานิพนธ์ฉบับนี้ ซึ่งจะเรียกว่าฟังก์ชัน XOR-LS ที่อินเด็กซ์ (i, j) หรือสมการที่ 3.3 จะสามารถฝังการสื่อสารแบบ ATAPE ที่มีขนาดของหน่วยประมวลผล $k = 2^{m+1}$ บนเครือข่าย HHC ได้อย่างสอดคล้องและไม่มีการขัดแย้งกันขณะมีการสื่อสารของหน่วยประมวลผล

ตัวอย่าง กำหนดให้ $GCD_0 = 0$ บนเครือข่าย 6-HHC ($N = 64$, $n = 6$ และ $m = 2$) จะได้ว่าจำนวนหน่วยประมวลผล $k = 8$ ดังนั้น โหนดต้นทาง S ถูกสร้างโดยใช้ขั้นตอนวิธีที่ 3.4 ซึ่งจะได้กลุ่มของโหนดต้นทาง $S = \{\underline{0}, 1, 2, 3, \underline{12}, 13, 14, 15\}$ เมื่อ $C_i = 0$ จะได้โหนดปลายทางแถวแรกของตารางลาติน ดังรูปที่ 3.14 ขั้นตอนต่อมาใช้สมการที่ 3.3 หาค่าของโหนดปลายทาง เช่น ในกรณี C_2 หรือแถวที่ 3 ของตารางลาติน จะได้ว่า $d_{i,j} = \{2, 3, 0, 1, 14, 15, 12, 13\}$ โดย $d_{2,4} = 14$ ถูกคำนวณโดย $i = 2, j = 4$ และ $C_i = 2$ จะมี ค่าของ $d_{2,4} = S_{(2 \oplus 4)} = S_6 = 14$ สำหรับค่าอื่นๆ ในแต่ละแถวสามารถคำนวณได้ในลักษณะเดียวกัน



รูปที่ 3.14 ตารางลาตินสำหรับการสื่อสารแบบ ATAPE บนเครือข่าย HHC เมื่อ $m = 2$

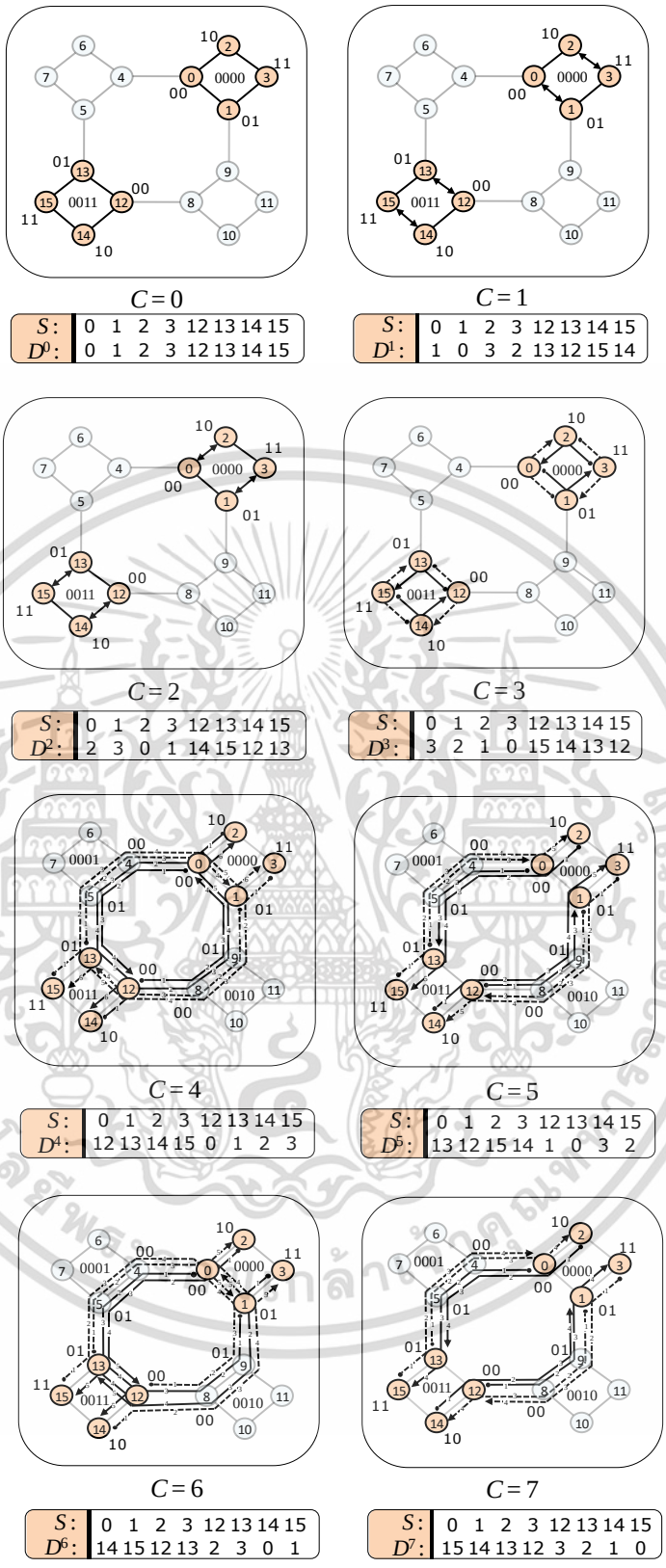
รูปที่ 3.10 แสดงตารางลาตินของการเรียงสับเปลี่ยนที่เป็นไปได้ บนเครือข่าย 6-HHC เมื่อ m = 2 และ k = 8 สังเกตว่าเมื่อ GCD_0 = 0 จะมีจำนวน Cross คือ 2 ประกอบด้วย $Cross_0$ และ $Cross_1$ โดย $Cross_0$ ประกอบด้วยโหนดชั้นนอก (External Node) คือ 0000 และ 0011 ดังนั้นจะมีโหนดเริ่มต้น (Starting Node) ประกอบด้วย 0, 1, 2, 3, 12, 13, 14 และ 15 ตามลำดับ สำหรับ $Cross_1$ ประกอบด้วยโหนดชั้นนอก คือ 0001 และ 0010 ดังนั้นจะมีโหนดเริ่มต้นประกอบด้วย 4, 5, 6, 7, 8, 9, 10 และ 11 ตามลำดับ

ตารางที่ 3.1 และรูปที่ 3.15 แสดงเส้นทางการสื่อสารแบบ ATAPE โดยการใช้ขั้นตอนวิธีที่ 3.3 และการเรียงลำดับไปข้างหน้าแบบวนรอบ บนเครือข่าย 6-HHC เมื่อ m = 2, k = 8 และ GCD_0 = 0 โดยการใช้โหนดต้นทางที่อยู่ใน Cross เลขคู่ ซึ่งก็คือ $Cross_0$ ดังนั้นจะมีโหนดเริ่มต้นประกอบด้วย 0, 1, 2, 3, 12, 13, 14 และ 15 เช่น โหนดต้นทาง S = 15(0011, 11) และปลายทาง D = 3(0000,11) บน C = 4 จะได้เส้นทางที่สั้นที่สุดตามขั้นตอนวิธีที่ 3.3 ดังต่อไปนี้

$S = 15 = (0011, 11) \xrightarrow{*} (0011, 01) \longrightarrow (0001, 01) \xrightarrow{*} (0001, 00) \xrightarrow{*} (0000, 00) \longrightarrow (0000, 01) \xrightarrow{*} (0000, 11) = 3 = D$

ตารางที่ 3.1 เส้นทางการสื่อสารแบบ ATAPE ในแต่ละตัวควบคุม (C) เมื่อ $Cross_0$ บน GCD_0=0

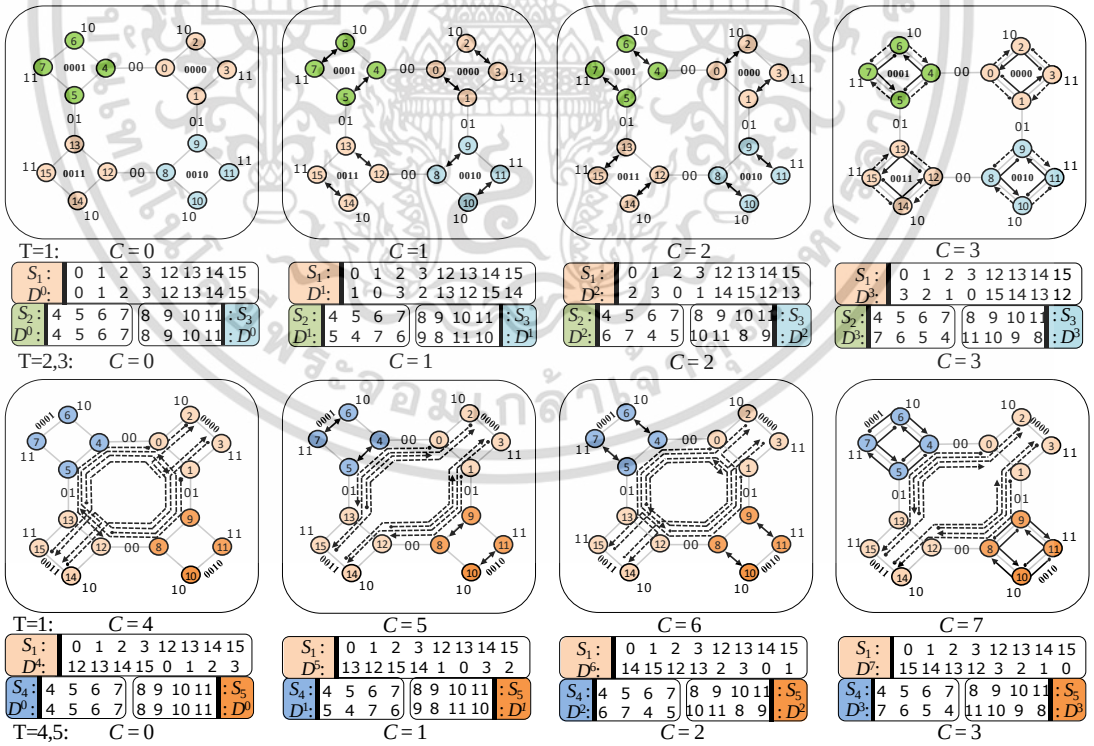
C=0	S=D	0:0000,00	1:0000,01	2:0000,10	3:0000,11	12:0011,00	13:0011,01	14:0011,10	15:0011,11
C=1	S	0:0000,00	1:0000,01	2:0000,10	3:0000,11	12:0011,00	13:0011,01	14:0011,10	15:0011,11
	D	1:0000,01	0:0000,00	3:0000,11	2:0000,10	13:0011,01	12:0011,00	15:0011,11	14:0011,10
C=2	S	0:0000,00	1:0000,01	2:0000,10	3:0000,11	12:0011,00	13:0011,01	14:0011,10	15:0011,11
	D	2:0000,10	3:0000,11	0:0000,00	1:0000,01	14:0011,10	15:0011,11	12:0011,00	13:0011,01
C=3	S	0:0000,00	1:0000,01	2:0000,10	3:0000,11	12:0011,00	13:0011,01	14:0011,10	15:0011,11
		0000,01(in)	0000,00(in)	0000,11(in)	0000,10(in)	0011,01(in)	0011,00(in)	0011,11(in)	0011,10(in)
	D	3:0000,11	2:0000,10	1:0000,01	0:0000,00	15:0011,11	14:0011,10	13:0011,01	12:0011,00
$1C$=4	S	0:0000,00	1:0000,01	2:0000,10	3:0000,11	12:0011,00	13:0011,01	14:0011,10	**15:0011,11**
		0001,00(ex) 0001,01(in) 0011,01(ex)	0010,01(ex) 0010,00(in) 0011,00(ex)	0000,00(in) 0001,00(ex) 0001,01(in) 0011,01(ex) 0011,00(in)	0000,01(in) 0010,01(ex) 0010,00(in) 0011,00(ex) 0011,01(in)	0010,00(ex) 0010,01(in) 0000,01(ex)	0001,01(ex) 0001,00(in) 0000,00(ex)	0011,00(in) 0010,00(ex) 0010,01(in) 0000,01(ex) 0000,00(in)	0011,01(in) 0001,01(ex) 0001,00(in) 0000,00(ex) 0000,01(in)
	D	12:0011,00	13:0011,01	14:0011,10	15:0011,11	0:0000,00	1:0000,01	2:0000,10	**3:0000,11**
C=5	S	0:0000,00	1:0000,01	2:0000,10	3:0000,11	12:0011,00	13:0011,01	14:0011,10	15:0011,11
		0001,00(ex) 0001,01(in)	0010,01(ex) 0010,00(in)	0000,00(in) 0001,00(ex) 0001,01(in) 0011,01(ex)	0000,01(in) 0010,01(ex) 0010,00(in) 0011,00(ex)	0010,00(ex) 0010,01(in)	0001,01(ex) 0001,00(in)	0011,00(in) 0010,00(ex) 0010,01(in) 0000,01(ex)	0011,01(in) 0001,01(ex) 0001,00(in) 0000,00(ex)
	D	13:0011,01	12:0011,00	15:0011,11	14:0011,10	1:0000,01	0:0000,00	3:0000,11	2:0000,10
C=6	S	0:0000,00	1:0000,01	2:0000,10	3:0000,11	12:0011,00	13:0011,01	14:0011,10	15:0011,11
		0001,00(ex) 0001,01(in) 0011,01(ex) 0011,00(in)	0010,01(ex) 0010,00(in) 0011,00(ex) 0011,01(in)	0000,00(in) 0001,00(ex) 0001,01(in) 0011,01(ex)	0000,01(in) 0010,01(ex) 0010,00(in) 0011,00(ex)	0010,00(ex) 0010,01(in) 0000,01(ex) 0000,00(in)	0001,01(ex) 0001,00(in) 0000,00(ex) 0000,01(in)	0011,00(in) 0010,00(ex) 0010,01(in) 0000,01(ex)	0011,01(in) 0001,01(ex) 0001,00(in) 0000,00(ex)
	D	14:0011,10	15:0011,11	12:0011,00	13:0011,01	2:0000,10	3:0000,11	0:0000,00	1:0000,01
C=7	S	0:0000,00	1:0000,01	2:0000,10	3:0000,11	12:0011,00	13:0011,01	14:0011,10	15:0011,11
		0001,00(ex) 0001,01(in) 0011,01(ex)	0010,01(ex) 0010,00(in) 0011,00(ex)	0000,00(in) 0001,00(ex) 0001,01(in)	0000,01(in) 0010,01(ex) 0010,00(in)	0010,00(ex) 0010,01(in) 0000,01(ex)	0001,01(ex) 0001,00(in) 0000,00(ex)	0011,00(in) 0010,00(ex) 0010,01(in)	0011,01(in) 0001,01(ex) 0001,00(in)
	D^7	15:0011,11	14:0011,10	13:0011,01	12:0011,00	3:0000,11	2:0000,10	1:0000,01	0:0000,00



รูปที่ 3.15 เส้นทางการสื่อสารแบบ ATAPE ในแต่ละตัวควบคุม (*C*) เมื่อ $Cross_0$ บน GCD_0=0

การจัดกลุ่มแบบ GCD ไม่เพียงแก้ปัญหาความขัดแย้งในขณะสื่อสารระหว่างหน่วยประมวลผล แต่ยังสามารถช่วยให้หน่วยประมวลผล ขนาด k ที่อยู่ในแต่ละ Cross ของแต่ละ GCD สามารถสื่อสารได้พร้อมกันอย่างสอดคล้อง โดยใช้หลักการที่ได้อธิบายไปก่อนหน้า คือ Cross เลขคู่ใช้การหาเส้นทางไปข้างหน้าแบบวนรอบ และ Cross เลขคี่ใช้การหาเส้นทางถอยกลับแบบวนรอบ โดยทั้งสองเส้นทางที่ได้จากขั้นตอนวิธีที่ 3.3 จะเป็นเส้นทางแบบสองทิศทางที่มีคุณสมบัติสมมาตร

รูปที่ 3.16 แสดงตัวอย่างการ ATAPE บนเครือข่าย 6-HHC (N = 64, n = 6 และ m = 2) ด้วยการใช้ 5 งาน (Tasks) ที่อยู่ใน GCD_0 โดยงานแรกประกอบด้วยหน่วยประมวลผลที่มีขนาด $k = 2^{m+1}$ และอีก 4 งานประกอบด้วยหน่วยประมวลผลที่มีขนาด $k = 2^m$ งานแรกถูกแทนด้วย T_1 จะถูกมอบหมายให้โหนดที่อยู่ใน $Cross_0$ ประกอบด้วย โหนด 0 – 3 และ 12 – 15 (โหนดสีส้มอ่อน) โดยจะใช้ตัวควบคุม C = 0 – 7 สำหรับงานที่ 2 และ 3 ถูกแทนด้วย T_2 และ T_3 ซึ่งมีขนาด k = 4 จะถูกมอบหมายให้โหนด 4 – 7 (โหนดสีเขียว) และ 8 – 11 (โหนดสีฟ้า) ตามลำดับ และจะใช้ตัวควบคุม C = 0 – 3 หลังจาก T_2 และ T_3 ประมวลผลเสร็จ หน่วยประมวลผลนั้นจะว่างสำหรับงานที่ 4 และ 5 โดยถูกแทนด้วย T_4 และ T_5 ที่มี k = 4 ซึ่งจะถูกมอบหมายให้โหนด 4 – 7 (โหนดสีน้ำเงิน) และ 8 – 11 (โหนดสีส้มเข้ม) ตามลำดับ ด้วยตัวควบคุม C = 0 – 3 สังเกตว่าถึงแม้ว่างานทั้งหมดจาก T_1 – T_5 จะมีขนาดของหน่วยประมวลผล k ที่ต่างกัน แต่สามารถสื่อสารแบบ ATAPE พร้อมกันแบบขนานได้โดยไม่มีการขัดแย้งกัน



รูปที่ 3.16 เส้นทางการสื่อสารแบบ ATAPE ในแต่ละตัวควบคุม (C) ของงาน T_1 – T_5 บน GCD_0=0

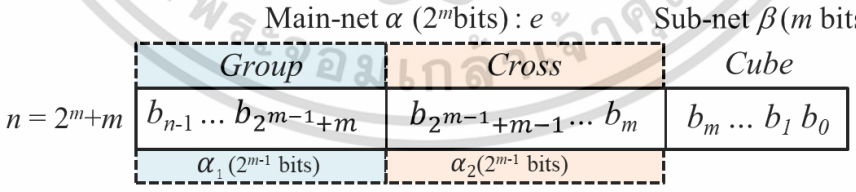
3.3 การแบ่งกลุ่มแบบ GCD และ GCS ที่ปรับเปลี่ยนได้ บนเครือข่าย HHC ด้วยหน่วยประมวลผลขนาด $k \geq 2^{m+1}$

ในหัวข้อนี้นำเสนอการแบ่งกลุ่มที่สามารถปรับเปลี่ยนได้ บนเครือข่าย HHC ด้วยวิธีการแบ่งกลุ่มแบบ GCD และการรวมกลุ่มคิวบ์ย่อยแบบไขว้ (Grouping of Cross Sub-cube : GCS) สำหรับหน่วยประมวลผล $k \geq 2^{m+1}$ โหนด ในหัวข้อย่อย 3.3.1 - 3.3.3 นำเสนอ บิตไวส์-GCD (Bitwise-GCD) สำหรับหน่วยประมวลผลขนาด $k = 2^{m+1}$, บิตไวส์-GCS (Bitwise GCS) สำหรับหน่วยประมวลผลขนาด $k = 2^{m+2}$ และ ซูเปอร์-GCS (Super-GCS) สำหรับหน่วยประมวลผลขนาด $k = 2^{m+3}$ ถึง $k = 2^n/2^{M-1}$ เมื่อ $n = 2^m+m$ และ $M = 2^{m-1}$

3.3.1 การแบ่งกลุ่มแบบบิตไวส์-GCD สำหรับหน่วยประมวลผลขนาด $k = 2^{m+1}$

หัวข้อ 3.2.2.1 ได้อธิบายการจัดกลุ่มของหน่วยประมวลผลแบบ GCD ซึ่งเป็นการจัดกลุ่มของหน่วยประมวลผลด้วยรูปแบบที่ไม่สามารถปรับเปลี่ยนได้ ด้วยสมการที่ 3.1 และแสดงตัวอย่างเมื่อ $m = 2$ ในรูปที่ 3.10 ซึ่งในหัวข้อนี้ จะนำเสนอวิธีที่มีความยืดหยุ่นมากขึ้น โดยใช้วิธีที่เรียกว่า การแบ่งกลุ่มแบบบิตไวส์-GCD ที่ต้องมีโหนดชั้นนอกที่ยังว่างอยู่หนึ่งโหนด ขนาด 2^m ที่เรียกว่า คิวบ์ย่อย (Sub-cube: sc) และทำการหาคู่ดูอัล-คิวบ์ (Dual-cube : dc) ขนาด 2^m ของ sc เพื่อให้ได้หน่วยประมวลผลทั้งหมด $k = 2^{m+1}$ โหนด เพื่อประกอบเป็น Cross ดังนั้นบิตไวส์-GCD จะเป็นการจัดกลุ่มระดับบิตเพื่อระบุถึง Cross ที่ต้องการอย่างเฉพาะเจาะจงได้ทันที (Runtime) บนเครือข่าย HHC

จากหัวข้อ 3.1.1 ระบุว่าโหนดใดๆ บนเครือข่าย HHC จะถูกอ้างอิงด้วย 2^m บิต ของ α และ m บิต ของ β เช่น 0001,10 คือ โหนดลำดับที่หก บน 6-HHC แต่ในหัวข้อนี้จะแบ่ง α บิต ออกเป็น 2 ส่วนเพื่ออ้างถึงการอยู่ใน Group (บิตสีฟ้า) และ Cross (บิตสีส้ม) ของโหนดที่ระบุ ดังรูปที่ 3.17



รูปที่ 3.17 การระบุอินเด็กซ์ระดับบิตของหน่วยประมวลผล เมื่อ $\alpha = \alpha_1 \alpha_2$ บนโหนดชั้นนอก (Main-net)

รูปที่ 3.17 อธิบายการระบุอินเด็กซ์ระดับบิตของ Group (2^{m-1} บิต, สีฟ้า) และ Cross (2^{m-1} บิต, สีส้ม) ของหน่วยประมวลผลที่อยู่บนเครือข่าย n-HHC โดย

กำหนดให้ α_1 คือ เลขฐานสองที่ใช้อ้างถึงการอยู่ใน Group ซึ่งประกอบด้วย 2^{m-1} บิต

α_2 คือ เลขฐานสองที่ใช้อ้างถึงการอยู่ใน Cross ซึ่งประกอบด้วย 2^{m-1} บิต

เช่น โหนด 7 บนเครือข่าย 6-HHC จะถูกแทนด้วย (0001,11) ดังนั้น อินเด็กซ์ของโหนดชั้นนอก คือ $\alpha = \alpha_1\alpha_2$ = 0001 เมื่อ α_1 = 00 และ α_2 = 01 และการหา dc ขนาด 2^m ด้วยการใช้ sc ขนาด 2^m เพื่อประกอบเป็น Cross ขนาด $k = 2^{m+1}$ บน GCD สามารถหาได้จากสมการที่ 3.4

$$\begin{aligned} e &= \alpha_1\, \alpha_2\,, \\ d &= \alpha_1\, \bar{\alpha}_2 \end{aligned} \qquad (3.4)$$

เมื่อ e คือ คิวบ์ย่อยเริ่มต้น (Input Sub-cube or External node)

d คือ ผลลัพธ์ดูอัล-คิวบ์ (Output Dual-cube)

$\bar{\alpha}_2$ คือ คอมพลีเมนต์บิต (Complement) ของ α_2

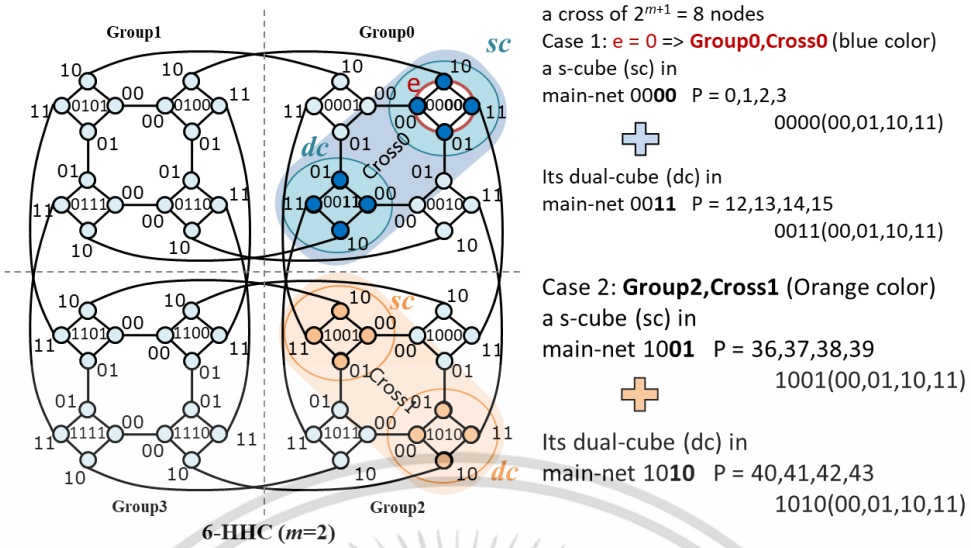
จากสมการที่ 3.4 จะได้ว่า โหนดของ sc คือ (e, xx...x) = ($\alpha_1\, \alpha_2$, xx...x) และ โหนดของ dc คือ (d, xx...x) = ($\alpha_1\, \bar{\alpha}_2$, xx...x) เมื่อ xx...x หมายถึง เลขฐานสองระดับบิตของโหนดชั้นใน เช่น xx = {00, 01, 10, 11} เมื่อ m = 2 ดังนั้น หน่วยประมวลผลจำนวน $k = 2^{m+1}$ ที่ประกอบด้วย sc และ dc จะสามารถสื่อสารเพื่อรับส่งข้อมูลกันโดยปราศจากการขัดแย้งกัน

ตัวอย่างเช่น บนเครือข่าย 6-HHC (N = 64, m = 2) ในแต่ละ Group α_1 จะประกอบด้วย 2 Cross คือ $Cross_0$ และ $Cross_1$ โดยที่แต่ละ Cross จะถูกสร้างจากบิตของ Cross คือ α_2 และ $\bar{\alpha}_2$ (ดูจากตารางที่ 3.2 ประกอบ)

ตารางที่ 3.2 การดำเนินการ บิตไวส์-GCD สำหรับ $k = 2^{m+1}$ = 8 เมื่อ N = 64, m = 2

Group α_1	Cross α_2	α_2	$\bar{\alpha}_2$	β	2^{m+1}= 8 Processors
0 : 00	*cross*0	00	11	00,01,10,11	0, 1, 2, 3, 12,13,14,15
	*cross*1	01	10	00,01,10,11	4, 5, 6, 7, 8, 9, 10, 11
1 : 01	*cross*0	00	11	00,01,10,11	16,17,18,19, 28,29,30,31
	*cross*1	01	10	00,01,10,11	20,21,22,23, 24,25,26,27
2 : 10	*cross*0	00	11	00,01,10,11	32,33,34,35, 44,45,46,47
	*cross*1	01	10	00,01,10,11	36,37,38,39, 40,41,42,43
3 : 11	*cross*0	00	11	00,01,10,11	48,49,50,51, 60,61,62,63
	*cross*1	01	10	00,01,10,11	52,53,54,55, 56,57,58,59

ตาราง 3.2 แสดงการแบ่งกลุ่มของหน่วยประมวลผลแบบบิตไวส์-GCD เพื่อที่จะสร้างกลุ่มย่อย Cross ขนาด $k = 2^{m+1}$ โดยเมื่อ m = 2 จะได้ Group $i = \alpha_1$ (= 00, 01, 10, 11) ในแต่ละ Group i ประกอบด้วย $Cross_0$ หรือ α_2 = 00 และ $Cross_1$ หรือ α_2 = 01 และใน $Cross_0$ จะประกอบด้วย sc = ($\alpha_1$00, xx) และ dc = ($\alpha_1$11, xx) ดั้งนั้น Group α_1 = 00 จะประกอบด้วย $Cross_0$ = {sc, dc} = { ($\alpha_1$00, xx), ($\alpha_1$11, xx) } = {(0000, xx), (0011, xx)} หรือเมื่อแทนด้วยเลขฐานสองจะได้ {(0, 1, 2, 3), (12, 13, 14, 15)} และ $Cross_1$ = {sc, dc} = {($\alpha_1$01, xx), ($\alpha_1$10, xx) } = {(0001, xx), (0010, xx)} = {(4, 5, 6, 7), (8, 9, 10, 11)}



รูปที่ 3.18 ตัวอย่างของการแบ่งกลุ่มแบบ GCD ขนาด k = 8 บนเครือข่าย 6-HHC (N = 64, m = 2)

รูปที่ 3.18 แสดง 2 ตัวอย่างในการแบ่งกลุ่มแบบ บิตไวส์-GCD โดยในตัวอย่างแรก ถ้ามีโหนดชั้นนอก (Main-net) ว่างอยู่ คือ sc = 0000 นั้นคือ Group α_1 = 00 และ Cross α_2 = 00 ดังนั้นคู่ดูอัล-คิวบ์ (dc) ของ sc = 0000 คือ $\alpha_1 \bar{\alpha}_2$ = 0011 (หมายถึง อยู่กลุ่มเดียวกัน α_1 = 00 แต่ค่าของ 2 บิตหลัง คือ = 11) ดังนั้น หน่วยประมวลผลทั้งหมดของ $Cross_0$ ในตัวอย่างแรก ซึ่งมี 8 หน่วยประมวลผล ประกอบด้วย 4 หน่วยประมวลผลแรกของ $\alpha_1\alpha_2$xx = 0000, xx = {(0000,00), (0000,01), (0000,10), (0000,11)} = {0, 1, 2, 3} และ 4 หน่วยประมวลผลสุดท้ายของ $\alpha_1 \bar{\alpha}_2$ xx = 0011, xx = {(0011,00), (0011,01), (0011,10), (0011,11)} = {12, 13, 14, 15} ดังนั้น $Cross_0$ = {0, 1, 2, 3, 12, 13, 14, 15} (รูปที่ 3.18 โหนดสีฟ้า) สำหรับในตัวอย่างที่ 2 ถ้าโหนดชั้นนอกที่ว่างอยู่ คือ sc = 1001 จะได้ dc = $\alpha_1 \bar{\alpha}_2$ = 1010 และจะได้หน่วยประมวลผล $Cross_1$ ทั้งหมด คือ {36, 37, 38, 39, 40, 41, 42, 43} (รูปที่ 3.18 โหนดสีส้ม) ตามลำดับ

3.3.2 การรวมกลุ่มแบบบิตไวส์-GCS สำหรับหน่วยประมวลผลขนาด $k = 2^{m+2}$

การรวมกลุ่มแบบบิตไวส์-GCS ถูกเสนอขึ้นมาเพื่อรองรับงานที่ต้องการหน่วยประมวลผลที่มีขนาด $k = 2^{m+2}$ โดยจะทำการเลือกกลุ่มที่ถูกต้องเหมาะสมสำหรับงานที่ต้องการ สังเกตว่า ถ้างานขนาด $k = 2^{m+1}$ การแบ่งกลุ่มของเครือข่าย HHC จะจัดสรรงานอยู่ในกลุ่มที่อยู่ใน GCD เดียวกัน หรือ α_1 คือ ค่าเดียวกัน เช่น α_1 = 00 คือ GCD_0 และจะไม่มีการสื่อสารออกนอกกลุ่ม แต่เมื่อขนาดของหน่วยประมวลผล คือ $k = 2^{m+2}$ จะทำให้มีการรับส่งข้อมูลข้าม GCD เช่น $k = 2^{m+2}$ = 16 บนเครือข่าย 6-HHC และ m = 2 จะมี 2 กลุ่ม GCD ที่อยู่ติดกัน (Adjacent Group) ที่สามารถจัดสรรงานขนาด 16 หน่วยประมวลผลได้ โดยจะเรียกการรวมกลุ่มประเภทนี้ว่า GCS ดังนั้น GCS ขนาด $k = 2^{m+2}$ จะประกอบด้วย 2 Cross ที่มีขนาด 2^{m+1} และอยู่ใน 2 GCD ที่อยู่ติดกัน

กำหนดให้ โหนดชั้นนอก (Main-net) e มีขนาด 2^m บิต คือ $\alpha_1\alpha_2$ ดังนั้นแต่ละ α จะประกอบด้วย 2^{m-1} บิต และ α_1 หมายถึง กลุ่ม หรือ GCD ที่โหนดนั้นอยู่ และ α_2 หมายถึง กลุ่มย่อย หรือ Cross ที่โหนดนั้นอยู่ (รูปที่ 3.17) เช่น $e = \alpha_1\alpha_2 = 1101$ บน 6-HHC หมายถึง โหนด e อยู่ใน GCD_3 ($\alpha_1 = 11$) และ $Cross_1$ ($\alpha_1 = 01$)

ขั้นตอนแรกของการรวมกลุ่มแบบ GCS คือ การหารูปแบบของการ Cross (Cross-Pattern) $p = \alpha_1 \oplus \alpha_2$ จากโหนดชั้นนอก e ซึ่งเป็นข้อมูลเข้า (Input) เนื่องจากก่อนหน้านี้อธิบายว่า GCS ขนาด 2^{m+2} จะประกอบด้วย 2 Cross ขนาด 2^{m+1} ดังนั้น ขั้นตอนต่อไป คือ การหา Cross แรก (First Sub-cube: fs) จากโหนดชั้นนอก e และ สมการที่ 3.4 ต่อมาสำหรับ Cross ที่สอง (Second Sub-cube: ss) ทำการหา GCD ที่อยู่ติดกับ GCD ที่ e มีตำแหน่งอยู่ โดยการใช้สมการที่ 3.5 และ 3.6

$$\text{adj-group } i = \alpha_1 \oplus 2^a \; ; 0 \le a \le M\text{-}1 \tag{3.5}$$

$$\text{cross } j = i \oplus p \quad ; \text{ถ้า } j < 2^{M-1} \text{ เมื่อ } M = 2^{m-1} \tag{3.6}$$

$$\text{หรือ} \qquad \text{cross } j = \overline{i \oplus p} \quad ; \text{กรณีอื่นๆ}$$

ตัวอย่าง บนเครือข่าย 6-HHC ($N = 64$, $m = 2$) ถ้ากำหนดให้ข้อมูลเข้า คือ โหนดชั้นนอก $e = 0000$ (หรือ $\alpha_1\alpha_2 = 0000$) ดังนั้น e อยู่บนกลุ่ม GCD $\alpha_1 = 00$ และ Cross $\alpha_2 = 00$ สำหรับการรวมกลุ่มของงานขนาด $k = 2^{m+2}$ จากสมการที่ 3.5 และ 3.6 จะสามารถสร้าง GCS ได้ $2^{M-1} = 2$ รูปแบบ (Patterns) เนื่องจากโหนดชั้นนอกที่ติดกัน (Adjacent Groups) กับ $e = 0000$ (หรือ GCD_0) คือ adj-group $i = \alpha_1 \oplus 2^a = 01$ และ 10 เมื่อ $a = 0$ และ 1 ตามลำดับ ดังแสดงในรูปที่ 3.19 และตารางที่ 3.3 สำหรับกรณีตั้งต้น (Default) จะใช้ $a = 0$ หรือดูจากรูปที่ 3.19 ดังนั้น เมื่อ $e = 0000$ (หรือ $\alpha_1\alpha_2 = 0000$ จะได้ว่า $\alpha_1 = 00$ และ $\alpha_2 = 00$) และ $p = \alpha_1 \oplus \alpha_2 = 00 \oplus 00 = 0$ จากสมการที่ 3.4 จะได้ดูอัล-คิวบ์ของ e คือ $d = \alpha_1\bar{\alpha}_2 = 0011$ ดังนั้นจะได้ว่าหน่วยประมวลผลทั้งหมดของ fs คือ $\alpha_1\alpha_2$, xx = 0000, (00,01,10,11) = 0, 1, 2, 3 และ $\alpha_1\bar{\alpha}_2$,xx = 0011, (00,01,10,11) = 12, 13, 14, 15 ต่อมาหน่วยประมวลผลทั้งหมดของ ss หาได้จากสมการ 3.5 และ 3.6 จะได้ว่า

$$\text{adj-group } i = \alpha_1 \oplus 2^a \; ; \text{เมื่อ } a = 0$$

$$= 00 \oplus 01 = 01$$

$$\text{และ} \qquad \text{cross } j = i \oplus p$$

$$= 01 \oplus 00 = 01$$

จะได้ว่า โหนดชั้นนอกโหนดแรกของ ss คือ $\alpha_1\alpha_2 = 0101$ (หรือ α_1 = adj-group i = 01 และ α_2 = cross j =01) ดังนั้น หน่วยประมวลผลทั้งหมดของ ss คือ $\alpha_1\alpha_2$, xx = 0101, (00, 01, 10, 11) = 20, 21, 22, 23 และ $\alpha_1\bar{\alpha}_2$,xx = 0101, (00,01,10,11) = 24, 25, 26, 27

สำหรับกรณีที่ a = 1 หรือดูจากรูปที่ 3.19 ซึ่งจะเป็นกรณีที่โหนดชั้นนอก เมื่อ a = 0 ไม่สามารถใช้ได้ ดังนั้น เมื่อ e = 0000 (หรือ $\alpha_1\alpha_2$ = 0000) จะได้ว่าหน่วยประมวลผลของ fs คือ sc = $\alpha_1\alpha_2$, xx = 0000, (00, 01, 10, 11) = 0, 1, 2, 3 และ dc = $\alpha_1\bar{\alpha}_2$,xx = 0011, (00, 01, 10, 11) = 12, 13, 14, 15 ต่อมาหน่วยประมวลผลทั้งหมดของ ss หาได้จากสมการ 3.5 และ 3.6 จะได้ว่า

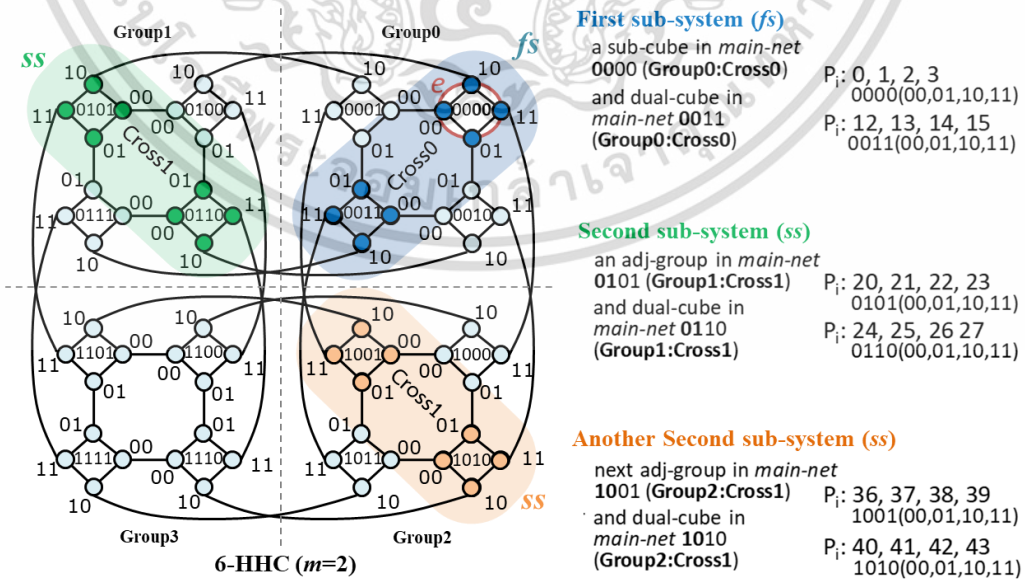
$$\text{adj-group } i = \alpha_1 \oplus 2^a \text{ ; เมื่อ } a = 1$$
$$= 00 \oplus 10 = 10$$

และ
$$\text{cross } j = \overline{i \oplus p}$$
$$= \overline{10 \oplus 00} = 01$$

จะได้ว่า โหนดชั้นนอกโหนดแรกของ ss คือ $\alpha_1\alpha_2$ = 1001 (หรือ α_1 = adj-group i = 10 และ α_2 = cross j =01) ดังนั้น หน่วยประมวลผลทั้งหมดของ ss คือ sc = $\alpha_1\alpha_2$, xx = 1001, (00, 01, 10, 11) = 36, 37, 38, 39 และ dc = $\alpha_1\bar{\alpha}_2$, xx = 1010, (00,01,10,11) = 40, 41, 42, 43

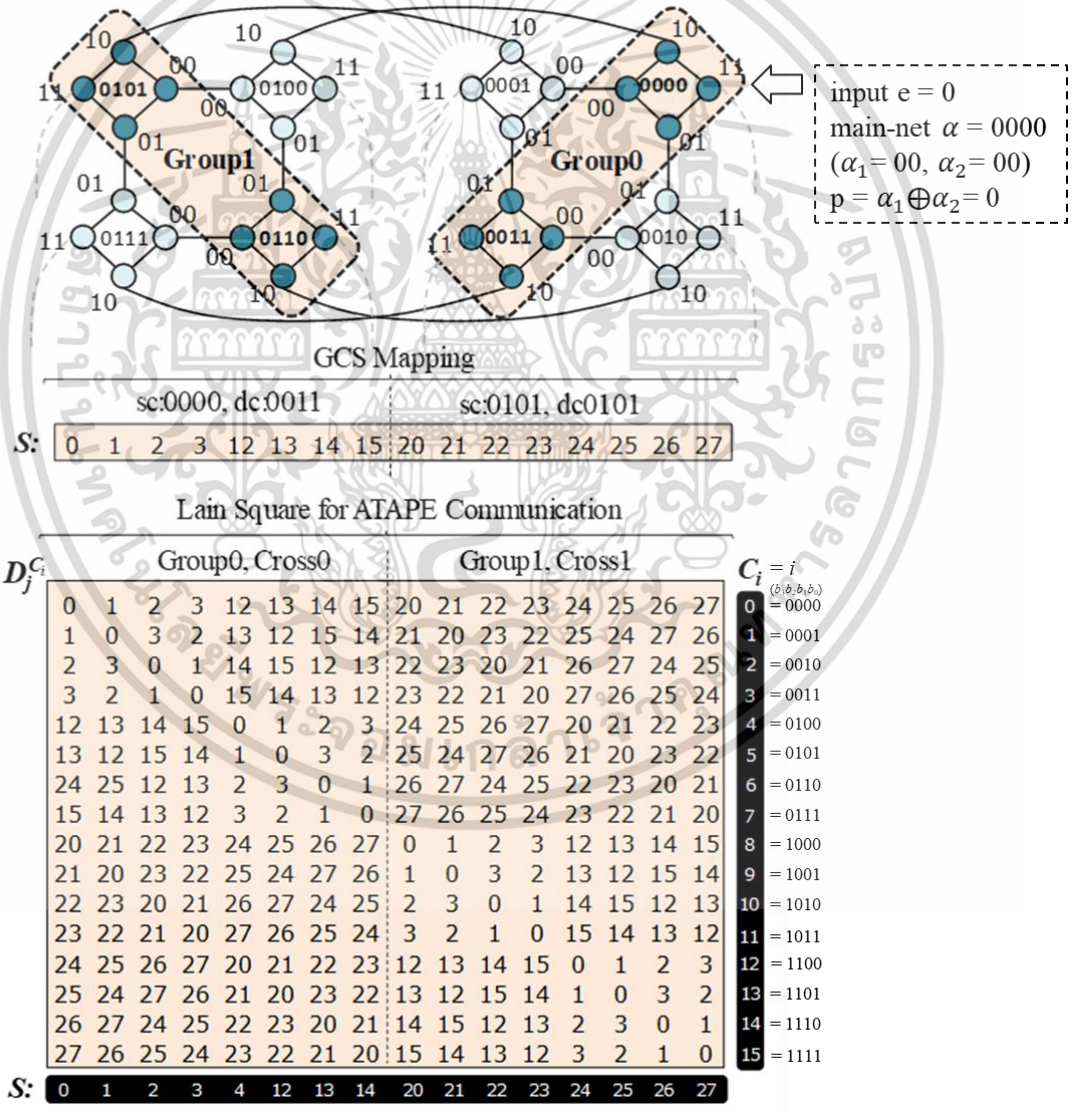
ตารางที่ 3.3 บิตไวส์-GCS สำหรับ $k = 2^{m+2}$ บนเครือข่าย 6-HHC (m = 2 และ e = 0000)

Default sub-system of input group 0 and its adjacent group 1					
Group α_1	Cross α_2	α_2	$\bar{\alpha}_2$	β	2^{m+2} = 16 Processors
0 : 00	*cross*0	00	11	00,01,10,11	0, 1, 2, 3, 12,13,14,15
1 : 01	*cross*1	01	10	00,01,10,11	20,21,22,23,24,25,26,27
Reconfigured sub-system of input group 0 and its adjacent group 2					
Group α_1	Cross α_2	α_2	$\bar{\alpha}_2$	β	2^{m+2} = 16 Processors
0 : 00	*cross*0	00	11	00,01,10,11	0, 1, 2, 3, 12,13,14,15
2 : 10	*cross*1	01	10	00,01,10,11	36,37,38,39,40,41,42,43

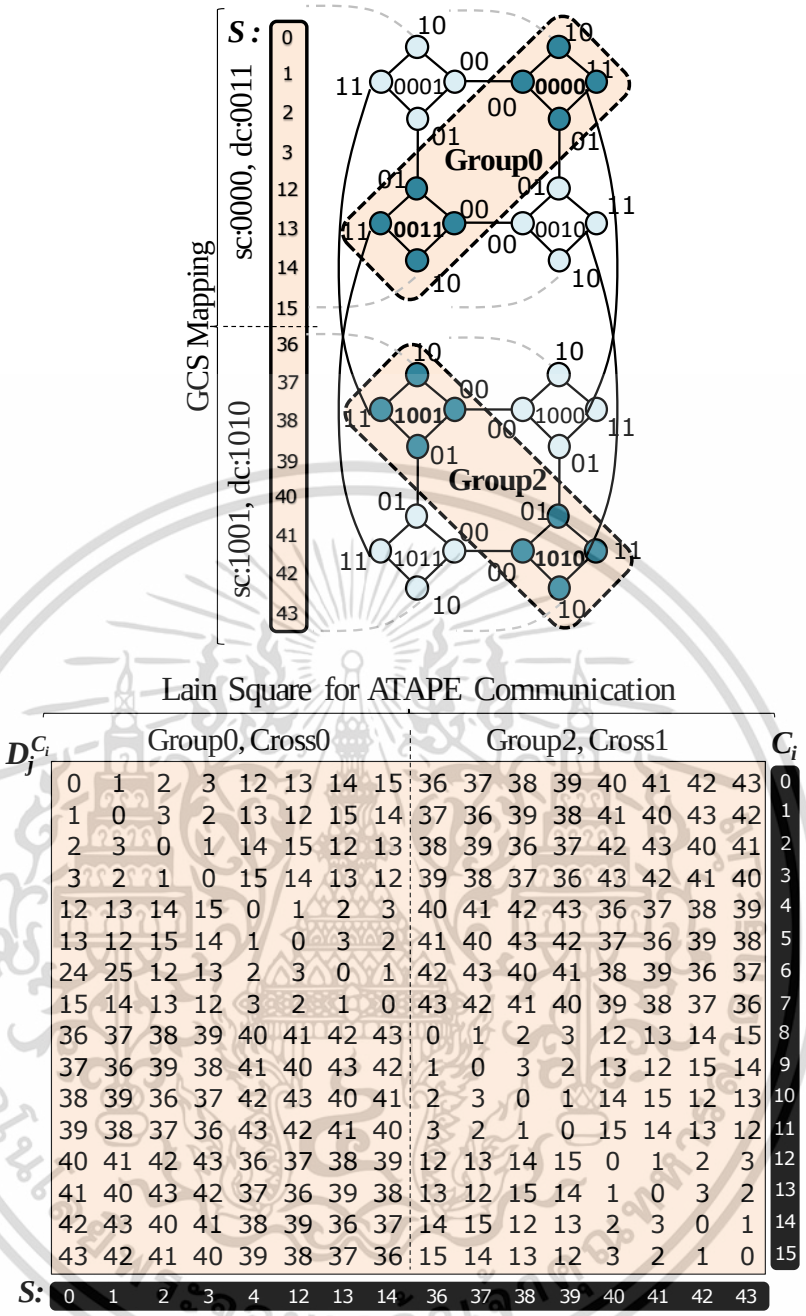


รูปที่ 3.19 ตัวอย่างของการแบ่งกลุ่มแบบ GCS บนเครือข่าย 6-HHC (N = 64, m = 2) เมื่อ e = 0000

เนื่องจากในวิทยานิพนธ์ฉบับนี้ได้นำการติดต่อสื่อสารเพื่อแลกเปลี่ยนข้อมูลแบบ ATAPE มาเพื่อทดสอบการสื่อสารของหน่วยประมวลผลที่ถูกแบ่งกลุ่ม ดังนั้นการสร้างตารางลาตินด้วยสมการที่ 3.3 เพื่อการ ATAPE เมื่อ $k = 2^{m+2}$ และ e = 0000 แสดงในรูปที่ 3.20 และ 3.21 ซึ่งประกอบด้วยการรวมกลุ่มแบบ GCS ด้วย 2 รูปแบบ คือ รูปแบบ a = 0 และ a = 1 โดยรูปที่ 3.20 แสดงตารางลาตินของการ ATAPE งานขนาด 16 หน่วยประมวลผล เมื่อ a = 0 (ค่าเริ่มต้น) และจะได้หน่วยประมวลผล (S) ทั้งหมด คือ fs = 0, 1, 2, 3, 12, 13, 14, 15 และ ss = 20, 21, 22, 23, 24, 25, 26, 27 ซึ่งจะมีตารางลาตินของโหนดปลายทาง ($D_j^{C_i}$) ของแต่ละตัวควบคุม (C_i) สำหรับตารางลาติน เมื่อ a = 1 จะได้หน่วยประมวลผล (S) ทั้งหมด คือ fs = 0, 1, 2, 3, 12, 13, 14, 15 และ ss = 36, 37, 38, 39, 40, 41, 42, 43 แสดงตารางลาตินในรูปที่ 3.21



รูปที่ 3.20 ตัวอย่างตารางลาตินขนาด 16×16 เพื่อการสื่อสารแบบ ATAPE ของการแบ่งกลุ่มแบบ GCS บนเครือข่าย 6-HHC (N = 64, m = 2) เมื่อ e = 0000 และ a = 0



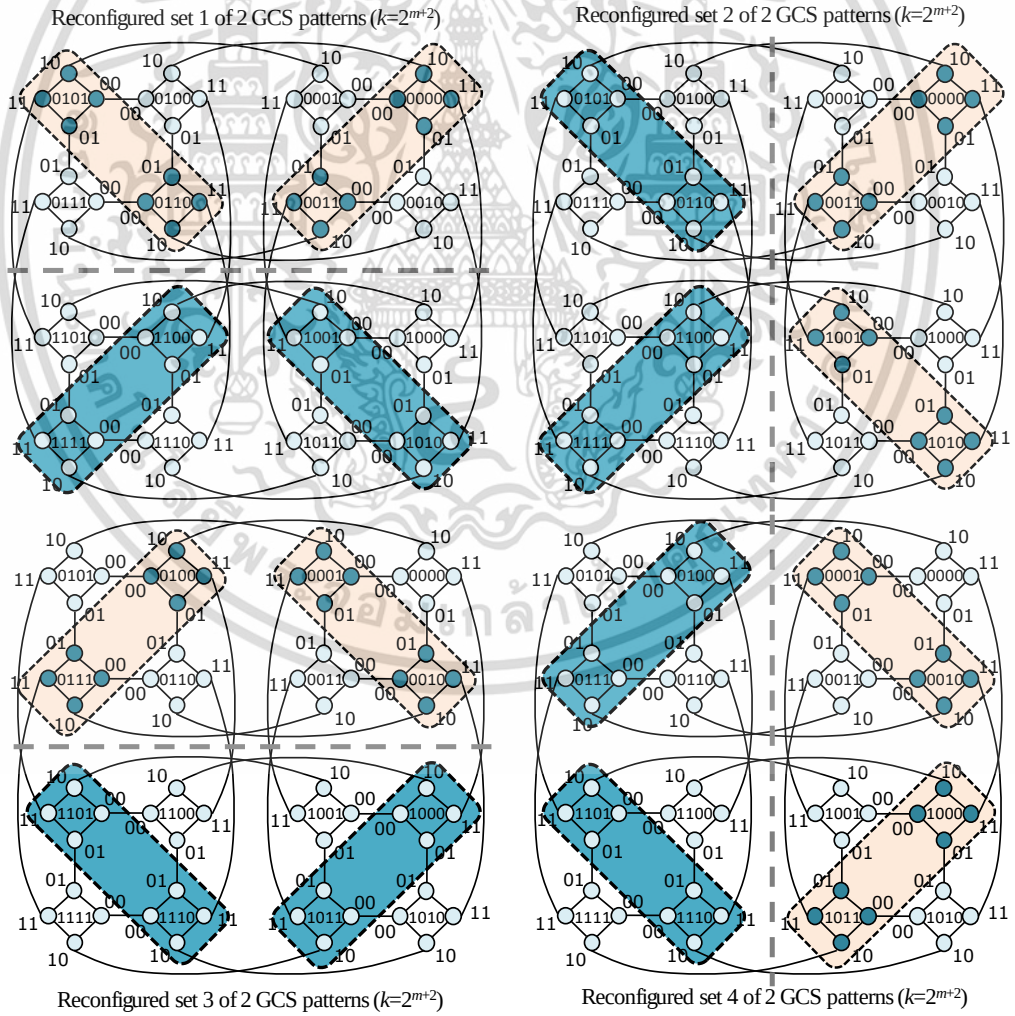
รูปที่ 3.21 ตัวอย่างตารางลาตินขนาด 16×16 เพื่อการสื่อสารแบบ ATAPE ของการแบ่งกลุ่มแบบ GCS บนเครือข่าย 6-HHC (N = 64, m = 2) เมื่อ e = 0000 และ a = 1

ตารางที่ 3.4 แสดงตัวอย่างอื่นๆ ของการรวมกลุ่มแบบ GCS ขนาด $k = 2^{m+2}$ เมื่อกำหนดให้ข้อมูลเข้า คือ โหนดชั้นนอก e = 0001 (หรือ $\alpha_1\alpha_2$ = 0001) จะได้ว่า Group α_1 = 00, Cross α_2 = 01 และ $p = \alpha_1 \oplus \alpha_2$ = 01 สำหรับกรณี a = 0 จะได้ว่าหน่วยประมวลผลของ fs คือ sc = $\alpha_1\alpha_2$,xx = 0001, (00,01,10,11) = 4, 5, 6, 7 และ dc = $\alpha_1\bar{\alpha}_2$,xx = 0010, (00,01,10,11) = 8, 9, 10, 11 ต่อมาทำการหาหน่วยประมวลผลทั้งหมดของ ss คือ sc = $\alpha_1\alpha_2$, xx = 0100, (00, 01, 10, 11) = 16, 17, 18, 19 และ dc = $\alpha_1\bar{\alpha}_2$,xx = 0111, (00,01,10,11) = 28, 29, 30, 31 ตามลำดับ

ตารางที่ 3.4 บิตไวส์-GCS สำหรับ $k = 2^{m+2}$ บนเครือข่าย 6-HHC ($m = 2$ และ $e = 0001$)

Default sub-system of input group 0 and its adjacent group 1					
Group α_1	Cross α_2	α_2	$\bar{\alpha}_2$	β	$2^{m+2} = 16$ Processors
0 : 00	*cross*1	01	10	00,01,10,11	4, 5, 6, 7, 8, 9,10,11
1 : 01	*cross*0	00	11	00,01,10,11	16,17,18,19, 28,29,30,31
Reconfigured sub-system of input group 0 and its adjacent group 2					
Group α_1	Cross α_2	α_2	$\bar{\alpha}_2$	β	$2^{m+2} = 16$ Processors
0 : 00	*cross*1	01	10	00,01,10,11	4, 5, 6, 7, 8, 9,10,11
2 : 10	*cross*0	00	11	00,01,10,11	32,33,34,35, 44,45,46,47

รูปที่ 3.22 แสดงการปรับเปลี่ยนรูปแบบของการแบ่งกลุ่มแบบ GCS ทั้งหมด 8 รูปแบบ ด้วยหน่วยประมวลผล $k = 2^{m+2}$ บนเครือข่าย 6-HHC เมื่อ $N = 64$ และ $m = 2$ โดยแต่ละรูปแบบมีข้อมูลเข้า หรือโหนดชั้นนอก e ที่แตกต่างกัน โดยรูปแบบทั้งหมดถูกเสนอมาเพื่อรองรับสำหรับกรณีที่รูปแบบตั้งต้น (Default) ไม่สามารถรองรับงานที่ต้องการได้ (Not Available)



รูปที่ 3.22 ตัวอย่างการปรับเปลี่ยนรูปแบบของการแบ่งกลุ่มแบบ GCS ทั้งหมด 8 รูปแบบ บนเครือข่าย 6-HHC ($N = 64$, $m = 2$) ที่มีหน่วยประมวลผลขนาด $k = 2^{m+2}$

3.3.3 การรวมกลุ่มแบบซูเปอร์-GCS สำหรับหน่วยประมวลผลขนาด $k > 2^{m+2}$

การรวมกลุ่มแบบซูเปอร์-GCS (Super-Grouping of Cross Sub-cube) ถูกเสนอขึ้นมาเพื่อรองรับงานที่ต้องการหน่วยประมวลเพื่อสื่อสารกันแบบ ATAPE ที่มีขนาดระหว่าง $2^{m+3} \le k \le 2^n/2^{M-1}$ เมื่อ $n = 2^m+m$ และ $M = 2^{m-1}$ ตัวอย่างเช่น ต้องการหน่วยประมวลผล บนเครือข่าย 6-HHC ($N = 64$ และ $m = 2$) สำหรับงานที่มีขนาด $k = 2^{m+3} = 32$ โดยซูเปอร์-GCS สามารถถูกสร้างจากการรวมกลุ่มของ GCD จำนวน 2^S กลุ่ม เมื่อ $S = 2, 3, ..., M$ ซึ่งแต่ละกลุ่ม GCD จะมีจำนวนของ Cross ที่จะถูกรวมกลุ่มกันแบบซูเปอร์-GCS ทั้งหมด 4 Cross และแต่ละ Cross จะประกอบด้วยโหนดชั้นนอก (sc และ dc) ขนาด 2^m จำนวน $2^{m+3}/2^m = 8$ โหนด โดยการรวมกลุ่มแบบซูเปอร์-GCS สามารถสร้างได้โดยการใช้สมการที่ 3.7 - 3.9 พร้อมด้วย สมการที่ 3.6 ที่อธิบายไปในหัวข้อที่ 3.3.2

กำหนดให้ $\boldsymbol{\alpha} = \alpha_1\alpha_2$ คือ เลขฐานสองของโหนดชั้นนอก (External Node) บนเครือข่าย n-HHC ที่มีขนาด 2^m บิต ซึ่งประกอบด้วย $\boldsymbol{\alpha}_1$ และ $\boldsymbol{\alpha}_2$ ที่มีขนาด 2^{m-1} บิตและ 2^{m-1} บิต ตามลำดับ

xx..x คือ เลขฐานสองที่เป็นไปได้ จาก 00...0 ถึง 11...1
เช่น xx = {00, 01, 10, 11}

$b_{n-1}... b_{n'+1}b_{n'}xx\boldsymbol{\alpha}_{2i}$ สำหรับ $k = 2^{m+3}$ (ทั้งหมด $2^2 = 4$ Group(i)) (3.7)

$b_{n-1}... b_{n'+1}xxx\boldsymbol{\alpha}_{2i}$ สำหรับ $k = 2^{m+4}$ (ทั้งหมด $2^3 = 8$ Group(i)) (3.8)

, ..., xxx...x$\boldsymbol{\alpha}_{2i}$ สำหรับ $k = 2^n/2^{M-1}$ (ทั้งหมด 2^M Group(i)) (3.9)

และ cross j (ของกลุ่มที่ i) $= i \oplus p$; ถ้า $j < 2^{M-1}$ เมื่อ $M = 2^{m-1}$

หรือ $= \overline{i \oplus p}$; อื่นๆ หรือ ถูกสร้างด้วยสมการที่ 3.6

รูปที่ 3.23 แสดงตัวอย่างการรวมกลุ่มแบบซูเปอร์-GCS สำหรับหน่วยประมวลผล 2^{m+3} บนเครือข่าย 6-HHC ($N = 64$ และ $m = 2$) ดังนั้นจะได้ว่า $k = 2^{m+3} = 32$ โหนด ถ้ากำหนดให้ข้อมูลเข้าคือ โหนดชั้นนอก $e = 0000$ (หรือ $\boldsymbol{\alpha}_1\boldsymbol{\alpha}_2 = 0000$) จะได้รูปแบบที่ $p = \boldsymbol{\alpha}_1 \oplus \boldsymbol{\alpha}_2 = 00 \oplus 00 = 0$ ซึ่งเป็นรูปแบบเริ่มต้น (Default) ดังรูปที่ 3.22 ก) จาก $\boldsymbol{\alpha} = \boldsymbol{\alpha}_1\boldsymbol{\alpha}_2 = 0000$ จะได้ว่า โหนดชั้นนอกของข้อมูลเข้า e อยู่ในกลุ่ม GCD ที่ $\boldsymbol{\alpha}_1 = 00$ และ Cross $\boldsymbol{\alpha}_2 = 00$ และจากสมการที่ 3.9 จะได้ว่าการรวมกลุ่มแบบซูเปอร์-GCS สำหรับ $k = 2^n/2^{M-1} = 32$ โหนด จะอยู่ในกลุ่มที่ $xx\boldsymbol{\alpha}_{2i} = \{00\boldsymbol{\alpha}_{2i}, 01\boldsymbol{\alpha}_{2i}, 10\boldsymbol{\alpha}_{2i}, 11\boldsymbol{\alpha}_{2i}\}$ เมื่อ $\boldsymbol{\alpha}_{2i}$ คือ Cross j ของแต่ละกลุ่มที่ i ที่สามารถหาได้ด้วยสมการที่ 3.6 นั้นคือ

กลุ่มที่ $i = 00$ จะได้ว่า Cross $j = i \oplus p = 00 \oplus 00 =$ **00**
กลุ่มที่ $i = 01$ จะได้ว่า Cross $j = i \oplus p = 01 \oplus 00 =$ **01**
กลุ่มที่ $i = 10$ จะได้ว่า Cross $j = \overline{i \oplus p} = \overline{10 \oplus 00} =$ **01**
กลุ่มที่ $i = 11$ จะได้ว่า Cross $j = \overline{i \oplus p} = \overline{11 \oplus 00} =$ **00**

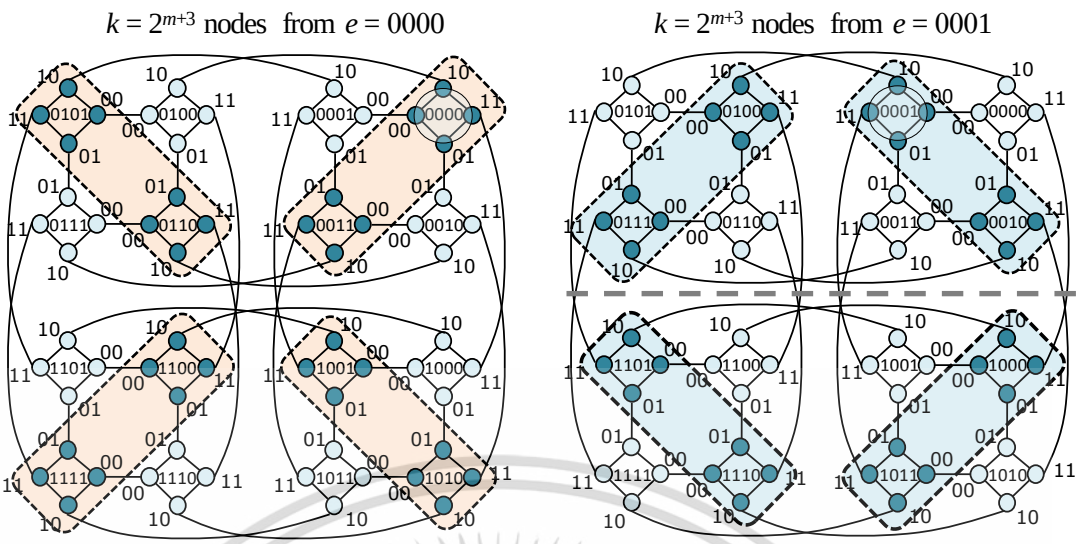
จะได้ว่าผลลัพธ์โหนดชั้นนอกสี่โหนดแรก (sc) ที่สามารถรวมกลุ่มแบบซูเปอร์-GCS ประกอบด้วย 00**00**, 01**01**, 10**01**, 11**00** และด้วยสมการที่ 3.4 จะทำให้ได้คู่ดูอัล-คิวบ์ (dc) ของแต่ละโหนด คือ 0011, 0110, 1010, 1111 ดังนั้นผลลัพธ์โหนดชั้นนอกของการรวมกลุ่มแบบซูเปอร์-GCS คือ {(0000, 0011), (0101, 0110), (1001, 1010) (1100, 1111)} ซึ่งทั้งหมดจะประกอบด้วยหน่วยประมวลผล S = {(0000,00), (0000,01), (0000,10), (0000,11), (0011,00), (0011,01), (0011,10), (0011,11), (0101,00), (0101,01), (0101,10), (0101,11), (0110,00), (0110,01), (0110,10), (0110,11), (1001,00), (1001,01), (1001,10), (1001,11), (1010,00), (1010,01), (1010,10), (1010,11), (1100,00), (1100,01), (1100,10), (1100,11), (1111,00), (1111,01), (1111,10), (1111,11)} = {0 - 3, 12 - 15, 20 - 23, 24 - 27, 36 - 39, 40 - 43, 48 - 51, 60 – 63} รูปของเลขฐาน 2 ตามลำดับ

รูปที่ 3.23 ข) แสดงตัวอย่างเมื่อข้อมูลเข้า คือ e = 0001 หรืออยู่ในรูปแบบที่ $p = \alpha_1 \oplus \alpha_2$ = 00⊕01 = 1 ซึ่งจะสามารถใช้รูปแบบนี้ถ้ารูปแบบบเริ่มต้น (p = 0) ไม่สามารถรองรับหน่วยประมวลผลขนาดที่ต้องการได้ โดยในกรณีนี้โหนดชั้นนอกของข้อมูลเข้า e อยู่ในกลุ่ม GCD ที่ α_1 = 00 และ Cross α_2 = 01 และจากสมการที่ 3.9 จะได้ว่าการรวมกลุ่มแบบซูเปอร์-GCS สำหรับ $k = 2^n/2^{M-1}$ = 32 โหนด จะอยู่ในกลุ่มที่ $xx\alpha_{2i}$ = {00α_{2i}, 01α_{2i}, 10α_{2i}, 11α_{2i} } และ Cross j ของแต่ละกลุ่มที่ i จะได้ดังนี้

กลุ่มที่ i = 00 จะได้ว่า Cross $j = i \oplus p$ = 00⊕01 = 01
กลุ่มที่ i = 01 จะได้ว่า Cross $j = i \oplus p$ = 01⊕01 = 00
กลุ่มที่ i = 10 จะได้ว่า Cross $j = \overline{i \oplus p}$ = $\overline{10 \oplus 01}$ = 00
กลุ่มที่ i = 11 จะได้ว่า Cross $j = \overline{i \oplus p}$ = $\overline{11 \oplus 01}$ = 01

จะได้ว่าผลลัพธ์โหนดชั้นนอกสี่โหนดแรก (sc) ที่สามารถรวมกลุ่มแบบซูเปอร์-GCS ประกอบด้วย 0001, 0100, 1000, 1101 และด้วยสมการที่ 3.4 จะทำให้ได้คู่ดูอัล-คิวบ์ (dc) ของแต่ละโหนด คือ 0010, 0111, 1011, 1110 ดังนั้นผลลัพธ์โหนดชั้นนอกของการรวมกลุ่มแบบซูเปอร์-GCS คือ {(0001, 0010), (0100, 0111), (1000, 1011) (1101, 1110)} ซึ่งทั้งหมดจะประกอบด้วยหน่วยประมวลผล S = {4 - 7, 8 - 11, 16 - 19, 28 - 31, 32 - 35, 44 - 47, 52 - 55, 56 – 59} ในรูปของเลขฐาน 2 ตามลำดับ

ลำดับต่อไป จะแสดงให้เห็นการรวมกลุ่มแบบ GCD และ GCS ของรูปแบบทั้งหมดที่ได้อธิบายในหัวข้อ 3.3.1 – 3.3.3 บนเครือข่าย 11-HHC (N = 2048 และ m = 3) เพื่อให้เห็นนวัตกรรมและผลงาน (Innovation and Contribution) ที่วิทยานิพนธ์ฉบับนี้นำเสนอชัดเจนยิ่งขึ้น โดยรูปที่ 3.27 - 3.28 แสดงการปรับเปลี่ยนรูปแบบของการรวมกลุ่มของหน่วยประมวลผลที่มีขนาดแตกต่างกัน เช่น $k = 2^{m+1}, 2^{m+2}, ..., 2^{m+5}$ จากหน่วยประมวลผลทั้งหมด N = 2048 หน่วยประมวลผล



ก) ข)

รูปที่ 3.23 ตัวอย่างการปรับเปลี่ยนรูปแบบของการแบ่งกลุ่มแบบซูเปอร์-GCS ทั้งหมด 2 รูปแบบ บนเครือข่าย 6-HHC (N = 64, m = 2) ที่มีหน่วยประมวลผลขนาด $k = 2^{m+3}$

รูปที่ 3.24 แสดงการแบ่งกลุ่มแบบ GCD ด้วย Cross ที่มีขนาด $k = 2^{m+1}$ = 16 หน่วยประมวลผล โดยมีโหนดข้อมูลเข้า $e = \alpha_1\alpha_2$ = 00000000 ดังนั้น sc = $\alpha_1\alpha_2$,xxx = (00000000, xxx) และดูอัล-คิวบ์ dc = $\alpha_1\overline{\alpha_2}$, xxx = (00001111, xxx) และมีหน่วยประมวลผลทั้งหมด 16 หน่วยประมวลผลดังนี้ (0, 1, 2, 3, 4, 5, 6, 7, 120, 121, 122, 123, 124, 125, 126, 127) และดูตัวอย่างจากตารางที่ 3.5 โดยตาราง 3.5 จะแสดงการรวมกลุ่มแบบ GCD ของโหนดข้อมูลเข้า e = 00000000, 00000001, ..., 11111111

รูปที่ 3.25 และตารางที่ 3.6 แสดงการรวมกลุ่มแบบ GCS ของหน่วยประมวลผล $k = 2^{m+2}$ = 32 บนเครือข่าย 11-HHC (N = 2048 และ m = 3) ที่ประกอบด้วยสอง Cross ที่อยู่ติดกัน ดังนั้นแต่ละ Cross จะมีหน่วยประมวลผล 16 หน่วยประมวลผล ถ้า e = 0 (α_1= 0000 ,α_2 = 0000, $p = \alpha_1 \oplus \alpha_2$ = 0000⊕0000 = 0 จะได้จำนวนหน่วยประมวลผลทั้งหมด 32 ด้วยประมวลผลดังนี้ (0 - 7, 120 - 127, 136 - 143, 240 – 247)

รูปที่ 3.26 แสดงตัวอย่างการรวมกลุ่มแบบซูเปอร์-GCS สำหรับหน่วยประมวลผลขนาด $k = 2^{m+3}$ = 64 บนเครือข่าย 11-HHC ที่ประกอบด้วยสี่ Cross i ของกลุ่มที่ i เมื่อ i = 0, 1, 2, 3 ส่วนรูปที่ 3.27 ก) แสดงการรวมกลุ่ม 8 กลุ่ม เมื่อ $k = 2^{m+4}$ = 128 (ตารางที่ 3.7) และ ข) $k = 2^{m+5}$ = 256หน่วยประมวลผล

ตารางที่ 3.5 การแบ่งกลุ่มแบบ GCD สำหรับ $k = 2^{m+1}$ บนเครือข่าย 11-HHC ($N = 2048$, $m = 3$)

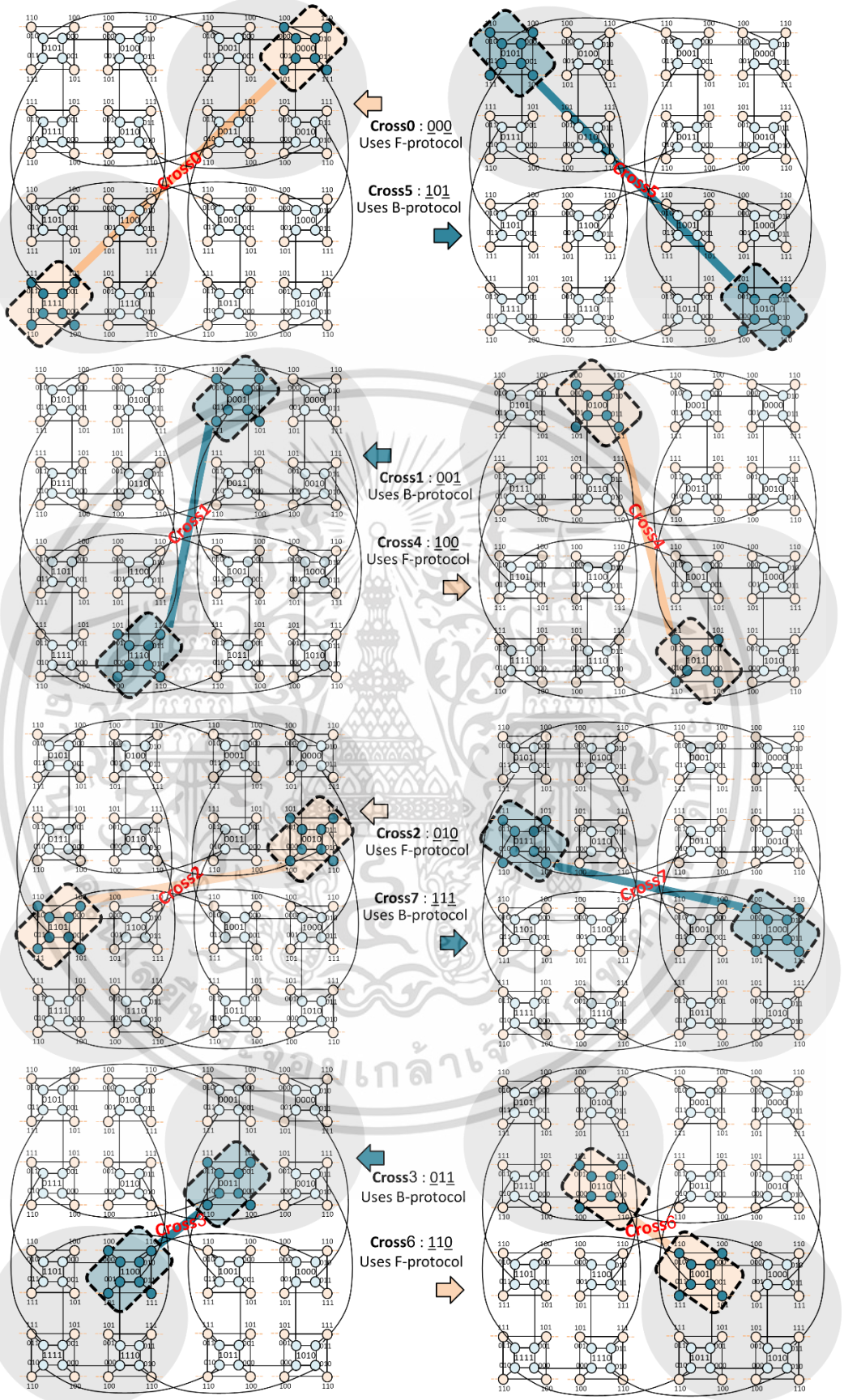
Group α_1	Cross α_2	α_2	$\bar{\alpha}_2$	β	$2^{m+1} = 16$ Ps
0 : 0000	*cross*0	0000	1111	000 - 111	0 - 7, 120 - 127
	*cross*1	0001	1110	000 - 111	8 - 15, 112 - 119
	*cross*2	0010	1101	000 - 111	16 - 23, 104 - 111
	*cross*3	0011	1100	000 - 111	24 - 31, 96 - 103
	*cross*4	0100	1011	000 - 111	32 - 39, 88 - 95
	*cross*5	0101	1010	000 - 111	40 - 47, 80 - 87
	*cross*6	0110	1001	000 - 111	48 - 55, 72 - 79
	*cross*7	0111	1000	000 - 111	56 - 63, 64 - 71
1 : 0001	*cross*0	0000	1111	000 - 111	128 - 135, 248 - 255
	*cross*1	0001	1110	000 - 111	136 - 143, 240 - 247
	*cross*2	0010	1101	000 - 111	144 - 151, 232 - 239
	⋮	⋮	⋮	⋮	⋮
	*cross*7	0111	1000	000 - 111	184 - 191, 192 - 199
⋮	⋮	⋮	⋮	⋮	⋮
15 : 1111	*cross*0	0000	1111	000 - 111	1920-1927, 2040-2047
	*cross*1	0001	1110	000 - 111	1928-1935, 2032-2039
	⋮	⋮	⋮	⋮	⋮
	*cross*7	0111	1000	000 - 111	1976-1983, 1984-1991

ตารางที่ 3.6 การรวมกลุ่มแบบ GCS สำหรับ $k = 2^{m+2}$ บนเครือข่าย 11-HHC

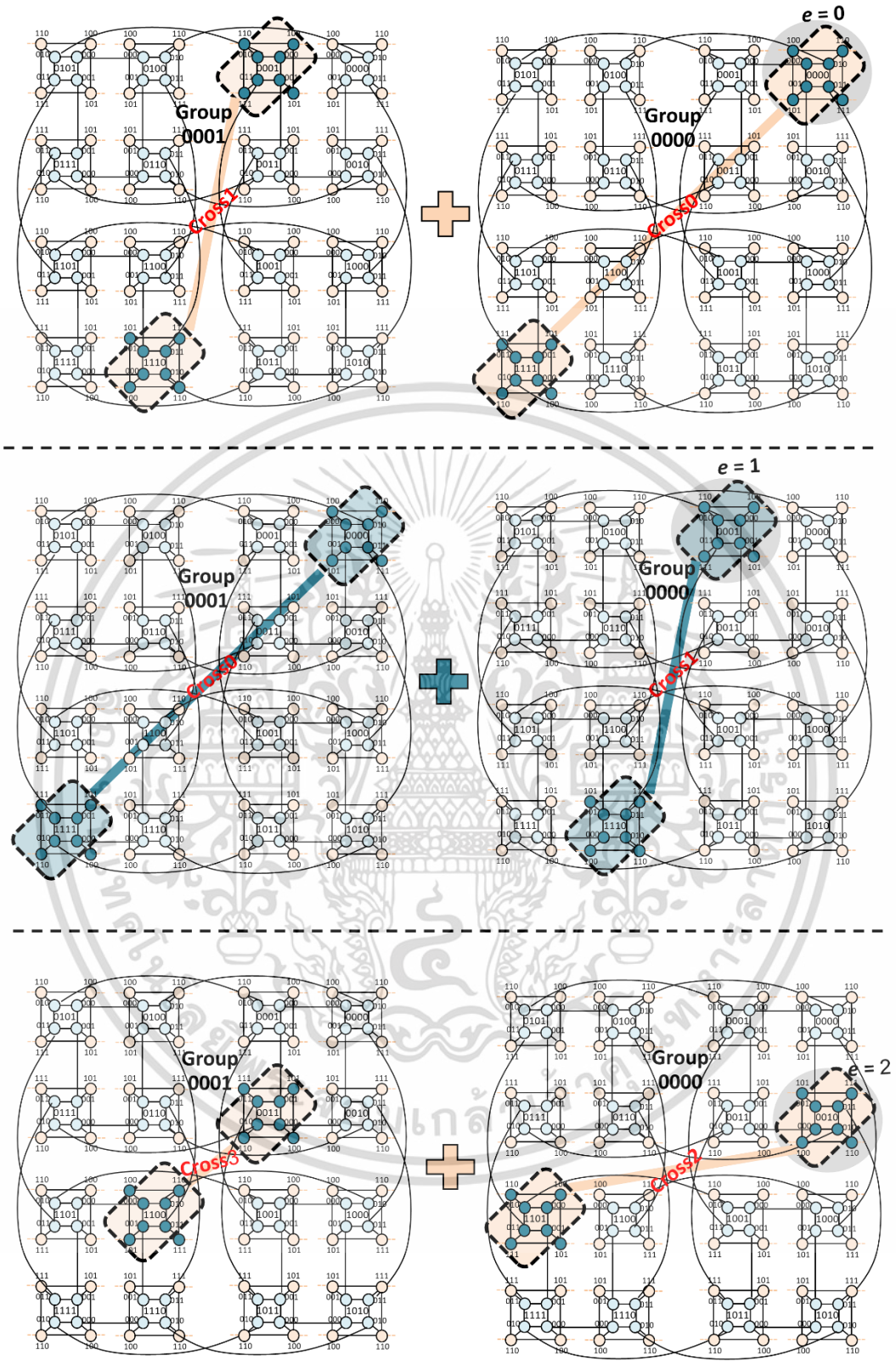
Group α_1	Cross α_2	α_2	$\bar{\alpha}_2$	β	$2^{m+2} = 32$ Ps (16x2)
***e* = 0**					
0 : 000**0**	*cross*0	0000	1111	000 - 111	0 - 7, 120 - 127
1 : 000**1**	*cross*1	0001	1110	000 - 111	136 - 143, 240 - 247
***e* = 1**					
0 : 000**0**	*cross*1	0001	1110	000 - 111	8 - 15, 112 - 119
1 : 000**1**	*cross*0	0000	1111	000 - 111	128 - 135, 248 - 255
***e* = 2**					
0 : 000**0**	*cross*2	0010	1101	000 - 111	16 - 23, 104 - 111
1 : 000**1**	*cross*3	0011	1100	000 - 111	152 - 259, 224 - 231
⋮	⋮	⋮	⋮	⋮	⋮
***e* = 7**					
0 : 000**0**	*cross*7	0111	1000	000 - 111	56 - 63, 64 - 71
1 : 000**1**	*cross*6	0110	1001	000 - 111	176 - 183,200 - 207

ตารางที่ 3.7 การรวมกลุ่มแบบซูเปอร์-GCS สำหรับ $k = 2^{m+4}$ บนเครือข่าย 11-HHC

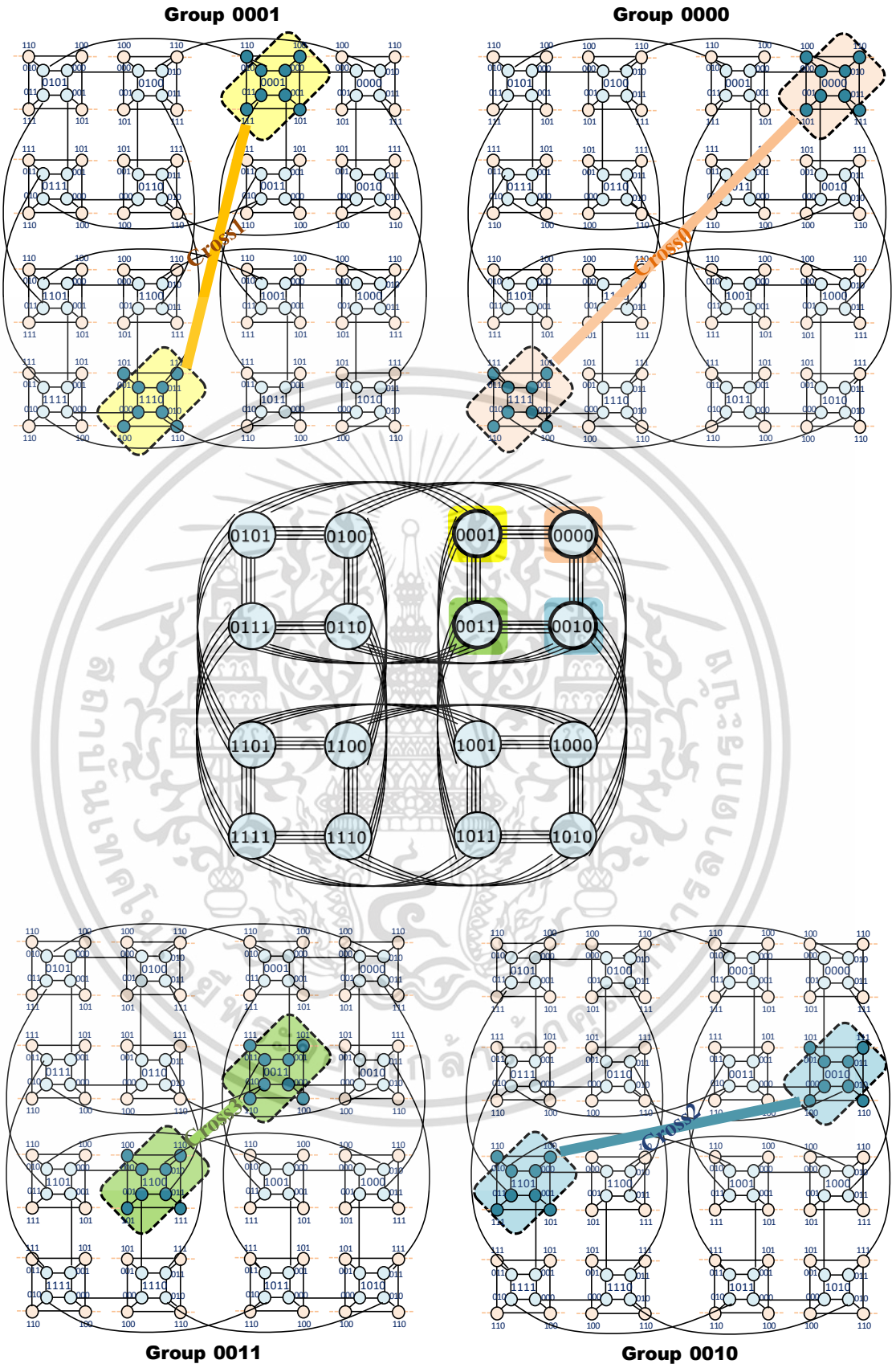
Group α_1	Cross α_2	α_2	$\bar{\alpha}_2$	β	$2^{m+4} = 128$ Ps (16x8)
***e* = 1**					
0 : 0000	*cross*0	0000	1111	000 - 111	0 - 7, 120 - 127
1 : 0001	*cross*1	0001	1110	000 - 111	136 - 143, 240 - 247
2 : 0010	*cross*2	0010	1101	000 - 111	272 - 279, 360 - 367
3 : 0011	*cross*3	0011	1100	000 - 111	408 - 415, 480 - 487
⋮	⋮	⋮	⋮	⋮	⋮
7 : 0111	*cross*7	0111	1000	000 - 111	952 - 959, 960 - 967
⋮	⋮	⋮	⋮	⋮	⋮
***e* = 7**					
0 : 0000	*cross*7	0111	1000	000 - 111	56 - 63, 64 - 71
⋮	⋮	⋮	⋮	⋮	⋮
7 : 0111	*cross*0	0000	1111	000 - 111	896 - 903, 1016 - 1023



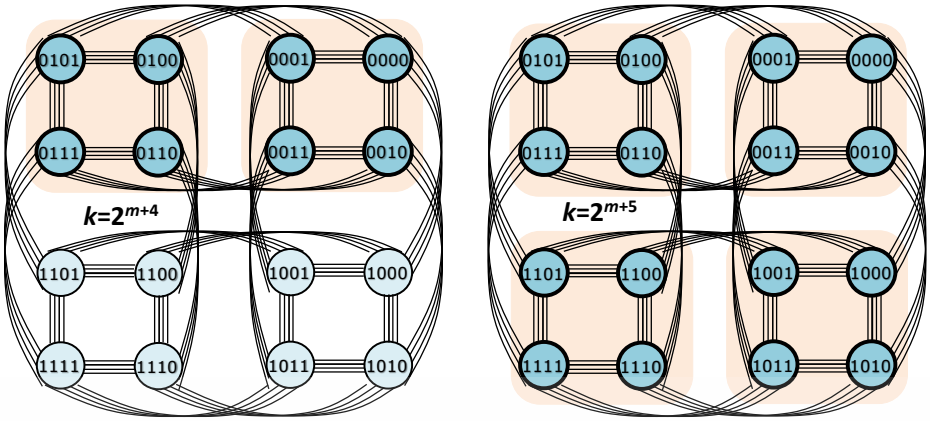
รูปที่ 3.24 ตัวอย่างการรวมกลุ่มบน 11-HHC (N = 2048 และ m = 3) เมื่อ $k = 2^{m+1} = 16$



รูปที่ 3.25 ตัวอย่างการรวมกลุ่มบน 11-HHC (N = 2048 และ m = 3) เมื่อ $k = 2^{m+2} = 32$



รูปที่ 3.26 ตัวอย่างการรวมกลุ่มบน 11-HHC (N = 2048 และ m = 3) $k = 2^{m+3}$



ก) ข)

รูปที่ 3.27 ตัวอย่างการรวมกลุ่มบน 11-HHC (N = 2048 และ m = 3) ก) $k = 2^{m+4}$; ข) $k = 2^{m+5}$

3.4 การจัดตารางงานที่ดีที่สุดบนเครือข่าย *n*-HHC

ในหัวข้อนี้นำเสนอการจัดตารางงาน (Task-scheduling) ที่ดีที่สุด ที่สามารถประยุกต์ใช้หลักการของ Cross เลขคู่และเลขคี่ (Even/Odd Protocol) ได้จริงบนเครือข่าย *n*-HHC ซึ่งถูกเสนอในหัวข้อ 3.4.1 และในหัวข้อ 3.4.2 นำเสนอต้นไม้แบบทวิภาคที่ปรับเปลี่ยนได้ (Reconfigurable Binary Tree) เพื่อรองรับการจับกลุ่มแบบ GCD และ GCS และหัวข้อสุดท้ายนำเสนอวิธีการจัดสรรและยกเลิกจัดสรรหน่วยประมวลผลอย่างมีประสิทธิภาพ นำเสนอในหัวข้อที่ 3.4.3

3.4.1 หลักการ Cross เลขคู่และเลขคี่ สำหรับการจัดสรรงานที่ดีที่สุด

จากการนำเสนอการรวมกลุ่มแบบ GCD และ GCS บนเครือข่าย *n*-HHC จะได้ว่าทุกๆ 2^M กลุ่มแบบ GCD สามารถสื่อสารเพื่อรับส่งข้อมูลของงานที่มีขนาด $k = 2^{m+1}$ ได้พร้อมกัน โดยใช้หนึ่งงานต่อหนึ่งกลุ่ม ในทำนองเดียวกัน ทุกๆ 2^{M-1} กลุ่มแบบซูเปอร์-GCS สามารถสื่อสารเพื่อรับส่งข้อมูลของงานหลายๆงานที่มีขนาด $k = 2^{m+2}$ ได้พร้อมกัน และต่อเนื่องกันไปแบบนี้เรื่อย ๆ จนถึงการแบ่งกลุ่มของงานที่มีขนาด $2^n/2^{M-1}$ ซึ่งสามารถดูได้จากตารางที่ 3.8

และจากการนำเสนอการแบ่ง Cross ที่ประกอบด้วย Cross เลขคู่และเลขคี่ เพื่อการหาเส้นทางด้วยการเรียงลำดับบิตไปข้างหน้าและถอยกลับแบบวนรอบ (Forward/Backward Reordering and Routing) ด้วยขั้นตอนวิธีที่ 3.3 ทำให้แต่ละ Cross สามารถสื่อสารเพื่อรับส่งข้อมูลด้วยตัวควบคุม (Control) เดียวกันได้พร้อมๆกัน (Synchronized) โดยไม่มีการขัดแย้งกันระหว่างสื่อสาร ดังนั้นในการรวมกลุ่มแบบซูเปอร์-GCS ด้วยการใช้รูปแบบการแบ่งแบบเลขคู่และเลขคี่ที่มีขนาด $k = 2^{m+1}, 2^{m+2}, ..., 2^n/2^{M-1}$ สามารถสื่อสารเพื่อรับส่งข้อมูลด้วยตัวควบคุมเดียวกันได้พร้อมๆกันโดยไม่มีการขัดแย้งกันระหว่างสื่อสาร และในทางกลับกัน พื้นที่ที่เหลือในแต่ละกลุ่มสามารถใช้เพื่อรองรับงานขนาด $k \le 2^m$ ได้อย่างอิสระ

ตารางที่ 3.8 จำนวนการจัดกลุ่มของหน่วยประมวลผลแบบต่างๆ เมื่อ m = 2, 3 และ 4

Under (F, B) protocol	*k* PEs per task		จำนวน Tasks (max)
	หัวข้อ 3.2	หัวข้อ 3.3	
N = 64 (*m*=2) 2 crosses per group	2^{m+1} = 8	2^{m+1} = 8	2 x 4 = 8
	N/A	2^{m+2} = 16	2 x 2 = 4
	N/A	2^{m+3} = 32	2 x 1 = 2
N = 2,048 (*m*=3) 8 crosses per group	2^{m+1} = 16	2^{m+1} = 16	8x16 = 128
	N/A	2^{m+2} = 32	8 x 8 = 64
	N/A	2^{m+3} = 64	8 x 4 = 32
	N/A	2^{m+4} = 128	8 x 2 = 16
	N/A	2^{m+5} = 256	8 x 1 = 8
N = 1,048,576 (*m*=4) 128 crosses per group	2^{m+1} = 32	2^{m+1} = 32	128 x 256 = 32,768
	N/A	2^{m+2} = 64	128 x 128 = 16,384
	N/A	2^{m+3} = 128	128 x 64 = 8,192
	N/A	2^{m+4} = 256	128 x 32 = 4,096
	N/A	⋮	⋮
	N/A	2^{m+9} = 8,192	128 x 1 = 128

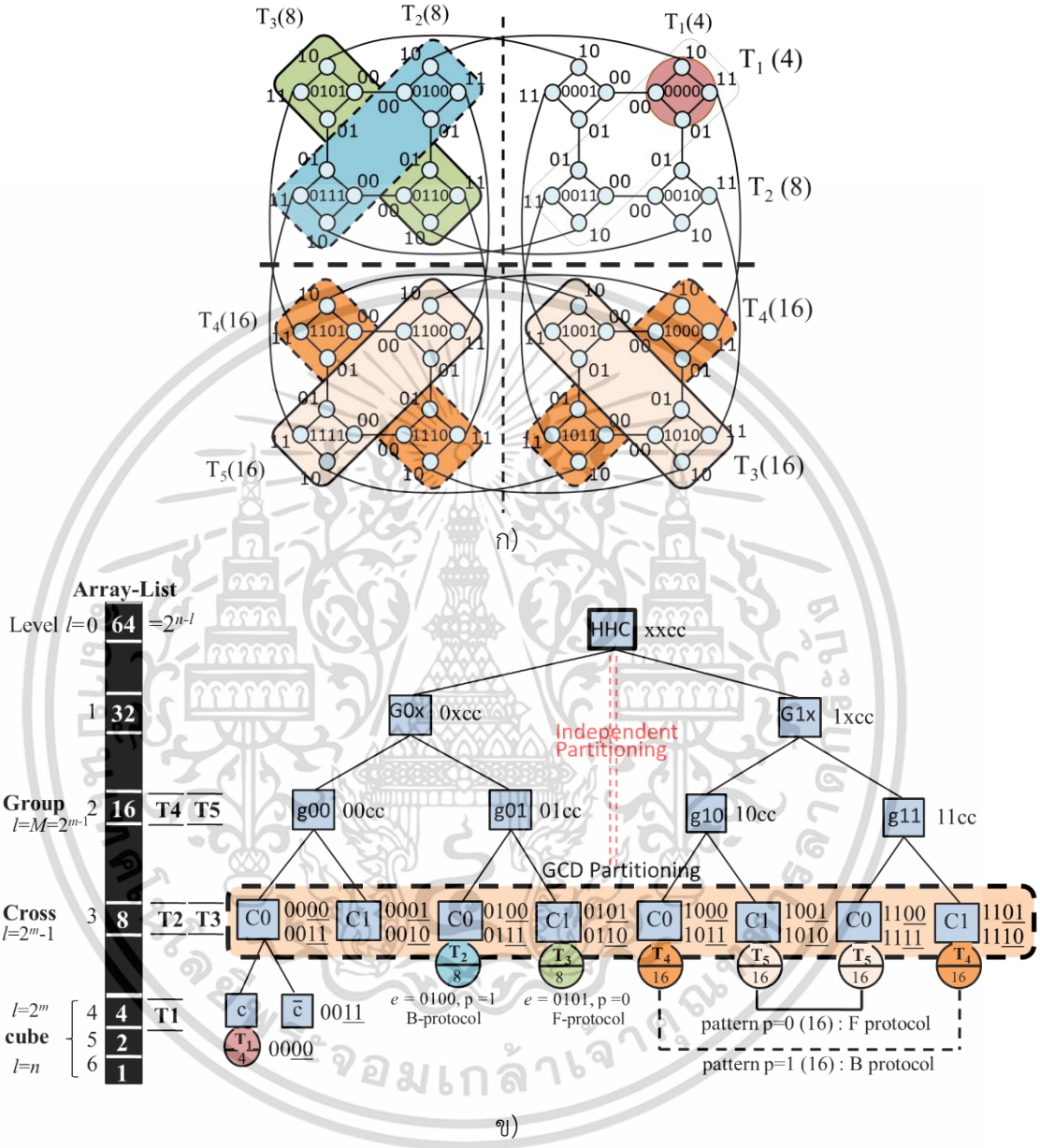
3.4.2 ต้นไม้แบบทวิภาคที่ปรับเปลี่ยนได้สำหรับการจัดสรรงานบนเครือข่าย HHC

หัวข้อนี้นำเสนอต้นไม้แบบทวิภาคที่ปรับเปลี่ยนได้ (Reconfigurable Binary Tree) สำหรับการจัดสรรงานและยกเลิกจัดสรรงานที่เหมาะสม บนเครือข่าย n-HHC ภายในเวลา O(N)

กำหนดให้ แต่ละโหนดบนต้นไม้แบบทวิภาค ถูกแทนด้วย 2^m บิต ของสี่สัญลักษณ์ดังนี้ ∑ = {0, 1, x, c} โดย x...x หมายถึง ทุกค่าบิตของเลขฐานสองที่เป็นไปได้ของกลุ่ม GCD เช่น xx = (00, 01, 10, 11) และ c...c หมายถึง ทุกค่าบิตของเลขฐานสองที่เป็นไปได้ของ Cross เช่น cc = 00 และ $\overline{cc}$ = 11 ในขั้นต้นโหนดรากของต้นไม้ (Root Node) ซึ่งจะอยู่ชั้น (level : l) ที่ 0 (l = 0) จะถูกแทนด้วย xx...xcc...c ที่ประกอบด้วยโหนดชั้นนอกจำนวน 2^{2^m} โหนด ต่อมาที่ชั้นที่ 1 (l = 1) จะเป็นชั้นของลูก (Child) ของโหนดในชั้นที่ 0 ซึ่งจะประกอบด้วยสองโหนด คือ ลูกทางซ้าย (Left-child : L) และลูกทางขวา (Right-child : R) และถูกแทนด้วย 0x...xc...c และ 1x...xc...c ตามลำดับ และสามารถดูรูปที่ 3.28 ประกอบ (N = 64 และ m = 2) ทำซ้ำโครงสร้างของต้นไม้แบบทวิภาคในรูปแบบนี้ไปเรื่อย ๆ จนถึงชั้นของกลุ่ม หรือ *group*-level (l = M) ซึ่งในชั้นนี้แต่ละกลุ่มจะมีอิสระต่อกัน และภายในแต่ละกลุ่มจะประกอบด้วย 2^{M-1} Cross (ที่มี 2^{m+1} หน่วยประมวลผล) ที่ไม่อิสระต่อกัน ดังนั้นจะมีเพียงหนึ่ง Cross ที่สามารถสื่อสารแบบ ATAPE ได้อย่างอิสระ ยกเว้นในกรณีที่กำหนดให้ Cross เลขคู่ใช้จัดลำดับบิตไปข้างหน้า และ Cross เลขคี่ใช้จัดลำดับบิตถอยกลับแบบวนรอบ จะทำให้สามารถให้ทุก Cross ในกลุ่ม สามารถสื่อสารแบบ ATAPE ได้พร้อมกันทั้งหมดภายใต้ตัวควบคุมเดียวกัน ต่อมาในชั้นของ *cross*-level (l = 2^m-1) จะประกอบด้วยสองดูอัล-คิวบ์ คือ c และ $\overline{c}$ ซึ่งในชั้นนี้จะเป็นโครงสร้างที่แตกต่างจากโครงสร้างต้นไม้แบบปรกติ

รูปที่ 3.28 แสดงตัวอย่างโครงสร้างต้นไม้แบบทวิภาค และการจัดสรรงาน บนเครือข่าย 6-HHC (N = 64, m = 2) ดังนั้นโหนดรากของต้นไม้จะถูกแทนด้วย xxcc ต่อมา ในชั้น *group*-level ที่มี l = M = 2 ซึ่งประกอบด้วยกลุ่มทั้งหมด 2^M = 4 กลุ่ม คือ 00cc, 01cc, 10cc และ 11cc ชั้นต่อมา คือ *cross*-level คือ Cross ที่อยู่ในกลุ่ม ดังนั้นชั้นนี้จะมีโหนดพ่อแม่ (Parents) คือ โหนดที่อยู่

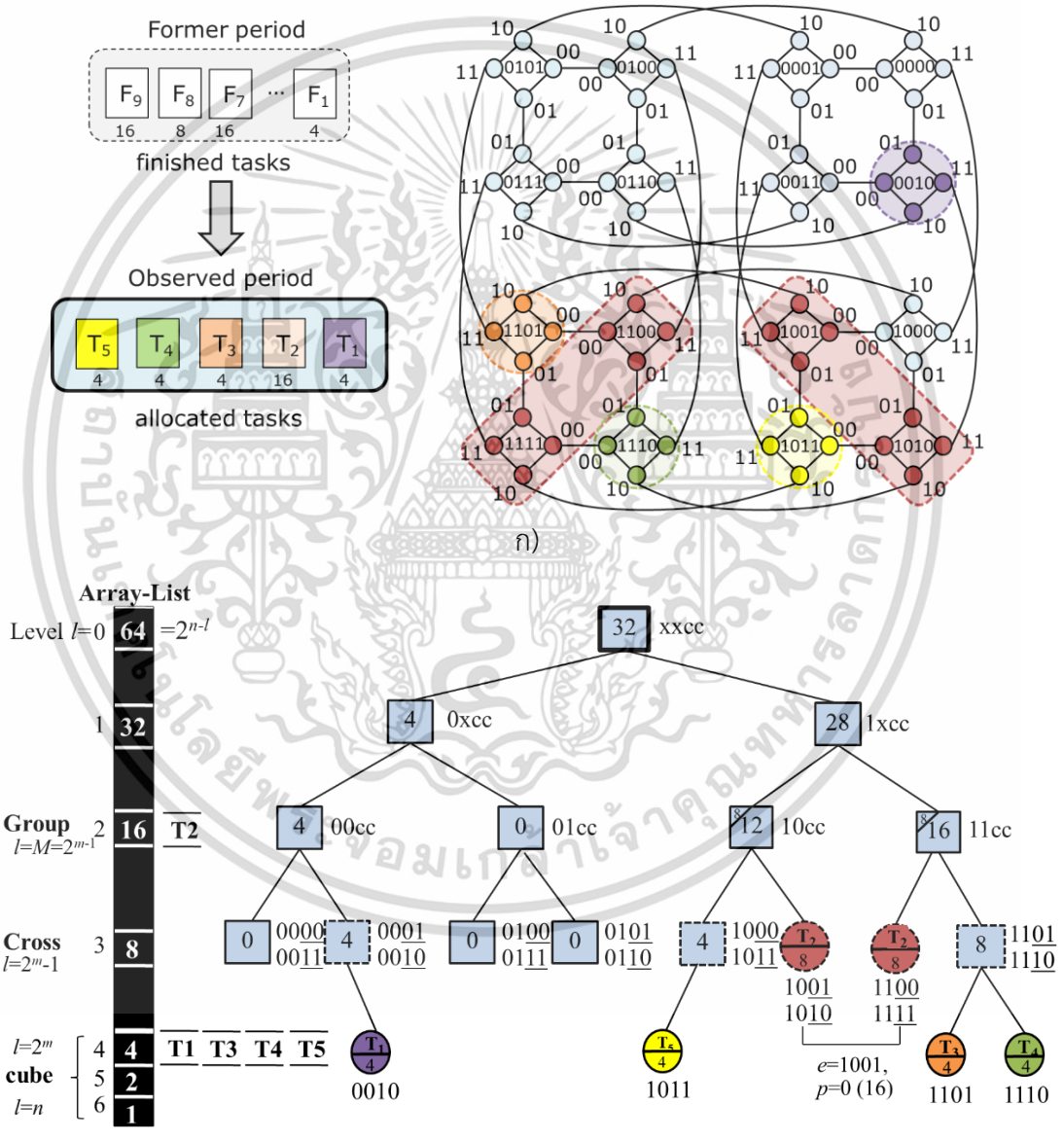
ใน *group*-level เช่น โหนดพ่อแม่คือ 00cc จะมี 2 โหนดลูกอยู่ในชั้น Cross คือ C0 และ C1 ซึ่ง C0 หมายถึง $Cross_0$ = (0000, 0011) และ C1 หมายถึง $Cross_1$ = (0001, 0010)



รูปที่ 3.28 ก) ตัวอย่างงานที่ถูกจัดสรรบน 6-HHC ข) โครงสร้างต้นไม้แบบทวิภาคที่ถูกจัดสรรงาน 4 งาน ใน *cross*-level และ *cube*-level

สำหรับงานขนาดเล็กที่มีขนาด $k = 2^m$ จะถูกจัดสรรให้อยู่บน m-ไฮเปอร์คิวบ์ หรือ *cube*-level ($l = 2^m, 2^{m+1}, ..., n$) ดังนั้นการแบ่งกลุ่มของ HC แบบปรกติสามารถประยุกต์ในชั้นนี้ได้โดยตรง หมายเหตุ เพื่อให้ง่ายต่อการอธิบาย สำหรับแต่ละโหนดในชั้น *cube*-level จะมีการเพิ่มบิต m-บิตของโหนดชั้นใน ต่อท้าย 2^m บิตของโหนดชั้นนอก

รูปที่ 3.28 แสดงการจัดสรรงานบนเครือข่าย 6-HHC (N = 64, m = 2) ซึ่งกรณีนี้ สมมติว่าระบบทั้งหมดยังไม่มีหน่วยประมวลผลใดๆ ถูกใช้งานอยู่ในระบบ และมีงานที่เข้าสู่ระบบพร้อมกันจำนวน 5 งาน คือ T_1 = 4, T_2 = 8, T_3 = 8, T_4 = 16 และ T_5 = 16 โดยทั้ง 5 งานจะถูกจัดสรรตามลำดับ เช่น ที่ level = 2 ของโครงสร้างต้นไม้ จะมี T_4 ถูกจัดสรร ด้วย cross-node C0 = (10**00**, 10**11**) อยู่ใน Group2 และ cross-node C1 = (11**01**, 11**10**) อยู่ใน Group3 ที่ level 4 จะมี T_1 ถูกจัดสรร ด้วยโหนด 0000 รูปที่ 3.29 แสดงตัวอย่างเพิ่มเติมสำหรับการจัดสรรของงานที่เข้าสู่ระบบ บนเครือข่าย 6-HHC โดยสมมุติว่า ปัจจุบันระบบมี 5 งาน คือ T_1 – T_5 ที่มีขนาด 4, 16, 4, 4 และ 4 ตามลำดับ



รูปที่ 3.29 ตัวอย่างการจัดสรรงาน บนเครือข่าย 6-HHC, m=2 ก) การจัดสรรงาน 5 งาน ข) การจัดสรรงานบนโครงสร้างต้นไม้แบบทวิภาค

สังเกตว่าในรูปที่ 3.29 ข) ในแต่ละโหนดบนโครงสร้างต้นไม้แบบทวิภาคจะมีตัวเลขฐาน 10 ซึ่งจะถูกเรียกว่า ค่ารวมสะสม (Accumulated Sum : *sum*) โดยค่า *sum* คือ การแสดงผลรวมของจำหน่วยประมวลผลของโหนดลูกที่ถูกใช้งานอยู่ทั้งหมด และถูกแสดงไว้ในโหนดแม่ เช่น โหนด 10cc ใน $l = 2$ จะมีค่า *sum* = 8 เนื่องจากโหนดลูกทางซ้ายและขวา มีหน่วยประมวลผลข้างละ 4 หน่วยประมวลผลกำลังถูกใช้งานอยู่ (*sum* = 4) ซึ่งเมื่อใช้วิธีการนี้จะทำให้การจัดสรรงานได้ง่ายขึ้นเนื่องจากจะทราบว่าแต่ละโหนดสามารถรองรับงานที่เข้าสู่ระบบได้หรือไม่

นอกจากนี้วิทยานิพนธ์ฉบับนี้เสนอการใช้ อาเรย์ลิสต์ (Array-lists) ซึ่งนำไปใช้ในการจัดสรรและยกเลิกจัดสรรงานอย่างมีประสิทธิภาพ โดยอาเรย์ลิสต์จะเก็บค่าจำกัด (Limited Value : *limit*) ที่สามารถจัดสรรงานลงในแต่ละชั้นของโครงสร้างต้นไม้ และเก็บค่าโหนดเริ่มต้น *e* และรูปแบบ *p* ของแต่ละงานที่ถูกจัดสรร วิธีนี้จะทำให้ทราบว่าในงานนั้นๆประกอบด้วยโหนดอะไรบ้างในโครงสร้างต้นไม้ ดังนั้นค่าอินเด็กซ์ (Index) *i* ของอาเรย์ลิสต์จะเก็บค่า $limit = 2^{n-l}$ เช่น เมื่อ *l* หรือ $i = 0$ จะเก็บค่า $2^{6-0} = 2^6 = 64$ ดังนั้นบนเครือข่าย 6-HHC ค่า *limit* ในแต่ละ *i* จะอยู่ในช่วง 64, 32, ..., 2, 1 ตามลำดับ เมื่อ $0 \le i, l \le n$ และ $n = \log_2 N$

รูปที่ 3.29 ข) แสดงตัวอย่างของการใช้อาเรย์ลิสต์ของ 5 งานที่ถูกจัดสรร ($T_1 - T_5$) ที่ $i = 2$ จะมี *limit* = 16 และจะเก็บลิสต์ของงานที่ถูกจัดสรร คือ T_2 ที่มี $e = 1001$ และ $p = 0$ เพื่ออ้างถึง *cross*-node C1 = (1001, 1010) ที่อยู่ใน GCD_3 และ C0 = (1000, 1111) ที่อยู่ GCD_3 ต่อมางาน T_1, T_3 ,T_4 และ T_5 จะถูกจัดลงในอินเด็กซ์ที่ $i = 4$ ซึ่งมี *limit* = 4 ในโหนดที่ 0010, 1101, 1110 และ 1011 ตามลำดับ

จากรูปที่ 3.29 ข) ใน *group*-level สังเกตว่ามีความแตกต่างกับโหนดอื่นๆบนโครงสร้างต้นไม้ เช่น โหนด 10cc และ 11cc จะมีสัญลักษณ์การบล็อก (Blocking Status) ซึ่งคือการบล็อกข้อมูลเข้าเพื่อหลีกเลี่ยงการขัดแย้งกัน ในกรณีที่ถ้ามีบางงานที่ถูกจัดสรรมาสู่กลุ่ม (GCD) ที่มี Cross จำนวนหนึ่งกำลังประมวลผลอยู่ก่อนหน้า ซึ่งในกลุ่มนั้นจะมีพื้นที่สำหรับจัดสรรหน่วยประมวลผลอีกจำนวนหนึ่ง และถ้าระบบจัดสรรอีก Cross ที่มีหน่วยประมวลผลขนาด $k = 2^{m+j}$ เมื่อ $j \ge 1$ มาเพื่อประมวลผลจะทำให้เกิดการขัดแย้งกับงานที่กำลังประมวลผลอยู่ก่อนหน้า เช่น หลังจาก T_2 ที่มีขนาดหน่วยประมวลผล $k = 2^{m+2} = 16$ ถูกจัดสรร ดังนั้น *group*-node ที่ 10cc และ 11cc จะถูกบล็อกไม่ให้หน่วยประมวลผลที่มีขนาด $k = 2^{m+j}$ ถูกจัดสรรเข้าสู่ระบบ ยกเว้นกรณีที่ ขนาดของหน่วยประมวลผล $k \le 2^m$ จะสามารถถูกจัดสรรเข้าสู่ระบบได้

3.4.3 การจัดสรรและยกเลิกจัดสรรหน่วยประมวลผลบนเครือข่าย HHC

ในหัวข้อนี้เสนอการจัดสรรและยกเลิกจัดสรรของงานจากตารางงานบนเครือข่าย HHC ซึ่งประกอบด้วย ขั้นตอนวิธีที่ 3.5 คือ การจัดสรรงานแบบเฟิร์สฟิต (First-fit Allocation), ขั้นตอนวิธีที่ 3.6 การจัดสรรงานแบบเบสท์ฟิต (Efficient Best-fit Allocation) สำหรับงานที่เข้าสู่ระบบ และสุดท้าย คือ การยกเลิกการจัดสรรสำหรับงานที่ประมวลผลเสร็จสิ้น ด้วยขั้นตอนวิธีที่ 3.7

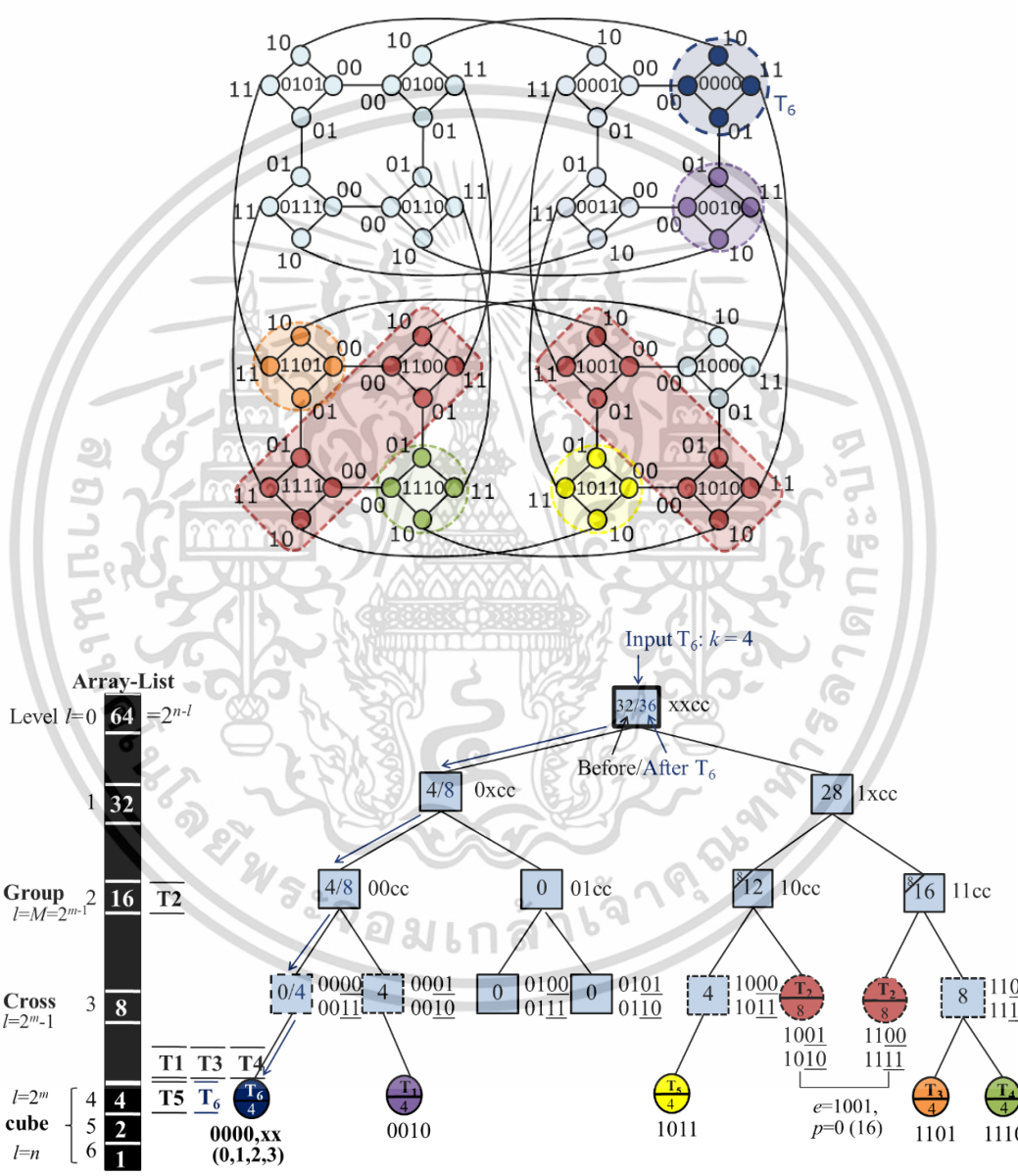
สำหรับขั้นตอนในการจัดสรรงาน สมมติข้อมูลเข้าขนาด k หมายถึงจำนวนหน่วยประมวลผลที่ต้องการใช้ระบบ เมื่อ $k \le 2^n/2^{M-1}$ เช่น $k \le 2^n/2$ ถ้า $m = 2$ ($N = 64$) และถ้าในระบบสามารถรองรับงานขนาด k ที่ต้องการเข้าสู่ระบบได้ ดังนั้นให้จัดสรรงานเข้าสู่ระบบ และทำการปรับปรุง (Update) ค่าต่างๆที่เกี่ยวข้องบนโครงสร้างต้นไม้ เช่น *sum* แต่ถ้าระบบไม่สามารถรองรับงานขนาด k ที่เข้าสู่ระบบได้ ให้ทำการนำงานขนาด k นั้นสู่คิวที่รอสำหรับประมวลผล (Waiting Queue)

ขั้นตอนวิธีที่ 3.5 ขั้นตอนการจัดสรรหน่วยประมวลผลแบบเฟิร์สฟิต บนเครือข่าย *n*-HHC

Step1. Form the root node (*P*), if the root *P* cannot satisfy the request (input) k (if *sum+k > limit*), put the request in the waiting queue.
Step2. Regular *cube*, *cross* GCD, or *combined* GCS ($k \le 2^n/2^{M-1}$).
Case1: $k \le 2^{m+1}$ (*cross* and *cube* nodes) in Section 3.1.
Apply DFS: depth *first* search (while $sum+k \le limit$) for the first available *cross*-node ($k = 2^{m+1}$) or *cube*-node ($k \le 2^m$) to allocate for the request k. If there is a free sub-system, then go to step 3. Otherwise, put the request in the waiting queue (for $k \le 2^m$)
Case2: $2^{m+2} \le k \le 2^n/2^{M-1}$ (*combined* nodes) in Section 3.2 - 3.3.
Apply DFS (while $sum+k \le limit$) for the first GCS pattern (*main-net e* and pattern *p*) to allocate for the request task k. If there is a free sub-system, go to step 3. Otherwise, put the request in the waiting queue.
Step3. Add the allocated task (with *main-net e* and pattern *p*) in the pointer-list at index *i* in the array-list at level *l*. Update the corresponding *sum* in each of all involved nodes in the tree.

รูปที่ 3.30 แสดงขั้นตอนการจัดสรรงานแบบเฟิร์สฟิต ด้วยข้อมูลเข้า T_5 ที่ต้องการหน่วยประมวลผลขนาด 4 หน่วยประมวลผล ($k = 2^m = 4$) บนเครือข่าย 6-HHC โดยกำหนดให้บนระบบมีงานเดิมที่ถูกประมวลผลอยู่ ประกอบด้วย T_1 มี 8 หน่วยประมวลผล บนโหนด (0010, 1011), T_2 มี 16 ประมวลผล บนโหนด C1 (10<u>01</u>, 10<u>10</u>) ในกลุ่มที่ 2 และ C0 (11<u>00</u>, 11<u>11</u>) ในกลุ่มที่ 3, T_3 มี 4 หน่วยประมวลผล บนโหนด 1101, T_4 มี 4 หน่วยประมวลผล บนโหนด 1110 และสำหรับข้อมูลเข้า T_5 ที่ต้องการหน่วยประมวลผล 4 หน่วยประมวลผล โดยการใช้ขั้นตอนวิธีที่ 3.5 เพื่อจัดสรรงานเข้าสู่ระบบ ขั้นตอนที่ 1 คือ นำข้อมูลเข้าสู่ระบบทางโหนดราก (Root Node) xxcc ซึ่งค่า *sum* ของ ระบบทั้งหมดหรือของโหนด xxcc คือ 32 ดังนั้นระบบจะสามารถรองรับงาน T_5 ได้ เนื่องจาก $sum+k = 32+4 \le 64 = limit$ ของในชั้น $l = 0$ ขั้นตอนที่ 2 ใช้วิธีการท่องกราฟแนวลึก (Depth First Search : DFS) บนต้นไม้เพื่อการจัดหาที่อยู่ (Accommodate) ที่เหมาะสมให้กับ T_5 โดยจะผ่านโหนด 0xcc ที่อยู่ชั้น $l = 1$ ($sum+k = 4+4 = 8 < limit = 32$) --> โหนด 00cc ที่อยู่ชั้น $l = 2$ ($sum+k = 4+4 = 8 < limit = 16$)

--> Cross โหนด C0: 0000, 0011 ในชั้น l = 3 (sum+k = 4+4 = 8 < $limit$ = 8) --> $cube$-level ในชั้น l = 4 และในชั้นนี้ จะมี $limit$ = 4 ซึ่งมีขนาดพอดีกับงาน T_5 ที่ต้องการ 4 หน่วยประมวลผล ดังนั้นงาน T_5 จะสามารถถูกจัดสรรลงในชั้นนี้ และถูกจัดลงในโหนด 0000,xx ซึ่งมีโหนดที่ถูกจัดสรรทั้งหมด คือ (0000,00), (0000,01), (0000,10) และ (0000,11) หลังจากได้โหนดที่สามารถรองรับงาน T_5 ได้แล้ว ขั้นต่อไปคือ ขั้นตอนที่ 4 คือการเก็บงาน T_5 และ e = 0000 ไว้ในอาเรย์ลิสต์ในชั้นที่โหนด 0000,xx อยู่ และสุดท้ายทำการปรับปรุงค่า sum = sum + k ของโหนดที่ท่องผ่านมาทั้งหมด จนถึงโหนดราก



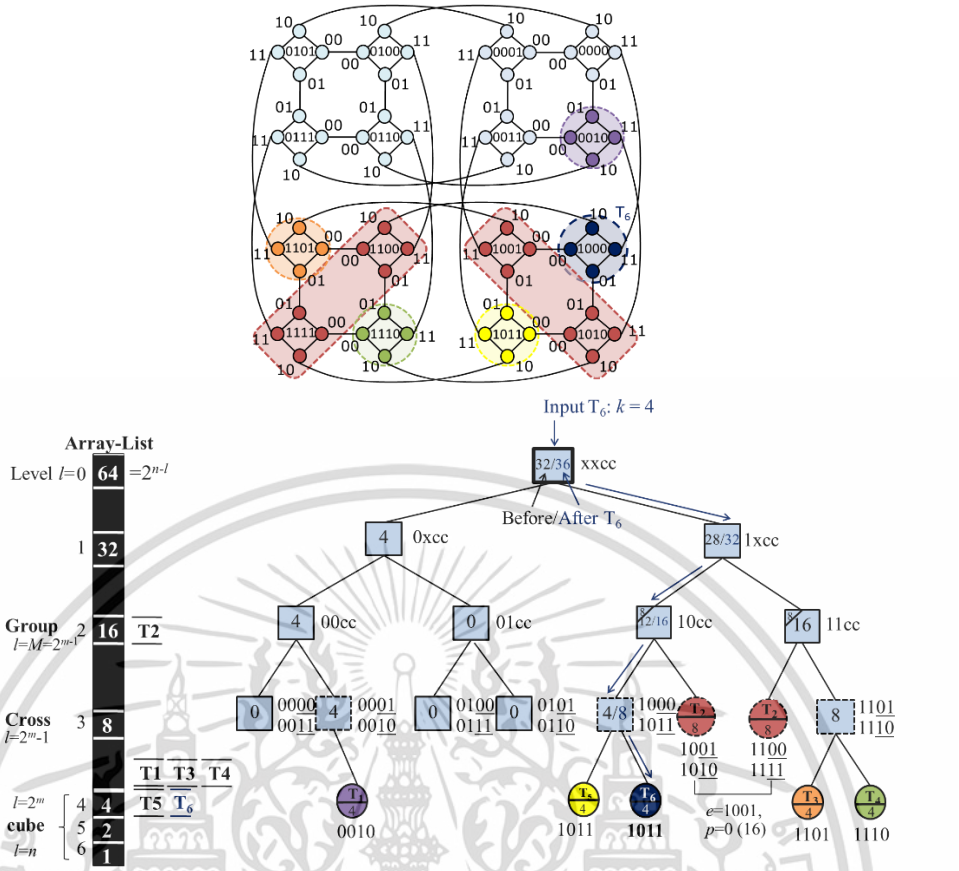
รูปที่ 3.30 ตัวอย่างการจัดสรรงาน T_6 = 4 โหนด ลงบน 6-HHC และโครงสร้างต้นไม้ ด้วยวิธีเฟิร์สฟิต

ขั้นตอนวิธีที่ 3.6 ขั้นตอนการจัดสรรหน่วยประมวลผลแบบเบสท์ฟิต บนเครือข่าย *n*-HHC

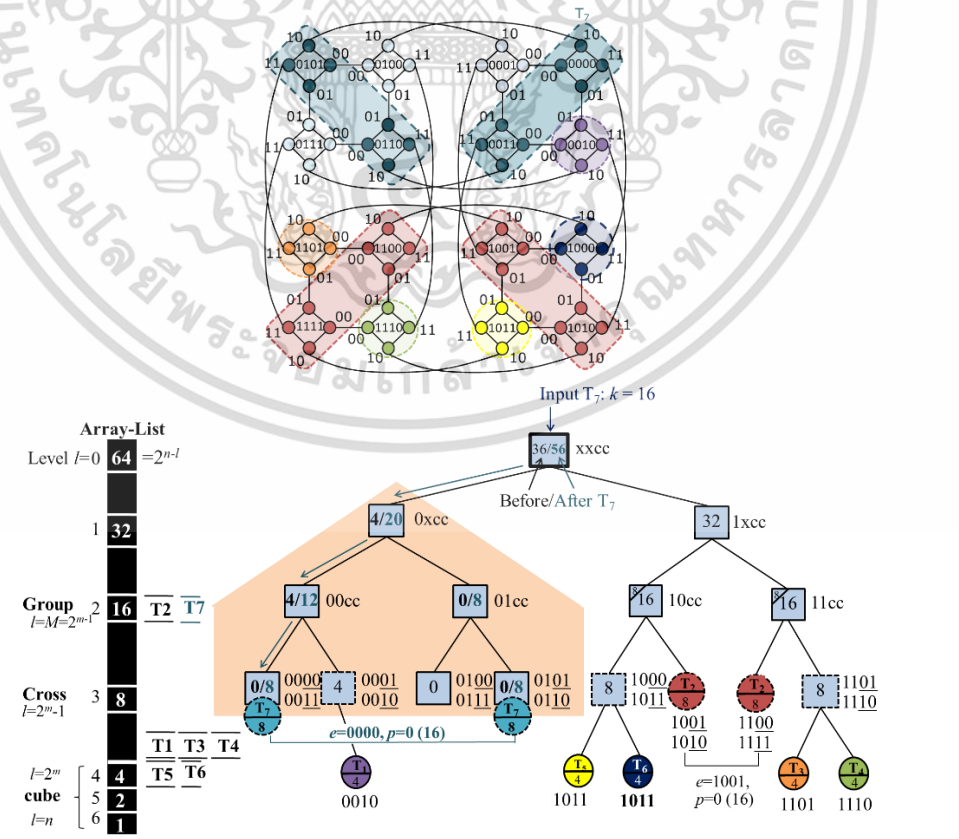
Step1. Form the root node (*P*), if the root *P* cannot satisfy the request *k* (*sum+k* > *limit*), put the request in the waiting queue.
Step2. Regular *cube*, *cross* GCD, or *combined* GCS ($k \le 2^n/2^{M-1}$) and
Step3. Extra combined GCS ($2^{m+1} \le k \le 2^n/2^{M-1}$). Similar to Algorithm 3.5, except replace DFS with DBS (depth *best* search). For DBS, compute L and R factors of child nodes at the parent node, where LF or RF = *climit* – (*csum* + *k*) and then go to the best node with *min*(LF, RF) ≥ 0.
Step4. Add the allocated task (with *main-net e* and pattern *p*) in the pointer-list at index *i* in the array-list at level *l*. Update the corresponding *sum* in each of all involved nodes in the tree.

ขั้นตอนวิธีที่ 3.6 แสดงขั้นตอนในการจัดสรรหน่วยประมวลผลแบบเบสท์ฟิต บนเครือข่าย *n*-HHC ซึ่งจะแตกต่างกับขั้นตอนวิธีที่ 3.5 โดยการเปรียบเทียบค่า *sum* ของโหนดลูกทางซ้ายมือ (Left Factor : LF) และขวามือ (Right Factor : RF) ของโหนดพ่อแม่ ถ้าลูกฝั่งใดมีค่า LF หรือ RF น้อยกว่า (min(LF,RF) ≥ 0) จะดำเนินวิธีการท่องกราฟดีที่สุด (Depth best search : DBS) ท่องไปในโครงสร้างต้นไม้ของลูกฝั่งนั้นเมื่อ LF และ RF = *limit* – (*sum* + *k*) ซึ่งการใช้วิธีนี้ในการจัดสรรหน่วยประมวลผล จะทำให้ระบบรองรับการจัดสรรหน่วยประมวลผลได้อย่างมีประสิทธิภาพมากขึ้น เนื่องจาก วิธีนี้จะเลือกจัดสรรหน่วยประมวลผลในกลุ่ม GCD ที่ใกล้จะเต็มก่อนเป็นอันดับแรก

รูปที่ 3.31 แสดงตัวอย่างของการจัดสรรหน่วยประมวลผลแบบเบสท์ฟิต บนเครือข่าย 6-HHC โดยมีงานใหม่ T_5 ที่ต้องการหน่วยประมวลผล $k = 2^m = 4$ ขั้นตอนที่ 3.6 เริ่มต้นที่โหนดราก xxcc ซึ่งสามารถรองรับหน่วยประมวลผล 4 หน่วยประมวลผลได้ เนื่องจาก *sum+k* = 32 + 4 = 36 ≤ 64 = *limit* ต่อมาขั้นตอนที่ 2 ทำการหาค่าที่น้อยที่สุดจาก LF และ RF ในชั้นที่ *l* = 1 โดย LF = *limit* – (*sum* + *k*) = 32 – (4 + 4) = 24 และ RF = *limit* – (*sum* + *k*) = 32 – (28 + 4) = 0 ดังนั้น min(LF, RF) = min(24, 0) = 0 ดังนั้น จึงดำเนินการ DBS ไปในลูกทางขวา (Right Child) คือ โหนด 1xcc ที่อยู่ชั้น *l* = 1 (*sum+k* = 28 + 4 = 8 = *limit* = 32) --> โหนด 10cc ที่อยู่ชั้น *l* = 2 (*sum* + *k* = 12 + 4 = 16 = *limit*) --> Cross โหนด C0: 1000, 1011 ในชั้น *l* = 3 (*sum+k* = 4+4 = 8 = *limit*) --> *cube*-level ในชั้น *l* = 4 และในชั้นนี้ จะมี *limit* = 4 ซึ่งมีขนาดพอดีกับงาน T_5 ที่ต้องการ 4 หน่วยประมวลผล ดังนั้นงาน T_5 จะสามารถถูกจัดสรรลงในชั้นนี้ และถูกจัดลงในโหนด 1000, xx (รูป 3.29 โหนดสีเขียว) ซึ่งมีโหนดที่ถูกจัดสรรทั้งหมด คือ (1000, 00), (1000, 01), (1000, 10) และ (1000, 11) หลังจากได้โหนดที่สามารถรองรับงาน T_5 ได้แล้ว ต่อไปขั้นตอนที่ 4 คือการเก็บงาน T_5 และ *e* = 1000 ไว้ในอาเรย์ลิสต์ในชั้นที่โหนด 1000, xx อยู่ และสุดท้ายทำการปรับปรุงค่า *sum* = *sum* + *k* ของโหนดที่ท่องผ่านมาทั้งหมดจนถึงโหนดราก เมื่อลองเปรียบเทียบจากรูปที่ 3.30 และ 3.31 จะเห็นว่าการจัดสรรหน่วยประมวลผลของเบสท์ฟิต จะมีการใช้พื้นที่ของระบบได้อย่างมีประสิทธิภาพ เช่น สมมติว่างาน T_1 บนโหนด 0010 และ 1011 ประมวลผลเสร็จทำให้บนกลุ่มที่ 1 หรือ GCD1 จะว่างทั้งหมด ขณะที่วิธีเฟิร์ทฟิตไม่ว่างทั้งหมด



รูปที่ 3.31 ตัวอย่างการจัดสรรงาน $T_6 = 4$ โหนดลงบน 6-HHC และโครงสร้างต้นไม้ ด้วยวิธีเบสท์ฟิต



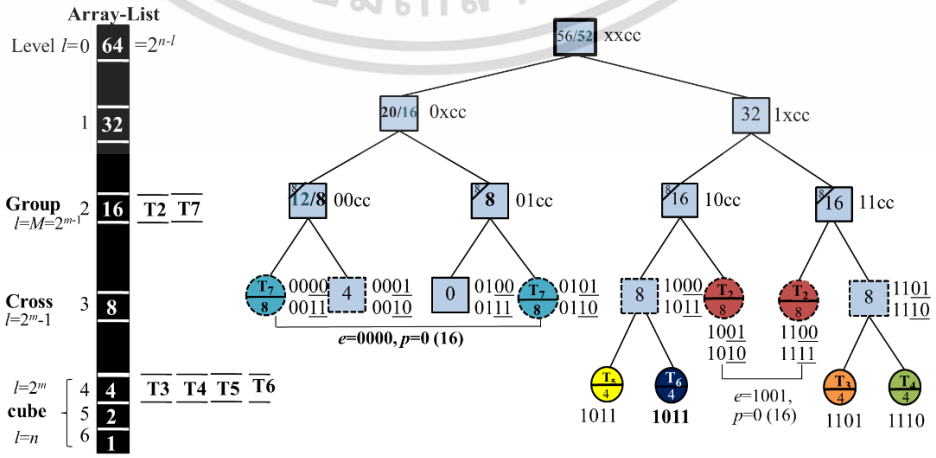
รูปที่ 3.32 ตัวอย่างการจัดสรรงาน $T_7 = 16$ ลงบน 6-HHC และโครงสร้างต้นไม้ ด้วยวิธีเบสท์ฟิต

รูปที่ 3.32 แสดงตัวอย่างของการจัดสรรหน่วยประมวลผลแบบเบสท์ฟิต บนเครือข่าย 6-HHC โดยมีงานใหม่ T_6 ที่ต้องการหน่วยประมวลผล $k = 2^{m+2} = 16$ หน่วยประมวลผล โดยใช้การรวมกลุ่มแบบ GCS เริ่มต้นขั้นตอนจากงาน T_6 เข้าสู่ระบบทางโหนดราก xxcc และตรวจสอบค่า min(LF,RF) = min(12, -16) แต่ค่า min ที่ได้ต้องมากว่าหรือเท่ากับ 0 ดังนั้นจากโหนดรากจึงดำเนินการท่องไปในต้นไม้ เข้าสู่ชั้นที่ $l = 1$ ด้วย DBS ไปทาง LF หรือโหนด 0xcc และในชั้นต่อไป $l = 2$ (*group*-level) จะเป็นชั้นที่ *limit* = $k = 16$ และจะสร้าง GCS สำหรับจัดสรรงาน T_6 ที่มีรูปแบบ $p = 0$ และ $e = 0000$ บนชั้นนี้ และจะประกอบด้วยสอง *cross*-node ในชั้น $l = 3$ คือ C0:(0000,0011) ในกลุ่มที่ 0 และ C1:(0101, 0110) ในกลุ่มที่ 1 (โหนดสีฟ้า) ต่อไปขั้นตอนที่ 4 คือ การเก็บงาน T_6 และ $e = 0000$ และ $p = 0$ ไว้ในอาเรย์ลิสต์ในชั้นที่ $l = 2$ และสุดท้ายทำการปรับปรุงค่า *sum* = *sum* + k ของโหนดที่ท่องผ่านมาทั้งหมดจนถึงโหนดราก และสังเกตว่าในโหนดที่ 00cc และ 01cc จะมีสัญลักษณ์การบล็อก เนื่องจากจะมีการขัดแย้งกัน ถ้ามีงานที่มีขนาด $k = 2^{m+j}$ เมื่อ $j \geq 1$ เข้าสู่ระบบ ดังนั้นจึงต้องมีการแสดงสัญลักษณ์เพื่อปฏิเสธงานดังกล่าว

ในกรณีที่ระบบมีงานที่ประมวลผลหรือใช้งานหน่วยประมวลผลบนระบบเสร็จสิ้น จะต้องทำการยกเลิกการจัดสรร (Deallocation) โดยการลบงานที่เสร็จสิ้นจากอาเรย์ลิสต์ พร้อมด้วยค่า e และ p (ถ้ามี) ด้วยการใช้ขั้นตอนวิธีที่ 3.7 เช่น รูปที่ 3.33 แสดงการยกเลิกจัดสรรงาน T_1 ที่มีหน่วยประมวลผลขนาด 8 คือ (0010,xx) และ (1011,xx) ขั้นตอนแรกจะทำการลบ T_1 รวมทั้งค่า e และรูปแบบ p ในอาเรย์ลิสต์ในอินเด็กซ์ที่ $i = n - \log_2 k = 3$ และต่อไป คือ ทำการปรับปรุงค่า *sum* ของโหนดที่เกี่ยวข้องทั้งหมดจนถึงโหนดรากของระบบ

ขั้นตอนวิธีที่ 3.7 ขั้นตอนการยกเลิกการจัดสรรหน่วยประมวลผลเมื่องานเสร็จสิ้น บน *n*-HHCs

Step1. For a finished task (of k processors), search in the array list at level $l = n\text{-}\log_2 k$ to free the corresponding nodes in the tree.
Step2. Delete that finished task from the array list and combine free nodes up to the *cross*-level.
Step3. Update the *sum* in all corresponding node up to the root and update the blocking status (if any) at the *group*-level.



รูปที่ 3.33 ตัวอย่างการยกเลิกการจัดสรรงาน T_1 ที่มี 4 หน่วยประมวลผล

บทที่ 4
การพิสูจน์ความถูกต้อง และประสิทธิภาพของขั้นตอนวิธี

ในหัวข้อนี้จึงนำเสนอการพิสูจน์ความถูกต้อง และการประเมินประสิทธิภาพของขั้นตอนวิธีที่นำเสนอในบทก่อนหน้านี้ ซึ่งประกอบด้วย การหาเส้นทางที่สั้นที่สุดแบบหนึ่งต่อหนึ่ง บนเครือข่าย HHC จะนำเสนอในหัวข้อ 4.1 การหาเส้นทางที่สั้นที่สุดแบบขนานสำหรับการสื่อสารแบบ ATAPE จะนำเสนอในหัวข้อที่ 4.2 การพิสูจน์ความถูกต้องในการสื่อสารพร้อมกันของ Cross เลขคู่ และ Cross เลขคี่ ในหัวข้อ 4.3 การพิสูจน์ความถูกต้องของการจัดสรรงานบนโครงสร้างต้นไม้แบบทวิภาค ด้วยการรวมกลุ่มแบบ GCD และ GCS ในหัวข้อที่ 4.4, การวิเคราะห์ความซับซ้อนทางด้านเวลาของขั้นตอนการจัดสรร/ยกเลิกจัดสรรงาน ในหัวข้อที่ 4.5 และสุดท้ายหัวข้อ 4.6 นำเสนอการประเมินประสิทธิภาพ และการเปรียบเทียบผล

4.1 การพิสูจน์ความถูกต้องของการหาเส้นทางที่สั้นที่สุดแบบหนึ่งต่อหนึ่ง บนเครือข่าย HHC

การหาเส้นทางที่สั้นที่สุด และไม่มีความขัดแย้งกันขณะสื่อสาร บนเครือข่าย HHC จะมีการนำเสนอการพิสูจน์ขั้นตอนต่างๆ ต่อไปนี้

1) การหาเส้นทางที่สั้นที่สุดของเครือข่ายด้านนอก หรือ μ ซึ่งถูกนำเสนอด้วยขั้นตอนวิธีที่ 3.1
2) การหาเส้นทางที่สั้นที่สุดจากต้นทาง S ไปยังปลายทาง D ด้วยการใช้ขั้นตอนวิธีที่ 3.2 และการเรียงลำดับบิตใหม่เพื่อให้ได้เส้นทางที่สั้นที่สุดด้วยการใช้ฟังก์ชัน SPordering

ในขั้นตอนที่ 1 การหา μ โดย $\mu = \{ c_0, c_1, \ldots, c_{r-1} \}$ ซึ่งจะเป็นขั้นตอนแรกในการหาเส้นทางที่สั้นที่สุด โดยหาได้จากระยะทางแฮมมิ่ง (Hamming Distance) ของเครือข่ายหลัก (Main-Network) ระหว่างโหนดต้นทาง $S = (\alpha_S, \beta_S)$ และปลายทาง $D = (\alpha_D, \beta_D)$ เนื่องจากโครงสร้างของระยะทางแฮมมิ่งจะมีบิตที่อยู่ติดกันที่ห่างกัน 1 บิต ดังนั้นบิตที่ได้จากขั้นตอนนี้ จะเป็นเส้นทางที่ห่างกันอย่างน้อย 1 บิตเช่นกัน นั้นหมายความว่าเส้นทางระหว่างต้นทาง S และ D จะเป็นเส้นทางที่สั้นที่สุด

สำหรับขั้นตอนที่ 2 ในการหาเส้นทางที่สั้นที่สุด จากขั้นตอนวิธีที่ 3.2 จะเป็นการพิจารณากรณี ที่มี 5 กรณี เห็นได้ชัดแจนว่าในกรณีที่ $\alpha_S = \alpha_D$ หรือหมายถึง ต้นทาง S และปลายทาง D อยู่ในเครือข่ายหลักเดียวกัน นั้นแสดงว่าจะเป็นการหาเส้นทางที่อยู่ในไฮเปอร์คิวบ์ (Hypercube Routing) ซึ่งเส้นทางไฮเปอร์คิวบ์จะเป็นเส้นทางที่สั้นที่สุดเสมอ ถ้าเป็นการเปลี่ยนบิตจากต้นทาง S สู่ปลายทาง D ทีละ 1 บิต สำหรับกรณีที่ $\alpha_S \neq \alpha_D$ หรือหมายถึง ต้นทาง S และปลายทาง D อยู่คนละเครือข่ายหลัก จะมีหลักการสำคัญในการได้มาซึ่งเส้นทางที่สั้นที่สุด คือ การตรวจสอบกรณีทั้ง 5 กรณี โดยทั้ง 5 กรณีจะเป็นการเรียงลำดับบิตที่ดีที่สุด คือ สั้นที่สุดและหลีกเลี่ยงการชนขณะสื่อสารแบบขนาน สุดท้ายสำหรับขั้นตอนที่ 3 ในการหาเส้นทางที่สั้นที่สุด จะถูกพิสูจน์ในทฤษฎีบทที่ 4.1

<u>ทฤษฎีบทที่ 4.1</u> บนเครือข่าย *n*-HHC เส้นทาง $P = \{p_0, p_1, ..., p_{r-1}\}$ ระหว่างโหนดต้นทาง $S = (\alpha_S, \beta_S)$ และปลายทาง $D = (\alpha_D, \beta_D)$ ใดๆ คือเส้นทางที่สั้นที่สุด ถ้า $|P| = r \leq | \alpha_S \oplus \alpha_D |_{bit1}$ และ ทุกๆ p_i ที่อยู่ติดกันในเส้นทาง P ต้องมีบิตที่ต่างกันน้อยที่สุด เมื่อ $0 \leq i \leq r-1$, $| P |$ คือ จำนวนอินเด็กซ์ (Index) ในเซต P และ $| P |_{bit1}$ คือ จำนวนบิต 1 ที่อยู่ในเซต P

พิสูจน์ สำหรับการสื่อสารแบบหนึ่งต่อหนึ่ง ของโหนดต้นทาง S และปลายทาง D บนเครือข่าย *n*-HHC ที่มีเส้นทางการสื่อสารที่สั้นที่สุด สามารถหาเส้นทางจากขั้นตอนวิธีที่ 3.2 ซึ่งจะเห็นได้ชัดว่า ถ้า S และ D อยู่ในเครือข่ายหลักเดียวกัน ($\alpha_S = \alpha_D$) จะเป็นการสื่อสารระหว่างต้นทาง S และปลายทาง D ที่เรียกว่า การสื่อสารในไฮเปอร์คิวบ์ (Hypercube Routing) ซึ่งจะเป็นเส้นทางที่สั้นที่สุดเสมอ สำหรับกรณีที่ S และ D ไม่อยู่ในเครือข่ายหลักเดียวกัน ($\alpha_S \neq \alpha_D$) การหาเส้นทางจากต้นทาง S จะต้องมีการสื่อสารข้ามระหว่างเครือข่ายหลักและเครือข่ายรอง สลับกันเรื่อย ๆ จนถึงปลายทาง D ดังนั้นในกรณีนี้ขั้นตอนแรกคือการหา $\mu = \{ c_0, c_1, ..., c_{r-1} \}$ ซึ่งก็คือ เซ็ตของบิต 1 จากการดำเนินการ $\alpha_S \oplus \alpha_D$ และเทียบบิตที่ได้กับ *m*-บิต เกรย์โค๊ด (*m*-bit Gray Code) ขั้นต่อมาคือการหาเส้นทางที่สั้นที่สุด (P) โดยการใช้การเรียงลำดับอินเด็กซ์ใน μ ใหม่ เพื่อให้ได้เส้นทางที่เหมาะสม โดยในขั้นตอนนี้จะแบ่งเป็น 5 กรณี ที่แตกต่างกันตามพารามิเตอร์ (Parameter) β_S และ β_D ใน μ ภายในแต่ละกรณีจะมีการใช้ฟังก์ชัน SPordering สำหรับการเรียงลำดับอินเด็กซ์ใน μ เพื่อที่จะทำให้อินเด็กซ์ที่ติดกันมีบิตที่แตกต่างกันน้อยที่สุด เมื่อพิจารณาแต่ละกรณีจะแบ่งเป็นกลุ่มๆ โดยกลุ่มแรกจะเป็นกรณีที่ $\beta_S \in \mu$ ซึ่งจะถูกจัดอยู่ในกรณี 1 – 3 ในกรณีเหล่านี้ จะเริ่มต้นโดยการให้ β_S เริ่มต้นเป็นอินเด็กซ์แรกใน P ต่อมาสำหรับกรณีที่ $\beta_D \in \mu$ จะใช้หลักการเดียวกัน คือ จะนำ β_D ไว้ท้ายสุดของอินเด็กซ์ ใน P ซึ่งจะพบในกรณีที่ 2 และ 4 โดยในทุกกรณีจะได้ผลลัพธ์ P ที่แต่ละอินเด็กซ์จะถูกจัดเรียงใหม่เพื่อให้ได้เส้นทางที่สั้นที่สุด โดยใช้ฟังก์ชัน SPorderfing ด้วย $O(|\mu|^2)$

4.2 การพิสูจน์ความถูกต้องของการหาเส้นทางที่สั้นที่สุดแบบขนาน

เนื่องจากขั้นตอนวิธีที่ 3.2 ไม่สามารถแก้ปัญหาการชนกันเมื่อรับส่งข้อมูลพร้อมกันแบบขนานได้ ดังแสดงในรูปที่ 3.6 ดังนั้นเพื่อแก้ปัญหานี้ ขั้นตอนวิธีที่ 3.3 จึงถูกเสนอมาเพื่อแก้ปัญหานี้ ซึ่งก็คือ ฟังก์ชัน dSPordering ซึ่งเป็นวิธีการเรียงอินเด็กซ์ ที่มีความยืดหยุ่นกว่าขั้นตอนวิธีที่ 3.2 โดยจะทำการหาตำแหน่ง d เพื่อเริ่มต้นเปรียบเทียบบิตที่อยู่ใน μ ด้วยการใช้วิธีเลื่อนไปข้างหน้าแบบวนกลับ (Forward) ซึ่งเป็นวิธีเริ่มต้น (Default) และการใช้วิธีถอยหลังแบบวนกลับ (Backward) ซึ่งสองเทคนิคนี้เป็นวิธีการหาเส้นทางที่เลี่ยงการชนกัน โดยจะเลือกเส้นทางที่สมมาตรกัน แต่ยังคงได้เส้นทางที่สั้นที่สุด ดังแสดงในรูปที่ 3.7 ซึ่งวิธีนี้ไม่จำเป็นต้องเริ่มต้นเปรียบเทียบที่บิตแรกของ μ หรือ $d = 0$

นอกจากนั้นขั้นตอนวิธีที่ 3.3 ยังสามารถลดกรณี (Case) ที่ใช้ในการตรวจสอบการเรียงลำดับอินเด็กซ์ใน μ ซึ่งจากเดิมในขั้นตอนวิธีที่ 3.2 จะประกอบด้วย 5 กรณี แต่ขั้นตอนวิธีที่ 3.3 ลดลงเหลือ 2 กรณี ก็คือกรณีที่ $\beta_S \in \mu$ หรือไม่เท่านั้น ดังนั้นยิ่งกรณีลดลงเท่าไร จะทำให้ความซับซ้อน และการชนกันของข้อมูล ลดลงเท่านั้น

โดยปกติแล้ว ขั้นตอนวิธีที่ 3.3 จะใช้วิธีการเรียงลำดับอินเด็กซ์ใหม่โดยการใช้การเรียงลำดับไปข้างหน้าแบบวนกลับซึ่งจะทำให้ได้เส้นทางที่สั้นที่สุด นอกจากนี้ยังเสนอการเรียงลำดับถอยหลังแบบวนกลับ ซึ่งถูกเสนอมาเพื่อหาทิศทางที่ตรงกันข้ามกัน (Bidirectional-Routing) เมื่อมีการรับส่งข้อมูลแบบ ATAPE โดยจะเห็นได้ชัดว่าขั้นตอนวิธีที่ 3.3 สามารถประยุกต์ใช้ได้โดยไม่เกิดความขัดแย้งกันสำหรับการหาเส้นทางการสื่อสารของหน่วยประมวลผลแบบขนานด้วยขนาด $k = 2^{m+1}$ โดยจะมีการพิสูจน์คล้ายกับทฤษฎีที่ 4.1

4.3 การพิสูจน์ความถูกต้องของการแบ่งกลุ่มแบบ GCD และการหาเส้นทางแบบสองทิศทางด้วย Cross เลขคู่ และเลขคี่

การแบ่งกลุ่มแบบ GCD ที่ภายในแต่ละกลุ่มประกอบด้วย Cross จำนวน $2^{2^{m-1}-1}$ และ $k = 2^{m+1}$ ซึ่งแต่ละ Cross ใน GCD สามารถสื่อสารเพื่อรับส่งข้อมูลแบบขนานที่ไม่มีความขัดแย้งกัน โดยถูกพิสูจน์ในทฤษฎีบทที่ 3.1 – 3.3

การพิสูจน์ความถูกต้องของการสื่อสารแบบ ATAPE โดยใช้การหาเส้นทางด้วยขั้นตอนวิธีที่ 3.3 และการแบ่งกลุ่มแบบ GCD ที่มีจำนวนหน่วยประมวลผล $k = 2^{m+1}$ โดยการใช้สมการที่ 3.3 $D_j^C = d_{i,j} = S_{(C_i \oplus j)}$ ไม่เพียงแค่ทำให้ได้เส้นทางการสื่อสารที่มีความสมมาตร (Symmetrical-path) แต่ยังครอบคลุมเงื่อนไขของทั้งเครือข่ายหลัก โดยการใช้ตัวควบคุม (Control Unit : C_i) จำนวน k ตัว เมื่อ $i = 0, 1, 2, ..., k\text{-}1$ และ C_i คือ $0, 1, 2, ..., k\text{-}1$ ดังนั้นแต่ละ C_i คือการหาระยะทางแฮมมิ่ง (Hamming Distance) ของต้นทาง S_i และ D_i ที่อยู่ในแต่ละ Cross

ทฤษฎีที่ 4.2 เสนอความถูกต้องของที่ประกอบด้วย 3 อย่าง คือ งานที่ถูกต้อง (Right Task), กลุ่มที่ถูกต้อง (Right Partition), เวลาที่ถูกต้อง (Right Time) และ ทฤษฎีที่ 4.3 นำเสนอความถูกต้องของ Cross เลขคู่และเลขคี่ที่มีทิศทางการสื่อสารที่ตรงกันข้ามกัน

<u>ทฤษฎีบทที่ 4.2</u> การหาเส้นทางการสื่อสาร ATAPE แบบขนาน ด้วยจำนวนหน่วยประมวลผลขนาด $k = 2^{m+1}$ สามารถสื่อสารโดยไม่มีการขัดแย้ง บนเครือข่าย n-HHC เมื่อ $n = 2^m+m$ โดยการใช้การแบ่งกลุ่มแบบ GCD ซึ่งทำให้ได้ความถูกต้องที่ประกอบด้วย 3 อย่าง (Triple Right Assignment) คือ งานที่ถูกต้อง (Right task), กลุ่มที่ถูกต้อง (Right Partition), เวลาที่ถูกต้อง (Right time)

พิสูจน์ ในขั้นตอนวิธีที่ 3.3 dSPordering ที่มีการเรียงลำดับไปข้างหน้าแบบวนกลับ (Forward) และถอยหลังแบบวนกลับ (Backward) สามารถทำให้ได้เส้นทางการสื่อสารที่ไม่มีความขัดแย้งกัน สำหรับการจัดกลุ่มแบบ GCD จะถูกใช้เพื่อการหาต้นทางที่เหมาะสมสำหรับงาน k งาน และใช้ฟังก์ชัน XOR จากสมการที่ 3.3 เพื่อหา k ปลายทางที่เหมาะสมในการสื่อสารด้วยทิศทางที่มีความสมมาตร (Symmetrical-path) เห็นได้ชัดว่าการขั้นตอนวิธีที่ 3.3 และทฤษฎีที่ 3.1 – 3.3 สามารถทำให้

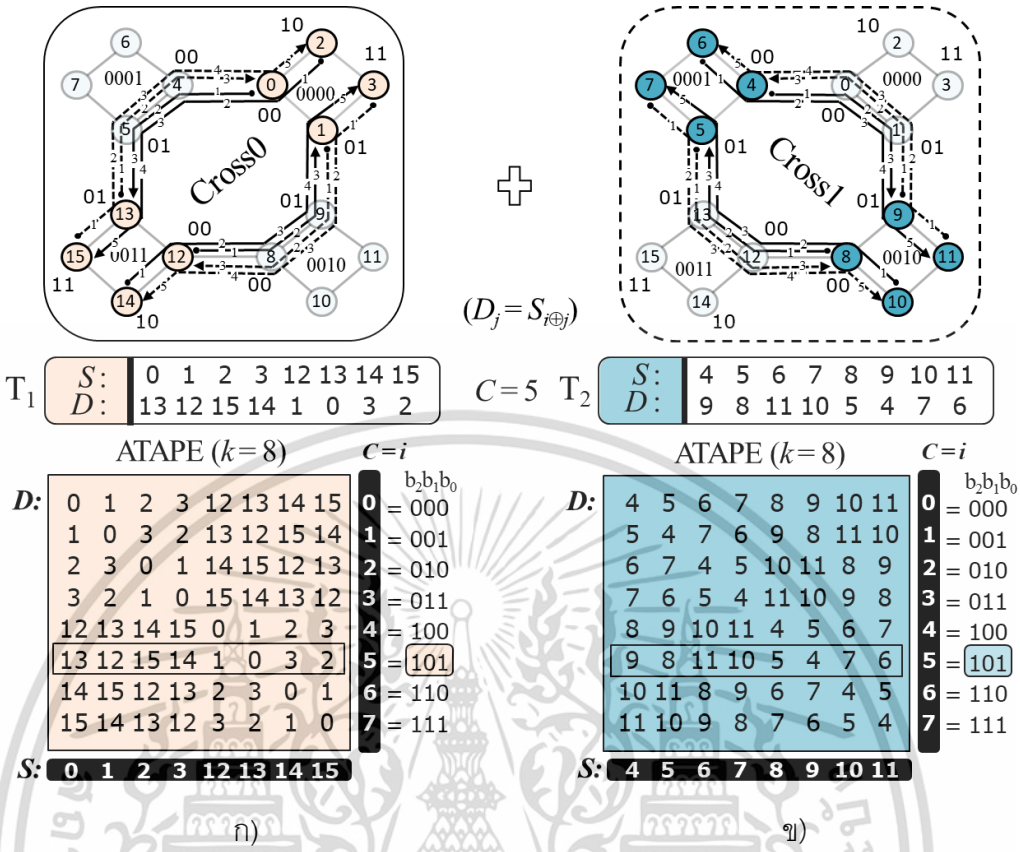
งานวิจัยนี้ได้รับความถูกต้อง 3 ประการ โดยสามารถระบุงานที่ถูกต้อง (Right Task) เหมาะสม สู่การแบ่งกลุ่มที่เหมาะสม (Right Partition) ภายในเวลาที่เหมาะสม (Right Time) สำหรับงานใดๆ ที่มีขนาด $k = 2^{m+1}$ โดยใช้การแบ่งกลุ่มแบบ GCD และการเรียงลำดับบิตใหม่ (Reordering) ทำให้ได้เส้นทางการสื่อสารภายในเวลา $O(|\mu|^2)$ โดยเป็นเส้นทางที่สั้นที่สุด, ไม่มีการชนกันขณะสื่อสาร และสื่อสารได้พร้อมกันในเวลาเดียวกัน (Time Multiplexing) ในแต่ละกลุ่ม □

<u>ทฤษฎีบทที่ 4.3</u> ทุกๆ Cross เลขคู่และเลขคี่ ภายใต้การแบ่งกลุ่มแบบ GCD บนเครือข่าย *n*-HHC เมื่อ $n = 2^m+m$ สามารถสื่อสารเพื่อรับส่งข้อมูลพร้อมกันแบบขนาน โดยไม่มีความขัดแย้งกัน

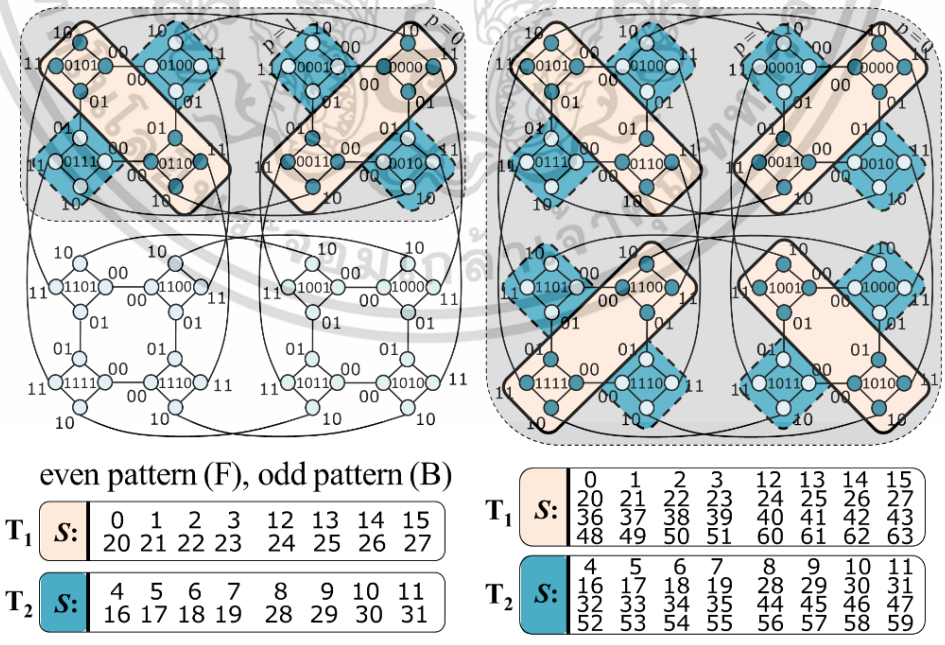
พิสูจน์ การใช้ประโยชน์ของ Cross ที่อยู่ในแต่ละ GCD แบบ 100% โดยที่ไม่มีการขัดแย้งกันขณะสื่อสาร สามารถทำได้โดยการใช้หลักการของการสื่อสารแบบ Cross เลขคู่และเลขคี่ (Evan and Odd Crosses) โดยกำหนดให้ Cross เลขคู่ใช้การเรียงลำดับอินเด็กซ์ไปข้างหน้าแบบวนกลับ และกำหนดให้ Cross เลขคี่ ใช้การเรียงลำดับอินเด็กซ์ไปข้างหลังแบบวนกลับ ซึ่งเมื่อมีการหาเส้นทางการสื่อสารด้วยวิธีดังกล่าว จะสังเกตได้ว่าเส้นทางทั้งคู่จะเป็นเส้นทางที่มีทิศทางที่ตรงกันข้ามกัน ซึ่งโดยปรกติแล้วการหาเส้นทางไปข้างหน้าแบบวนกลับ เพียงวิธีเดียวจะใช้ประสิทธิภาพของเส้นเชื่อมระหว่างหน่วยประมวลผลที่อยู่ในแต่ละกลุ่มได้เพียงแค่ 50% และสังเกตได้ว่าจะเหลือพื้นที่สำหรับเส้นทางแบบย้อนหลัง (Backward) อีก 50% ที่ยังไม่ได้ใช้ประโยชน์ และสามารถใช้งานได้ ดังแสดงในรูปที่ 4.1 เมื่อ $k = 2^{m+1}$, $m = 2$ และ $C = 5$ บนเครือข่าย 6-HHC โดยรูปที่ 4.1 ก) แสดงให้เห็นถึงการหาเส้นทางการสื่อสารแบบ ATAPE ของ $Cross_0$ โดยใช้เส้นทางไปข้างหน้าแบบวนกลับ (Forward) ซึ่งเป็นวิธีเริ่มต้นในการหาเส้นทางด้วยขั้นตอนวิธีที่ 3.3 รูปที่ 4.1 ข) แสดงเส้นทางการสื่อสารแบบ ATAPE ของ $Cross_1$ ที่มีทิศทางในลักษณะตรงกันข้ามกับ $Cross_0$ แต่ทั้งสอง Cross สามารถสื่อสารแบบ ATAPE ได้พร้อมกันในเวลาเดียวกันและไม่มีความขัดแย้งกัน เมื่อ $k = 8$ และรูปที่ 4.2 เมื่อ $k = 16$ และ 32 ตามลำดับ □

สำหรับในกรณีที่เครือข่าย HHC มีขนาดใหญ่ขึ้น เช่น $m = 3$, $N = 2048$ จะมี Cross เลขคู่และเลขคี่ ที่การรับส่งข้อมูลแบบ ATAPE ที่มีทิศทางตรงกันข้ามกัน คือ ($Cross_0$, $Cross_5$), ($Cross_2$, $Cross_7$), ($Cross_3$, $Cross_6$), และ ($Cross_4$, $Cross_1$) ตามลำดับ ดังแสดงในรูปที่ 4.3 ก) ด้วยวิธีการหาเส้นทางแบบตรงกันข้ามกัน โดยให้ Cross เลขคู่ ใช้การหาเส้นทางไปข้างหน้าแบบวนกลับ และ Cross เลขคี่ใช้การหาเส้นทางถอยหลังแบบวนกลับ จะสามารถสื่อสารแบบ ATAPE โดยไม่มีความขัดแย้งกัน สำหรับตัวอย่างที่มีความซับซ้อนมากขึ้นแสดงในรูปที่ 4.3 ข) กรณีที่ $m = 4$

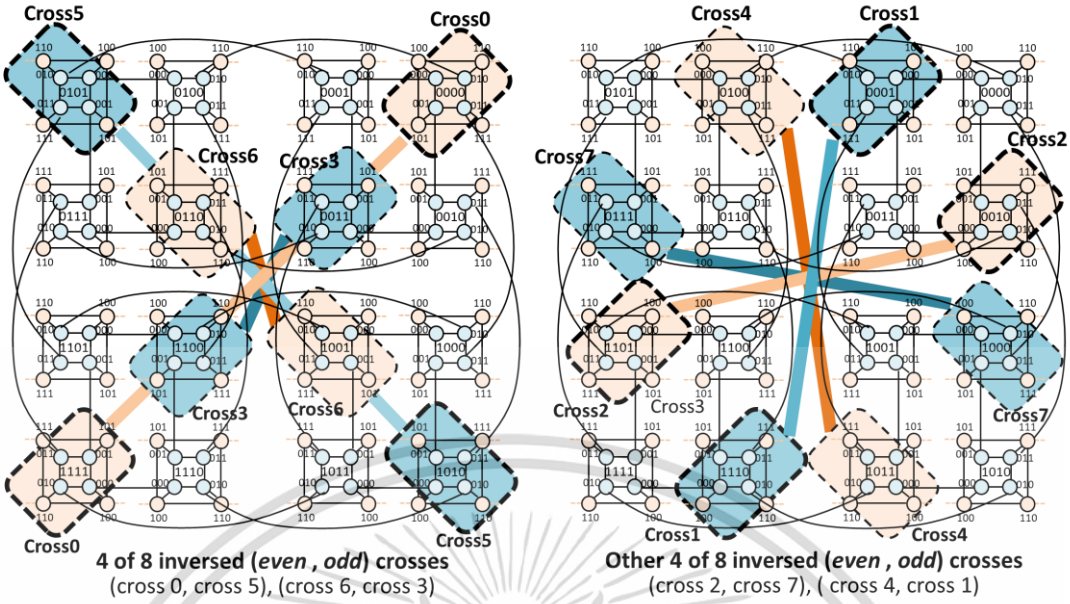
ในการทดลองเพื่อประเมินประสิทธิภาพของการสื่อสารแลกเปลี่ยนข้อมูลแบบ ATAPE ถูกทดสอบโดยการเขียนโปรแกรมด้วยภาษาจาวา (JAVA Programming Language) และถูกทดสอบด้วยงานที่มีขนาด $k = 2^{m+1}$ ในการจำลองเครือข่าย HHC กำหนดให้การสื่อสารระหว่างโหนด *A* ไปยังโหนด *B* ใช้เวลา 1 สัญญาณนาฬิกา (Clock) และทุกๆคู่ของโหนด *A* และ *B* สามารถส่งข้อมูลผ่านเส้นเชื่อมเดียวกันได้ ถ้าไม่ได้อยู่ในเวลาสัญญาณนาฬิกาเดียวกัน หรือ เป็นทิศทางที่ตรงกันข้ามกัน



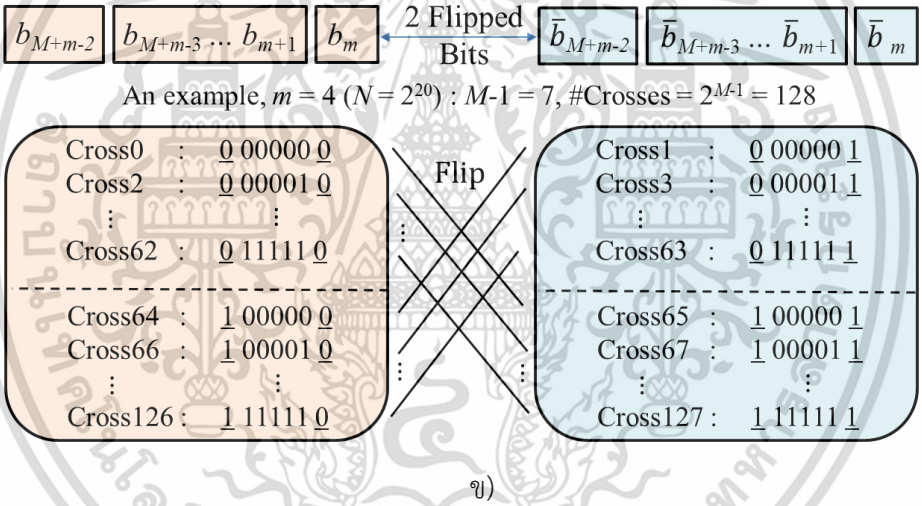
รูปที่ 4.1 ตัวอย่างเส้นทางการสื่อสารแบบขนานของ $Cross_0$ (ก) และ $Cross_1$ (ข) บนเครือข่าย 6-HHC เมื่อ k = 8



รูปที่ 4.2 ตัวอย่างกลุ่มของการสื่อสารแบบขนานของ Cross เลขคู่ และเลขคี่ บนเครือข่าย 6-HHC เมื่อ k = 16 และ 32



ก)



ข)

รูปที่ 4.3 ตัวอย่างกลุ่มของการสื่อสารแบบขนานของ Cross เลขคู่ และเลขคี่

ก) เมื่อ m = 3 และ ข) เมื่อ m = 4

ตารางที่ 4.1 แสดงตัวอย่างของเส้นทาง และการจัดการด้านเวลาของการสื่อสารแบบ ATAPE บนเครือข่าย HHC เมื่อ m = 2 และ C = 5 หรือได้แสดงเส้นทางการสื่อสารในรูปที่ 4.1 โดยการติดต่อสื่อสารเริ่มจาก ทุกต้นทาง S = [0, 1, 2, 3, 12, 13, 14, 15] ถึงทุกปลายทาง D = [13, 12, 15, 14, 1, 0, 3] จากตารางจะเห็นว่า ในสัญญาณนาฬิกาที่ 1 – 5 ไม่มีเส้นทางจาก $u \rightarrow v$ ใดๆ ที่มีต้นทางและปลายทางเหมือนกัน ดั้งนั้นวิธีนี้จะเป็นการพิสูจน์ว่าในแต่ละตัวควบคุม (C) จะไม่มีการขัดแย้งกันระหว่างสื่อสารเพื่อแลกเปลี่ยนข้อมูลแบบ ATAPE

ตารางที่ 4.1 เส้นทางการสื่อสารระหว่าง *S* และ *D* ผ่าน 5 สัญญาณนาฬิกา (Clock) และ *C*=5 ของ 2 Cross เมื่อ *k* = 8

Clock		1		2		3		4		5	
C=5	*S* = 0:0000,00	→	0001,00	→	0001,01	→	13:0011,01=*D*				
	S = 1:0000,01	→	0010,01	→	0010,00	→	12:0011,00=*D*				
	S = 2:0000,10	→	0000,00	→	0001,00	→	0001,01	→	0011,01	→	15:0011,11=*D*
	S = 3:0000,11	→	0000,01	→	0010,01	→	0010,00	→	0011,00	→	14:0011,10=*D*
	S = 12:0011,00	→	0010,00	→	0010,01	→	1:0000,01=*D*				
	S = 13:0011,01	→	0001,01	→	0001,00	→	0:0000,00=*D*				
	S = 14:0011,10	→	0011,00	→	0010,00	→	0010,01	→	0000,01	→	3:0000,11=*D*
	S = 15:0011,11	→	0011,01	→	0001,01	→	0001,00	→	0000,00	→	2:0000,10= *D*

Clock		1		2		3		4		5	
C=5	*S*=4:0001,00	→	0000,00	→	0000,01	→	9:0010,01=*D*				
	S=5:0001,01	→	0011,01	→	0011,00	→	8:0010,00=*D*				
	S =6:0001,10	→	0001,00	→	0000,00	→	0000,01	→	0010,01	→	11:0010,11=*D*
	S =7:0001,11	→	0001,01	→	0011,01	→	0011,00	→	0010,00	→	10:0010,10=*D*
	S =8:0010,00	→	0011,00	→	0011,01	→	5:0001,01=*D*				
	S =9:0010,01	→	0000,01	→	0000,00	→	4:0001,00=*D*				
	S =10:0010,10	→	0010,00	→	0011,00	→	0011,01	→	0001,01	→	7:0001,11=*D*
	S =11:0010,11	→	0010,01	→	0000,01	→	0000,00	→	0001,00	→	6:0001,10=*D*

4.4 การพิสูจน์ความถูกต้องของการจัดสรรงานบนโครงสร้างต้นไม้แบบทวิภาค ด้วยการรวมกลุ่มแบบ GCD และ GCS

ในหัวข้อนี้จะพิสูจน์ความถูกต้องในการจัดสรรงานจากตารางงานลงบนโครงสร้างต้นไม้แบบทวิภาค โดยหน่วยประมวลผลที่ถูกจัดสรรจะสามารถแบ่งได้ 3 รูปแบบ ตามชั้น (level) ของต้นไม้ที่หน่วยประมวลผลที่ถูกจัดสรร ประกอบด้วย

1. ชั้น *cross*-node ที่สามารถจัดสรรหน่วยประมวลผลขนาด $k = 2^{m+1}$
2. ชั้น *cube*-node ที่สามารถจัดสรรหน่วยประมวลผลขนาด $k = 2^m$
3. ชั้น *processor*-node ที่สามารถจัดสรรหน่วยประมวลผลขนาด $k = 1$

เนื่องจากโครงสร้างของเครือข่าย HHC เป็นโครงสร้างที่มีโครงสร้างพื้นฐานมาจากเครือข่าย HC ดังนั้น การจัดสรรงานลงบนเครือข่าย HHC จะคล้ายกับการจัดสรรบนเครือข่าย HC โดยโครงสร้างต้นไม้แบบทวิภาคสำหรับเครือข่าย HHC จะเริ่มที่โหนดราก ถูกแทนด้วย xx...xcc...c จนถึงชั้น *group*-level จะเป็นการแบ่งกลุ่มบนโครงสร้างต้นไม้แบบปรกติ คือ ลูกทางซ้าย (Left-child: L)

= 0 และลูกทางขวา (Right-child : R) = 1 เช่น สองโหนดในชั้นลูกของโหนดราก (l = 1) จะมีลูกทางซ้าย ถูกแทนด้วย 0x...xcc...c และลูกทางขวาถูกแทนด้วย 1x...xcc...c

ความถูกต้องของการจัดสรรงานที่ถูกรวมกลุ่มบนเครือข่าย HHC สามารถพิสูจน์ได้โดยการแบ่งเป็น 2 กรณีดังนี้

กรณีที่ 1: การจัดสรรงานลงบนคิวบ์ และการรวมคิวบ์ บนเครือข่าย HC ($N = 2^n$)

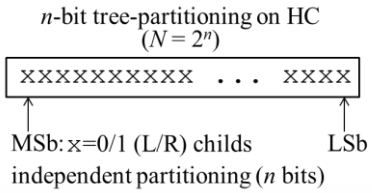
การแบ่งอินเด็กซ์ระดับ n-บิตของเครือข่าย HC แสดงในรูปที่ 4.4 ก) และรูปที่ 4.4 ข) แสดงการแบ่งอินเด็กซ์ระดับ 2^m-บิตภายนอกของเครือข่าย HHC ตัวอย่างเช่น บน HC (N = 64, n = 6) จะมีโหนดราก คือ xxxxxx จะถูกแบ่งเป็น (0xxxxx, 1xxxxx) ในชั้นที่ l = 1, ..., (000000 - 111111) ในชั้นที่ l = 6 ขณะที่บน n-HHC จะมีการแบ่งอินเด็กซ์ระดับบิตแบบเดียวกับ HC ตั้งแต่โหนดรากที่ชั้นที่ l = 0 จนถึงชั้น l = M โดย 2^m บิต จะประกอบด้วยอินเด็กซ์ของกลุ่มจำนวน 2^{m-1} บิต ถูกแทนด้วย xx...x และอีก 2^{m-1} บิต คือ อินเด็กซ์ของ Cross ถูกแทนด้วย cc...c รูปที่ 4.5 ก) แสดงโครงสร้างต้นไม้แบบทวิภาคภายใต้การจัดสรรหน่วยประมวลผลบนเครือข่าย HC ที่มีขนาด $N = 2^n$ = 8 หน่วยประมวลผล โดยโหนดรากถูกแทนด้วย (xxx) ที่ชั้นที่ l = 0, โหนด (0xx, 1xx) ในชั้นที่ l = 1, (00x, 01x, 10x, 11x) ในชั้นที่ l = 2 และ (000, 001, 010, 011, 100, 101, 110, 111) ในชั้นที่ l = 3 และโหนด 0xx ถูกจัดสรรงานที่มีขนาด 4 หน่วยประมวลผล รูปที่ 4.5 ข) แสดงตัวอย่างการรวมคิวบ์ของ 2 คิวบ์ คือ (0xx, 1xx) โดยสามารถรวมกลุ่มกันได้โดยตรงในต้นไม้ในชั้นที่ l = 1 และมี 8 หน่วยประมวลผลที่ถูกจัดสรร สำหรับบนเครือข่าย 6-HHC จะได้โหนดรากถูกแทนด้วย xxcc โดยจะถูกแบ่งเป็น (0xcc, 1xcc) ในชั้นที่ l = 1, (00cc, 01cc, 10cc, 11cc) ในชั้นที่ l = 2 หรือใน *group*-level และในชั้นนี้แต่ละกลุ่มจะสื่อสารกันอย่างอิสระไม่มีการขัดแย้งกันระหว่ากลุ่ม ในแต่ละกลุ่มจะมี Cross จำนวน 2^{M-1} cross (2 คิวบ์ต่อCross) และจะถูกจัดสรรลงบน *cross*-level รูปที่ 4.7 ก) แสดงหนึ่ง Cross ที่มีขนาด C0: (0000, 0011) โดยใช้สมการที่ 3.4 ที่มีค่าโหนดเริ่มต้นภายนอก e = 0000

กรณีที่ 2: การจัดสรรงานลงบนการรวมกลุ่มของหน่วยประมวลผลแบบ GCS บนเครือข่าย HHC

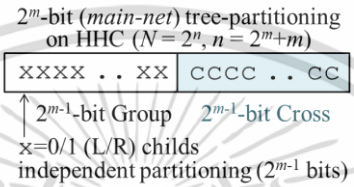
การรวมกลุ่มแบบ GCS โดยใช้สมการที่ 3.5 – 3.9 จะได้การจัดสรรงานสู่หน่วยประมวลผลในชั้น *cross*-level หรือ l= 2^m-1 โดยที่ขนาดของ GCS คือ $2^{m+2} \le k \le 2^n/2^{M-1}$ ตัวอย่างเช่น รูปที่ 4.6 ข) แสดงการรวมกลุ่มแบบ GCS ที่มีขนาด $k = 2^{m+2}$ โดยประกอบด้วยสอง Cross จากการแบ่งกลุ่มแบบ GCD สองกลุ่ม คือ GCD_0 (0<u>0</u>cc) และ GCD_1 (0<u>1</u>cc) โดยใช้สมการที่ 3.5 ที่มีค่าโหนดเริ่มต้นคือ e = 0000 ดังนั้นจะได้ GCS ที่ประกอบด้วย Cross ที่ 1 คือ C0(00<u>00</u>, 00<u>11</u>) ใน GCD_0 และ Cross ที่ 2 คือ C1(01<u>01</u>, 01<u>10</u>) ใน GCD_1

รูปที่ 4.6 ค) แสดงการรวมกลุ่มแบบซูเปอร์-GCS ด้วยขั้นตอนวิธีที่ 3.9 ที่มีขนาด $k = 2^{m+2}$ หรือประกอบด้วย 4 Cross จาก 4 กลุ่ม (GCD_0-GCD_3) คือ <u>00</u>cc, <u>01</u>cc, <u>10</u>cc และ <u>11</u>cc ที่มีค่าโหนดเริ่มต้น คือ e = 0000 และจำนวน Cross ที่หนึ่งคือ C0(00<u>00</u>, 00<u>11</u>) ในกลุ่ม GCD_0, Cross ที่

สองคือ C1(0101, 0110) ในกลุ่ม GCD_1, Cross ที่สามคือ C1(1010, 1001) ในกลุ่ม GCD_2 และ Cross ที่สี่คือ C0(1100, 1111) ในกลุ่ม GCD_3 ตามลำดับ

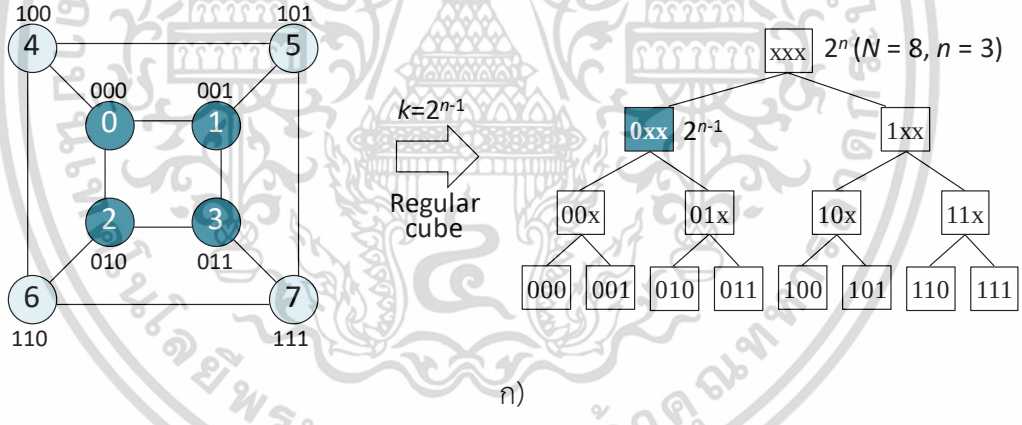


ก)

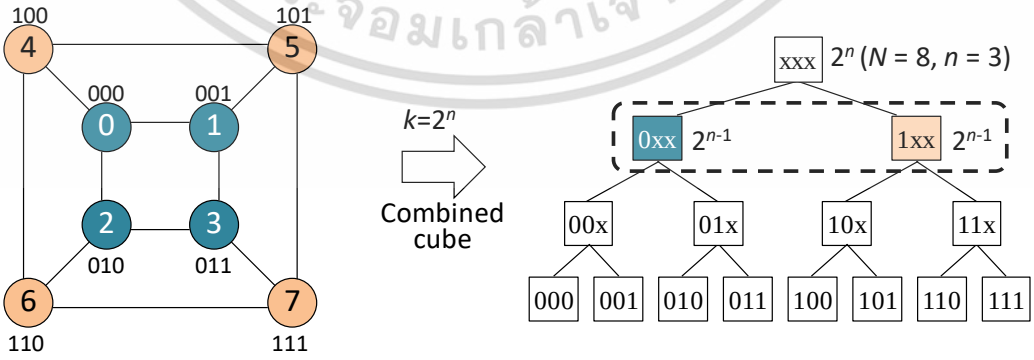


ข)

รูปที่ 4.4 การแบ่งอินเด็กซ์ระดับบิตของ ก) เครือข่าย HC และ ข) เครือข่าย HHC

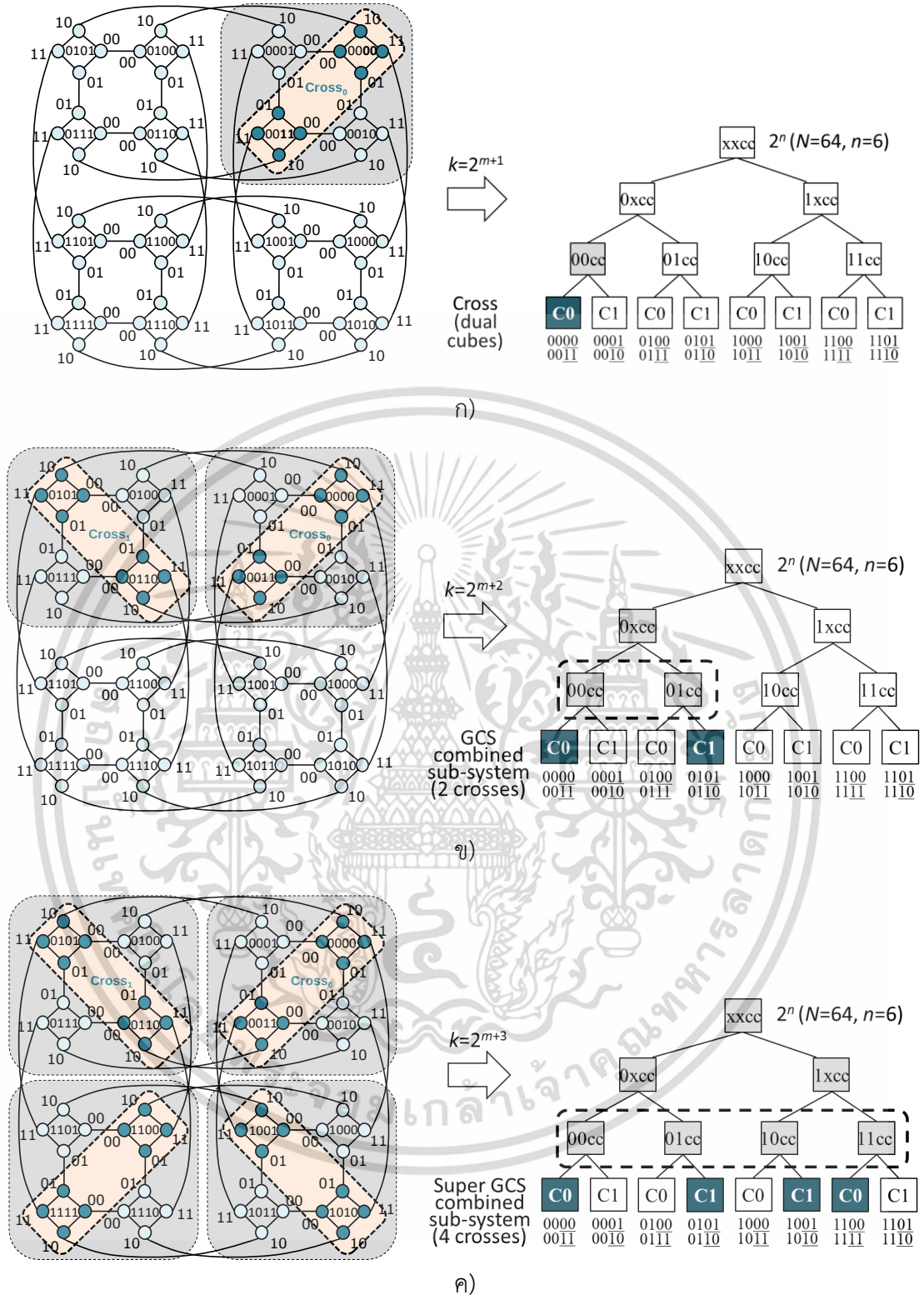


ก)



ข)

รูปที่ 4.5 ตัวอย่างการแบ่งกลุ่มบนเครือข่าย HC (*N*=8, *n*=3) ก) หนึ่งโหนด (0xx) บน 2-HC ข) การรวมสองโหนด 2-HCs (0xx, 1xx)



รูปที่ 4.6 ตัวอย่างของการจัดสรรหน่วยประมวลผลขนาด $k \geq 2^{m+1}$ บนเครือข่าย 6-HHC

ก) GCD ขนาด $k = 2^{m+1}$ ข) GCS ขนาด $k = 2^{m+2}$ และ ค) Super-GCS ขนาด $k = 2^{m+3}$

4.5 การวิเคราะห์ความซับซ้อนทางด้านเวลา (Time Complexity Analysis)

ภายใต้การแบ่งกลุ่มแบบ GCD และ GCS เพื่อการจัดสรรหน่วยประมวลผลบนโครงสร้างต้นไม้แบบทวิภาค ซึ่งแต่ละงานจะถูกจัดสรรสู่หน่วยประมวลผลในชั้น *cross*-node ($l = 2^{m-1}$) จน ถึงชั้น *processor*-node ($l = n$) และความซับซ้อนทางด้านเวลาของการจัดสรร/ยกเลิกจัดสรรหน่วยประมวลผล แสดงในตารางที่ 4.2 ซึ่งถูกวิเคราะห์และอธิบายได้ดังนี้

<u>สำหรับการจัดสรรหน่วยประมวลผล (Processor Allocation)</u>

- ความซับซ้อนทางด้านเวลาของการท่องไปในต้นไม้แบบทวิภาค ด้วยวิธี DBS สำหรับงานที่ร้องขอหน่วยประมวลผลขนาด k ด้วยกรณีที่ดีที่สุด (Best Case) คือ $O(2^m)$ สำหรับ $k = 2^{m+1}$ และสำหรับกรณีที่แย่ที่สุด (Worst Case) คือ $O(2^m+m)$ สำหรับ $k \le 2^m$ หรือ *cube*-node ดังนั้นชัดเจนว่าสำหรับกรณีเฉลี่ย (Average Case) คือ $O(2^m)$ สำหรับ $2^{m+1} < k \le 2^n/2^{M-1}$
- ความซับซ้อนทางด้านเวลาของการหาโหนดเริ่มต้น e คือ $O(2^m)$ ด้วยกรณีที่ดีที่สุด สำหรับ *cross*-node และสำหรับกรณีที่แย่ที่สุดคือ $O(2^M)$ สำหรับการรวมกลุ่มแบบ GCS ที่ประกอบด้วยจำนวนเครือข่ายชั้นนอกมากที่สุด $= (2^n/2^{M-1})/2^m \approx 2^M$ เมื่อ $M = 2^{m-1}$
- ความซับซ้อนทางด้านเวลาของการปรับปรุงค่า *sum* คือ $O(2^m)$ ด้วยกรณีที่ดีที่สุด สำหรับกรณีที่แย่ที่สุดคือ $O(2^M)$ ซึ่งจะทำการปรับปรุงจาก *cross*-node ถึง โหนดราก $\approx 2^M + 2^{M-1} + \ldots + 2^1 + 2^0 = 2^{M+1}$

<u>สำหรับการยกเลิกจัดสรรหน่วยประมวลผล (Processor Deallocation)</u>

- ขั้นแรก คือ การค้นหาอินเด็กซ์ของงานที่จะยกเลิกในอาเรย์ลิสต์ ในตำแหน่งที่ $i = n\text{-}\log_2 k$ ด้วย $O(1)$ สำหรับกรณีที่ดีที่สุด ซึ่งคือ กรณีที่งานที่จะยกเลิกอยู่โหนดแรกของลิสต์ i และสำหรับกรณีที่แย่ที่สุด คือ กรณีที่งานที่จะยกเลิกอยู่โหนดสุดท้ายของลิสต์ n ด้วย $O(N)$
- ต่อมาคือการคืนโหนด e และโหนดอื่นๆ (ถ้ามี) จากรูปแบบ p ของโหนด e เมื่องาน T ประมวลผลเสร็จสิ้น ภายในเวลา $O(1)$ สำหรับกรณีที่ดีที่สุด และสำหรับกรณีที่แย่ที่สุด $O(2^M)$
- ความซับซ้อนทางด้านเวลาของการปรับปรุงค่า *sum* คือ $O(2^m)$ ด้วยกรณีที่ดีที่สุด สำหรับกรณีที่แย่ที่สุดคือ $O(2^M)$ ซึ่งจะทำการปรับปรุงจาก *cross*-node ถึงโหนดราก $\approx 2^M + 2^{M-1} + \ldots + 2^1 + 2^0 = 2^{M+1}$

ดังนั้นความซับซ้อนทางด้านเวลาของการจัดสรรงานกรณีที่ดีที่สุด คือ $O(2^m)$ และสำหรับกรณีที่แย่ที่สุด คือ $O(2^M)$ และความซับซ้อนทางด้านเวลาของการยกเลิกการจัดสรรงานกรณีที่ดีที่สุด คือ $O(2^m)$ และสำหรับกรณีที่แย่ที่สุด คือ $O(N)$ <u>หมายเหตุ</u> $N = 2^n$, $n = 2^m+m$ และ $M = 2^{m-1}$

ตารางที่ 4.2 ความซับซ้อนทางด้านเวลาของการจัดสรร และยกเลิกจัดสรรหน่วยประมวลผล

Time Complexity		Best Case	Worst Case
(Best fit) Allocation	- depth best search (DBS) for the request *k*	$O(2^m)$	$O(2^m+m) = O(n)$
	- allocate nodes by *e* and *p*	$O(2^m)$	$O(2^M)$
	- update accumulated *sum*	$O(2^m)$	$O(2^M)$
Total		$O(2^m)$	$O(2^M)$
Deallocation	- search in list for T_{finish}	O(1)	O(*N*)
	- free nodes in tree	O(1)	$O(2^M)$
	- update accumulated *sum*	$O(2^m)$	$O(2^M)$
Total		$O(2^m)$	O(*N*)

4.6 การประเมินประสิทธิภาพ และการเปรียบเทียบผลการวิจัย

ในด้านการเปรียบเทียบประสิทธิภาพการวิจัยที่นำเสนอ กับวิธีการอื่นๆที่มีอยู่ได้ถูกแสดงในตารางที่ 4.3 สำหรับการหาเส้นทางสื่อสารแบบหนึ่งต่อหนึ่ง งานวิจัยนี้ได้นำเสนอโดยการเริ่มต้นที่ต้นทางใดๆ (Arbitrary Source) ขณะที่วิธีการอื่น ได้ผลกับเฉพาะกรณีที่ต้นทางเริ่มต้นที่ 0 เท่านั้น [22] ต่อมาสำหรับการสื่อสารแบบขนานแบบหนึ่งต่อหนึ่ง ที่มีขนาดของหน่วยประมวลผลจำนวน $k = 2^m$ ซึ่งมีขนาดของหน่วยประมวลผลจำนวนมากกว่างานวิจัยอื่นๆ ที่ทำการหาเส้นทางที่แตกต่างกันจำนวน $k = \lceil (m+1)/2 \rceil$ เส้นทาง และวิทยานิพนธ์ฉบับนี้เสนอการแบ่งกลุ่มแบบ GCD และ GCS เพื่อรองรับงานที่มีขนาดใหญ่ขึ้น หรือมีขนาดดังสมการต่อไปนี้

$$
\begin{aligned}
k &\le \frac{2^n}{2^{M-1}} \\
&\le 2^{2^m+m} / 2^{2^{m-1}-1} \\
&\le 2^{2^m+m-2^{m-1}-1} \\
&\le 2^{2^m-2^{m-1}+m-1} \\
k &\le 2^A \quad \text{เมื่อ} \quad A = 2^m - 2^{m-1} + m - 1 \qquad (4.1)
\end{aligned}
$$

ซึ่งทำให้แต่ละ Cross ที่อยู่ในแต่ละกลุ่มสามารถสื่อสารแบบ ATAPE ได้อย่างเต็มประสิทธิภาพ นั้นคือสามารถสื่อสารได้พร้อมกันในเวลาเดียวกันและไม่มีความขัดแย้งกัน สุดท้ายวิทยานิพนธ์ฉบับนี้เสนอการจัดสรรงานบนโครงสร้างต้นไม้แบบทวิภาคที่ดีที่สุด เพื่อรองรับงานที่สามารถแบ่งกลุ่มแบบ GCD และ GCS ที่สามารถนำไปใช้ได้จริง บนเครือข่าย HHC

ตารางที่ 4.3 การเปรียบเทียบประสิทธิภาพของขั้นตอนวิธี

Features	2007 [22]	2013 [4]	This study
N2N routing ($S = 0$)	✓	✓	✓
N2N routing (arbitrary S)	-	✓	✓
Parallel *N2N* routing	-	✓	✓
Routing-conflict solution	✓	✓	✓
Group partitioning	-	-	✓
Maximum task sizes	2	$k \leq \lceil (m+1)/2 \rceil$	$k \leq 2^n/2^{M-1}$
Parallel applications	-	-	ATAPE
Fully system-utilization	-	-	✓
Task-scheduling	-	-	✓

หมายเหตุ เมื่อ $N = 2^n$, $n = 2^m+m$ และ $M = 2^{m-1}$

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

5.1 สรุปผลการวิจัย

ในวิทยานิพนธ์ฉบับนี้เสนอการออกแบบเส้นทางการติดต่อสื่อสารระหว่างหน่วยประมวลผลบนเครือข่าย HHC ซึ่งเส้นทางที่ได้ต้องไม่มีการชนกัน (หรือขัดแย้งกัน) ระหว่างสื่อสาร รวมถึงเส้นทางที่ได้ต้องเป็นเส้นทางที่สั้นที่สุด โดยการหาเส้นทางประกอบด้วยการหาเส้นทางสื่อสารที่สั้นที่สุดแบบหนึ่งต่อหนึ่งบนเครือข่าย HHC ที่ถูกนำเสนอในหัวข้อที่ 3.1 ซึ่งเป็นการหาเส้นทางระหว่างต้นทาง *S* ใดๆ สู่ปลายทาง *D* ใดๆ ที่ประกอบด้วย 2 ขั้นตอนหลักๆ คือ 1. การหาเส้นทางที่สั้นที่สุดของโหนดชั้นนอก (External Node) และ 2. การจัดลำดับใหม่เพื่อให้ได้เส้นทางที่สั้นที่สุด โดยอธิบายขั้นตอนวิธีไว้ในขั้นตอนวิธีที่ 3.2 ซึ่งเป็นขั้นตอนการจัดลำดับบิตใหม่แบบคงที่ (Static SP-Ordering) ซึ่งผลลัพธ์เส้นทางที่ได้จากขั้นตอนวิธีนี้ได้เส้นทางที่สั้นที่สุดแบบหนึ่งต่อหนึ่ง แต่ยังไม่เหมาะสมเมื่อต้องการสื่อสารแบบขนานที่ประกอบด้วย ต้นทาง *S* และ ปลายทาง *D* หลายๆ คู่ ด้วยเหตุนี้วิธีการหาเส้นทางที่สั้นที่สุดแบบขนานบนเครือข่าย HHC จึงถูกเสนอในหัวข้อที่ 3.2 เพื่อแก้ปัญหาการชนกันดังกล่าวโดยอธิบายด้วยขั้นตอนวิธีที่ 3.3 ซึ่งเป็นการปรับปรุงขั้นตอนวิธีมาจากขั้นตอนวิธีที่ 3.2 โดยจะลดกรณีลงจาก 5 กรณี เหลือ 2 กรณี และเสนอการเรียงลำดับบิตแบบพลวัต (Dynamic SP-Ordering) ที่ประกอบด้วยการจัดลำดับบิตไปข้างหน้าแบบวนกลับ (Forward Routing) และการเรียงลำดับบิตไปข้างหลังแบบวนกลับ (Backward Routing) เพื่อให้ได้มาซึ่งเส้นทางที่เป็นเส้นทางที่สมมาตรกัน และรองรับการสื่อสารแบบขนานที่มีหลายกลุ่ม

จัดกลุ่มแบบดูอัล-คิวบ์แบบไขว้ (Grouping of Cross Dual-cubes : GCD) ถูกเสนอขึ้น เพื่อจัดกลุ่มของหน่วยประมวลผลให้อยู่ในกลุ่มที่ถูกต้องเหมาะสม เนื่องจากการสื่อสารออกนอกกลุ่มจะควบคุมการชนกันระหว่างสื่อสารได้ยาก โดยเครือข่าย *n*-HHC จะประกอบด้วยจำนวน GCD ทั้งหมด $2^{2^{m-1}}$ กลุ่ม ภายในแต่ละ GCD จะประกอบด้วยกลุ่มย่อย ที่เรียกว่าดูอัล-คิวบ์แบบไขว้ (Cross of Dual-cube : Cross) ขนาด $k = 2^{m+1}$ ซึ่งจำนวนหน่วยประมวลผลทั้งหมดใน Cross สามารถสื่อสารกันได้อย่างอิสระและเป็นเส้นทางที่สั้นที่สุด สำหรับในสถานการณ์ที่ระบบต้องการหน่วยประมวลผลมากขึ้น วิทยานิพนธ์ฉบับนี้เสนอการรวมกลุ่มที่รองรับหน่วยประมวลผลขนาด $2^{m+1} < k \leq 2^n/2^{M-1}$ ซึ่งถูกเรียกว่าการรวมกลุ่มคิวบ์ย่อยแบบไขว้ (Grouping of Cross Sub-cubes : GCS) ซึ่งนอกจากจะสามารถรองรับขนาดของหน่วยประมวลผลที่เพิ่มขึ้นอย่างยอดเยี่ยมแล้ว แต่ละกลุ่มคิวบ์ย่อยสามารถสื่อสารโดยไม่ขัดแย้งกัน ถ้าแต่ละกลุ่มนั้นเริ่มประมวลผลพร้อมกัน ขณะที่ในกลุ่มอื่นๆ สามารเริ่มประมวลผลได้อย่างอิสระ นอกจากนี้งานวิทยานิพนธ์นี้เสนอการจัดสรรหน่วยประมวลผลบนโครงสร้างต้นไม้แบบทวิภาค เพื่อการจัดสรรหน่วยประมวลผลที่รวมกลุ่มแบบ GCD และ GCS โดย

ประกอบด้วยขั้นตอนการจัดสรรแบบเฟิร์สฟิต และเบสท์ฟิต ซึ่งมีความซับซ้อนทางด้านเวลาสำหรับการจัดสรร/ยกเลิกจัดสรร คือ O(*N*)

สุดท้ายนี้วิทยานิพนธ์ฉบับนี้ได้เสนอฟังก์ชัน $D_j^{C_i} = d_{i,j} = S_{(C_i \oplus j)}$ เพื่อกำหนดรูปแบบการสื่อสารแบบขนานที่เรียกว่า ATAPE ซึ่งวิทยานิพนธ์ฉบับนี้จะนำการสื่อสารแบบ ATAPE มาเพื่อทดสอบการหาเส้นทางรับส่งข้อมูลของหน่วยประมวลผล และการแบ่งกลุ่มแบบ GCD และ GCS ที่ได้นำเสนอ ด้วยขนาดของหน่วยประมวลผล คือ $k \leq 2^n/2^{M-1}$ เมื่อ $M = 2^{m-1}$ ดังนั้น ภายใน GCD และ GCS จะประกอบด้วย Cross ซึ่งแต่ละ Cross สามารถสื่อสารแบบ ATAPE กันได้อย่างอิสระด้วยเส้นทางที่สั้นที่สุด โดยการกำหนดให้ Cross เลขคู่ ใช้การหาเส้นทางโดยเรียงลำดับบิตไปข้างหน้าแบบวนรอบและ Cross เลขคี่ ใช้การหาเส้นทางโดยเรียงลำดับบิตถอยกลับแบบวนรอบ ดังนั้นขั้นตอนนี้จะทำให้การรับส่งข้อมูลแบบ ATAPE บนเครือข่าย HHC จะสามารถใช้ประโยชน์จากความสมมาตรของเครือข่ายไฮเปอร์คิวบ์แบบชั้นได้อย่างเต็มประสิทธิภาพ 100%

5.2 ข้อเสนอแนะ

- เสนอการสื่อสารเพื่อแลกเปลี่ยนข้อมูลสำหรับกลุ่มคิวบ์ย่อยของหน่วยประมวลผล ให้สื่อสารกันได้โดยไม่ขัดแย้งกัน และไม่จำเป็นต้องเริ่มพร้อมกัน
- เสนอการนำการสื่อสารบนเครือข่าย HHC ไปใช้จริง ด้วยการจัดสรรงาน (Task Scheduling) ที่มีประสิทธิภาพกว่าวิธีที่มีการนำเสนอในวิทยานิพนธ์ฉบับนี้
- เสนอการประยุกต์วิธีรับส่งข้อมูลแบบขนานชนิดอื่นๆ และแอพพลิเคชันอื่นๆ ที่มีประสิทธิภาพบนเครือข่ายไฮเปอร์คิวบ์แบบชั้น

เอกสารอ้างอิง

[1] Abd-El-Barr, M. and Al-Somani, T.F. 2011. “Performance Comparison of Hierarchical Interconnection Networks.” *IEEE Pacific Rim Conference on Communications (PacRim)*. n.d. : 191–196.

[2] Abd-El-Barr, M. and Gebali, F. 2014. “Relibility Analysis and Fault Tolerance for Hypercube Multi-computer Networks.” *Information Sciences.* 276 : 295–318.

[3] Bossard, A. and Kaneko, K. 2011. “Set-to-Set Disjoint-path Routing in Perfect Hierarchical Hypercube.” *The Fourth International C* Conference on Computer Science and Software Engineering*. n.d. : 51–57.

[4] Bossard, A. and Kaneko, K. 2013. “*k*–pairwise Disjoint Paths Routing in Perfect Hierarchical Hypercubes.” *The J. Supercomputing*. 67(2) : 485–495.

[5] Bossard, A. Kaneko, K. and Peng, S. 2011. “Node-to-set Disjoint-path Routing Inperfect Hierarchical Hypercubes.” *International Conference on Computational Science (ICCS 2011)*. 4 : 442–451.

[6] Chen, C. Agrawal, D.P. and Burk, J.R. 1993. “dBCube: a New Class of Hierarchical Multiprocessor Interconnection Networks with Area Efficient Layout.” *IEEE Trans. Parallel and Distributed System*, 4(12) : 1332–1344.

[7] Ghose, K. and Desai, K.R. 1995. “Hierarchical Cubic Networks.” *IEEE Trans. Parallel and Distributed Systems*. 6(4) : 427–435.

[8] Lai, C. 2014. “An Feffcient Construction of One-to-many Node-disjoint Paths in Folded Hypercubes.” *J.Parallel and Distributed Computing*. 74(4) : 2310–2316.

[9] Lai, C. 2012. “Optimal Construction of All Shortest Node-Disjoint Paths in Hypercubes with Applications.” *IEEE Trans. Parallel and Distributed Systems*. 23(6) : 1129–1134.

[10] Lai, C. 2017. “Optimal Construction of Node-Disjoint Shortest Paths in Folded Hypercubes.” *J. Parallel and Distributed Computing.* 102 : 37–41.

[11] Malluhi, Q.M. and Bayoumi, M.A. 1994. “The Hierarchical Hypercube: A New Interconnection Topology for Massively Parallel Systems.” *IEEE Trans. Parallel and Distributed Systems*. 5(1) : 17–30.

[12] Petagon, R. and Werapun, J. 2016. “Embedding the Optimal All-to-all Personalized Exchange on Multistage Interconnection Networks^{+}.” *J. Parallel and Distributed Computing*, 88 : 15–30.

[13] Petagon, R. and Werapun, J. 2017. “VA-DE: Valuable ATAPE with Dynamic Embedding and Super-pipeline Scheduling on Partitionable Multistage Interconnection Network^{+}.” *J. Parallel and Distributed Computing.* 102 : 1–15.

[14] Prisacari, B. Rodriguez, G. and Minkenberg, C. 2013. “Generalized Hierarchical All-to-all Exchange Patterns.” *IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS)*. n.d. : 537–547.

[15] Qiao, B. Shi, F. and Ji, W. 2007. “THIN: A New Hierarchical Interconnection Network-on-Chip for SOC.” *The 7th international conference on Algorithms and architectures for parallel processing*. 7 : 446–457.

[16] Qiu, K. 2008. “An efficient Disjoint Shortest Paths Routing Algorithm for the Hypercube.” *14th IEEE International Conference on Parallel and Distributed Systems (ICPADS’08)*. 14 : 43–47.

[17] Saad, Y. and Schultz, M.H. 1988. “Topological Properties of Hypercubes.” *IEEE Trans. Computer.* 37(7) : 867–872.

[18] Scott, D. 1991. “Efficient All-to-all Communication Patterns in Hypercube and Mesh Topologies.” *Distributed Memory Computing Conference.* 6 : 398-403.

[19] Shi, Y. Hou, Z. and Song, J. 2010. “Hierarchical Interconnection Networks with Folded Hypercubes as Basic Clusters. ” *The Fourth International Conference / Exhibition on High Performance Computing in the Asia-Pacific Region*. 1: 134–137.

[20] Singh, R. D. and Murali, R. 2016. “Hamiltonian Laceability Properties in Hierarchical Hypercube Network.” *International journal of computer application. 6* : 1–10.

[21] Takabatake, T. Kaneko, K. and Ito, H. 1999. “Generalized Hierarchical Completely-connected Networks. ”*Fourth International Symposium on Parallel Architectures, Algorithms, and Networks.* n.d. : 68–73.

[22] Wu, R.Y. Chen, G.H. Kuo, Y.L. and Chang, G.J. 2007. “Node-disjoint Paths in Hierarchical Hypercube Networks.” *Information Sciences*. 117(19) : 4200–4207.

[23] Yang, Y. and Wang, J. 2000. “Optimal All-to-all Personalized Exccchange in Self-routable Multistage Networks.” *IEEE Trans. Parallel and Distributed Systems*. 11(3) : 261–274.

[24] Zhu, Q. Xu, J.M. Hou, X. and Xu, M. 2007. “On Reliability of the Folded Hypercubes.” *Information Sciences*. 177(8) : 1782–1788.

[25] Zhao, S. L. and Hao, R. X. 2019. Reliability Assessment of Hierarchical Hypercube Networks. *IEEE Access*. 7 : 54015–54023.

ภาคผนวก

ภาคผนวก ก

ผลงานวิจัยตีพิมพ์ที่ Journal of Parallel and Distributed Computing

Journal homepage: www.elsevier.com/locate/jpdc

Phisutthangkoon, N. and Werapun, J. 2021. “Shortest-Path Routing for Optimal All-to-All Personalized-Exchange Embedding on Hierarchical Hypercube.” *J. Parallel Distrib. Comput.* 150C : 139 – 154.

Journal of Parallel and Distributed Computing 150 (2021) 139–154

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Shortest-path routing for optimal all-to-all personalized-exchange embedding on hierarchical hypercube networks

Nuntipat Phisutthangkoon *[,1], Jeeraporn Werapun

Computer Science Department, King Mongkut's Institute of Technology, Ladkrabang, Bangkok, 10520, Thailand

ARTICLE INFO

Article history:
Received 31 January 2020
Received in revised form 18 September 2020
Accepted 3 January 2021
Available online 12 January 2021

Keywords:
Hierarchical hypercube (HHC) networks
Parallel shortest-path routing
Gray-code mapping and forward/backward reordering
Grouping of cross dual-cube (GCD) partitioning
ATAPE (all-to-all personalized exchange) embedding

ABSTRACT

Hypercube (HC) networks ($N=2^n$) provide efficient communication for parallel-and-distributed computing (PDC) but the HC-based multi-processor (MP) system is costly and not scalable, while hierarchical hypercube (HHC) networks ($N=2^n$, $n=2^m+m$) are less expensive and more scalable. However, the traditional HHC-routing easily conflicts, especially when executing multiple tasks ($k > 2^m$ nodes-per-task). In the past, the node-disjoint-path routing could be used to avoid the conflict but that reliable (S, D) routing limited S(source)=0. Later, the parallel $N2N$ (node-to-node) disjoint-path was proposed for reliable routing but limited $(m+1)/2$ pairs-of-nodes. Therefore, this study proposes the generalized $N2N$ shortest-path (SP) routing (with arbitrary S) and the parallel $N2N$ SP-routing, based on our hypothesis "the shortest-path routing and the proper HHC-partitioning can avoid the HHC-conflict directly". Next, our innovation and contribution are 1. the GCD (grouping of cross dual-cube) partitioning to solve the HHC-conflict for $k \leq 2^{m+1}$ nodes-per-task, and 2. the SP-ATAPE (all-to-all personalized exchange) embedding on the HHC-MPs. The correctness of the SP-routing was proven and the ATAPE communication was experimented to validate our conflict solution. The ATAPE results confirmed that in any group the GCD mapping could make all tasks ($k = 2^{m+1}$ nodes-per-task), synchronized with the same control, working without the conflict.

1. Introduction

In the past ten years, a number of research studies in the modern interconnection networks [1,14,18] and the efficient reliable routing [2,8,20,22] were proposed in literature. Recently, each of the parallel and distributed systems (PDSs) and high performance computing (HPC) machines contains massive processors, such as the research CM (Connection Machine) [6] with more than 2^{16} processors and the commercial GPU-machines with million or more processors. For the next generation of scalable systems, the designing of the efficient network topologies [10,18] along with the powerful reliable-routing [7,9,15] is the crucial research because the interconnection topology and its effective self-routing, incorporated with the reliability, have extremely influenced the implementation-cost and performance of the PDS and HPC systems.

In fact, there exist two key factors in the interconnection networks required for implementing on the parallel computing machines [10,19]. The first one is the scalability, where this characteristic has directly affected the system cost. The second factor is the efficient self-routing (for the effective communication) as well as the reliability or fault tolerance [22]. In the past, the hypercube (HC) network (with the high node-degree) was the most efficient communication but its construction cost was expensive, and hence it was not scalable. In contrast, the mesh network was the most scalable network and inexpensive but its communication was not efficient (because of the low node-degree). Thus, in order to design the efficient interconnection network for the scalable multi-processor (MP) systems, we need to balance between the communication and the construction-cost, such as the node degree should be high for the efficient communication (like the hypercube network) and the construction-cost should be inexpensive for the scalability (like the mesh network) as much as possible.

One of the interesting interconnection networks that has the low-cost of high-degree as well as a good diameter is the hierarchical network [10,18,19]. The hierarchical network of static interconnection networks consists of many levels, where each level is composed of a basic topology including the hypercube network, the mesh network, or the ring network. Currently, the hierarchical topologies dominate the PDS and HPC systems as well as the recent commercial data-center machines with a variation of level-based communications, designed for many high performance applications, such as the AI (artificial intelligent)

* Corresponding author.
E-mail addresses: 56605012@kmitl.ac.th (N. Phisutthangkoon), jeeraporn.we@kmitl.ac.th (J. Werapun).

[1] Research grant CW-012-2/2562.

https://doi.org/10.1016/j.jpdc.2021.01.004
0743-7315/

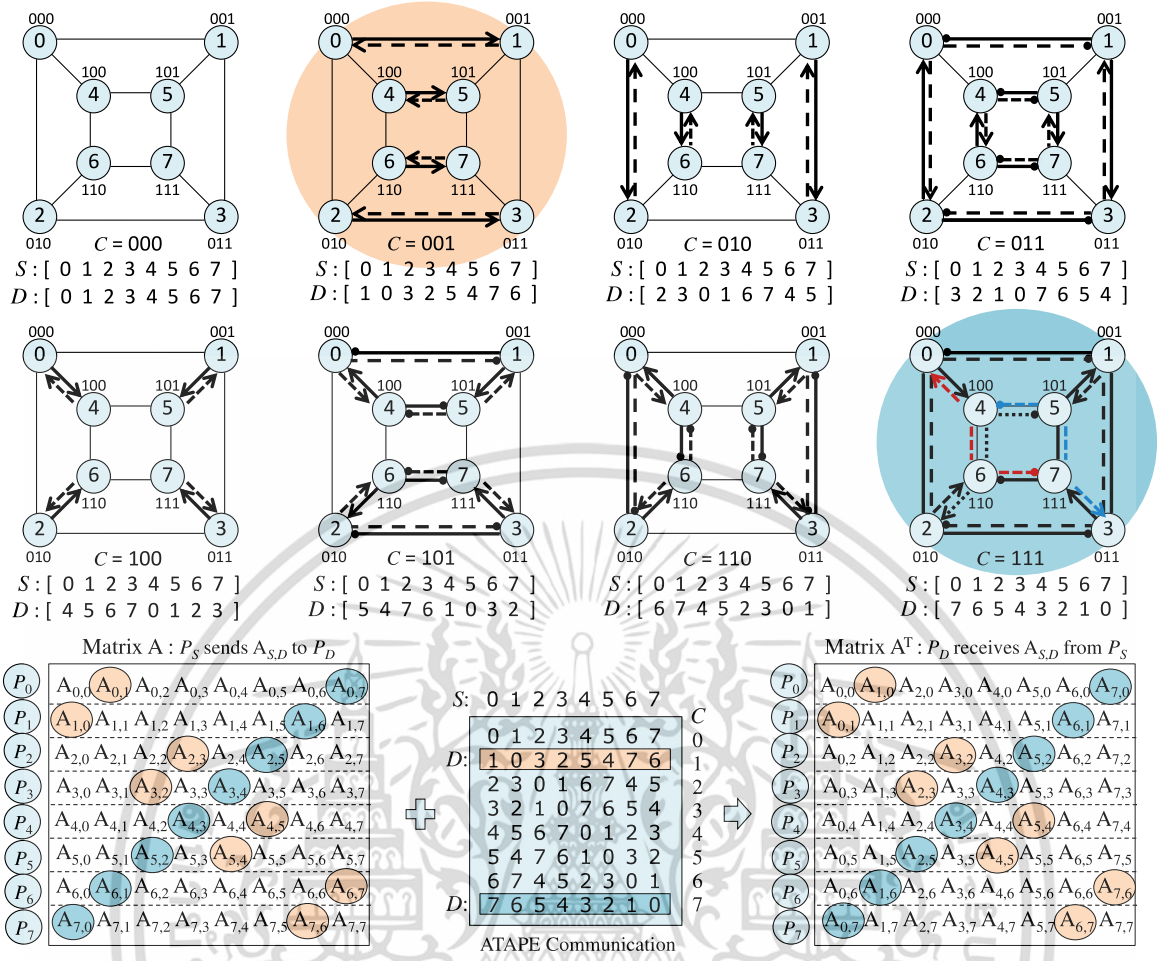


Fig. 1. An example of N permutations of ATAPE communication ($D = S \oplus C$) for a matrix A^T on a hypercube network ($N = 8$).

computing, the quantum-computing, and the data-science applications in the big-data era. However, some of the hierarchical networks cannot perform the flexible self-routing, and hence their routing may easily conflict.

Therefore, in this research we are interested to fulfill the self-routing of a popular hierarchical network, namely the hierarchical hypercube (HHC) network ($N = 2^n$, $n = 2^m + m$, $m \geq 2$), especially to solve the HHC conflict, for a new track of the upcoming scalable MP-systems.

The hypercube (HC) network ($N = 2^n$) [16] is one of the most efficient communication networks for parallel algorithms and HPC applications. However, the HC network is not scalable for implementing the multi-processor (MP) systems because of its expensive cost for large N.

The hierarchical hypercube (HHC) network ($N = 2^n$, $n = 2^m + m$, $m \geq 2$) [10] was proposed to achieve the advantages in the efficient communication and the symmetrical topology like the HC network, while being able to connect many significant numbers of nodes with the lower connection cost. That interesting HHC network is more suitable for the scalable MPs than the HC network. However, the traditional HHC routing [10] easily conflicts, especially when executing multiple tasks ($k > 2^m$ nodes per task). The HHC will be one of valuable networks for the scalable MP systems if the HHC-conflict is solved properly. In 2007, the node-disjoint-path routing [20] from S (source) to D (destination) was proposed for reliable routing ($S = 0$) and it could be used to avoid the HHC conflict. Later in 2013, the k-pairwise disjoint-path routing [4] was presented for reliable parallel-routing on the HHCs but it limited at most $\lceil (m+1)/2 \rceil$ pairs of nodes. However, the HHC conflict should be solved systematically for the parallel (S, D) self-routing before adding the reliability and basically it must work for the ATAPE (all-to-all personalized exchange) communication.

The ATAPE is one of the important parallel tasks that is the most solid operation and repeatedly applied in many applications (such as parallel FFT (fast-Fourier-transformation), parallel A^T (matrix-transposition), etc.). The ATAPE is an efficient communication for sending distinct messages to other processors in N rounds on the MP system ($N = 2^n$). For example, Fig. 1 shows the optimal ATAPE communication ($D = S \oplus C$) [17] for the optimal matrix A^T in $O(N+\log_2 N)$ on the HC networks. In 2016, the research of embedding the ATAPE on-chip [11] was proposed for the optimal communication on multi-stage networks. However, the existing ATAPE functions [11–13,17] cannot work directly on the HHC-MP systems.

While the HHC network is scalable, the basic HHC-routing easily conflicts since the cost reduction of the HHC (via the hierarchical reduction of edges) causes the limitation of routing paths, especially when $k > 2^m$ nodes. Thus, the HHC networks will be perfect for the scalable MP-systems if the HHC conflict is solved properly.

In this study, first we propose the arbitrary S shortest-path (SP) routing form S (source) to D (destination) and the parallel shortest-path (S, D)-routing on the HHC networks, based on our hypothesis "the shortest-path routing and the proper HHC-partitioning can reduce the conflict on the HHC-MPs directly". Our key contribution is the grouping of cross dual-cube (GCD) partitioning, proposed to support multi-tasks with $k = 2^{m+1}$ nodes (per task). In our GCD solution, the system conflict is

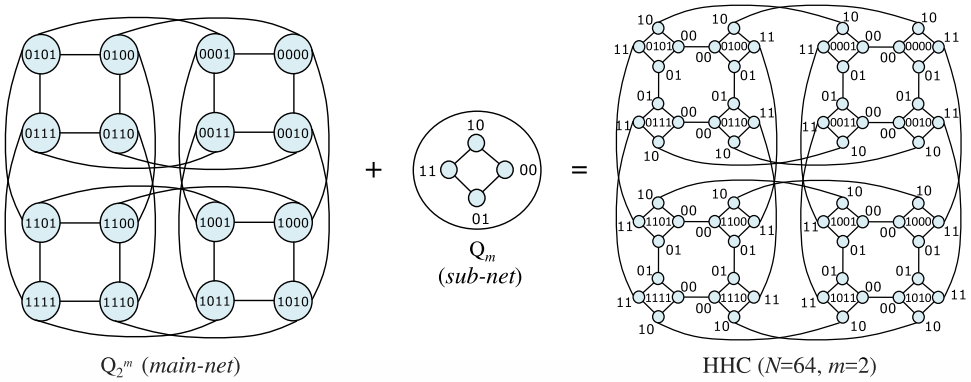


Fig. 2. A structure of the 6-HHC network ($N = 2^n = 64, m = 2, n = 2^m+m = 6$) from the *main-net* Q_{2^m} and the *sub-net* Q_m.

reduced to solve in any group (2^{M-1} crosses per group, where $M = 2^{m-1}$). Under GCD partitioning, all (even, odd) crosses (in any group) are allowed, incorporated with the (forward, backward) protocol for full bidirectional-routing. In our analysis, the correctness of the shortest paths is proven. In our experiment, the ATAPE communication is used to evaluate our HHC-conflict solution. The ATAPE results must confirm that the GCD mapping could make all tasks ($k \leq 2^{m+1}$ nodes per task) in any group, synchronized with the same control, working without the conflict.

The rest of this paper is organized as follows: Section 2 demonstrates the structure and important features of the HHC networks, the disjoint paths for reliable routing on the HHCs, and the existing ATAPE (all-to-all personalized exchange) functions. Section 3 proposes the generalized *N*2*N* shortest-path routing (arbitrary *S*) for the reliable routing on the HHC systems. Section 4 presents the efficient parallel *N*2*N* SP-routing, incorporated with the proper GCD partitioning, and the shortest-path ATAPE embedding for $k \leq 2^{m+1}$ nodes (per task) on the HHC-MPs. In Section 5, the correctness of the parallel *N*2*N* SP-routing is proven and in experiment the ATAPE application is used to verify our conflict solution. Finally, Section 6 discusses the conclusion and our future study.

2. Related work

The structures and features of the hierarchical hypercube (HHC) networks were reviewed in Section 2.1. The node-disjoint-path reliable-routing (to avoid the conflict) on the HHC networks was addressed in Section 2.2. Finally, the existing ATAPE (all-to-all personalized exchange) functions were demonstrated in Section 2.3.

2.1. Hierarchical hypercube (HHC) networks

Malluhi and Bayoumi [10] primarily proposed the hierarchical hypercube (HHC) networks in 1994. In that research, the new HHC topology could connect a significant number of nodes, while maintaining a low cost and a small diameter. In order to utilize the HHC networks efficiently, key properties of the HC and HHC networks were explored in Definitions 1–3.

Definition 1. The *n*-hypercube (HC) or Q_n network [16] consists of 2^n nodes ($N = 2^n$) and each node is represented by a binary sequence of *n* bits ($b_{n-1}\, b_{n-2} \ldots b_1\, b_0$). Any two nodes in the Q_n network are adjacent if and only if they differ in exactly one bit position or the Hamming distance (HD) between their positions is equal to 1.

The efficient communication is the key advantage of the HC network but it is expensive when *n* increases, and hence it is not scalable for the large multi-processor (MP) systems. The HHC network ($N = 2^n$, $n = 2^m+m$) was introduced to preserve the key properties of the HC with the lower cost that would be suitable for the MP systems.

Definition 2. The (2^m+*m*)-hierarchical hypercube (HHC) networks ($N = 2^n$, $n = 2^m+m$) [10] can be constructed by starting with a 2^m-hypercube (Q_{2^m}), called the *main-net*, and then replaces each node of Q_{2^m} with an *m*-hypercube (Q_m), called the *sub-net*. Therefore, the total number of nodes of the HHC are equal to $2^{2^m} x 2^m = 2^{2^m+m}$ nodes. Each node (or processor) has a unique *n*-bit address of a pair of 2^m bits (in the *main-net*) and *m* bits (in the *sub-net*), represented by $\{(\alpha, \beta) \mid \alpha = b_{n-1}\, b_{n-2} \ldots b_m, \beta = b_{m-1}\, b_{m-2} \ldots b_0\}$, where $b_i \in \{0,1\}$, $0 \leq i \leq n-1$, n =$2^m + m$, and $m \geq 2$.

Fig. 2 depicts a structure of the 6-HHC network ($N = 64$, $n = 6$, and $m = 2$) [20]. In (*S*, *D*) self-routing on the HHC, addresses of nodes from *S* (source) to *D* (destination) can be identified by $S = (\alpha_S, \beta_S)$ and $D = (\alpha_D, \beta_D)$, where α_S and β_S denote the *main-net* ID and the *sub-net* ID of *S*, and α_D and β_D are those IDs of *D*. In addition, there is an edge (*A*, *B*) between nodes *A* and *B*, if and only if they fall into either one of the following two conditions [10]:

- The first condition, called the internal edge, is $\alpha_S = \alpha_D$ and the Hamming distance of β_S and β_D is equal to 1.
- The second condition, called the external edge, is $\alpha_S = \alpha_D \oplus 2^{\beta_S}$ and $\beta_S = \beta_D$, where $\oplus$ represents the bitwise exclusive-or (XOR) operation.

Definition 3. An *m*-bit Gray-code [10] is a binary representation, where two adjacent node-numbers differ by only one bit. The 1-bit Gray-code is $G_1 = (0, 1)$ and G_m^* is defined as reverse ordering of G_m. Thus, G_m can be obtained by using the iteration in terms of G_{m-1} as ($0G_{m-1}$, $1G_{m-1}^*$). For instance, $G_2 = (\mathbf{0}G_1, \mathbf{1}G_1^*) = (\mathbf{00}, \mathbf{01}, \mathbf{11}, \mathbf{10})$ and $G_3 = (\mathbf{0}G_2, \mathbf{1}G_2^*) = (\mathbf{000}, \mathbf{001}, \mathbf{011}, \mathbf{010}, \mathbf{110}, \mathbf{111}, \mathbf{101}, \mathbf{100})$.

The HHC network was presented with the node-to-node self-routing [10], Algorithm 1. That routing starts from the current node *C* to the destination node *D* by using the Scube-routing (in any *sub-net*) and the *m*-bit Gray-code mapping for the shortest external-path routing (across the *main-nets*). However, for parallel tasks (such as ATAPE) that basic routing may conflict, especially for executing the ATAPE on the sub-system ($k > 2^m$ nodes). Note: The routing conflict means the data collision (i.e., transferring messages through the same edge at a time).

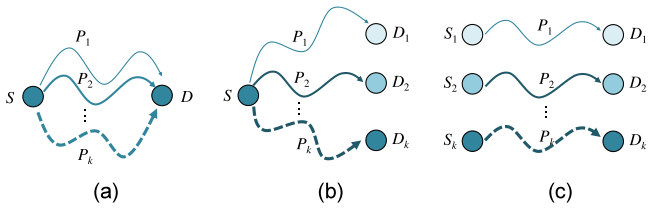


Fig. 3. The disjoint-path (DP) methods: (a) the node-disjoint paths, (b) the node-to-set DPs, and (c) the k -pairwise DPs.

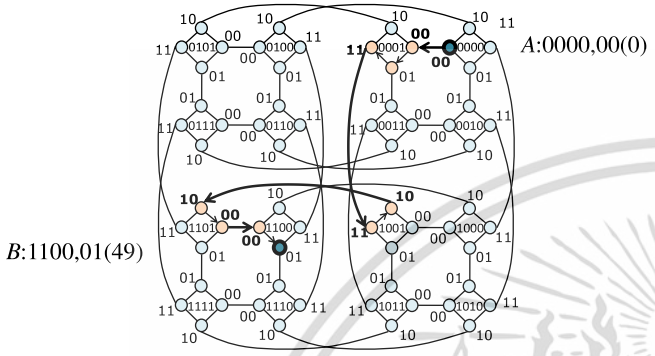


Fig. 4. An example of the HHC routing (Algorithm 2) from a source $A = 0000{,}00\ (0)$ to a destination $B = 1100{,}01\ (49)$.

Algorithm 1: Node–to–node self-routing on HHC networks.

```
Node–to–node Routing
1.  if (C ≠ D) then
2.      if (α_C = α_D) then
3.          Scube–routing(β_C, β_D);
4.      else
5.          T = C ⊕ D;
6.          I = {indices of 1's in α_T};
7.          if (β_C ∈ I) then
8.              routing on external edge;
9.          else
10.             Scube–routing (β_C, β_D);
11. end
```

2.2. Node-Disjoint Paths on HHC Networks

For reliable HHC-routing, many algorithms [3–5,20] were introduced to find the disjoint paths that could be used to avoid the conflict on the HHC networks. The original node-disjoint path routing on the HHCs was proposed in 2007, focusing on the reliable node-to-node ($N2N$) routing. Later in 2011, the reliable one-to-many (or node to set) disjoint-path routing was presented for routing and broadcasting on the HHCs. Lastly in 2013, the k-pairwise disjoint-path routing was introduced for the reliable routing of parallel communication on the HHCs.

2.2.1. The node-disjoint paths

In 2007, Wu et al. [20] introduced the node-disjoint paths for reliable routing on the HHCs, see Fig. 3(a). The idea of that $N2N$ routing, a conflict solution, from the source node $A = (A_{ex}, A_{in})$ to the destination $B = (B_{ex}, B_{in})$ is composed of two main steps. The first step finds a shortest external-path and second step uses that path to construct the corresponding node-disjoint paths (for m+1 paths).

Definition 4. Suppose that A and B are two distinct nodes of the HHC network. An external edge sequence (EES) [20] is a *main-net* path in the HHC network, represented by a sequence of the external edges from A to B.

Algorithm 2 shows the computing of the node-disjoint paths on the HHC networks. The advantage of that work is to find m+1 paths for reliable routing from A to B with a maximum length is less than or equal to $\{2^{m+1}+2m+1,\ 2^{m+1}+m+4\}$, where those paths could be used to avoid the routing conflict on the HHC networks.

Algorithm 2: Node–disjoint paths of HHC routing.

```
Node–disjointPath
1.  Assume A = (0^{2^m}, 0^m);
2.  Finding external edge sequences (EES);
3.  Collect indices i of B_ex (external bits of B) with b_i = 1;
4.  Construct a set π by arranging all indices i into an m–bit
    Gray-code;
5.  Construction 2 of m+1 paths, depending on four cases:
6.      Case 1: A_in ∈π and B_in ∈ π
7.              P_1 = π^{C_0}, P_2 = π^{C_{(z+1) mod r}};
8.      Case 2: A_in ∉ π and B_in∈ π
9.              P_1 = (0^m, π, 0^m), P_2 = π^{C_{(z+1) mod r}};
10.     Case 3: A_in ∈ π and B_in ∉ π
11.             P_1 = (θ^{B_in}, B_in), P_2 = π^{C_0};
12.     Case 4: A_in ∉ π and B_in ∉ π
13.             P_1 = (0^m, π, 0^m), P_2 = (θ^{B_in}, B_in);
14. Other m-1 paths are obtained according to the first m-1 un-
    used EES of π^{C_0}, π^{C_1}, ..., π^{C_{r-1}};
15. Return m +1 paths;
```

Fig. 4 shows an example of routing on the HHC ($N = 64$, $n = 2^m+m = 6$, $m = 2$) from a source node $A = 0$ or $(A_{ex}, A_{in}) = (0000{,}00)$ to the destination $B = 49\ (1100{,}01)$.

Let $\longrightarrow$ denote the external edge and

$\overset{*}{\longrightarrow}$ denote the internal edge(s).

In that case, the external edge sequence (EES) = (11, 10) since there exist 1s in B_{ex} at position 2 and position 3. Clearly, one of the k paths with length = 9 (passing 10 nodes) is P_1 (in Case 4), derived as follows:

$A(0000{,}00) \longrightarrow (0001{,}00) \overset{*}{\longrightarrow} (0001{,}01) \overset{*}{\longrightarrow} (0001{,}11) \longrightarrow (1001{,}11) \overset{*}{\longrightarrow} (1001{,}10) \longrightarrow (1101{,}10) \overset{*}{\longrightarrow} (1101{,}00) \longrightarrow (1100{,}00) \overset{*}{\longrightarrow} B(1100{,}01)$.

However, Algorithm 2 works for a source $A = 0$ only.

2.2.2. The node-to-set disjoint paths

In 2011, Bossard et al. [5] introduced the node-to-set disjoint-path routing on the HHCs (see Fig. 3(b)) for a subset broadcasting. One efficient step of that work focused on reducing the node-to-set problem to the local node-to-set on a sub-cube (2^m). Algorithm 3 briefly describes the main process of four steps to find the node-to-set disjoint paths, where the results are k paths from a source S to k destinations (D_i) and each path length is not greater than $m2^m+2^m+2m+4$ in $O(km2^m)$, where $1 \le i \le k$ and $k \le m+1$.

Algorithm 3: Node–to–set disjoint path on HHC networks.

```
HHC-N2S
1.  if (|D∩Q_m(s_0)| ≥ k-1) then
2.      Solving the N2S routing in a Q_m by Cube-N2S;
3.  else
4.      Step1 "Preprocessing"
5.        Distribute all D to distinct subcubes;
6.      Step2 "Hypercube routing"
7.        Apply Cube-N2S onto Hypercube;
8.      Step3 "Path discarding"
9.        Remove the unnecessary paths created in Step2;
10.     Step4 "Sub-cube routing"
11.       Convert cube-level paths back to HHC-level paths;
```

2.2.3. The k-pairwise disjoint paths

In 2013, Bossard et al. [4] proposed the k-pairwise disjoint paths for reliable routing (see Fig. 3(c)) on the HHCs. Algorithm 4 describes the main functions of finding the k-pairwise (S, D) paths $(0 \le S, D \le N-1)$ for $k = \lceil (m+1)/2 \rceil$ pairs of nodes,

where the length of each path is computed in $O(2^{5m})$ and is not greater than $2^{m+1} + m(2^{m+1}+1) + 4$ but that reliable routing allows $k \leq \lceil (m+1)/2 \rceil$ nodes only.

Algorithm 4: *k*–pairwise disjoint paths on HHC networks.

```
k–pairewise
1. for all external–nodes with |T(σ)| ≥ 2 do
2.       Spreading all nodes of T;
3. P = Generates disjoint–paths in Q_{2^m};
4. for all p ∈ P do
5.       R = Generates disjoint–paths in HHC;
6. return the path R;
```

However, without the proper HHC partitioning, such reliable routing methods [4,5,20] and their conflict solutions might not work properly on the HHC systems.

2.3. All-to-All Personalized Exchange (ATAPE)

All-to-all personalized exchange (ATAPE) is an efficient communication that is constantly applied in many parallel applications (i.e., parallel matrix A^T, etc.) on the MP-systems ($N = 2^n$) and can be used to verify the conflict on the HHC-MPs. In ATAPE processing, every processor must send N personalized messages to others within N rounds. Sections 2.3.1–2.3.3 describe the existing ATAPE functions on the popular networks [11–13,17].

2.3.1. ATAPE on hypercube networks

D. Scott [17] proposed the XOR-based ATAPE communication, see Eq. (2.1), for the efficient parallel (S, D) routing on the hypercube (HC) networks from N sources (S) to N destinations (D), where $0 \leq S, D \leq N-1$ and N rounds of controls $C = 0, 1, 2, \ldots, N-1$.

$$D = S \oplus C \tag{2.1}$$

For example, Fig. 1 illustrates the ATAPE communication and its application for a parallel A^T (matrix transposition) on the HC network ($N = 8$) in $O(N+\log_2 N)$. The main idea of that ATAPE with the Gray-code mapping for the A^T on the HC is performed in N iterations. In each iteration, every node (P_i) exchanges the message with each of other nodes (i.e., in the last step ($C = N-1$), $P_0 \rightarrow P_7$, $P_1 \rightarrow P_6$, $P_2 \rightarrow P_5$, $P_3 \rightarrow P_4$, $P_4 \rightarrow P_3$, $P_5 \rightarrow P_2$, $P_6 \rightarrow P_1$, $P_7 \rightarrow P_0$, etc.).

2.3.2. Hierarchical ATAPE on dragonfly networks

In 2013, B. Prisacari [13] presented a hierarchical ATAPE on the hierarchical dragonfly networks of l-levels, defined generally in Eq. (2.2).

$$h^D = (h^S + h^P)\%N^l \tag{2.2}$$

For example, Fig. 5 shows the 3-level dragonfly network, where the first level (or bottom level) is composed of processors P_i, the second level (or local level) connects to the other routers R_i in the same group, and the third level (or global level) connects to each group G_i for routing to other groups. The ATAPE in Eq. (2.2) was claimed as the generalized ATAPE communication on the dragonfly network, according to its flexible connection.

2.3.3. Embedding ATAPE on multistage networks

In 2000, the ATAPE [21] was introduced on multistage interconnection networks (MINs) in optimal $O(N+\log_2 N)$ by using switch-pattern routing. However, the pattern-routing cannot be embedded on chip. Later in 2016, two efficient ATAPE functions [11] were proposed for embedding on MINs$^+$ in $O(N+\log_2 N)$. The first ATAPE function was presented in Eq. (2.3) with controls $C = 0$ to $N-1$ to provide N permutations of Latin squares, specified by the arbitrary order between 0 and N.

$$D = S \oplus (C + order)\%N \tag{2.3}$$

Table 1
Advantages and limitations of three popular networks.

	Advantages	Limitations
Networks		
1. Hypercube (HC)	- efficient communication - efficient ATAPE	- not scalable - expensive cost
2. Hierarchical Hypercube (HHC) [10]	- scalable - inexpensive	- easily conflict - not easy for ATAPE
3. Dragonfly [13]	- flexible communication - flexible ATAPE	- expensive cost (additional switches)
HHC-Routing		
Basic HHC Routing [10]	- simple *N2N*-routing (regular GC mapping)	- not guarantee SP - conflict for parallel routing, $k > 2^m$ nodes
Node-disjoint-Path HHC-Routing [20]	- reliable *N2N* routing	- for *N2N* routing - work for $S = 0$ only
k-pairwise disjoint-path HHC-routing [4]	- reliable parallel routing for $k \leq \lceil (m+1)/2 \rceil$ nodes	- conflict when $k > \lceil (m+1)/2 \rceil$ nodes
Parallel SP-HHC routing (proposed in Sections 3 and 4)	- parallel *N2N* SP-routing - partitioning for $k \leq 2^{m+1}$ - SP-ATAPE embedding	- conflict when $k > 2^{m+1}$ nodes

Next, in general the f-in-1 dynamic function was presented in Eq. (2.4) to allow a number of flexible f initial permutations, specified by users ($\rho[i]$, $i = 0, 1, 2, \ldots, N-1$) for other fN Latin-squares of the proper permutations.

$$D = \rho[(S + C + order)\%N] \tag{2.4}$$

In 2017, the flexible ATAPE partitioning and embedding were proposed on the $x \times x$ crossbar of partitionable MINs$^+$ [12], presented in Eq. (2.5). That partitionable ATAPE was embedded for multiple tasks with super-pipelining, where y represents section 0, 1, …, $x-1$ and l refers to the control ($l = 0, 1, 2, \ldots, N/2^t$).

$$D^y = S^y \oplus (yN/x + l)\%N \tag{2.5}$$

However, those ATAPE functions (in Eqs. (2.1)–(2.5)) could not work directly on the HHC networks.

Therefore, this study focuses on solving the HHC conflict directly by the parallel shortest-path routing for ATAPE embedding on the HHCs. Table 1 compares three popular networks. The HC network is efficient in ATAPE communication but not scalable and costly. The HHC is inexpensive and scalable but easily conflicts, especially when executing $k > 2^m$ nodes. The dragonfly is the most flexible ATAPE communication but the most expensive. Thus, solving the HHC conflict is a crucial research. In 2007, the node-disjoint-path routing [20] could be used to avoid the HHC conflict but it limited $S = 0$. In 2013, the reliable parallel-routing [4] could be used to avoid the conflict (in parallel) but supporting $k \leq \lceil (m+1)/2 \rceil$ nodes only. Therefore, this study proposes to solve the HHC-conflict directly: the arbitrary *N2N* shortest-path routing (in Section 3), the parallel *N2N* SP-routing (in Section 4.1), and the proper HHC partitioning (in Section 4.2) before embedding the shortest-path ATAPE communication.

3. Shortest-path routing on HHC networks

First, we introduce the *N2N* shortest-path routing on the HHC networks from any source $S = (\alpha_S, \beta_S)$ to any destination $D = (\alpha_D, \beta_D)$, $0 \leq S, D \leq N-1$, for not only the efficient (S, D)-routing but also the conflict solution. Later, in Section 4 we propose the efficient parallel SP-routing, the proper HHC partitioning, and the

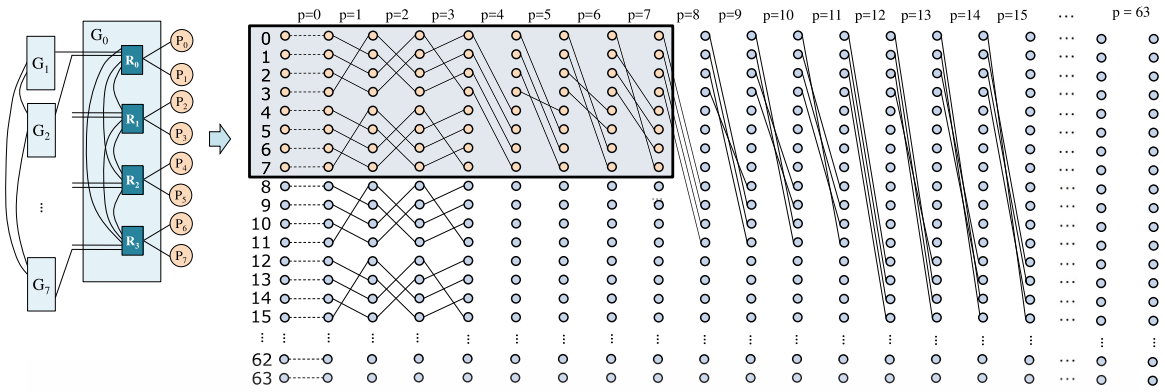


Fig. 5. The structure of the dragonfly network with $N = 64$ and the 2nd level sub-system ($N' = 8$).

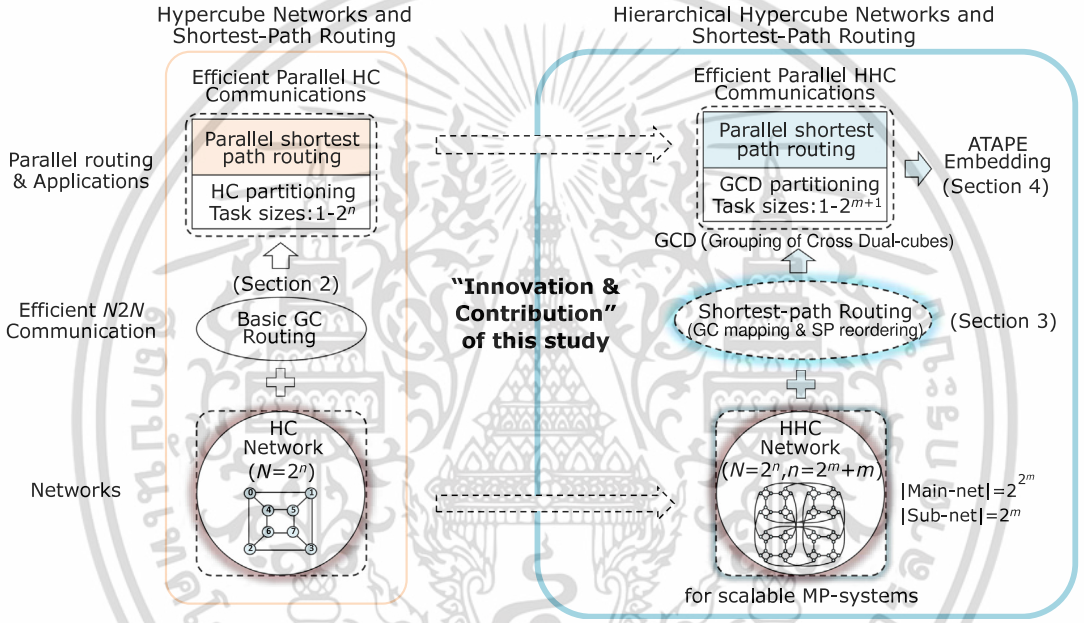
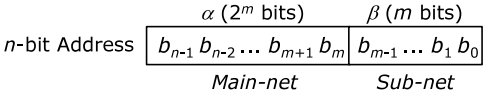


Fig. 6. The proposed parallel *N2N* SP-routing and ATAPE embedding on HHC networks, compared to the GC routing on HC networks.

ATAPE embedding on the HHCs, see Fig. 6 (our innovation and contribution).

Formally, let an n-bit address of any node on the HHC network ($N = 2^n$, $n = 2^m+m$, $m \geq 2$) be divided into two partitions (α, β) for the proper routing: 1. the external 2^m bits of the *main-net* ($\alpha = b_{n-1}\ldots b_{m+1}b_m$) and 2. the internal m bits of the *sub-net* ($\beta = b_{m-1}\ldots b_1 b_0$).



The contribution of our shortest-path (SP) routing (from S to D) focuses on two main functions: 1. the generalized EES (external edge sequence) with arbitrary S for the shortest external path across the *main-nets* (Algorithm 5 in Section 3.1), extended from the EES ($S = 0$) [20] and 2. the complete SP (Algorithm 6 in Section 3.2) by reordering the generalized EES for the SP-routing on the HHC networks, according to the arbitrary S (= 0, 1, 2, …, $N - 1$).

3.1. Shortest external-path (μ) across main-nets

Algorithm 5 was designed by combining Algorithm 1 (arbitrary S) and Algorithm 2 (EES: $S = 0$) for the generalized EES from the external nodes (α_S and α_D) on the *main-nets* of the arbitrary (S, D), where the source node $S = (\alpha_S, \beta_S)$ and the destination node $D = (\alpha_D, \beta_D)$. First, the Hamming distance (HD) between α_S and α_D is determined by $\alpha^* = \alpha_S \oplus \alpha_D = \{\{b_i\} \mid b_i \in \{b_{2^m+m-1}, b_{2^m+m-2}, \ldots, b_m\}\}$, where $\oplus$ denotes the bitwise-XOR operation to realize the HD (α_S, α_D) for routing across the particular *main-nets*. From the binary α^*, we include an index i into the index set if $b_i = 1$ and create the generalized EES (μ) by mapping indices i with the m -bit Gray-code.

For example, Fig. 7 displays the generalized EES for the SP-routing from S to D on the 6-HHC ($N = 64$, $n = 6$, and $m = 2$). Let S: (α_S, β_S) = 3: (0000, 11), D: (α_D, β_D) = 61: (1111, 01), and the Hamming distance (α_S, α_D) $\alpha^* = \alpha_S \oplus \alpha_D$ = 0000⊕1111 = 1111. Next, construct the index i, corresponding to $b_i = 1$ in α^*, which is the index set = {3, 2, 1, 0}. Then, map the index set with the m-bit Gray-code ($m = 2$, GC = {00, 01, 11, 10}) to obtain μ = {00, 01, 11, 10} for the shortest external path across the *main-nets*.

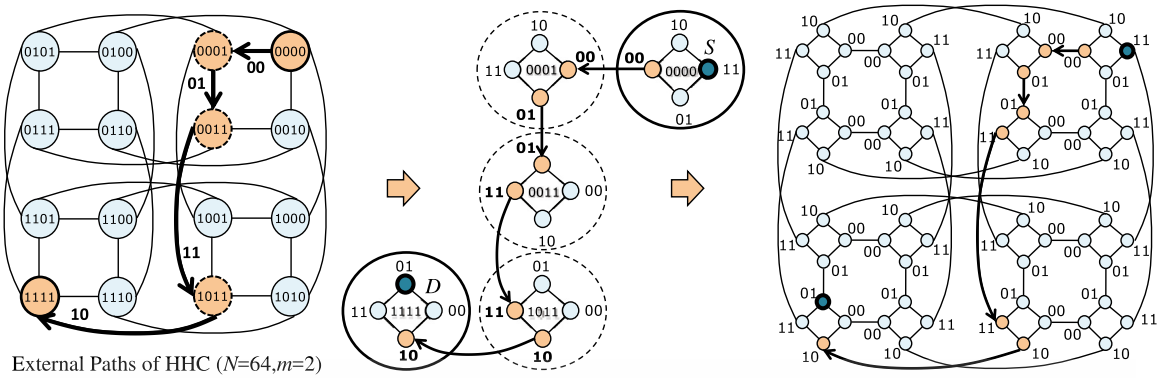


Fig. 7. An example of the generalized EES (μ) from S = (0000, 11) to D = (1111, 01), where μ = (00, 01, 11, 10).

Algorithm 5: Get shortest external path for routing from an arbitrary *S*.

```
GetExPath
1. index = Ø; μ = Ø;
2. α* = αS⊕αD = {b2^m+m−1, b2^m+m−2,..., bm};
3. for α* with bi = 1 do
4.       index = index ∪ i;
5. end for
6. for indexi do   // GC mapping for a shortest external-path
7.       μ = μ ∪ GC[indexi];
8. end for
9. return μ;
```

3.2. N2N shortest path by reordering μ

Algorithm 6 (GetSPathHHC) was designed to fulfill the shortest path (SP) P from the generalized EES (μ) in Algorithm 5 for the *N2N* SP-routing from S to D. This function is composed of two main policies: 1. the basic SP ($\alpha_S = \alpha_D$) and 2. the complex SP ($\alpha_S \neq \alpha_D$). In case $\alpha_S = \alpha_D$, the SP-routing is located within the same *main-net*, where the path between S and D is the HC routing on the cube of 2^m nodes. In case $\alpha_S \neq \alpha_D$, S and D are not located within the same *main-net*, and hence reordering some elements in $\mu = (c_0, c_1, \ldots, c_{r-1})$ for the SP-routing (arbitrary S) is the significant process of Algorithm 6. According to the internal nodes (β_S, β_D), and the shortest external-path (μ), there are five cases like Algorithm 2 ($S = 0$) [20], except the reordering in μ for the complete shortest-path P (with any S). The *SPordering* is our key contribution, computed in $O(|\mu|^2)$ based on the m-bit GC for the shortest sub-paths of successive nodes in μ. Then, the full shortest-path P is built from the best ordering of all elements in μ, where $|\mu| = r$.

For example, Fig. 8 demonstrates the construction of the shortest paths (P) on the 6-HHC ($N = 64$, $m = 2$, and $n = 2^m+m = 6$), such as the SP-routing of Case 1: (S, D) = (23, 43) in Fig. 8(a) and the SP-routing of Case 2: (S, D) = (3, 61) in Fig. 8(b). In particular, our focus is the *SPordering* (in Algorithm 6) for reordering all elements of μ (based on the m -bit Gray-code to find the shortest path P), where every of two successive nodes in P has a minimum number of different bits. Fig. 8(a) displays the SP-routing from $S = (\alpha_S, \beta_S) = (0101,11) = 23$ to $D = (\alpha_D, \beta_D) = (1010,11) = 43$. First, apply Algorithm 5, the operation $\alpha_S \oplus \alpha_D = 1111$ and $\mu = \{00, 01, 11, 10\}$. Then, apply Algorithm 6, the construction of the shortest path P in Case 1, where the successive nodes in P are rearranged by the path $P = \{\beta_S, SPordering(\mu - \beta_S)\}$ = {11, *SPordering*(μ - {11})}. Hence, the *SPordering*(μ - {11}) = *SPordering*(00, 01, 10) is reordered from $temp = \beta_S = 11$ by starting at index 0 in μ to find the closest bit to *temp*, then update new *temp* and repeat the process until all elements of μ are reordered. Finally, the *SPordering*(00, 01, 10) = (01, 00, 10) and the shortest path P = {11, *SPordering*(μ - {11})} = (11, 01, 00, 10) with length = 8, which traverses from S to D across 4 internal edges and 4 external edges alternately. See the proof of correctness (the *N2N* SP-routing by Algorithm 6) in Section 5.1.

Table 2
Key contributions of proposed algorithms in Sections 3–4.

SP-routing Algorithms	Key Contributions	Limitations
Algorithm 5: Hamming Distance (S,D)&GC mapping	- (arbitrary S) external SP (shortest path) ($0 \le S, D \le N-1$)	- not complete SP
Algorithms 5 and 6: SP reordering and routing	- *N2N* SP-routing ($0 \le S, D \le N-1$)	- not cover parallel SP-routing
Algorithms 5 and 7: Dynamic SP (F/B) reordering	- parallel *N2N* SP-routing ($0 \le S, D \le N-1$)	- may conflict for $k > 2^m$ nodes
Algorithms 5, 7, and 8: GCD partitioning & SP (F/B) reordering and routing	- parallel *N2N* SP-routing - proper HHC-partitioning - full ($k = 2^{m+1}$) utilization	- may conflict for $k > 2^{m+1}$ nodes

Algorithm 6: Reordering the external path for *N2N* SP-routing.

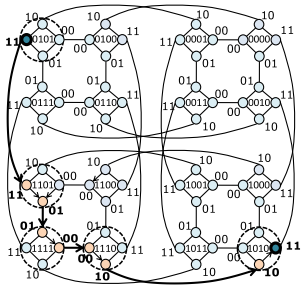
```
GetSPathHHC
1.  P = Ø; μ = GetExPath (Algorithm 5)
2.  if (αS = αD) then
3.        set Pin (HC internal–path routing);
4.  else (αS ≠ αD)
5.        Case 1: βs ∈ μ and βs = βD
6.                P = P ∪ {βS, SPordering (μ−{βS})};
7.        Case 2: βs, βD ∈ μ and βs ≠ βD
8.                P = P ∪ {βS, SPordering (μ−{βS,βD}), βD};
9.        Case 3: βs ∈ μ and βD ∉ μ
10.               P = P ∪ {βS, SPordering (μ−{βS})};
11.       Case 4: βs ∉ μ and βD ∈ μ
12.               P = P ∪ { SPordering (μ−{βD}), βD};
13.       Case 5: βs ∉ μ and βD ∉ μ
14.               P = P ∪ { SPordering (μ)};
15. return P;

SPordering in O(|μ|^2), |μ|=r (HD between main-nets of S & D)
1.  temp = βs; r' = r ;
2.  while( |ρ| < r) do
3.    for( i = 0; i < r'; i++) do
4.       find μ[i] that is closest to temp;
5.       temp = μ[i]; p = p ∪ μ[i]; update |μ|-1; r' = r'-1; break;
6.    end for
7.  end while
```

Note: for reliable routing, Algorithm 6 can be extended directly from μ to find other paths (P_i). Table 2 illustrates contributions vs. limitations of Algorithms 5–6 and our new solution for parallel SP-routing (in Section 4).

4. ATAPE embedding and HHC partitioning

First, we propose the efficient parallel *N2N* shortest-path routing ($0 \le S, D \le N-1$) in Section 4.1. Second, we propose the GCD (grouping of cross dual-cube) partitioning to avoid the conflict on

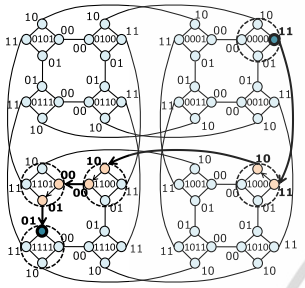


$\alpha^* = \alpha_S \oplus \alpha_D = (1111)$
$\mu = \{c_0, c_1, c_2, c_3\} = \{00,01,11,10\}$
Case 1: $P = \{\beta_S$, SPordering$(\mu - \{\beta_S\})\}$
$= \{11,$ SPordering$(\mu - \{11\})\}$
SPordering$(\mu - \{11\})$=SPordering$(00,01,\cancel{11},10)$
$=\{01,00,10\}$
then $P = \{11,01,00,10\}$

HHC routing-path from S to D

Main-Net	Sub-Net
0101	S: 0101, 11
1101	1101, **11** 1101, 01
11**1**1	1111, **01** 1111, 00
111**0**	1110, **00** 1110, 10
101**0**	1010, **10** D: 1010, 11
4 steps	4+4 steps

(a) $(S, D) = (23, 43)$: Case 1 in Algorithm 6



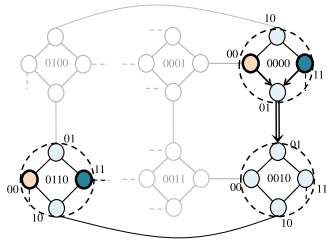
$\alpha^* = \alpha_S \oplus \alpha_D = (1111)$
$\mu = \{c_0, c_1, c_2, c_3\} = \{00,01,11,10\}$
Case 2: $P = \{\beta_S$, SPordering$(\mu - \{\beta_S \beta_D\}), \beta_D\}$
$= \{11,$ SPordering$(\mu - \{11,01\}), 01\}$
SPordering$(\mu - \{11,01\})$=SPordering$\{00,\cancel{01},\cancel{11},1$
$= \{10,00\}$
then $P = \{11,10,00,01\}$

HHC routing-path from S to D

Main-Net	Sub-Net
0000	S: 0000, 11
1000	1000, **11** 1000, 10
11**0**0	1100, **10** 1100, 00
110**1**	1101, **00** 1101, 01
111**1**	D: 1111, **01**
4 steps	4+3 steps

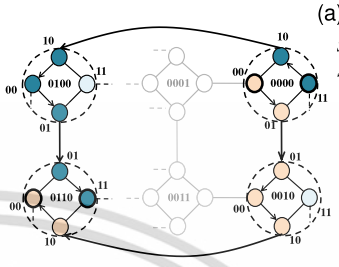
(b) $(S, D) = (3, 61)$: Case 2 in Algorithm 6

Fig. 8. An example of the SP construction (P) on the HHC ($N = 64$).



$S_1 = (0000, 00) = 0$; $D_1 = (0110,00) = 24$
$S_2 = (0000, 11) = 3$; $D_2 = (0110,11) = 27$
$\alpha^* = \alpha_S \oplus \alpha_D = 0110$; Thus $\mu = \{01, 10\}$
Case 5: $P = \{$SPordering$(\mu)\}$
2-bit-GC = {00, 01, 11, 10}
SPordering compares $\beta_{S1} = 00$ with μ
then (path $S_1 \rightarrow D_1$) $P_1 = \{01, 10\}$
SPordering compares $\beta_{S2} = 11$ with μ
then (path $S_2 \rightarrow D_2$) $P_2 = \{01, 10\}$

(a)



$S_1 = (0000, 00) = 0$; $D_1 = (0110,00) = 24$
$S_2 = (0000, 11) = 3$; $D_2 = (0110,11) = 27$
$\alpha^* = \alpha_S \oplus \alpha_D = 0110$; Thus $\mu = \{01, 10\}$
Case $\beta_S \notin \mu$: $P = \{d$SPordering$(\mu)\}$
2-bit-GC = {00, 01, 11, 10}
dSPordering compares $\beta_{S1} = 00$ with 01
then path $(S_1 \rightarrow D_1)$ $P_1 = \{01, 10\}$
dSPordering compares $\beta_{S2} = 11$ with 10
then path $(S_2 \rightarrow D_2)$ $P_2 = \{10, 01\}$

HHC routing-path from S_1 to D_1

Main-Net	Sub-Net
0000	S_1 :0000, 00 0000, 01
00**1**0	0010, **01** 0010, 00 0010, 10
011**0**	0110, **10** D_1 :0110, 00
2 steps	2+4 steps

HHC routing-path from S_2 to D_2

Main-Net	Sub-Net
0000	S_2 :0000, 11 0000, 10
0**1**00	0100, **10** 0100, 11 0100, 01
011**0**	0110, **01** D_2 :0110, 11
2 steps	2+4 steps

(b)

Fig. 9. An example of parallel SP-routing on the HHC: (a) regular *SPordering* (conflict) and (b) dynamic *dSPordering* (no conflict).

the HHCs and the shortest-path ATAPE embedding in Section 4.2. Lastly, all crosses ($k = 2^{m+1}$ nodes per cross) in any group are allowed for multiple tasks by the F (forward)/B (backward) protocol (for full bidirectional-routing on the HHCs).

4.1. Parallel shortest-paths by dynamic reordering

Algorithm 7 was designed for parallel *N2N* SP-routing on the HHCs. This algorithm focuses on two functional keys: 1. dynamic SP-reordering (from β_S, find the next dynamic d) for symmetrical shortest-path routing and 2. case reduction (1. $\beta_S \in \mu$ and 2. $\beta_S \notin \mu$) for the efficient bidirectional-routing. The (forward/backward) *dSPordering* is our significant key to maintain the parallel shortest paths and the full bidirectional-routing on the HHCs. The (default) forward *dSPordering* is the circular right-shift reordering and the backward *dSPordering* is the circular left-shift reordering (to utilize the opposite directions).

Algorithm 7: Dynamic reordering the external path for parallel routing.

```
GetParallelSPathHHC
1.  P = Ø; μ = GetExPath (Algorithm 5)
2.  if (αS = αD) then
3.      set Pin (HC internal–path routing);
4.  else (αS ≠ αD)
5.      if (βs ∈ μ) then P = P ∪ {βS, dSPordering (μ–{βS})};
6.      else             P = P ∪ {dSPordering (μ)};
7.  return P;
```

*d*SPordering (Forward)	*d*SPordering (Backward)
temp = β_s; r' = r ;	*temp* = β_s; r' = r ;
d:=(next index of β_s in μ) %r ;	*d*:=(back index of $\beta_s$in μ+r)%r ;
while(|p| < r) **do**	**while**(|p| < r) **do**
for(i = 0; i < r'; i++) **do**	**for**(i = 0; i < r'; i++) **do**
id = (i + d) % r';	id = (d - i + r') % r';
find μ[id] closest to *temp*;	find μ[id] closest to *temp*;
temp = μ[id]; p = p ∪ μ[id];	*temp* = μ[id]; p = p ∪ μ[id];
update |μ|-1; r' = r'-1;	update |μ|-1; r' = r'-1;
d = (id+1)%r'; **break**;	d =(id-1+r')%r'; **break**;
end for	**end for**
end while	**end while**

For example, Fig. 9(a) shows a congestion of two paths on the HHC ($N = 64$, $m = 2$) from $S_1 = (0000, 00)$ to $D_1 = (0110, 00)$ and $S_2 = (0000, 11)$ to $D_2 = (0110, 11)$. By applying Algorithm 6 ($\alpha_S \oplus \alpha_D = 0110$ and $\mu_1 = \mu_2 = \{01, 10\}$), both paths are in Case 5 ($\beta_s \notin \mu$ and $\beta_D \notin \mu$), where $P_1 = SPordering(\mu_1) = \{01, 10\}$ and $P_2 = SPordering(\mu_2) = \{01, 10\}$. These two paths conflict at the edge $e = (u, v)$, $u = (0000, 01)$ and $v = (0010, 01)$, in the routing from $S_1 \rightarrow D_1$ and $S_2 \rightarrow D_2$ (in the second step) since S_1 and S_2 reside in the same *main-net*.

Applying Algorithm 7 can avoid the conflict by the dynamic *dSPordering* for the shortest-path routing. Fig. 9(b) shows the (forward) reordering and routing of both paths with no conflict. In this solution, these paths are in case $\beta_s \notin \mu$ (of Algorithm 7) with $\mu = \{01, 10\}$, where each path starts at the next index (d) from its β_S (for the symmetrical-path routing). For instance, P_1 starts at $d = 01$ (from $\beta_{S1} = 00$) and P_2 starts at $d = 10$ (from $\beta_{S2} = 11$) for $P_1 = \{01, 10\}$ and $P_2 = \{10, 01\}$. Clearly, the proper right-shift of dynamic index d in the (forward) *dSPordering* generates the symmetrical paths that can avoid the conflict. Next (in Section 4.2), both of (forward/backward) reordering functions will be applied in the ATAPE application.

4.2. HHC partitioning and SP-ATAPE embedding

Under traditional HHC-partitioning, the parallel *N2N* SP-routing (Algorithm 7) can process on the HHC-MPs without the conflict ($0 \le S, D \le N - 1$) for $k \le 2^m$ nodes only.

Fig. 10(a) shows the regular HHC-partitioning for $k = 2^{m+1}$ nodes from two adjacent cubes ($2x2^m$ nodes). In this case, there is a routing conflict of two paths from $S_1 = (0000, 01)$ to $D_1 = (0001, 01)$ and $S_2 = (0000, 10)$ to $D_2 = (0001, 10)$ at the edge $e = (u, v)$, where $u = (0000, 00)$ and $v = (0001, 00)$. Fig. 10(b) displays our proper HHC-partitioning (for $k = 2^{m+1}$ nodes) from a cross of two dual cubes for the symmetrical-path routing (i.e., $S_1 = (0000,$

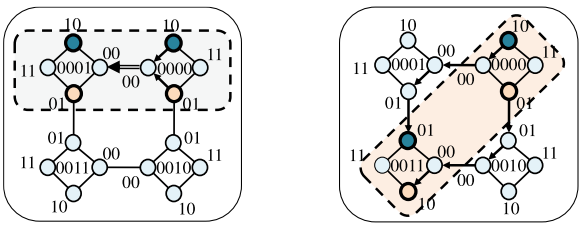


Fig. 10. A proper sub-system (2^{m+1}) on the HHC ($N = 64$, $m = 2$): (a) adjacent cubes (conflict) and (b) a cross of dual cubes (no conflict).

01) to D_1 = (0011, 10) and S_2 = (0000, 10) to D_2 = (0011, 01), and so on for others that can use shared edges in different clocks).

In Section 4.2.1, we propose the grouping of cross dual-cube (GCD) partitioning to work with Algorithm 7 (parallel SP-routing) for $k = 2^{m+1}$ nodes per cross. In Section 4.2.2, we present the shortest-path ATAPE communication ($k \le 2^{m+1}$ nodes), which will be used to evaluate our conflict solution in the experiment (in Section 5.4).

4.2.1. GCD (Grouping of Cross Dual-Cube) Partitioning

The objective of our GCD partitioning is to find the right cross of dual cubes ($k = 2^{m+1}$ nodes) in the right group that can avoid the conflict for the parallel *N*2*N* SP-routing (across the symmetrical paths), proven in Theorems 4.1–4.3. See a corresponding example in Fig. 11 ($N = 64$).

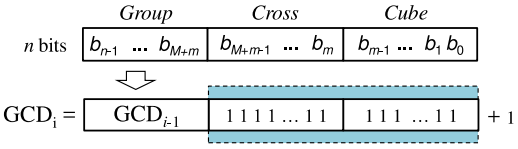
Theorem 4.1. *On partitionable HHCs ($N = 2^n$, $n = 2^m + m$), the minimum size of the independent group (in the GCD partitioning) for $k = 2^{m+1}$ nodes without the conflict is $N' = 2^{n'}$, $n' = M + m$, $M = 2^{m-1}$, and a number of crosses (two cubes ($2x2^m$) per cross) in each group are equal to 2^{M-1}.*

Proof. According to GCD partitioning (to avoid the conflict), the minimum size of a group is 2^{M+m} nodes (or 2^M *main-nets*) and the minimum $M = 2^{m-1}$ since on the HHC ($m \ge 2$) the value M cannot be less than 2^{m-1}. For instance, if |group| $= 2^{M+m}$ and $M = 2^{m-2}$, then $2^{M+m} = 2^{m+1}$, which causes the conflict in communication among $k = 2^{m+1}$ nodes ($m = 2$), see Fig. 10(a). Thus, the minimum |group| must be 2^{M+m} and $M = 2^{m-1}$, see Fig. 10(b). Then, #crosses per group are $2^{n'}/2^{m+1} = 2^{M-1}$, where $0 \le cross\ j \le 2^{M-1} - 1$. □

Theorem 4.2. *Under GCD partitioning, a number of independent groups on the HHC-MP systems ($N = 2^n$, $n = 2^m+m$) is $N/2^{n'} = 2^M$, where $n' = M+m$ and $M = 2^{m-1}$. In each group ($N' = 2^{n'}$ nodes or processors), GCD_i (a starting node of a group i) can be computed from GCD_{i-1} by Eq. (4.1) in O(1), where $1 \le i \le 2^M - 1$.*

$$GCD_i = GCD_{i-1} + 2^{M+m} \text{ and } GCD_0 = 0 \quad (4.1)$$

Proof. From Theorem 4.1, the group size is $N' = 2^{n'}$, and hence a number of groups $= N/2^{n'} = 2^{2^m+m}/2^{2^{m-1}+m} = 2^{2^{m-1}} = 2^M$. In each group, a number of crosses $= 2^{M-1}$. Under GCD partitioning, the starting nodes (or processors) of two successive GCDs can be defined as follows: the first node of GCD_i = the last node of GCD_{i-1}+1, where $GCD_0 = 0$, see the corresponding example in Fig. 11.



From the above figure, a decimal number of the $M+m$-bit address is $b_{M+m-1}\ldots b_1 b_0 = 111\ldots11$ (maximum) $= \sum_{i=0}^{M+m-1} 2^i$.

$$\text{Therefore, } GCD_i = GCD_{i-1} + \sum_{i=0}^{M+m-1} 2^i + 1$$
$$= GCD_{i-1} + (2^{M+m-1+1} - 1) + 1$$
$$= GCD_{i-1} + 2^{M+m}. \quad \square$$

In each group, all 2^{M-1} crosses ($cross_j$, $0 \le j < 2^{M-1}$) are defined by Algorithm 8 (for 2^{m+1} source (S) nodes per cross) from GCD_i, which finds dual cubes in $cross_0$ of group i first (i.e., the first cube and the last cube) for symmetrical-path routing. For group 0, $cross_0$ contains dCube1 $= GCD_0 = 0$ and dCube2 $= GCD_0$ +| group| −| cube| $= 0 + 2^{M+m} - 2^m = 16 - 4 = 12$. Then for $cross_{j+1}$, next dCube1 $= d$Cube1 $+ 2^m = 0 + 4 = 4$ and next dCube2 $= d$Cube2 $- 2^m = 12 - 4 = 8$.

Algorithm 8: Pre-processing for 2^{m+1} sources (S) of crosses by GCD_i.

```
GetSourceOfSubSystem (GCDi)
1.  dCube1=GCDi; dCube2=GCDi+|group|-2^m; // |group|=2^(M+m)
2.  for (cross j=0; j<#crosses; j++) do          // 2 cubes per cross
3.      for (processor p=0; p<2^m; p++) do       // 2x2^m nodes
4.          S[p] = dCube1+p;        // S on dual Cube 1 (of cross_j)
5.          S[p+2^m] = dCube2+p;    // S on dual Cube 2 (of cross_j)
6.      end for
7.      dCube1 = dCube1+2^m; dCube2 = dCube2-2^m;
8.  end for
```

For example, for $k = 2^{m+1}$ nodes on the HHC ($N = 64$, $n = 6$, $m = 2$), all GCDs (GCD_i, $0 \le i < 4$) are 0, 16, 32, 48 (the starting nodes of all groups), defined by Eq. (4.1). For $GCD_0 = 0$, a number of crosses are equal to $2^{M-1} = 2$ (in case of $m = 2$), which are $cross_0$ and $cross_1$. Then, all sources in $cross_0$ and $cross_1$ are determined by Algorithm 8, where all sources of $cross_0$ = {0, 1, 2, 3, 12, 13, 14, 15} from dCube1 = 0, dCube2 = 12, and all sources of $cross_1$ = {4, 5, 6, 7, 8, 9, 10, 11} from dCube1 = 4, dCube2 = 8 (see Fig. 11 for 2^{m+1} sources in each $cross_j$ of all groups). Fig. 12 shows all GCDs (GCD_i, $0 \le i \le 15$) on $N = 2048$ ($m = 3$) of all groups, which are 0, 128, 256, 384, 512, 640, 768, 896, 1024, 1152, 1280, 1408, 1536, 1664, 1792 and 1920.

Theorem 4.3. *Under GCD partitioning, all independent groups can be processed at the same time. In each group, even crosses apply the forward (F) reordering (for $k = 2^{m+1}$ nodes per cross). At the same time, odd crosses apply the backward (B) reordering by using the same control.*

Proof. Fig. 13(a) displays the GCD address and #crosses (per group) $= 2^{M-1}$, where $M = 2^{m-1}$. Fig. 13(b) shows the GCD address of $N = 64$ ($m = 2$), two crosses per group. Usually, only $cross_0$ with the forward (F) reordering is allowed for the symmetrical-path routing between nodes of dual cubes, except routing in the opposite directions by the backward (B) reordering on $cross_1$. Fig. 13(c) displays the GCD address of $N = 2048$ ($m = 3$), eight crosses per group. In any group, all (even, odd) crosses can work with the (F, B) protocol, where the flipped crosses ((0 : 000, 5 : 101), (2 : 010, 7 : 111), (4 : 100, 1 : 001), (6 : 110, 3 : 011)) cannot use the same rule of reordering. Thus, the even crosses (50%) can apply the forward *dSPordering* (Algorithm 7).

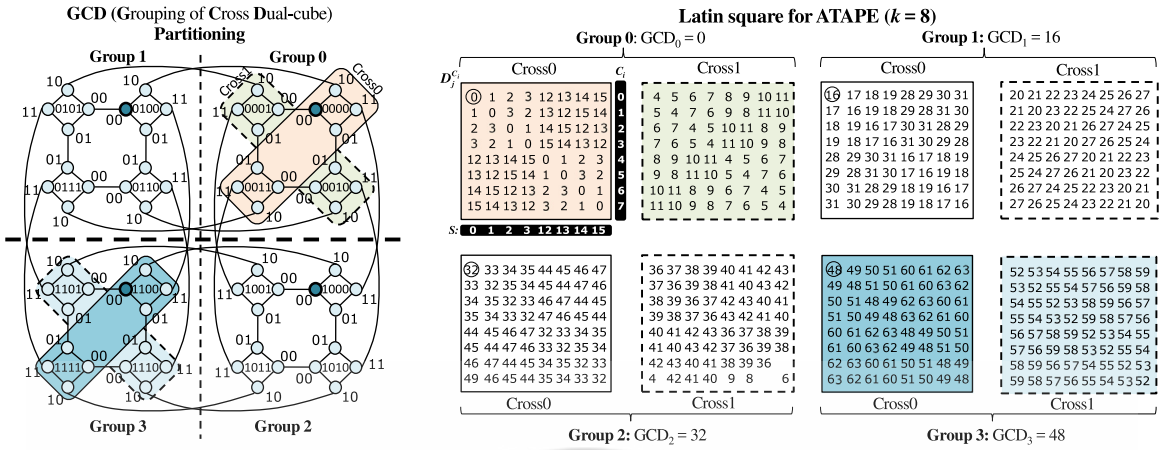


Fig. 11. An example of the GCD partitioning for the shortest-path Latin squares on a 6-HHC ($N = 64$, $n = 6$, $m = 2$) with $k = 2^{m+1} = 8$.

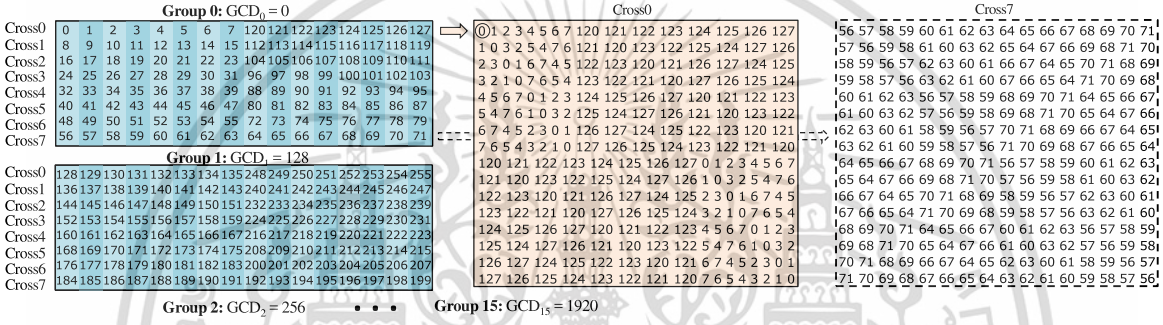


Fig. 12. An example of GCDs, Latin squares, and *cross* sub-systems on an 11-HHC ($N = 2048$, $n = 11$, $m = 3$) with $k = 2^{m+1} = 16$.

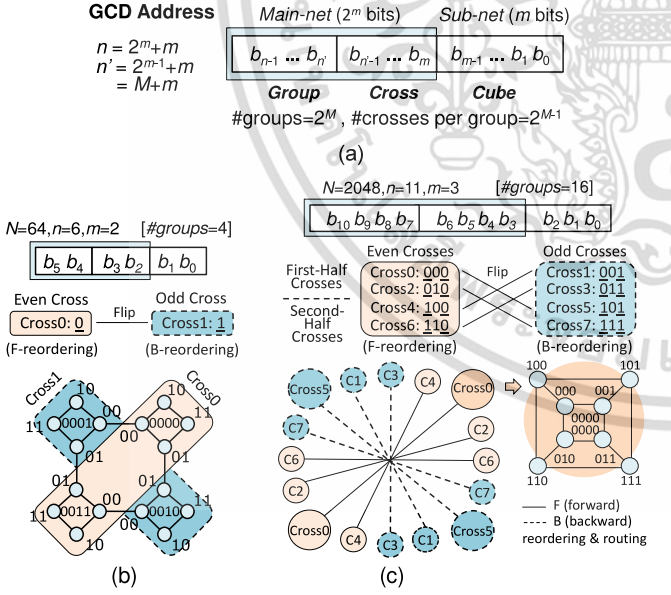
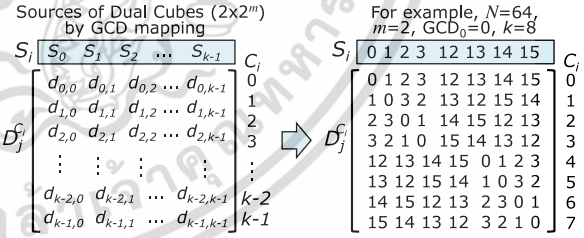


Fig. 13. A GCD-address format and (F, B) protocol for (even, odd) crosses: (a) GCD address; (b) $N = 64$ ($m = 2$); and (c) $N = 2048$ ($m = 3$).

The odd crosses (another 50%) can apply the backward *dSPordering* to routing in the opposite directions across the symmetrical paths. In general, all crosses (with the (F, B) protocol) can be utilized at the same time, by synchronized with the same control, proven in Section 5.3. □

4.2.2. ATAPE (All-to-All Personalized Exchange)

This section presents the shortest-path ATAPE, an application of parallel *N*2*N* SP-routing ($k \leq 2^{m+1}$), on the HHC networks by the indirect XOR operation in Eq. (4.2).



$$D_j^{C_i} = d_{i,j} = S_{(C_i \oplus j)} \tag{4.2}$$

For the SP-ATAPE communication, a Latin square (LS) of destinations (D) is computed in k iterations of controls (C) from the proper sources (S) defined by Algorithm 8 and the indirect XOR in Eq. (4.2). For example ($N = 64$, $m = 2$, $k = 2^{m+1} = 8$), all S (source) nodes are computed by Algorithm 8 (using $\text{GCD}_0 = 0$ on group 0), which is $S = \{0, 1, 2, 3, 12, 13, 14, 15\}$. Next, the LS of ATAPE is defined, according to the controls $C_i = i$ ($= 0, 1, \ldots, k - 1$), see Fig. 11 (all possible (shortest-path) LSs of the ATAPE communication). The global addresses of each LS are generated to work with Algorithm 7 (parallel *N*2*N* SP-routing). In each GCD_i (i.e., $\text{GCD}_0 = 0$ in group 0 on $N = 64$, $m = 2$), there exist two crosses ($cross_0$, $cross_1$). The $cross_0$ (on *main-nets* 0000 and 0011) is composed of $k = 8$ processors $S = \{0, 1, 2, 3, 12, 13, 14, 15\}$. The $cross_1$ (on *main-nets* 0001 and 0010) is composed of 8 processors $S = \{4, 5, 6, 7, 8, 9, 10, 11\}$.

Table 3
Forward reordering-and-routing paths of shortest-path ATAPE in each control (C), where $GCD_0 = 0$ (on $Cross_0$).

$C = 0$	$S = D$	0:0000,00	1:0000,01	2:0000,10	3:0000,11	12:0011,00	13:0011,01	14:0011,10	15:0011,11
$C = 1$	S	0:0000,00	1:0000,01	2:0000,10	3:0000,11	12:0011,00	13:0011,01	14:0011,10	15:0011,11
	D	1:0000,01	0:0000,00	3:0000,11	2:0000,10	13:0011,01	12:0011,00	15:0011,11	14:0011,10
$C = 2$	S	0:0000,00	1:0000,01	2:0000,10	3:0000,11	12:0011,00	13:0011,01	14:0011,10	15:0011,11
	D	2:0000,10	3:0000,11	0:0000,00	1:0000,01	14:0011,10	15:0011,11	12:0011,00	13:0011,01
$C = 3$	S	0:0000,00	1:0000,01	2:0000,10	3:0000,11	12:0011,00	13:0011,01	14:0011,10	15:0011,11
		0000,01(in)	0000,00(in)	0000,11(in)	0000,10(in)	0011,01(in)	0011,00(in)	0011,11(in)	0011,10(in)
	D	3:0000,11	2:0000,10	1:0000,01	0:0000,00	15:0011,11	14:0011,10	13:0011,01	12:0011,00
$C = 4$	S	0:0000,00	1:0000,01	2:0000,10	3:0000,11	12:0011,00	13:0011,01	14:0011,10	15:0011,11
		0001,00(ex) 0001,01(in) 0011,01(ex)	0010,01(ex) 0010,00(in) 0011,00(ex)	0000,00(in) 0001,00(ex) 0001,01(in) 0011,01(ex) 0011,00(in)	0000,01(in) 0010,01(ex) 0010,00(in) 0011,00(ex) 0011,01(in)	0010,00(ex) 0010,01(in) 0000,01(ex)	0001,01(ex) 0001,00(in) 0000,00(ex)	0011,00(in) 0010,00(ex) 0010,01(in) 0000,01(ex) 0000,00(in)	0011,01(in) 0001,01(ex) 0001,00(in) 0000,00(ex) 0000,01(in)
	D	12:0011,00	13:0011,01	14:0011,10	15:0011,11	0:0000,00	1:0000,01	2:0000,10	3:0000,11
$C = 5$	S	0:0000,00	1:0000,01	2:0000,10	3:0000,11	12:0011,00	13:0011,01	14:0011,10	15:0011,11
		0001,00(ex) 0001,01(in)	0010,01(ex) 0010,00(in)	0000,00(in) 0001,00(ex) 0001,01(in) 0011,01(ex)	0000,01(in) 0010,01(ex) 0010,00(in) 0011,00(ex)	0010,00(ex) 0010,01(in)	0001,01(ex) 0001,00(in)	0011,00(in) 0010,00(ex) 0010,01(in) 0000,01(ex)	0011,01(in) 0001,01(ex) 0001,00(in) 0000,00(ex)
	D	13:0011,01	12:0011,00	15:0011,11	14:0011,10	1:0000,01	0:0000,00	3:0000,11	2:0000,10
$C = 6$	S	0:0000,00	1:0000,01	2:0000,10	3:0000,11	12:0011,00	13:0011,01	14:0011,10	15:0011,11
		0001,00(ex) 0001,01(in) 0011,01(ex) 0011,00(in)	0010,01(ex) 0010,00(in) 0011,00(ex) 0011,01(in)	0000,00(in) 0001,00(ex) 0001,01(in) 0011,01(ex)	0000,01(in) 0010,01(ex) 0010,00(in) 0011,00(ex)	0010,00(ex) 0010,01(in) 0000,01(ex) 0000,00(in)	0001,01(ex) 0001,00(in) 0000,00(ex) 0000,01(in)	0011,00(in) 0010,00(ex) 0010,01(in) 0000,01(ex)	0011,01(in) 0001,01(ex) 0001,00(in) 0000,00(ex)
	D	14:0011,10	15:0011,11	12:0011,00	13:0011,01	2:0000,10	3:0000,11	0:0000,00	1:0000,01
$C = 7$	S	0:0000,00	1:0000,01	2:0000,10	3:0000,11	12:0011,00	13:0011,01	14:0011,10	15:0011,11
		0001,00(ex) 0001,01(in) 0011,01(ex)	0010,01(ex) 0010,00(in) 0011,00(ex)	0000,00(in) 0001,00(ex) 0001,01(in)	0000,01(in) 0010,01(ex) 0010,00(in)	0010,00(ex) 0010,01(in) 0000,01(ex)	0001,01(ex) 0001,00(in) 0000,00(ex)	0011,00(in) 0010,00(ex) 0010,01(in)	0011,01(in) 0001,01(ex) 0001,00(in)
	D	15:0011,11	14:0011,10	13:0011,01	12:0011,00	3:0000,11	2:0000,10	1:0000,01	0:0000,00

For parallel shortest-path routing ($N = 64$), Table 3 and Fig. 14 show the symmetrical routing paths of the ATAPE communication ($C = 0, 1, 2, \ldots, 7$) by GCD_0 (on $cross_0$). For instance, with $C = 4$ in case of $S = 15(0011, 11)$ and $D = 3(0000, 11)$ the SP-routing is derived as follows:

$S(15)$: 0011,11 $\xrightarrow{*}$ 0011,01(in) $\rightarrow$ 0001,01(ex) $\xrightarrow{*}$ 0001,00(in) $\rightarrow$ 0000,00(ex) $\xrightarrow{*}$ 0000,01(in) $\xrightarrow{*}$ $D(3)$: 0000,11.

The symmetrical paths of the SP-ATAPE communication (between two dual cubes ($k = 8$) in any group) can work with no conflict, including $C \geq k/2$ (densely bidirectional routing with time multiplex) such as $C = 7$ (111). Under this time multiplex, shared edges are allowed in different clocks, where the 2-cross bits (b_3b_2) of the *main-net* and the 2 bits (b_1b_0) of the *sub-net* (in k pairs) may be fully utilized. See $C = 7$ (Table 3) for parallel SP-routing from $S = \{0, 1, 2, 3, 12, 13, 14, 15\}$ to $D = \{15, 14, 13, 12, 3, 2, 1, 0\}$.

-$S = 0, D = 15$: $clk_1(0, 4)$, $clk_2(4, 5)$, $clk_3(5, 13)$, $clk_4(13, 15)$.
-$S = 1, D = 14$: $clk_1(1, 9)$, $clk_2(9, 8)$, $clk_3(8, 12)$, $clk_4(12, 14)$.
-$S = 2, D = 13$: $clk_1(2, 0)$, $clk_2(0, 4)$, $clk_3(4, 5)$, $clk_4(5, 13)$.
-$S = 3, D = 12$: $clk_1(3, 1)$, $clk_2(1, 9)$, $clk_3(9, 8)$, $clk_4(8, 12)$.
-$S = 12, D = 3$: $clk_1(12, 8)$, $clk_2(8, 9)$, $clk_3(9, 1)$, $clk_4(1, 3)$.
-$S = 13, D = 2$: $clk_1(13, 5)$, $clk_2(5, 4)$, $clk_3(4, 0)$, $clk_4(0, 2)$.
-$S = 14, D = 1$: $clk_1(14, 12)$, $clk_2(12, 8)$, $clk_3(8, 9)$, $clk_4(9, 1)$.
-$S = 15, D = 0$: $clk_1(15, 13)$, $clk_2(13, 5)$, $clk_3(5, 4)$, $clk_4(4, 0)$.

Fig. 14 displays the big picture of the HHC routing ($N = 64$, $m = 2$) for the ATAPE communication ($k = 8$) with controls $C = 0$ to 7 on $cross_0$ (nodes 0–3 & 12–15) of group 0, where the parallel shortest paths are determined by the forward reordering and routing. Note that in each of $C = 4$ to 7, there are some shared edges but used in different clocks (with no conflict). Fig. 15 shows another example of the HHC routing of five ATAPE-tasks (one of 2^{m+1} nodes and four of $4x2^m$ nodes). The first task (T_1, $k = 8$) with $GCD_0 = 0$ can assign on nodes 0–3 & 12–15 ($cross_0$) with controls $C = 0$ to 7. At the same time, two smaller tasks (T_2&T_3, $k = 4$) can assign to nodes 4–7 & 8–11 ($cross_1$) with $C = 0$ to 3 by using the regular cube-routing (on *sub-nets*). After T_2 and T_3 finished, tasks (T_4&T_5, $k = 4$, $C = 0$ to 3) can process on nodes 4–7 & 8–11 ($cross_1$).

5. Correctness and experiment

To confirm our hypothesis "the shortest-path routing and the proper HHC-partitioning can avoid the conflict", the correctness of the shortest paths of the *N2N* SP-routing and the parallel *N2N* SP-routing ($0 \leq S, D \leq N - 1$) were conducted in Sections 5.1–5.2. Then, the correctness of the GCD partitioning for $k = 2^{m+1}$ nodes per cross was proven in Section 5.3. Finally, the experiment was performed in Section 5.4 to confirm no conflict by using the ATAPE communication on the HHCs, especially under the (F, B) protocol on all (even, odd) crosses in any group.

5.1. Correctness of N2N shortest-path routing

The *N2N* SP-routing on the HHCs was achieved by

1. Algorithm 5: Generalized EES (μ) by Gray-code (GC)-mapping for the shortest external path (S to D) and

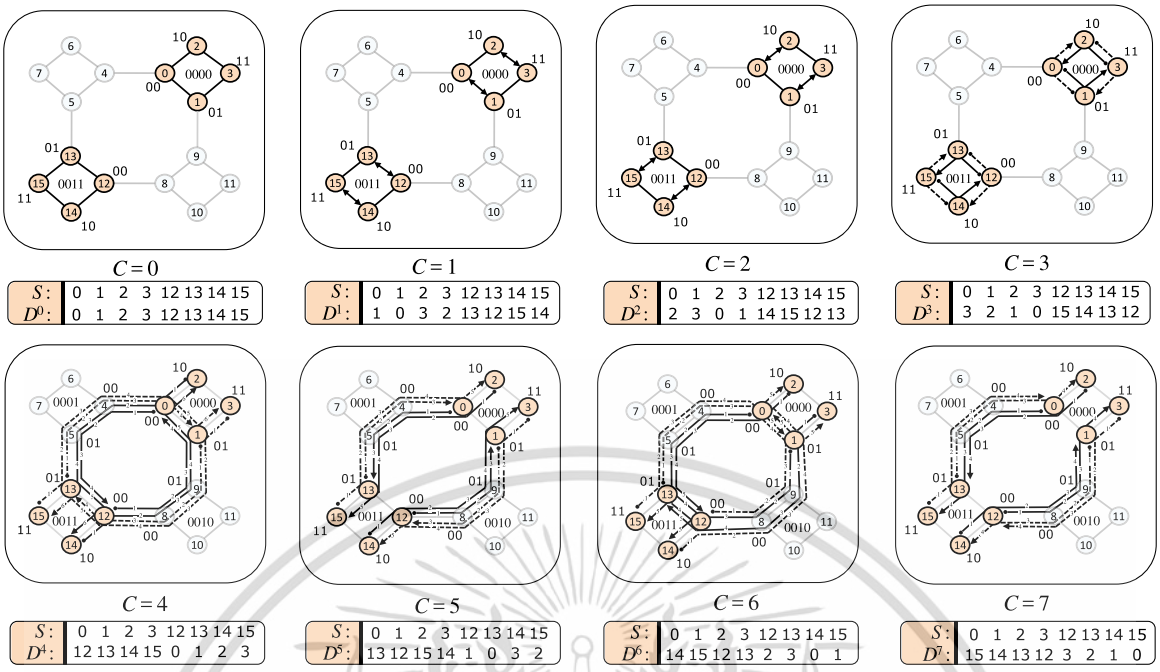


Fig. 14. An example of ATAPE ($k = 2^{m+1}$) routing paths with time multiplex in each of k controls (C) with $GCD_0 = 0$ (on $cross_0$), where S = source processors and D^C = destination processors on the HHC ($N = 64$, $m = 2$, $k = 8$).

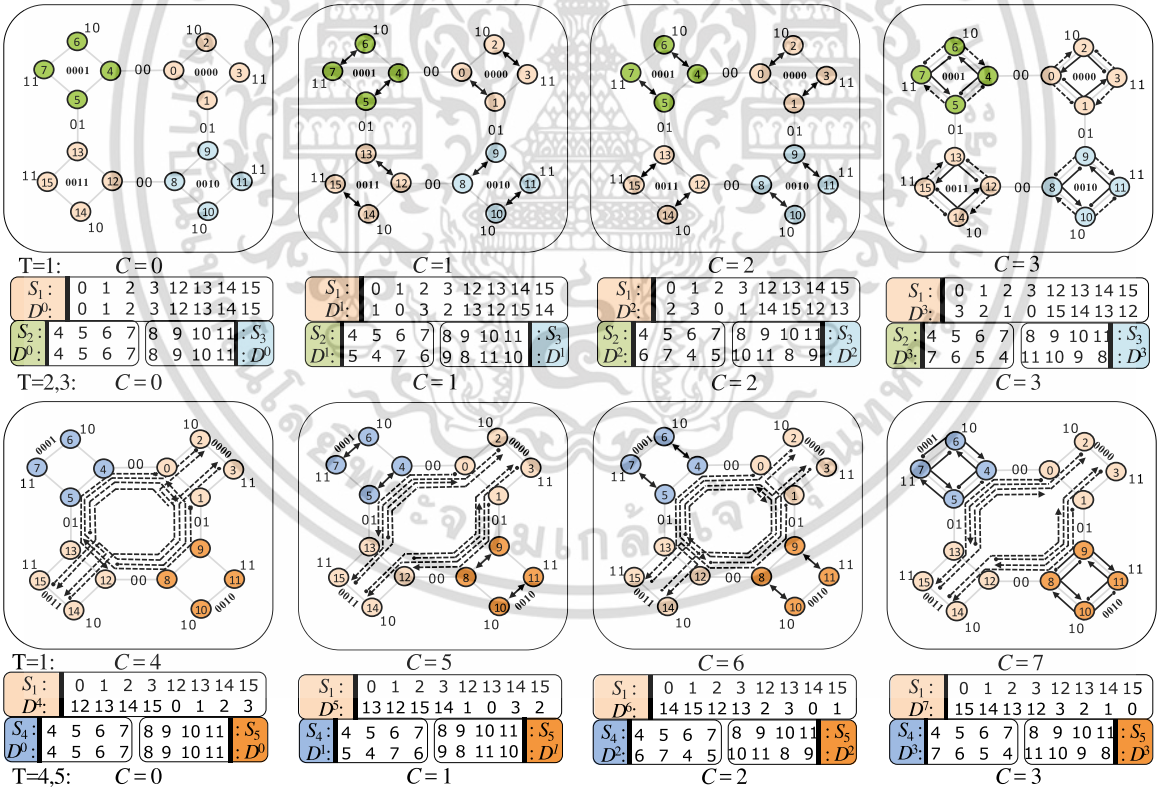


Fig. 15. An example of routing paths of ATAPE ($k = 2^{m+1}$) with $GCD_0 = 0$ (on $cross_0$) and others 4 tasks ($k = 2^m$), where S_T = source processors for task T and D^C = destination processors under k controls (C) on the HHC ($N = 64$, $m = 2$, $k \leq 8$).

2. Algorithm 6: Five cases of the complete (S, D) SP-routing through the GC-reordering of nodes in μ.

The generalized EES ($\mu = \{c_0, c_1, \ldots, c_{r-1}\}$) is the initial process of the shortest path, which is the Hamming distance (HD) of the *main-nets* of $S = (\alpha_S, \beta_S)$ and $D = (\alpha_D, \beta_D)$. The HD $\alpha^* = \alpha_S \oplus \alpha_D$ and GC-mapping in $O(|\mu|)$ are used to go through each *main-net* in μ only once to confirm the shortest external path. Then, the complete shortest path (P) is handled in five cases (of β_S, β_D, and μ) plus the proper ordering of elements in μ, proven in Theorem 5.1.

Theorem 5.1. *On the n-HHC ($N = 2^n$, $n = 2^m$+m), the path $P = \{p_0, p_1, \ldots, p_{r-1}\}$ between any two nodes $S = (\alpha_S, \beta_S)$ and $D = (\alpha_D, \beta_D)$, starting from β_S, is a shortest path if and only if $|P| = r \leq |\alpha_S \oplus \alpha_D|_{bit=1}$ and every of two successive p_i and p_j in P are closest (or minimum different bits), where $0 \leq i, j \leq r-1$ and $|P| = a$ number of elements in P.*

Proof. For the *N2N* routing from S to D ($0 \leq S, D \leq N-1$) on the n-HHC ($N = 2^n$, $n = 2^m$+m), the shortest path (P) from S to D can be obtained by Algorithm 6. If S and D reside in the same *main-net* ($\alpha_S = \alpha_D$), then applying the HC routing will be shortest. Otherwise ($\alpha_S \neq \alpha_D$), the HHC routing must go through the *main-net* and *sub-net* alternately. In this case, the generalized EES $\mu = \{c_0, c_1, \ldots, c_{r-1}\}$ with GC-mapping provides the shortest external-path. Then, five cases of (β_S, β_D, and μ) combination are determined (from β_S) before applying the *SPordering* function. In order to go through β_S and β_D only once in the shortest path P, if $\beta_S \in \mu$ (the major condition) then the next *main-net* (in P) must be connected from β_S in the *main-net* of S (in cases 1–3). In the minor condition, if $\beta_D \in \mu$ and $\beta_D \neq \beta_S$ then the *main-net* of D (in P) must be connected from β_D (cases 2 and 4). Finally, other elements in μ must be reordered by the *SPordering* function to make two successive p_i and p_j in the path P be closest in efficient $O(|\mu|^2)$ time, according to the initial GC-mapping in μ. Thus, the path P is shortest since each element of μ is utilized only once and two successive p_i and p_j in path P are closest (Algorithm 6) plus the shortest HC routing in each *sub-net*. □

5.2. Correctness of parallel N2N SP-routing

For the parallel *N2N* SP-routing, Algorithm 7 provides the symmetrical-path routing to avoid the conflict by the *dSPordering* (dynamic SP-reordering) to find the next position d (in μ) by the circular right-shift operation. This solution works (with no conflict) for any sub-system of $k \leq 2^m$ nodes, verified (in experiment) by random S and D in range $[0-(N-1)]$. In Algorithm 7, we focus on the common condition (either $\beta_S \in \mu$ or $\beta_S \notin \mu$) and the proper bidirectional SP-routing to avoid the conflict. We provide the F (forward)/B (backward) reordering and routing to avoid the conflict and maintain the shortest paths, and hence achieve the optimal ATAPE communication. The F-reordering is the (default) parallel SP-routing and the B-reordering is used to routing in the opposite directions (for full bidirectional-routing) at the same time.

5.3. Correctness of GCD partitioning and bidirectional routing of all crosses with (F, B) protocol

To solve the HHC conflict, we proposed not only the SP-routing but also the GCD (grouping of cross dual-cubes) partitioning (to handle the conflict locally in groups). In each group, any cross ($k = 2^{m+1}$ nodes) can work with no conflict, proven in Theorems 4.1–4.3. Next, the parallel (S, D) SP-routing (Algorithm 7) and GCD mapping (Algorithm 8) were verified by the ATAPE communication ($D_j^{C_i} = S_{(C_i \oplus j)}$) for the symmetrical-path routing of k controls. Each control C_i is a Hamming distance between *main-nets* of S and D, where m+1 bits of C_i are 0..00(0), 0..01(1), 0..10(2), …, and 1..11($k-1$). Theorem 5.2 verified the GCD partitioning and ATAPE mapping. Theorem 5.3 proved the (F, B) protocol for all (even, odd) crosses.

Theorem 5.2. *The parallel N2N SP-routing of XOR-based ATAPE communication ($k = 2^{m+1}$ nodes) can be performed without the conflict on the n-HHC ($N = 2^n$, $n = 2^m$+m) by the GCD partitioning and mapping with the triple-right assignment (right task, right partition, and right time).*

Proof. For parallel *N2N* SP-routing, the dynamic index d in the (forward/backward) *dSPordering* of Algorithm 7 can support the shortest paths (proven in Section 5.2). To avoid the conflict for $k = 2^{m+1}$ nodes, we use the GCD partitioning to find the proper k sources and the XOR-operation to map them to k destinations for the symmetrical-path routing. Applying Algorithm 7 (parallel *N2N* SP-routing) and Theorems 4.1–4.3 (GCD partitioning) for the ATAPE mapping could achieve the triple-right assignment. In particular, the GCD partitioning could map the right task to the right partition (for $k = 2^{m+1}$ nodes) and the GC-mapping and reordering could find the shortest paths in efficient $O(|\mu|^2)$ for routing without the conflict, confirmed by the experiment in Section 5.4. □

Our GCD partitioning allows all independent groups process the different tasks (each of $k = 2^{m+1}$ nodes) at the same time since they use the different high-order group-bits for their local-path routing. Then, the HHC conflict can be solved locally in each group, proven in Theorem 5.3. Note that small tasks ($k \leq 2^m$ nodes) can be processed independently anywhere (in *sub-nets*) at any time.

Theorem 5.3. *For GCD partitioning, all (even, odd) crosses (in any group) under the (F, B) protocol can execute at the same time by synchronized with the same control C.*

Proof. Under GCD partitioning, one cross (per group) can be used independently for $k = 2^{m+1}$ nodes with group utilization = $100/2^{M-1}$% (see Table 4), proven in Theorem 4.1.

For 50% group-utilization, even crosses (0, 2, 4, …, $2^{M-1}-2$) with the F-protocol are allowed by synchronized with the same control (C) since the symmetrical paths of adjacent crosses (i.e., (0, 1), (2, 3), …, or (even, odd) crosses) may conflict and cannot use the same F-protocol.

For 100% group utilization, we introduce the backward (B) reordering and routing to fulfill the opposite directions. Thus, we propose the (F, B) protocol (in Algorithm 7) to work with (even, odd) crosses to avoid the conflict, according to the following conditions:

- Flipped crosses cannot use the same F-protocol, Fig. 16.
- Adjacent crosses cannot use the same F-protocol, Fig. 17.
- Cross classification is made equally under cases 1 and 2.

The flipped crosses (in any group) must use the opposite rule of reordering and routing to avoid the conflict. For example ($N = 64$), Fig. 16 (left) shows the ATAPE ($C = 5$) on $cross_0$ (F-protocol). At the same time, $cross_1$ in Fig. 16 (right) can execute another ATAPE ($C = 5$) in the opposite directions (B-protocol) across their symmetrical paths. In this particular example, there is no conflict since it meets all key conditions, see confirmed results in Section 5.4.

In general, Fig. 17 ($N = 2048$ and $N = 2^{20}$), to work with the (F, B) protocol all crosses (per group) are classified into (even, odd) crosses. Fig. 17(a) displays flipped crosses in any group of $N = 2048$ ($m = 3$), which are (0:000, 5:101), (2:010, 7:111), (4:100, 1:001), and (6:110, 3:011)). Under symmetrical-path routing (between dual cubes in any cross), cross 0:000 conflicts with its adjacent cross 1:001 and its flipped cross 5:101, and hence they cannot use the same F-protocol. Similarly, cross 2:010 conflicts with crosses 3:011 and 7:111; cross 4:100 conflicts with crosses 5:101 and 1:001; and cross 6:110 conflicts with crosses 7:111 and 3:011. To avoid the conflict, if the even crosses apply the F-protocol, the odd crosses must use the B-protocol by synchronized with the same control C. Finally, for the massive $N = 2^{20}$ ($m = 4$) in Fig. 17(b), all (even, odd) flipped crosses with the (F, B) protocol, synchronized with the same control C, can be used practically. This general example meets all key conditions, confirmed (no conflict) in our experiment (in Section 5.4). □

Table 4
Group utilization of our three successive solutions by GCD partitioning and mapping with the (forward, backward) protocol.

Three cross-solutions and proper F/B protocol (in any group)	#Tasks (of k=2^{m+1} nodes) per group	Group utilization and System utilization	
1. One (independent) cross (per group) with F (forward) reordering and routing (or protocol)	1	$m = 2$	50%
		$m = 3$	12.5%
		$m = 4$	0.78%
2. Even crosses (with F protocol) (0, 2, 4,..., 2^{M-1}-2), $M = 2^{m-1}$	$2^{M-1}/2$	m = 2, 3, 4, ...	50%
3. All even crosses (F protocol) and odd crosses (B protocol)	2^{M-1}	m = 2, 3, 4, ...	100%

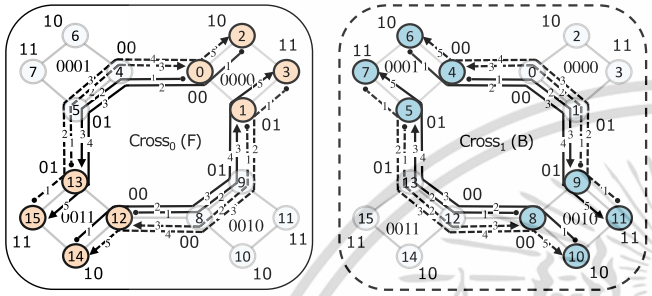


Fig. 16. A routing example of crosses in group 0 ($m = 2$): by forward (F) protocol on $cross_0$ and backward (B) protocol on $cross_1$.

5.4. Experiment to confirm no conflict on HHCs

The experiment was conducted to evaluate our conflict solution by implementing the following pseudo-code (using java programming language) for parallel (S, D) SP-routing of the ATAPE communication ($k = 2^{m+1}$ nodes per task) on the HHCs ($N = 64$ ($m = 2$), $N = 2048$ ($m = 3$), etc.).

```
ATAPE Communication (k=2^(m+1), C: iteration control (0,1,.., k-1))
  group i=0; compute GCD_i by eq. (4.1) for i=0,1,2,…, or 2^M-1; M=2^(m-1)
  call Algorithm 8 to define all sources (S) in each of 2^(M-1) crosses;
  for all sources (S) in each of all crosses (0,1,…,2^(M-1)-1) pardo
     compute D_j = S_{C⊕j} by eq. (4.2); // for j=0 to k-1 in each cross
     if (cross%2=0) direction='F' else direction='B';
     call Algorithm 7 parallelShortestPaths (P: S→D, direction: F/B);
     do HHC-Routing (S, D) and update RoutingTable (RT_{clock,edge});
  end for all; // see an Example in Fig. 18 (N = 64 (m = 2), C = 5)
  for each clock in RT, report "no-conflict" if all edges are different;
end ATAPE.
HHC-Routing (P: Shortest Path from S to D)
  for element i in path P (i = 0, 1, 2, … , |P|-1)
     p=P[i]; mNet=mainNet[p]; sNet=subNet[p];
     mainNet&subNet-routing (mNet, sNet) and local HC-routing;
     // according to the definition of the HHC network
  end for i
end SP-Routing // see an Example in Fig. 9(b) and Table 3 (N=64).
```

After assigning a group i, all source (S) nodes in each cross were defined (by Algorithm 8). Then, all S-nodes route to their destinations ($D = S_{C\oplus j}$) by Algorithm 7 and (S, D) paths were recorded in a routing table. We used that table to evaluate the ATAPE communication (to ensure that each group could execute (even, odd) crosses by the (F, B) protocol at the same time). Let a communication time utilize one clock. Under time multiplex, at any clock there is no conflict if all pairs of nodes A and B transfer messages through different edges. For instance, Table 5 displays T_1-routing (F-protocol) on $cross_0$ of group 0 ($N = 64$ ($m = 2$), $k = 8$) for ATAPE ($C = 5$). Routing from $S = [0, 1, 2, 3, 12, 13, 14, 15]$ to $D = [13, 12, 15, 14, 1, 0, 3, 2]$ was confirmed (no conflict or no shared edge(s) at any clock). At the same time, T_2-routing (B-protocol) on $cross_1$ of group 0 (Table 6, $C = 5$) from $S = [4, 5, 6, 7, 8, 9, 10, 11]$ to $D = [9, 8, 11, 10, 5, 4, 7, 6]$ was also confirmed (no conflict).

Table 5
Routing table ($N = 64$) for ATAPE-T_1 ($k = 8$, $C = 5$) on $cross_0$ (F).

Clocks	1		2		3		4		5	
S=0:0000,00	→	0001,00	→	0001,01	→	**13**:0011,01=**D**				
S=1:0000,01	→	0010,01	→	0010,00	→	**12**:0011,00=**D**				
S =2:0000,10	→	0000,00	→	0001,00	→	0001,01	→	0011,01	→	**15**:0011,11=**D**
S =3:0000,11	→	0000,01	→	0010,01	→	0010,00	→	0011,00	→	**14**:0011,10=**D**
S =12:0011,00	→	0010,00	→	0010,01	→	**1**:0000,01=**D**				
S =13:0011,01	→	0001,01	→	0001,00	→	**0**:0000,00=**D**				
S =14:0011,10	→	0011,00	→	0010,00	→	0010,01	→	0000,01	→	**3**:0000,11=**D**
S =15:0011,11	→	0011,01	→	0001,01	→	0001,00	→	0000,00	→	**2**:0000,10=**D**

Table 6
Routing table ($N = 64$) for ATAPE-T_2($k = 8$, $C = 5$) on $cross_1$ (B).

Clocks	1		2		3		4		5	
S=4:0001,00	→	0000,00	→	0000,01	→	**9**:0010,01=**D**				
S=5:0001,01	→	0011,01	→	0011,00	→	**8**:0010,00=**D**				
S =6:0001,10	→	0001,00	→	0000,00	→	0000,01	→	0010,01	→	**11**:0010,11=**D**
S =7:0001,11	→	0001,01	→	0011,01	→	0011,00	→	0010,00	→	**10**:0010,10=**D**
S =8:0010,00	→	0011,00	→	0011,01	→	**5**:0001,01=**D**				
S =9:0010,01	→	0000,01	→	0000,00	→	**4**:0001,00=**D**				
S =10:0010,10	→	0010,00	→	0011,00	→	0011,01	→	0001,01	→	7:0001,11=**D**
S =11:0010,11	→	0010,01	→	0000,01	→	0000,00	→	0001,00	→	**6**:0001,10=**D**

Table 7
Performance comparison of four routing-conflict solutions.

Routing Features	1994 [10]	2007 [20]	2013 [4]	This study
$N2N$ (S, D) routing	any S	$S = 0$	any S	any S
Parallel $N2N$ routing	✓	–	✓	✓
Conflict solution	–	✓	✓	✓
Reliable routing	–	✓	✓	✓
Maximum task-sizes	$k = 2^m$	2	$k = \lceil (m+1)/2 \rceil$	$k = 2^{m+1}$
GCD partitioning	–	–	–	✓
Parallel application	–	–	–	SP-ATAPE

Routing by other controls ($C \le k - 1$) was made without the conflict. Then, the (even, odd) crosses with the (F, B) protocol were confirmed with no conflict (see Fig. 18, parallel routing of two ATAPE tasks with $C = 5$ working on $cross_0$ (F) and $cross_1$ (B)). Note: to handle some critical I/O ports (i.e., 2-inputs → 2-outputs by passing the same node), we use either FCFS or one delay ($\Delta\tau$) with the minID first (if they come at the same time), such as on intermediate nodes 0, 1, 4, 5, 8, 9, 12, and 13 (at the end of clocks 1, 2, 3).

Table 7 presents our conflict solution on the HHCs, compared to the existing methods [4,10,20]. For parallel $N2N$ routing, we can support multiple tasks ($k \le 2^{m+1}$ nodes per task), while the existing routing [10] worked for $k \le 2^m$ nodes per task. Moreover, under GCD partitioning and mapping, our SP-ATAPE ($k \le 2^{m+1}$ nodes per task) can be applied on the HHCs without the conflict.

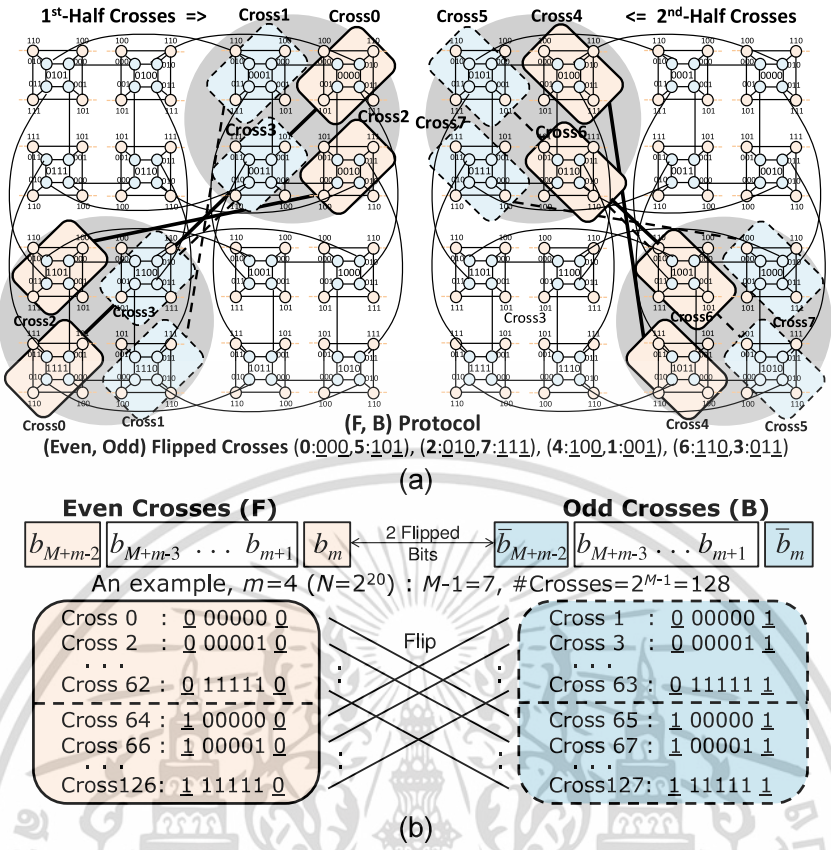


Fig. 17. (F, B) protocol for (even, odd) crosses: (a) $m = 3$ and (b) $m = 4$.

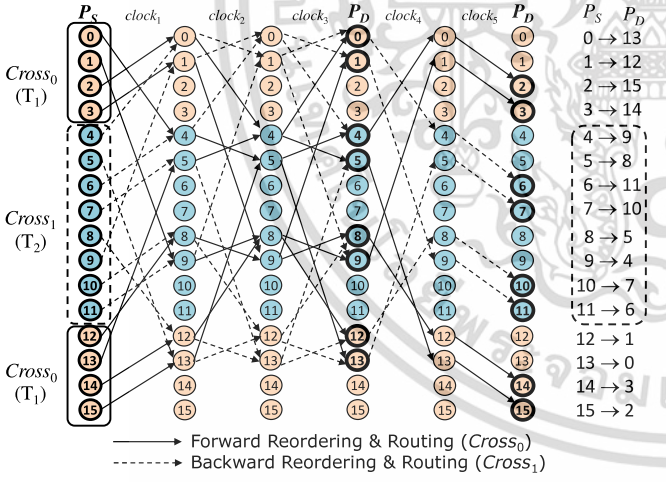


Fig. 18. Parallel routing of all nodes in $cross_0$ (0–3, 12–15) and $cross_1$ (4–7, 8–11) with $C = 5$ in ATAPE communication (T_1&T_2) in group 0.

6. Conclusion and future study

The HHC conflict-solution is one of the crucial researches for the scalable MP systems. In this study, we propose the parallel *N2N* shortest-path routing, based on our hypothesis "the shortest-path routing and the proper HHC-partitioning can reduce the conflict on the HHCs directly". Therefore, our key contribution is the GCD (grouping of cross dual-cube) partitioning (for $k = 2^{m+1}$ nodes per task), where the global conflict is reduced to handle locally in each group. Within any group, the HHC conflict is solved by mapping the (F: forward, B: backward) protocol on all (even, odd) crosses (by synchronized with the same control). This is the best conflict-solution, improved over our initial conflict-solution (allowing even crosses per group only). Finally, the experiment was conducted to evaluate our conflict solution. In this experiment, the ATAPE results confirmed that the GCD mapping could make all crosses working for $k = 2^{m+1}$ nodes per task (in any group) at the same time. Moreover, all groups allowed different tasks ($k \leq 2^{m+1}$ nodes per task) with different operations at the same time. Furthermore, the shortest-path ATAPE can be embedded on-chip for the optimal communication of many parallel applications (such as parallel FFT, parallel matrix transposition, etc.). In our future study, we will research "the cross-based combining" on the HHC networks for $k \geq 2^{m+2}$ nodes (per task).

CRediT authorship contribution statement

Nuntipat Phisutthangkoon: Conceptualization, Methodology, Writing - original draft, Art work (drawing figures), Proof of correctness, Investigation, Programming (in experiment). **Jeeraporn Werapun:** Supervision, Writing - review & editing, Art work (refining figures), Proof of correctness (validation and correction).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

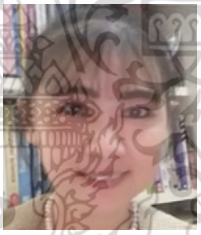
The authors wish to gratefully thank the faculty of Science, King Mongkut's Institute of Technology Lardkrabang (KMITL),

Bangkok, Thailand for the research grant to Mr. Nuntipat Phisutthangkoon, a doctoral student in Computer Science, during 2013–2015 and 2020–2022 (CW-012-2/2562). They deeply thank Phra-phrom Kunaporn (P.A. Pa-yut-tao) and his "Put-ta-tum" Buddhism handbook for fulfilling their positive thinking and exact understanding. Lastly, the merit of their work will be glorious honor to their beloved King Bhumibol Adulyadej for the great inspiration to their optimistic study.

References

[1] M. Abd-El-Barr, T.F. Al-Somani, Performance comparison of hierarchical interconnection networks, in: IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, PacRim, August 2011, pp. 191–196.
[2] M. Abd-El-Barr, F. Gebali, Relibility analysis and fault tolerance for hypercube multi-computer networks, Inform. Sci. 276 (2014) 295–318.
[3] A. Bossard, K. Kaneko, Set-to-set disjoint-path routing in perfect hierarchical hypercube, in: The Fourth International C* Conference on Computer Science and Software Engineering, Canada, 2011, pp. 51–57.
[4] A. Bossard, K. Kaneko, *k* – pairwise disjoint paths routing in perfect hierarchical hypercubes, J. Supercomput. 67 (2) (2013) 485–495.
[5] A. Bossard, K. Kaneko, S. Peng, Node-to-set disjoint-path routing inperfect hierarchical hypercubes, in: International Conference on Computational Science, ICCS 2011, Procedia Computer Science 4, 2011, pp.442–451.
[6] W.D. Hillis, The Connection Machine, MIT Press, Cambridge, MA, 1985.
[7] C. Lai, Optimal construction of all shortest node-disjoint paths in hypercubes with applications, IEEE Trans. Parallel Distrib. Syst. 23 (6) (2012) 1129–1134.
[8] C. Lai, An effcient construction of one-to-many node-disjoint paths in folded hypercubes, J. Parallel Distrib. Comput. 74 (4) (2014) 2310–2316.
[9] C. Lai, Optimal construction of node-disjoint shortest paths in folded hypercubes, J. Parallel Distrib. Comput. 102 (2017) 37–41.
[10] Q.M. Malluhi, M.A. Bayoumi, The hierarchical hypercube: A new interconnection topology for massively parallel systems, IEEE Trans. Parallel Distrib. Syst. 5 (1) (1994) 17–30.
[11] R. Petagon, J. Werapun, Embedding the optimal all-to-all personalized exchange on multistage interconnection networks^{+}, J. Parallel Distrib. Comput. 88 (2016) 15–30.
[12] R. Petagon, J. Werapun, VA-DE: Valuable ATAPE with dynamic embedding and super-pipeline scheduling on partitionable multistage interconnection network^{+}, J. Parallel Distrib. Comput. 102 (2017) 1–15.
[13] B. Prisacari, G. Rodriguez, C. Minkenberg, Generalized hierarchical all-to-all exchange patterns, in: IEEE 27th International Symposium on Parallel & Distributed Processing, IPDPS, May 2013, pp. 537–547.
[14] B. Qiao, F. Shi, W. Ji, THIN: A new hierarchical interconnection network-on-chip for SOC, in: The 7th international conference on Algorithms and architectures for parallel processing, ICA3PP'07, 2007, pp. 446–457.
[15] K. Qiu, An efficient disjoint shortest paths routing algorithm for the hypercube, in: 14th IEEE International Conference on Parallel and Distributed Systems, ICPADS'08, Dec. 2008, pp. 43–47.
[16] Y. Saad, M.H. Schultz, Topological properties of hypercubes, IEEE Trans. Comput. 37 (7) (1988) 867–872.
[17] D. Scott, Efficient all-to-all communication patterns in hypercube and mesh topologies, in: Distributed Memory Computing Conference, The Sixth, apr-1, May 1991, pp. 398–403.
[18] Y. Shi, Z. Hou, J. Song, Hierarchical interconnection networks with folded hypercubes as basic clusters, in: The Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region, Vol. 1, 2010, pp. 134–137.
[19] T. Takabatake, K. Kaneko, H. Ito, Generalized hierarchical completely-connected networks, in: Fourth International Symposium on Parallel Architectures, Algorithms, and Networks, I-SPAN'99, Jun 1999, pp. 68–73.
[20] R.Y. Wu, G.H. Chen, Y.L. Kuo, G.J. Chang, Node-disjoint paths in hierarchical hypercube networks, Inform. Sci. 117 (19) (2007) 4200–4207.
[21] Y. Yang, J. Wang, Optimal all-to-all personalized exccchange in self-routable multistage networks, IEEE Trans. Parallel Distrib. Syst. 11 (3) (2000) 261–274.
[22] Q. Zhu, J.M. Xu, X. Hou, M. Xu, On reliability of the folded hypercubes, Inform. Sci. 177 (8) (2007) 1782–1788.

Nuntipat Phisutthangkoon received the B.S. degree in Applied Mathematics in 2011 and the M.S. degree in Computer Science in 2013 from King Mongkut's Institute of Technology Ladkrabang (KMITL), Bangkok, Thailand. Mr. Phisutthangkoon is a Ph.D. candidate (in Computer Science), at KMITL, Bangkok, Thailand. His research interests include parallel algorithms, parallel and distributed computing, parallel optimization, and hierarchical interconnection networks.



Jeeraporn Werapun received the B.S. degree from Kasetsart University and the M.S. degree from Chulalongkorn University, Bangkok, Thailand. She received the M.S. degree in Computer Science in 1993 and the D.Sc. degree in Computer Engineering (Parallel and Distributed Systems) in 1999 from the George Washington University, Washington, D.C., U.S.A. Dr. Werapun is an Associate Professor of Computer Science at King Mongkut's Institute of Technology, Ladkrabang (KMITL), Bangkok, Thailand. Her research interests include parallel algorithms, parallel and distributed computing, and multistage networks.

ภาคผนวก ข

ผลงานวิจัยตีพิมพ์ที่ Parallel Computing

Journal homepage: www.journals.elsevier.com/parallel-computing

Phisutthangkoon, N. and Werapun, J. 2022. “Optimal ATAPE-Task Scheduling on Reconfigurable and Partitionable Hierarchical Hypercube Networks.” *Parallel Computing*. 111 : 102923.

Parallel Computing 111 (2022) 102923

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Optimal ATAPE task scheduling on reconfigurable and partitionable hierarchical hypercube networks

Nuntipat Phisutthangkoon *, Jeeraporn Werapun

Computer Science Department, King Mongkut's Institute of Technology, Ladkrabang, Bangkok, 10520, Thailand

ARTICLE INFO

Keywords:
Reconfigurable-and-partitionable hierarchical hypercube (RP-HHC) networks
Grouping of cross dual-cube (GCD) partitioning
Grouping of cross sub-system (GCS) combining
Optimal ATAPE (all-to-all personalized exchange) communication
Reconfigured binary tree for efficient task scheduling

ABSTRACT

The hierarchical hypercube (HHC) network ($N=2^n$, $n=2^m+m$, $m\geq 2$) is one of scalable networks for the large-scale parallel-computing systems since its implementation cost is lower than the popular hypercube (HC) network. However, the complicated HHC-routing may conflict (due to the reduction-of-edges in the hierarchical construction) and cannot be reconfigured simply for $k > 2^m$ nodes-per-task. Thus, a crucial research to fulfill the HHC-network is the conflict-free routing since the conflict is not handled easily during runtime. Recently, the HHC-conflict was solved by the shortest-path routing and the special HHC-partitioning, called the grouping-of-cross dual-cube (GCD) partitioning, for $k=2^{m+1}$ nodes-per-task. In this study, we propose the reconfigurable-and-partitionable HHC (RP-HHC) network for $2^{m+2} \leq k \leq N/2^{M-1}$ nodes-per-task ($M=2^{m-1}$) by the grouping-of-cross sub-system (GCS) combining (the optimal reconfiguring). Moreover, we present the efficient task-scheduling and processor-allocation, incorporated with the reconfigured binary-tree. Since the first-fit policy is fast but returns low-performance while the best-fit policy yields high-performance but is time-consuming, we propose the efficient best-fit driven-policy to work as fast as the first-fit policy while maintaining the high-performance. The correctness of new reconfiguring and time-complexity of task-allocation were analyzed. In experiments, all-to-all personalized exchange (ATAPE) communication was assessed to confirm no-conflict on the RP-HHC systems.

1. Introduction

In recent years, interconnection networks are playing a significant role in parallel computing systems. The efficient interconnection network provides the fast data-communication among processors and other storage-devices in high-performance computing machines. For implementing the parallel computer systems, applying either the static interconnection network or the dynamic interconnection network must be efficient. The static network affords less implementation-cost than the dynamic network by the fixed connection among processors. In the past, the static interconnection networks and corresponding routing were proposed in the literature [1,2,8-10,16,17,20,22-24] for connecting massive processors with the effective cost and communication. One of the popular static networks is the hypercube (HC) network [18] because of its attractive properties such as low diameter, efficient communication, regularity, symmetry, etc. However, the traditional HC network may not be suitable for the scalable multi-processor systems, according to its expensive cost for large N. For the scalable parallel system, the hierarchical hypercube (HHC) network ($N = 2^n$, $n = 2^m+m$, $m \geq 2$) [12] with the lower cost is more suitable for million or more processors, while retaining most of the attractive HC-properties. In practice, the hierarchical topologies dominate the parallel computing machines with a variation of level-based communications, especially for the modern high-performance computing such as AI and data-science applications in the big-data era [15].

The HHC network is one of inspiring hierarchical networks for the large-scale parallel systems. However, the complicated HHC routing may conflict due to the reduced edges in the hierarchical construction, where the conflict refers to the data collision of transferring messages through the same edge at a time. Therefore, one of the crucial researches on the HHC is to solve the routing conflict since it cannot be handled easily during runtime.

In 2007, the node-disjoint-path routing was proposed for reliable routing on the HHCs [23] and could be used to avoid the conflict for S (source) = 0. In 2011, the node-to-set disjoint-path routing [3,5] was introduced to broadcast on the HHCs and later (2013) the k-pairwise disjoint-path routing [4] was introduced for reliable parallel-routing for $k \leq (m+1)/2$ nodes. In early 2021, the parallel shortest-path routing and

* Corresponding author.
E-mail addresses: 56605012@kmitl.ac.th (N. Phisutthangkoon), jeeraporn.we@kmitl.ac.th (J. Werapun).

https://doi.org/10.1016/j.parco.2022.102923
Received 2 March 2021; Received in revised form 10 December 2021; Accepted 27 March 2022
Available online 1 April 2022

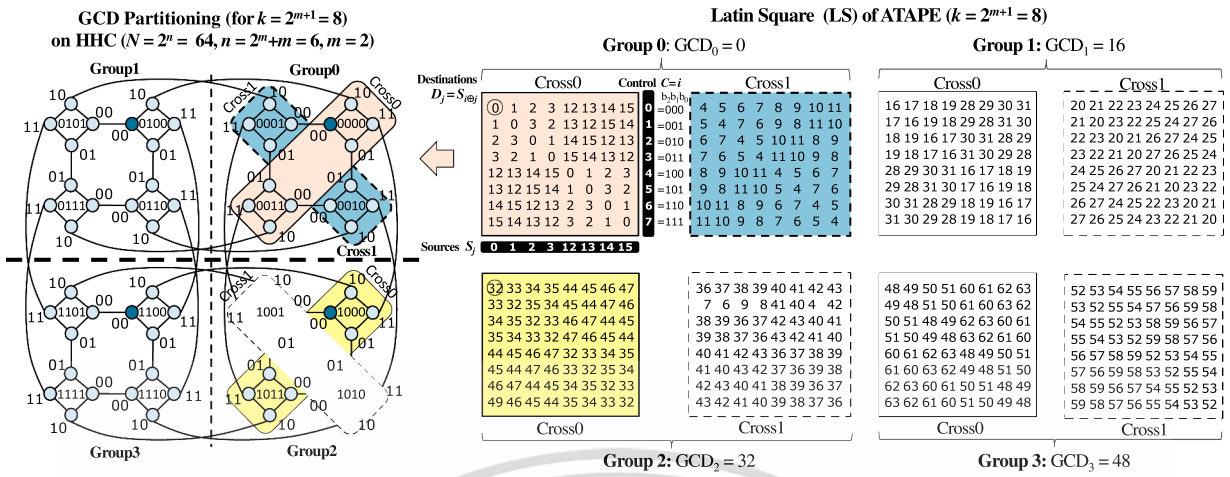


Fig. 1. An example of grouping of cross dual-cube (GCD) partitioning ($k = 2^{m+1}$ nodes per cross) on the HHC ($N = 64$, $m = 2$, $k = 8$), where GCD_g refers the first node of each group ($g = 0, 1, \ldots, 2^M\text{-}1$; $M = 2^{m-1}$) and in a cross each LS of ATAPE maps sources (S) to destinations (D).

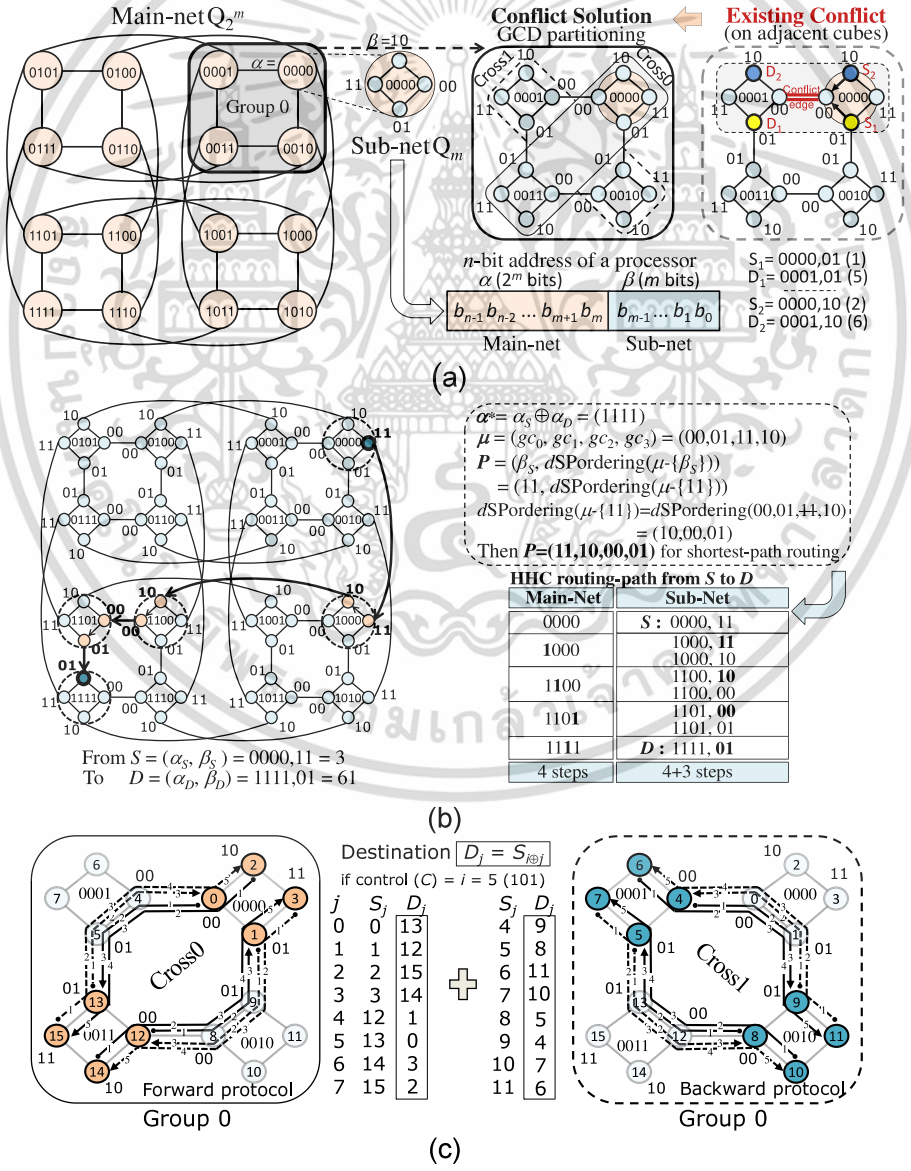


Fig. 2. a) The HHC construction (N=64, m=2) plus a proper conflict solution, b) an example of SP-routing ($S = 3$, $D = 61$), and c) the forward/backward protocol for (S, D) routing on even/odd crosses.

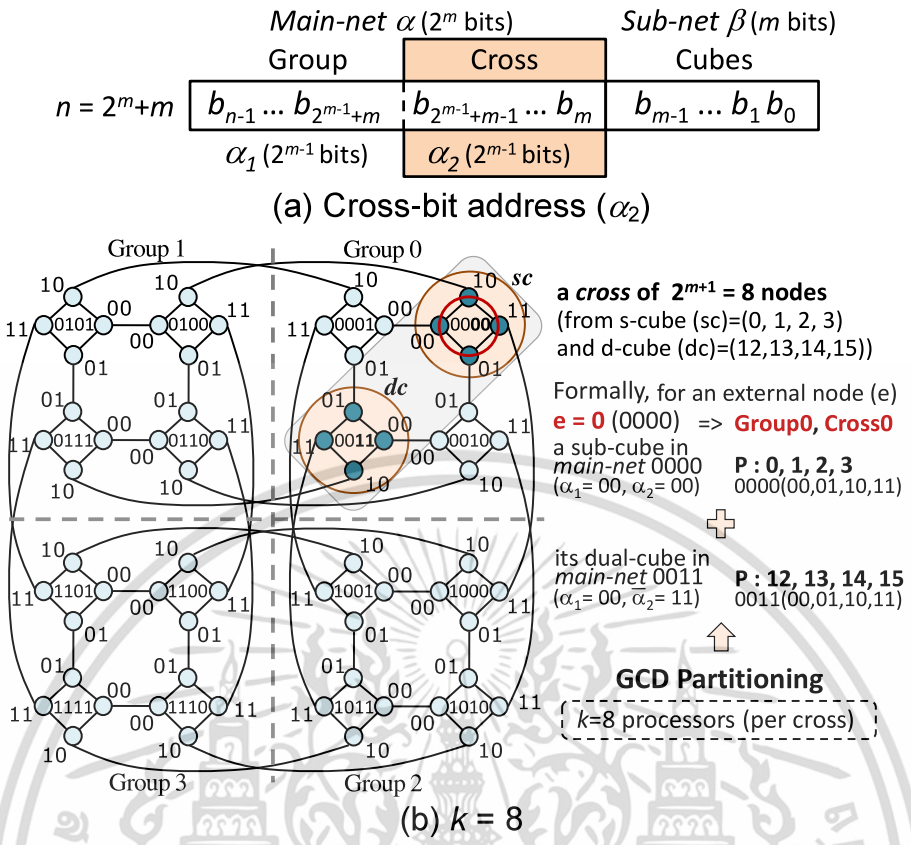


Fig. 3. An example of GCD partitioning ($N = 64$, $m = 2$, $k = 8$).

Table 1
Bitwise GCD mapping for $k = 2^{m+1} = 8$ ($N = 64$, $m = 2$).

Group α_1	Cross α_2	α_2	$\bar{\alpha}_2$	β	8 global processors
0: 00	*cross*0	00	11	00,01,10,11	0, 1, 2, 3, 12,13,14,15
	*cross*1	01	10	00,01,10,11	4, 5, 6, 7, 8, 9, 10, 11
1: 01	*cross*0	00	11	00,01,10,11	16,17,18,19, 28,29,30,31
	*cross*1	01	10	00,01,10,11	20,21,22,23, 24,25,26,27
2: 10	*cross*0	00	11	00,01,10,11	32,33,34,35, 44,45,46,47
	*cross*1	01	10	00,01,10,11	36,37,38,39, 40,41,42,43
3: 11	*cross*0	00	11	00,01,10,11	48,49,50,51, 60,61,62,63
	*cross*1	01	10	00,01,10,11	52,53,54,55, 56,57,58,59

the grouping of cross dual-cube (GCD) partitioning [15] were proposed to solve the HHC conflict directly for $k \leq 2^{m+1}$ nodes per task. In theory, a challenging research on the HHC networks is to solve the HHC conflict absolutely for $2^{m+1} < k < N$ nodes per task.

In practice, the parallel computing machines should also support the effective task scheduling as well as the processor allocation (PA) for efficiently mapping the incoming tasks to the proper sets of processors with no conflict. For instance, on the popular HC networks the set of chosen processors are usually a direct sub-cube to maintain the efficient system utilization by minimizing the internal and external fragmentation. In addition, beside the efficient PA and task scheduling, the reconfigured partitions must be realized in efficient time.

In the past, many PA strategies were proposed in the literatures by employing the useful data structures (i.e., lists, trees, etc.) for the particular networks [6,7,11,21,25]. On the HC networks, the binary tree was frequently used to build the efficient PA-recognition and task-scheduling since the binary tree structure could maintain the relation of the direct sub-cubes as well as the indirect sub-cubes systematically. According to the hierarchical manner of the tree, it is efficient to handle the allocation of parallel tasks on particular sub-systems. It is also efficient to handle the deallocation of finished tasks and gather them with other free nodes (back to the root) to maintain the maximum free space. On the HHC networks, the PA and task scheduling (with no routing conflict) cannot be handled easily, especially for the large tasks ($k > 2^m$ nodes per task). Thus, the efficient PA and task scheduling on the HHCs (with conflict-free routing) is a crucial research.

In practice, a well-known all-to-all personalized exchange (ATAPE) communication can be used to verify the HHC conflict since the ATAPE process requires k iterations ($k < N$) and at least $k/2$ iterations involve the dense communications among k pairs of (S, D) processors.

The optimal ATAPE [13,14,19] is one of the most solid communications in parallel computing because of its important role in many applications (i.e., all-to-all broadcasting, the matrix transposition, the fast Fourier transformation, etc.). In the ATAPE communication, every of k ($\leq N$) processors sends a distinct message to every other processors within k rounds (or iterations). The XOR-based ATAPE works systematically along n-dimensional routing on the HC networks but it cannot work properly on the HHC networks when $k > 2^m$. Recently, the shortest-path ATAPE [15], incorporated with the grouping of cross dual-cube (GCD) partitioning, was proposed to solve the conflict on the partitionable HHCs for $k = 2^{m+1}$ nodes per task, see Fig. 1 ($N = 64$ ($m = 2$), $k = 8$) and see more detail of the GCD partitioning in Section 2.2. However, when k is scaling up (more than 2^{m+1} nodes), the requested partition ($k > 2^{m+1}$ nodes) cannot work on the HHCs.

In this study, we propose the reconfigurable and partitionable HHC (RP-HHC) networks in order to support the large tasks (for k up to $N/2^{M-}$

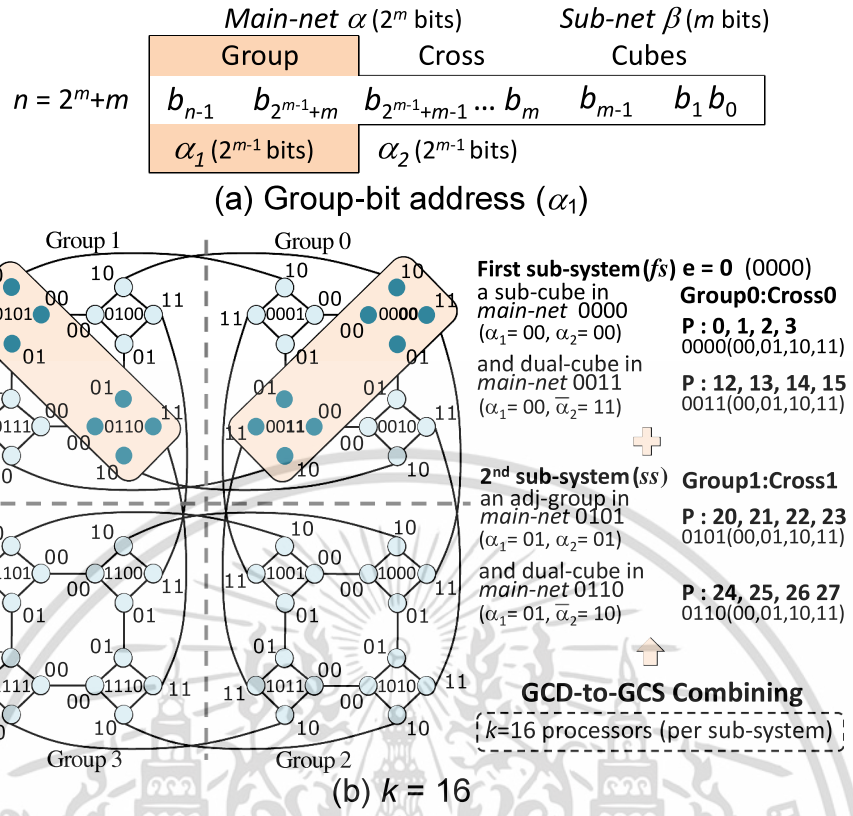


Fig. 4. An example of GCD-GCS combining ($N = 64$, $m = 2$, $k = 16$).

[1] nodes per task; $M = 2^{m-1}$) without the conflict (such as k up to $N/2$ nodes for $m = 2$ ($N = 64$) and k up to $N/8$ nodes for $m = 3$ ($N = 2048$), etc.). Our innovation is the efficient embedding of the bitwise GCD (grouping of cross dual-cube) partitioning and GCS (grouping of cross sub-system) combining to fulfill the RP-HHC for $k \le N/2^{M-1}$ nodes per task. Moreover, our additional contribution is the efficient task-scheduling and processor allocation by employing the reconfigured binary-tree structure. In particular, two effective allocation strategies are studied: 1. the first-fit driven-policy (is fast but low-performance) and 2. the best-fit driven-policy (with high-performance but is time consuming). Then, the efficient best-fit strategy (with the accumulated sum of utilized processors in each of tree nodes) is presented in O(N), as fast as the first-fit strategy, while maintaining the high-performance. In theory, the time complexity (O(N)) of the proposed best-fit strategy is analyzed and the correctness of the GCD-to-GCS reconfiguring for a variety of task sizes ($k \le N/2^{M-1}$ nodes per task) is proven. In experiments, the solid ATAPE communication is assessed to verify the parallel (S, D) routing on the HHC partitions ($\le N/2^{M-1}$ nodes) and results must confirm no conflict.

The rest of this paper is organized as follows: Section 2 addresses the related work in parallel shortest-path routing (to avoid the conflict) on the HHCs and the proper HHC partitioning (for $k \le 2^{m+1}$ nodes per task). Section 3 proposes the reconfigurable and partitionable HHCs (RP-HHCs) for a variety of reconfigured partitions ($2^{m+1} \le k \le N/2^{M-1}$ nodes per task). Section 4 presents the ATAPE communication on the RP-HHCs and the reconfigured binary-tree for the efficient task scheduling. Section 5 performs the experiments to verify the conflict-free routing on the HHC partitions by applying the ATAPE communication. Section 6 discusses the conclusion of this study.

2. Previous work

For solving the conflict on the HHC networks, the parallel shortest-path routing and the proper HHC partitioning were reviewed in Sections 2.1 and 2.2.

Definition 1. The hierarchical hypercube (HHC) networks ($N = 2^n$, $n = 2^m+m$, $m \ge 2$) [12] can be built by starting with a 2^m–hypercube (Q_{2^m}), called the *main–net*, and then replaces each node of Q_{2^m} with an m-hypercube (Q_m), called the *sub–net* for total 2^{2^m+m} nodes, see Fig. 2(a) ($N = 64$, $n = 6$, $m = 2$). Each processor has a unique n-bit address of 2^m bits (in the *main–net*) and m bits (in the *sub–net*).

2.1. Parallel shortest-path routing on HHCs

The advantage of the HHC network is the lower construction-cost than the HC network but its complicated routing may conflict (i.e., data collision of transferring messages (in parallel) through the same edge at the same time). For example, Fig. 2(a) shows a routing conflict among processors in a regular partition of two (adjacent) cubes ($k = 2^{m+1} = 8$ (nodes 0 – 7)), where a parallel communication of two paths ($S_1 = 1$ to $D_1 = 5$ and $S_2 = 2$ to $D_2 = 6$) have a conflict at the edge $e = (u, v)$, $u = (0000,00)$ and $v = (0001,00)$.

In early 2021, N. Pisutthangkoon et al. [15] proposed the parallel shortest paths (Algorithm 1) and the proper HHC partitioning (Section 2.2) to avoid the routing conflict on the HHC networks. In that study, Algorithm 1 creates the shortest external-path first from the *main-nets* (α) of any source ($S = (\alpha_S, \beta_S)$) to any destination ($D = (\alpha_D, \beta_D)$), according to the Humming distance between them ($\alpha^* = \alpha_S \oplus \alpha_D = (b_{n-1}, b_{n-2}, \ldots, b_m)$). If $b_i = 1$ the index i is included to the external edge sequence

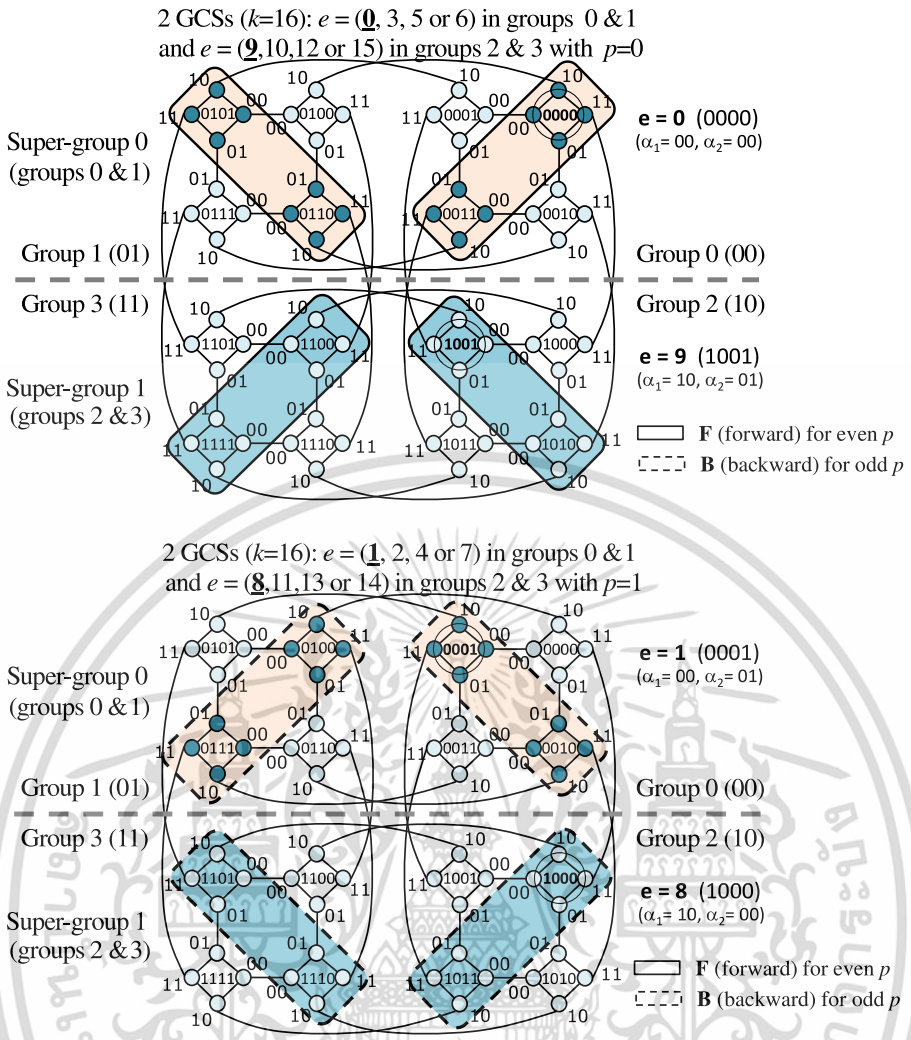


Fig. 5. Four reconfigured sub-systems with $k = 2^{m+2}$ ($N = 64$, $m = 2$).

Table 2
Bitwise GCS mapping for $k=2^{m+2}$ ($N = 64$, $m = 2$, $e = 0,1$).

Group α_1 / $e = 0$	Cross α_2	α_2	$\bar{\alpha}_2$	β	16 global processors
0: 00	*cross*0	00	11	00,01,10,11	0, 1, 2, 3, 12,13,14,15
1: 01	*cross*1	01	10	00,01,10,11	20,21,22,23, 24,25,26,27
$e = 1$					
0: 00	*cross*1	01	10	00,01,10,11	4, 5, 6, 7, 8, 9,10,11
1: 01	*cross*0	00	11	00,01,10,11	16,17,18,19, 28,29,30,31

Table 3
Bitwise GCS mapping for $k=2^{m+2}$ ($N = 64$, $m = 2$, $e = 8,9$).

Group α_1 / $e = 8$	Cross α_2	α_2	$\bar{\alpha}_2$	β	16 global processors
2: 10	*cross*0	00	11	00,01,10,11	32,33,34,35, 44,45,46,47
3: 11	*cross*1	01	10	00,01,10,11	52,53,54,55, 56,57,58,59
$e = 9$					
2: 10	*cross*1	01	10	00,01,10,11	36,37,38,39, 40,41,42,43
3: 11	*cross*0	00	11	00,01,10,11	48,49,50,51, 60,61,62,63

(EES) $\mu = (gc_0, gc_1, \ldots, gc_{r-1})$ by mapping to the m-bit Gray code. Finally, the best order of the shortest path P (returned from Algorithm 1) is obtained by reordering all elements of μ in $O(r^2)$; $r = |\mu|$, see Fig. 2(b) for the SP routing (from $S = 3$ to $D = 61$). The key contribution of that study is the forward (F) / backward (B) reordering for the conflict-free SP-routing and the cross-based partitioning for full bidirectional routing on all even/odd crosses (in Section 2.2).

Definition 2. An m–bit Gray-code G_m [12] is a binary code $G_m = (0G_{m-1}, 1G^*_{m-1})$, where two adjacent node-numbers differ by only one bit and G^*_m is the reverse order of G_m. For instance, $G_1 = (0, 1)$, $G_2 = (0G_1, 1G^*_1) = (00, 01, 11, 10)$ and $G_3 = (0G_2, 1G^*_2) = (000, 001, 011, 010, 110, 111, 101, 100)$.

2.2. Proper partitioning on partitionable HHCs

For arbitrary $0 \le S, D \le N$-1, Algorithm 1 can support the conflict-free routing in any set of 2^m nodes. For a request 2^{m+1}, the regular nodes (0 to 2^{m+1}-1) on two cubes of the regular partitioning caused a conflict (see Fig. 2(a)). To solve that conflict, the grouping of cross dual-cube (GCD) partitioning [15] was proposed (see Theorem 1). That new partitioning can avoid the conflict by reforming 2^M independent groups ($M = 2^{m-1}$) for SP-routing (in each group).

Theorem 1. Under GCD partitioning, the minimum size of each independent group must be equal to 2^{M+m} nodes to reform 2^{M-1} dependent crosses (per group). A cross (2^{m+1} nodes) consists of two dual-cubes

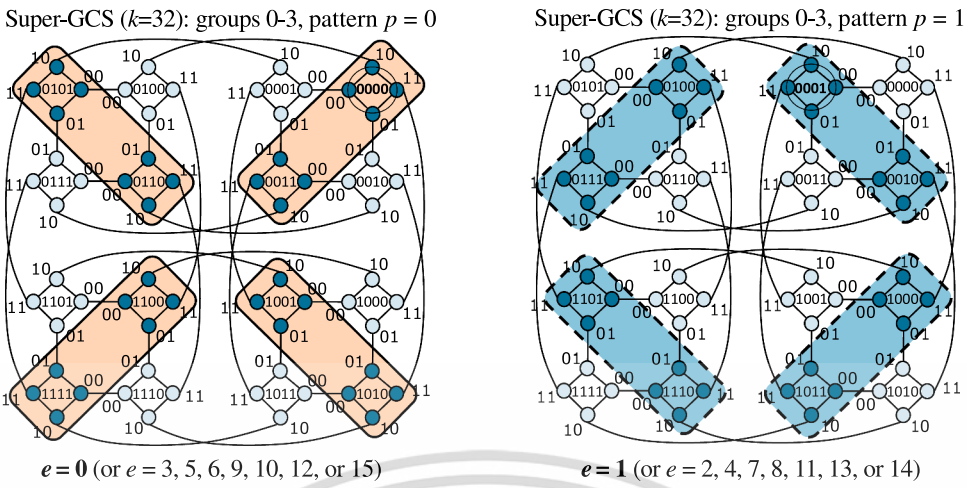


Fig. 6. An example of two reconfigured patterns ($k = N/2$, $N = 64$).

Table 4
Super-GCS mapping for $k = 2^{m+3} = 32$ ($N = 64$, $m = 2$).

Group α_1 $e = 0$	Cross α_2	α_2	$\bar{\alpha}_2$	β	32 global processors
0: 00	*cross*0	00	11	00,01,10,11	0, 1, 2, 3, 12,13,14,15
1: 01	*cross*1	01	10	00,01,10,11	20,21,22,23, 24,25,26,27
2: 10	*cross*1	01	10	00,01,10,11	36,37,38,39, 40,41,42,43
3: 11	*cross*0	00	11	00,01,10,11	48,49,50,51, 60,61,62,63
$e = 1$					
0: 00	*cross*1	01	10	00,01,10,11	4, 5, 6, 7, 8, 9,10,11
1: 01	*cross*0	00	11	00,01,10,11	16,17,18,19, 28,29,30,31
2: 10	*cross*0	00	11	00,01,10,11	32,33,34,35, 44,45,46,47
3: 11	*cross*1	01	10	00,01,10,11	52,53,54,55, 56,57,58,59

(each of 2^m adjacent nodes) that can communicate without the conflict.

In any group (of the HHC), the global processors in each of all crosses can be pre-defined by Algorithm 2. For each of all groups (g), the starting node GCD_g was derived by Eq. (2.1) to be used in Algorithm 2.

$$GCD_g = GCD_{g-1} + 2^{M+m} \text{ and } GCD_0 = 0; g = 1, 2, .., 2^{M-1} \quad (2.1)$$

For example, Fig. 1 shows all GCD_g ($0 \le g < 4$) = 0, 16, 32, 48 (the starting node of group g) to define the global processors for $k = 8$ (to work for local nodes 0–7 in the user program) on the HHC ($N = 64$). In each group, cross c and its sources (S) are created by Algorithm 2. For $GCD_0 = 0$, a number of crosses is equal to $2^{M-1} = 2$. In group 0, all sources of cross 0 are (0, 1, 2, 3, 12, 13, 14, 15) with dCube1 = GCD_0 = 0 and dCube2 = 16 – 4 = 12 and those of cross 1 are (4, 5, 6, 7, 8, 9, 10, 11) with dCube1 = 4 and dCube2 = 12 – 4 = 8. In that study, the ATAPE communication was used to map k sources (S_0, S_1, .., S_j, .., S_{k-1}) of any cross (Algorithm 2) to the right destinations (D_0, D_1, .., D_j, .., D_{k-1}), by Eq. (2.2), in k rounds of dimensional controls (C = 000, 001, 010, …, 111), see all (S, D) mappings in Fig. 1.

$$D_j = S_{(i \oplus j)}; \text{ where } C = i \text{ and } i, j = 0, 1, 2, \ldots, k-1 \quad (2.2)$$

In each group, if even crosses use F-protocol, odd crosses can use B-protocol (with the same C), proven in [15]. See an example in Fig. 2(c) for C = 5 (= 101) and $D_j = S_{(5 \oplus j)}$.

3. Reconfigurable and partitionable HHC networks by GCD-to-GCS reconfiguring

In this study, we propose the reconfigurable and partitionable HHC (RP-HHC) networks by embedding the proper HHC-partitioning and combining for $k \le N/2^{M-1}$ nodes per task. Sections 3.1 – 3.3 present the bitwise GCD (grouping of cross dual-cube) partitioning ($k = 2^{m+1}$), bitwise GCS (grouping of cross sub-system) combining ($k = 2^{m+2}$), and super-GCS combining ($k = 2^{k+3}$, …, $N/2^{M-1}$).

3.1. Bitwise GCD partitioning

In GCD partitioning [15], the HHC network was partitioned into 2^M independent groups. In each group, two dual-cubes (2×2^m nodes) were formed in a cross structure to avoid the routing conflict. In GCD pre-processing, Algorithm 2 defined 2^{m+1} sources in each of 2^{M-1}

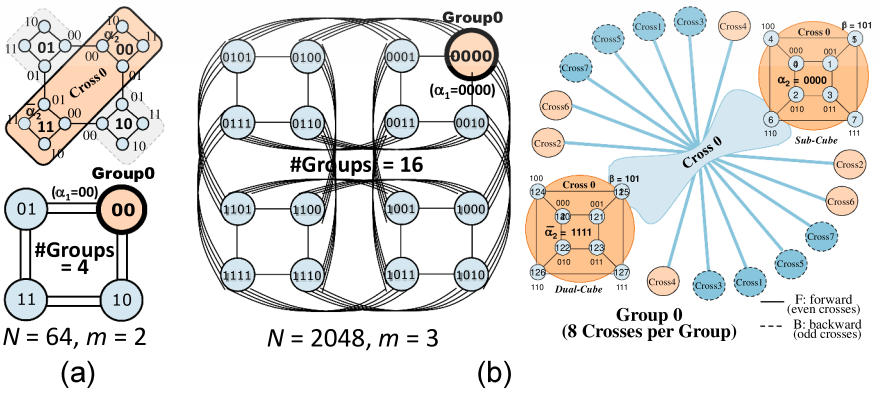


Fig. 7. Group-based partitioning (2^M groups, 2^{M-1} crosses per group, $k = 2^{m+1}$ nodes per cross): a) $N = 64$, $m = 2$ and b) $N = 2048$, $m = 3$.

Table 5
Bitwise GCD mapping for $k = 2^{m+1}$ ($N = 2048$, $m = 3$).

Group α_1	Cross α_2	α_2	$\overline{\alpha}_2$	β	16 global processors
0: 0000	*cross0*	0000	1111	000 - 111	0 - 7, 120 - 127
	cross1	0001	1110	000 - 111	8 - 15, 112 - 119
	cross2	0010	1101	000 - 111	16 - 23, 104 - 111
	cross3	0011	1100	000 - 111	24 - 31, 96 - 103
	cross4	0100	1011	000 - 111	32 - 39, 88 - 95
	cross5	0101	1010	000 - 111	40 - 47, 80 - 87
	cross6	0110	1001	000 - 111	48 - 55, 72 - 79
	cross7	0111	1000	000 - 111	56 - 63, 64 - 71
…	…	…	…	…	…
15: 1111	*cross0*	0000	1111	000 - 111	1920-1927, 2040-2047
	…	…	…	…	…
	cross7	0111	1000	000 - 111	1976-1983, 1984-1991

dependent crosses (per group) for all 2^M groups before working with the parallel SP-routing (Algorithm 1). However, that GCD pre-processing consumes computing time and space.

In this study, we redesign the efficient GCD partitioning ($k = 2^{m+1}$) before applying in the proper GCS combining for $k > 2^{m+1}$ nodes (in Sections 3.2 – 3.3). The bitwise-GCD partitioning is proposed in O(1) to recognize any specific cross ($k = 2^{m+1}$) at runtime, defined in Definition 3. See a practical example ($N = 64$, $k = 2^{m+1} = 8$) in Fig. 3.

Definition 3. A cross of two dual cubes (in a group) on the HHC ($N = 2^n$, $n = 2^m+m$) can be defined from an external-node e (in one cube) on the *main-net* α, Fig. 2(a). The input node e can be represented by $\alpha = \alpha_1\alpha_2$ bits (α_1 = the group bits, α_2 = the cross bits), Fig. 3(a). Then, another cube (of that cross) can be determined directly in Eq. (3.1).

$$\text{From an input, sub}-\text{cube } (sc)\ e = \alpha_1\alpha_2,\ \text{the output dual}-\text{cube } (dc)\ d = \alpha_1\overline{\alpha}_2 \tag{3.1}$$

Fig. 3(b) displays the GCD partitioning ($k = 2^{m+1} = 8$) from an external-node $e = 0$ in *main-net* $\alpha = \alpha_1\alpha_2 =$ 00 00 (group $\alpha_1 =$ 00, cross $\alpha_2 =$ 00). The dual cube d is in *main-net* $\alpha_1\overline{\alpha}_2 =$ 00 11 (the same group $\alpha_1 =$ 00 but $\overline{\alpha}_2 =$ 11). In this cross, all processors are $\alpha_1\alpha_2$ xx = 00 00 (00, 01, 10, 11) = (0, 1, 2, 3) and $\alpha_1\overline{\alpha}_2$ xx = 00 11 (00, 01, 10, 11) = (12, 13, 14, 15). In addition, Table 1 shows the GCD mapping for $k = 2^{m+1} = 8$ nodes by a given $e = \alpha = \alpha_1\alpha_2$ of group $i = \alpha_1$ (= 0, 1, 2, 3) for cross 0 and cross 1 on the RP-HHC ($N = 64$, $m = 2$).

The correctness of the cross in Eq. (3.1) is similar to the cross structure [15], defined in Theorem 1 (to avoid the conflict). In this study, term α_1 (2^{m-1} bits) refers to the group address. Terms α_2 (2^{m-1} bits) and $\overline{\alpha}_2$ (2^{m-1} bits) refer to the cross addresses of two dual cubes. For each cross, two cubes (*sc, dc*) are defined by two *main-nets* $\alpha_1\alpha_2$ and $\alpha_1\overline{\alpha}_2$. The dual cube d is in the same group α_1 and invert $\overline{\alpha}_2$ for the shortest path (SP) routing in that group. All processors of the sub cube (*sc*) and dual cube (*dc*) are $\alpha_1\alpha_2$ xx..x and $\alpha_1\overline{\alpha}_2$ xx..x, where xx..x denotes all binary addresses of the *sub-net* (i.e., xx = 00, 01, 10, 11). In the cross structure, 2^{m+1} nodes can communicate with no conflict.

3.2. Bitwise GCD-to-GCS combining

The GCD-to-GCS combining is proposed to formulate the cross-based sub-system (for $k = 2^{m+2}$ nodes) from two adjacent groups, residing in a super-group (s-group), in order to work with the parallel SP-routing (Algorithm 1).

Our GCD-to-GCS reconfiguring (Definition 4) is composed of two crosses of the adjacent groups, see Fig. 4 ($N = 64$, $k = 2^{m+2} = 16$). Each combined sub-system (k) can work without the conflict due to the reconfigured pattern p (Definition 5) of the right crosses in the right groups.

Definition 4. A combined sub-system (of two crosses) is formulated from an input $e = \alpha = \alpha_1\alpha_2$ (in group α_1, cross α_2), Fig. 4(a). First, the (nearest) *adj*-group (in the least significant bit) of group α_1 is defined in Eq. (3.2) in order to retain the shortest path in the HHC routing. Each cross (j) in each of two groups is defined in Eq. (3.3).

$$\text{The(nearest)}adj-\text{group (of group } \alpha_1) = \alpha_1 \oplus 0..01 \tag{3.2}$$

$$\text{The cross } j \text{ (of group } i) = i \oplus p; \text{if } j < 2^{M-1};\ M = 2^{m-1}$$

$$\text{otherwise, the cross } j \text{ (of group } i) = \overline{i \oplus p} \tag{3.3}$$

Definition 5. The initial pattern (p) is $p = \alpha_1 \oplus \alpha_2$ and if $p \geq 2^{M-1}$ then adjust $p = \overline{p}$ ($< 2^{M-1}$) for $0 \leq p \leq 2^{M-1}$-1. In this study, the dynamic function (p) can be determined systematically to avoid the routing conflict on the RP-HHCs by mapping to the proper crosses (in each group) or mapping to the combined sub-systems (in each s-group).

For example ($N = 64$, $m = 2$), Fig. 4(b) displays the reconfigured sub-system ($k = 2^{m+2} = 16$ nodes) from an input $e = \alpha = \alpha_1\alpha_2 =$ 00 00 (group $\alpha_1 =$ 00, cross $\alpha_2 =$ 00). After applying the GCS combining in Eqs. (3.2) – (3.3) to determine the first sub-system (*fs*) of group 0 and the second sub-system (*ss*) of the *adj*-group i, all processors in *fs* (in the *main-net* 0000) are $\alpha_1\alpha_2$ xx = 00 00 (00, 01, 10, 11) = (0, 1, 2, 3) and $\alpha_1\overline{\alpha}_2$ xx = 00 11 (00, 01, 10, 11) = (12, 13, 14, 15). All processors of *ss* in the *adj*-group ($i = \alpha_1 \oplus 01 = 00 \oplus 01 = 01$, $p = \alpha_1 \oplus \alpha_2 = 00 \oplus 00 = 00$, and cross $j = i \oplus p = 01 \oplus 00 = 01$ (in the *main-net* 0101)) are $\alpha_1\alpha_2$ xx = 01 01 (00, 01, 10, 11) = (20, 21, 22, 23) and $\alpha_1\overline{\alpha}_2$ xx = 01 10 (00, 01, 10, 11) = (24, 25, 26, 27). Fig. 5 shows all of four reconfigured sub-systems ($N = 64$) for $k = 16$ nodes from $2^{M-1} = 2$ independent s-groups (i.e., s-group 0 (0-1), s-group 1 (2-3)).

Table 2 and Fig. 5 show the GCS combining ($k = 16$) of $e = 0$ and $e = 1$ (from groups 0 and 1). From a given $e = 0$ ($\alpha = \alpha_1\alpha_2 =$ 0000, group $\alpha_1 =$ 00, cross $\alpha_2 =$ 00), the *adj*-group i is $i = \alpha_1 \oplus 01 = 00 \oplus 01 = 01$ ($p = \alpha_1 \oplus \alpha_2 = 00 \oplus 00 = 0$, cross $j = i \oplus p = 01$). For $e = 1$, the reconfigured sub-system is constructed from $e = 1$ (or $\alpha = \alpha_1\alpha_2 =$ 0001, group 00, cross 01, $p = 00 \oplus 01 = 01$ for *fs*) and the computed *ss* is in the *adj*-group $i = 00 \oplus 01 = 01$, cross $j = i \oplus p = 01 \oplus 01 = 00$.

Table 3 and Fig. 5 show the GCS combining ($k = 16$) of $e = 8$ and $e = 9$ (from groups 2 and 3). For $e = 8$ (*main-net* 1000 (group 10, cross 00), $p = \alpha_1 \oplus \alpha_2 = \overline{10} = 01$ for *fs*), its corresponding *ss* is in the *adj*-group $i = 10 \oplus 01 = 11$ and cross $j = \overline{i} \oplus p = \overline{10} = 01$. Therefore, all processors ($k = 16$) are *fs* (1000xx, 1011xx) = (32, 33, 34, 35, 44, 45, 46, 47) and *ss* (1101xx, 1110xx) = (52, 53, 54, 55, 56, 57, 58, 59). Similarly, for $e = 9$ (*main-net* 1001, $p = \alpha_1 \oplus \alpha_2 = \overline{11} = 00$) in group 10, its *adj*-group $i = 10 \oplus 01 = 11$ and cross $j = \overline{i \oplus p} = \overline{11} = 00$. Thus, all processors ($k = 16$) are *fs* (1001xx, 1010xx) = (36 - 39, 40 - 43) and *ss* (1100xx, 1111xx) = (48 - 51, 60 - 63).

3.3. Bitwise super-GCS combining

In order to fulfill the GCS combining (our innovation), we propose the super-GCS combining (for $2^{m+3} \leq k \leq N/2^{M-1}$ nodes) from the collaborating ($k/2^{m+1}$) crosses of the right groups (one cross per group), defined in Definition 6. First, the reconfigured sub-system ($k = 2^{m+3}$ nodes) is formulated from $k/2^{m+1}$ crosses with two cubes (*sc, dc*) per cross (or 4 crosses (32 processors) on $N = 64$ ($m = 2$)), see Fig. 6 (two patterns ($p = 0, 1$) for $k = 32$ processors).

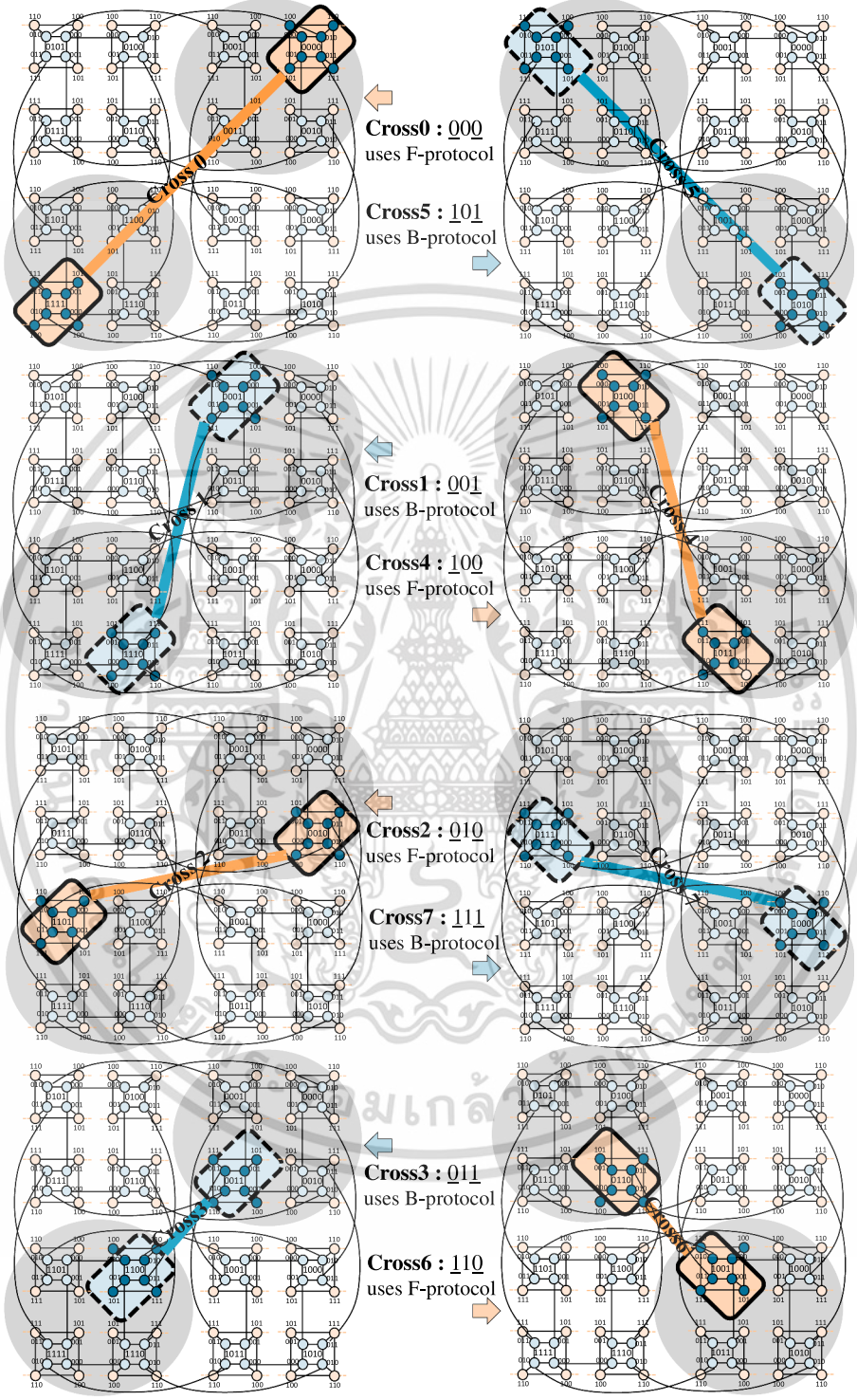


Fig. 8. An example of GCD partitioning ($k = 16$ nodes per cross) on $N = 2048$, $m = 3$ to work with (F, B) protocol (in Algorithm 1).

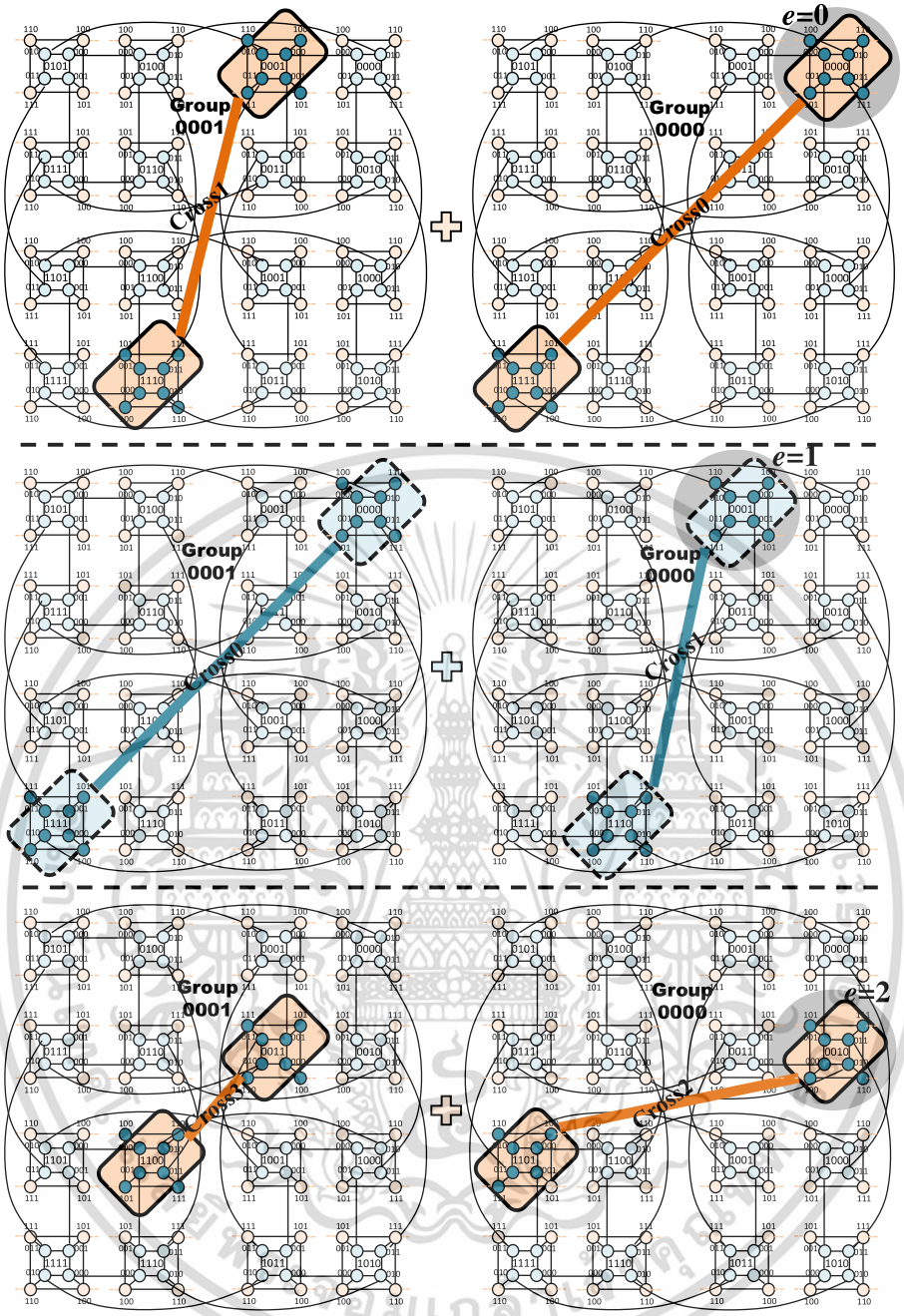


Fig. 9. An example of GCS combining of $k = 2^{m+2} = 32$ nodes ($N = 2048$, $m = 3$) with $e = 0$ - 2 (for $p = 0$ - 2, three of eight patterns).

Definition 6. The super-GCS combining is reconfigured from $k/|\text{cross}| = 2^s$ groups in the s-group (one cross per group; $s = 2, 3, \ldots, M = 2^{m-1}$). From an input e (or a *main-net* $\alpha = \alpha_1\alpha_2$, group α_1, cross α_2, pattern $p = \alpha_1 \oplus \alpha_2$), all *main-nets* (in the s-group) are defined as follows:

$$b_{n-1}..b_{n'+1}b_{n'}\text{xx}\ \alpha_{2i} \text{ for } k = 2^{m+3}\left(\text{in } 2^2\text{groups } (i)\right) \tag{3.4}$$

$$b_{n-1}..b_{n'+1}\text{xxx}\ \alpha_{2i} \text{ for } k = 2^{m+4}\left(\text{in } 2^3\text{groups } (i)\right) \tag{3.5}$$

$$, \ldots, \ \text{xxx..x}\ \alpha_{2i} \text{ for } k = N/2^{M-1}\left(\text{in } 2^M\text{groups } (i)\right) \tag{3.6}$$

where x..x reforms binary strings (i.e., xx = 00, 01, 10, 11) from the least significant bit of the partial bits of group α_1 (to create all groups (i)) and α_{2i} is cross j (of group i) $= i \oplus p$ if $j < 2^{M-1}$ from Eq. (3.3) or cross $j = \overline{i \oplus p}$ if $j \geq 2^{M-1}$.

For example, Fig. 6 and Table 4 illustrate the super-GCS combining ($k = 2^{m+3} = N/2 = 32$ on $N = 64$) for $e = 0$ (the first reconfigured sub-system) and $e = 1$ (another sub-system). For $e = 0$ (*main-net* $\alpha = \alpha_1\alpha_2 =$ 0000, group $\alpha_1 =$ 00, cross $\alpha_2 =$ 00, $p = \alpha_1 \oplus \alpha_2 =$ 0), the s-group is composed of four groups (00 – 11) by Eq. (3.6), where xxα_{2i} means (00α_{2i}, 01α_{2i}, 10α_{2i}, 11α_{2i})) and α_{2i} means each cross j (of group i), computed by Eq. (3.3). Then, eight *main-nets* from (group 0, cross 0 (00: 11)), (group 1, cross 1 (01: 10)), (group 2, cross 1 (01: 10)), and (group 3, cross 0 (00: 11)) are (00(00, 11), 01(01, 10), 10(01, 10), 11(00, 11)) for $S =$ (0 - 3, 12 - 15, 20 - 23, 24 - 27, 36 - 39, 40 - 43, 48 - 51, 60 – 63). For $e = 1$ (*main-net* $\alpha = \alpha_1\alpha_2 =$ 0001, $p = \alpha_1 \oplus \alpha_2 = 1$), eight *main-nets* are (00α_{2i}, 01α_{2i}, 10α_{2i}, 11α_{2i}) = (00(01, 10), 01(00, 11), 10(00, 11), 11 (01, 10)) for $S =$ (4 - 7, 8 - 11, 16 - 19, 28 - 31, 32 - 35, 44 - 47, 52 - 55, 56 – 59).

In addition, we present the complex GCD-to-GCS mapping on $N =$ 2048 ($m = 3$). Fig. 7 displays the GCD partitioning (2^M groups, 2^{M-1}

Table 6
Bitwise GCS mapping for $k = 2^{m+2}$ ($N = 2048$, $m = 3$).

Group α_1 $e = 0$	Cross α_2	α_2	$\overline{\alpha}_2$	β	32 global processors
0: 0000 1: 0001	*cross*0 *cross*1	0000 0001	1111 1110	000 - 111 000 - 111	0 - 7, 120 - 127 136 - 143, 240 - 247
$e = 1$					
0: 0000 1: 0001	*cross*1 *cross*0	0001 0000	1110 1111	000 - 111 000 - 111	8 - 15, 112 - 119 128 - 135, 248 - 255
$e = 2$					
0: 0000 1: 0001 …	*cross*2 *cross*3 …	0010 0011 …	1101 1100 …	000 - 111 000 - 111 …	16 - 23, 104 - 111 152 - 159, 224 - 231 …
$e = 7$					
0: 0000 1: 0001	*cross*7 *cross*6	0111 0110	1000 1001	000 - 111 000 - 111	56 - 63, 64 - 71 176 - 183, 200 - 207

crosses per group, 2^{m+1} nodes per cross) on the basis $N = 64$ ($m = 2$), compared to the complex $N = 2048$ ($m = 3$). Table 5 demonstrates the GCD mapping ($N = 2048$ ($m = 3$)) of groups 0 - 15 ($e = 0$ (group 0, cross 0)), where *sc* = 0000 0000 xxx (0 - 7), *dc* = 0000 1111 xxx (120 - 127), and xxx = (000, 001, 010,…, 111).

Fig. 8 displays an obvious example ($N = 2048$ ($m = 3$)) of the GCD partitioning (8 crosses per group), where the (even, odd) crosses can apply the (F, B) protocol (in Algorithm 1). Fig. 9 and Table 6 show the GCS combining of two crosses ($k = 2^{m+2} = 32$) in adjacent groups, first sub-system (*fs*) and second sub-system (*ss*) from $e = 0$ - 7 (in group $\alpha_1 = 0$) and its adjacent group $i = \alpha_1 \oplus \alpha_2 = 1$. For $e = 0$ ($\alpha_1 = 0000$, $\alpha_2 = 0000$, $p = \alpha_1 \oplus \alpha_2 = 0$), all 32 processors are (0 - 7, 120 - 127, 136 - 143, 240 - 247). Fig. 10(a) and Table 7 display the results of $k = 2^{m+3} = 64$ nodes from 4 groups (one cross per group) for $e = 0$ - 7 in group 0 ($p = 0$ - 7). For $e = 0$ ($\alpha_1 = 0000$, $\alpha_2 = 0000$, $p = \alpha_1 \oplus \alpha_2 = 0$), four groups (in the s-group) are 00xx α_{2i} = (0000 α_{2i}, 0001 α_{2i}, 0010 α_{2i}, 0011 α_{2i}). Fig. 10(b) and Table 8 show the results of $k = 2^{m+4} = 128$ nodes from 8 groups (one cross per group) and $k = 2^{m+5} = N/8 = 256$ nodes from 16 groups.

4. All-to-all personalized exchange (ATAPE) and task scheduling on RP-HHCs

After reconfiguring the sub-system for $k \le N/2^{M-1}$ source (*S*) nodes (in Section 3), the proper destinations (*D*) is another key to avoid the routing conflict. In Section 4.1, the ATAPE communication of parallel (*S, D*) routing is used to clarify no routing conflict. In Section 4.2, the optimal (ideal) task-scheduling is addressed. In Sections 4.3 and 4.4, the practical task-scheduling and processor allocation (incorporated with the special binary tree) are presented.

4.1. ATAPE communication on RP-HHC networks

In this study, the ATAPE communication is utilized to clarify the proper SP-routing on the RP-HHCs. The XOR-based ATAPE (*k* rounds of dim-controls *C* = (000..00, 000..01, 000..10, 000..11, …, 111..11)) can avoid the routing conflict on the reconfigured partitions ($2^{m+2} \le k \le N/2^{M-1}$).

First, a reconfigured partition is formed by Eqs. (3.1) – (3.6) for *k* sources (*S*). Next, the corresponding destinations (*D*) in that partition are identified by Eq. (2.2) $D_j = S_{(i \oplus j)}$ in *k* dim-controls $C = i$ ($0 \le i, j \le k$-1). For example, Fig. 11 displays a reconfigured partition ($k = 16$) on the RP-HHC ($N = 64$) with $e = 0$ (in group 0, cross 0), where 16 sources are defined by Definition 4 (S = (0 1 2 3 12 13 14 15 20 21 22 23 24 25 26 27)), see detail in Fig. 4 and Table 2. For this ATAPE ($k = 16$), the destinations (*D*) in each of *k* rounds are recognized in the Latin square of *k* permutations, such as for $C = 0001$, D = (1 0 3 2 13 12 15 14 21 20 23 22 25 24 27 26).

4.2. Optimal ATAPE task scheduling by (even, odd) reconfigured patterns with the (F, B) protocol

By GCD partitioning, 2^M groups can work independently. In a group, 2^{M-1} dependent crosses (2^{m+1} nodes per cross) can work at a time with the proper (*S, D*)-connections. In the past, all (even, odd) crosses with the (F, B) protocol are allowed if (*S, D*) connections are set by the same dim-control [15]. In this study, the optimal GCS combining (Theorem 2) and ideal task scheduling (Theorem 3) can be achieved by the (F, B) protocol on all (even, odd) patterns.

Theorem 2. (Even-odd patterns of GCS combining): For GCD-to-GCS combining, all 2^{M-1} super-groups (s-groups) can work for larger tasks ($k = 2^{m+2}$ nodes (2 crosses) per task). For super-GCS combining, all 2^{M-2} s-groups can work for bigger tasks ($k = 2^{m+3}$ nodes (4 crosses) per task), and so on until there is only one s-group for the biggest tasks ($k = N/2^{M-1}$ nodes (2^M crosses) per task). In any s-group, all (even, odd) patterns ($k = 2^{m+2}$, …, or $N/2^{M-1}$) can apply the (F, B) protocol ($k \le N/2^{M-1}$ nodes per task) for full bidirectional-routing, proven in Section 5.2.

Theorem 3. (Ideal (ATAPE) task scheduling): The optimal (ideal) task scheduling (or packing) can be achieved on all 2^{M-1} (even, odd) reconfigured patterns (in any s-group) under the (F, B) protocol since we can pack the maximum numbers of ATAPE tasks (for the same *k* nodes per task).

For example, Table 9 presents the maximum reconfigured partitions ($k \le N/2^{M-1}$) for the optimal task-scheduling on the RP-HHCs ($N = 64$ ($m = 2$, #crosses (per group) = $2^{M-1} = 2$), $N = 2048$ ($m = 3$, #crosses (per group) = 8), etc.). In each group/s-group, all crosses (2^{m+1} nodes per cross) or all partitions ($\le N/2^{M-1}$ nodes per partition) are classified into (even, odd) patterns to use the (F, B) protocol at the same time. For instance, if there are two incoming tasks T_1 and T_2 ($k = 16$ nodes per task), then two reconfigured partitions ($p = 0$, 1 (see detail in Table 2 and Fig. 22 (b)). In the s-group (groups 0 – 1), all nodes (0 – 31) are assigned to T_1 ($p = 0$, S = (0-3 12-15 20-23 24-27)) and T_2 ($p = 1$, S = 4-7 8-11 16-19 28-31) at the same time, where the (F, B) protocol can work for the (even, odd) patterns (*p*) in parallel (*S, D*)s with the same dim-control (*C*).

4.3. Reconfigured binary-tree structure for RP-HHCs

For the dynamic environment (one task coming at a time), the proposed data structure for the RP-HHC is the reconfigured binary-tree to support the efficient processor allocation (for incoming tasks) and deallocation (for finished tasks). Each node (of this special tree) is represented by a 2^m-bit string (for the *main-net*) of four symbols $\sum$ = {0, 1, x, c}. The notation x..x denotes all possible binary strings (i.e., xx = (00, 01, 10, 11), etc.) and c..c represents the dual *cross*-bit strings (i.e., (cc: $\overline{cc}$) = (00: 11) and (01: 10), etc.).

At the beginning, the initial string xx..x cc..c (in the root) represents the system ($N = 2^n$), containing 2^{2^m} *main-nets*. At level $l = 1$ (after partitioning), the left-child (L) and the right-child (R) of the root are reformed by two strings 0x..xcc..c and 1x..xcc..c, see Fig. 12(a-b) for $N =$

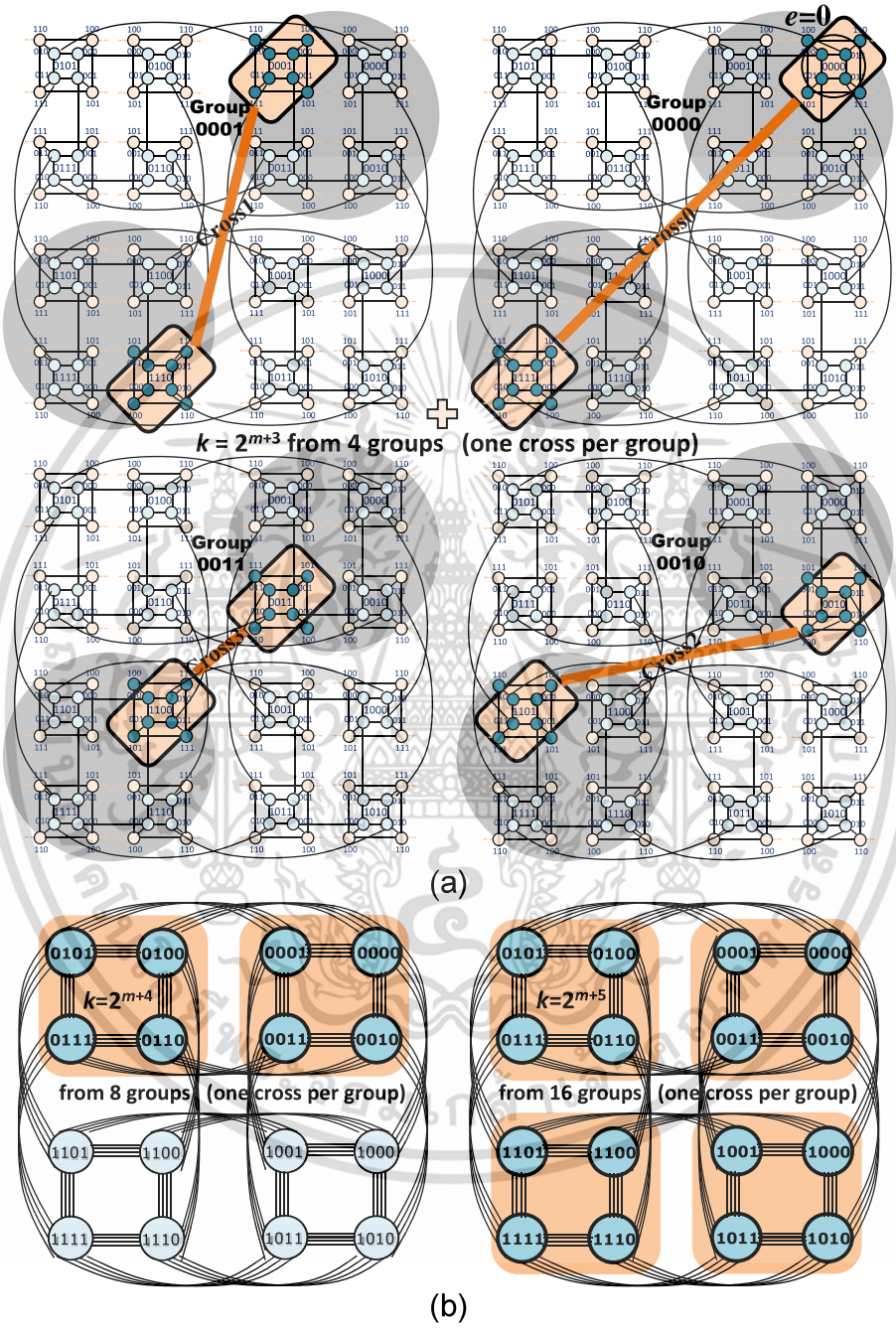


Fig. 10. An example of super-GCS combining ($N = 2048$): a) $k = 2^{m+3} = 64$ (one of eight patterns) and b) $k = 2^{m+4} = 128$, $k = 2^{m+5} = 256$.

Table 7
Super-GCS mapping for $k = 2^{m+3}$ ($N = 2048$, $m = 3$).

Group α_1 $e = 0$	Cross α_2	α_2	$\overline{\alpha}_2$	β	64 global processors
0: 0**000**	*cross*0	0000	1111	000 - 111	0 - 7, 120 - 127
1: 0**001**	*cross*1	0001	1110	000 - 111	136 - 143, 240 - 247
2: 0**010**	*cross*2	0010	1101	000 - 111	272 - 279, 360 - 367
3: 0**011**	*cross*3	0011	1100	000 - 111	408 - 415, 480 - 487
$e = 1$					
0: 0**000**	*cross*1	0001	1110	000 - 111	8 - 15, 112 - 119
1: 0**001**	*cross*0	0000	1111	000 - 111	128 - 135, 248 - 255
2: 0**010**	*cross*3	0011	1100	000 - 111	280 - 287, 352 - 359
3: 0**011**	*cross*2	0010	1101	000 - 111	400 - 407, 488 - 495
$e = 2$					
0: 0**000**	*cross*2	0010	1101	000 - 111	16 - 23, 104 - 111
1: 0**001**	*cross*3	0011	1100	000 - 111	152 - 159, 224 - 231
2: 0**010**	*cross*0	0000	1111	000 - 111	256 - 263, 376 - 383
3: 0**011**	*cross*1	0001	1110	000 - 111	392 - 399, 496 - 503
…	…	…	…	…	…
$e = 7$					
0: 0**000**	*cross*7	0111	1000	000 - 111	56 - 63, 64 - 71
1: 0**001**	*cross*6	0110	1001	000 - 111	176 - 183, 200 - 207
2: 0**010**	*cross*5	0101	1010	000 - 111	296 - 303, 336 - 343
3: 0**011**	*cross*4	0100	1011	000 - 111	416 - 423, 472 - 479

Table 8
Super-GCS mapping for $k = 2^{m+4}$ ($N = 2048$, $m = 3$).

Group α_1 $e = 0$	Cross α_2	α_2	$\overline{\alpha}_2$	β	128 global processors
0: 0**000**	*cross*0	0000	1111	000 - 111	0 - 7, 120 - 127
1: 0**001**	*cross*1	0001	1110	000 - 111	136 - 143, 240 - 247
2: 0**010**	*cross*2	0010	1101	000 - 111	272 - 279, 360 - 367
3: 0**011**	*cross*3	0011	1100	000 - 111	408 - 415, 480 - 487
4: 0**100**	*cross*4	0100	1011	000 - 111	544 - 551, 600 - 607
5: 0**101**	*cross*5	0101	1010	000 - 111	680 - 687, 720 - 727
6: 0**110**	*cross*6	0110	1001	000 - 111	816 - 823, 840 - 847
7: 0**111**	*cross*7	0111	1000	000 - 111	952 - 959, 960 - 967
…	…	…	…	…	…
$e = 7$					
0: 0**000**	*cross*7	0111	1000	000 - 111	56 - 63, 64 - 71
…	…	…	…	…	…
7: 0**111**	*cross*0	0000	1111	000 - 111	896 - 903, 1016-1023

64 ($m = 2$). This binary partitioning is repeated until reaching the *group*-level ($l = M = 2^{m-1}$), where 2^M groups can work independently. In each group, the GDC partitioning creates 2^{M-1} dependent crosses (2^{m+1} nodes per cross).

At the *cross*-level ($l = 2^m$-1), each *cross*-node ($k = 2^{m+1}$ processors) is composed of two dual cubes (c and $\overline{c}$), according to the GCD partitioning, see Fig. 12(c-d) for $N = 64$ ($m = 2$) and Fig. 14 for $N = 2048$ ($m = 3$). For each *cube*-node (2^m processors), the additional m-bit string (xx..x) is appended to refer to all internal processors in the *sub-net*.

Lastly, for small tasks on the m-hypercube (or HC with $k \le 2^m$) at the *cube*-level ($l = 2^m$, 2^m+1, …, n), the existing HC partitioning can be applied directly (i.e., L = 0, R = 1 of the corresponding *sub-net* (m bits)). Therefore, the smaller tasks ($k \le 2^m$) can be executed independently on the RP-HHC at anywhere (in the *sub-nets*) in any time.

During runtime, for the task-allocation (on the PR-HHC), the full binary sub-tree (from the root to the *cross*-level) is created, where the *cross*-nodes can be allocated directly for $2^{m+1} \le k \le N/2^{M-1}$ processors per task. For example, see Fig. 12(c-d) $N = 64$ ($m = 2$), there were five incoming tasks (T_1 ($k_1 = 4$), T_2 ($k_2 = 8$), T_3 ($k_3 = 8$), T_4 ($k_4 = 16$), T_5 ($k_5 = 16$)) and the allocations were task T_1 on a *cube*-node (0000), tasks (T_2, T_3) on *cross*-nodes C_0 (0100, 0111) with $e = 0100$ ($p = 1$ (B-protocol)) and C_1 (0101, 0110) with $e = 0101$ ($p = 0$ (F-protocol)) of group 1, and tasks (T_4, T_5) on two combined sub-systems of groups 2 and 3 (i.e., T_4 (with $e = 1000$, $p = 1$ (B)) and T_5 (with $e = 1001$, $p = 0$ (F))).

Moreover, in order to retain time complexity and to handle the high system utilization for the dynamic environment, the accumulated *sum* of utilized processors (inside each of tree nodes) is incorporated. For example, Fig. 13 shows the reconfigured binary-tree of the current system status. Assume at the observed period, there are four tasks (T_1 – T_4), allocated on 28 working processors, according to the allocation/deallocation (in the former period). In Fig. 13(b), the node 10cc at level 2 of the tree has the accumulated *sum* = 8 because in the left child (*sum* = 0), no processor allocated, and in the right child (*sum* = 8) since eight processors are working.

In addition, for the efficient deallocation, the array-lists of the limited (maximum) processors (per node at level l of the tree) plus the linked lists of the allocated tasks are incorporated in each level ($l = 0, 1, \ldots, n$). For $N = 64$ ($n = 6$), the limited processors (at level l) $= 2^{n-l}$ (= 64, 32, …, 2, 1), where $0 \le l \le n$ and $n = \log_2 N$. Each task (in the allocated list) contains e (main-net) and p (pattern) to refer to the corresponding nodes in the reconfigured tree.

In particular, Fig. 13 includes the array-lists of four allocated tasks (T_1 – T_4). At index $l = 2$ of the array (or level $l = 2$ of the tree), the limited processors (LP) = 16 plus a list of the allocated task T_2 ($e = 1001$, $p = 0$ to refer to two *cross*-nodes C_1 = (1001, 1010) of group 2 and C_0 = (1100, 1111) of group 3). At index $l = 4$ (LP = 4), tasks T_1, T_3, and T_4 are allocated on nodes 0010, 1101, and 1110.

In the practical system, the reconfigured binary-tree plus the array-lists are necessary for the efficient allocation / deallocation on the RP-HHCs, especially for the scheduler of the operating system (OS). Fig. 14 depicts a complex example of the reconfigured tree for $N = 2048$ ($m = 3$), $2^M = 16$ groups, and $2^{M-1} = 8$ crosses per group.

Note that in the dynamic environment, if only some crosses in any group/s-group have been allocated for the same $k = 2^{m+j}$ processors per task ($j \ge 1$), the remaining processors of these groups can be used with limitation (allowing $k \le 2^m$ nodes per task only). Table 10 addresses the proper blocking-condition, which will be concerned to avoid the (possible) runtime conflict. This blocking can be handled directly on the corresponding nodes of the reconfigured tree. For example, Fig. 13(b) shows a blocking on the RP-HHC ($N = 64$, $m = 2$) after the task T_2 ($2^{m+2} = 16$) was allocated, the *group*-nodes 10cc and 11cc must be blocked, which will not allow for $k > 2^m$ nodes per task.

4.4. Efficient processor allocation/deallocation

Under dynamic environment, two popular strategies for new incoming tasks are presented, which are the (simple) first-fit allocation (Algorithm 3) and the (efficient) best-fit allocation (Algorithm 4) plus the deallocation (Algorithm 5). The efficient allocation/deallocation can be performed in O(N) by employing the reconfigured binary-tree (in Section 4.3), incorporated with the accumulated *sum* plus the array-lists. For an incoming task, if there is the available sub-system, then that task is allocated and the corresponding nodes (in the tree) are updated. Otherwise, that task is added in the waiting queue. Assume the input k denotes a number of input processors for the new incoming task, where $k \le N/2^{M-1}$ processors per task. Each of tree nodes (at level l) contains an accumulated *sum* ($\le 2^{n-l}$), while the array-lists at an index l specify the limited size ($= 2^{n-l}$ max processors) and the linked list of the working tasks to refer to the utilized nodes in the tree.

For example, Fig. 15(a) shows the first-fit allocation for a new task

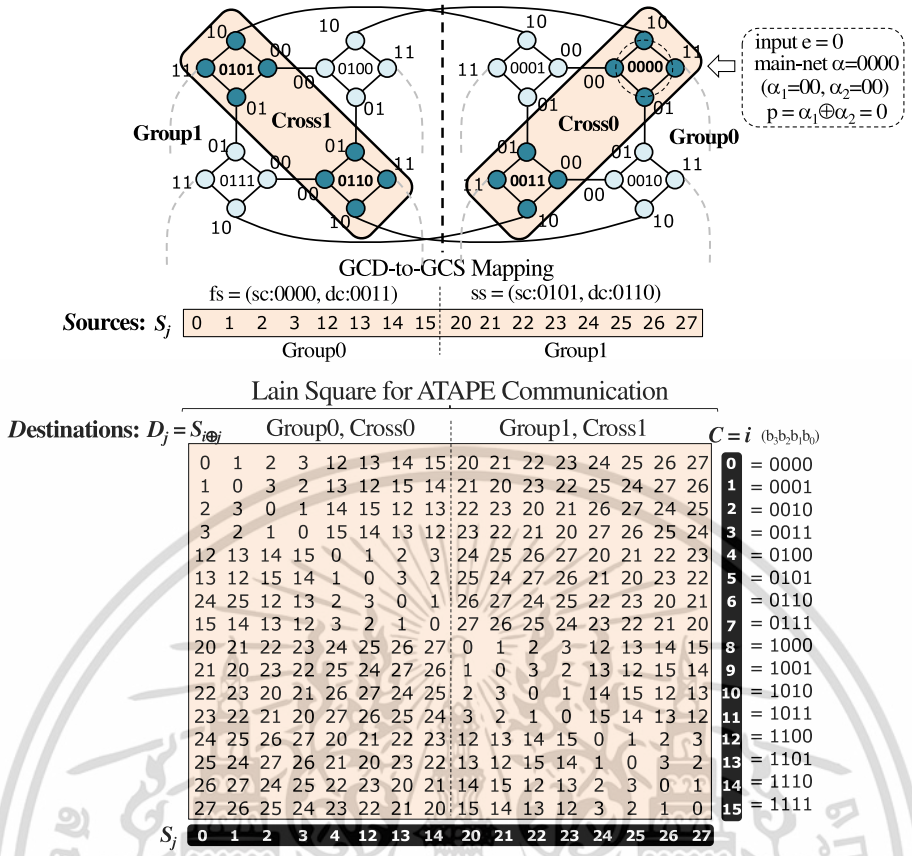


Fig. 11. An example of ATAPE communication on a reconfigured partition ($k = 16$, $N = 64$, $m = 2$) by GCD-to-GCS combining ($e = 0$).

(T_5) that requests 4 processors ($k = 2^m = 4$) on the RP-HHC ($N = 64$). From the current allocation (Fig. 13), assume there are four tasks allocated (T_1 ($k_1 = 4$) on (0010), T_2 ($k_2 = 16$) on C_1 (1001, 1010) of group 2 and C_0 (1100, 1111) of group 3, T_3 ($k_3 = 4$) on 1101, and T_4 ($k_4 = 4$) on 1110). Then, the allocation for T_5 ($k = 4$) starts searching from the root node (xxcc), where the current *sum* (= 28) satisfies the condition *sum* + *k* (= 28 + 4 = 32) < *limit* (= 64). Next, the DFS process goes through node 0xcc (*sum* + *k* = 4 + 4 = 8 < *limit* = 32) → node 00cc (*sum* + *k* = 8) < *limit* = 16) → *cross* node C_0: 0000, 0011 (*sum*+*k* = 4 < *limit* = 8) for the first available *cube* node 0000 ($k = 4$) that can allocate for the requested task T_5 at the level $l = 2^m = 4$, a new circle node, for 0000,xx processors (or 0000,00 (0), 0000,01 (1), 0000,10 (2), and 0000,11 (3)) on the RP-HHC. Then, the allocated task T_5 is appended in the (array) list at the index $l = 4$ with the *main-net* $e = 0000$ to refer to the utilized node in the tree. Finally, the accumulated *sums* of the corresponding nodes are updated from the allocated node 0000 up to the root (where *sum* before/after T_5 = 28/32).

Usually the first-fit allocation is simple and fast but yields the low system-utilization (or high fragmentation), while the best-fit allocation yields the high system-utilization (or low fragmentation) but is time consuming. Therefore, we propose the efficient best-fit allocation (in Algorithm 4), incorporated with the accumulated *sum* that can be processed in O(*N*) time as the first-fit allocation, while maintaining the high performance.

Algorithm 4 presents the efficient best-fit allocation in O(*N*), analyzed in Section 5.3, according to the accumulated *sum* for the DBS policy (the min(LF, RF) best-fitting). Fig. 15(b) displays an example of the best-fit allocation for a new task (T_5) that requests $k = 2^m = 4$ nodes. The allocation starts from the root node (xxcc) that satisfies the condition *sum* + *k* (= 28 + 4 = 32) < *limit* (= 64). Next, the best-fit factor of L/R-child nodes are computed before applying DBS through node 1xcc (*sum*+*k* = 24 + 4 = 28 < *limit* = 32) → node 10cc (*sum* + *k* = 8 + 4 = 12 < *limit* = 16) → *cross* node C_0: 1000, 1011 (*sum* + *k* = 4 < *limit* = 8) for the best available *cube* node 1000 ($k = 4$) that can allocate for the requested task T_5 at the level $l = 4$, a new circle node, for 1000,xx processors (32 - 35) (or 1000,00 (32), 1000,01 (33), 1000,10 (34), 1000,11 (35)) on the RP-HHC. Then, the allocated task T_5 is appended in the (array) list at the index $l = 4$ with *main-net* $e = 1000$ to refer to the utilized node in the tree and update the accumulated *sums* of the corresponding nodes up to the root. Fig. 15 also displays the effect of the system fragmentation from the first-fit and best-fit allocations, after assuming T_1 (on 0010) was finished. In the best fit, the whole group 1 is free (Fig. 15(b-1)), but in the first fit the whole group 1 is not free (Fig. 15(a-2)).

Fig. 16 shows another example (the GCS combining) for the new task T_6 ($k = 16$). With the best-fit policy, from the root (xxcc), apply DBS to the best-bit node 0xcc at level $l = 1$, containing two adjacent groups (00cc, 01cc) at level $l = 2$ for the best available GCS pattern ($e = 0$) of two *cross* nodes (C_0: 0000, 0011 (in group 0) and C_1: 0101, 0110 (in group 1) at level $l = 3$). Then, the allocated task T_6 is appended in the (array) list at the index $l = 2$ ($e = 0$) and update the accumulated *sums* of the corresponding nodes to the root. In this case, the blocking status must be set at the *group*-level on nodes 00cc and 01cc. Note that this task T_6 ($k = 16$) cannot be allocated by the first-fit policy.

Table 9
Ideal scheduling on (even, odd) patterns + (F, B) protocol.

#crosses(per group)= 2^{M-1}	k nodes per task: This study	Previous work[15]	#groups in s-group	#s-groups (in HHC)	#Tasks = #crosses × #s-groups
$N = 64$ (m=2)	$2^{m+1} = 8$	$2^{m+1} = 8$	1	$2^M = 4$	$2 \times 4 = 8$
#crosses = 2	$2^{m+2} = 16$	-	2	$2^{M-1} = 2$	$2 \times 2 = 4$
	$2^{m+3} = 32$	-	4	$2^{M-2} = 1$	$2 \times 1 = 2$
N = 2,048 (m=3)	$2^{m+1} = 16$	$2^{m+1} = 16$	1	$2^M = 16$	8×16=128
#crosses = 8	$2^{m+2} = 32$	-	2	$2^{M-1} = 8$	$8 \times 8 = 64$
	$2^{m+3} = 64$	-	4	$2^{M-2} = 4$	$8 \times 4 = 32$
	$2^{m+4} = 128$	-	8	$2^{M-3} = 2$	$8 \times 2 = 16$
	$2^{m+5} = 256$	-	16	$2^{M-4} = 1$	$8 \times 1 = 8$
N = 1,048,576 (m=4)	$2^{m+1} = 32$	$2^{m+1} = 32$	1	$2^M = 256$	128×256
#crosses = 128	$2^{m+2} = 64$	-	2	$2^{M-1} = 128$	128×128
	$2^{m+3} = 128$	-	4	$2^{M-2} = 64$	128×64
	$2^{m+4} = 256$	-	8	$2^{M-3} = 32$	128×32
	…	-	…	…	…
	$2^{m+9} = 8192$	-	256	$2^{M-8} = 1$	128×1

Note: For other static environments, assume there are a number of tasks ($k \le N/2^{M-1}$ nodes per task) waiting for scheduling. First, all tasks are sorted in decreasing order of sizes (k) to execute the set of the largest tasks first. All tasks are packed (according to the order) in each s-group of the same requested size (k) as much as possible.

When any task is finished, Algorithm 5 is called to deallocate that task and combine the corresponding free nodes. For example, Fig. 17 illustrates the tree after free the finished task T_1 ($k = 4$). First, T_1 is deleted from the (array) list (at index $l = n - \log_2 k = 4$). After the tree-node (0010) is free, the corresponding free-nodes are combined to the *cross*-level and the accumulated *sums* of the associated nodes are updated. Then, if the waiting queue is not empty, either Algorithm 3 (the first-fit allocation) or Algorithm 4 (the best-fit allocation) will be called to allocate the waiting tasks with FCFS (first-come, first-serve).

5. Correctness analysis and experiments

In this section, we aim to validate the dynamic RP-HHC reconfiguring for $k \le N/2^{M-1}$ nodes per task (our innovation in Section 3) and confirm with the ATAPE application (our contribution in Section 4). First, Section 5.1 reviews the correctness idea of the bitwise GCD-to-GCS reconfiguring. Section 5.2 presents the correctness of all (even, odd) patterns with the (F, B) protocol. Section 5.3 addresses the efficient time-complexity of the best-fit allocation/deallocation. Lastly, the experiments are conducted (in Section 5.4) to confirm no conflict when executing the ATAPE communication ($k \le N/2^{M-1}$) on the RP-HHCs.

5.1. Correctness of GCD-to-GCS reconfiguring

The innovation of the GCD-to-GCS reconfiguring on the RP-HHCs for $k = 2^{m+2}$ up to $N/2^{M-1}$ nodes per task (in Section 3) was achieved on top of the GCD partitioning, our previous work [15]. That partitioning formulates the *cross* structure (for $k = 2^{m+1}$ nodes) to avoid the routing conflict by working with the parallel SP-routing (Algorithm 1).

The correctness of the cross (2^{m+1}) and the combined crosses ($2^{m+2} \le k \le N/2^{M-1}$) in Eqs. (3.1) – (3.6) can be analyzed naturally on the binary-tree partitioning (by the most significant bit first), like the HC partitioning, from the root to the group level. For the RP-HHC, three node types are involved: 1. *cross* nodes ($k = 2^{m+1}$), 2. *cube* nodes ($k \le 2^m$), and 3. *processor* nodes ($k = 1$). The partitioning (from the root (xx…xcc…c) to the *group*-level ($l = 2^{m-1}$)) is the regular binary partitioning (L (left-child) = 0, R (right-child) = 1). In our dynamic reconfiguring, all 2^M groups can be assigned to tasks ($k = 2^{m+1}$ nodes per cross) independently (one cross per group). In each group, the routing conflict can be solved locally by the GCD partitioning with the cross structure ($k = 2^{m+1}$) from level $l = M+1$ to the *cross*-level ($l = 2^m$-1). We can achieve the GCD-to-GCS combining for $2^{m+2} \le k \le N/2^{M-1}$ (from levels $l = 2^m$-1 up to $l = 0$), like the HC combining (CASE I), where the correctness of the HHC combining was proven in CASEs II-III.

CASE I: Cube and combined cube on HCs ($N = 2^n$).

Under HC partitioning, Fig. 18(a) displays the binary tree of the HC ($N = 8$, $n = 3$), partitioned from the root (xxx) at level 0 to (0xx, 1xx) at level 1, (00x, 01x, 10x, 11x) at level 2, and (000, 001, 010, …, 111) at level 3. Fig. 18(b) shows an example of the combined nodes (0xx, 1xx) in the binary tree. In general ($N = 2^n$), Fig. 19 illustrates the n-bit partitioning on the HC networks (Fig. 19(a)) from MSb (the most significant bit b_{n-1}) and the 2^m-bit *main-net* partitioning on the HHC networks (Fig. 19(b)).

CASE II: GCD partitioning on RP-HHCs ($N=2^n$, $n=2^m+m$).

The HHC partitioning from the root (levels $l = 0$ - M) is 0/1 partitioning (like the HC) on the high-order 2^{m-1}-bit group (of the 2^m-bit *main-net*), see Fig. 20. For example ($N = 64$), the root (xxcc) is partitioned to (0xcc, 1xcc) at level 1 and (00cc, 01cc, 10cc, 11cc) at level 2, where 2^M groups can work independently. In each group, the GCD partitioning (on the low-order 2^{m-1}-bit cross) for 2^{M-1} dependent crosses (two dual-cubes per cross) is performed at the *cross*-level. Fig. 20(a) shows a cross C_0 ($k = 2^{m+1}$ on (0000, 0011)), defined by Eq. (3.1), where all k processors (in each cross) can communicate without the conflict [15]. Then, to avoid the routing conflict with other crosses (in the same group), the (F, B) protocol (Algorithm 1) is used for the (even (F), odd (B)) crosses, see Fig. 8 ($N = 2048$), since the opposite crosses ((0:000, 5:101), (1:001, 4:100), (2:010, 7:111), (3:011, 6:110)) cannot use the same F/B protocol.

CASE III: GCS combining on RP-HHCs ($N=2^n$, $n=2^m+m$).

From level $l = 2^m$-1 (the *cross*-level), the GCS combining ($k \ge 2^{m+2}$ by Eqs. (3.2)–(3.6)) can be reconfigured from the proper *cross*-nodes (of adjacent groups) up to the root (for $k = N/2^{M-1}$ processors), according to the binary (group) partitioning (like the HC partitioning). For example, Fig. 20(b) shows a set of two crosses ($k = 2^{m+2}$) from two adjacent groups (00cc, 01cc) by Eq. (3.2), C_0 (0000, 0011) in group 0 and C_1 (0101, 0110) in group 1. Fig. 20(c) displays a set of four crosses (in a super-group) from four groups (00cc, 01cc, 10cc, 11cc) by Eq. (3.6), C_0 (group 0), C_1 (group 1), C_1 (group 2), C_0 (group 3). All k processors can communicate with no conflict, under the cross-based SP-routing of the GCD partitioning.

5.2. Correctness of (even, odd) patterns with (F, B) protocol for full (group/super-group) utilization

The objective of this study is to solve the HHC conflict completely and achieve the full (group) utilization plus the maximum task-packing (or optimal task-scheduling). Our reconfiguring relies on the cross structure of GCD partitioning ($k = 2^{m+1}$ nodes per cross) and the GCS combining ($2^{m+2} \le k \le N/2^{M-1}$ nodes), which can be performed dynamically during runtime. Each s-group (super-group) combines the

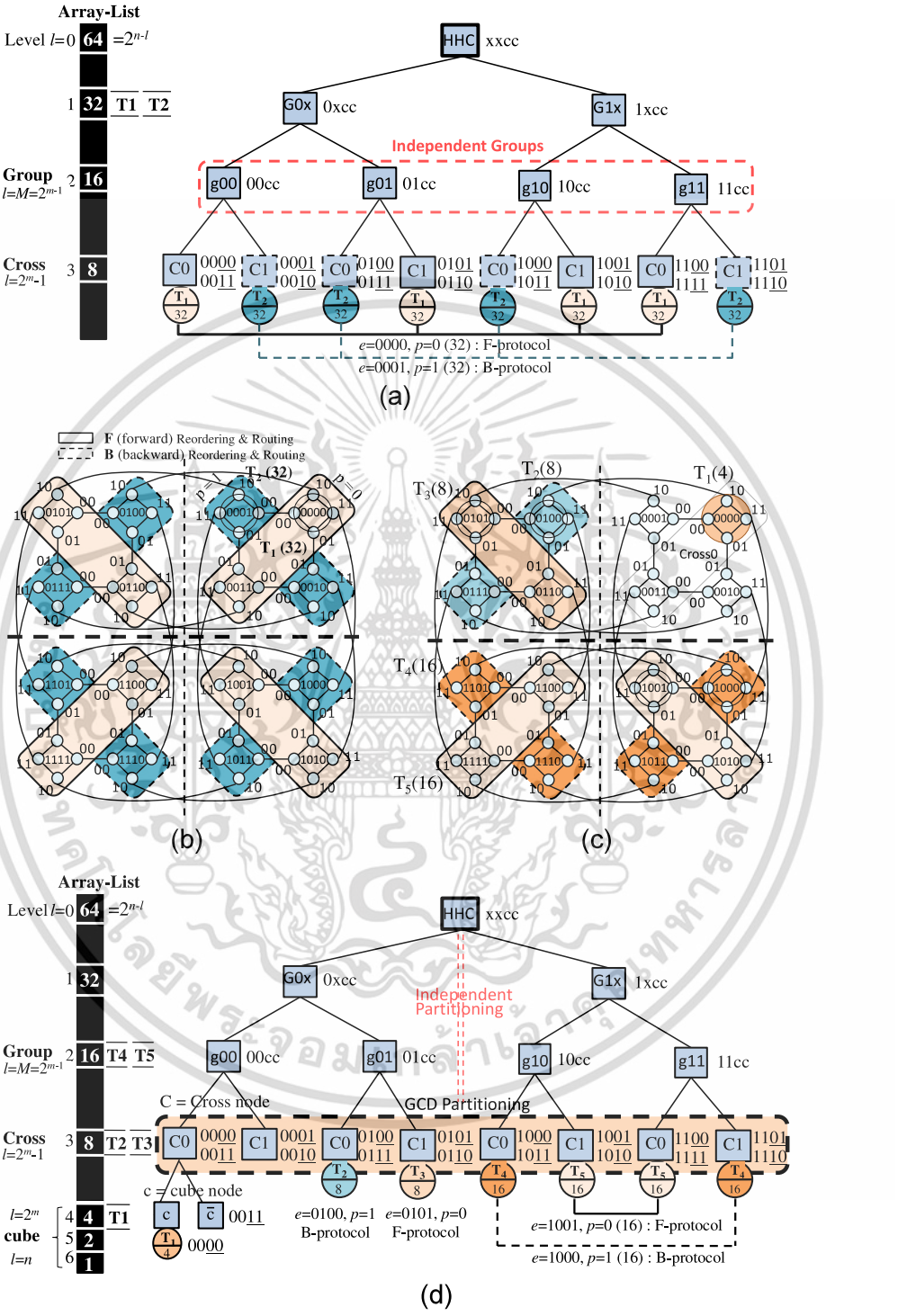


Fig. 12. An example ($N = 64$, $m = 2$): a-b) optimal task-scheduling (T_1 ($k_1 = 32$), T_2 ($k_2 = 32$)) and c-d) 5 task-scheduling (T_1 – T_5).

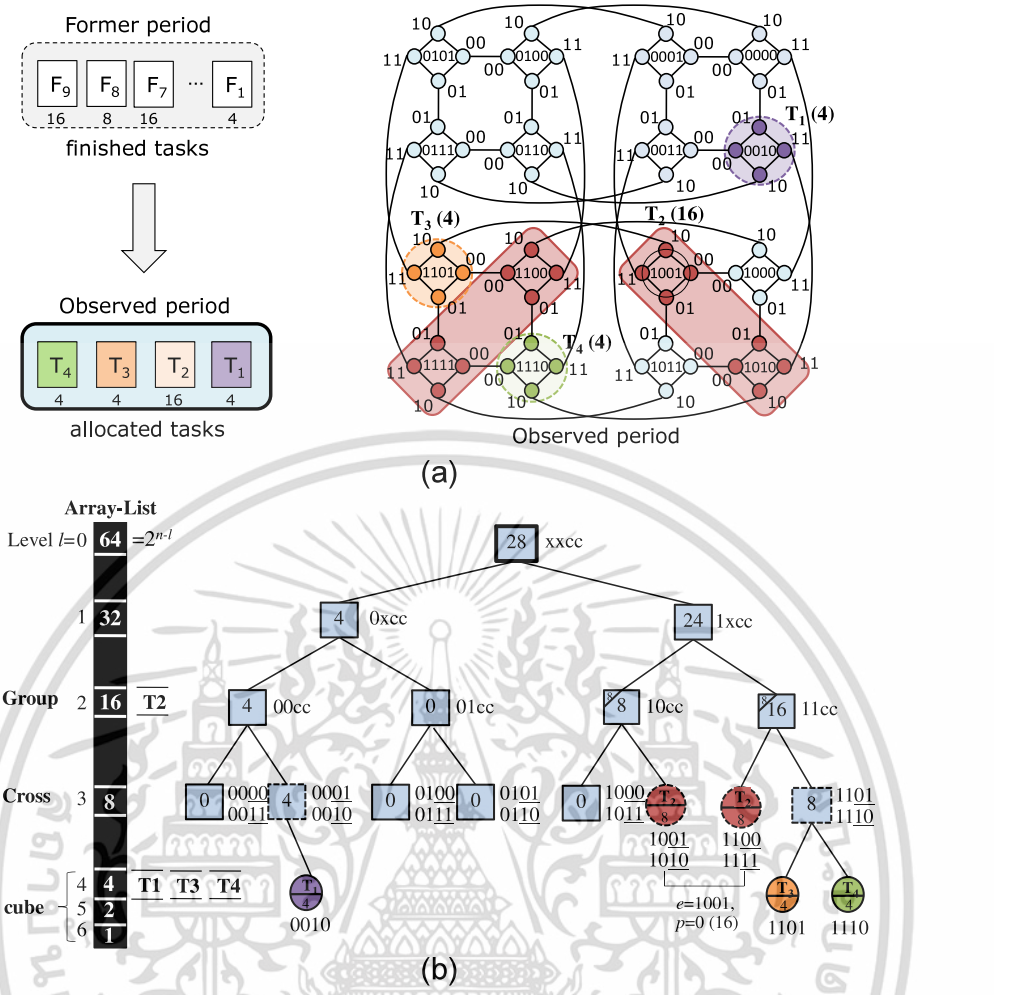


Fig. 13. An example of an intermediate result of task scheduling ($N = 64$, $m = 2$): a) four allocated tasks and b) the reconfigured tree.

right crosses of adjacent groups with the reconfigured pattern p (Definition 5). All s-groups can work independently for one task ($k \le N/2^{M-1}$) per s-group. Note: In each group, the proper parallel (S, D) functions must be used to avoid the conflict of some (shared) edges with other dependent crosses, where the right (S, D) connections can be realized by the XOR operation $D_j = S_{i \oplus j}$.

Next, for the optimal task-scheduling (in Section 4.2), all 2^{M-1} reconfigured partitions (in each s-group) are allowed for multi-tasks ($2^{m+1} < k \le N/2^{M-1}$ nodes per task). In particular, all (even, odd) patterns can execute at the same time under the (F, B) protocol (Algorithm 1 (F: forward, B: backward)) for full bidirectional routing if all parallel (S, D) connections are performed with the same dim-control (i.e., XOR operations). See examples ($N = 64$) in Fig. 21 (ATAPE: $k = 8$) and Fig. 22 ($k = 8, 16, 32$).

See also Section 5.4 (the experiments) to evaluate the conflict-free routing (in practice) on all reconfigured partitions ($2^{m+1} \le k \le N/2^{M-1}$) by using the frequency used ATAPE communication with k dim-controls on the RP-HHCs (i.e., $N = 64$ ($m = 2$), $N = 2048$ ($m = 3$), etc.).

5.3. Time complexity of the best-fit allocation

Under GCD partitioning, a full binary sub-tree was created from the root to 2^{M-1} cross nodes (at level $l = 2^m - 1$) in $O(2^M)$. Table 11 presents the time complexity ($O(N)$) of the best-fit allocation/deallocation, see the following details:

In the sub-system allocation (with the best-fit policy),

1 DBS (depth best search) for an input k in the best case is $O(2^m)$ for a *cross*-node ($k = 2^{m+1}$) and the worst case is $O(2^m + m) = O(n)$ for a *cube*-node ($k < 2^m$) by using the best-fit min(LF, RF) to reach the best-fit node directly.
2 Allocate nodes in tree is $O(2^m)$ in the best case (for the *cross*-node) and $O(2^M)$ in the worst case for the super-GCS combining for $k = (N/2^{M-1})/2^m \approx 2^M$; $M = 2^{m-1}$.
3 Update the accumulated *sum* is $O(2^m)$ in the best case and $O(2^M)$ in the worst case since updating from the *cross*-nodes to the root $\approx 2^M + 2^{M-1} + \ldots + 2^1 + 2^0 = 2^{M+1} - 1$.

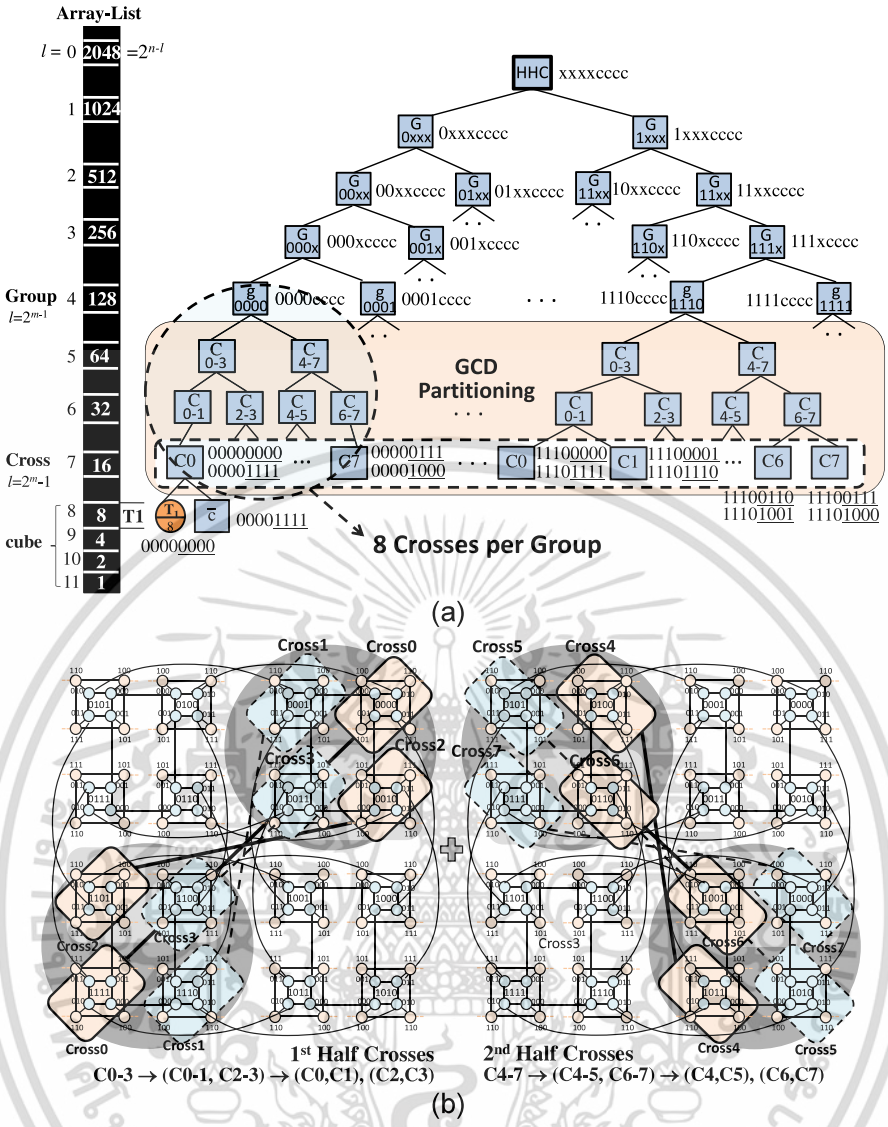


Fig. 14. a) A complex reconfigured-tree ($N = 2048$, $m = 3$) and b) the GCD partitioning and mapping in each group of the RP-HHC.

Table 10
Blocking condition for $k = 2^{m+j}$ ($j \geq 1$) on RP-HHCs.

For ideal task-scheduling, 2^{M-1} tasks ($k = 2^{m+j}$) can be allocated at the same time by the (F, B) protocol (even patterns (F) & odd patterns (B)). Note: see F/B (forward/backward) *dSPreordering* (in Algorithm 1).
Otherwise, if not all crosses are allocated for $k = 2^{m+j}$, the corresponding groups must be blocked (not allowing $k > 2^m$ per task) to avoid the conflict. Under blocking, only task sizes $k \leq 2^m$ (on *cube*-nodes) are allowed.

In the sub-system deallocation,

1 Search in the array-list (at index $i = n\text{-}\log_2 k$) for the T_{finish} is O(1) in the best case (if T is the first node of list i) and O(N) in the worst case (if T is the last node of list n).
2 Free *cross*-nodes (from T_{finish} (e, p)) is O(1) in the best case and O(2^M) in the worst case ($\leq (k = N/2^{M-1})/2^m \approx 2^M$ combined *cross*-nodes).
3 Update the accumulated *sum* is O(2^m) in the best case and O(2^M) in the worst case since updating from the *cross*-nodes to the root $\approx 2^M + 2^{M-1} + \ldots + 2^1 + 2^0 = 2^{M+1} - 1$.

5.4. *Experiments to confirm no conflict on RP-HHCs when executing ATAPE communication ($k \leq N/2^{M-1}$)*

The experiments were conducted to evaluate not only the reconfiguring ($k \leq N/2^{M-1}$ nodes) but also the ATAPE SP-routing by implementing the following pseudo-code.

ATAPE Communication (*e*: external node, *k* PEs, *C*: *k* dim-controls)
  for input *e*, **configure a sub-system (*k* PEs)** by Eqs. (3.1) - (3.6);
  for all **sources *S*** (in each of reconfigured sub-systems) pardo
    compute $\boldsymbol{D_j = S_{i \oplus j}}$ for ATAPE comm. by Eq. (2.2); // see Fig. 21
    if(cross%2=0) direction='F' else direction='B'; // Forward/Backward
    call Algorithm 1: parallelShortestPaths (***P*: *S*→*D***, direction: F/B);
    do HHC-Routing (*S, D*); update RoutingTable (RT); //see Table 12
  end for all; // see k = 8, 16, 32 (on N = 64, m = 2) in Fig. 22
  report "no-conflict" if in each t (time) all edges (in RT) are different;
end ATAPE.
HHC-Routing (*P*: Shortest Path from *S* to *D*) // see Fig. 2(b)
  for element i in path P (i = 0, 1, 2, …, |P|-1)
    p=P[i]; mNet=mainNet[p]; sNet=subNet[p];
    mainNet-subNet-routing (mNet, sNet) and local HC-routing;
  end for i
end SP-Routing // see Fig. 21, the (F, B)-protocol on (even, odd) patterns

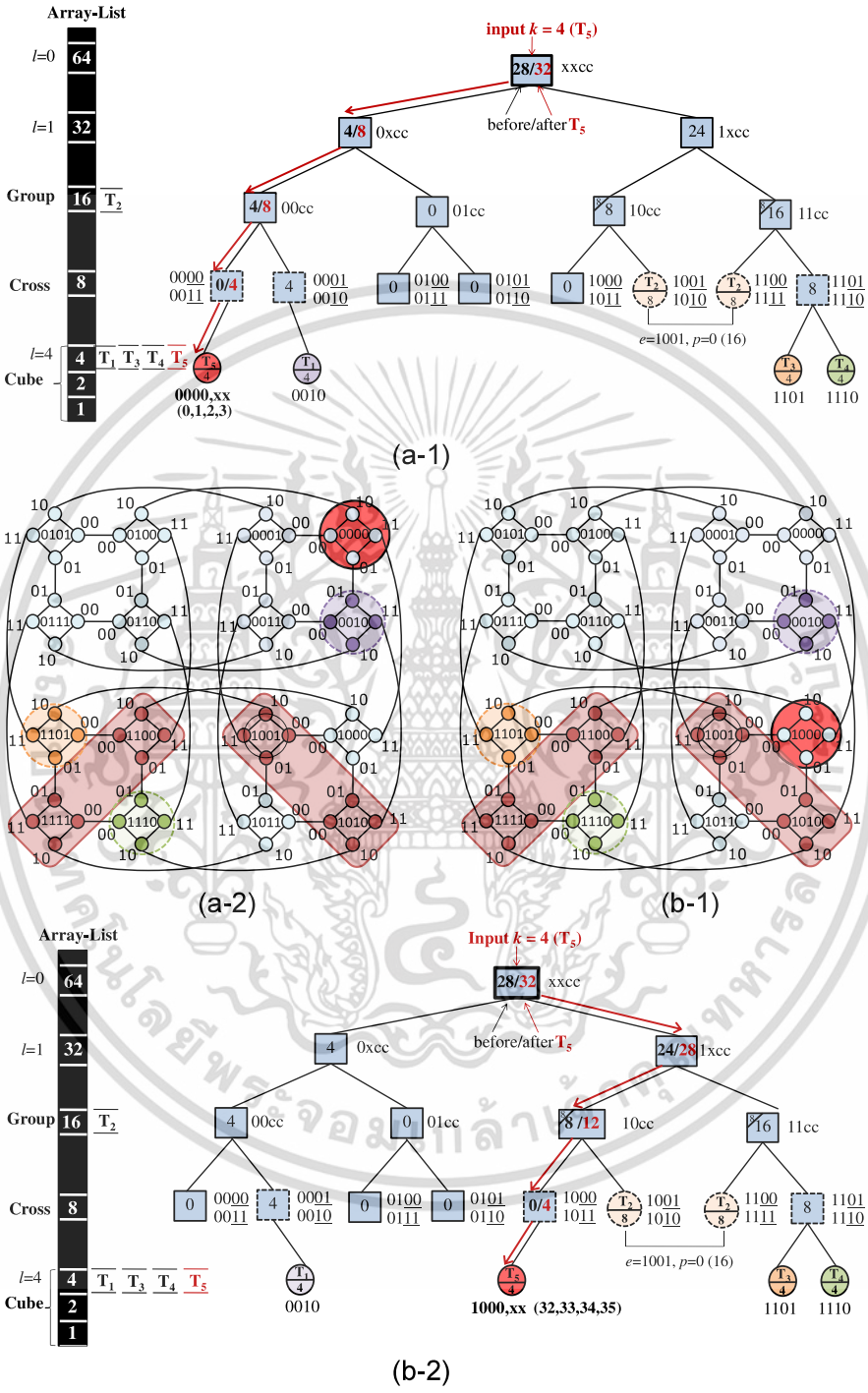


Fig. 15. An example of allocation with input $k = 2^m = 4$ by two policies: a) simple first-fit allocation and b) efficient best-fit allocation.

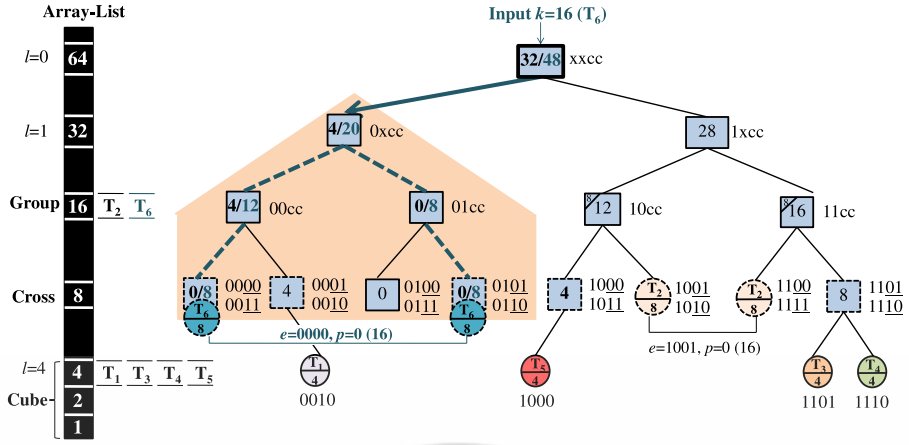


Fig. 16. An example of best-fit allocation with input $k = 2^{m+2} = 16$ (T_6).

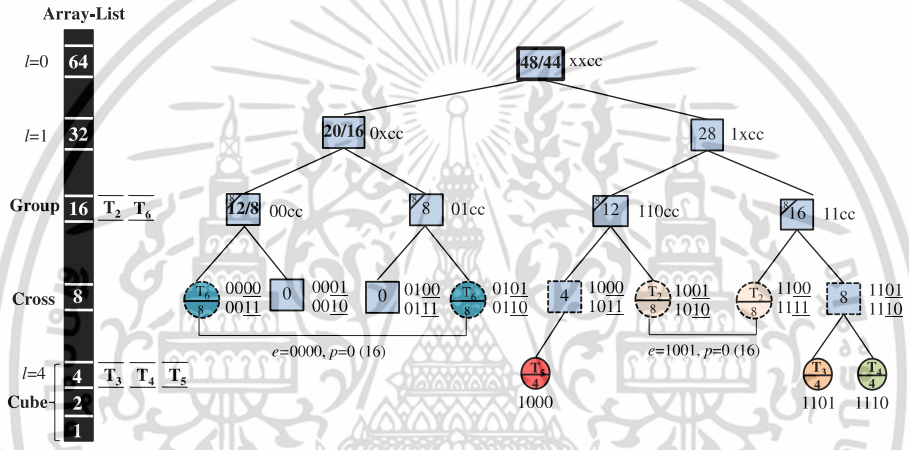


Fig. 17. An example of deallocation of task T_1 ($k = 4$) from Fig. 16.

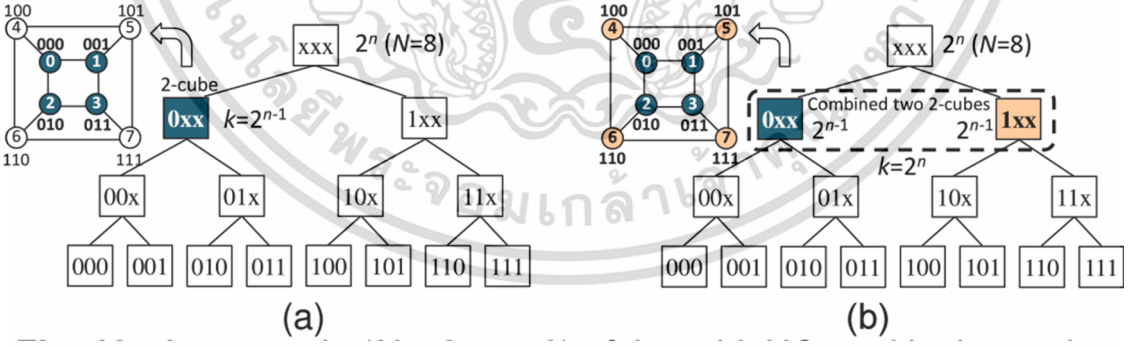


Fig. 18. An example ($N = 8$, $n = 3$) of the n-bit HC partitioning and combining: a) one node (0xx); b) two combined nodes (0xx, 1xx).

For each test (with a pattern $p = 0, 1, .., 2^{M-1}-1$ (Definition 5)), a partition (k) was reconfigured from $e = \alpha_1\alpha_2$ (group α_1, cross α_2), related to the pattern p. First, k sources (S) were formulated by the proper Eqs. (3.1) – (3.6), implemented directly in functions. Next, k destinations ($D_j = S_{i\oplus j}$) were set by Eq. (2.2) for the ATAPE communication (in k rounds). Then, all (S, D)-routing paths were defined by Algorithm 1. For the conflict-free evaluation, a routing table (k x T) or a 2D-array (k x N) is used to record all edges = (u, v)s of each path P in each time t (= 1, 2, …, T (< N)) for k parallel (S, D)-routing paths. Lastly, there is no conflict if in each t (= 1, 2, …, T) all edges are different.

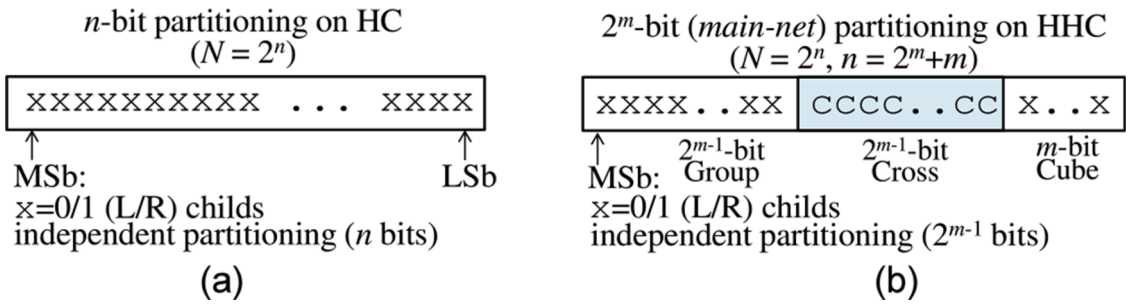


Fig. 19. The n-bit format: a) HC-partitioning and b) HHC-partitioning.

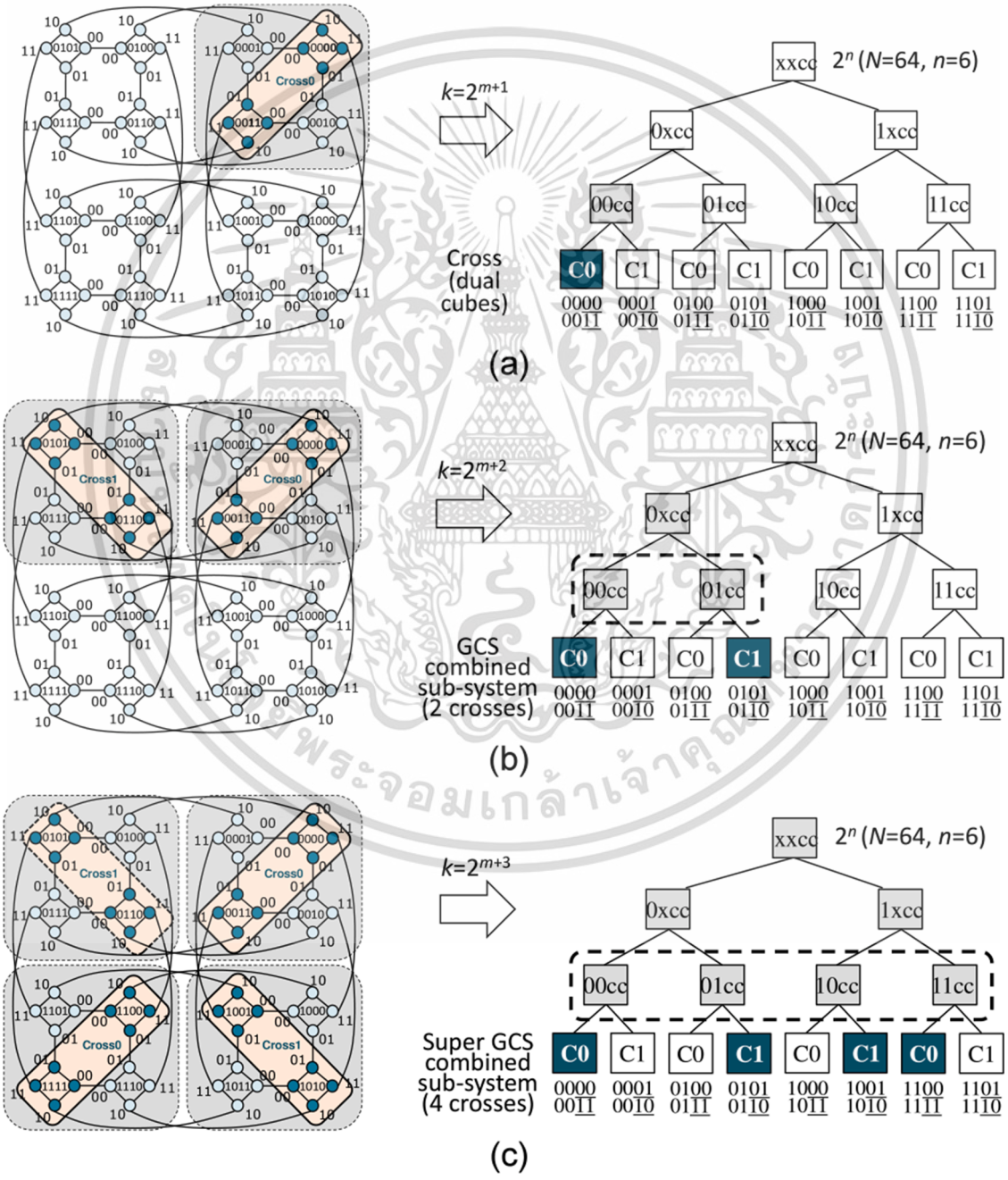


Fig. 20. An example of $k \geq 2^{m+1}$ nodes per task ($N = 64$, $m = 2$): a) GCD ($k = 2^{m+1}$); b) GCS ($k = 2^{m+2}$); and c) super-GCS ($k = 2^{m+3}$).

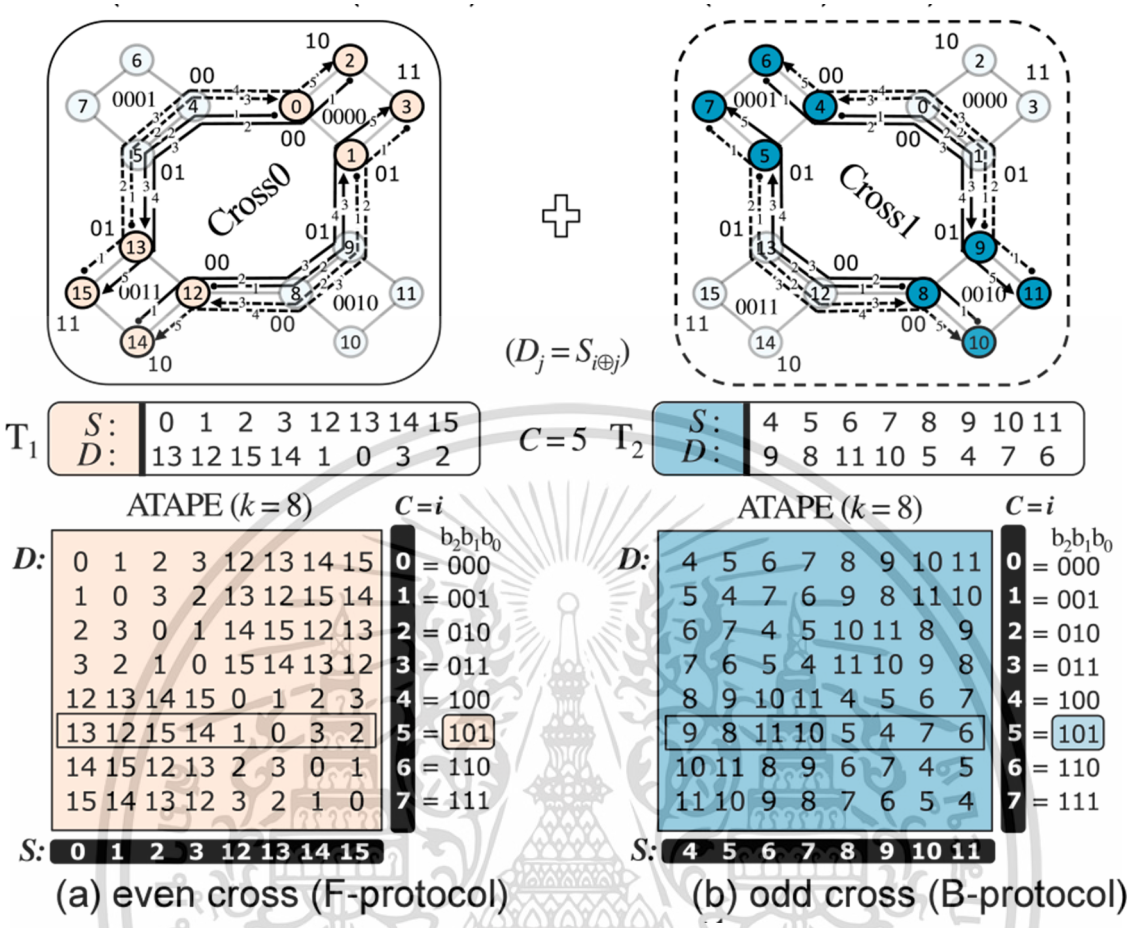


Fig. 21. An example ($N = 64$, $m = 2$, $k = 2^{m+1} = 8$) of full bidirectional-routing of T_1 & T_2 (ATAPE ($D_j = S_{i\oplus j}$), $C = 5$) with (F, B) protocol.

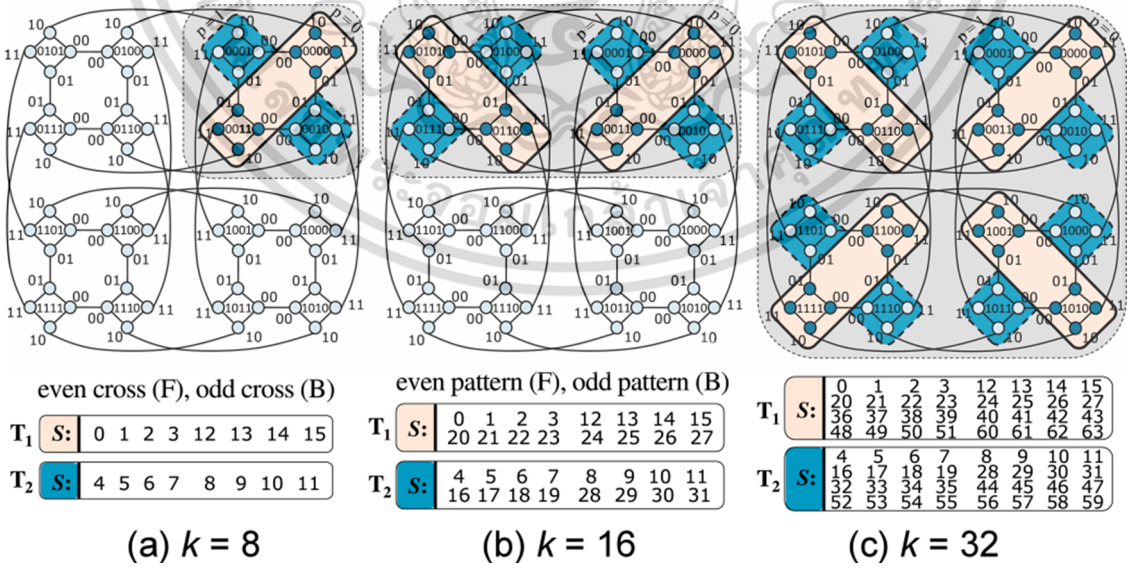


Fig. 22. An example ($N = 64$, $m = 2$, $k = 8, 16, 32$) of full group/s-group utilization on (even, odd) patterns with (F, B) protocol.

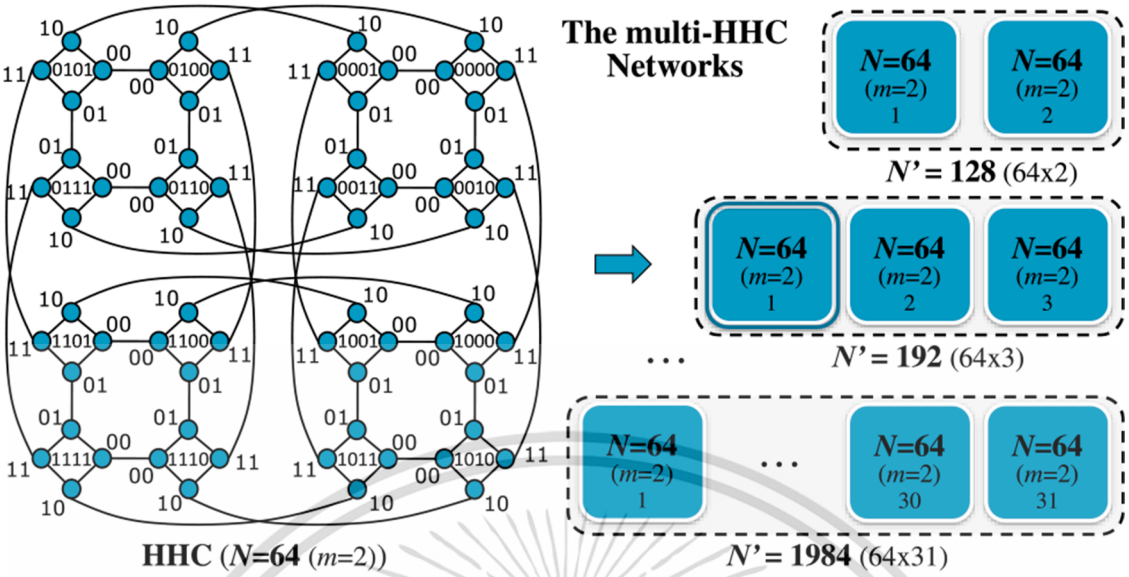


Fig. 23. The model of multi-HHC networks ($N' = Nx1$, N=64 (m=2)).

Table 11
Time complexity of sub-system allocation/deallocation.

(Best-fit) Allocation	Best Case	Worst Case
1. DBS for the request k	$O(2^m)$	$O(2^m+m) = O(n)$
2. allocate tree nodes	$O(2^m)$	$O(2^M)$
3. update accumulated *sum*	$O(2^m)$	$O(2^M)$
Total	$\mathbf{O(2^m)}$	$\mathbf{O(2^M)}$
Deallocation	**Best Case**	**Worst Case**
1. search in list for T_{finish}	O(1)	O(N)
2. free nodes in tree	O(1)	$O(2^M)$
3. update accumulated *sum*	$O(2^m)$	$O(2^M)$
Total	$\mathbf{O(2^m)}$	**O(N)**

Table 12 shows a routing table (RT) for the ATAPE communication ($k = 8$, $C = 5$) on $cross_0$ (group 0) of the RP-HHC ($N = 64$). The parallel SP-routing on $cross_0$ from k sources S = (0, 1, 2, 3, 12, 13, 14, 15) to k destinations D = (13, 12, 15, 14, 1, 0, 3, 2) was performed with no conflict since in each time t = 1, 2, 3, 4, 5 all messages are transferred through the different edges (i.e., at $t = 1$ all different edges are (0, 4), (1, 9), (2, 0), (3, 1), (12, 8), (13, 5), (14, 12), (15,13), etc.). This test ($C = 5$) reported "no-conflict" since in each of all times (t = 1 - 5) all edges are different.

For other k ($> 2^{m+1}$), we did the same evaluation for each of all patterns (p = 0, 1, .., 2^{M-1}-1). Next, all (even, odd) patterns with the (F, B) protocol were observed to confirm no conflict (when using the same dim-control (for all k (S, D)s)). Fig. 21 shows an example ($N = 64$, $k = 8$, $C = 5$) of parallel SP routing from S = (0-3, 12-15) to D = (13, 12, 15, 14, 1, 0, 3, 2) on $cross_0$ (F-protocol) plus the routing from S = (4-7, 8-11) to D = (9, 8, 11, 10, 5, 4, 7, 6) on $cross_1$ (B-protocol) at the same time. See Table 13 for other experimental results ($2^{m+1} \le k \le N/2^{M-1}$). Note: the dynamic environment (applying Algorithms 3 - 5) will be experimented (by a simulation study) in our future work.

In addition, Table 14 compares the contributions of our study (flexible k up to $N/2^{M-1}$ nodes per task) and the existing solutions [4,12, 15] (limited $k \le 2^{m+1}$ nodes per task).

Table 13
Experimental results of ATAPE SP-routing on RP-HHCs.

ATAPE on sub-systems $k \le N/2^{M-1}$ $N = 64$ ($m = 2$)	Partitions	SP-Routing
k=8: pattern 0 / cross 0 (F), pattern 1 (B)	in 4 groups	no conflict
k=16: pattern 0 (F), pattern 1 (B)	in 2 s-groups	no conflict
k=32: pattern 0 (F), pattern 1 (B)	in 1 s-group	no conflict
N = 2048 (m = 3)		
k=16: even patterns (F), odd-patterns (B)	in 16 groups	no conflict
k=32: even patterns (F), odd patterns (B)	in 8 s-groups	no conflict
k=64: even patterns (F), odd patterns (B)	in 4 s-groups	no conflict
k=128: even patterns(F), odd patterns(B)	in 2 s-groups	no conflict
k=256: even patterns(F), odd patterns(B)	in 1 s-group	no conflict

Table 12
Routing table for ATAPE ($N = 64$, $k = 8$, $C = 5$) on $cross_0$.

Clock/Time	1		2		3		4		5	
***S*=0**:0000,00	→	0001,00	→	0001,01	→	**13**:0011,01=***D***				
***S*=1**:0000,01	→	0010,01	→	0010,00	→	**12**:0011,00=***D***				
***S* =2**:0000,10	→	0000,00	→	0001,00	→	0001,01	→	0011,01	→	**15**:0011,11=***D***
***S* =3**:0000,11	→	0000,01	→	0010,01	→	0010,00	→	0011,00	→	**14**:0011,10=***D***
***S* =12**:0011,00	→	0010,00	→	0010,01	→	**1**:0000,01=***D***				
***S* =13**:0011,01	→	0001,01	→	0001,00	→	**0**:0000,00=***D***				
***S* =14**:0011,10	→	0011,00	→	0010,00	→	0010,01	→	0000,01	→	**3**:0000,11=***D***
***S* =15**:0011,11	→	0011,01	→	0001,01	→	0001,00	→	0000,00	→	**2**:0000,10=***D***

Table 14
Comparison of our contribution and existing methods.

Year	Routing Technique	Maximum task sizes $N=2^n$ ($n=2^m+m$)	N=64 (m=2)	N=2048 (m=3)
1994	Basic *N2N* routing [12].	$k = 2^m$	$k = 4$	$k = 8$
2013	*k*-pairwise disjoint path routing [4].	$k=(m+1)/2$	$k = 2$	$k = 2$
2021	Parallel *N2N*-SP routing and GCD partitioning [15].	$k = 2^{m+1}$	$k = 8$	$k = 16$
This study	GCS reconfiguring ($k \le N/2^{M-1}$) and optimal task-scheduling & efficient processor allocation.	$k = N/2^{M-1}$	$k = N/2 = 32$	$k = N/8 = 256$

Table 15
Features of HHC (of size *N*) and multi-HHC (of size *N*x*I*).

$N = 2^n$	m	Maximum task size $= N/2^{M-1}$	|Cross| $= 2^{m+1}$	#Groups $= 2^M$	#Crosses per group $= 2^{M-1}$
$64 = 2^5$	2	32	8	4	2
64 x I (128,192, …,1984)	2	32	8	4 x I	2
$2048 = 2^{11}$	3	256	16	16	8
2048 x I (4096, …, $< 2^{20}$)	3	256	16	16 x I	8
$1048576 = 2^{20}$	4	8192	32	256	128
2^{20} x I (2×2^{20}, …, $< 2^{37}$)	4	8192	32	256 x I	128
$128G = 2^{37}$	5	2^{22}	64	2^{16}	2^{15}
2^{37} x I (2×2^{37}, 3×2^{37},…)	5	2^{22}	64	2^{16} x I	2^{15}

Moreover, our RP-HHCs are suitable for the scalable multiprocessor systems for the regular $N = 2^n$ ($n = 2^m+m$, $m \ge 2$) and the expandable N x I for the multi-HHC (a pooling of HHCs (N processors per HHC) for a total of N x I processors), see the model in Fig. 23 and the features in Table 15 (i.e., $N = 64$ and $N' = 128, 192, 256, \ldots, 1984$, etc.).

Algorithm 1
Parallel shortest path for SP-routing on HHCs.

GetParallelSPathHHC

1 $P = \emptyset$; $\alpha^* = \alpha_S \oplus \alpha_D = \{b_{n-1}, b_{n-2}, \ldots, b_m\}$, $n = 2^m+m$, $m \ge 2$;
2 μ = GC mapping (index i of α^* if $b_i = 1$) = $\{gc_0, gc_1, .., gc_{r-1}\}$;
3 **if** ($\alpha_S = \alpha_D$) **then** set P_{in} (HC internal–path routing);
4 **else** ($\alpha_S \ne \alpha_D$)
5 **if** ($\beta_s \in \mu$) **then** $P = P \cup \{ \beta_s$,*d*SPordering ($\mu - \{\beta_s\}$)};
6 **else** $P = P \cup \{$*d*SPordering (μ)};
7 **return** *P*; // the best shortest-path (SP) order for SP-routing

***d*SPordering (μ)** // F: forward
temp = β_s; $r' = r$;
d = (index next to β_s in μ) %r;
while(|p| < r) **do**
for(i = 0; $i < r'$; i++) **do**
$id = (i + d)$ % r';
find $\mu[id]$ closest to *temp*;
temp = $\mu[id]$; $p = p \cup \mu[id]$;
update |μ|-1, $r' = r'$-1;
$d = (id+1)$%r'; **break**;
end for
end while (return *p*)

***d*SPordering (μ)** // B: backward
temp = β_s; $r' = r$;
d=((index back from β_s in μ)+r)%r;
while(|p| < r) **do**
for(i = 0; $i < r'$; i++) **do**
$id = (d - i + r')$ % r';
find $\mu[id]$ closest to *temp*;
temp = $\mu[id]$; $p = p \cup \mu[id]$;
update |μ|-1, $r' = r'$-1;
$d = (id-1+r')$%r'; **break**;
end for
end while (return *p*)

The forward *dSPreordering* works with the circular right-shift and the backward *dSPreordering* works with the circular left-shift (for routing in the opposite directions).

Algorithm 2
Pre-processing for global sources (in 2^{M-1} crosses).

GetSourceOfSubSystem (input: GCD_g)

1 $dCube1 = GCD_g$; $dCube2 = GCD_g + 2^{M+m}$-2^m; // for $cross_0$
2 **for** (cross $c = 0$; $c < 2^{M-1}$; c++) **do** // 2 cubes per cross
3 **for** (processor $p = 0$; $p < 2^m$; p++) **do** // 2×2^m processors
4 $S[p] = dCube1+p$; // S on dual Cube 1
5 $S[p+2^m] = dCube2+p$; // S on dual Cube 2
6 **end for**
7 $dCube1 = dCube1+2^m$; $dCube2 = dCube2$-2^m; // for next cross c
8 **end for**

Algorithm 3
(Simple) first-fit allocation on RP-HHCs.

Form the root, apply DFS (depth *first* search) for the request k of the incoming task.
Step1. Find regular *cube, cross* GCD, or *combined* GCS ($k \le N/2^{M-1}$).
Case1: $k \le 2^m$ (*cube* nodes).
Apply DFS (while $sum+k \le limit$) for the first free *cube*-node ($k \le 2^m$). If there is a free cube, then go to step 2. Otherwise, put the request in the waiting queue.
Case2: $k = 2^{m+1}$ (*cross* nodes (Section 3.1)).
Apply DFS (while $sum+k \le limit$) for the first free *cross*-node ($k = 2^{m+1}$). If there is a free cross, then go to step 2. Otherwise, put the request in the waiting queue.
Case3: $2^{m+2} \le k \le N/2^{M-1}$ (*combined* nodes (Sections 3.2 - 3.3)).
Apply DFS (while $sum+k \le limit$) for the first free GCS pattern (*main-net e* and pattern p). If there is a free sub-system, go to step 2. Otherwise, put the request in the waiting queue.
Step2. Add the allocated task (*main-net e* and pattern p) in the list at index l of the array-lists. Update the corresponding *sum* in each of the involved nodes in the tree and set blocking (if any).

Algorithm 4
(Efficient) best-fit allocation on RP-HHCs.

Form the root, apply DBS (depth *best* search) for the request k of the incoming task.
Step1. Find regular *cube, cross* GCD, or *combined* GCS ($k \le N/2^{M-1}$).
Steps 1-3 are similar to Algorithm 3, except replace DFS (depth *first* search) with DBS (depth *best* search). For DBS, compute L and R factors (LF, RF) of child nodes, where LF or RF = *limit* - ($sum+k$), before go to the best-fit node (min(LF, RF) ≥ 0) directly.
Step2. Add the allocated task (*main-net e* and pattern p) in the list at index l of the array-lists. Update the corresponding *sum* in each of the involved nodes in the tree and set blocking (if any).

Algorithm 5
Sub-system deallocation on RP-HHCs.

Step1. For a finished task (of k processors), search in the array-lists at index l (level l) = n-$\log_2 k$ to free the particular nodes in the tree.
Step2. Delete that finished task from the lists (at index i) and deallocate node (in tree) and combine free nodes up to the *cross*-level.
Step3. Update *sum* of all corresponding nodes up to the root and also update the blocking status (if any) at the *group*-level.

6. Conclusion

In this study, we proposed the reconfigurable and partitionable HHC (RP-HHC) networks ($N = 2^n$, $n = 2^m + m$) to support $2^{m+1} \le k \le N/2^{M-1}$ processors (per task) that can avoid the routing conflict. For the RP-HHC network, the conflict-free routing is very important since the HHC conflict cannot be handled easily during runtime. Thus, the innovation of this research is the optimal GCD-to-GCS reconfiguring ($2^{m+1} \le k \le N/2^{M-1}$), such as the maximum $k = N/2$ (for $N = 64$). Under GCD-to-GCS reconfiguring, all (even, odd) reconfigured patterns can be utilized at the same time by the (F, B) protocol if all (S, D)-connections are performed with the same dim-control (for full bidirectional-routing), observed by the ATAPE communication. In addition, the reconfigured binary-tree is incorporated to recognize the GCD and GCS patterns efficiently in the task scheduling and processor allocation (with the best-fit driven policy in O(N)). Finally, the experiments ($N \ge 64$) were conducted to confirm no routing conflict when performing the XOR-based ATAPE routing on the reconfigured partitions. In our future study, we will focus on evaluating the best-fit strategy (the dynamic environment) by a simulation study to observe the RP-HHC performance (i.e., system utilization and fragmentation, etc.).

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The authors wish to gratefully thank many people involving the (Ph. D.) research grant CW-012-2/2562 for Mr. Nuntipat Phisutthangkoon during 2013 – 2015 and 2020 – 2022 (the School of Science, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand) and deeply thank Phra-phrom Kunaporn (P.A. Pa-yut-tao) for his "Put-ta-tum" Buddhism handbook that can fulfill the positive thinking (one of the effective tools of this research). Lastly, the merit of this work will be glorious honor to the beloved King Bhumibol Adulyadej for the great inspiration of this study.

References

[1] M. Abd-El-Barr, T.F. Al-Somani, Performance comparison of hierarchical interconnection networks, in: IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, PacRim, August 2011, pp. 191–196.
[2] M. Abd-El-Barr, F. Gebali, Relibility analysis and fault tolerance for hypercube multi-computer networks, Inform. Sci. 276 (2014) 295–318.
[3] A. Bossard, K. Kaneko, Set-to-set disjoint-path routing in perfect hierarchical hypercube, in: The Fourth International C* Conference on Computer Science and Software Engineering, Canada, 2011, pp. 51–57.
[4] A. Bossard, K. Kaneko, k – Pairwise disjoint paths routing in perfect hierarchical hypercubes, J. Supercomput. 67 (2) (2013) 485–495.
[5] A. Bossard, K. Kaneko, S. Peng, Node-to-set disjoint-path routing inperfect hierarchical hypercubes, in: International Conference on Computational Science, ICCS 2011, Procedia Computer Science 4, 2011, pp. 442–451.
[6] K. Chronaki, et al., Task scheduling techniques for asymmetric multi-core systems, IEEE Trans. Parallel Distrib. Syst. 28 (7) (2017) 2074–2087.
[7] D.Das Sharma, D.K. Pradhan, Fast and efficient etrategies for cubic and non-cubic allocation in hypercube multiprocessors, in: Proc. Intl. Conf. Paralel Processing, Jan. 1993, pp. 118–127, vol. I.
[8] C. Lai, Optimal construction of all shortest node-disjoint paths in hypercubes with applications, IEEE Trans. Parallel Distrib. Syst. 23 (6) (2012) 1129–1134.
[9] C. Lai, An effcient construction of one-to-many node-disjoint paths in folded hypercubes, J. Parallel Distrib. Comput. 74 (4) (2014) 2310–2316.
[10] C. Lai, Optimal construction of node-disjoint shortest paths in folded hypercubes, J. Parallel Distrib. Comput. 102 (2017) 37–41.
[11] W. Lin, W. Shen, Tree-based task scheduling model and dynamic load-balancing algorithm for P2P computing, in: 10th IEEE International Conference on Computer and Information Technology-CIT, Guangzhou, China, South China University of Technology, 2010.
[12] Q.M. Malluhi, M.A. Bayoumi, The hierarchical hypercube: a new interconnection topology for massively parallel systems, IEEE Trans. Parallel Distrib. Syst. 5 (1) (1994) 17–30.
[13] R. Petagon, J. Werapun, Embedding the optimal all-to-all personalized exchange on multistage interconnection networks$^+$, J. Parallel Distrib. Comput. 88 (2016) 15–30.
[14] R. Petagon, J. Werapun, VA-DE: valuable ATAPE with dynamic embedding and super-pipeline scheduling on partitionable multistage interconnection network$^+$, J. Parallel Distrib. Comput. 102 (2017) 1–15.
[15] N. Pisutthangkoon, J. Werapun, Shortest-path routing for optimal all-to-all personalized-exchange embedding on hierarchical hypercube networks, J. Parallel Distrib. Comput. 150 (2021) 139–154.
[16] B. Qiao, F. Shi, W. Ji, THIN: a new hierarchical interconnection network-on-chip for SOC, in: The 7th international conference on Algorithms and architectures for parallel processing, ICA3PP'07, 2007, pp. 446–457.
[17] K. Qiu, An efficient disjoint shortest paths routing algorithm for the hypercube, in: 14th IEEE International Conference on Parallel and Distributed Systems, ICPADS'08, Dec. 2008, pp. 43–47.
[18] Y. Saad, M.H. Schultz, Topological properties of hypercubes, IEEE Trans. Comput. 37 (7) (1988) 867–872.
[19] D. Scott, Efficient all-to-all communication patterns in hypercube and mesh topologies, in: Distributed Memory Computing Conference, The Sixth, May 1991, pp. 398–403, apr-1.
[20] Y. Shi, Z. Hou, J. Song, Hierarchical Interconnection networks with folded hypercubes as basic clusters, in: The Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region 1, 2010, pp. 134–137.
[21] J. Srisawat (Werapun), A Unified Approach to Processor Allocation and Task Scheduling for Partitionable Parallel Architectures, The Department of Electrical and Computer Engineering, The George Washington University, Washington, D.C., U.S.A., 1999.
[22] T. Takabatake, K. Kaneko, H. Ito, Generalized hierarchical completely-connected networks, in: Fourth International Symposium on Parallel Architectures, Algorithms, and Networks, I-SPAN'99, Jun 1999, pp. 68–73.
[23] R.Y. Wu, G.H. Chen, Y.L. Kuo, G.J. Chang, Node-disjoint paths in hierarchical hypercube networks, Inform. Sci. 117 (19) (2007) 4200–4207.
[24] Q. Zhu, J.M. Xu, X. Hou, M. Xu, On reliability of the folded hypercubes, Inform. Sci. 177 (8) (2007) 1782–1788.
[25] D. Zydek, H. Selvaraj, Fast and efficient processor allocation algorithms for torus-based chip multiprocessor, Comput. Electr. Eng. 37 (2011) 91–105.

Nuntipat Phisutthangkoon received the B.S. degree in Applied Mathematics in 2011 and the M.S. degree in Computer Science in 2013 from King Mongkut's Institute of Technology Ladkrabang (KMITL), Bangkok, Thailand. Mr. Phisutthangkoon is a Ph.D. candidate (in Computer Science), at KMITL, Bangkok, Thailand. His research interests include parallel algorithms, parallel and distributed computing, parallel optimization, and hierarchical interconnection networks.

Jeeraporn Werapun received the B.S. degree from Kasetsart University and the M.S. degree from Chulalongkorn University, Bangkok, Thailand. She received the M.S. degree in Computer Science in 1993 and the D.Sc. degree in Computer Engineering (Parallel and Distributed Systems) in 1999 from the George Washington University, Washington, D.C., U.S.A. Dr. Werapun is an Associate Professor of Computer Science at King Mongkut's Institute of Technology, Ladkrabang (KMITL), Bangkok, Thailand. Her research interests include parallel algorithms and architectures, parallel and distributed computing, and multistage interconnection networks.

ประวัติผู้เขียน

ชื่อ	เรืออากาศตรี นันทิพัฒน์ พิศุทธางกูร	
วัน เดือน ปีเกิด	3 มีนาคม 2532	
ที่อยู่ปัจจุบัน	230 หมู่ 6 ต.บ้านโพธิ์ อ.เมืองตรัง จ.ตรัง	
ประวัติการศึกษา	(2554) วิทยาศาสตรบัณฑิต สาขาคณิตศาสตร์ประยุกต์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง	3.04
	(2556) วิทยาศาสตรมหาบัณฑิต สาขาวิทยาการคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง	3.62
ทุนการศึกษาที่ได้รับ	ทุนอุดหนุนการศึกษา คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง	
ผลงานวิชาการ	1. Phisutthangkoon, N. and Werapun, J. 2013. “Data Partitioning for Parallel Transpose of Matrix Multiplication.” *International Joint Conference on Computer Science and Software Engineering (JCSSE 2013).* 1 : 25–30.	
	2. Phisutthangkoon, N. and Werapun, J. 2021. “Shortest-Path Routing for Optimal All-to-All Personalized-Exchange Embedding on Hierarchical Hypercube.” *J. Parallel Distrib. Comput.* 150C : 139–154.	
	3. Phisutthangkoon, N. and Werapun, J. 2022. “Optimal ATAPE-Task Scheduling on Reconfigurable and Partitionable Hierarchical Hypercube Networks.” *Parallel Computing*. 111 : 102923.	