

การร่างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน
บนระบบเครือข่ายแบบไดนามิก

PARALLEL CONSTRUCTION OF INDEPENDENT SPANNING TREES
ON DYNAMIC NETWORKS



วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาทางคณิตศาสตร์ประยุกต์และวิทยาศาสตร์คอมพิวเตอร์

สาขาวิชาวิทยาการคอมพิวเตอร์

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2551

KMITL-2003-SC-M-020-853

การสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน
บนระบบเครือข่ายแบบไดนามิก

PARALLEL CONSTRUCTION OF INDEPENDENT SPANNING TREES
ON DYNAMIC NETWORKS



รอสลิน เพตะกร
ROSELIN PETAGON

เลขหมู่.....
เลขทะเบียน.....81359
วัน,เดือน,ปี...1.1 ส.ย. 2551

b.....
i.....

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาการคอมพิวเตอร์
บัณฑิตวิทยาลัย

เอกสารนี้เป็นเอกสารที่เผยแพร่ภายใต้การสงวนลิขสิทธิ์ของสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา พ.ศ. 2551 จนถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

**PARALLEL CONSTRUCTION OF INDEPENDENT SPANNING TREES
ON DYNAMIC NETWORKS**



**A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF SCIENCE IN COMPUTER SCIENCE
SCHOOL OF GRADUATE STUDIES**

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2008

KMITL – 2008 – SC – M – 020 - 353



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
COPYRIGHT 2008
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้
SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์

การสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนานบนระบบเครือข่ายแบบไดนามิก

Parallel Construction of Independent Spanning Trees on Dynamic Networks

ชื่อนักศึกษา

นางสาวรสลิน เพตะกร

รหัสประจำตัว

49067512

ปริญญา

วิทยาศาสตรมหาบัณฑิต

สาขาวิชา

วิทยาการคอมพิวเตอร์

อาจารย์ที่ปรึกษาวิทยานิพนธ์

ผศ.ดร.จิรพร

วีระพันธุ์

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
ผศ.ดร.ศรันย์	อินทโกสุม	
ผศ.ดร.จิรพร	วีระพันธุ์	
รศ.ดร.วีระ	บุญจริง	
ดร.เฉลิมศักดิ์	เลิศวงศ์เสถียร	

วัน / เดือน / ปี ที่สอบ 26 พฤษภาคม 2551 เวลา 18.00 น. เป็นต้นไป

สถานที่สอบ ณ อาคารจุฬารามวลัยลักษณ์ 1 ห้อง 210

บัณฑิตวิทยาลัยรับรองแล้ว

(รศ.ดร.รวีวรรณ ชินะตระกูล)

คณบดีบัณฑิตวิทยาลัย

วันที่ 30 เดือน พฤษภาคม พ.ศ. 2551

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หัวข้อวิทยานิพนธ์	การสร้างต้นไม้แบบทอดข้ามที่แตกต่างกันด้วยวิธีการแบบขนานบนระบบเครือข่ายแบบไดนามิก
นักศึกษา	นางสาวรสลิน เพตะกร
รหัสประจำตัว	49067512
ปริญญา	วิทยาศาสตรมหาบัณฑิต
สาขาวิชา	วิทยาการคอมพิวเตอร์
พ.ศ.	2551
อาจารย์ที่ปรึกษาวิทยานิพนธ์	ผศ.ดร. จีรพร วีระพันธุ์

บทคัดย่อ

งานวิจัยนี้นำเสนอขั้นตอนวิธีการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยจำนวนที่มากที่สุด บนระบบเครือข่ายแบบไดนามิก (Parallel Construction of Independent Spanning Trees on Dynamic Networks) ด้วยความซับซ้อนด้านเวลาเท่ากับ $O(k^2)$ ซึ่ง k คือจำนวนบิตที่ใช้แทนเลขประจำหน่วยประมวลผลที่ $N = 2^k$ หน่วย โดยงานวิจัยนี้ได้พัฒนาขั้นตอนวิธีดังกล่าวจากเครือข่ายแบบไฮเปอร์คิวบ (Hypercube Network) ให้ยืดหยุ่นมากขึ้นโดยการออกแบบและปรับปรุงวิธีการให้สามารถประมวลผลบนเครือข่ายแบบไดนามิก อาทิ เครือข่ายแบบคิวบ (Cube Network) เครือข่ายแบบโอเมก้า (Omega Network) และเครือข่ายแบบฟลิป (Flip Network) ได้ ซึ่งโครงสร้างต้นไม้แบบทอดข้ามที่ได้จากวิธีการที่นำเสนอจะมีความยาวของเส้นทางเฉลี่ยจากโหนดราก (Root Node) จนกระทั่งถึงโหนดใบ (Leaf Node) ในต้นไม้ที่มีค่าน้อยที่สุดเท่ากับความยาวเฉลี่ยของต้นไม้ที่น้อยที่สุดในเครือข่ายแบบไฮเปอร์คิวบ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Thesis	Parallel Construction of Independent Spanning Trees on Dynamic Networks
Student	Miss Roselin Petagon
Student ID.	49067512
Degree	Master of Science
Program	Computer Science
Year	2008
Thesis Advisor	Asst. Prof. Dr. Jeeraporn Werapun

ABSTRACT

This study presents a flexible parallel algorithm based on the Latin square scheme to generate an optimal amount of ISTs (Independent Spanning Trees) on dynamic networks. Time complexity of our proposed parallel algorithm is $O(k^2)$, where k is a number of bits, representing the address of each processing element (PE) of all PEs ($N = 2^k$). Our dynamic approach introduces the simple and flexible switch-setting rule and hence can easily be implemented on the fly on a variety of the dynamic networks such as Cube, Omega, Flip networks. Furthermore, the average path length of each generated tree is minimized similar to that of the efficient parallel ISTs algorithm on the static hypercube network.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กิตติกรรมประกาศ

วิทยานิพนธ์นี้มีโอกาสจะสำเร็จลุล่วงไปได้ด้วยดี หากมิได้รับคำแนะนำ คำชี้แจง ความรู้ และความเอาใจใส่จาก ผศ.ดร.จิรพร วีระพันธุ์ ผู้เป็นอาจารย์ที่ปรึกษา ซึ่งท่านได้สละเวลาให้กับข้าพเจ้าอย่างเต็มที่ จึงใคร่ขอขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณ รศ.ดร.วีระ บุญจริง ผศ.ดร.ศรัณย์ อินทโกสม และดร.เฉลิมศักดิ์ เลิศวงศ์เสถียร คณะกรรมการสอบวิทยานิพนธ์ ที่กรุณาให้คำแนะนำตลอดจนข้อชี้แนะจนในที่สุดทำให้วิทยานิพนธ์ฉบับนี้สำเร็จลงได้

ขอขอบพระคุณมหาวิทยาลัยราชภัฏเชียงใหม่ ที่สนับสนุนและดูแลเรื่องค่าใช้จ่ายต่างๆ ระหว่างศึกษาเป็นอย่างดีอีกด้วย

ขอขอบพระคุณบิดา มารดา ที่สนับสนุนให้ได้เรียนในระดับที่ได้ตั้งใจ และเป็นกำลังใจในระหว่างการศึกษาเป็นอย่างดี

ขอขอบคุณ นางสาวพัชรินทร์ บัวเย็น พี่ๆ และเพื่อนๆ ทุกคนที่ให้คำปรึกษา และช่วยอำนวยความสะดวกในด้านต่างๆ

สำหรับคุณงามความดีและประโยชน์อันใดที่เกิดขึ้นจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้กับบิดา มารดา อาจารย์ทุกท่านซึ่งเป็นที่เคารพรักยิ่ง ตลอดจนญาติพี่น้อง และเพื่อนๆ ทุกคน

รสลิน เพตะกร

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ

	หน้า
บทคัดย่อภาษาไทย.....	I
บทคัดย่ออังกฤษ.....	II
กิตติกรรมประกาศ.....	III
สารบัญ.....	IV
สารบัญตาราง	VII
สารบัญรูป	VIII
บทที่ 1 บทนำ.....	1
1.1 ความเป็นมาและความสำคัญของปัญหา.....	1
1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา.....	3
1.3 สมมติฐานของการศึกษา.....	3
1.4 ขอบเขตการวิจัย	4
1.5 ขั้นตอนการศึกษาและการดำเนินงานวิจัย	4
1.6 ประโยชน์ที่คาดว่าจะได้รับ.....	5
บทที่ 2 ทฤษฎีและงานวิจัยที่เกี่ยวข้อง	6
2.1 สถาปัตยกรรมคอมพิวเตอร์แบบขนาน	6
2.1.1 สถาปัตยกรรมคอมพิวเตอร์แบบขนานที่แบ่งตามกลไกการควบคุม	6
2.1.2 สถาปัตยกรรมคอมพิวเตอร์แบบขนานที่แบ่งตามองค์ประกอบของพื้นที่การใช้งาน	8
2.1.3 เครือข่ายการเชื่อมต่อระหว่างหน่วยประมวลผล.....	10
2.2 โครงสร้างเครือข่ายการเชื่อมต่อระหว่างหน่วยประมวลผลแบบสแตติก.....	11
2.2.1 เครือข่ายแบบเส้นตรงและวงแหวน	12
2.2.2 เครือข่ายการเชื่อมต่อแบบสมมาตร	12
2.2.3 เครือข่ายแบบดาข่าย.....	13
2.2.4 เครือข่ายแบบต้นไม้.....	14
2.2.5 เครือข่ายแบบไฮเปอร์คิว.....	16
2.3 โครงสร้างเครือข่ายการเชื่อมต่อระหว่างหน่วยประมวลผลแบบไดนามิก.....	17
2.3.1 ระบบเครือข่ายการเชื่อมต่อแบบคิจิตอลบัส.....	17
2.3.2 ระบบเครือข่ายการเชื่อมต่อแบบมัลติสแตจ	18

สารบัญ (ต่อ)

หน้า

2.2.3	เครื่องข่ายแบบคิว.....	21
2.2.4	เครื่องข่ายแบบโอเมก้า.....	22
2.2.5	เครื่องข่ายแบบฟลิป.....	24
2.4	ต้นไม้แบบทอดข้ามที่แตกต่าง.....	25
2.5	จตุรัสลาติน	25
2.6	การสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน บนเครื่องข่ายแบบไฮเปอร์คิว	26
2.6.1	ความรู้เบื้องต้นสำหรับการสร้างต้นไม้แบบทอดข้ามที่แตกต่าง.....	26
2.6.2	ขั้นตอนวิธีการสร้างต้นไม้แบบทอดข้ามที่แตกต่าง บนเครื่องข่ายแบบไฮเปอร์คิว k มิติ.....	30
บทที่ 3	การสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน บนระบบเครื่องข่ายแบบไดนามิก	37
3.1	ต้นไม้แบบทอดข้ามที่แตกต่าง.....	37
3.2	คุณสมบัติเบื้องต้น.....	38
3.3	ขั้นตอนวิธีการแบบขนานบนเครื่องข่ายแบบมัลติสเตจ สำหรับสร้างต้นไม้แบบทอดข้ามที่แตกต่าง.....	41
3.3.1	การสร้างฟังก์ชันเส้นทางบนเครื่องข่ายแบบมัลติสเตจ สำหรับสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน	42
3.3.2	การสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน โดยใช้ฟังก์ชันเส้นทางบนเครื่องข่ายแบบมัลติสเตจ	57
3.4	การประยุกต์ใช้การสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน โดยใช้ฟังก์ชันเส้นทางบนเครื่องข่ายมัลติสเตจ.....	60
3.4.1	การสร้างต้นไม้แบบทอดข้ามที่แตกต่างบนเครื่องข่ายแบบคิว.....	60
3.4.2	การสร้างต้นไม้แบบทอดข้ามที่แตกต่างเครื่องข่ายแบบโอเมก้า.....	63
3.4.3	การสร้างต้นไม้แบบทอดข้ามที่แตกต่าง บนเครื่องข่ายแบบฟลิป.....	67
บทที่ 4	การวิเคราะห์ความซับซ้อนของขั้นตอนวิธี.....	71
4.1	ความซับซ้อนด้านเวลาของขั้นตอนวิธีการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน บนระบบเครื่องข่ายแบบไดนามิก.....	71

สารบัญ (ต่อ)

หน้า

4.2 การคำนวณจำนวนของต้นไม้แบบทอดข้ามที่แตกต่างที่มีจำนวนมากที่สุด บนระบบ เครือข่ายแบบโคนามิก.....	74
4.3 การคำนวณเส้นทางที่แตกต่างกันของต้นไม้แบบทอดข้ามที่แตกต่าง บนระบบเครือข่าย แบบโคนามิก.....	78
บทที่ 5 บทสรุป.....	83
เอกสารอ้างอิง	85
ภาคผนวก.....	87
ผลงานวิจัยที่ได้รับการตีพิมพ์	
ประวัติผู้เขียน	95

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญตาราง

ตารางที่	หน้า
2.1 แสดงข้อมูลทั้งหมดสำหรับการสร้าง T_0 บน Q_4	31
2.2 แสดงข้อมูลทั้งหมดสำหรับการสร้าง T_1 บน Q_4	32
2.3 แสดงข้อมูลทั้งหมดสำหรับการสร้าง T_2 บน Q_4	33
2.4 แสดงข้อมูลทั้งหมดสำหรับการสร้าง T_3 บน Q_4	34
3.1 แสดงข้อมูลบางส่วนของฟังก์ชันเส้นทาง (R_0, R_1, R_2) สำหรับ T_0 ($i = 0$) เมื่อ $k = 3$	44
3.2 แสดงข้อมูลบางส่วนของฟังก์ชันเส้นทาง (R_0, R_1, R_2) สำหรับ T_1 ($i = 1$) เมื่อ $k = 3$	46
3.3 แสดงข้อมูลบางส่วนของฟังก์ชันเส้นทาง (R_0, R_1, R_2) สำหรับ T_2 ($i = 2$) เมื่อ $k = 3$	47
3.4 แสดงข้อมูลบางส่วนของฟังก์ชันเส้นทาง (R_0, R_1, R_2, R_3) สำหรับ T_0 ($i = 0$) เมื่อ $k = 4$	50
3.5 แสดงข้อมูลบางส่วนของฟังก์ชันเส้นทาง (R_0, R_1, R_2, R_3) สำหรับ T_1 ($i = 1$) เมื่อ $k = 4$	52
3.6 แสดงข้อมูลบางส่วนของฟังก์ชันเส้นทาง (R_0, R_1, R_2, R_3) สำหรับ T_2 ($i = 2$) เมื่อ $k = 4$	54
3.7 แสดงข้อมูลบางส่วนของฟังก์ชันเส้นทาง (R_0, R_1, R_2, R_3) สำหรับ T_3 ($i = 3$) เมื่อ $k = 4$	57

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป

รูปที่	หน้า
2.1 สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอสไอเอสดี	6
2.2 สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอสไอเอ็มดี	7
2.3 สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอ็มไอเอสดี	7
2.4 สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอ็มไอเอ็มดี	8
2.5 แบบจำลองระบบคอมพิวเตอร์แบบขนานที่ใช้การส่งผ่านข้อความ.....	9
2.6 แบบจำลองเครือข่ายแบบสแตติก	11
2.7 แบบจำลองคอมพิวเตอร์ในเครือข่ายแบบสแตติกที่ใช้สวิตช์	11
2.8 เครือข่ายแบบ (ก) เส้นตรงและ (ข) วงแหวนจำนวน N โหนด	12
2.9 เครือข่ายการเชื่อมต่อแบบสมบูรณ์ที่มีขนาด $N = 3 - 6$ โหนด	13
2.10 เครือข่ายแบบตาข่ายขนาด 2 มิติ	13
2.11 เครือข่ายแบบทอรัสขนาด 3×3	14
2.12 ระบบเครือข่ายแบบต้นไม้ทวิภาค	15
2.13 เครือข่ายต้นไม้แบบอ้วนขนาด $N = 16$	15
2.14 เครือข่ายแบบไฮเปอร์คิวบ 1 - 2 มิติ	16
2.15 เครือข่ายแบบไฮเปอร์คิวบ 3 มิติ ขนาด $N = 8$	16
2.16 เครือข่ายแบบไฮเปอร์คิวบ 4 มิติ ขนาด $N = 16$	17
2.17 ระบบเครือข่ายการเชื่อมต่อแบบดิจิทัลสลับ	18
2.18 ระบบเครือข่ายการเชื่อมต่อแบบมัลติสเตจ	18
2.19 แสดงลักษณะการเชื่อมต่อของสวิตช์ขนาด 2×2 แบบ 1-1	19
2.20 แสดงลักษณะการเชื่อมต่อของสวิตช์ขนาด 2×2 แบบ 1-n	19
2.21 แสดงลักษณะการเชื่อมต่อของสวิตช์ขนาด 2×2 ($N=8$)	19
2.22 รูปแบบการเชื่อมต่อระหว่างสเตจแบบเฟอร์เฟลซัปเฟล ($N = 8$)	20
2.23 รูปแบบการเชื่อมต่อระหว่างสเตจแบบอินเวอร์สเฟอร์เฟลซัปเฟล ($N = 8$)	20
2.24 ตัวอย่างหนึ่งของกราฟที่แสดงการเรียงสับเปลี่ยนที่ $N = 5$ (a, b, c, d, e, f) โดยฟังก์ชัน $\pi = (a, b, c)(d, e)$	21

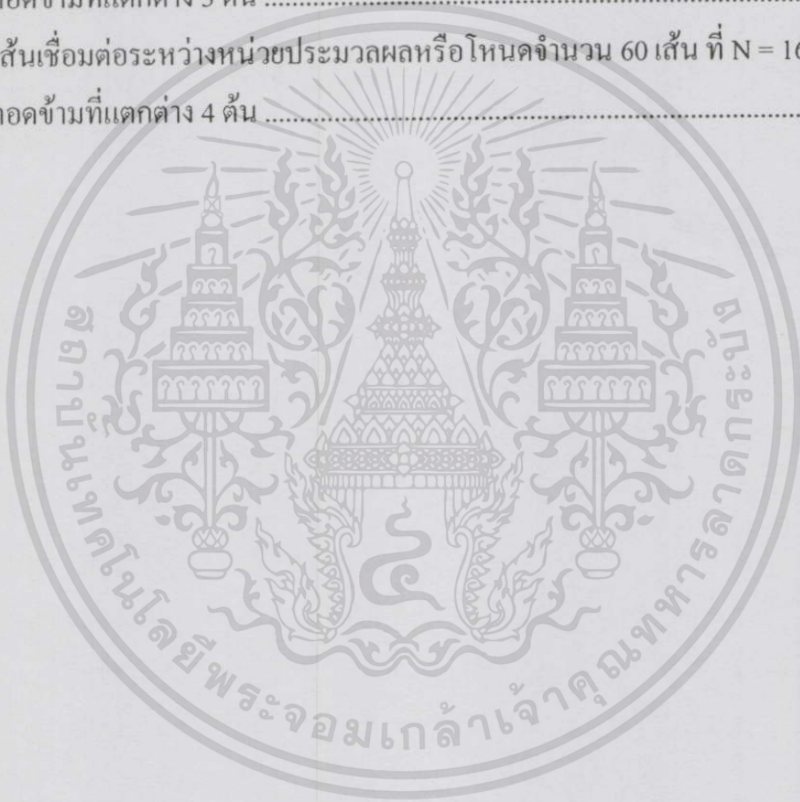
เอกสารนี้เป็นเอกสารที่สงวนลิขสิทธิ์ไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่สามารถนำไปใช้ประโยชน์ด้านอื่นๆ
 ไม่ว่าการตีความใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป (ต่อ)

รูปที่	หน้า
2.26 แสดงการเชื่อมต่อบนเครือข่ายแบบคิวนำสำหรับฟังก์ชันเส้นทาง E_i ($N = 8$)	23
2.27 ลักษณะการเชื่อมต่อระหว่างสแตจของเครือข่ายแบบโอเมก้า ($N = 8$)	24
2.28 ลักษณะการเชื่อมต่อระหว่างสแตจของเครือข่ายแบบฟลิป ($N = 8$)	24
2.29 แสดงการเชื่อมต่อของเครือข่ายแบบไฮเปอร์คิวบ 4 มิติ	26
2.30 แสดงชุดของต้นไม้แบบทอดข้ามที่แตกต่างกัน 4 แบบ (T_0, T_1, T_2, T_3) บนเครือข่ายแบบไฮเปอร์คิวบ 4 มิติ	27
3.1 ชุดต้นไม้แบบทอดข้ามที่แตกต่างกันบนระบบเครือข่ายแบบไดนามิก ($N = 8$)	38
3.2 ชุดต้นไม้แบบทอดข้ามที่แตกต่างกันบนระบบเครือข่ายแบบไดนามิก ($N = 16$)	38
3.3 แสดงการเชื่อมต่อระหว่างโหนด X และ Y	42
3.4 แสดงฟังก์ชันเส้นทาง 3 ฟังก์ชันสำหรับสร้างต้นไม้แบบทอดข้ามที่แตกต่างกันบนเครือข่ายแบบมัลติสแตจ ($N = 8$)	58
3.5 แสดงฟังก์ชันเส้นทาง 4 ฟังก์ชันสำหรับสร้างต้นไม้แบบทอดข้ามที่แตกต่างกันบนเครือข่ายแบบมัลติสแตจ ($N = 16$)	59
3.6 แสดงฟังก์ชันเส้นทาง 3 ฟังก์ชัน สำหรับสร้างต้นไม้แบบทอดข้าม บนเครือข่ายแบบคิวนำ ($N = 8$)	61
3.7 แสดงฟังก์ชันเส้นทาง 4 ฟังก์ชัน สำหรับสร้างต้นไม้แบบทอดข้าม บนเครือข่ายแบบคิวนำ ($N = 16$)	62
3.8 แสดงฟังก์ชันเส้นทาง 3 ฟังก์ชัน สำหรับสร้างต้นไม้แบบทอดข้าม บนระบบเครือข่ายแบบโอเมก้า ($N = 8$)	64
3.9 แสดงฟังก์ชันเส้นทาง 4 ฟังก์ชัน สำหรับสร้างต้นไม้แบบทอดข้าม บนเครือข่ายแบบโอเมก้า ($N = 16$)	65
3.10 แสดงฟังก์ชันเส้นทาง 3 ฟังก์ชัน สำหรับสร้างต้นไม้แบบทอดข้าม บนเครือข่ายแบบฟลิป ($N = 8$)	67
3.11 แสดงฟังก์ชันเส้นทาง 4 ฟังก์ชัน สำหรับสร้างต้นไม้แบบทอดข้าม บนเครือข่ายแบบฟลิป ($N = 16$)	69
4.1 แสดงสแตจจำนวน k สแตจ	72
4.2 แสดงการเชื่อมต่อระหว่าง Y กับ X โดยการศึกษาเท่านั้น ไม่คอมพิวเตอร์ในแง่ไปใช้ประโยชน์ด้วย	73
4.3 แสดงฟังก์ชันการเชื่อมต่อระหว่างหน่วยประมวลกับหนึ่งบนเครือข่ายแบบมัลติสแตจ ที่มีจำนวน 4 หน่วยประมวลผลเมื่อ $k=2$	74

สารบัญรูป (ต่อ)

รูปที่	หน้า
4.4 แสดงจำนวนเส้นเชื่อมจำนวน 2 เส้น ที่ $N = 2$ และ $k = 1$	75
4.5 แสดงจำนวนเส้นเชื่อมจำนวน 8 เส้น ที่ $N = 4$ และ $k = 2$	75
4.6 แสดงจำนวนเส้นเชื่อมจำนวน 24 เส้นทาง ที่ $N = 8$ และ $k = 3$	75
4.7 แสดงจำนวนเส้นเชื่อมจำนวน 64 เส้นทาง ที่ $N = 16$ และ $k = 4$	76
4.8 แสดงเส้นเชื่อมต่อระหว่างหน่วยประมวลผลหรือโหนดจำนวน 21 เส้นที่ $N = 8$ ของต้นไม้ แบบทอดข้ามที่แตกต่าง 3 ต้น	77
4.9 แสดงเส้นเชื่อมต่อระหว่างหน่วยประมวลผลหรือโหนดจำนวน 60 เส้น ที่ $N = 16$ ของต้นไม้ แบบทอดข้ามที่แตกต่าง 4 ต้น	77



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันปัญหาหลายปัญหาเกี่ยวกับระบบเครือข่าย (Network System) อาจจะใช้โครงสร้างข้อมูลแบบกราฟ (Graph: $G(V, E)$) ในการอธิบายองค์ประกอบของระบบเครือข่ายนั้นๆ เพื่อที่จะสามารถหาคำตอบที่ซับซ้อนของปัญหาต่างๆ ได้ง่ายขึ้น และปัญหาที่น่าสนใจปัญหาหนึ่งที่มีการประยุกต์ใช้กราฟได้อย่างมีประสิทธิภาพ เพื่อการแก้ปัญหาได้ง่ายยิ่งขึ้นในยุคเทคโนโลยีสารสนเทศในปัจจุบัน อาทิ ปัญหาการกระจายข้อมูล (Data Broadcasting) ที่มีความน่าเชื่อถือสูงของระบบเครือข่ายคอมพิวเตอร์ (Computer Network) และระบบคอมพิวเตอร์แบบขนาน (Parallel Computers) โดยในระบบดังกล่าวโหนดต่างๆ (Nodes: V) ในกราฟ G แทนคอมพิวเตอร์ในระบบเครือข่ายคอมพิวเตอร์แบบทั่วไป หรือหน่วยประมวลผลในระบบเครือข่ายคอมพิวเตอร์แบบขนาน และเส้นขอบ (Edges: E) ในกราฟ G แทนลิงค์ต่างๆ ระหว่างโหนด

การกระจายข้อมูลไปในเครือข่าย หมายถึงกระบวนการส่งข้อมูลหรือข้อความจากโหนดต้นทาง (Source: S) ไปยังโหนดปลายทาง (Destination) หรือโหนดอื่นๆ ทั้งหมดที่เป็นโหนดสมาชิกในระบบเครือข่าย และในทางปฏิบัติกระบวนการกระจายข้อมูลอาจจะไม่สามารถทำได้สำเร็จในบางกรณี ซึ่งสาเหตุอาจเนื่องมาจากการขัดข้องที่อาจจะเกิดขึ้นที่โหนดหรือที่ลิงค์ในขณะที่รับ/ส่งข้อมูล ทำให้ไม่สามารถส่งข้อมูลต่อไปยังโหนดปลายทางบางโหนดได้สำเร็จ ทั้งนี้กระบวนการกระจายข้อมูลจะสำเร็จได้ ก็ต่อเมื่อ ทุกๆ โหนดที่เป็นโหนดปลายทางในเครือข่ายได้รับข้อมูลจากต้นทางภายในเวลาที่กำหนด แต่ในทางปฏิบัติการกระจายข้อมูลบางครั้งอาจไม่ประสบผลสำเร็จ และสาเหตุที่มักพบได้บ่อยๆ คือลิงค์ระหว่างโหนดคู่ใดๆ เกิดขัดข้อง หรือลิงค์ดังกล่าวไม่ว่างหรือไม่พร้อมที่จะรับ/ส่งข้อมูล เนื่องจากในขณะนั้นอาจจะมียังข้อมูลถูกส่งผ่านลิงค์นั้นเป็นจำนวนมาก ทำให้การกระจายข้อมูลในครั้งนั้นไม่สำเร็จไปด้วย โหนดผู้ส่งหรือโหนดต้นทางจึงต้องรอเวลาเพื่อเริ่มทำการส่งหรือกระจายข้อมูลใหม่อีกครั้ง ซึ่งจะสามารถทำได้หลังจากที่มีการแก้ปัญหาที่ลิงค์นั้นได้แล้ว หรือลิงค์นั้นว่างและพร้อมที่จะรับ/ส่งข้อมูลได้แล้ว ซึ่งอาจจะใช้เวลารอเป็นเวลานานเกินกำหนด ดังนั้นจึงต้องมีวิธีการจัดการที่เหมาะสม ณ เวลานั้น โดยไม่จำเป็นต้องรอ เช่น ควรจะต้องมีชุดเส้นทางเลือกอื่นที่ไม่ผ่านลิงค์ที่ไม่พร้อมนั้น สำรองไว้หลายๆ ชุดสำหรับเอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านอื่นๆ ความน่าเชื่อถือที่สูงขึ้น เพื่อรองรับการขัดข้องที่อาจเกิดขึ้นจากหลายๆ ลิงค์ในระบบเครือข่ายนั้น ไม่ว่ากรณีใดๆ เช่น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้ ด้วยเหตุนี้จึงต้องการชุดเส้นทางที่แตกต่างเพื่อการกระจายข้อมูลจำนวน k ชุด ($k > 1$) เพื่อแก้ปัญหาดังกล่าวและเพื่อประยุกต์ใช้ในกรณีอื่นๆ นักวิจัยได้ทำการศึกษาเพื่อกำหนดชุดเส้นทาง

เลือกดังกล่าว จากการพิจารณาโครงสร้างต้นไม้แบบทอดข้าม (Spanning Trees: STs) ชนิดหนึ่งจากกราฟเครือข่าย $G(V, E)$ และโครงสร้างชนิดนี้ จะเรียกว่า ต้นไม้แบบทอดข้ามที่แตกต่าง (Independent Spanning Trees: ISTs) ที่มีโหนดราก (Root Node: r) หรือโหนดต้นทางเดียวกัน การสร้างต้นไม้แบบทอดข้ามที่แตกต่างจำนวน k ต้น โดยมีโหนดราก r เดียวกัน จึงเป็นทางเลือกหนึ่งที่น่าสนใจในการนำไปประยุกต์ใช้เพื่อเพิ่มความน่าเชื่อถือในการกระจายข้อมูลที่อาจกล่าวได้ว่า มีความสามารถทนต่อความผิดพลาดในการกระจายข้อมูล (Fault-Tolerant Broadcasting in Networks) ได้สูงถึง k เท่า

หลักการโดยทั่วไปของกระบวนการกระจายข้อมูลที่มีความน่าเชื่อถือสูง (Reliable Broadcasting) หรือทนต่อความผิดพลาดสูง (Fault-Tolerant Broadcasting) ดังกล่าว ที่ได้จากการสร้างต้นไม้แบบทอดข้ามที่แตกต่างจำนวน k ต้น จะเป็นการส่งข้อมูลไปตามต้นไม้แบบทอดข้ามที่แตกต่าง จากโหนดราก (r) หรือโหนดต้นทาง ไปพร้อมกันทั้ง k ต้น และในขณะนั้น ถ้ามีลิงค์ใดเกิดขัดข้องหรือไม่พร้อมที่จะรับ/ส่งข้อมูล การกระจายข้อมูลก็ยังสามารถดำเนินต่อไปได้จนสำเร็จ เนื่องจากยังมีช่องทางอื่น ที่จะสามารถส่งข้อมูลต่อไปได้ จนครบทุกโหนดปลายทางในเครือข่าย โดยไม่จำเป็นต้องผ่านลิงค์นั้น

ในอดีตที่ผ่านมาได้มีการศึกษางานวิจัยเกี่ยวกับการสร้างต้นไม้แบบทอดข้ามที่แตกต่าง ด้วยวิธีการแบบขนานมากมายบนหลายรูปแบบเครือข่าย ไม่ว่าจะเป็นเครือข่ายแบบกราฟเชิงระนาบ (Planar Graphs Network) [4, 5], เครือข่ายแบบคอर्डอลริง (Chordal Rings Network) [6, 17], เครือข่ายแบบกราฟผลคูณ (Product Graphs Network) [9], เครือข่ายแบบไฮเปอร์คิว (Hypercubes Network) [16, 18], เครือข่ายทอรัสแบบหลายมิติ (Multidimensional Tori Network) [14] เป็นต้น จากผลงานการวิจัยที่ผ่านมาพบว่า การพัฒนาขั้นตอนวิธีสำหรับการสร้างต้นไม้แบบทอดข้ามที่แตกต่างมี 2 แนวทาง คือ 1) การออกแบบขั้นตอนวิธีสร้างต้นไม้แบบทอดข้ามที่แตกต่างให้มีประสิทธิภาพด้านเวลาที่ดีขึ้น และ 2) การลดความสูงของการสร้างต้นไม้แบบทอดข้าม โดยผลงานวิจัยในตอนแรกเป็นการออกแบบขั้นตอนวิธีที่มีความซับซ้อนด้วยเวลาเป็นเส้นตรง คือ $O(d)$ วิธีนี้จะเป็นการสร้างต้นไม้แบบทอดข้ามที่แตกต่างบนเครือข่ายแบบคอर्डอลริง $CR(N, d)$ ขนาด N โดย d เป็นจำนวนของดีกรีของโหนดที่กำหนดให้และโดยทั่วไปจะกำหนดให้ d เป็น 3 หรือ 4 ดีกรี โดยเมื่อดีกรี $d = 4$ สามารถสร้างต้นไม้แบบทอดข้ามที่แตกต่างได้ 4 ต้น และเมื่อเร็วๆ นี้มีการนำเสนอการสร้างต้นไม้แบบทอดข้ามที่แตกต่างบนเครือข่ายแบบไฮเปอร์คิว (Hypercube Network) บนพื้นฐานของหลักการที่เรียกว่า จตุรัสลาตินของระยะทางแฮมมิง (Hamming Distance Latin Square Scheme: HDLS) บนเครือข่ายไฮเปอร์คิว k มิติขนาด $N = 2^k$ ด้วยความซับซ้อนด้านเวลาเอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าเท่ากับ $O(k)$ ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้ งานวิจัยทั้งหมดที่กล่าวมาแล้วทั้ง 2 แนวทางเป็นการนำเสนอขั้นตอนวิธีการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน (Parallel Construction of Independent

Spanning Tree) บนระบบเครือข่ายการเชื่อมต่อแบบสแตติก (Static Interconnection Network) เท่านั้น กล่าวคือ ต้นไม้แบบทอดข้ามบนเครือข่ายสแตติกแบบหนึ่ง จะแตกต่างกับต้นไม้แบบทอดข้ามบนเครือข่ายสแตติกแบบอื่นๆ และไม่สามารถนำมาประยุกต์ใช้ร่วมกันได้

งานวิจัยนี้ได้นำเสนอขั้นตอนวิธีการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน บนระบบเครือข่ายแบบไดนามิก (Parallel Construction of ISTs on Dynamic Networks) โดยในงานวิจัยนี้จะนำเสนอวิธีการใหม่ซึ่งปรับปรุงมาจากการสร้างต้นไม้แบบทอดข้ามที่แตกต่างแบบขนานบนเครือข่ายไฮเปอร์คิวบ (Parallel Construction of ISTs on Hypercube Network) ซึ่งระบบเครือข่ายแบบไดนามิก (Dynamic Networks) จะมีความยืดหยุ่นในการประมวลผล เนื่องจากสามารถกำหนดค่าสวิตช์ที่สามารถเข้าคู่ได้กับการติดต่อสื่อสารแบบที่ต้องการได้ในโปรแกรมของผู้ใช้ และสามารถสร้างต้นไม้แบบทอดข้าม บนระบบเครือข่ายแบบไดนามิกได้หลากหลาย อาทิ เครือข่ายแบบคิวบ (Cube Network) เครือข่ายแบบโอเมก้า (Omega Network) และเครือข่ายแบบฟลิป (Flip Network) ซึ่งเครือข่ายแบบต่างๆ ดังกล่าวสามารถประยุกต์ใช้ขั้นตอนวิธีที่ออกแบบไว้ร่วมกันได้ และต้นไม้แบบทอดข้ามที่แตกต่างที่สร้างได้จากเครือข่ายแบบไดนามิกที่ได้นำเสนอ จะมีความยาวของเส้นทางเฉลี่ยของแต่ละต้นไม้ที่น้อยที่สุด เหมือนกับการสร้างบนเครือข่ายไฮเปอร์คิวบที่มีขนาดเท่ากัน

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

วิทยานิพนธ์นี้มีวัตถุประสงค์เพื่อศึกษาขั้นตอนวิธีการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน บนระบบเครือข่ายแบบไดนามิก ดังนี้

- 1) ออกแบบ และปรับวิธีการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน บนเครือข่ายแบบไฮเปอร์คิวบ ให้สามารถสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน บนระบบเครือข่ายแบบไดนามิก
- 2) ทำการประยุกต์ใช้ขั้นตอนวิธีที่ออกแบบในการสร้างต้นไม้แบบทอดข้ามด้วยวิธีการแบบขนาน บนระบบเครือข่ายแบบไดนามิก เช่น เครือข่ายแบบคิวบ (Cube Network) แบบโอเมก้า (Omega Network) และแบบฟลิป (Flip Network)

1.3 สมมติฐานของการศึกษา

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น การเพิ่มประสิทธิภาพในด้านระบบเครือข่ายการเชื่อมต่อที่ยืดหยุ่น โดยการออกแบบและปรับปรุงขั้นตอนวิธีการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน บนระบบ

เครือข่ายการเชื่อมต่อแบบสแตติก เพื่อให้สามารถสร้างต้นไม้แบบทอดข้ามได้ บนระบบเครือข่ายแบบไดนามิกซึ่งเป็นระบบที่รองรับการทำงานที่ยืดหยุ่น โดยการกำหนดฟังก์ชันเส้นทาง(Routing Function) ที่เหมาะสม และการปรับค่าสวิตซ์ตามฟังก์ชันที่ต้องการให้สามารถประยุกต์ใช้ได้ ในหลายเครือข่าย อาทิเครือข่ายแบบคิว (Cube Network) แบบโอเมก้า (Omega Network) และแบบฟลิป (Flip Network)

1.4 ขอบเขตการวิจัย

วิทยานิพนธ์นี้มีขอบเขตการวิจัยเพื่อทำการศึกษาค้นคว้าขั้นตอนวิธีการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนานบนระบบเครือข่ายแบบไดนามิก เช่นเครือข่ายแบบคิว (Cube Network) แบบโอเมก้า (Omega Network) และแบบฟลิป (Flip Network) เมื่อกำหนดให้โหนดต้นทางที่ r เป็นโหนด 0 ($r = 0$)

1.5 ขั้นตอนการศึกษาและดำเนินงานวิจัย

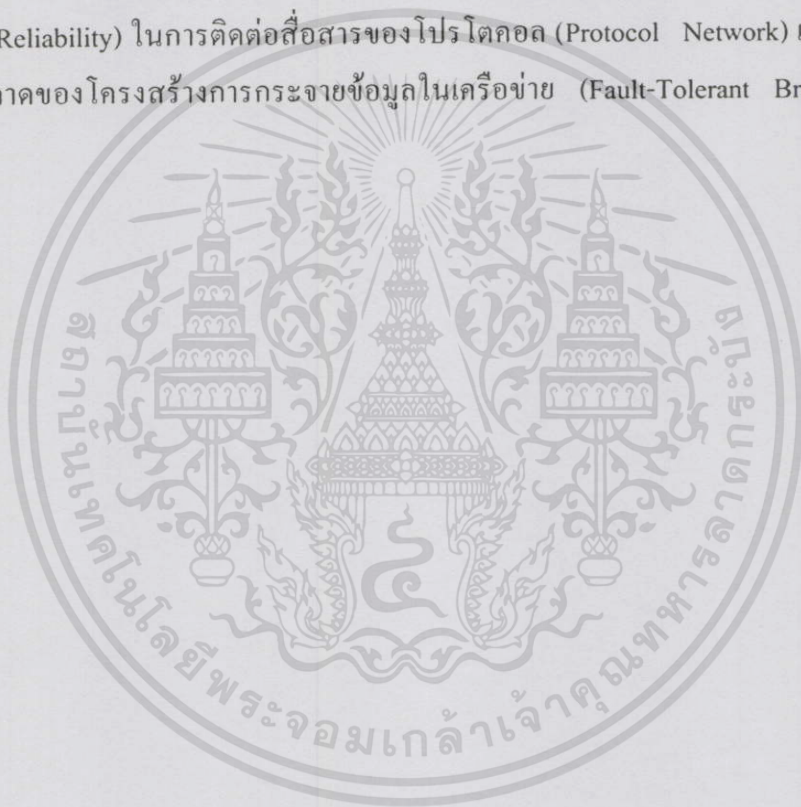
วิทยานิพนธ์นี้มีขั้นตอนการศึกษาและการดำเนินงานวิจัย ดังนี้

- 1) ศึกษาขั้นตอนวิธีการการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนานบนระบบเครือข่ายแบบสแตติก เช่น คอร์ดอลริง (Chordal Rings) ไฮเปอร์คิว (Hypercubes) และทอรัสแบบหลายมิติ (Multidimensional Tori) รวมทั้งศึกษางานวิจัยที่เกี่ยวข้องกับระบบเครือข่ายแบบไดนามิก
- 2) ทำการตั้งสมมติฐานโดยคาดว่าขั้นตอนวิธีการออกแบบฟังก์ชันเส้นทาง(Routing function) ที่เหมาะสม และการปรับค่าสวิตซ์ตามฟังก์ชันดังกล่าวทำให้สามารถสร้างต้นไม้แบบทอดข้ามด้วยวิธีการแบบขนาน บนระบบเครือข่ายแบบไดนามิก เช่นเครือข่ายแบบคิว (Cube Network) แบบโอเมก้า (Omega Network) และแบบฟลิป (Flip Network)
- 3) ออกแบบขั้นตอนวิธีใหม่จากการปรับปรุงจากวิธีการแบบขนานบนระบบเครือข่ายแบบสแตติกโดยเฉพาะไฮเปอร์คิว เพื่อใช้ในการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนานบนระบบเครือข่ายแบบไดนามิก ซึ่งมีความยืดหยุ่นสูงกว่าแบบสแตติก
- 4) วิเคราะห์ความซับซ้อนด้านเวลาของขั้นตอนวิธีของการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนานบนระบบเครือข่ายแบบไดนามิก สรุปผล พร้อมเสนอแนวทางการพัฒนางานวิจัย อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้
- 5) เขียนวิทยานิพนธ์

1.6 ประโยชน์ที่คาดว่าจะได้รับ

วิทยานิพนธ์นี้มีประโยชน์ที่คาดว่าจะได้รับ ดังนี้

- 1) ทำให้ได้ขั้นตอนวิธีแบบขนานที่ยืดหยุ่นในการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน บนระบบเครือข่ายแบบไดนามิก
- 2) ทำให้สามารถสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนานบนระบบเครือข่ายแบบไดนามิกหลายแบบ เช่น เครือข่ายแบบคิว (Cube Network) แบบโอเมก้า (Omega Network) และแบบฟลิป (Flip Network)
- 3) สามารถนำแนวคิดจากขั้นตอนวิธีที่นำเสนอ ไปประยุกต์ใช้กับการพัฒนาความน่าเชื่อถือ (Reliability) ในการติดต่อสื่อสารของโปรโตคอล (Protocol Network) และการทนต่อความผิดพลาดของโครงสร้างการกระจายข้อมูลในเครือข่าย (Fault-Tolerant Broadcasting in Networks)



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 2

ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

เนื้อหาในบทนี้จะกล่าวถึงทฤษฎีที่เกี่ยวข้องกับสถาปัตยกรรมคอมพิวเตอร์แบบขนาน ก่อน โดยเน้นโครงสร้างเครือข่ายการเชื่อมต่อระหว่างหน่วยประมวลผลแบบสแตติก (Static Interconnection Network) และเครือข่ายการเชื่อมต่อระหว่างหน่วยประมวลผลแบบไดนามิก (Dynamic Interconnection Network) จากนั้นจะกล่าวถึงทฤษฎีงานวิจัยที่เกี่ยวข้องในการสร้าง ต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน

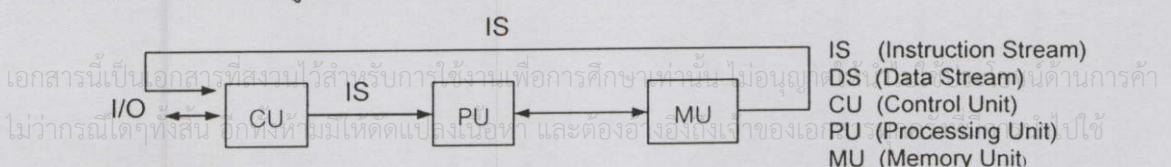
2.1 สถาปัตยกรรมคอมพิวเตอร์แบบขนาน

สถาปัตยกรรมคอมพิวเตอร์แบบขนาน (A Taxonomy of Parallel Architecture) นี้ สามารถจำแนกได้หลายรูปแบบขึ้นอยู่กับปัจจัยที่เราให้ความสนใจ ซึ่ง Vipin, K. และคณะ [8] ได้ แบ่งสถาปัตยกรรมคอมพิวเตอร์แบบขนานตามปัจจัยต่างๆ ได้ดังต่อไปนี้คือ กลไกการควบคุม (Control Mechanism) องค์ประกอบของพื้นที่การใช้งาน (Address-Space Organization) การติดต่อสื่อสารระหว่างหน่วยประมวลผล (Interconnection Network) ขนาดและจำนวนของหน่วยประมวลผล (Granularity of Processors)

2.1.1 สถาปัตยกรรมคอมพิวเตอร์แบบขนานที่แบ่งตามกลไกการควบคุม

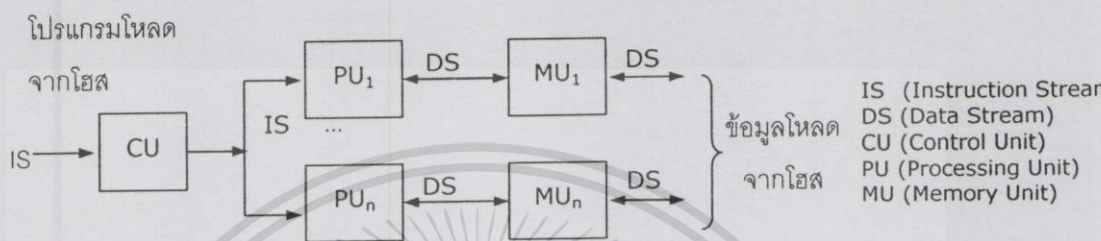
หน่วยประมวลผลในระบบคอมพิวเตอร์ไม่ว่าจะเป็นการทำงานภายใต้ส่วนควบคุม กลางหรือ ส่วนควบคุมที่มีการทำงานที่เป็นอิสระแก่กัน ซึ่งในสถาปัตยกรรมของ Flynn [12] ได้ จำแนกออกเป็น 4 ชนิดเมื่อมองในแง่ของคำสั่ง และข้อมูลที่ถูกประมวลผล คือ

2.1.1.1 สถาปัตยกรรมแบบเอสไอเอสดี (SISD: Single Instruction over Single Data) หมายถึงระบบที่ ณ เวลาหนึ่งๆ จะมีคำสั่งเดียว (Single Instruction) ทำงานอยู่บนข้อมูลค่าเดียว (Single Data) ซึ่งเครื่องคอมพิวเตอร์ทุกๆ ไปที่มีหน่วยประมวลผลเดียวจะมีความสามารถแบบนี้อยู่แล้ว



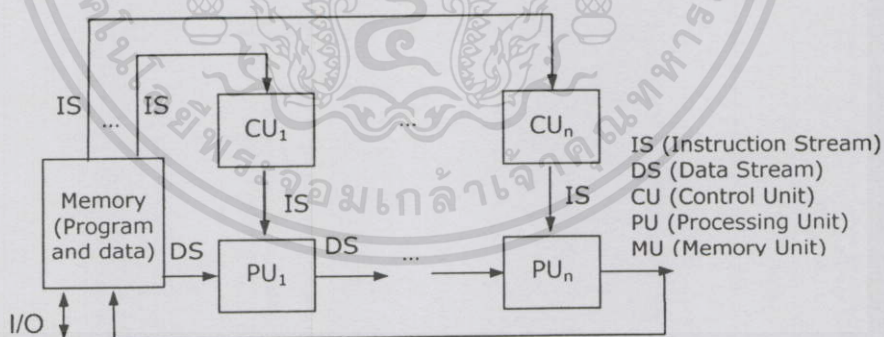
รูปที่ 2.1 สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอสไอเอสดี

2.1.1.2 สถาปัตยกรรมแบบเอสไอเอ็มดี (SIMD: Single Instruction over Multiple Data) หมายถึงระบบที่ ณ เวลาหนึ่งๆ จะมีการประมวลผลคำสั่งอยู่ 1 คำสั่ง ซึ่งจะถูกส่งไปให้หน่วยประมวลผลต่างๆ ในระบบ และทุกหน่วยประมวลผลจะประมวลผลคำสั่งเดียวกัน (Single Instruction) แต่จะปฏิบัติการอยู่บนข้อมูลคนละชุดกัน (Multiple Data) โดยแต่ละหน่วยประมวลผลจะสามารถเข้าถึงข้อมูลในหน่วยความจำแต่ละหน่วยได้พร้อมๆ กัน ซึ่งเป็นระบบคอมพิวเตอร์แบบขนานที่ใช้กับงานเฉพาะด้าน



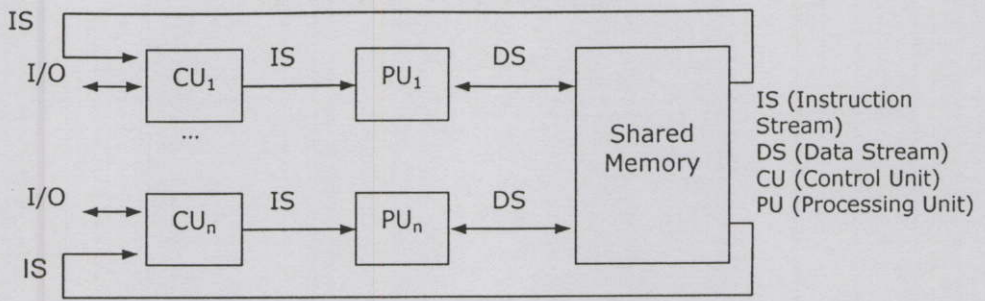
รูปที่ 2.2 สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอสไอเอ็มดี

2.1.1.3 สถาปัตยกรรมแบบเอ็มไอเอสดี (MISD: Multiple Instructions over Single Data) จะเป็นระบบที่ ณ เวลาหนึ่งๆ จะมีหลายๆ คำสั่ง (Multiple Instructions) ถูกประมวลผลในระบบ โดยหลายๆ หน่วยประมวลผล และแต่ละคำสั่งจะทำการประมวลผลบนข้อมูลชุดเดียวกัน (Single Data) ซึ่งในทางปฏิบัติจะไม่ค่อยพบเครื่องคอมพิวเตอร์แบบขนานแบบเอ็มไอเอสดี (MISD) เท่าไรนัก



รูปที่ 2.3 สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอ็มไอเอสดี

2.1.1.4 สถาปัตยกรรมแบบเอ็มไอเอ็มดี (MIMD: Multiple Instructions over Multiple Data) จะเป็นสถาปัตยกรรมที่มีหน่วยประมวลผลหลายๆ หน่วยอยู่บนระบบซึ่งเป็นระบบคอมพิวเตอร์แบบขนานชนิดหนึ่งที่ใช้กับงานทั่วๆ ไปได้ และแต่ละหน่วยประมวลผลสามารถประมวลผลคำสั่งหลายๆ คำสั่ง (Multiple Instructions) บนข้อมูลคนละชุดกัน (Multiple Data) ได้พร้อมๆ กัน



รูปที่ 2.4 สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอ็มไอเอ็มดี

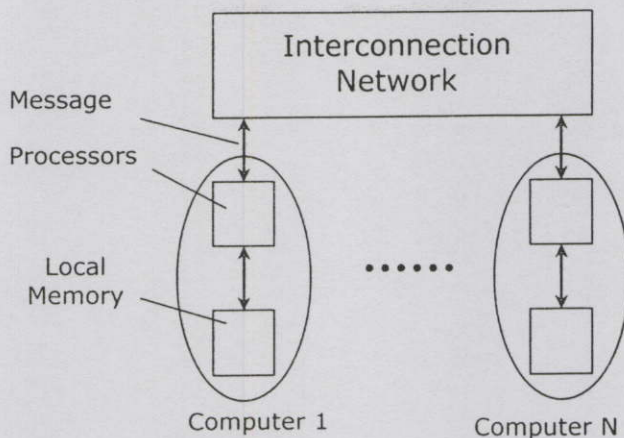
งานวิจัยนี้ให้ความสนใจการจัดรูปทรงของระบบเครือข่าย (Topology) ของเครือข่ายที่มีระบบเครือข่ายการเชื่อมต่อแบบไดนามิก (Dynamic Interconnection Networks) โดยเน้นในส่วน of เครือข่ายการเชื่อมต่อแบบมัลติสแตจ (Multistages Interconnection Networks: MINs) ซึ่งใช้ได้ทั้งสถาปัตยกรรมแบบเอ็มไอเอ็มดี (MIMD) และสถาปัตยกรรมแบบเอสไอเอ็มดี (SIMD)

2.1.2 สถาปัตยกรรมคอมพิวเตอร์แบบขนานที่แบ่งตามองค์ประกอบของพื้นที่การใช้งาน

การหาคำตอบของปัญหามระบบการประมวลผลแบบขนานบางปัญหา สนใจการติดต่อสื่อสารกันระหว่างหน่วยประมวลผล ดังนั้นจึงสามารถแบ่งระบบการประมวลผลแบบขนานดังกล่าวออกเป็น 2 องค์ประกอบ [8, 11, 22] คือ

2.1.2.1 สถาปัตยกรรมการส่งผ่านข้อความ

สถาปัตยกรรมการส่งผ่านข้อความ (Message-Passing Architecture) เป็นการติดต่อสื่อสารระหว่างหน่วยประมวลผลโดยใช้วิธีการส่งผ่านข้อความ หน่วยประมวลผลแต่ละหน่วยประมวลผลจะมีหน่วยความจำภายในสำหรับเก็บข้อมูล (Local Memory) ซึ่งหน่วยประมวลผลของคอมพิวเตอร์เครื่องอื่นๆ จะไม่สามารถเข้าถึงหน่วยความจำนี้ได้ เพราะเหตุนี้เองระบบคอมพิวเตอร์แบบขนานที่ใช้การส่งผ่านข้อความจึงมีชื่อเรียกอีกอย่างหนึ่งว่า ระบบคอมพิวเตอร์หลายเครื่อง (Multicomputers) ในระบบคอมพิวเตอร์หลายเครื่องนั้น หน่วยความจำทั้งหมดจะกระจายอยู่ในคอมพิวเตอร์แต่ละเครื่อง หน่วยความจำที่อยู่ในแต่ละเครื่องจะมีการกำหนดตำแหน่งของหน่วยข้อมูลเป็นของตนเอง (Local Address Space) โดยคอมพิวเตอร์แต่ละเครื่องสามารถที่จะสื่อสารถึงกันได้โดยการส่งข้อความ (Message) ซึ่งข้อความนั้นจะประกอบไปด้วยข้อมูลที่ใช้ในการประมวลผลและข้อมูลอื่นๆ ที่จำเป็นในการติดต่อสื่อสาร เช่น ข้อมูลที่ระบุว่าต้องส่งข้อความนี้ไปให้คอมพิวเตอร์เครื่องใด เป็นต้น เพราะว่าการสื่อสารระหว่างคอมพิวเตอร์ทำได้โดยการส่งข้อความ ระบบคอมพิวเตอร์แบบนี้จึงมีชื่อเรียกว่าระบบคอมพิวเตอร์แบบขนานที่ใช้การส่งผ่านข้อความนั่นเองดังแสดงในรูปที่ 2.5



รูปที่ 2.5 แบบจำลองระบบคอมพิวเตอร์แบบขนานที่ใช้การส่งผ่านข้อความ

ระบบคอมพิวเตอร์แบบขนานที่ใช้การส่งผ่านข้อความมีข้อดีในด้านการขยายระบบซึ่งจะสามารถทำได้ง่ายแค่เพียงนำคอมพิวเตอร์เครื่องใหม่มาเชื่อมเข้ากับระบบเครือข่าย ตัวอย่างเช่น ระบบโอแคม (Occam) และระบบที่ใช้หน่วยประมวลผลของอินเทล (Intel) นอกจากนี้ระบบไม่มีความจำเป็นที่จะต้องจัดการกับการเข้าถึงหน่วยความจำโดยหน่วยประมวลผลหลายๆ หน่วยพร้อมกัน ซึ่งอาจทำให้เวลาในการทำงานของโปรแกรมลดลงได้อย่างมีนัยสำคัญ ข้อดีข้อสุดท้ายของระบบคอมพิวเตอร์แบบขนานที่ใช้การส่งผ่านข้อความคือ การพัฒนาระบบจะทำได้โดยง่าย เพราะสามารถใช้คอมพิวเตอร์ที่มีอยู่แล้วบนเครือข่ายได้ทันทีโดยไม่ต้องมีการดัดแปลงฮาร์ดแวร์ เพราะการพัฒนาทำได้ง่าย จึงทำให้ระบบมีความทันสมัย มีอายุการใช้งานที่นานกว่าระบบคอมพิวเตอร์แบบขนานที่ใช้หน่วยความจำร่วมกัน ข้อเสียของระบบคอมพิวเตอร์แบบขนานที่ใช้การส่งผ่านข้อความคือ ข้อมูลไม่สามารถที่จะถูกใช้ร่วมกันได้ จึงต้องทำให้มีการคัดลอกข้อมูลชุดเดิมอยู่หลายชุด ผ่านเครือข่ายการเชื่อมต่อระหว่างคอมพิวเตอร์ ซึ่งถ้าการใช้งานจำเป็นต้องใช้ข้อมูลเดิมๆ อาจทำให้การทำงานช้าลงได้

2.1.2.2 สถาปัตยกรรมการใช้พื้นที่ร่วมกัน

สถาปัตยกรรมการใช้พื้นที่ร่วมกัน (Shared-Address-Space Architecture)

ได้มีการจัดเตรียมฮาร์ดแวร์สนับสนุนสำหรับการอ่านและการเขียนซึ่งจะสามารถเข้าถึงข้อมูลได้ทุกๆ หน่วยประมวลผลที่มีการใช้พื้นที่ร่วมกัน ระบบคอมพิวเตอร์แบบเอ็มไอเอ็มดีที่มีการใช้พื้นที่ร่วมกัน (MIMD Shared-Address-Space Computers) จะมีการอ้างอิงบ่อยๆ ในระบบคอมพิวเตอร์หลายหน่วยประมวลผล (Multiprocessors) ในระบบคอมพิวเตอร์ที่มีการใช้พื้นที่ร่วมกันจะมีหน่วยความจำที่ใช้ร่วมกัน โดยการเข้าถึงข้อมูลในหน่วยความจำร่วมของแต่ละหน่วยประมวลผลเอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับกรใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้ไปใช้ประโยชน์ด้านการค้า จะต้องผ่านระบบการเชื่อมต่อเครือข่าย (Interconnection Network) ดังนั้นจึงเรียกว่า คอมพิวเตอร์ไม่จำกัดใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้ แบบขนานที่ใช้หน่วยความจำร่วมกัน (Shared-Memory Parallel Computers) ข้อเสียของสถาปัตยกรรมนี้ก็คือ อัตราการส่งข้อมูล (Bandwidth) ของระบบการเชื่อมต่อในเครือข่ายที่จะต้อง

มั่นใจได้ว่ามีประสิทธิภาพดีเพียงพอ เพราะว่าในวงรอบของแต่ละคำสั่ง ทุกๆ หน่วยประมวลผลต้องการจะเข้าถึงข้อมูลในหน่วยความจำที่มีการเปิดให้ใช้ร่วมกันในระบบเครือข่าย ดังนั้นจึงทำให้ระบบเครือข่ายต้องทำงานได้ช้าลง และทำให้ระบบการใช้พื้นที่ร่วมกันมีประสิทธิภาพลดลงได้อีกด้วย

พื้นฐานในระบบหน่วยประมวลผลหลายหน่วย (Multiprocessors) มีวิธีการสร้างอยู่ด้วยกัน 2 รูปแบบดังนี้คือ

1) ระบบยูเอเอ็มเอโมเดล (UMA Model: Uniform Memory Access Model) เป็นระบบที่ทุกๆ หน่วยประมวลผลสามารถเข้าถึงหน่วยความจำร่วมใดก็ได้ในระบบโดยใช้เวลาในการเข้าถึงเท่ากัน

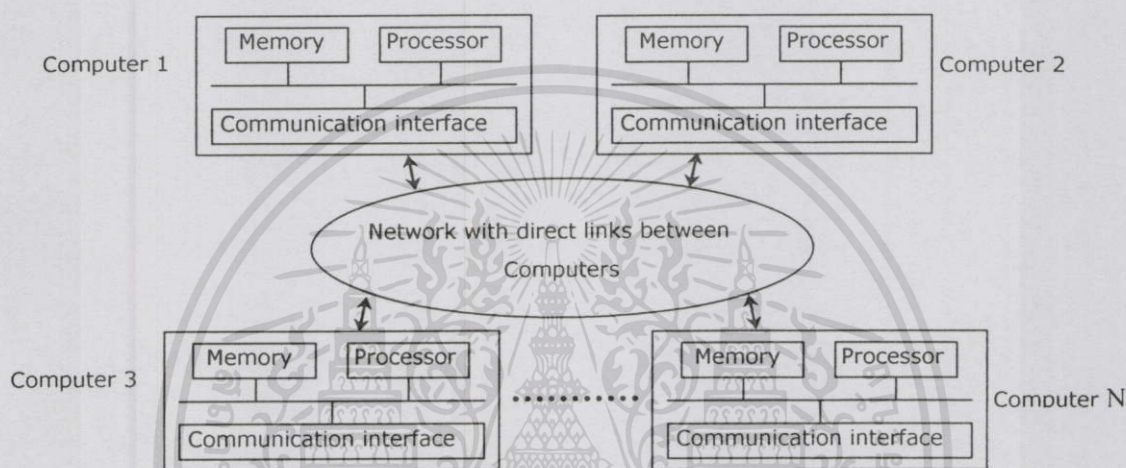
2) ระบบเอ็นยูเอเอ็มเอโมเดล (NUMA Model: Non-Uniform Access Model) เป็นระบบที่ หน่วยประมวลผลแต่ละหน่วยจะสามารถติดต่อหน่วยความจำที่อยู่ใกล้ตัวเองได้เร็วกว่าหน่วยความจำที่ต่อกับหน่วยประมวลผลอื่น แต่ถ้าหน่วยประมวลผลต้องการติดต่อกับหน่วยความจำส่วนอื่น จะต้องทำผ่านทางระบบเครือข่ายการเชื่อมต่อ (Interconnection Network)

2.1.3 เครือข่ายการเชื่อมต่อระหว่างหน่วยประมวลผล

ในระบบเครือข่ายการเชื่อมต่อระหว่างหน่วยประมวลผล (Interconnection Networks) ในซูเปอร์คอมพิวเตอร์แบบขนานเชิงวิจัย จะแตกต่างจากระบบคอมพิวเตอร์เชิงพาณิชย์ทั่วไป เนื่องจากคอมพิวเตอร์ทั่วไปอาจใช้การเชื่อมต่อแบบบัสซึ่งอนุญาตให้มีอุปกรณ์เพียงหนึ่งตัวทำการเข้าถึงบัสได้ในช่วงเวลาใดเวลาหนึ่งเท่านั้น ซึ่งมีข้อจำกัดอยู่ที่การใช้บัสร่วมกัน แต่เนื่องจากในเครื่องคอมพิวเตอร์แบบขนานต้องการความเร็วในการทำงานสูงจึงต้องสามารถอนุญาตให้หลายๆ หน่วยประมวลผลสามารถติดต่อสื่อสารได้พร้อมๆ กัน และทำการใช้หน่วยความจำหลายๆ ส่วนได้พร้อมๆ กัน ซึ่งการใช้พื้นที่ร่วมกันของคอมพิวเตอร์และ การส่งผ่านข้อความของคอมพิวเตอร์จะสามารถทำได้ โดยผ่านเครือข่ายการเชื่อมต่อระหว่างหน่วยประมวลผล (Interconnection Network) เครือข่ายการเชื่อมต่อระหว่างหน่วยประมวลผลสามารถแบ่งออกได้ 2 ประเภทคือ แบบสแตติก (Static) และ แบบไดนามิก (Dynamic) ระบบเครือข่ายแบบสแตติก (Static Network) จะมีการเชื่อมต่อแบบจุดต่อจุดระหว่างหน่วยประมวลผลโดยตรง ระบบเครือข่ายแบบสแตติกจะทำการส่งผ่านข้อความในการติดต่อสื่อสาร ส่วนระบบเครือข่ายแบบไดนามิก (Dynamic Network) จะมีการสร้างรูปแบบการติดต่อสื่อสารที่เปลี่ยนแปลงได้ตามความต้องการของผู้ใช้ โดยใช้การปรับสวิตซ์ในการเชื่อมต่อกันระหว่างลิงก์ ซึ่งจะเป็นการสร้างเส้นทางหลายๆ เส้นทางระหว่างหน่วยประมวลผลและหน่วยความจำทำให้สามารถติดต่อกันได้พร้อมๆ กัน ระบบเครือข่ายแบบไดนามิกจะเป็นการเชื่อมต่อที่ยืดหยุ่น และโดยทั่วไปจะใช้สำหรับการสร้างคอมพิวเตอร์ในการใช้พื้นที่ร่วมกัน

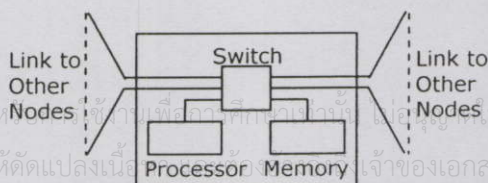
2.2 โครงสร้างเครือข่ายการเชื่อมต่อระหว่างหน่วยประมวลผลแบบสแตติก

สถาปัตยกรรมประเภทการส่งผ่านข้อความจะใช้รูปแบบเครือข่ายการเชื่อมต่อระหว่างหน่วยประมวลผลแบบสแตติก (Static Interconnection Network) ในสถาปัตยกรรมดังกล่าว การเชื่อมต่อระหว่างหน่วยประมวลผล ซึ่งรูปแบบเครือข่ายการเชื่อมต่อที่พบเห็นกันมากที่สุดคือ เครือข่ายแบบสแตติก (Static Network) ในเครือข่ายแบบสแตติก คอมพิวเตอร์หรือโหนดจะมีเส้นเชื่อม (Link) เชื่อมต่อกัน โดยตรงแบบจุดต่อจุด [2, 3, 8, 11, 19, 20, 21]



รูปที่ 2.6 แบบจำลองเครือข่ายแบบสแตติก

ในรูปที่ 2.6 คอมพิวเตอร์จำนวน N เครื่อง ติดต่อกันโดยผ่านเครือข่ายที่เชื่อมต่อกันโดยตรงในคอมพิวเตอร์ แต่ละเครื่องจะมีหน่วยประมวลผล หน่วยความจำ และส่วนต่อเพื่อการสื่อสาร (Communication Interface) ช่วยให้การติดต่อสื่อสารระหว่างคอมพิวเตอร์เป็นไปได้อย่างมีประสิทธิภาพ ในคอมพิวเตอร์บางเครื่องอาจมีการใช้สวิตช์ (Switch) เพื่อให้การส่งข้อความไปยังคอมพิวเตอร์เครื่องอื่นๆ เป็นไปได้อย่างมีประสิทธิภาพมากยิ่งขึ้น แสดงดังรูปที่ 2.7 โดยที่หน่วยประมวลผลไม่ต้องมาทำหน้าที่ในการสื่อสาร สวิตช์จะทำหน้าที่คล้ายๆ กับหน่วยประมวลผลที่ทำหน้าที่ในการสื่อสารโดยเฉพาะนั่นเอง



รูปที่ 2.7 แบบจำลองคอมพิวเตอร์ในเครือข่ายแบบสแตติกที่ใช้สวิตช์

เส้นเชื่อมระหว่างโหนดสองโหนดในเครือข่ายแบบนี้ สามารถที่จะเป็นได้ทั้งแบบทิศทางเดียว (Unidirection) และแบบสองทิศทาง (Bidirection)

เครือข่ายแบบสแต็คที่สำคัญมีอยู่ 4 ชนิดคือ เครือข่ายแบบเส้นตรงและวงแหวน (Linear/Ring Networks) เครือข่ายแบบตาข่าย (Mesh Network) เครือข่ายแบบไฮเปอร์คิวบ์ (Hypercube Network) และเครือข่ายแบบต้นไม้ (Tree Network)

2.2.1 เครือข่ายแบบเส้นตรงและวงแหวน

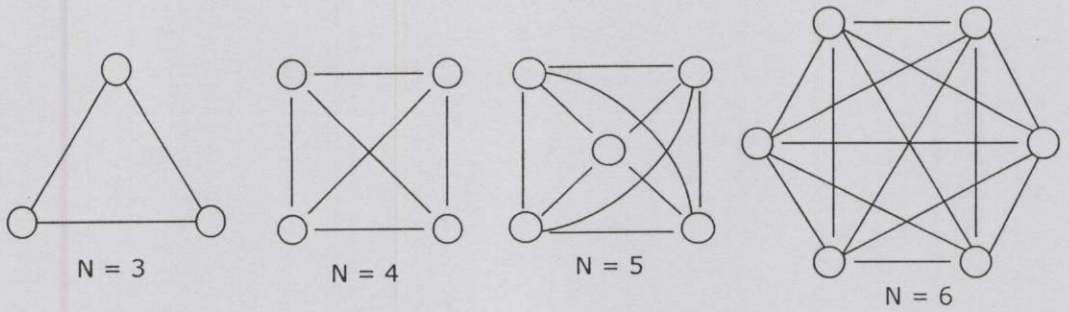
เครือข่ายแบบเส้นตรง (Linear Network) ประกอบด้วยแถวของโหนดที่เชื่อมต่อกันโดยเส้นเชื่อมจะเชื่อมระหว่างโหนดที่อยู่ติดกันเท่านั้น สำหรับเครือข่ายแบบวงแหวน (Ring Network) จะมีเส้นเชื่อมเพิ่มขึ้นอีกหนึ่งเส้นคือ เส้นเชื่อมที่เชื่อมระหว่างโหนดที่อยู่ห่างกันมากที่สุด เครือข่ายทั้งสองแบบแสดงดังรูปที่ 2.8 แต่ละโหนดจะมีเส้นเชื่อมสองเส้น แต่ละเส้นจะเชื่อมต่อไปยังโหนดที่อยู่ข้างๆ เครือข่ายแบบเส้นตรงจะมีเส้นผ่านศูนย์กลาง (Network Diameter) เท่ากับ $N-1$ และมีไบเซกชันวิดท์ (Bisection Width) เท่ากับ 1 ในขณะที่เครือข่ายแบบวงแหวนมีเส้นผ่านศูนย์กลางเท่ากับ $\left\lfloor \frac{N}{2} \right\rfloor$ และมีไบเซกชันวิดท์เท่ากับ 2



รูปที่ 2.8 เครือข่ายแบบ (ก) เส้นตรงและ (ข) วงแหวนจำนวน N โหนด

2.2.2 เครือข่ายการเชื่อมต่อแบบสมบูรณ์

เครือข่ายการเชื่อมต่อแบบสมบูรณ์ (Completely Connected Networks) เป็นเครือข่ายที่แต่ละโหนดมีเส้นเชื่อมถึงโหนดที่เหลือทุกๆ โหนด ดังนั้นหากเครือข่ายมีจำนวนโหนดทั้งหมด N โหนด แต่ละโหนดจะมีการเชื่อมต่อไปยังโหนดอื่นๆ อีกจำนวน $N-1$ โหนด โดยผ่านเส้นเชื่อมทั้งหมด $N-1$ เส้น ดังนั้นเส้นเชื่อมทั้งหมดในเครือข่ายมีจำนวนเท่ากับ $N(N-1)/2$ ทำให้เส้นผ่านศูนย์กลางมีค่าเท่ากับ 1 แต่การเชื่อมต่อแบบนี้มีต้นทุนในการสร้างที่สูงและมีความซับซ้อนในการสร้าง จึงเหมาะสำหรับเครือข่ายขนาดเล็กเท่านั้น ตัวอย่างของเครือข่ายการเชื่อมต่อแบบสมบูรณ์สามารถแสดงได้ดังรูปที่ 2.9

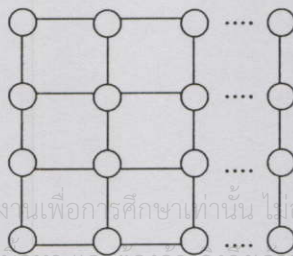


รูปที่ 2.9 เครือข่ายการเชื่อมต่อแบบสมบูรณ์ที่มีขนาด $N = 3 - 6$ โหนด

ในทางปฏิบัติ เมื่อเครือข่ายไม่ได้เป็นแบบสมบูรณ์ เครือข่ายการเชื่อมต่อระหว่างโหนดที่ไม่มีเส้นเชื่อมกันโดยตรงจำเป็นต้องใช้ขั้นตอนวิธีในการหาเส้นทาง (Routing Algorithms) ขั้นตอนวิธีในการหาเส้นทางของเครือข่ายแบบเส้นตรงและเครือข่ายแบบวงแหวนเป็นขั้นตอนวิธีแบบง่ายๆ คือ เพียงแค่เดินทางจากซ้ายไปขวา หรือขวาไปซ้าย ซึ่งขึ้นอยู่กับทิศทางว่าทิศทางใดจะสั้นกว่ากัน แต่สำหรับเครือข่ายแบบอื่นๆ แล้ว ขั้นตอนวิธีในการหาเส้นทางไม่ได้ง่าย แต่จะมีความซับซ้อนตั้งแต่ซับซ้อนน้อยไปจนถึงซับซ้อนมาก

2.2.3 เครือข่ายแบบตาข่าย

เครือข่ายแบบตาข่าย (Mesh Networks) แบบสองมิติคือ แถวลำดับสองมิติ (Two-Dimensional Array) ของโหนดที่แต่ละโหนดมีเส้นเชื่อมไปยังโหนดที่ใกล้ที่สุด 4 โหนด ยกเว้นโหนดรอบๆ แสดงดังรูปที่ 2.10 เส้นผ่านศูนย์กลางของเครือข่ายแบบตาข่ายที่มีขนาด $N = \sqrt{n} \times \sqrt{n}$ คือ $2(\sqrt{n} - 1)$ โดยที่โหนดสองโหนดที่อยู่ห่างกันมากที่สุดคือ โหนดที่อยู่ในมุมทแยงกันและไบเซกชันวิคต์ของเครือข่ายนี้เท่ากับ $\sqrt{n}$ ฟังก์ชันเส้นทางของบนเครือข่ายแบบตาข่าย จะมีการค้นหาไปทางแถวก่อนที่จะไปทางคอลัมน์ สำหรับเครือข่ายที่มีโหนดที่อยู่ตรงขอบของเครือข่ายสามารถที่จะเชื่อมต่อกับโหนดที่อยู่ตรงกันข้ามกันได้ เครือข่ายแบบนี้จะมีชื่อเรียกโดยเฉพาะว่า ทอรัส (Torus) โหนดแต่ละโหนดในทอรัสที่มีขนาด $\sqrt{n} \times \sqrt{n}$ จะมีเส้นเชื่อมทั้งหมดสี่เส้นและมีจำนวนเส้นเชื่อมทั้งหมด $2N$ แสดงดังรูปที่ 2.11 เมื่อ $N = 9$ หรือ $\sqrt{n} \times \sqrt{n} = 3 \times 3$

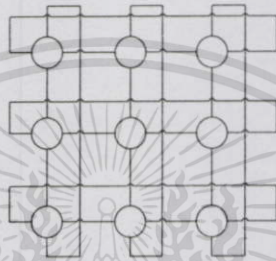


รูปที่ 2.10 เครือข่ายแบบตาข่ายขนาด 2 มิติ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เครือข่ายแบบตาข่ายและทอรัสเป็นเครือข่ายที่ค่อนข้างจะเป็นที่นิยม เนื่องจากมีรูปแบบที่ง่ายและสามารถที่จะขยายได้ง่าย ในเครือข่ายแบบตาข่ายที่มีสามมิติโหนดแต่ละโหนดจะเชื่อมต่ออยู่กับอีกสองโหนดในระนาบของ x , y , และ z เครือข่ายแบบตาข่ายเป็นที่นิยมใช้มากกับปัญหาหลายๆ ปัญหาทางด้านวิศวกรรมศาสตร์และวิทยาศาสตร์ เพราะส่วนใหญ่งานเหล่านี้จะทำงานกับแถวลำดับที่มีสองหรือสามมิติ

ระบบคอมพิวเตอร์ที่ใช้เครือข่ายแบบตาข่ายได้แก่ ระบบอินเทลทัชสโตนเดลต้าคอมพิวเตอร์ (Intel Touchstone Delta Computer) และเจ แมชชีน (The J-Machine) ที่มหาวิทยาลัยเอ็มไอที (MIT : Massachusetts Institute of Technology) [8]

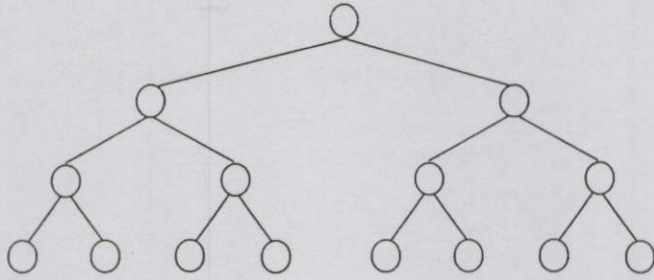


รูปที่ 2.11 เครือข่ายแบบทอรัสขนาด 3×3

2.2.4 เครือข่ายแบบต้นไม้

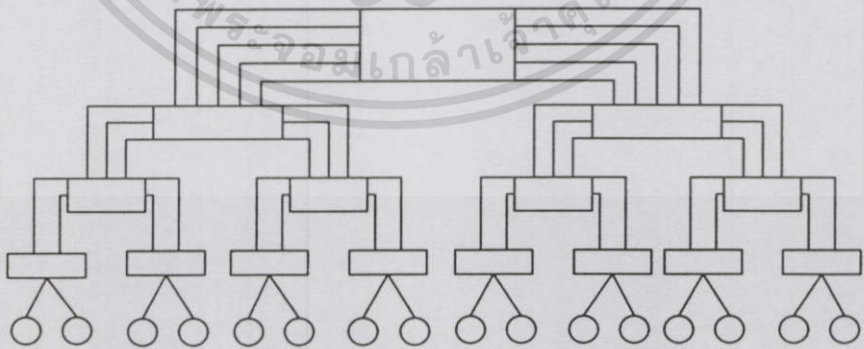
ระบบเครือข่ายแบบต้นไม้ (Tree Network) ที่นิยมสร้างคือ เครือข่ายต้นไม้ทวิภาค (Binary Tree Network) เป็นเครือข่ายที่โหนดแต่ละโหนดจะมีลูกได้อย่างมากที่สุดสองโหนด โหนดที่อยู่บนสุดเรียกว่าโหนดราก (Root Node) แสดงดังรูปที่ 2.12 แต่ละโหนดจะมีเส้นเชื่อมไปยังโหนดลูก (Child Node) ได้อย่างมากที่สุดสองโหนดและมีเส้นเชื่อมไปยังโหนดแม่ (Parent Node) อีกหนึ่งเส้น ยกเว้นโหนดรากที่จะไม่มีเส้นเชื่อมไปยังโหนดแม่ ในระดับแรกภายใต้โหนดรากจะมีโหนดอยู่สองโหนด ระดับต่อมาจะมีโหนดอยู่ทั้งหมดสี่โหนด และในระดับที่ j จะมีโหนดอยู่ทั้งหมด 2^j โหนด และมีจำนวนโหนดทั้งหมด $2^{j+1} - 1$ ในเครือข่าย ซึ่งทำให้เครือข่ายแบบนี้มีเส้นผ่านศูนย์กลางเท่ากับ $2(\log N)$ และมีใบเชิงชันวัดที่เท่ากับ 1 เครือข่ายต้นไม้ที่มีโหนดครบในแต่ละระดับเรียกว่าเครือข่ายต้นไม้แบบสมบูรณ์ (Complete Binary Tree) ความสูงของเครือข่ายต้นไม้คือจำนวนของเส้นเชื่อมจากโหนดรากไปจนถึงโหนดใบไม้ (Leaf Nodes) ที่อยู่ตำแหน่งต่ำที่สุดของเครือข่าย เครือข่ายแบบต้นไม้ไม่จำเป็นจะต้องสมบูรณ์หรือเป็นขนาดของเลขฐานสอง ในเครือข่ายต้นไม้แบบเอ็ม-อารีย์ (m-ary Tree) แต่ละโหนดจะเชื่อมกับโหนดลูกอีกไม่เกิน m โหนด

ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2.12 ระบบเครือข่ายแบบต้นไม้ทวิภาค

ระบบเครือข่ายแบบต้นไม้ไม่เหมาะสำหรับขั้นตอนวิธีการทำงานแบบแบ่งส่วน (Divide-and-Conquer Algorithms) ระบบเครือข่ายชนิดนี้มีข้อจำกัดคือ ถ้าหากการสื่อสารที่มีคำขอที่อยู่ในลักษณะกระจาย (Uniform Request Patterns) ข้อมูลตามคำขอเหล่านี้จะไปรอการส่งต่อไปยังอีกด้านพร้อมๆ กันหลายชุดของข้อมูลอยู่ที่โหนดรากทำให้เกิดสภาพคอขวด (Bottleneck) การแก้ปัญหาดังกล่าวจะทำโดยใช้เครือข่ายต้นไม้แบบอ้วน (Fat Tree Network) เสนอขึ้นโดย Leiserson [11] ในปี 1985 ซึ่งเพิ่มจำนวนของเส้นเชื่อมจะมากขึ้นเรื่อยๆ จากโหนดใบไม้ไปจนถึงโหนดราก แสดงดังรูปที่ 2.13 ต่อมา Leiserson ได้พัฒนาแนวคิดนี้เป็นเครือข่ายต้นไม้แบบอ้วนครอบคลุมจักรวาล (Universal Fat Tree) ซึ่งจำนวนของเส้นเชื่อมระหว่างโหนดจะมีอัตราการเติบโตแบบเลขชี้กำลัง (Exponential Growth) เมื่อเข้าใกล้โหนดราก ทำให้มีการระบายของข้อมูลได้มากขึ้น และลดสภาพของคอขวด ตัวอย่างของระบบที่ใช้เครือข่ายแบบต้นไม้ที่เป็นที่รู้จักกันมากที่สุดคือ คอมพิวเตอร์ซีเอ็มไฟว์ (CM5: Connection Machine Computer) ที่ใช้เครือข่ายต้นไม้แบบอ้วนแบบสี่ทิศทาง (4-Ary Fat Tree) โดย Hwang [3] ในปี 1993

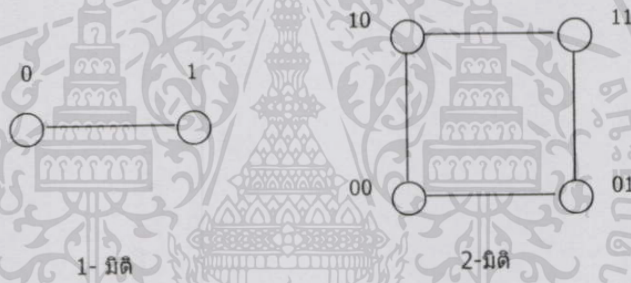


รูปที่ 2.13 เครือข่ายต้นไม้แบบอ้วนขนาด $N = 16$

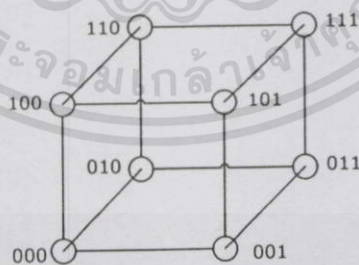
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.2.5 เครือข่ายแบบไฮเปอร์คิว

ไฮเปอร์คิว d มิติ (d-Dimensional Hypercube) ประกอบด้วยไฮเปอร์คิวย่อยที่มี $d-1$ มิติ สองหน่วย โดยมีเส้นเชื่อมจำนวน $N/2$ เส้นเชื่อมระหว่างกัน เมื่อ N คือ จำนวนโหนดทั้งหมดใน d มิติ เครือข่ายนี้มีเส้นผ่านศูนย์กลางเท่ากับ $\log N$ รูปที่ 2.14 แสดงลักษณะของไฮเปอร์คิว 1 - 2 มิติ ซึ่งโหนดแต่ละโหนดจะเชื่อมอยู่กับอีกหนึ่งโหนดในแต่ละทิศของมิติ ยกตัวอย่างเช่นในไฮเปอร์คิวสามมิติแสดงดังรูปที่ 2.15 การเชื่อมต่อทั้งหมดระหว่างโหนดจะทำให้เกิดรูปเครือข่ายในลักษณะของกล่อง แต่ละโหนดจะมีเลขประจำโหนดซึ่งอยู่ในรูปของเลขฐานสองจำนวน d หลัก โดยที่เลขประจำโหนดแต่ละหลักคือ หนึ่งในจำนวนมิติของไฮเปอร์คิว โหนดในไฮเปอร์คิวสามมิติจะมีเลขประจำโหนดอยู่สามหลัก เช่น โหนด 000 จะมีเส้นเชื่อมไปยังโหนด 001, 010, และ 100 ส่วนโหนด 111 จะมีเส้นเชื่อมไปยังโหนด 110, 101, และ 011 ในเครือข่ายแบบนี้โหนดสองโหนดจะมีเส้นเชื่อมกันก็ต่อเมื่อเลขประจำโหนดต่างกันหนึ่งบิตเท่านั้น หลักการนี้ใช้ได้กับไฮเปอร์คิวทุกๆ มิติ



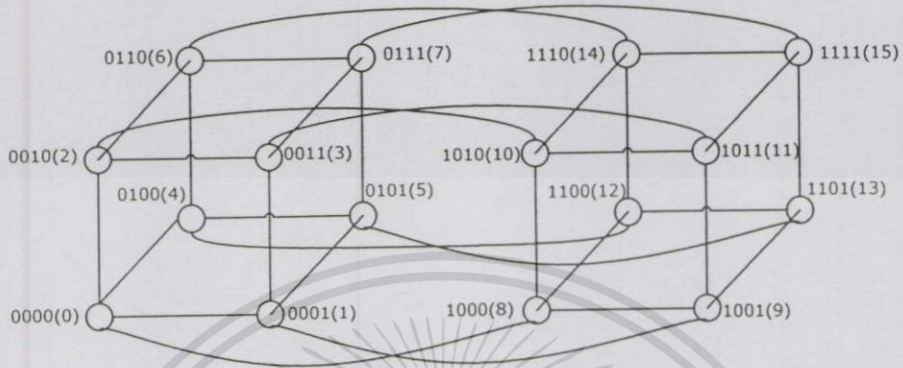
รูปที่ 2.14 เครือข่ายแบบไฮเปอร์คิว 1 - 2 มิติ



รูปที่ 2.15 เครือข่ายแบบไฮเปอร์คิว 3 มิติ ขนาด $N = 8$

คุณสมบัติที่ตัวอย่างหนึ่งของไฮเปอร์คิวคือ มีเส้นผ่านศูนย์กลางอยู่ในรูปของ $\log N$ เมื่อ N เป็นจำนวนโหนดทั้งหมด ซึ่งเป็นคุณสมบัติที่ดีมาก เพราะเส้นผ่านศูนย์กลางมีอัตราการเติบโตที่ไม่เร็วเท่าใดนัก อย่างไรก็ตาม อีกข้อหนึ่งที่น่าสนใจคือ การเชื่อมต่อและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้ซ้ำเมื่อจำนวนของโหนดเพิ่มขึ้น

รูปที่ 2.16 แสดงไฮเปอร์คิวบ 4 มิติ ขนาด $N = 16$ ที่ประกอบด้วยไฮเปอร์คิวบ 3 มิติ 2 หน่วยที่เชื่อมกันอยู่ โดยใช้เส้นเชื่อม 8 เส้น หรือมีไบเซกชันวิคท์เท่ากับ $N/2$ นั่นเอง ตัวอย่างของระบบคอมพิวเตอร์ที่ใช้เครือข่ายไฮเปอร์คิวบคือ คอสมิก คิว (Cosmic Cube) ที่สถาบันแคลเทค (Caltech: California Institute of Technology) [2] ในช่วงทศวรรษที่ 1980



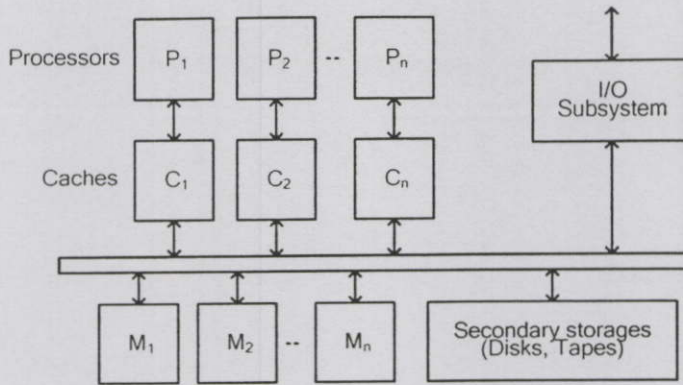
รูปที่ 2.16 เครือข่ายแบบไฮเปอร์คิวบ 4 มิติ ขนาด $N = 16$

2.3 โครงสร้างเครือข่ายการเชื่อมต่อระหว่างหน่วยประมวลผลแบบไดนามิก

ในปัจจุบันจะมีการอธิบายความสำคัญของเครือข่ายการเชื่อมต่อระหว่างหน่วยประมวลผลแบบไดนามิก (Dynamic Interconnection Networks) ซึ่งจะนิยมใช้ในสถาปัตยกรรมของพื้นที่การใช้งาน (Address - Space Organization) ซึ่งเครือข่ายการเชื่อมต่อระหว่างหน่วยประมวลผลแบบไดนามิก [2, 8] สามารถจะกำหนดเส้นเชื่อมต่อได้ใหม่ โดยการปรับการเชื่อมต่อภายในของสวิตช์ในระบบเครือข่ายการเชื่อมต่อแบบนี้สามารถใช้ฟังก์ชันเส้นทาง (Routing Function) แบบขนานแบบต่างๆ ได้หลายแบบ

2.3.1 ระบบเครือข่ายการเชื่อมต่อแบบดิจิทัลบัส

ระบบเครือข่ายการเชื่อมต่อแบบนี้เป็นการเชื่อมต่อที่ไม่ซับซ้อน และราคาไม่แพง เหมาะสำหรับระบบที่มีจำนวนหน่วยประมวลผลไม่มากนักซึ่งแสดงในรูปที่ 2.17 โดยแสดงถึงการเชื่อมต่อระหว่าง โมดูลของหน่วยความจำ (Memory Modules) หน่วยประมวลผล และอุปกรณ์การเชื่อมต่อภายนอก (Peripheral Devices) เพื่อการติดต่อสื่อสารแลกเปลี่ยนข้อมูลระหว่างกัน เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



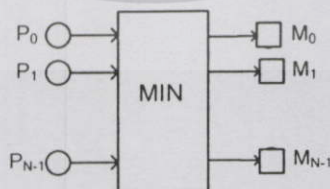
รูปที่ 2.17 ระบบเครือข่ายการเชื่อมต่อแบบบัส

2.3.2 ระบบเครือข่ายการเชื่อมต่อแบบมัลติสแตจ

ระบบเครือข่ายการเชื่อมต่อแบบมัลติสแตจ (Multistages Interconnection Networks: MINs) จะมีการประยุกต์ใช้ที่เหมาะสมทั้งในคอมพิวเตอร์แบบเอ็มไอเอ็มดี (MIMD) และ เอสไอเอ็มดี (SIMD) [8, 15] ระบบเครือข่ายแบบนี้จะมีการใช้ในการเชื่อมต่อระหว่างหน่วยประมวลผลกับหน่วยประมวลผล และ หน่วยประมวลผลกับหน่วยความจำ ดังแสดงในรูปที่ 2.18



(ก) การเชื่อมต่อระหว่างหน่วยประมวลผลกับหน่วยประมวลผล



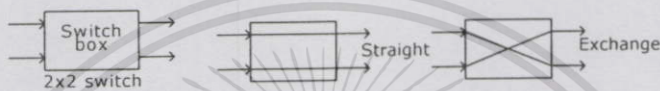
(ข) การเชื่อมต่อระหว่างหน่วยประมวลผลกับหน่วยความจำ

เอกสารนี้เป็นเอกสารที่สงวนไว้เพื่อใช้ประกอบการเรียนการสอนเท่านั้น ไม่ควรนำเอกสารนี้ไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

โครงสร้างโดยทั่วไปของเครือข่ายแบบมัลติสเตจ ประกอบด้วยองค์ประกอบต่างๆ ดังต่อไปนี้ คือ

2.3.2.1 สวิตช์ (Switch)

โมดูลของสวิตช์ $a \times b$ ซึ่งมี a เป็นอินพุต และ b เป็นเอาต์พุต และปกติ $a = b = 2^k$ ที่ $k \geq 0$ เช่น $2 \times 2, 4 \times 4, 8 \times 8$ เป็นต้น การเชื่อมต่อสามารถเชื่อมต่อได้แบบ 1 ต่อ 1 (One-to-One) ที่เป็นไปได้ 2 รูปแบบ คือแบบตรง (Straight) และแบบไขว้ (Exchange) ดังแสดงในรูปที่ 2.19 นอกจากนั้น 1 อินพุต สามารถมีเอาต์พุตได้มากกว่า 1 สำหรับสวิตช์แบบ 2×2 และ แสดงรูปแบบการเชื่อมต่อแบบ 1 ต่อ n (One-to-Many) แสดงดังรูปที่ 2.20 แต่ลักษณะการเชื่อมต่อแบบ n ต่อ 1 (Many-to-One) จะไม่อนุญาตให้สามารถทำได้



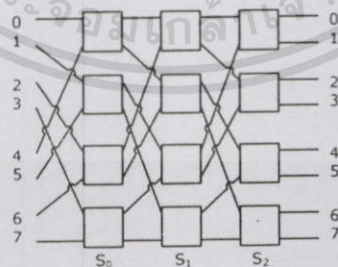
รูปที่ 2.19 แสดงลักษณะการเชื่อมต่อของสวิตช์ขนาด 2×2 แบบ 1-1



รูปที่ 2.20 แสดงลักษณะการเชื่อมต่อของสวิตช์ขนาด 2×2 แบบ 1-n

2.3.2.2 สเตจ (Stages)

สมมติให้ ขนาดของสวิตช์ (Switch Size) = a ($= b$) ถ้าขนาดของอินพุต = N ดังนั้น จำนวนของสเตจ = $\log_2 N$ จำนวนของสวิตช์ต่อสเตจ = N/b ยกตัวอย่างเช่น การใช้สวิตช์ขนาด 2×2 และ $N=8$ ดังนั้น จำนวนของสเตจ = $\log_2 8 = 3$ และจำนวนของสวิตช์ต่อสเตจ = $N/2 = 4$ แสดงในรูปที่ 2.21

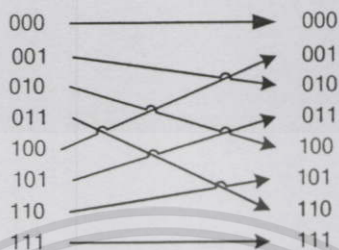


รูปที่ 2.21 แสดงลักษณะการเชื่อมต่อของสวิตช์ขนาด 2×2 ($N=8$)

เอกสารนี้เป็นเอกสารที่สงวนไว้ 2.3.2.3 รูปแบบการเชื่อมต่อระหว่างสเตจ อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าการณีใดๆทั้งสิ้น อีกทั้งห้ามตีตัวอย่างของรูปแบบการเชื่อมต่อระหว่างสเตจ (InterStage Connection: ISC)

จะประกอบด้วย แบบที่นิยมใช้ 2 แบบดังนี้ คือ

1) แบบเพอร์เฟกซ์เฟล (Perfect Shuffle) คือการเชื่อมต่อระหว่างสแตจ สำหรับ $N = 2^k$ หน่วยประมวลผล (PEs) โดยจะใช้ k บิตแทน เลขประจำแต่ละหน่วยประมวลผล คือ $X = (x_{k-1} \dots x_1 x_0)$ ดังนั้น การเชื่อมต่อระหว่างสแตจแบบนี้ เมื่อพิจารณาการเชื่อมต่อระหว่าง X และ Y ใดๆ จะเป็นการเลื่อนซ้าย 1 บิต (Left Shifting) แล้ววนกลับมาใช้แทนโดย $Y = (x_{k-2} \dots x_0 x_{k-1})$ ตัวอย่างสำหรับการเพอร์เฟกซ์เฟลของ 8 หน่วยประมวลผล ($N = M = 8$) แสดงดังรูปที่ 2.22



รูปที่ 2.22 รูปแบบการเชื่อมต่อระหว่างสแตจแบบการเพอร์เฟกซ์เฟล ($N = 8$)

2) แบบอินเวิร์สเพอร์เฟกซ์เฟล (Inverse Perfect Shuffle) คือ การเชื่อมต่อระหว่างสแตจ สำหรับ $N = 2^k$ หน่วยประมวลผล (PEs) โดยจะใช้ k บิตแทน เลขประจำแต่ละหน่วยประมวลผลคือ $X = (x_{k-1} \dots x_1 x_0)$ ดังนั้นการเชื่อมต่อระหว่างสแตจแบบนี้ เมื่อพิจารณาการเชื่อมต่อระหว่าง X และ Y ใดๆจะเป็นการเลื่อนขวา 1 บิต (Right Shifting) แล้ววนกลับมาใช้แทนโดย $Y = (x_0 x_{k-1} \dots x_1)$ ตัวอย่างสำหรับการอินเวิร์สเพอร์เฟกซ์เฟลของ 8 หน่วยประมวลผล ($N = M = 8$) แสดงดังรูปที่ 2.23



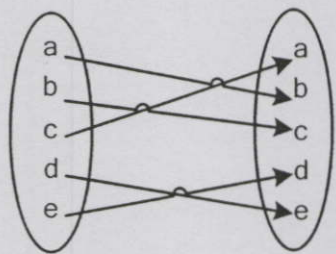
รูปที่ 2.23 รูปแบบการเชื่อมต่อระหว่างสแตจแบบการอินเวิร์สเพอร์เฟกซ์เฟล ($N = 8$)

2.3.2.4 ฟังก์ชันเส้นทางของข้อมูล (Data Routing Functions) ประกอบด้วย

1) การสลับ (Shuffle) และการเปลี่ยน (Exchange) โดยฟังก์ชันการเปลี่ยน

จะสามารถกำหนดได้ ดังนี้ ถ้า $X = (x_{k-1} \dots x_1 x_0)$ จะได้ $E_i(x) = (x_{k-1} \dots x_i \dots x_0)$ ซึ่ง $0 \leq i \leq k-1$ เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า และ บิตที่ x_i เป็นการอินเวิร์สบิตที่ i ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมีเหตุดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2) การเรียงสับเปลี่ยน (Permutation) เป็นการเชื่อมต่อแบบ 1 ต่อ 1 ถ้ามีวัตถุ N วัตถุ ดังนั้นจะสามารถสร้างการเรียงสับเปลี่ยนได้ $N!$ กลุ่มการเรียงสับเปลี่ยน คือชุดของทุกๆ การเรียงสับเปลี่ยนดังรูปที่ 2.24 ซึ่งเป็นตัวอย่างของกราฟที่ $N = 5$ (a, b, c, d, e, f)



รูปที่ 2.24 ตัวอย่างหนึ่งของกราฟที่แสดงการเรียงสับเปลี่ยนที่ $N = 5$ (a, b, c, d, e, f)
โดยฟังก์ชัน $\pi = (a, b, c)(d, e)$

ฟังก์ชันการเรียงสับเปลี่ยน จะสามารถใช้สัญลักษณ์การหมุนโดยรอบ เช่น การเรียงสับเปลี่ยน $\pi = (a, b, c)(d, e)$ จะมี 2 รอบดังนี้ รอบที่ 1 คือ $a \rightarrow b, b \rightarrow c, c \rightarrow a$ รอบที่ 2 คือ $d \rightarrow e, e \rightarrow d$

3) การกระจายข้อมูล (Broadcasting) คือการส่งข้อมูลจากหนึ่งต้นทางไปยังจุดหมายปลายทางหลายๆ เครื่องในเครือข่าย

4) การกระจายแบบมัลติคาสต์ (Multicasting) คือการส่งข้อมูลจากหลายต้นทางไปยังจุดหมายปลายทางหลายๆ เครื่องในเครือข่าย

5) การกระจายข้อมูลเฉพาะบุคคล (Personalized Broadcasting) เป็นการส่งข้อมูลจากหนึ่งต้นทาง โดยสามารถเลือกได้ว่าต้องการส่งข้อมูลให้บุคคลใดในกลุ่มหรือ ระบุเครื่องปลายทาง ซึ่งไม่ใช่ทุกเครื่องในเครือข่าย

2.3.3 เครือข่ายแบบคิว

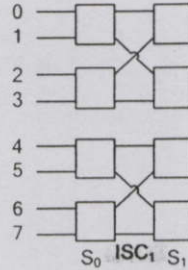
เครือข่ายแบบคิว (Cube Network) [8, 13] เป็นเครือข่ายแบบมัลติสเตจ แบบหนึ่ง ที่มีจำนวนหน่วยประมวลผล $N = 2^k$ และใช้สวิตช์ $a \times b$ ตัวอย่างเช่น ถ้า $N = 8, k = 3$ และใช้สวิตช์ขนาด 2×2 ดังนั้นจำนวนสเตจ = $\log_2 N = 3$ และจำนวนของสวิตช์ต่อสเตจ = $N/2 = 4$ และแต่ละหน่วยประมวลผลจะมีเลขประจำแทนด้วย $k = 3$ บิต ($x = b_2 b_1 b_0$) ซึ่งเครือข่ายแบบคิวจะใช้รูปแบบเอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า การเชื่อมต่อระหว่างสเตจ 3 แบบ (ISC_1, ISC_2, ISC_3) ดังนี้

ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

(การกลับตำแหน่งของบิตจำนวน 2 บิต)

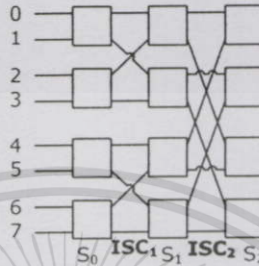
00	→	00
01	→	10
10	→	01
11	→	11

000 → 000
001 → 010
010 → 001
011 → 011
100 → 100
101 → 110
110 → 101
111 → 111



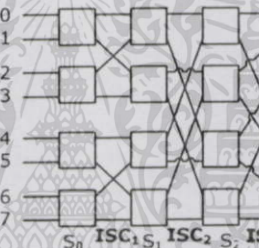
(ก) ISC_1 เป็นการกลับตำแหน่งของบิต (Inverse Bit Address) จำนวน 2 บิตแรก (b_1b_0) เท่านั้น

000 → 000
001 → 100
010 → 010
011 → 110
100 → 001
101 → 101
110 → 011
111 → 111



(ข) ISC_2 เป็นการกลับตำแหน่งของบิต (Inverse Bit Address) จำนวน 3 บิต ($b_2b_1b_0$)

000 → 000
001 → 100
010 → 001
011 → 101
100 → 010
101 → 110
110 → 011
111 → 111



(ค) ISC_3 เป็นการสลับตำแหน่งของบิต จำนวน 3 บิตแบบการอินเวอร์สเฟล็กซ์เพิล

รูปที่ 2.25 ลักษณะการเชื่อมต่อระหว่างสเตจของเครือข่ายแบบคิว (N = 8)

การเชื่อมต่อระหว่างสเตจของเครือข่ายแบบคิวขนาด $N = 8$ แสดงในรูปที่ 2.25 (ค) ซึ่งจะมีรูปแบบการติดต่อสื่อสาร 3 แบบ เช่นเดียวกับเครือข่ายแบบไฮเปอร์คิว 3 มิติ คือ ฟังก์ชันการเรียงสับเปลี่ยนแบบสลับ (Exchange Permutation) ในตำแหน่งบิต i เมื่อ $0 \leq i \leq k-1$ และ $N = 2^k$ คือ

$$E_i(b_{k-1}b_{k-2} \dots b_{i+1}b_i b_{i-1} \dots b_1b_0) = b_{k-1}b_{k-2} \dots b_{i+1}\bar{b}_i b_{i-1} \dots b_1b_0$$

ตัวอย่างเช่น ถ้าพิจารณาเลขประจำโหนดแบบไบนารี 3 บิต ($k = 3$) ในระบบ $N = 8$ ฟังก์ชันการติดต่อสื่อสาร 3 แบบแสดงได้ดังนี้ คือ

$$\text{ถ้า } i = 2 \text{ คือ } E_2(0) = 4, E_2(1) = 5, E_2(2) = 6, E_2(3) = 7, E_2(4) = 0, \dots, E_2(7) = 3$$

เอกสารนี้เป็นเอกสารที่สงวนลิขสิทธิ์โดยมหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี การนำเอกสารนี้ไปใช้โดยไม่ผ่านการอนุญาตถือว่าผิดลิขสิทธิ์และจะถูกลงโทษตามกฎหมาย

ถ้า $i = 1$ คือ $E_1(0) = 2, E_1(1) = 3, E_1(2) = 0, E_1(3) = 1, E_1(4) = 6, E_1(5) = 7, E_1(6) = 4, E_1(7) = 5$

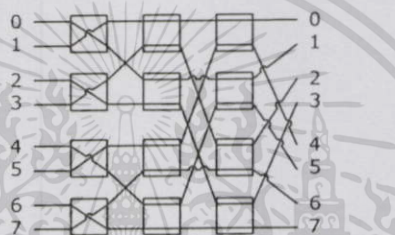
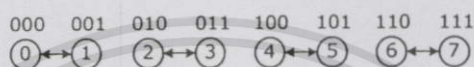
$$\text{ถ้า } i = 0 \text{ คือ } E_0(0) = 1, E_0(1) = 0, E_0(2) = 3, E_0(3) = 2, E_0(4) = 5, E_0(5) = 4, E_0(6) = 7, E_0(7) = 6$$

เครือข่ายแบบคิวจะมีการปรับสวิตช์ตามฟังก์ชันเส้นทางการเชื่อมต่อระหว่างหน่วยประมวลผล ดังต่อไปนี้

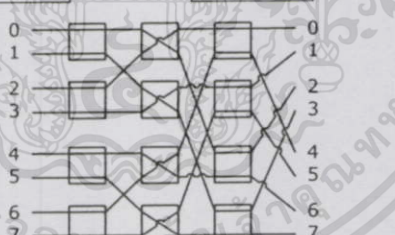
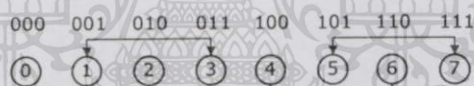
สำหรับฟังก์ชัน E_0 เป็นการกำหนดให้สแตจ 0 (S_0) เป็นการสับสวิตช์แบบไขว้ (Exchange) และกำหนดค่าของสแตจ 1 และ 2 (S_1 & S_2) ให้เป็นการสับสวิตช์แบบตรง (Straight)

สำหรับฟังก์ชัน E_1 เป็นการกำหนดให้สแตจ 1 (S_1) เป็นการสับสวิตช์แบบไขว้ และกำหนดค่าของสแตจ 0 และ 2 (S_0 & S_2) ให้เป็นการสับสวิตช์แบบตรง

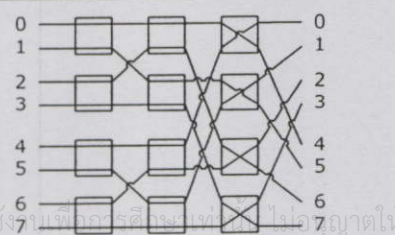
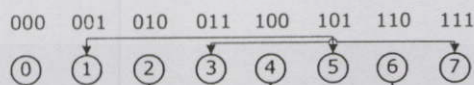
สำหรับฟังก์ชัน E_2 เป็นการกำหนดให้สแตจ 2 (S_2) เป็นการสับสวิตช์แบบไขว้ และกำหนดค่าของสแตจ 0 และ 1 (S_0 & S_1) ให้เป็นการสับสวิตช์แบบตรง ดังจะแสดงให้เห็นดังรูป 2.26



(ก) ฟังก์ชันเส้นทาง E_0



(ข) ฟังก์ชันเส้นทาง E_1



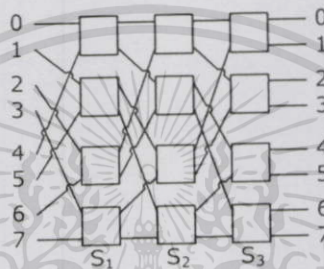
(ค) ฟังก์ชันเส้นทาง E_2

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเฉพาะกิจของหน่วยงานที่เกี่ยวข้องเท่านั้น ไม่ควรเผยแพร่หรือใช้เพื่อวัตถุประสงค์อื่นใดโดยไม่ได้รับอนุญาตจากหน่วยงานต้นสังกัด

รูปที่ 2.26 แสดงการเชื่อมต่อบนเครือข่ายแบบคิวสำหรับฟังก์ชันเส้นทาง E_i ($N = 8$)

2.3.4 เครือข่ายแบบโอเมก้า

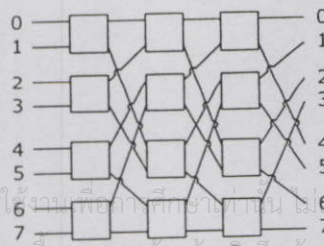
เครือข่ายแบบโอเมก้า (Omega Network) [8, 13] เป็นเครือข่ายแบบมัลติสเตจแบบหนึ่ง ที่มีจำนวนหน่วยประมวลผล $N = 2^k$ และใช้สวิตช์ $a \times b$ ตัวอย่างเช่น ถ้า $N = 8$, $k = 3$ และใช้สวิตช์ขนาด 2×2 ดังนั้น จำนวนสเตจ = $\log_2 N = 3$ และจำนวนของสวิตช์ต่อสเตจ = $N/2 = 4$ และแต่ละหน่วยประมวลผลจะมีเลขประจำแทนด้วย $k = 3$ บิต ($x = b_2b_1b_0$) ซึ่งเครือข่ายแบบโอเมก้าจะใช้รูปแบบการเชื่อมต่อระหว่างสเตจ (ISC) แบบเฟอร์เฟลซบเฟิล ทุกๆ สเตจซึ่งจะแสดงแต่ละสเตจของเครือข่ายแบบโอเมก้าดังแสดงในรูปที่ 2.27



รูปที่ 2.27 ลักษณะการเชื่อมต่อระหว่างสเตจของเครือข่ายแบบโอเมก้า ($N=8$)

2.3.5 เครือข่ายแบบฟลิป

เครือข่ายแบบฟลิป (Flip Network) [8, 13] เป็นเครือข่ายแบบมัลติสเตจแบบหนึ่งที่มีจำนวนหน่วยประมวลผล $N = 2^k$ และใช้สวิตช์ $a \times b$ ตัวอย่างเช่น ถ้า $N = 8$, $k = 3$ และใช้สวิตช์ 2×2 และจำนวนสเตจ = $\log_2 N = 3$ จำนวนของสวิตช์ต่อสเตจ = $N/2 = 4$ และแต่ละหน่วยประมวลผลจะมีเลขประจำแทนด้วย $k = 3$ บิต ($x = b_2b_1b_0$) ซึ่งเครือข่ายแบบฟลิปจะใช้รูปแบบการเชื่อมต่อระหว่างสเตจ แบบอินเวิร์สเฟอร์เฟลซบเฟิล ทุกๆ สเตจซึ่งจะแสดงแต่ละสเตจของเครือข่ายแบบฟลิปดังแสดงในรูปที่ 2.28



รูปที่ 2.28 ลักษณะการเชื่อมต่อระหว่างสเตจของเครือข่ายแบบฟลิป ($N=8$)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา S6 และ S7 อย่าง S2 ถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.4 ต้นไม้แบบทอดข้ามที่แตกต่าง

F. Bao และคณะ [1] ให้นิยามของการสร้างต้นไม้แบบทอดข้ามที่แตกต่าง ในปี ค.ศ. 1998 ดังต่อไปนี้

นิยาม ต้นไม้แบบทอดข้ามที่แตกต่าง (Independent Spanning Tree)

กราฟ ต้นไม้แบบทอดข้ามที่แตกต่าง 2 ต้นใด ๆ จาก k ต้นที่สร้างจากกราฟ $G = (V, E)$ จะเรียกว่า “Independent” ได้ก็ต่อเมื่อต้นไม้ทั้ง 2 ต้นดังกล่าว มีโหนดรากเป็นโหนดเดียวกันคือ r แต่เส้นทางจากโหนด $v \in V$ ใดๆ ไปยัง r ของต้นไม้แบบทอดข้ามทั้ง 2 ต้นจะแตกต่างกันทุกลิงค์ และทุกเส้นทาง

กล่าวคือ เส้นเชื่อมหรือลิงค์ $e = (i, j)$ ใดๆ $\in E$ ที่อยู่ในต้นไม้แบบทอดข้ามที่แตกต่าง ทั้ง k ต้นจะแตกต่างกันทั้งหมดด้วย

2.5 จตุรัสลาติน

จตุรัสลาติน (Latin Square) ที่รู้จักกันโดยกรณีทั่วไป คือ จตุรัสที่มีขนาด $n \times n$ โดยที่จะมี n สัญลักษณ์ที่เกิดขึ้นเรียงแตกต่างกันในทุกแถว และทุกหลัก ดังนั้นแต่ละ n ที่จะเกิดขึ้นในจตุรัสจะไม่ซ้ำกันในแต่ละแถวและแต่ละคอลัมน์ โดยค่าในแถวแรก และคอลัมน์แรกคือ $(1, 2, 3, \dots, n)$ ดังนั้นค่าในจตุรัสลาติน ขนาด $n \times n$ คือ

$$\begin{bmatrix} 1 & 2 & 3 & \dots & n \\ 2 & 3 & 4 & \dots & 1 \\ 3 & 4 & 5 & \dots & 2 \\ \vdots & & & & \\ n & 1 & 2 & \dots & n-1 \end{bmatrix}$$

ตัวอย่างเช่น ถ้า $n = 3$ จะได้จตุรัสลาติน ขนาด 3×3 มีค่าดังนี้คือ

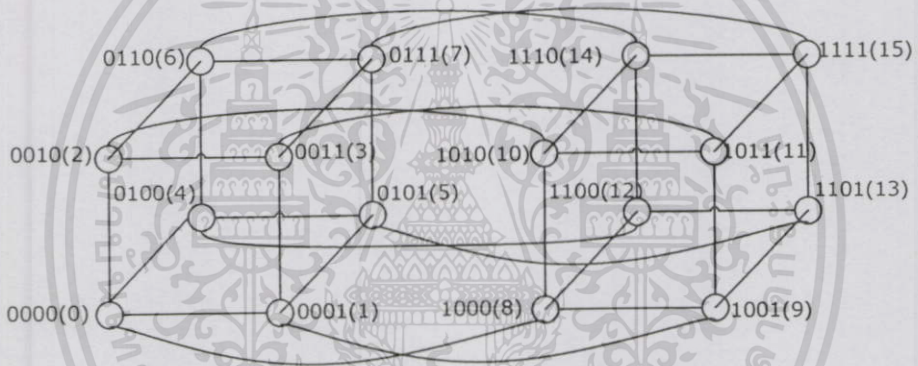
$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix}$$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ค่าในจตุรัสลาติน ขนาด 3×3 ดังกล่าวจะแสดงโดยใช้สัญลักษณ์ 1, 2 และ 3 ดังนั้น จตุรัสนี้ใช้สัญลักษณ์ทั้งหมด $n = 3$ ตัว ทำให้จตุรัสนี้มีขนาด 3×3 และแต่ละแถวและคอลัมน์จะมี สัญลักษณ์ 1, 2 และ 3 เรียงแตกต่างกันไปด้วย

2.6 การสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน บนเครือข่ายแบบไฮเปอร์คิว

ในปี ค.ศ. 2007 J.-S. Yang และคณะ [18] ได้นำเสนอการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนานบนเครือข่ายแบบไฮเปอร์คิว (Parallel Construction of Independent Spanning Trees on Hypercube) ซึ่งเครือข่ายแบบไฮเปอร์คิว k มิติ (Q_k) คือกราฟที่ประกอบด้วย $N = 2^k$ โหนด ซึ่งมีเลขประจำโหนดเป็นไบนารีสตริงที่มีความยาวเท่ากับ k และ 2 โหนดที่เชื่อมต่อกันจะประกอบด้วยเลขประจำโหนดที่แตกต่างกันเพียง 1 ตำแหน่งเท่านั้น



รูปที่ 2.29 แสดงการเชื่อมต่อของเครือข่ายแบบไฮเปอร์คิว 4 มิติ

2.6.1 ความรู้เบื้องต้นสำหรับการสร้างต้นไม้แบบทอดข้ามที่แตกต่าง

ในหัวข้อนี้จะนำเสนอสัญลักษณ์และคุณสมบัติ สำหรับสร้างต้นไม้แบบทอดข้ามที่แตกต่าง (ISTs : Independent Spanning Trees)

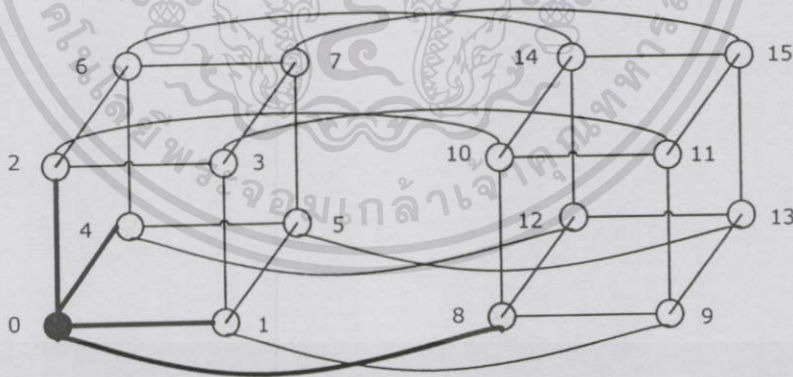
กำหนดให้	$N_G(x)$	แทนโหนดที่อยู่ติดกัน หรือโหนดประชิด (Adjacent Node) ของโหนด x ในกราฟ G
	$parent(T, x)$	แทนพ่อแม่ของโหนด x ซึ่งเป็นโหนดประชิดของ x ($\neq r$) ในเส้นทาง $T[r, x]$ สำหรับต้นไม้ T ที่มี r เป็นโหนดราก
	T_i	แทนต้นไม้แบบทอดข้ามที่แตกต่าง เมื่อ $0 \leq i \leq k-1$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้ไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ไฮเปอร์คิวบ์ k มิติ (Q_k) มีโหนดที่สมมาตร เราจะสามารถสร้างต้นไม้แบบทอดข้ามที่แตกต่างที่มีโหนด r เป็นรากได้ k ต้น ถ้าเราให้โหนด 0 เป็นรากของต้นไม้แบบทอดข้ามที่แตกต่างใน Q_k สมมุติให้ $\mathcal{T}$ เป็นชุดของต้นไม้แบบทอดข้ามที่แตกต่าง (รากที่ 0) ของ Q_k ($\mathcal{T} = \{T_i; 0 \leq i \leq k-1\}$) จะพบว่าโหนดประชิดของ 0 คือ $N_{Q_k} = 2^0, 2^1, \dots, 2^{k-1}$ จากการเชื่อมต่อของ Q_k ดังแสดงดังรูปที่ 2.30 (ก) เมื่อ $k = 4$ ($N_{Q_k} = 1, 2, 4, 8$) และการกำหนด $\mathcal{T} \in \mathcal{T}$ ทำให้โหนด 0 มีลูกได้ 1 โหนดเท่านั้นในทุกๆ ต้นไม้แบบทอดข้าม T ดังแสดงดังรูปที่ 2.30 (ข) ดังนั้นสำหรับโหนดรากที่ 0 โหนดลูกของโหนด 0 คือ 2^i ($0 \leq i \leq k-1$) ตัวอย่างเช่น ต้นไม้แบบทอดข้ามที่แตกต่าง 4 ต้น ของไฮเปอร์คิวบ์ 4 มิติ (Q_4) แสดงในรูปที่ 2.30 (ข) ซึ่งจะใช้แทนโดย T_0, T_1, T_2 และ T_3 จากซ้ายไปขวาตามลำดับ โดยโหนดลูกของโหนด 0 คือ โหนด 1, 2, 4, 8

กราฟแฮมิลโทเนียน (Hamiltonian Graph) เป็นปัญหาการหาเส้นทางแบบหนึ่งโดยเฉพาะอย่างยิ่ง จะมีการสร้างเมตริกซ์เรียกว่าจัตุรัสลาตินของวงจรแฮมิลโทเนียน (Hamiltonian Circuit Latin Square: HCLS) [18] เพื่อค้นหาชุดของโหนดที่เชื่อมต่อกันบนไฮเปอร์คิวบ์ ซึ่งจะนำมาประยุกต์ใช้สำหรับการส่งข้อมูลจากโหนดต้นทางไปยังโหนดปลายทางบน Q_k ซึ่งจะเรียกว่าจัตุรัสลาตินของระยะทางแฮมมิง (Hamming Distance Latin Square: HDLS)

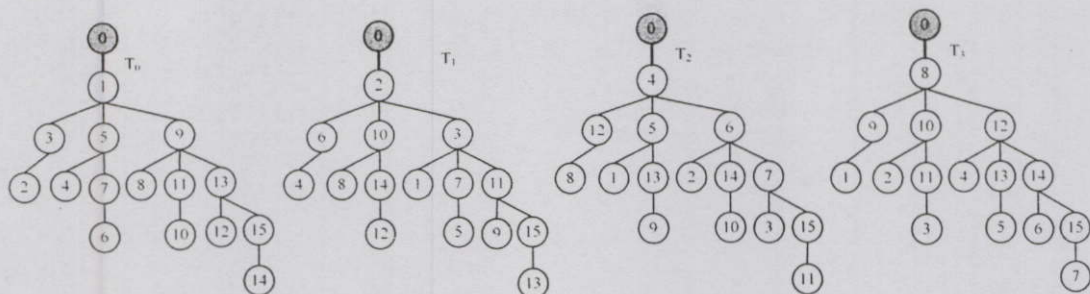
ลาตินสแควร์ก็คือเมตริกซ์ที่มีขนาดเท่ากับ n^2 โดยเลือกชุดของ H จาก n ที่แตกต่างกันซึ่งจะไม่มีส่วนที่ทับซ้อนกันในแต่ละแถวหรือคอลัมน์ของเมตริกซ์ บนไฮเปอร์คิวบ์ Q_k และ โหนด x ($x \neq r$) $\in Q_k$ ที่เลขประจำโหนดแบบไบนารีสตริง $x = x_1x_2\dots x_n$



(ก)

รูปที่ 2.30 แสดงชุดของต้นไม้แบบทอดข้ามที่แตกต่าง 4 แบบ (T_0, T_1, T_2, T_3) บนเครือข่ายแบบไฮเปอร์คิวบ์ 4 มิติ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



(ข)

รูปที่ 2.30 (ต่อ)

เมตริกซ์จัตุรัสลาตินของระยะทางแฮมมิง จะสร้างจากการพิจารณาค่า x ที่เป็นลาตินสแควร์ โดยค่าในแถวแรกของเมตริกซ์จะเป็นตำแหน่ง i ของบิต x_i ที่มีค่าเป็น 1 คือจะเลือกจากชุดของ $H_k = \{i: 0 \leq i \leq k-1 \text{ และ } x_i = 1\}$ เนื่องจากระยะทางแฮมมิง ของ x ใดๆ ที่ไม่เท่ากับ 0 ไปยังโหนดรากที่ 0 ใน Q_k เป็นการกำหนดค่าของ $|H_k|$ ซึ่งจะสามารถสร้างเมตริกส์ได้ 2 เมตริกส์ ดังนี้

$$I_k(x) = \begin{bmatrix} i_0 & i_1 & \dots & i_{t-2} & i_{t-1} \\ i_1 & i_2 & \dots & i_{t-1} & i_0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ i_{t-2} & i_{t-1} & \dots & i_{t-4} & i_{t-3} \\ i_{t-1} & i_0 & \dots & i_{t-3} & i_{t-2} \end{bmatrix} \text{ และ } D_k(x) = \begin{bmatrix} i_{t-1} & i_{t-2} & \dots & i_1 & i_0 \\ i_{t-2} & i_{t-3} & \dots & i_0 & i_{t-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ i_1 & i_0 & \dots & i_3 & i_2 \\ i_0 & i_{t-1} & \dots & i_2 & i_1 \end{bmatrix}$$

เมตริกซ์ $I_k(x)$ เรียกว่าการหมุนรอบที่เพิ่มขึ้นของจัตุรัสลาตินของระยะทางแฮมมิง และค่า i_{j+1} ใน $I_k(x)$ จะเรียกว่า ลูกหลาน (Successor) ของ i_j จะแสดงโดยใช้สัญลักษณ์ $\text{succ}(i_j)$ และค่า i_{j-1} ใน $I_k(x)$ จะเรียกว่า บรรพบุรุษ (Predecessor) ของ i_j แสดงโดยใช้สัญลักษณ์ $\text{pred}(i_j)$ โดยชี้ที่ j และ $j+1$ ($j-1$ ตามลำดับ) การเรียงค่า i_j ที่แตกต่างสามารถคำนวณได้ด้วย $\text{mod } t$ และในทำนองเดียวกันเมตริกซ์ $D_k(x)$ เรียกว่า การหมุนรอบที่ลดลงของจัตุรัสลาตินของระยะทางแฮมมิง

ในการพิสูจน์ว่าเส้นทางที่ได้ใน T_i ทุกค่า $0 \leq i \leq k-1$ จะไม่ซ้ำกันจากเส้นเชื่อมที่ต่างกันจะพิจารณาจากการนำค่าในเมตริกซ์จัตุรัสลาติน ไปประยุกต์ใช้ในการสร้าง T_i จะได้เส้นทางที่ไม่ซ้ำเนื่องจากถ้าให้ $A = a_1, a_2, \dots, a_i$ เป็นลำดับของจำนวนจริงที่ไม่เป็นลบ จะสามารถหาเลข

(ฐาน 10) ประจำโหนดในเส้นทางอีกจำนวน $k - 1$ โหนดจากผลรวมสองยกกำลัง (Prefix-Powersum) ได้โดยใช้สูตรดังนี้ $\sum_{i=1}^L 2^{a_i}$ ที่ $1 \leq L \leq t$ เพราะแต่ละแถวในเมตริกซ์ $I_k(x)$ เป็นลำดับไม่ซ้ำกันใดๆทั้งสิ้น อีกทั้งห้ามมีให้ตัดแบบลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้ จำนวนจริงที่ไม่เป็นลบ ดังแสดงในตัวอย่างที่ 2.1 – 2.3

ตัวอย่างที่ 2.1 พิจารณาที่โหนด $x = 1101$ (13) ใน Q , จะสามารถสร้างเมตริกซ์จัตุรัสลาตินได้ 2 เมตริกซ์ดังต่อไปนี้

$$I_k(x) = \begin{bmatrix} 0 & 2 & 3 \\ 2 & 3 & 0 \\ 3 & 0 & 2 \end{bmatrix} \text{ และ } D_k(x) = \begin{bmatrix} 3 & 2 & 0 \\ 2 & 0 & 3 \\ 0 & 3 & 2 \end{bmatrix}$$

เนื่องจาก $X = x_3x_2x_1x_0 = 1101$ มีตำแหน่งที่ $x_i = 1$ คือ $[0 \ 2 \ 3]$ ซึ่งเป็นค่าในแถวแรกของเมตริกซ์ การนำค่าในเมตริกซ์จัตุรัสลาติน ไปประยุกต์ใช้ในการสร้าง T_i จะได้เส้นทางที่ไม่ซ้ำจากโหนด $13 \rightarrow 0$ โดยการหาโหนดในเส้นทางอีก $k-1$ โหนดซึ่งคำนวณจากผลรวมสองยกกำลังของแต่ละแถวใน $I_k(x)$ สามารถคำนวณได้ 3 เส้นทางที่ไม่ซ้ำกัน คือ

เมื่อพิจารณา $i_j = 0$ จะมีโหนดในเส้นทางอีก 2 โหนด ซึ่งคำนวณได้ดังนี้ คือ

$2^2 + 2^3 = 12$ (จากแถวที่ 1), $2^2 = 4$ (จากแถวที่ 3) ดังนั้น
เส้นทางที่ 1 คือ $13 \rightarrow 12 \rightarrow 4 \rightarrow 0$

เมื่อพิจารณา $i_j = 2$ จะมีโหนดในเส้นทางอีก 2 โหนด ซึ่งคำนวณได้ดังนี้ คือ

$2^3 = 8$ (จากแถวที่ 1), $2^3 + 2^0 = 9$ (จากแถวที่ 2) ดังนั้น
เส้นทางที่ 2 คือ $13 \rightarrow 9 \rightarrow 8 \rightarrow 0$

เมื่อพิจารณา $i_j = 3$ จะมีโหนดในเส้นทางอีก 2 โหนด ซึ่งคำนวณได้ดังนี้ คือ

$2^0 = 1$ (จากแถวที่ 2), $2^0 + 2^2 = 5$ (จากแถวที่ 3) ดังนั้น
เส้นทางที่ 3 คือ $13 \rightarrow 5 \rightarrow 1 \rightarrow 0$

ในทำนองเดียวกัน ถ้าพิจารณาที่โหนด $y = 1111$ (15) ใน Q , จะสามารถสร้างเมตริกซ์ได้ดังต่อไปนี้

$$I_k(y) = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 0 \\ 2 & 3 & 0 & 1 \\ 3 & 0 & 1 & 2 \end{bmatrix}$$

ในกรณีนี้ เนื่องจาก $Y = y_3y_2y_1y_0 = 1111$ มีตำแหน่งที่ $y_i = 1$ คือ $[0 \ 1 \ 2 \ 3]$ ซึ่งเป็นค่าในแถวแรกของเมตริกซ์ และการนำค่าในเมตริกซ์จัตุรัสลาติน ไปประยุกต์ใช้ในการสร้าง T_i จะได้เส้นทางที่ไม่ซ้ำจากโหนด $15 \rightarrow 0$ โดยการคำนวณจากผลรวมสองยกกำลังของแต่ละแถวใน $I_k(x)$ สามารถคำนวณได้ 4 เส้นทางที่ไม่ซ้ำกัน คือ

- เมื่อพิจารณา $i_j = 0$ จะมีโหนดในเส้นทางอีก 3 โหนด ซึ่งคำนวณได้ดังนี้ คือ
 $2^1 + 2^2 + 2^3 = 14$ (จากแถวที่ 1), $2^1 = 2$ (จากแถวที่ 3) ,
 $2^1 + 2^2 = 6$ (จากแถวที่ 4) ดังนั้น เส้นทางที่ 1 คือ
 $15 \rightarrow 14 \rightarrow 6 \rightarrow 2 \rightarrow 0$
- เมื่อพิจารณา $i_j = 1$ จะมีโหนดในเส้นทางอีก 3 โหนด ซึ่งคำนวณได้ดังนี้ คือ
 $2^2 + 2^3 = 12$ (จากแถวที่ 1), $2^2 + 2^3 + 2^0 = 13$ (จากแถวที่ 2),
 $2^2 = 4$ (จากแถวที่ 4) ดังนั้น เส้นทางที่ 2 คือ
 $15 \rightarrow 15 \rightarrow 13 \rightarrow 12 \rightarrow 4 \rightarrow 0$
- เมื่อพิจารณา $i_j = 2$ จะมีโหนดในเส้นทางอีก 3 โหนด ซึ่งคำนวณได้ดังนี้ คือ
 $2^3 = 8$ (จากแถวที่ 1), $2^3 + 2^0 = 9$ (จากแถวที่ 2),
 $2^3 + 2^0 + 2^1 = 11$ (จากแถวที่ 3) ดังนั้น เส้นทางที่ 3 คือ
 $15 \rightarrow 15 \rightarrow 11 \rightarrow 9 \rightarrow 8 \rightarrow 0$
- เมื่อพิจารณา $i_j = 3$ จะมีโหนดในเส้นทางอีก 3 โหนด ซึ่งคำนวณได้ดังนี้ คือ
 $2^0 = 1$ (จากแถวที่ 2), $2^0 + 2^1 = 3$ (จากแถวที่ 3),
 $2^0 + 2^1 + 2^2 = 7$ (จากแถวที่ 4) ดังนั้น เส้นทางที่ 4 คือ
 $15 \rightarrow 15 \rightarrow 7 \rightarrow 3 \rightarrow 1 \rightarrow 0$

2.6.2 ขั้นตอนวิธีการสร้างต้นไม้แบบทอดข้ามที่แตกต่าง บนเครือข่ายแบบไฮเปอร์คิว k มิติ

เส้นทางของไฮเปอร์คิว k มิติ (Q_k) จะใช้สัญลักษณ์ $x \xrightarrow{+2^i} y$ ถ้า $x + 2^i = y$ และ $x \xrightarrow{-2^i} y$ ถ้า $x - 2^i = y$ เมื่อโหนด x และ y เป็น โหนดประชิดกัน

Algorithm GEN-PARENTS

begin

for all every vertex $x(\neq 0) \in Q_k$ with binary string $x = x_{k-1}x_{k-2}...x_0$ **pardo**

for $i=0$ to $k-1$ **do**

if $(x_i=1)$ $parent(T_i, x) = x - 2^{succ(i)}$; // where $pred(i)$ is referred to $I_k(x)$

if $(x_i=0)$ $parent(T_i, x) = x + 2^i$;

end for i

end for all

End GEN-PARENTS

จากขั้นตอนวิธีการดังกล่าวจะสามารถสร้างต้นไม้แบบทอดข้ามที่แตกต่างจำนวน k ต้น บน Q_k ด้วยความซับซ้อนทางด้านเวลาเท่ากับ $O(kN)$ โดยที่ $N = 2^k$ ด้วยวิธีการแบบลำดับแต่ถ้าสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน บน Q_k จะสามารถทำได้ด้วยความซับซ้อนทางด้านเวลาเท่ากับ $O(k)$

จากขั้นตอนวิธีการจะสามารถสร้างชุด $\mathcal{T} = \{T_i : 0 \leq i \leq k-1\}$ โดยที่รากของต้นไม้แบบทอดข้ามที่แตกต่าง เริ่มจาก โหนด 0

ตัวอย่างที่ 2.2 ขั้นตอนวิธีสามารถสร้างต้นไม้แบบทอดข้ามที่แตกต่างทุกๆ ต้นไม้ ดังต่อไปนี้

พิจารณาที่การสร้าง T_0 จาก Q_4 ได้ดังนี้

โดยให้ X ใดๆ ที่ไม่เท่ากับ 0 ใน Q_4 เป็นไบนารีสตริงค์ $X = x_3x_2x_1x_0$

กรณีที่ 1 ถ้า $x_0 = 0$ จะได้ $X \in \{2, 4, 6, 8, 10, 12, 14\}$ ดังนั้นจะได้ $parent(T_0, x) = x+2^0 = x+1$

กรณีที่ 2 ถ้า $x_0 = 1$ จะแบ่งได้หลายกรณีดังนี้

$X \in \{1\}$ ดังนั้น $succ(0) = 0$ จะได้ $parent(T_0, x) = x-2^0 = 1-1 = 0$

$X \in \{3, 7, 11, 15\}$ ดังนั้น $succ(0) = 2$ จะได้ $parent(T_0, x) = x-2^1 = x-2$

$X \in \{13\}$ ดังนั้น $succ(0) = 1$ จะได้ $parent(T_0, x) = x-2^2 = 13-4 = 9$

$X \in \{9\}$ ดังนั้น $succ(0) = 3$ จะได้ $parent(T_0, x) = x-2^3 = 9-8 = 1$

ในตารางที่ 2.1 จะแสดงข้อมูลทั้งหมดสำหรับการสร้าง T_0 ดังแสดงในรูปที่ 2.30 (ข)

ตารางที่ 2.1 แสดงข้อมูลทั้งหมดสำหรับการสร้าง T_0 บน Q_4

x	Binary string	Succ (i)	$2^{Succ(i)}$	Parent (T_2, x)
1	0001	0	1	0
2	0010	-	-	3
3	0011	1	2	1
4	0100	-	-	5
5	0101	2	4	1
6	0110	-	-	7
7	0111	1	2	5
8	1000	-	-	9
9	1001	3	8	1
10	1010	-	-	11
11	1011	1	2	9
12	1100	-	-	13
13	1101	1	2	11
14	1110	-	-	15
15	1111	1	2	13

พิจารณาการสร้าง T_1 จาก Q_4 ได้ดังนี้

โดยให้ X ใดๆ ที่ไม่เท่ากับ 0 ใน Q_4 เป็นไบนารีสตริงค์ $X = x_3x_2x_1x_0$

กรณีที่ 1 ถ้า $x_1 = 0$ จะได้ $X \in \{1, 4, 5, 8, 9, 12, 13\}$ ดังนั้นจะได้ $\text{parent}(T_1, x) = x+2^1 = x+2$

กรณีที่ 2 ถ้า $x_1 = 1$ จะแบ่งได้หลายกรณีดังนี้

$X \in \{3\}$ ดังนั้น $\text{succ}(1) = 0$ จะได้ $\text{parent}(T_1, x) = x-2^0 = 3-1 = 2$

$X \in \{2\}$ ดังนั้น $\text{succ}(1) = 1$ จะได้ $\text{parent}(T_1, x) = x-2^1 = 2-2 = 0$

$X \in \{6, 7, 14, 15\}$ ดังนั้น $\text{succ}(1) = 2$ จะได้ $\text{parent}(T_1, x) = x-2^2 = x-4$

$X \in \{10, 11\}$ ดังนั้น $\text{succ}(1) = 3$ จะได้ $\text{parent}(T_1, x) = x-2^3 = x-8$

ในตารางที่ 2.2 จะแสดงข้อมูลทั้งหมดสำหรับการสร้าง T_1 ดังแสดงในรูปที่ 2.30 (ข)

ตารางที่ 2.2 แสดงข้อมูลทั้งหมดสำหรับการสร้าง T_1 บน Q_4

x	Binary string	Succ (i)	$2^{\text{Succ (i)}}$	Parent (T_1, x)
1	0001	-	-	3
2	0010	1	2	0
3	0011	0	1	2
4	0100	-	-	6
5	0101	-	-	7
6	0110	2	4	2
7	0111	2	4	3
8	1000	-	-	10
9	1001	-	-	11
10	1010	3	8	2
11	1011	3	8	3
12	1100	-	-	14
13	1101	-	-	15
14	1110	2	4	10
15	1111	2	4	11

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

พิจารณาที่การสร้าง T_2 จาก Q_4 ได้ดังนี้

โดยให้ x ใดๆ ที่ไม่เท่ากับ 0 ใน Q_4 เป็นไบนารีสตริงค์ $X = x_3x_2x_1x_0$

กรณีที่ 1 ถ้า $x_2 = 0$ จะได้ $X \in \{1, 2, 3, 8, 9, 10, 11\}$ ดังนั้นจะได้ $\text{parent}(T_2, x) = x+2^2 = x+4$

กรณีที่ 2 ถ้า $x_2 = 1$ จะแบ่งได้หลายกรณีดังนี้

$X \in \{5, 7\}$ ดังนั้น $\text{succ}(2) = 0$ จะได้ $\text{parent}(T_2, x) = x-2^0 = x-1$

$X \in \{6\}$ ดังนั้น $\text{succ}(2) = 1$ จะได้ $\text{parent}(T_2, x) = x-2^1 = 6-2 = 4$

$X \in \{4\}$ ดังนั้น $\text{succ}(2) = 2$ จะได้ $\text{parent}(T_2, x) = x-2^2 = 4-4 = 0$

$X \in \{12, 13, 14, 15\}$ ดังนั้น $\text{succ}(2) = 3$ จะได้ $\text{parent}(T_2, x) = x-2^3 = x-8$

ในตารางที่ 2.3 จะแสดงข้อมูลทั้งหมดสำหรับการสร้าง T_2 ดังแสดงในรูปที่ 2.30 (ข)

ตารางที่ 2.3 แสดงข้อมูลทั้งหมดสำหรับการสร้าง T_2 บน Q_4

x	Binary string	Succ (i)	$2^{\text{Succ (i)}}$	Parent (T_2, x)
1	0001	-	-	5
2	0010	-	-	6
3	0011	-	-	7
4	0100	2	4	0
5	0101	0	1	4
6	0110	1	2	4
7	0111	0	1	6
8	1000	-	-	12
9	1001	-	-	13
10	1010	-	-	14
11	1011	-	-	15
12	1100	3	8	4
13	1101	3	8	5
14	1110	3	8	6
15	1111	3	8	7

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

พิจารณาที่การสร้าง T_3 จาก Q_4 ได้ดังนี้

โดยให้ X ใดๆ ที่ไม่เท่ากับ 0 ใน Q_4 เป็นไบนารีสตริงค์ $X = x_3x_2x_1x_0$

กรณีที่ 1 ถ้า $x_3 = 0$ จะได้ $X \in \{1, 2, 3, 4, 5, 6, 7\}$ ดังนั้นจะได้ $\text{parent}(T_3, x) = x+2^3 = x+8$

กรณีที่ 2 ถ้า $x_3 = 1$ จะแบ่งได้หลายกรณีดังนี้

$X \in \{9, 11, 13, 15\}$ ดังนั้น $\text{succ}(3) = 0$ จะได้ $\text{parent}(T_3, x) = x-2^0 = x-1$

$X \in \{10, 14\}$ ดังนั้น $\text{succ}(3) = 1$ จะได้ $\text{parent}(T_3, x) = x-2^1 = x-2$

$X \in \{12\}$ ดังนั้น $\text{succ}(3) = 2$ จะได้ $\text{parent}(T_3, x) = x-2^2 = 12-4 = 8$

$X \in \{8\}$ ดังนั้น $\text{succ}(3) = 3$ จะได้ $\text{parent}(T_3, x) = x-2^3 = 8-8 = 0$

ในตารางที่ 2.4 จะแสดงข้อมูลทั้งหมดสำหรับการสร้าง T_3 ดังแสดงในรูปที่ 2.30 (ข)

ตารางที่ 2.4 แสดงข้อมูลทั้งหมดสำหรับการสร้าง T_3 บน Q_4

x	Binary string	Succ (i)	$2^{\text{Succ (i)}}$	Parent (T_3, x)
1	0001	-	-	9
2	0010	-	-	10
3	0011	-	-	11
4	0100	-	-	12
5	0101	-	-	13
6	0110	-	-	14
7	0111	-	-	15
8	1000	3	8	0
9	1001	0	1	8
10	1010	1	2	8
11	1011	0	1	10
12	1100	2	4	8
13	1101	0	1	12
14	1110	1	2	12
15	1111	0	1	14

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เมื่อสร้างต้นไม้แบบทอดข้ามที่แตกต่าง จากขั้นตอนวิธีการครบทุกๆ ต้นไม้แบบทอดข้ามที่แตกต่างแล้ว ดังนั้นจะสังเกตเห็นว่าเส้นทางจากรากของ T_i ไปยังโหนดอื่นๆ จะมีเส้นทางที่ไม่ซ้ำกัน ซึ่งจะสามารถค้นหาเส้นทางที่ไม่ซ้ำกันได้โดยการเชื่อม x และ 0 ใน T_i ได้ดังต่อไปนี้ ถ้า $i \in H_x$ สมมติให้ $i = i_p$ ที่ $p = 0, 1, \dots, r-1$ ซึ่งเส้นทางที่ไม่ซ้ำจะหาได้จาก

$$x \xrightarrow{-2^{i_{p-1}}} (x - 2^{i_{p-1}}) \xrightarrow{-2^{i_{p-2}}} (x - 2^{i_{p-1}} - 2^{i_{p-2}}) \xrightarrow{-2^{i_{p-3}}} \dots \xrightarrow{-2^{i_1}} (2^{i_0}) \xrightarrow{-2^{i_0}} 0$$

และถ้า $i \notin H_x$ ซึ่งก็คือเส้นทางที่ไม่ซ้ำกันของ x ไปด้วย $x + 2^i$ ใน T_i

ตัวอย่างที่ 2.3 พิจารณาเส้นทางที่ไม่ซ้ำกัน สำหรับทุกๆ โหนดใน Q_4 ดังต่อไปนี้ 1111(15), 1110(14), 1101(13), 1100(12), 1011(11), 1010(10), 1001(9), 1000(8), 0111(7), 0110(6), 0101(5), 0100(4), 0011(3), 0010(2) และ 0001(1) โดยจะแสดงเส้นทางที่ไม่ซ้ำกันจากเลขประจำโหนดต่างๆ ดังกล่าว ไปยัง 0 ในทุกๆ ต้นไม้แบบทอดข้าม T_i ดังต่อไปนี้ จากรูปที่ 2.30 (ข)

$T_0[15, 0] \quad 15 \xrightarrow{-2^1} 13 \xrightarrow{-2^2} 9 \xrightarrow{-2^3} 1 \xrightarrow{-2^0} 0$	$T_0[7, 0] \quad 7 \xrightarrow{-2^1} 5 \xrightarrow{-2^2} 1 \xrightarrow{-2^0} 0$
$T_1[15, 0] \quad 15 \xrightarrow{-2^2} 11 \xrightarrow{-2^3} 3 \xrightarrow{-2^0} 2 \xrightarrow{-2^1} 0$	$T_1[7, 0] \quad 7 \xrightarrow{-2^2} 3 \xrightarrow{-2^0} 2 \xrightarrow{-2^1} 0$
$T_2[15, 0] \quad 15 \xrightarrow{-2^3} 7 \xrightarrow{-2^0} 6 \xrightarrow{-2^1} 4 \xrightarrow{-2^2} 0$	$T_2[7, 0] \quad 7 \xrightarrow{-2^0} 6 \xrightarrow{-2^1} 4 \xrightarrow{-2^2} 0$
$T_3[15, 0] \quad 15 \xrightarrow{-2^0} 14 \xrightarrow{-2^1} 12 \xrightarrow{-2^2} 8 \xrightarrow{-2^3} 0$	$T_3[7, 0] \quad 7 \xrightarrow{+2^3} 15 \xrightarrow{-2^0} 14 \xrightarrow{-2^1} 12 \xrightarrow{-2^2} 8 \xrightarrow{-2^3} 0$
$T_0[14, 0] \quad 14 \xrightarrow{+2^0} 15 \xrightarrow{-2^1} 13 \xrightarrow{-2^2} 9 \xrightarrow{-2^3} 1 \xrightarrow{-2^0} 0$	$T_0[6, 0] \quad 6 \xrightarrow{+2^0} 7 \xrightarrow{-2^1} 5 \xrightarrow{-2^2} 1 \xrightarrow{-2^0} 0$
$T_1[14, 0] \quad 14 \xrightarrow{-2^2} 10 \xrightarrow{-2^3} 2 \xrightarrow{-2^1} 0$	$T_1[6, 0] \quad 6 \xrightarrow{-2^2} 2 \xrightarrow{-2^1} 0$
$T_2[14, 0] \quad 14 \xrightarrow{-2^3} 6 \xrightarrow{-2^1} 4 \xrightarrow{-2^2} 0$	$T_2[6, 0] \quad 6 \xrightarrow{-2^1} 4 \xrightarrow{-2^2} 0$
$T_3[14, 0] \quad 14 \xrightarrow{-2^1} 12 \xrightarrow{-2^2} 8 \xrightarrow{-2^3} 0$	$T_3[6, 0] \quad 6 \xrightarrow{+2^3} 14 \xrightarrow{-2^1} 12 \xrightarrow{-2^2} 8 \xrightarrow{-2^3} 0$
$T_0[13, 0] \quad 13 \xrightarrow{-2^2} 9 \xrightarrow{-2^3} 1 \xrightarrow{-2^0} 0$	$T_0[5, 0] \quad 5 \xrightarrow{+2^2} 1 \xrightarrow{-2^0} 0$
$T_1[13, 0] \quad 13 \xrightarrow{+2^1} 15 \xrightarrow{-2^2} 11 \xrightarrow{-2^3} 3 \xrightarrow{-2^0} 2 \xrightarrow{-2^1} 0$	$T_1[5, 0] \quad 5 \xrightarrow{+2^1} 7 \xrightarrow{-2^2} 3 \xrightarrow{-2^0} 2 \xrightarrow{-2^1} 0$
$T_2[13, 0] \quad 13 \xrightarrow{-2^3} 5 \xrightarrow{-2^0} 4 \xrightarrow{-2^2} 0$	$T_2[5, 0] \quad 5 \xrightarrow{-2^0} 4 \xrightarrow{-2^2} 0$
$T_3[13, 0] \quad 13 \xrightarrow{-2^0} 12 \xrightarrow{-2^2} 8 \xrightarrow{-2^3} 0$	$T_3[5, 0] \quad 5 \xrightarrow{+2^3} 13 \xrightarrow{-2^0} 12 \xrightarrow{-2^2} 8 \xrightarrow{-2^3} 0$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

$$\begin{aligned}
T_0[12, 0] & 12 \xrightarrow{+2^0} 13 \xrightarrow{-2^2} 9 \xrightarrow{-2^3} 1 \xrightarrow{-2^0} 0 \\
T_1[12, 0] & 12 \xrightarrow{+2^1} 14 \xrightarrow{-2^2} 10 \xrightarrow{-2^3} 2 \xrightarrow{-2^1} 0 \\
T_2[12, 0] & 12 \xrightarrow{-2^3} 4 \xrightarrow{-2^2} 0 \\
T_3[12, 0] & 12 \xrightarrow{-2^2} 8 \xrightarrow{-2^3} 0
\end{aligned}$$

$$\begin{aligned}
T_0[4, 0] & 4 \xrightarrow{+2^0} 5 \xrightarrow{-2^2} 1 \xrightarrow{-2^0} 0 \\
T_1[4, 0] & 4 \xrightarrow{+2^1} 6 \xrightarrow{-2^2} 2 \xrightarrow{-2^1} 0 \\
T_2[4, 0] & 4 \xrightarrow{-2^2} 0 \\
T_3[4, 0] & 4 \xrightarrow{+2^3} 12 \xrightarrow{-2^2} 8 \xrightarrow{-2^3} 0
\end{aligned}$$

$$\begin{aligned}
T_0[11, 0] & 11 \xrightarrow{-2^1} 9 \xrightarrow{-2^3} 1 \xrightarrow{-2^0} 0 \\
T_1[11, 0] & 11 \xrightarrow{-2^3} 3 \xrightarrow{-2^0} 2 \xrightarrow{-2^1} 0 \\
T_2[11, 0] & 11 \xrightarrow{+2^2} 15 \xrightarrow{-2^3} 7 \xrightarrow{-2^0} 6 \xrightarrow{-2^1} 4 \xrightarrow{-2^2} 0 \\
T_3[11, 0] & 11 \xrightarrow{+2^0} 10 \xrightarrow{-2^1} 8 \xrightarrow{-2^3} 0
\end{aligned}$$

$$\begin{aligned}
T_0[3, 0] & 3 \xrightarrow{-2^1} 1 \xrightarrow{-2^0} 0 \\
T_1[3, 0] & 3 \xrightarrow{-2^0} 2 \xrightarrow{-2^1} 0 \\
T_2[3, 0] & 3 \xrightarrow{+2^2} 7 \xrightarrow{-2^0} 6 \xrightarrow{-2^1} 4 \xrightarrow{-2^2} 0 \\
T_3[3, 0] & 3 \xrightarrow{+2^3} 11 \xrightarrow{-2^0} 10 \xrightarrow{-2^1} 8 \xrightarrow{-2^3} 0
\end{aligned}$$

$$\begin{aligned}
T_0[10, 0] & 10 \xrightarrow{+2^0} 11 \xrightarrow{-2^1} 9 \xrightarrow{-2^3} 1 \xrightarrow{-2^0} 0 \\
T_1[10, 0] & 10 \xrightarrow{-2^3} 2 \xrightarrow{-2^1} 0 \\
T_2[10, 0] & 10 \xrightarrow{+2^2} 14 \xrightarrow{-2^3} 6 \xrightarrow{-2^1} 4 \xrightarrow{-2^2} 0 \\
T_3[10, 0] & 10 \xrightarrow{-2^1} 8 \xrightarrow{-2^3} 0
\end{aligned}$$

$$\begin{aligned}
T_0[2, 0] & 2 \xrightarrow{+2^0} 3 \xrightarrow{-2^1} 1 \xrightarrow{-2^0} 0 \\
T_1[2, 0] & 2 \xrightarrow{-2^1} 0 \\
T_2[2, 0] & 2 \xrightarrow{+2^2} 6 \xrightarrow{-2^1} 4 \xrightarrow{-2^2} 0 \\
T_3[2, 0] & 2 \xrightarrow{+2^3} 10 \xrightarrow{-2^1} 8 \xrightarrow{-2^3} 0
\end{aligned}$$

$$\begin{aligned}
T_0[9, 0] & 9 \xrightarrow{-2^3} 1 \xrightarrow{-2^0} 0 \\
T_1[9, 0] & 9 \xrightarrow{+2^1} 11 \xrightarrow{-2^3} 3 \xrightarrow{-2^0} 2 \xrightarrow{-2^1} 0 \\
T_2[9, 0] & 9 \xrightarrow{+2^2} 13 \xrightarrow{-2^3} 5 \xrightarrow{-2^0} 4 \xrightarrow{-2^2} 0 \\
T_3[9, 0] & 9 \xrightarrow{-2^0} 8 \xrightarrow{-2^3} 0
\end{aligned}$$

$$\begin{aligned}
T_0[1, 0] & 1 \xrightarrow{-2^0} 0 \\
T_1[1, 0] & 1 \xrightarrow{+2^1} 3 \xrightarrow{-2^0} 2 \xrightarrow{-2^1} 0 \\
T_2[1, 0] & 1 \xrightarrow{+2^2} 5 \xrightarrow{-2^0} 4 \xrightarrow{-2^2} 0 \\
T_3[1, 0] & 1 \xrightarrow{+2^3} 9 \xrightarrow{-2^1} 8 \xrightarrow{-2^3} 0
\end{aligned}$$

$$\begin{aligned}
T_0[8, 0] & 8 \xrightarrow{+2^0} 9 \xrightarrow{-2^3} 1 \xrightarrow{-2^0} 0 \\
T_1[8, 0] & 8 \xrightarrow{+2^1} 10 \xrightarrow{-2^3} 2 \xrightarrow{-2^1} 0 \\
T_2[8, 0] & 8 \xrightarrow{+2^2} 12 \xrightarrow{-2^3} 4 \xrightarrow{-2^2} 0 \\
T_3[8, 0] & 8 \xrightarrow{-2^3} 0
\end{aligned}$$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 3

การสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน บนระบบเครือข่ายแบบไดนามิก

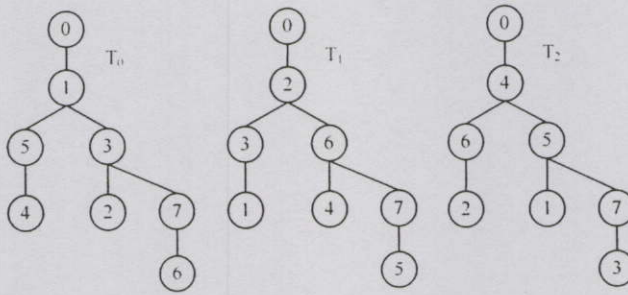
งานวิจัยนี้เป็นการวิจัยที่เสนอขั้นตอนวิธีการที่ยืดหยุ่นสำหรับการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน (Parallel Construction of Independent Spanning Tree) บนระบบเครือข่ายแบบไดนามิก (Dynamic Network) โดยจะเริ่มจากการนำขั้นตอนวิธีของหลักการที่เรียกว่าจัตุรัสลาตินของระยะทางแฮมมิง (Hamming Distance Latin Square) ที่นำมาสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนานบนเครือข่ายแบบไฮเปอร์คิวบิกที่มีผู้เสนอไว้แล้ว [8] มาใช้สร้างฟังก์ชันเส้นทาง (Routing Function) สำหรับกำหนดค่าสวิตช์ให้กับระบบเครือข่ายแบบไดนามิก และเมื่อสร้างฟังก์ชันเส้นทางได้จนครบทุกฟังก์ชัน ก็ทำการสร้างต้นไม้แบบทอดข้ามที่แตกต่างโดยใช้ฟังก์ชันเส้นทางได้ บนระบบเครือข่ายแบบไดนามิกซึ่งสามารถประยุกต์ใช้ได้บน 3 เครือข่าย คือ เครือข่ายแบบคิวบิก (Cube Network) เครือข่ายแบบโอเมก้า (Omega Network) และเครือข่ายแบบฟลิป (Flip Network) เป็นต้น

เนื้อหาในบทที่ 3 นี้จะมีการแบ่งออกเป็น 4 ส่วนดังต่อไปนี้ ส่วนที่ 1 แสดงต้นไม้แบบทอดข้ามที่แตกต่าง (ISTs) ส่วนที่ 2 คุณสมบัติเบื้องต้น ส่วนที่ 3 ขั้นตอนวิธีการแบบขนาน สำหรับการสร้างต้นไม้แบบทอดข้ามที่แตกต่างบนระบบเครือข่ายแบบไดนามิกแบบมัลติสเตจ (Multistages Interconnection Network: MINs) [015, 03, 012, 08] ส่วนที่ 4 การประยุกต์ใช้การสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน โดยใช้ฟังก์ชันเส้นทางบนเครือข่ายแบบมัลติสเตจ

3.1 ต้นไม้แบบทอดข้ามที่แตกต่าง

ในการสร้างต้นไม้แบบทอดข้ามที่แตกต่าง (Independent Spanning Trees: ISTs) ที่มีโหนดราก $r = 0$ บนระบบเครือข่ายแบบไดนามิก (Dynamic Network) เมื่อกำหนดให้ $parent(T, x)$ แทนโหนดพ่อแม่ของโหนด x ($\neq r$) ซึ่งเป็นโหนดที่เชื่อมต่อกับ x ในเส้นทาง $T[r, x]$ ดังนั้นจึงสามารถแสดงต้นไม้แบบทอดข้ามที่แตกต่างดังแสดงในรูปที่ 3.1 ($N = 8$) และรูปที่ 3.2 ($N = 16$) ถ้าจำนวนหน่วยประมวลผลทั้งหมดจะเท่ากับ 8 หน่วยประมวลผล ($N = 8$) จะสามารถสร้างต้นไม้ได้ 3 ต้น ใช้สัญลักษณ์แทนคือ T_0 , T_1 และ T_2 และในกรณีทั่วไปที่มีจำนวนหน่วยประมวลผลทั้งหมดเท่ากับ $N = 2^k$ โดยที่ k เป็นจำนวนบิตของเลขประจำโหนดซึ่งใช้แทนเป็นเลขฐาน 2 ซึ่งจะสามารถ

สร้างต้นไม้แบบทอดข้ามที่แตกต่างได้ k ต้น ทั้งบนเครือข่ายแบบไฮเปอร์คิว และระบบเครือข่ายแบบไดนามิก



รูปที่ 3.1 ชุดต้นไม้แบบทอดข้ามที่แตกต่างบนระบบเครือข่ายแบบไดนามิก ($N = 8$)



รูปที่ 3.2 ชุดต้นไม้แบบทอดข้ามที่แตกต่างบนระบบเครือข่ายแบบไดนามิก ($N = 16$)

3.2 คุณสมบัติเบื้องต้น

ในเบื้องต้นจะนำเสนอจตุรัสลาติน [7] ซึ่งนำมาจากการประยุกต์ใช้บนเครือข่ายแบบไฮเปอร์คิว [18] โดยนำเสนอไว้ในหัวข้อ 2.6.1 ให้สามารถสร้างต้นไม้แบบทอดข้ามที่แตกต่างบนระบบเครือข่ายแบบไดนามิก โดยให้ $x = (b_{k-1}, b_{k-2}, \dots, b_0)$ เป็นเลขประจำโหนดใดๆ แบบไบนารีของหน่วยประมวลผล และให้ $E(x)$ เป็นการเรียงสับเปลี่ยนแบบสลับ (Exchange Permutation) (ซึ่งมีการการออกแบบไว้ในหัวข้อที่ 2.3.3 ในบทที่ 2) หรือจะเรียกว่าเส้นทางการสับสวิตช์ (Exchange Path) ในหัวข้อนี้

$$E(b_{k-1}b_{k-2} \dots b_{i-1}b_ib_{i-1} \dots b_1b_0) = b_{k-1}b_{k-2} \dots b_{i+1}\bar{b}_ib_{i-1} \dots b_1b_0$$

เป็นฟังก์ชันที่สับเปลี่ยนบิต (b_i) ในตำแหน่งที่ i เมื่อ $0 \leq i \leq k-1$ ดังนั้นจึงสามารถเอกสารนี้เป็นเอกสารทูลงทุนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ออกแบบเมตริกซ์ที่เรียกว่าเส้นทางการสับเปลี่ยนที่เพิ่มขึ้นในจตุรัสลาติน (Increasing Exchange Path Latin Square : EPLS) หรือเรียกสั้นๆ ว่าเมตริกซ์จตุรัสลาติน ได้ดังต่อไปนี้

$$I_k(x) = \begin{bmatrix} i_0 & i_1 & \cdots & i_{t-2} & i_{t-1} \\ i_1 & i_2 & \cdots & i_{t-1} & i_0 \\ \vdots & & & & \\ i_{t-2} & i_{t-1} & \cdots & i_{t-4} & i_{t-3} \\ i_{t-1} & i_0 & \cdots & i_{t-3} & i_{t-2} \end{bmatrix}$$

ในแถวที่ 1 ของเมตริกซ์จัตุรัสลาติน $(i_0 < i_1 < \cdots < i_{t-1})$ โดยที่ i_j ใช้แทนตำแหน่งใดๆ ที่เป็น 1 ใน x ที่ซึ่ง $0 \leq i_j \leq k-1$ (เช่น $i_j = 0$ = ตำแหน่งบิตที่น้อยที่สุด, $i_j = k-1$ = ตำแหน่งบิตที่มากที่สุด) และ t จะใช้แทนจำนวนของบิตที่เป็น 1 ใน x

ในเมตริกซ์จัตุรัสลาติน ถ้าชี้ที่ i_{j-1} ใน $I_k(x)$ ก็จะเรียกว่า บรรพบุรุษ (Predecessor) ของ i_j ($pred(i_j) = i_{j-1}$) แสดงโดยใช้สัญลักษณ์ $pred(i_j)$ ซึ่งถ้าเป็นเมตริกซ์ขนาด 1×1 บรรพบุรุษ คือตัวของมันเอง ($pred(i_j) = i_j$)

ตัวอย่างที่ 3.1 การหาเมตริกซ์จัตุรัสลาติน กรณี $N=8$

สำหรับขนาด $N=8$ และ $k=3$ ทุกๆ เลขประจำไบนารีแบบไบนารีที่อาจจะเป็นไปได้ของระบบที่มี 8 หน่วยประมวลผล (PEs) คือ 000, 001, 010, 011, 100, 101, 110, 111 เช่น ถ้าพิจารณาที่เลขประจำไบนารีของ $s = 111_2$ (หรือ 7) ดังนั้นจึงสามารถสร้างเมตริกซ์ขนาด 3×3 ได้ดังนี้

$$I_k(s) = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \end{bmatrix}$$

ในกรณีอื่นๆ ถ้าพิจารณาที่เลขประจำไบนารีของ $u = 011_2$ (หรือ 3), $v = 101_2$ (หรือ 5) และ $w = 110_2$ (หรือ 6) ดังนั้นจึงสามารถสร้างเมตริกซ์ขนาด 2×2 ได้ดังนี้

$$I_E(u) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, I_E(v) = \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix}, I_E(w) = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

ในกรณีต่อมาถ้าพิจารณาที่เลขประจำไบนารีของ $x = 001_2$ (หรือ 1), $y = 010_2$ (หรือ 2)

และ $z = 100_2$ (หรือ 4) ดังนั้นจึงสามารถสร้างเมตริกซ์ขนาด 1×1 ได้ดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

$$I_E(x) = [0], I_E(y) = [1], I_E(z) = [2]$$

ตัวอย่างที่ 3.2 การหาเมตริกซ์จัตุรัสลาติน กรณี $N = 16$

สำหรับขนาด $N = 16$ และ $k = 4$ ทุกๆ เลขประจำโหนดแบบไบนารีที่อาจจะเป็นไปได้ของระบบที่มี 16 หน่วยประมวลผล (PEs) คือ 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111 เช่นถ้าพิจารณาที่เลขประจำโหนดของ $a = 1111_2$ (หรือ 15) ดังนั้นจึงสามารถสร้างเมตริกซ์ขนาด 4×4 ได้ดังนี้

$$I_E(a) = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 0 \\ 2 & 3 & 0 & 1 \\ 3 & 0 & 1 & 2 \end{bmatrix}$$

ในกรณีต่อมา ถ้าพิจารณาที่เลขประจำโหนดของ $b = 1011_2$ (หรือ 11), $c = 1101_2$ (หรือ 13) และ $d = 1110_2$ (หรือ 14) ดังนั้นจึงสามารถสร้างเมตริกซ์ขนาด 3×3 ได้ดังนี้

$$I_E(b) = \begin{bmatrix} 0 & 1 & 3 \\ 1 & 3 & 0 \\ 3 & 0 & 1 \end{bmatrix}, I_E(c) = \begin{bmatrix} 0 & 2 & 3 \\ 2 & 3 & 0 \\ 3 & 0 & 2 \end{bmatrix}, I_E(d) = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix}$$

ในกรณีอื่นๆ ถ้าพิจารณาที่เลขประจำโหนดของ $e = 1001_2$ (หรือ 9), $f = 1010_2$ (หรือ 10) และ $g = 1100_2$ (หรือ 12) ดังนั้นจึงสามารถสร้างเมตริกซ์ขนาด 2×2 ได้ดังนี้

$$I_E(e) = \begin{bmatrix} 0 & 3 \\ 3 & 0 \end{bmatrix}, I_E(f) = \begin{bmatrix} 1 & 3 \\ 3 & 1 \end{bmatrix}, I_E(g) = \begin{bmatrix} 2 & 3 \\ 3 & 2 \end{bmatrix}$$

ในกรณีสุดท้ายถ้าพิจารณาที่เลขประจำโหนดของ $h = 1000_2$ (หรือ 8) ดังนั้นจึงสามารถสร้างเมตริกซ์ขนาด 1×1 ได้ดังนี้

$$I_E(x) = [3]$$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.3 ขั้นตอนวิธีการแบบขนานบนเครือข่ายแบบมัลติสเตจสำหรับสร้างต้นไม้แบบทอดข้ามที่แตกต่าง

ในหัวข้อนี้จะนำเสนอขั้นตอนวิธีการแบบขนานสำหรับสร้างต้นไม้แบบทอดข้ามที่แตกต่าง บนเครือข่ายแบบมัลติสเตจ (MINs: Multistages Interconnection Networks) ใดๆ ซึ่งงานวิจัยนี้จะใช้ขั้นตอนวิธีการดังต่อไปนี้

Algorithm PARALLEL CONSTRUCTION OF ISTs

Begin

For all PEs with address $x (\neq r)$ pardo

Where $x = b_{k-1}b_{k-2}...b_0$

Part I: // Create k routing functions (R_i)

For $i=0$ to $k-1$ do

If ($b_i=0$)

parent(x)= $E_i(x)$;

$R_i = R_i \cup \{(x, \text{parent}(x))\}$ for T_i

End if

If ($b_i=1$)

parent(x)= $E_{\text{pred}(i)}(x)$;

$R_{\text{pred}(i)} = R_{\text{pred}(i)} \cup \{(x, \text{parent}(x))\}$ for T_i

End if

End for i

Part II: // Construct k ISTs using R_i

For $i=0$ to $k-1$ do

Create partially $T_0, T_1, ..., T_{k-1}$ using R_i ;

End for i

End for all x

End

สำหรับขั้นตอนวิธีการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนานบนเครือข่ายแบบมัลติสเตจ สามารถแบ่งออกได้เป็น 2 ส่วน ดังต่อไปนี้คือ

1. การสร้างฟังก์ชันเส้นทางบนเครือข่ายแบบมัลติสเตจ สำหรับสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน
2. การสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนานโดยใช้ฟังก์ชันเส้นทางบนเครือข่ายแบบมัลติสเตจ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.3.1 การสร้างฟังก์ชันเส้นทางบนเครือข่ายแบบมัลติสเทจ สำหรับสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน

ในขั้นตอนวิธีการนี้ เพื่อให้สามารถเข้าใจกระบวนการหาฟังก์ชันเส้นทางได้ง่ายขึ้นได้ สมมุติให้ : X แทนเลขประจำโหนดใดๆ แบบไบนารีดังนี้

$$X = (b_{k-1}b_{k-2} \dots b_i \dots b_2b_1b_0)$$

และให้ Y เป็นพ่อและแม่ของ X ($parent(x)$) ซึ่งจะแสดงความสัมพันธ์ได้ดังรูปที่ 3.3



รูปที่ 3.3 แสดงการเชื่อมต่อระหว่างโหนด X และ Y

ในขั้นตอนการสร้างฟังก์ชันเส้นทางซึ่งก็คือการสร้างเส้นเชื่อมต่อระหว่างโหนด X และ Y ดังนั้น การค้นหา Y จะแบ่งได้อีก 2 กรณี ดังต่อไปนี้

กรณีที่ 1: ถ้า $b_i = 0$ ดังนั้น

$$Y = (b_{k-1}b_{k-2} \dots b_{i+1}\bar{b}_ib_{i-1} \dots b_1b_0)$$

ที่ซึ่ง $\bar{b}_i$ คือการสลับบิต ณ ตำแหน่งของบิตที่ i

กรณีที่ 2: ถ้า $b_i = 1$ จึงต้องการเมตริกซ์จัดรหัสลาติน เพื่อค้นหาตำแหน่งของบิตที่เหมาะสมที่ j โดยที่ $j = pred(i)$ ซึ่งมีการออกแบบไว้ในหัวข้อที่ 3.2 ดังนั้น

$$Y = (b_{k-1}b_{k-2} \dots \bar{b}_j \dots b_2b_1b_0)$$

ที่ซึ่ง $\bar{b}_j$ คือการสลับบิต ณ ตำแหน่งของบิตที่ j

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตัวอย่างที่ 3.3 การสร้างฟังก์ชันเส้นทางสำหรับสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนานบนเครือข่ายแบบมัลติสเตจ ($N = 8$)

สำหรับเครือข่ายแบบมัลติสเตจ ขนาด $N = 8$ ในกรณีของ $k = 3$ จะได้ต้นไม้แบบทอดข้ามที่แตกต่าง 3 ต้น ดังรูปที่ 3.1 ซึ่งก่อนจะสร้างต้นไม้ดังกล่าว จะต้องสร้างฟังก์ชันเส้นทาง R_0 , R_1 และ R_2 ให้ครบทุกฟังก์ชันดังจะสามารถคำนวณได้ต่อไปนี้

ในการทำซ้ำรอบที่ 1 ตารางที่ 3.1 แสดงบางส่วนของฟังก์ชันเส้นทาง R_0 , R_1 และ R_2 สำหรับสร้าง T_0 ทุกๆ $N-1$ หน่วยประมวลผล คำนวณจากขั้นตอนวิธีในส่วนแรกเมื่อ $i = 0$ ดังนี้

```

Part I: // Create k routing functions ( $R_i$ )
For  $i=0$  to  $k-1$  do
  If ( $b_i=0$ )
     $\text{parent}(x)=E_i(x)$ ;
     $R_i = R_i \cup \{(x, \text{parent}(x))\}$  for  $T_i$ 
  End if
  If ( $b_i=1$ )
     $\text{parent}(x)=E_{\text{pred}(i)}(x)$ ;
     $R_{\text{pred}(i)} = R_{\text{pred}(i)} \cup \{(x, \text{parent}(x))\}$  for  $T_i$ 
  End if
End for i
  
```

กรณีที่ 1: ถ้า $b_0 = 0$ สมาชิกของ X ทั้งหมดที่เป็นไปได้คือ 010(2), 100(4), 110(6) ดังนั้น

$$Y = (b_{k-1}b_{k-2} \dots b_i \dots b_2b_1\bar{b}_0)$$

ที่จะสามารถค้นหา Y ได้จากการสลับบิตตำแหน่งที่ 0 ของ X คือ 011(3), 101(5), 111(7) และเขียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_0 ดังนี้
 $R_0 = R_0 \cup \{(2 \rightarrow 3), (4 \rightarrow 5), (6 \rightarrow 7)\}$

กรณีที่ 2: ถ้า $b_0 = 1$ สมาชิกของ X ที่เป็นไปได้ทั้งหมดคือ 001(1), 011(3), 101(5), 111(7) ดังนั้น

$$Y = (b_{k-1}b_{k-2} \dots \bar{b}_i \dots b_2b_1b_0)$$

จะสามารถหา Y ได้ดังต่อไปนี้

ถ้า $X = 001(1)$ ดังนั้น $j = \text{pred}(0) = 0$ (ซึ่งได้อ้างถึงในเมตริกซ์ $I_k(x)$) ที่นี้

Y ก็คือการสลับบิตตำแหน่งที่ 0 ของ X ดังนี้

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้
 $X = 001(1)$ ดังนั้น $Y = 000(0)$

และเขียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_0 ดังนี้ $R_0 = R_0 \cup \{(1 \rightarrow 0)\}$

ถ้า $X = 011(3)$ ดังนั้น $j = \text{pred}(0) = 1$ (ซึ่งได้ข้างถึงในเมตริกซ์ $I_E(u)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 1 ของ x ดังนี้

$$X = 011(3) \text{ ดังนั้น } Y = 001(1)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_1 ดังนี้ $R_1 = R_1 \cup \{(3 \rightarrow 1)\}$

ถ้า $X \in \{101(5), 111(7)\}$ ดังนั้น $\text{pred}(0) = 2$ (ซึ่งได้ข้างถึงในเมตริกซ์ $I_E(v)$ และ $I_E(s)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 2 ของ X ดังนี้

$$X = 101(5) \text{ ดังนั้น } Y = 001(1)$$

$$X = 111(7) \text{ ดังนั้น } Y = 011(3)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_2 ดังนี้

$$R_2 = R_2 \cup \{(5 \rightarrow 1), (7 \rightarrow 3)\}$$

สรุปสำหรับ $i = 0$ จะสามารถสร้างฟังก์ชันเส้นทาง (บางส่วน) สำหรับ T_0 ได้ดังต่อไปนี้

$$R_0 = \{(1 \rightarrow 0), (2 \rightarrow 3), (4 \rightarrow 5), (6 \rightarrow 7)\}$$

$$R_1 = \{(3 \rightarrow 1)\}$$

$$R_2 = \{(5 \rightarrow 1), (7 \rightarrow 3)\}$$

ตารางที่ 3.1 แสดงบางส่วนของฟังก์ชันเส้นทาง (R_0, R_1, R_2) สำหรับ T_0 ($i = 0$) เมื่อ $k = 3$

x	Binary string	$\text{pred}(i)$	$E_{\text{pred}(i)}$	$\text{parent}(x)$	R_i	$R_{\text{pred}(i)}$
1	001	0	0	0	-	0
2	010	-	-	3	0	-
3	011	1	1	1	-	1
4	100	-	-	5	0	-
5	101	2	1	1	-	2
6	110	-	-	7	0	-
7	111	2	3	3	-	2

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ในการทำซ้ำรอบที่ 2 ตารางที่ 3.2 แสดงบางส่วนของฟังก์ชันเส้นทาง R_0, R_1 และ R_2 สำหรับสร้าง T_i ทุกๆ $N-1$ หน่วยประมวลผลคำนวณจากขั้นตอนวิธีในส่วนแรกเมื่อ $i = 1$ ดังนี้

กรณีที่ 1: ถ้า $b_1 = 0$ สมาชิกของ X ทั้งหมดที่เป็นไปได้คือ 001(1), 100(4), 101(5) ดังนั้น

$$Y = (b_{k-1}b_{k-2} \dots b_i \dots b_2\bar{b}_1b_0)$$

ที่ซึ่งจะสามารถค้นหา Y ได้จากการสลับบิตตำแหน่งที่ 1 ของ X คือ 011(3), 110(6), 111(7) และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_1 ดังนี้ $R_1 = R_0 \cup \{(1 \rightarrow 3), (4 \rightarrow 6), (5 \rightarrow 7)\}$

กรณีที่ 2: ถ้า $b_1 = 1$ สมาชิกของ X ที่เป็นไปได้ทั้งหมดคือ 010(2), 011(3), 110(6), 111(7) ดังนั้น

$$Y = (b_{k-1}b_{k-2} \dots \bar{b}_1 \dots b_2b_1b_0)$$

จะสามารถหา Y ได้ดังต่อไปนี้

ถ้า $X \in \{011(3), 111(7)\}$ ดังนั้น $j = \text{pred}(1) = 0$ (ซึ่งได้อ้างถึงในเมตริกซ์ $I_E(u)$ และ $I_E(s)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 0 ของ X ดังนี้

$$X = 011(3) \text{ ดังนั้น } Y = 010(2)$$

$$X = 111(7) \text{ ดังนั้น } Y = 110(6)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_0 ดังนี้

$$R_0 = R_0 \cup \{(3 \rightarrow 2), (7 \rightarrow 6)\}$$

ถ้า $X = 010(2)$ ดังนั้น $j = \text{pred}(1) = 1$ (ซึ่งได้อ้างถึงในเมตริกซ์ $I_E(y)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 1 ของ X ดังนี้

$$X = 010(2) \text{ ดังนั้น } Y = 000(0)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_1 ดังนี้ $R_1 = R_1 \cup \{(2 \rightarrow 0)\}$

ถ้า $X = 110(6)$ ดังนั้น $j = \text{pred}(1) = 2$ (ซึ่งได้อ้างถึงในเมตริกซ์ $I_E(w)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 2 ของ X ดังนี้

$$X = 110(6) \text{ ดังนั้น } Y = 010(2)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปใน R_2 ดังนี้ $R_2 = R_2 \cup \{(6 \rightarrow 2)\}$

สรุปสำหรับ $i = 1$ จะสามารถสร้างฟังก์ชันเส้นทาง (บางส่วน) สำหรับ T_1 ได้ดังต่อไปนี้

$$R_0 = R_0 \text{ สำหรับ } T_0 \cup \{(3 \rightarrow 2), (7 \rightarrow 6)\}$$

$$R_1 = R_1 \text{ สำหรับ } T_0 \cup \{(1 \rightarrow 3), (4 \rightarrow 6), (5 \rightarrow 7), (2 \rightarrow 0)\}$$

$$R_2 = R_2 \text{ สำหรับ } T_0 \cup \{(6 \rightarrow 2)\}$$

ตารางที่ 3.2 แสดงบางส่วนของฟังก์ชันเส้นทาง (R_0, R_1, R_2) สำหรับ T_1 ($i = 1$) เมื่อ $k = 3$

x	Binary string	$pred(i)$	$E_{pred(i)}$	$parent(x)$	R_i	$R_{pred(i)}$
1	001	-	-	3	1	-
2	010	1	0	0	-	1
3	011	0	2	2	-	0
4	100	-	-	6	1	-
5	101	-	-	7	1	-
6	110	2	2	2	-	2
7	111	0	6	6	-	0

ในการทำซ้ำรอบที่ 3 ตารางที่ 3.3 แสดงบางส่วนของฟังก์ชันเส้นทาง R_0, R_1 และ R_2 สำหรับสร้าง T_2 ทุกๆ $N-1$ หน่วยประมวลผล คำนวณจากขั้นตอนวิธีในส่วนแรกเมื่อ $i = 2$ ดังนี้

กรณีที่ 1: ถ้า $b_2 = 0$ สมาชิกของ X ทั้งหมดที่เป็นไปได้คือ 001(1), 010(2), 011(3) ดังนั้น

$$Y = (b_{k-1} b_{k-2} \dots b_i \dots \overline{b_2} b_1 b_0)$$

ที่จะสามารถค้นหา Y ได้จากการสลับบิตตำแหน่งที่ 2 ของ X คือ 101(5), 110(6), 111(7) และเขียนเข้า ($X \rightarrow Y$) ไปในฟังก์ชัน R_2 ดังนี้

$$R_2 = R_2 \cup \{(1 \rightarrow 5), (2 \rightarrow 6), (3 \rightarrow 7)\}$$

กรณีที่ 2: ถ้า $b_2 = 1$ สมาชิกของ X ที่เป็นไปได้ทั้งหมดคือ 100(4), 101(5), 110(6), 111(7) ดังนั้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้ $Y = (b_{k-1} b_{k-2} \dots \overline{b_i} \dots \overline{b_2} b_1 b_0)$ กรุณาให้ไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้
จะสามารถหา Y ได้ดังต่อไปนี้

ถ้า $X = 101(5)$ ดังนั้น $j = \text{pred}(2) = 0$ (ซึ่งได้อ้างถึงในเมตริกซ์ $I_E(v)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 0 ของ X ดังนี้

$$X = 10\bar{1}(5) \text{ ดังนั้น } Y = 100(4)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_0 ดังนี้ $R_0 = R_0 \cup \{(5 \rightarrow 4)\}$

ถ้า $X \in \{110(6), 111(7)\}$ ดังนั้น $j = \text{pred}(2) = 1$ (ซึ่งได้อ้างถึงในเมตริกซ์ $I_E(w)$ และ $I_E(s)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 1 ของ X ดังนี้

$$X = 1\bar{1}0(6) \text{ ดังนั้น } Y = 100(4)$$

$$X = 1\bar{1}1(7) \text{ ดังนั้น } Y = 101(5)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_1 ดังนี้

$$R_1 = R_1 \cup \{(6 \rightarrow 4), (7 \rightarrow 5)\}$$

ถ้า $X = 100(4)$ ดังนั้น $j = \text{pred}(2) = 2$ (ซึ่งได้อ้างถึงในเมตริกซ์ $I_E(z)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 2 ของ x ดังนี้

$$X = 100(4) \text{ ดังนั้น } Y = 000(0)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปใน R_2 ดังนี้ $R_2 = R_2 \cup \{(4 \rightarrow 0)\}$

สรุปสำหรับ $i=2$ จะสามารถสร้างฟังก์ชันเส้นทาง (บางส่วน) สำหรับ T_2 ได้ดังต่อไปนี้

$$R_0 = R_0 \text{ สำหรับ } T_0 \text{ และ } T_1 \cup \{(5 \rightarrow 4)\}$$

$$R_1 = R_1 \text{ สำหรับ } T_0 \text{ และ } T_1 \cup \{(6 \rightarrow 4), (7 \rightarrow 5)\}$$

$$R_2 = R_2 \text{ สำหรับ } T_0 \text{ และ } T_1 \cup \{(1 \rightarrow 5), (2 \rightarrow 6), (3 \rightarrow 7), (4 \rightarrow 0)\}$$

ตารางที่ 3.3 แสดงบางส่วนของฟังก์ชันเส้นทาง (R_0, R_1, R_2) สำหรับ T_2 ($i=2$) เมื่อ $k=3$

x	Binary string	$\text{pred}(i)$	$E_{\text{pred}(i)}$	$\text{parent}(x)$	R_i	$R_{\text{pred}(i)}$
1	001	-	-	5	2	-
2	010	-	-	6	2	-
3	011	-	-	7	2	-
4	100	2	0	0	-	2
5	101	0	4	4	-	0
6	110	1	4	4	-	1
7	111	1	5	5	-	1

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตัวอย่างที่ 3.4 การสร้างฟังก์ชันเส้นทางสำหรับสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนานบนเครือข่ายแบบมัลติสเตจ ($N = 16$)

สำหรับเครือข่ายแบบมัลติสเตจ ขนาด $N = 16$ ในกรณีของ $k = 4$ จะได้ต้นไม้แบบทอดข้ามที่แตกต่าง 4 ต้น ดังรูปที่ 3.2 ซึ่งก่อนจะสร้างต้นไม้ดังกล่าว จะต้องสร้างฟังก์ชันเส้นทาง R_0, R_1, R_2 และ R_3 ให้ครบทุกฟังก์ชันจึงจะสามารถคำนวณได้ต่อไป

ในการทำซ้ำรอบที่ 1 ตารางที่ 3.4 แสดงบางส่วนของฟังก์ชันเส้นทาง R_0, R_1, R_2 และ R_3 สำหรับสร้าง T_0 ทุกๆ $N-1$ หน่วยประมวลผล คำนวณจากขั้นตอนวิธีในส่วนแรกเมื่อ $i = 0$ ดังนี้

Part I: // Create k routing functions (R_i)

For $i=0$ to $k-1$ do

 If ($b_i=0$)

$\text{parent}(x)=E_i(x);$

$R_i = R_i \cup \{(x, \text{parent}(x))\}$ for T_i

 End if

 If ($b_i=1$)

$\text{parent}(x)=E_{\text{pred}(i)}(x);$

$R_{\text{pred}(i)} = R_{\text{pred}(i)} \cup \{(x, \text{parent}(x))\}$ for T_i

 End if

End for i

กรณีที่ 1: ถ้า $b_0 = 0$ สมาชิกของ X ทั้งหมดที่เป็นไปได้คือ 0010(2), 0100(4), 0110(6), 1000(8), 1010(10), 1100(12), 1110(14) ดังนั้น

$$Y = (b_k, b_{k-1}, \dots, \bar{b}_i, \dots, \bar{b}_1, \bar{b}_0)$$

ที่จะสามารถค้นหา Y ได้จากการสลับบิตตำแหน่งที่ 0 ของ X คือ 0011(3), 0101(5), 0111(7), 1001(9), 1011(11), 1101(13), 1111(15) และยูเนียน ($X \rightarrow Y$) เข้าไปใน R_0 ดังนี้ $R_0 = R_0 \cup \{(2 \rightarrow 3), (4 \rightarrow 5), (6 \rightarrow 7), (8 \rightarrow 9), (10 \rightarrow 11), (12 \rightarrow 13), (14 \rightarrow 15)\}$

กรณีที่ 2: ถ้า $b_0 = 1$ สมาชิกของ X ที่เป็นไปได้ทั้งหมดคือ 0001(1), 0011(3), 0101(5), 0111(7), 1001(9), 1011(11), 1101(13), 1111(15) ดังนั้น

$$Y = (b_k, b_{k-1}, \dots, \bar{b}_i, \dots, b_1, b_0)$$

จะสามารถหา Y ได้ดังต่อไปนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่ควรเผยแพร่ไปยังสาธารณะโดยไม่ได้รับอนุญาต
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้
 Y ก็คือการสลับบิตตำแหน่งที่ 0 ของ X ดังนี้

$$X = 0001(1) \text{ ดังนั้น } Y = 0000(0)$$

$$X = 000\underline{1}(1) \text{ ดังนั้น } Y = 0000(0)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปใน R_0 ดังนั้น $R_0 = R_0 \cup \{(1 \rightarrow 0)\}$

ถ้า $X = 001\underline{1}(3)$ ดังนั้น $j = \text{pred}(0) = 1$ (ซึ่งได้อ้างถึงในเมตริกซ์ $I_E(u)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 1 ของ X ดังนี้

$$X = 00\underline{1}1(3) \text{ ดังนั้น } Y = 0001(1)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปใน R_1 ดังนั้น $R_1 = R_1 \cup \{(3 \rightarrow 1)\}$

ถ้า $X \in \{010\underline{1}(5), 011\underline{1}(7)\}$ ดังนั้น $j = \text{pred}(0) = 2$ (ซึ่งได้อ้างถึงในเมตริกซ์ $I_E(v)$ และ $I_E(s)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 2 ของ X ดังนี้

$$X = 0\underline{1}01(5) \text{ ดังนั้น } Y = 0001(1)$$

$$X = 0\underline{1}11(7) \text{ ดังนั้น } Y = 0011(3)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปใน R_2 ดังนั้น $R_2 = R_2 \cup \{(5 \rightarrow 1), (7 \rightarrow 3)\}$

ถ้า $X \in \{100\underline{1}(9), 101\underline{1}(11), 110\underline{1}(13), 111\underline{1}(15)\}$ ดังนั้น $j = \text{pred}(0) = 3$ (ซึ่งได้อ้างถึงในเมตริกซ์ $I_E(c)$, $I_E(b)$, $I_E(c)$ และ $I_E(a)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 3 ของ X ดังนี้

$$X = \underline{1}001(9) \text{ ดังนั้น } Y = 0001(1)$$

$$X = \underline{1}011(11) \text{ ดังนั้น } Y = 0011(3)$$

$$X = \underline{1}101(13) \text{ ดังนั้น } Y = 0101(5)$$

$$X = \underline{1}111(15) \text{ ดังนั้น } Y = 0111(7)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปใน R_3 ดังนี้

$$R_3 = R_3 \cup \{(9 \rightarrow 1), (11 \rightarrow 3), (13 \rightarrow 5), (15 \rightarrow 7)\}$$

สรุปสำหรับ $i = 0$ จะสามารถสร้างฟังก์ชันเส้นทาง (บางส่วน) สำหรับ T_0 ได้ดังต่อไปนี้

$$R_0 = \{(1 \rightarrow 0), (2 \rightarrow 3), (4 \rightarrow 5), (6 \rightarrow 7), (8 \rightarrow 9), (10 \rightarrow 11), \\ (12 \rightarrow 13), (14 \rightarrow 15)\}$$

$$R_1 = \{(3 \rightarrow 1)\}$$

$$R_2 = \{(5 \rightarrow 1), (7 \rightarrow 3)\}$$

$$R_3 = \{(9 \rightarrow 1), (11 \rightarrow 3), (13 \rightarrow 5), (15 \rightarrow 7)\}$$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 3.4 แสดงบางส่วนของฟังก์ชันเส้นทาง (R_0, R_1, R_2, R_3) สำหรับ T_0 ($i = 0$) เมื่อ $k = 3$

x	Binary string	$pred(i)$	$E_{pred(i)}$	$parent(x)$	R_i	$R_{pred(i)}$
1	0001	0	0	0	-	0
2	0010	-	-	3	0	-
3	0011	1	1	1	-	1
4	0100	-	-	5	0	-
5	0101	2	1	1	-	2
6	0110	-	-	7	0	-
7	0111	2	3	3	-	2
8	1000	-	-	9	0	-
9	1001	3	1	1	-	3
10	1010	-	-	11	0	-
11	1011	3	3	3	-	3
12	1100	-	-	13	0	-
13	1101	3	5	5	-	3
14	1110	-	-	15	0	-
15	1111	3	7	7	-	3

ในการทำซ้ำรอบที่ 2 ตารางที่ 3.5 แสดงบางส่วนของฟังก์ชันการหา R_0, R_1, R_2 และ R_3 สำหรับสร้าง T_i ทุกๆ $N-1$ หน่วยประมวลผล หน่วยประมวลผล จำนวนจากขั้นตอนวิธีในส่วนแรกเมื่อ $i = 1$ ดังนี้

กรณีที่ 1: ถ้า $b_1 = 0$ สมาชิกของ X ทั้งหมดที่เป็นไปได้คือ 0001(1), 0100(4), 0101(5), 1000(8), 1001(9), 1100(12), 1101(13) ดังนั้น

$$Y = (b_{k-1}b_{k-2} \dots b_i \dots b_2 \bar{b}_1 b_0)$$

ที่ซึ่งจะสามารถค้นหา Y ได้จากการสลับบิตตำแหน่งที่ 1 ของ X คือ 0011(3), 0110(6), 0111(7), 1010(10), 1011(11), 1110(14), 1111(15) และยูเนียน($X \rightarrow Y$) เข้าไปในฟังก์ชัน R_1 ดังนี้ $R_1 = R_0 \cup \{(1 \rightarrow 3), (4 \rightarrow 6), (5 \rightarrow 7), (8 \rightarrow 10), (9 \rightarrow 11), (12 \rightarrow 14), (13 \rightarrow 15)\}$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กรณีที่ 2: ถ้า $b_1 = 1$ สมาชิกของ X ที่เป็นไปได้ทั้งหมดคือ 00/0(2), 00/1(3), 01/0(6), 01/1(7), 10/0(10), 10/1(11), 11/0(14), 11/1(15) ดังนั้น

$$Y = (b_{k-1}b_{k-2} \dots \bar{b}_j \dots b_2b_1b_0)$$

จะสามารถหา Y ได้ดังต่อไปนี้

ถ้า $X \in \{00/1(3), 01/1(7), 10/1(11), 11/1(15)\}$ ดังนั้น $j = \text{pred}(1) = 0$ (ซึ่งได้อ้างอิงถึงในเมตริกซ์ $I_E(u)$, $I_E(s)$, $I_E(b)$ และ $I_E(a)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 0 ของ X ดังนี้

$$X = 001\underline{1}(3) \text{ ดังนั้น } Y = 0010(2)$$

$$X = 011\underline{1}(7) \text{ ดังนั้น } Y = 0110(6)$$

$$X = 101\underline{1}(11) \text{ ดังนั้น } Y = 1010(10)$$

$$X = 111\underline{1}(15) \text{ ดังนั้น } Y = 1110(14)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_0 ดังนี้

$$R_0 = R_0 \cup \{(3 \rightarrow 2), (7 \rightarrow 6), (11 \rightarrow 10), (15 \rightarrow 14)\}$$

ถ้า $X = 00/0(2)$ ดังนั้น $j = \text{pred}(1) = 1$ (ซึ่งได้อ้างอิงถึงในเมตริกซ์ $I_E(y)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 1 ของ X ดังนี้

$$X = 00\underline{0}(2) \text{ ดังนั้น } Y = 0000(0)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_1 ดังนี้ $R_1 = R_1 \cup \{(2 \rightarrow 0)\}$

ถ้า $X = 01/0(6)$ ดังนั้น $j = \text{pred}(1) = 2$ (ซึ่งได้อ้างอิงถึงในเมตริกซ์ $I_E(w)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 2 ของ X ดังนี้

$$X = 0\underline{1}0(6) \text{ ดังนั้น } Y = 0010(2)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปใน R_2 ดังนี้ $R_2 = R_2 \cup \{(6 \rightarrow 2)\}$

ถ้า $X \in \{10/0(10), 11/0(14)\}$ ดังนั้น $j = \text{pred}(1) = 3$ (ซึ่งได้อ้างอิงถึงในเมตริกซ์ $I_E(f)$ และ $I_E(d)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 3 ของ x ดังนี้

$$X = \underline{1}010(10) \text{ ดังนั้น } Y = 0010(2)$$

$$X = \underline{1}110(14) \text{ ดังนั้น } Y = 0110(6)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_3 ดังนี้ นำไปใช้ประโยชน์ด้านการคำนวณ

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งยังมีให้คัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

$$R_3 = R_3 \cup \{(10 \rightarrow 2), (14 \rightarrow 6)\}$$

ที่ซึ่งจะสามารถค้นหา Y ได้จากการสลับบิตตำแหน่งที่ 2 ของ X คือ 0101(5), 0110(6), 0111(7), 1100(12), 1101(13), 1110(14), 1111(15) และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_2 ดังนั้น $R_2 = R_2 \cup \{(1 \rightarrow 5), (2 \rightarrow 6), (3 \rightarrow 7), (8 \rightarrow 12), (9 \rightarrow 13), (10 \rightarrow 14), (11 \rightarrow 15)\}$

กรณีที่ 2: ถ้า $b_j = 1$ สมาชิกของ X ที่เป็นไปได้ทั้งหมดคือ 0100(4), 0101(5), 0110(6), 0111(7), 1100(12), 1101(13), 1110(14), 1111(15); ดังนั้น

$$Y = (b_{k-1} b_{k-2} \dots \bar{b}_j \dots b_2 b_1 b_0)$$

จะสามารถหา Y ได้ดังต่อไปนี้

ถ้า $X \in \{0101(5), 1101(13)\}$ ดังนั้น $j = \text{pred}(2) = 0$ (ซึ่งได้อ้างอิงถึงในเมตริกซ์ $I_E(v)$ และ $I_E(c)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 0 ของ X ดังนี้

$$X = 0101(5) \text{ ดังนั้น } Y = 0100(4)$$

$$X = 1101(13) \text{ ดังนั้น } Y = 1100(12)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_0 ดังนี้

$$R_0 = R_0 \cup \{(5 \rightarrow 4), (13 \rightarrow 12)\}$$

ถ้า $X \in \{0110(6), 0111(7), 1110(14), 1111(15)\}$ ดังนั้น $j = \text{pred}(2) = 1$ (ซึ่งได้อ้างอิงถึงในเมตริกซ์ $I_E(w)$, $I_E(s)$, $I_E(d)$ และ $I_E(a)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 1 ของ X ดังนี้

$$X = 0110(6) \text{ ดังนั้น } Y = 0100(4)$$

$$X = 0111(7) \text{ ดังนั้น } Y = 0101(5)$$

$$X = 1110(14) \text{ ดังนั้น } Y = 1100(12)$$

$$X = 1111(15) \text{ ดังนั้น } Y = 1101(13)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_1 ดังนี้

$$R_1 = R_1 \cup \{(6 \rightarrow 4), (7 \rightarrow 5), (14 \rightarrow 12), (15 \rightarrow 13)\}$$

ถ้า $X = 0100(4)$ ดังนั้น $j = \text{pred}(2) = 2$ (ซึ่งได้อ้างอิงถึงในเมตริกซ์ $I_E(z)$) ที่นี้ เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่ควรเผยแพร่ไปใช้ประโยชน์ด้านการค้า Y ก็คือการสลับที่บิตตำแหน่งที่ 2 ของ X ดังนี้
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้
 $X = 0100(4) \text{ ดังนั้น } Y = 0000(0)$

และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_2 ดังนี้ $R_2 = R_2 \cup \{(4 \rightarrow 0)\}$

ถ้า $X = 1100(12)$ ดังนั้น $j = \text{pred}(2) = 3$ (ซึ่งได้อ้างถึงในเมตริกซ์ $I_E(g)$)
 ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 3 ของ X ดังนี้

$$X = 1100(12) \text{ ดังนั้น } Y = 0100(4)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_3 ดังนี้ $R_3 = R_3 \cup \{(12 \rightarrow 4)\}$

สรุปสำหรับ $i = 2$ จะสามารถสร้างฟังก์ชันเส้นทาง (บางส่วน) สำหรับ T_2 ได้ดังต่อไปนี้

$$R_0 = R_0 \text{ สำหรับ } T_0 \text{ และ } T_1 \cup \{(5 \rightarrow 4), (13 \rightarrow 12)\}$$

$$R_1 = R_1 \text{ สำหรับ } T_0 \text{ และ } T_1 \cup \{(6 \rightarrow 4), (7 \rightarrow 5), (14 \rightarrow 12), \\ (15 \rightarrow 13)\}$$

$$R_2 = R_2 \text{ สำหรับ } T_0 \text{ และ } T_1 \cup \{(1 \rightarrow 5), (2 \rightarrow 6), (3 \rightarrow 7), (4 \rightarrow 0), \\ (8 \rightarrow 12), (9 \rightarrow 13), (10 \rightarrow 14), (11 \rightarrow 15)\}$$

$$R_3 = R_3 \text{ สำหรับ } T_0 \text{ และ } T_1 \cup \{(12 \rightarrow 4)\}$$

ตารางที่ 3.6 แสดงบางส่วนของฟังก์ชันเส้นทาง (R_0, R_1, R_2, R_3) สำหรับ T_2 ($i = 2$) เมื่อ $k = 4$

x	Binary string	$\text{pred}(i)$	$E_{\text{pred}(i)}$	$\text{parent}(x)$	R_i	$R_{\text{pred}(i)}$
1	0001	-	-	5	2	-
2	0010	-	-	6	2	-
3	0011	-	-	7	2	-
4	0100	2	0	0	-	2
5	0101	0	4	4	-	0
6	0110	1	4	4	-	1
7	0111	1	5	5	-	1
8	1000	-	-	12	2	-
9	1001	-	-	13	2	-
10	1010	-	-	14	2	-
11	1011	-	-	15	2	-
12	1100	3	4	4	-	3
13	1101	0	12	12	-	0

เอกสาร 14 เป็นเอกสาร 1/110 งวนไว้สำหรับภา 1 ใช้งานเพื่อกา 12 กษาเท่านั้น ไม่ 12 อนุญาตให้นำไปใช้ประโยชน์ 1 ในการค้า
 ไม่ว่าการ 15 ตาทั้งสิน 1/111 ห้ามนำให้ดัดแ 1 ลงเนื้อหา และ 13 ้องอ้างอิงถึงเจ้า 13 งเอกสารทุกครั้งที่มีการนำ 1 ไปใช้

ในการทำซ้ำรอบที่ 4 ตารางที่ 3.7 แสดงบางส่วนของฟังก์ชันเส้นทาง R_0 , R_1 , R_2 และ R_3 สำหรับสร้าง T_3 ทุกๆ $N-1$ หน่วยประมวลผลหน่วยประมวลผล จำนวนจากขั้นตอนวิธีในส่วนแรกเมื่อ $i = 3$ ดังนี้

กรณีที่ 1: ถ้า $b_3 = 0$ สมาชิกของ X ทั้งหมดที่เป็นไปได้คือ 0001(1), 0010(2), 0011(3), 0100(4), 0101(5), 0110(6), 0111(7) ดังนั้น

$$Y = (b_{k-1}b_{k-2} \dots b_i \dots \bar{b}_3 b_2 b_1 b_0)$$

ที่จะสามารถค้นหา Y ได้จากการสลับบิตตำแหน่งที่ 3 ของ X คือ 1001(9), 1010(10), 1011(11), 1100(12), 1101(13), 1110(14), 1111(15) และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_3 ดังนี้ $R_3 = R_3 \cup \{(1 \rightarrow 9), (2 \rightarrow 10), (3 \rightarrow 11), (4 \rightarrow 12), (5 \rightarrow 13), (6 \rightarrow 14), (7 \rightarrow 15)\}$

กรณีที่ 2: ถ้า $b_3 = 1$ สมาชิกของ X ที่เป็นไปได้ทั้งหมดคือ 1000(8), 1001(9), 1010(10), 1011(11), 1100(12), 1101(13), 1110(14), 1111(15) ดังนั้น

$$Y = (b_{k-1}b_{k-2} \dots \bar{b}_3 \dots b_2 b_1 b_0)$$

จะสามารถหา Y ได้ดังต่อไปนี้

ถ้า $X = 1001(9)$ ดังนั้น $j = \text{pred}(3) = 0$ (ซึ่งได้อ้างถึงในเมตริกซ์ $I_E(c)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 0 ของ X ดังนี้

$$X = 100\bar{1}(9) \text{ ดังนั้น } Y = 1000(8)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_0 ดังนี้ $R_0 = R_0 \cup \{(9 \rightarrow 8)\}$

ถ้า $X \in \{1010(10), 1011(11)\}$ ดังนั้น $j = \text{pred}(3) = 1$ (ซึ่งได้อ้างถึงในเมตริกซ์ $I_E(f)$ และ $I_E(b)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 1 ของ X ดังนี้

$$X = 10\bar{1}0(10) \text{ ดังนั้น } Y = 1000(8)$$

$$X = 10\bar{1}1(11) \text{ ดังนั้น } Y = 1001(9)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_1 ดังนี้

$$R_1 = R_1 \cup \{(10 \rightarrow 8), (11 \rightarrow 9)\}$$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ถ้า $X \in \{1010(12), 1101(13), 1110(14), 1111(15)\}$ ดังนั้น $j = \text{pred}(3) = 2$ (ซึ่งได้อ้างถึงในเมตริกซ์ $I_E(g)$, $I_E(c)$, $I_E(d)$ และ $I_E(a)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 2 ของ X ดังนี้

$$X = 1010(12) \text{ ดังนั้น } Y = 1000(8)$$

$$X = 1101(13) \text{ ดังนั้น } Y = 1001(9)$$

$$X = 1110(14) \text{ ดังนั้น } Y = 1010(10)$$

$$X = 1111(15) \text{ ดังนั้น } Y = 1011(11)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_2 ดังนี้

$$R_2 = R_2 \cup \{(12 \rightarrow 8), (13 \rightarrow 9), (14 \rightarrow 10), (15 \rightarrow 11)\}$$

ถ้า $X = 1000(8)$ ดังนั้น $j = \text{pred}(3) = 3$ (ซึ่งได้อ้างถึงในเมตริกซ์ $I_E(h)$) ที่นี้ Y ก็คือการสลับที่บิตตำแหน่งที่ 3 ของ X ดังนี้

$$X = 1000(8) \text{ ดังนั้น } Y = 0000(0)$$

และยูเนียน $(X \rightarrow Y)$ เข้าไปในฟังก์ชัน R_3 ดังนี้ $R_3 = R_3 \cup \{(8 \rightarrow 0)\}$

สรุปสำหรับ $i = 3$ จะสามารถสร้างฟังก์ชันเส้นทาง (บางส่วน) สำหรับ T_3 ได้ดังต่อไปนี้

$$R_0 = R_0 \text{ สำหรับ } T_0, T_1 \text{ และ } T_2 \cup \{(9 \rightarrow 8)\}$$

$$R_1 = R_1 \text{ สำหรับ } T_0, T_1 \text{ และ } T_2 \cup \{(10 \rightarrow 8), (11 \rightarrow 9)\}$$

$$R_2 = R_2 \text{ สำหรับ } T_0, T_1 \text{ และ } T_2 \cup \{(12 \rightarrow 8), (13 \rightarrow 9), (14 \rightarrow 10), (15 \rightarrow 11)\}$$

$$R_3 = R_3 \text{ สำหรับ } T_0, T_1 \text{ และ } T_2 \cup \{(1 \rightarrow 9), (2 \rightarrow 10), (3 \rightarrow 11), (4 \rightarrow 12), (5 \rightarrow 13), (6 \rightarrow 14), (7 \rightarrow 15), (8 \rightarrow 0)\}$$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 3.7 แสดงบางส่วนของฟังก์ชันเส้นทาง (R_0, R_1, R_2, R_3) สำหรับ T_3 ($i = 3$) เมื่อ $k = 4$

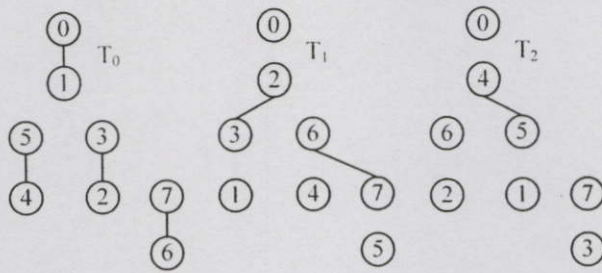
x	Binary string	$pred(i)$	$E_{pred(i)}$	$parent(x)$	R_i	$R_{pred(i)}$
1	0001	-	-	9	3	-
2	0010	-	-	10	3	-
3	0011	-	-	11	3	-
4	0100	-	-	12	3	-
5	0101	-	-	13	3	-
6	0110	-	-	14	3	-
7	0111	-	-	15	3	-
8	1000	3	0	0	-	3
9	1001	0	8	8	-	0
10	1010	1	8	8	-	1
11	1011	1	9	9	-	1
12	1100	2	8	8	-	2
13	1101	2	9	9	-	2
14	1110	2	10	10	-	2
15	1111	2	11	11	-	2

3.3.2 การสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนานโดยใช้ฟังก์ชันเส้นทางบนเครือข่ายแบบมัลติสเตจ

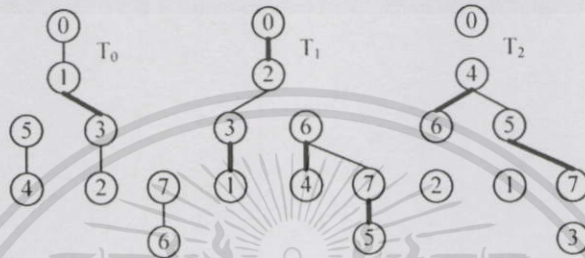
ในขั้นตอนที่สองนี้จะแสดงการสร้างเส้นเชื่อมต่อด้านไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนานโดยใช้ฟังก์ชันเส้นทางจากขั้นแรกจนกระทั่งได้ต้นไม้แบบทอดข้ามที่แตกต่างจนครบทุกๆ ต้น ดังจะแสดงในตัวอย่างต่อไปนี้

ตัวอย่างที่ 3.5 แสดงการสร้างเส้นเชื่อมของต้นไม้แบบทอดข้ามที่แตกต่าง (T_0, T_1, T_2) บนเครือข่ายแบบมัลติสเตจดังรูปที่ 3.4 โดยใช้ฟังก์ชันเส้นทาง ซึ่งที่นี่มีขนาดหน่วยประมวลผล (โหนด) ทั้งหมด 8 หน่วยประมวลผล ($N = 8$) และเลขฐาน 2 (b_1, b_0) แทนจำนวน 3 บิต ($k = 3$) ของโหนดใดๆ ดังนั้นเมื่อใช้ฟังก์ชันเส้นทาง (R_0, R_1, R_2) จนครบทุกๆ ฟังก์ชันก็จะสามารถสร้างต้นไม้แบบทอดข้ามที่แตกต่างได้จนครบทุกๆ ต้นเช่นกัน

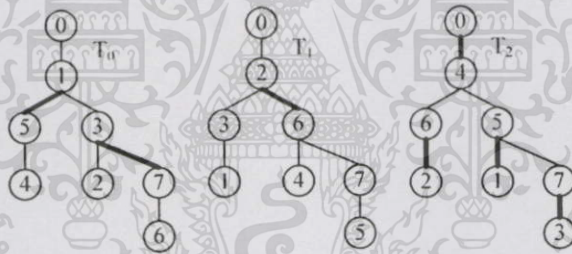
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปเผยแพร่บนสื่อออนไลน์โดยไม่ได้รับอนุญาตจากเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



(ก) รอบที่ 1 ของฟังก์ชันเส้นทาง $R_0 = \{(1 \rightarrow 0), (2 \leftrightarrow 3), (4 \leftrightarrow 5), (6 \leftrightarrow 7)\}$



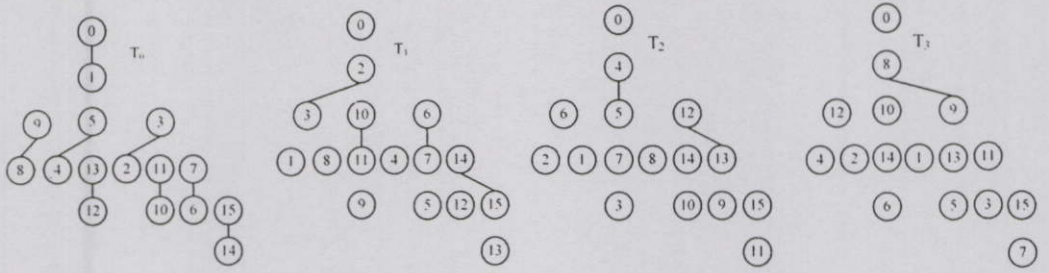
(ข) รอบที่ 2 ของฟังก์ชันเส้นทาง $R_1 = \{(2 \rightarrow 0), (1 \leftrightarrow 3), (4 \leftrightarrow 6), (5 \leftrightarrow 7)\}$



(ค) รอบที่ 3 ของฟังก์ชันเส้นทาง $R_2 = \{(4 \rightarrow 0), (1 \leftrightarrow 5), (2 \leftrightarrow 6), (3 \leftrightarrow 7)\}$

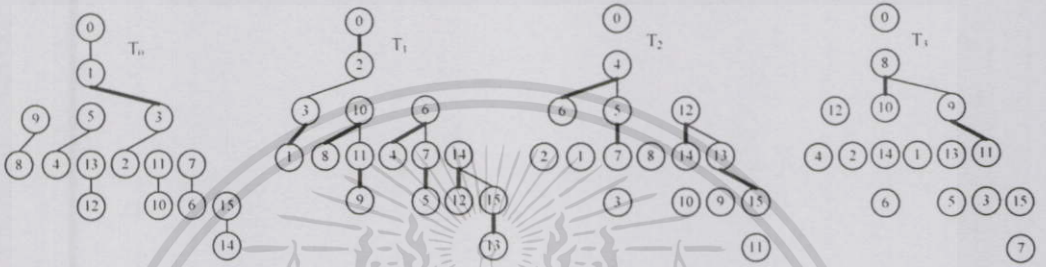
รูปที่ 3.4 แสดงฟังก์ชันเส้นทาง 3 ฟังก์ชันสำหรับสร้างต้นไม้แบบทอดข้ามที่แตกต่างบนเครือข่ายแบบมัลติสเตจ ($N = 8$)

ตัวอย่างที่ 3.6 แสดงการสร้างเส้นเชื่อมของต้นไม้แบบทอดข้ามที่แตกต่าง (T_0, T_1, T_2, T_3) บนเครือข่ายแบบมัลติสเตจ ดังรูปที่ 3.5 โดยใช้ฟังก์ชันเส้นทาง ซึ่งที่นี่มีขนาดหน่วยประมวลผล (โหนด) ทั้งหมด 16 หน่วยประมวลผล ($N = 16$) และเลขฐาน 2 (b_3, b_2, b_1, b_0) แทนจำนวน 4 บิต ($k = 4$) ของโหนดใดๆ ดังนั้นเมื่อใช้ฟังก์ชันเส้นทาง (R_0, R_1, R_2, R_3) จนครบทุกๆ ฟังก์ชันก็จะได้นี้ต้นไม้แบบทอดข้ามที่แตกต่างครบทุกๆ ต้นเช่นกัน การศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



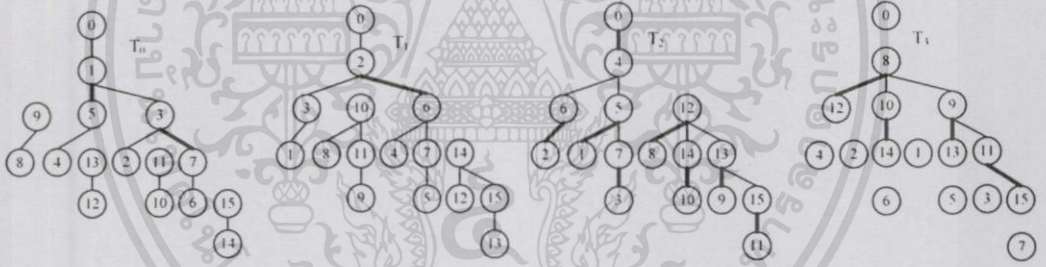
(ก) รอบที่ 1 ของฟังก์ชันเส้นทาง

$$R_0 = \{(1 \rightarrow 0), (2 \leftrightarrow 3), (4 \leftrightarrow 5), (6 \leftrightarrow 7), (8 \rightarrow 9), (10 \leftrightarrow 11), (12 \leftrightarrow 13), (14 \leftrightarrow 15)\}$$



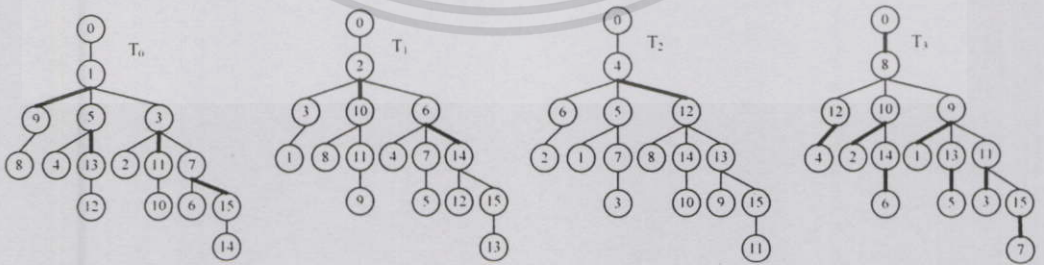
(ข) รอบที่ 2 ของฟังก์ชันเส้นทาง

$$R_1 = \{(2 \rightarrow 0), (1 \leftrightarrow 3), (4 \leftrightarrow 6), (5 \leftrightarrow 7), (8 \rightarrow 10), (9 \leftrightarrow 11), (12 \leftrightarrow 14), (13 \leftrightarrow 15)\}$$



(ค) รอบที่ 3 ของฟังก์ชันเส้นทาง

$$R_2 = \{(4 \rightarrow 0), (1 \leftrightarrow 5), (2 \leftrightarrow 6), (3 \leftrightarrow 7), (8 \rightarrow 12), (9 \leftrightarrow 13), (10 \leftrightarrow 14), (11 \leftrightarrow 15)\}$$



(ง) รอบที่ 4 ของฟังก์ชันเส้นทาง

$$R_3 = \{(8 \rightarrow 0), (1 \leftrightarrow 9), (2 \leftrightarrow 10), (3 \leftrightarrow 11), (4 \rightarrow 12), (5 \leftrightarrow 13), (6 \leftrightarrow 14), (7 \leftrightarrow 15)\}$$

เอกสารนี้เผยแพร่ภายใต้สัญญาอนุญาตแบบเปิด (CC-BY) โดยสำนักงานส่งเสริมการค้าในต่างประเทศ ณ นครเชียงใหม่

ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 3.5 แสดงฟังก์ชันเส้นทาง 4 ฟังก์ชันสำหรับสร้างต้นไม้แบบทอดข้ามที่แตกต่างกัน

บนเครือข่ายแบบมัลติสแตจ (N = 16)

3.4 การประยุกต์ใช้การสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนานโดยใช้ฟังก์ชันเส้นทางบนเครือข่ายแบบมัลติสเตจ

ในหัวข้อนี้คือ การประยุกต์ใช้ขั้นตอนวิธีการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนานบนเครือข่ายแบบมัลติสเตจ ซึ่งจะสามารถสร้างได้บน 3 เครือข่ายคือ เครือข่ายแบบคิว เครือข่ายแบบโอเมก้า และเครือข่ายแบบฟลิป

3.4.1 การสร้างต้นไม้แบบทอดข้ามที่แตกต่างบนเครือข่ายแบบคิว

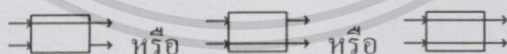
ในส่วนนี้จะแสดงการสร้างต้นไม้แบบทอดข้ามที่แตกต่างจำนวน k ต้น บนเครือข่ายแบบคิว ซึ่งจะสามารถแสดงได้ดังตัวอย่างดังต่อไปนี้

ตัวอย่างที่ 3.7 ตัวอย่างนี้จะแสดงที่ขนาดของหน่วยประมวลผลคือ 8 หน่วยประมวลผล ($N = 8$) และจะใช้แทนเลขฐาน 2 ได้ 3 บิต ($k = 3$) ซึ่งมีการอธิบายการสร้างบางส่วนของทุกๆ ต้นไม้แบบทอดข้าม (T_0, T_1, T_2) ในรูปที่ 3.4 โดยใช้ฟังก์ชันเส้นทาง (R_0, R_1, R_2) ที่แสดงในรูปที่ 3.6

ในรอบที่ 1 (รูปที่ 3.6 (ก)) จะสร้างบางส่วนของ (T_0, T_1, T_2) จากฟังก์ชันเส้นทาง $R_0 = \{(1 \rightarrow 0), (2 \leftrightarrow 3), (4 \leftrightarrow 5), (6 \leftrightarrow 7)\}$ โดยการปรับทุกสวิตช์ในสเตจ S_0 แบบไขว้ (Exchange)

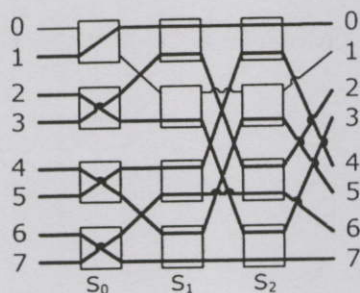


และปรับทุกสวิตช์ในสเตจ $S_1 + S_2$ แบบตรง (Straight)

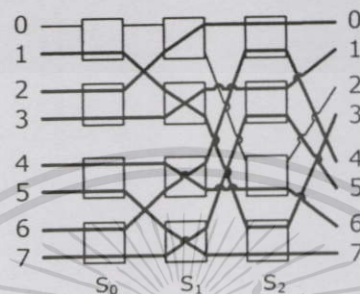


ในรอบที่ 2 (รูปที่ 3.6 (ข)) จะสร้างบางส่วนของ (T_0, T_1, T_2) จากฟังก์ชันเส้นทาง $R_1 = \{(2 \rightarrow 0), (1 \leftrightarrow 3), (4 \leftrightarrow 6), (5 \leftrightarrow 7)\}$ โดยการปรับทุกสวิตช์ในสเตจ S_1 แบบไขว้ และปรับทุกสวิตช์ในสเตจ $S_0 + S_2$ แบบตรง

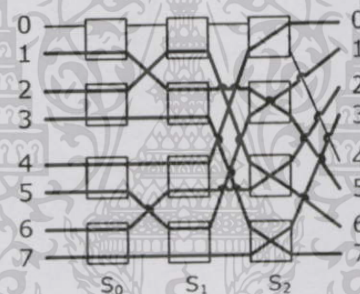
ในรอบที่ 3 (รูปที่ 3.6 (ค)) จะสร้างบางส่วนของ (T_0, T_1, T_2) จากฟังก์ชันเส้นทาง $R_2 = \{(4 \rightarrow 0), (1 \leftrightarrow 5), (2 \leftrightarrow 6), (3 \leftrightarrow 7)\}$ โดยการปรับทุกสวิตช์ในสเตจ S_1 แบบไขว้ และปรับทุกสวิตช์ในสเตจ $S_0 + S_1$ แบบตรง



(ก) รอบที่ 1 ของฟังก์ชันเส้นทาง $R_0 = \{(1 \rightarrow 0), (2 \leftrightarrow 3), (4 \leftrightarrow 5), (6 \leftrightarrow 7)\}$



(ข) รอบที่ 2 ของฟังก์ชันเส้นทาง $R_1 = \{(2 \rightarrow 0), (1 \leftrightarrow 3), (4 \leftrightarrow 6), (5 \leftrightarrow 7)\}$



(ค) รอบที่ 3 ของฟังก์ชันเส้นทาง $R_2 = \{(4 \rightarrow 0), (1 \leftrightarrow 5), (2 \leftrightarrow 6), (3 \leftrightarrow 7)\}$

รูปที่ 3.6 แสดงฟังก์ชันเส้นทาง 3 ฟังก์ชัน สำหรับสร้างต้นไม้แบบทอดข้าม บนเครือข่ายแบบคิว ($N = 8$)

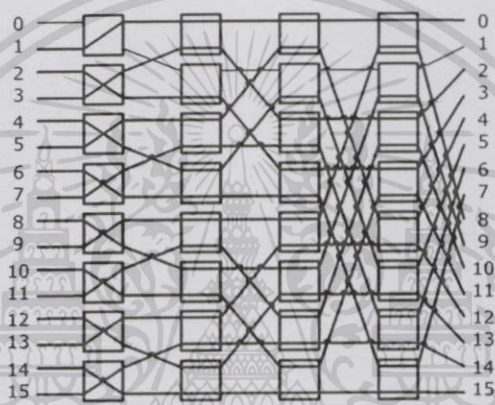
ตัวอย่างที่ 3.8 ตัวอย่างนี้จะแสดงที่ขนาดของหน่วยประมวลผล 16 หน่วยประมวลผลบนเครือข่ายแบบมัลติสเตจ ($N = 16$) และจะใช้แทนเลขฐาน 2 ได้ 4 บิต ($k = 4$) ซึ่งมีการอธิบายการสร้างบางส่วนของทุกๆ ต้นไม้แบบทอดข้าม (T_0, T_1, T_2, T_3) ในรูปที่ 3.5 โดยใช้ฟังก์ชันเส้นทาง (R_0, R_1, R_2, R_3) ที่แสดงในรูปที่ 3.7

ในรูปที่ 3.7 (ก) จะสร้างบางส่วนของ (T_0, T_1, T_2, T_3) จากฟังก์ชันเส้นทาง $R_0 = \{(1 \rightarrow 0), (2 \leftrightarrow 3), (4 \leftrightarrow 5), (6 \leftrightarrow 7), (8 \rightarrow 9), (10 \leftrightarrow 11), (12 \leftrightarrow 13), (14 \leftrightarrow 15)\}$ โดยการปรับทุกสวิตช์ในสเตจ S_0 แบบไขว้ (Exchange) และปรับทุกสวิตช์ในสเตจ $S_1 + S_2 + S_3$ แบบตรง (Straight)

ในรอบที่ 2 (รูปที่ 3.7 (ข)) จะสร้างบางส่วนของ (T_0, T_1, T_2, T_3) จากฟังก์ชันเส้นทาง $R_1 = \{(2 \rightarrow 0), (1 \leftrightarrow 3), (4 \leftrightarrow 6), (5 \leftrightarrow 7), (8 \rightarrow 10), (9 \leftrightarrow 11), (12 \leftrightarrow 14), (13 \leftrightarrow 15)\}$ โดยการปรับทุกสวิตช์ในสแตจ S_1 แบบไขว้ และปรับทุกสวิตช์ในสแตจ $S_0 + S_2 + S_3$ แบบตรง

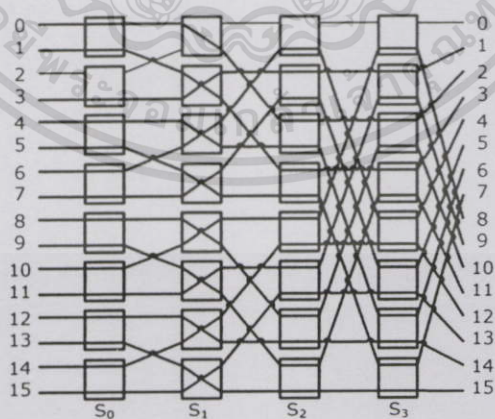
ในรอบที่ 3 (รูปที่ 3.7 (ค)) จะสร้างบางส่วนของ (T_0, T_1, T_2, T_3) จากฟังก์ชันเส้นทาง $R_2 = \{(4 \rightarrow 0), (1 \leftrightarrow 5), (2 \leftrightarrow 6), (3 \leftrightarrow 7), (8 \rightarrow 12), (9 \leftrightarrow 13), (10 \leftrightarrow 14), (11 \leftrightarrow 15)\}$ โดยการปรับทุกสวิตช์ในสแตจ S_2 แบบไขว้ และปรับทุกสวิตช์ในสแตจ $S_0 + S_1 + S_3$ แบบตรง

ในรอบที่ 4 (รูปที่ 3.7 (ง)) จะสร้างบางส่วนของ (T_0, T_1, T_2, T_3) จากฟังก์ชันเส้นทาง $R_3 = \{(8 \rightarrow 0), (1 \leftrightarrow 9), (2 \leftrightarrow 10), (3 \leftrightarrow 11), (4 \rightarrow 12), (5 \leftrightarrow 13), (6 \leftrightarrow 14), (7 \leftrightarrow 15)\}$ โดยการปรับทุกสวิตช์ในสแตจ S_3 แบบไขว้ และปรับทุกสวิตช์ในสแตจ $S_0 + S_1 + S_2$ แบบตรง



(ก) รอบที่ 1 ของฟังก์ชันเส้นทาง

$$R_0 = \{(1 \rightarrow 0), (2 \leftrightarrow 3), (4 \leftrightarrow 5), (6 \leftrightarrow 7), (8 \rightarrow 9), (10 \leftrightarrow 11), (12 \leftrightarrow 13), (14 \leftrightarrow 15)\}$$



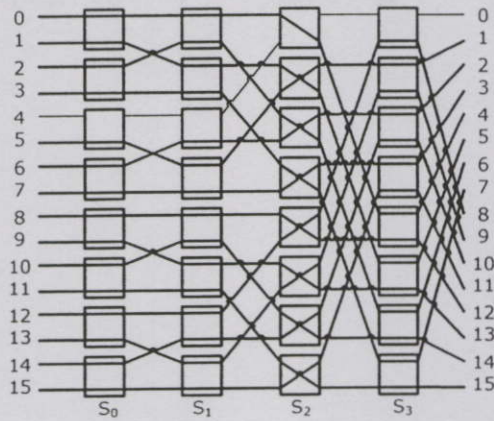
(ข) รอบที่ 2 ของฟังก์ชันเส้นทาง

$$R_1 = \{(2 \rightarrow 0), (1 \leftrightarrow 3), (4 \leftrightarrow 6), (5 \leftrightarrow 7), (8 \rightarrow 10), (9 \leftrightarrow 11), (12 \leftrightarrow 14), (13 \leftrightarrow 15)\}$$

ไม่ว่ากรณีใดๆ ที่เส้น อีกทั้งยังมีให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

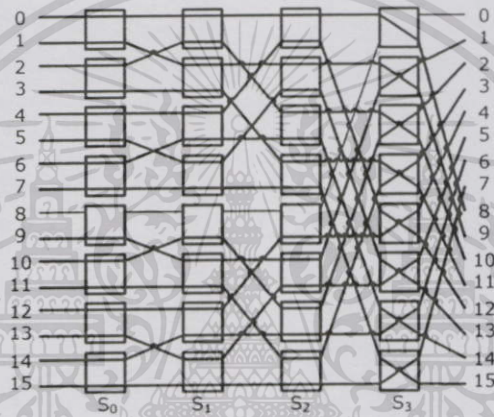
รูปที่ 3.7 แสดงฟังก์ชันเส้นทาง 4 ฟังก์ชัน สำหรับสร้างต้นไม้แบบทอดข้ามที่แตกต่างกัน

บนเครือข่ายแบบกิว ($N = 16$)



(ค) รอบที่ 3 ของฟังก์ชันเส้นทาง

$$R_2 = \{(4 \rightarrow 0), (1 \leftrightarrow 5), (2 \leftrightarrow 6), (3 \leftrightarrow 7), (8 \rightarrow 12), (9 \leftrightarrow 13), (10 \leftrightarrow 14), (11 \leftrightarrow 15)\}$$



(ง) รอบที่ 4 ของฟังก์ชันเส้นทาง $R_3 = \{(8 \rightarrow 0), (1 \leftrightarrow 9), (2 \leftrightarrow 10), (3 \leftrightarrow 11), (4 \rightarrow 12), (5 \leftrightarrow 13), (6 \leftrightarrow 14), (7 \leftrightarrow 15)\}$

รูปที่ 3.7 (ต่อ)

3.4.2 การสร้างต้นไม้แบบทอดข้ามที่แตกต่าง บนเครือข่ายแบบโอเมก้า

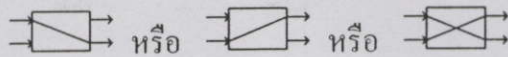
ในส่วนนี้จะแสดงการสร้างต้นไม้แบบทอดข้ามที่แตกต่างจำนวน k ต้น บนเครือข่ายแบบโอเมก้า ซึ่งจะสามารถแสดงได้ดังตัวอย่างดังต่อไปนี้

ตัวอย่างที่ 3.9 แสดงที่ขนาดของหน่วยประมวลผล 8 หน่วยประมวลผล ($N = 8$) และจะใช้แทนเลขฐาน 2 ได้ 3 บิต ($k = 3$) ซึ่งมีการอธิบายการสร้างบางส่วนของทุกๆ ต้นไม้แบบทอดข้าม (T_0, T_1, T_2) ในรูปที่ 3.4 โดยใช้ฟังก์ชันเส้นทาง (R_0, R_1, R_2) ที่แสดงในรูปที่ 3.8

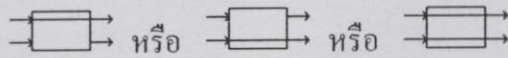
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ในรอบที่ 1 (รูปที่ 3.8 (ก)) จะสร้างบางส่วนของ (T_0, T_1, T_2) จากฟังก์ชันเส้นทาง ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

$R_0 = \{(1 \rightarrow 0), (2 \leftrightarrow 3), (4 \leftrightarrow 5), (6 \leftrightarrow 7)\}$ โดยการปรับทุกตัวในสเตจ S_2 แบบไขว้

(Exchange)

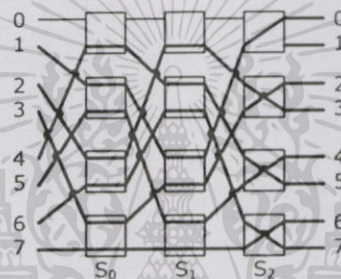


และปรับทุกสวิตช์ในสเตจ $S_0 + S_1$ แบบตรง (Straight)

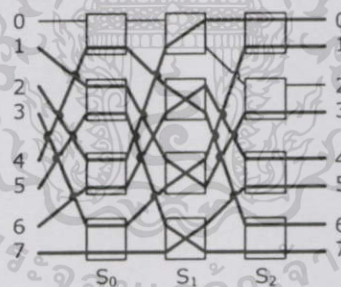


ในรอบที่ 2 (รูปที่ 3.8 (ข)) จะสร้างบางส่วนของ (T_0, T_1, T_2) จากฟังก์ชันเส้นทาง $R_1 = \{(2 \rightarrow 0), (1 \leftrightarrow 3), (4 \leftrightarrow 6), (5 \leftrightarrow 7)\}$ โดยการปรับทุกสวิตช์ในสเตจ S_1 แบบไขว้ และปรับทุกสวิตช์ในสเตจ $S_0 + S_2$ แบบตรง

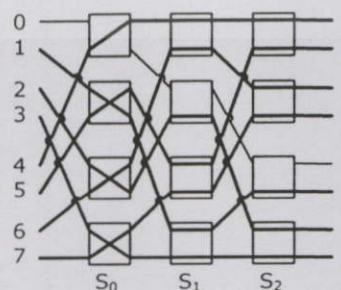
ในรอบที่ 3 (รูปที่ 3.8 (ค)) จะสร้างบางส่วนของ (T_0, T_1, T_2) จากฟังก์ชันเส้นทาง $R_2 = \{(4 \rightarrow 0), (1 \leftrightarrow 5), (2 \leftrightarrow 6), (3 \leftrightarrow 7)\}$ โดยการปรับทุกสวิตช์ในสเตจ S_0 แบบไขว้ และปรับทุกสวิตช์ในสเตจ $S_1 + S_2$ แบบตรง



(ก) รอบที่ 1 ของฟังก์ชันเส้นทาง $R_0 = \{(1 \rightarrow 0), (2 \leftrightarrow 3), (4 \leftrightarrow 5), (6 \leftrightarrow 7)\}$



(ข) รอบที่ 2 ของฟังก์ชันเส้นทาง $R_1 = \{(2 \rightarrow 0), (1 \leftrightarrow 3), (4 \leftrightarrow 6), (5 \leftrightarrow 7)\}$



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับใช้ในการเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
(ค) รอบที่ 3 ของฟังก์ชันเส้นทาง $R_2 = \{(4 \rightarrow 0), (1 \leftrightarrow 5), (2 \leftrightarrow 6), (3 \leftrightarrow 7)\}$
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งยังมีให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 3.8 แสดงฟังก์ชันเส้นทาง 3 ฟังก์ชัน สำหรับสร้างต้นไม้แบบทอดข้าม

บนเครือข่ายแบบโอเมก้า ($N = 8$)

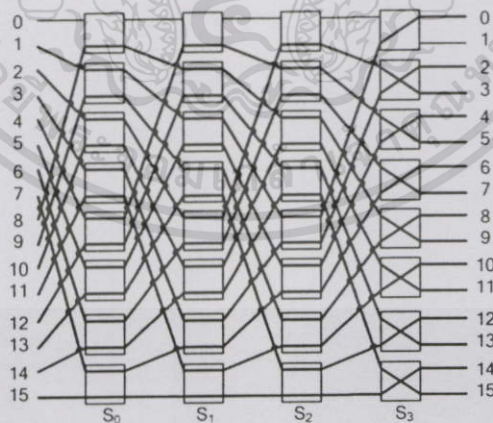
ตัวอย่างที่ 3.10 ตัวอย่างนี้จะแสดงที่ขนาดของหน่วยประมวลผล 16 หน่วยประมวลผล ($N = 16$) และจะใช้แทนเลขฐาน 2 ได้ 4 บิต ($k = 4$) ซึ่งมีการอธิบายการสร้างบางส่วนของทุกๆ ต้นไม้แบบทอดข้าม (T_0, T_1, T_2, T_3) ในรูปที่ 3.5 โดยใช้ฟังก์ชันเส้นทาง (R_0, R_1, R_2, R_3) ที่แสดงในรูปที่ 3.9

ในรอบที่ 1 (รูปที่ 3.9 (ก)) จะสร้างบางส่วนของ (T_0, T_1, T_2, T_3) จากฟังก์ชันเส้นทาง $R_0 = \{(1 \rightarrow 0), (2 \leftrightarrow 3), (4 \leftrightarrow 5), (6 \leftrightarrow 7), (8 \rightarrow 9), (10 \leftrightarrow 11), (12 \leftrightarrow 13), (14 \leftrightarrow 15)\}$ โดยการปรับทุกสวิตช์ในสเตจ S_1 แบบไขว้ (Exchange) และปรับทุกสวิตช์ในสเตจ $S_0 + S_1 + S_2$ แบบตรง (Straight)

ในรอบที่ 2 (รูปที่ 3.9 (ข)) จะสร้างบางส่วนของ (T_0, T_1, T_2, T_3) จากฟังก์ชันเส้นทาง $R_1 = \{(2 \rightarrow 0), (1 \leftrightarrow 3), (4 \leftrightarrow 6), (5 \leftrightarrow 7), (8 \rightarrow 10), (9 \leftrightarrow 11), (12 \leftrightarrow 14), (13 \leftrightarrow 15)\}$ โดยการปรับทุกสวิตช์ในสเตจ S_2 แบบไขว้ และปรับทุกสวิตช์ในสเตจ $S_0 + S_1 + S_3$ แบบตรง

ในรอบที่ 3 (รูปที่ 3.9 (ค)) จะสร้างบางส่วนของ (T_0, T_1, T_2, T_3) จากฟังก์ชันเส้นทาง $R_2 = \{(4 \rightarrow 0), (1 \leftrightarrow 5), (2 \leftrightarrow 6), (3 \leftrightarrow 7), (8 \rightarrow 12), (9 \leftrightarrow 13), (10 \leftrightarrow 14), (11 \leftrightarrow 15)\}$ โดยการปรับทุกสวิตช์ในสเตจ S_1 แบบไขว้ และปรับทุกสวิตช์ในสเตจ $S_0 + S_2 + S_3$ แบบตรง

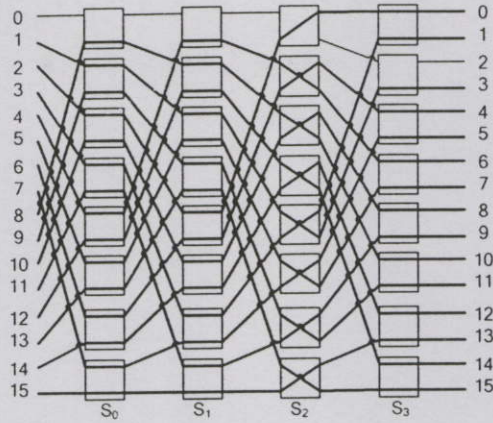
ในรอบที่ 4 (รูปที่ 3.9 (ง)) จะสร้างบางส่วนของ (T_0, T_1, T_2, T_3) จากฟังก์ชันเส้นทาง $R_3 = \{(8 \rightarrow 0), (1 \leftrightarrow 9), (2 \leftrightarrow 10), (3 \leftrightarrow 11), (4 \rightarrow 12), (5 \leftrightarrow 13), (6 \leftrightarrow 14), (7 \leftrightarrow 15)\}$ โดยการปรับทุกสวิตช์ในสเตจ S_0 แบบไขว้ และปรับทุกสวิตช์ในสเตจ $S_1 + S_2 + S_3$ แบบตรง



(ก) รอบที่ 1 ของฟังก์ชันเส้นทาง

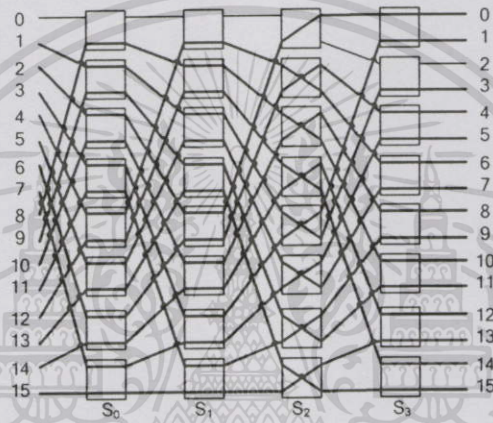
$$R_0 = \{(1 \rightarrow 0), (2 \leftrightarrow 3), (4 \leftrightarrow 5), (6 \leftrightarrow 7), (8 \rightarrow 9), (10 \leftrightarrow 11), (12 \leftrightarrow 13), (14 \leftrightarrow 15)\}$$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับใช้ภายในเพื่อการศึกษาเท่านั้น ไม่ควรเผยแพร่ไปยังสาธารณะโดยไม่ผ่านการคัดค้าน
รูปที่ 3.9 แสดงฟังก์ชันเส้นทาง 4 ฟังก์ชันสำหรับสร้างต้นไม้แบบทอดข้ามที่แตกต่างกัน
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งยังมีให้คำปรึกษาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้
บนเครือข่ายแบบ โอเมก้า ($N = 16$)



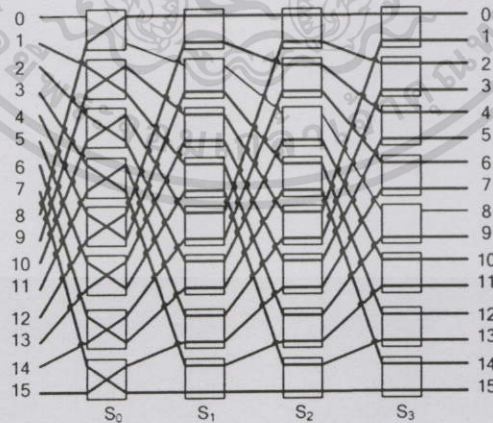
(ข) รอบที่ 2 ของฟังก์ชันเส้นทาง

$$R_1 = \{(2 \rightarrow 0), (1 \leftrightarrow 3), (4 \leftrightarrow 6), (5 \leftrightarrow 7), (8 \rightarrow 10), (9 \leftrightarrow 11), (12 \leftrightarrow 14), (13 \leftrightarrow 15)\}$$



(ค) รอบที่ 3 ของฟังก์ชันเส้นทาง

$$R_2 = \{(4 \rightarrow 0), (1 \leftrightarrow 5), (2 \leftrightarrow 6), (3 \leftrightarrow 7), (8 \rightarrow 12), (9 \leftrightarrow 13), (10 \leftrightarrow 14), (11 \leftrightarrow 15)\}$$



(ง) รอบที่ 4 ของฟังก์ชันเส้นทาง

$$R_3 = \{(8 \rightarrow 0), (1 \leftrightarrow 9), (2 \leftrightarrow 10), (3 \leftrightarrow 11), (4 \rightarrow 12), (5 \leftrightarrow 13), (6 \leftrightarrow 14), (7 \leftrightarrow 15)\}$$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา **รูปที่ 3.9 (ต่อ)** ถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.4.3 การสร้างต้นไม้แบบทอดข้ามที่แตกต่าง บนเครือข่ายแบบฟลิป

ในส่วนนี้จะแสดงการสร้างต้นไม้แบบทอดข้ามที่แตกต่างจำนวน k ต้น บนเครือข่ายแบบฟลิป ซึ่งจะสามารถแสดงได้ดังตัวอย่างดังต่อไปนี้

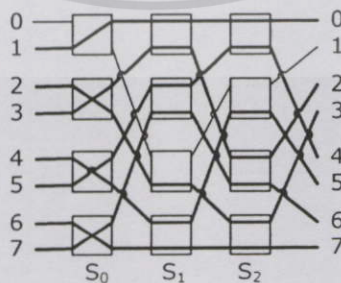
ตัวอย่างที่ 3.11 แสดงที่ขนาดของหน่วยประมวลผลคือ 8 หน่วยประมวลผล ($N = 8$) และจะใช้แทนเลขฐาน 2 ได้ 3 บิต ($k = 3$) ซึ่งมีการอธิบายการสร้างบางส่วนของทุกๆ ต้นไม้แบบทอดข้าม (T_0, T_1, T_2) ในรูปที่ 3.4 โดยใช้ฟังก์ชันเส้นทาง (R_0, R_1, R_2) ที่แสดงในรูปที่ 3.10

ในรอบที่ 1 (รูปที่ 3.10 (ก)) จะสร้างบางส่วนของ (T_0, T_1, T_2) จากฟังก์ชันเส้นทาง $R_0 = \{(1 \rightarrow 0), (2 \leftrightarrow 3), (4 \leftrightarrow 5), (6 \leftrightarrow 7)\}$ โดยการปรับทุกสวิตช์ในสเตจ S_0 แบบไขว้ (Exchange)

และปรับทุกสวิตช์ในสเตจ $S_1 + S_2$ แบบตรง (Straight)

ในรอบที่ 2 (รูปที่ 3.10 (ข)) จะสร้างบางส่วนของ (T_0, T_1, T_2) จากฟังก์ชันเส้นทาง $R_1 = \{(2 \rightarrow 0), (1 \leftrightarrow 3), (4 \leftrightarrow 6), (5 \leftrightarrow 7)\}$ โดยการปรับทุกสวิตช์ในสเตจ S_1 แบบไขว้และปรับทุกสวิตช์ในสเตจ $S_0 + S_2$ แบบตรง

ในรอบที่ 3 (รูปที่ 3.10 (ค)) จะสร้างบางส่วนของ (T_0, T_1, T_2) จากฟังก์ชันเส้นทาง $R_2 = \{(4 \rightarrow 0), (1 \leftrightarrow 5), (2 \leftrightarrow 6), (3 \leftrightarrow 7)\}$ โดยการปรับทุกสวิตช์ในสเตจ S_2 แบบไขว้และปรับทุกสวิตช์ในสเตจ $S_0 + S_1$ แบบตรง

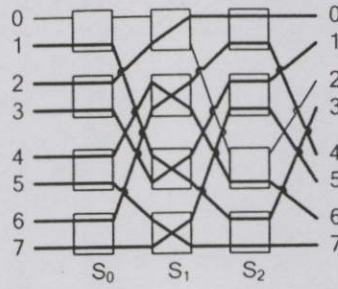


(ก) รอบที่ 1 ของฟังก์ชันเส้นทาง $R_0 = \{(1 \rightarrow 0), (2 \leftrightarrow 3), (4 \leftrightarrow 5), (6 \leftrightarrow 7)\}$

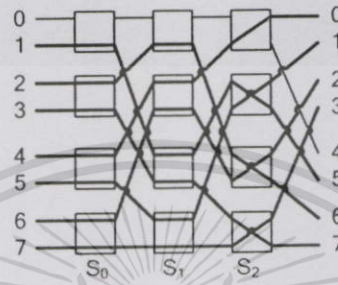
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น หากท่านมีข้อสงสัยใดๆ กรุณาติดต่อทางผู้จัดทำเอกสารนี้

รูปที่ 3.10 แสดงฟังก์ชันเส้นทาง 3 ฟังก์ชัน สำหรับสร้างต้นไม้แบบทอดข้าม

บนเครือข่ายแบบฟลิป ($N = 8$)



(ข) รอบที่ 2 ของฟังก์ชันเส้นทาง $R_1 = \{(2 \rightarrow 0), (1 \leftrightarrow 3), (4 \leftrightarrow 6), (5 \leftrightarrow 7)\}$



(ค) รอบที่ 3 ของฟังก์ชันเส้นทาง $R_2 = \{(4 \rightarrow 0), (1 \leftrightarrow 5), (2 \leftrightarrow 6), (3 \leftrightarrow 7)\}$

รูปที่ 3.10 (ต่อ)

ตัวอย่างที่ 3.12 แสดงที่ขนาดของหน่วยประมวลผล 16 หน่วยประมวลผล ($N = 16$) และจะใช้แทนเลขฐาน 2 ได้ 4 บิต ($k = 4$) ซึ่งมีการอธิบายการสร้างบางส่วนของทุกๆ ดันไม้แบบทอดข้าม (T_0, T_1, T_2, T_3) ในรูปที่ 3.5 โดยใช้ฟังก์ชันเส้นทาง (R_0, R_1, R_2, R_3) ที่แสดงในรูปที่ 3.11

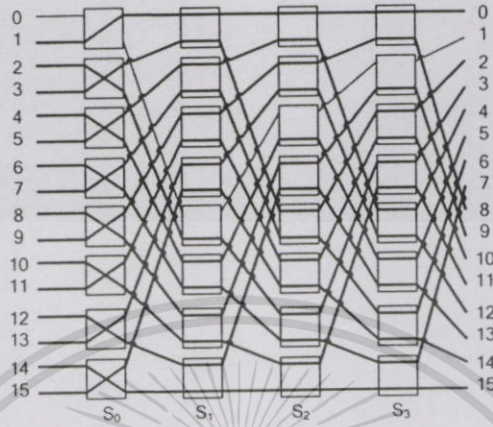
ในรอบที่ 1 (รูปที่ 3.11 (ก)) จะสร้างบางส่วนของ (T_0, T_1, T_2, T_3) จากฟังก์ชันเส้นทาง $R_0 = \{(1 \rightarrow 0), (2 \leftrightarrow 3), (4 \leftrightarrow 5), (6 \leftrightarrow 7), (8 \rightarrow 9), (10 \leftrightarrow 11), (12 \leftrightarrow 13), (14 \leftrightarrow 15)\}$ โดยการปรับทุกสวิตช์ในสเตจ S_0 แบบไขว้ (Exchange) และปรับทุกสวิตช์ในสเตจ $S_1 + S_2 + S_3$ แบบตรง (Straight)

ในรอบที่ 2 (รูปที่ 3.11 (ข)) จะสร้างบางส่วนของ (T_0, T_1, T_2, T_3) จากฟังก์ชันเส้นทาง $R_1 = \{(2 \rightarrow 0), (1 \leftrightarrow 3), (4 \leftrightarrow 6), (5 \leftrightarrow 7), (8 \rightarrow 10), (9 \leftrightarrow 11), (12 \leftrightarrow 14), (13 \leftrightarrow 15)\}$ โดยการปรับทุกสวิตช์ในสเตจ S_1 แบบไขว้ และปรับทุกสวิตช์ในสเตจ $S_0 + S_2 + S_3$ แบบตรง

ในรอบที่ 3 (รูปที่ 3.11 (ค)) จะสร้างบางส่วนของ (T_0, T_1, T_2, T_3) จากฟังก์ชันเส้นทาง $R_2 = \{(4 \rightarrow 0), (1 \leftrightarrow 5), (2 \leftrightarrow 6), (3 \leftrightarrow 7), (8 \rightarrow 12), (9 \leftrightarrow 13), (10 \leftrightarrow 14), (11 \leftrightarrow 15)\}$ โดยการปรับทุกสวิตช์ในสเตจ S_2 แบบไขว้ และปรับทุกสวิตช์ในสเตจ $S_0 + S_1 + S_3$ แบบตรง

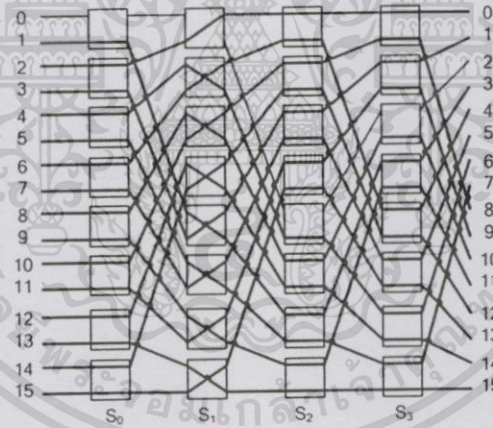
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับงานวิจัยและการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ในรอบที่ 4 (รูปที่ 3.11 (ง)) จะสร้างบางส่วนของ (T_0, T_1, T_2, T_3) จากฟังก์ชันเส้นทาง $R_3 = \{(8 \rightarrow 0), (1 \leftrightarrow 9), (2 \leftrightarrow 10), (3 \leftrightarrow 11), (4 \rightarrow 12), (5 \leftrightarrow 13), (6 \leftrightarrow 14), (7 \leftrightarrow 15)\}$ โดยการปรับทุกสวิตช์ในสเตจ S_3 แบบไขว้ และปรับทุกสวิตช์ในสเตจ $S_0 + S_1 + S_2$ แบบตรง



(ก) รอบที่ 1 ของฟังก์ชันเส้นทาง

$$R_0 = \{(1 \rightarrow 0), (2 \leftrightarrow 3), (4 \leftrightarrow 5), (6 \leftrightarrow 7), (8 \rightarrow 9), (10 \leftrightarrow 11), (12 \leftrightarrow 13), (14 \leftrightarrow 15)\}$$

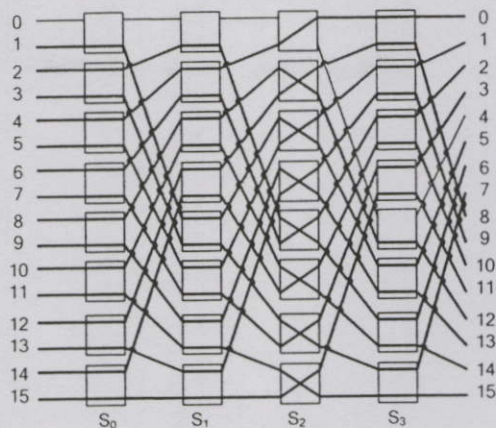


(ข) รอบที่ 2 ของฟังก์ชันเส้นทาง

$$R_1 = \{(2 \rightarrow 0), (1 \leftrightarrow 3), (4 \leftrightarrow 6), (5 \leftrightarrow 7), (8 \rightarrow 10), (9 \leftrightarrow 11), (12 \leftrightarrow 14), (13 \leftrightarrow 15)\}$$

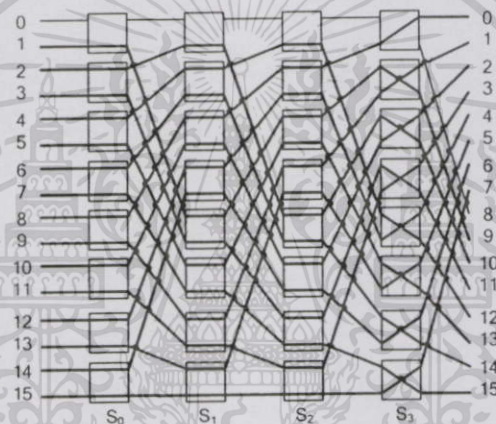
รูปที่ 3.11 แสดงฟังก์ชันเส้นทาง 4 ฟังก์ชันสำหรับสร้างต้นไม้แบบทอดข้ามที่แตกต่างบนเครือข่ายแบบฟลิป ($N = 16$)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



(ค) รอบที่ 3 ของฟังก์ชันเส้นทาง

$$R_2 = \{(4 \rightarrow 0), (1 \leftrightarrow 5), (2 \leftrightarrow 6), (3 \leftrightarrow 7), (8 \rightarrow 12), (9 \leftrightarrow 13), (10 \leftrightarrow 14), (11 \leftrightarrow 15)\}$$



(ง) รอบที่ 4 ของฟังก์ชันเส้นทาง

$$R_3 = \{(8 \rightarrow 0), (1 \leftrightarrow 9), (2 \leftrightarrow 10), (3 \leftrightarrow 11), (4 \rightarrow 12), (5 \leftrightarrow 13), (6 \leftrightarrow 14), (7 \leftrightarrow 15)\}$$

รูปที่ 3.11 (ต่อ)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 4

การวิเคราะห์ความซับซ้อนของขั้นตอนวิธี

ในงานวิจัยเรื่องการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน บนระบบเครือข่ายแบบไดนามิก ซึ่งได้นำเสนอขั้นตอนวิธีเพื่อให้ได้ผลลัพธ์ไว้ในบทที่ 3 สำหรับเนื้อหาในบทนี้จะกล่าวถึงการวิเคราะห์ความซับซ้อนด้านเวลาของขั้นตอนวิธีการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน บนระบบเครือข่ายแบบไดนามิกในหัวข้อที่ 4.1 และการแสดงการหาจำนวนต้นไม้แบบทอดข้ามที่แตกต่าง จำนวนที่มากที่สุดบนระบบเครือข่ายแบบไดนามิกในหัวข้อที่ 4.2 และในหัวข้อที่ 4.3 จะเป็นการแสดงว่าเส้นทางที่สร้างในต้นไม้แบบทอดข้ามที่แตกต่าง บนระบบเครือข่ายแบบไดนามิก เป็นเส้นทางที่แตกต่างกัน

4.1 ความซับซ้อนด้านเวลาของขั้นตอนวิธีการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน บนระบบเครือข่ายแบบไดนามิก

ในการเตรียมพร้อมเพื่อเริ่มทำการประมวลผลแบบขนานบนระบบเครือข่ายแบบไดนามิก ขนาด N ทุกๆ หน่วยประมวลผลถูกกำหนดให้ทำการประมวลผล (PE) พร้อมๆ กันด้วยคำสั่ง “for all PEs” โดยแต่ละหน่วยประมวลผลจะมีเลขประจำแบบไบนารี คือ $X = (b_{k-1}b_{k-2} \dots b_0)$ ดังนั้นคำสั่งนี้จะใช้ความซับซ้อนด้านเวลาเท่ากับ $O(1)$

การประมวลผลจากนั้นสามารถแบ่งการทำงานออกเป็น 2 ขั้นตอนดังนี้

ขั้นตอนที่ 1 ขั้นตอนวิธีการสำหรับสร้างฟังก์ชันเส้นทาง (Routing Functions)

```
Part I: // Create k routing functions ( $R_i$ )
For  $i=0$  to  $k-1$  do
  If ( $b_i=0$ )
     $parent(x)=E_i(x)$ ;
     $R_i = R_i \cup \{(x, parent(x))\}$  for  $T_i$ 
  End if
  If ( $b_i=1$ )
     $parent(x)=E_{pred(i)}(x)$ ;
     $R_{pred(i)}=R_{pred(i)} \cup \{(x, parent(x))\}$  for  $T_i$ 
  End if
End for i
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้เผยแพร่ไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

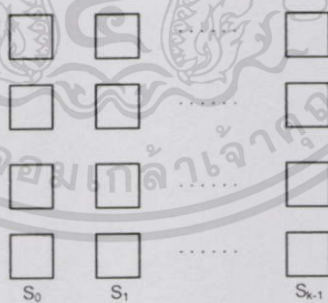
ในขั้นตอนที่ x ทุกๆ หน่วยประมวลผลของ x เมื่อ $x = 1, 2, 3, \dots, k-1$ จะสามารถสร้างฟังก์ชันเส้นทางได้จำนวน k ฟังก์ชันซึ่งเป็นค่าความสัมพันธ์ของ x และ $\text{parent}(x)$ โดยคำนวณผลลัพธ์ $\text{parent}(x)$ ทุกๆ ค่าของ X ซึ่งมี N ค่าไปพร้อมๆ กัน ดังนี้

พิจารณาจาก $X = (b_{k-1} b_{k-2} \dots b_0)$ ซึ่งจะแทนค่าของ X เป็นไบนารี และการคำนวณฟังก์ชันเส้นทางจะมีความสัมพันธ์กับตำแหน่ง b_i ของ X โดยที่ $0 \leq i \leq k$

ดังนั้นความซับซ้อนด้านเวลาสำหรับขั้นตอนวิธีการแบบขนานในขั้นตอนที่เท่ากับ $O(k)$ เพราะทุกๆ หน่วยประมวลผลของ x จะทำพร้อมๆ กันเป็นจำนวน k รอบ และในแต่ละรอบจะพิจารณาเงื่อนไขเพียง 2 เงื่อนไขเท่านั้นดังนี้ ด้วยความซับซ้อนด้านเวลาเท่ากับ $O(1)$

1. ถ้ากรณีของ $b_i = 0$ ดังนั้นจึงต้องการหาเส้นเชื่อมต่อระหว่าง $\text{parent}(x)$ กับ X ใดๆ โดยสร้างฟังก์ชันเส้นทาง ณ ตำแหน่ง i ใดๆ
2. ถ้ากรณีของ $b_i = 1$ ดังนั้นจึงต้องการหาเส้นเชื่อมต่อระหว่าง $\text{parent}(x)$ กับ X ใดๆ เพื่อสร้างฟังก์ชันเส้นทาง ณ ตำแหน่ง $\text{pred}(i)$ ใดๆ

และในการเชื่อมแต่ละคู่ จำนวน $N/2$ คู่พร้อมๆ กัน จะต้องผ่านสเตจอีกจำนวน k สเตจสำหรับกำหนดค่าให้กับสวิตช์ในแต่ละสเตจ บนระบบเครือข่ายแบบไดนามิก แสดงดังรูปที่ 4.1 เพื่อสร้างเส้นเชื่อมต่อระหว่าง $\text{parent}(x)$ กับ X ในฟังก์ชันเส้นทาง



รูปที่ 4.1 แสดงสเตจจำนวน k สเตจ

ซึ่งจะใช้ความซับซ้อนด้านเวลาเท่ากับ $O(k)$ ดังนั้นเมื่อทำขั้นตอนที่ 1 จนครบโดยผ่าน k สเตจจะใช้ความซับซ้อนด้านเวลาเท่ากับ $O(k^2)$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ขั้นตอนที่ 2 ขั้นตอนวิธีการสำหรับการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน โดยใช้ฟังก์ชันเส้นทางจากขั้นตอนที่ 1

Part II: // Construct k ISTs using R_i
 For $i=0$ to $k-1$ do
 Create partially $T_0, T_1, \dots, T_{k-1}$ using R_i ;
 End for i

ในขั้นนี้ ทุกๆ หน่วยประมวลผลของ x เมื่อ $X = 1, 2, 3, \dots, k-1$ จะสร้างต้นไม้แบบทอดข้ามที่แตกต่าง โดยใช้ฟังก์ชันเส้นทางจำนวน k ฟังก์ชันไปพร้อมๆ กัน ในส่วนของขั้นตอนนี้ก็ต้องการ k รอบเช่นกันในการหาเส้นเชื่อมต่อระหว่างโหนดพ่อหรือแม่ของ X ($parent(x)$) คือ Y กับ X ใดๆ ผ่าน k แสดงแสดงดังรูปที่ 4.2 ดังนั้นที่นี้สามารถคำนวณได้ด้วยเวลาซับซ้อนด้านเวลาเท่ากับ $O(k^2)$

รูปที่ 4.2 แสดงการเชื่อมต่อระหว่าง Y กับ X ใดๆ

เมื่อทำครบทั้ง 2 ขั้นตอนจะสามารถคำนวณความซับซ้อนด้านเวลาได้ดังสมการที่ 4.1

ดังต่อไปนี้

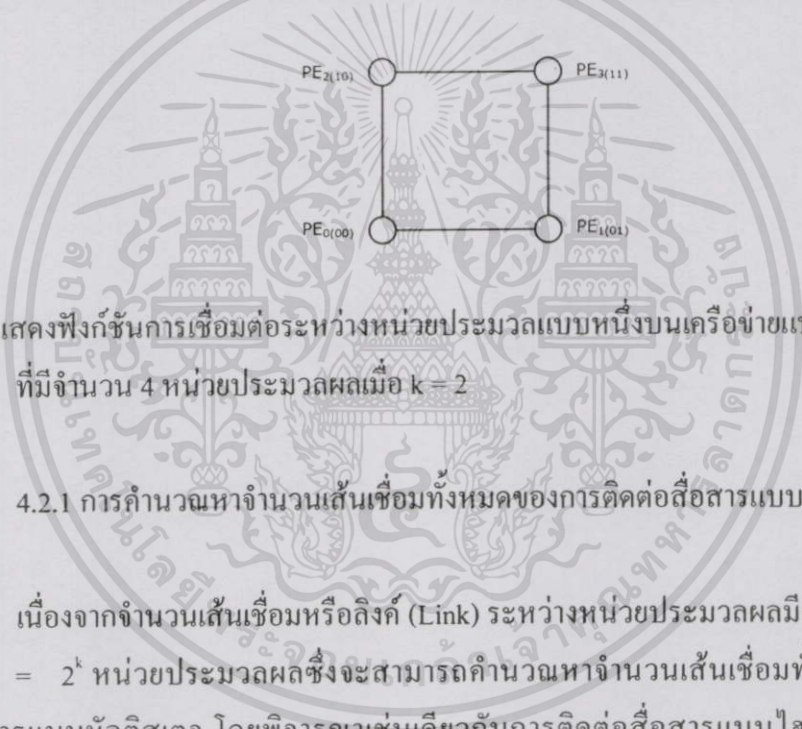
$$\begin{aligned} \text{ความซับซ้อนด้านเวลา} &= O(k^2) + O(k^2) \\ &= O(2k^2) = O(k^2) \end{aligned} \quad (4.1)$$

ดังนั้นสามารถสรุปได้ว่า ขั้นตอนวิธีการสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน บนระบบเครือข่ายแบบไดนามิก ใช้ความซับซ้อนด้านเวลาเท่ากับ $O(k^2)$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.2 การคำนวณจำนวนของต้นไม้แบบทอดข้ามที่แตกต่างที่มีจำนวนมากที่สุด บนระบบเครือข่ายแบบไดนามิก

หัวข้อนี้จะกล่าวถึงการคำนวณเพื่อหาจำนวนของต้นไม้แบบทอดข้ามที่แตกต่างที่มีจำนวนมากที่สุด บนระบบเครือข่ายแบบไดนามิก ซึ่งวิธีการคำนวณนี้จะเหมือนกับวิธีการคำนวณหาจำนวนของต้นไม้แบบทอดข้ามที่แตกต่างบนเครือข่ายแบบไฮเปอร์คิว เนื่องจากการติดต่อสื่อสารแบบมัลติสแตจ ซึ่งเป็นการติดต่อสื่อสารแบบไดนามิกแบบหนึ่งนั้นมีการใช้หน่วยประมวลผลจำนวน $N = 2^k$ หน่วย ที่มีการเชื่อมต่อถึงกันผ่าน k สแตจ เมื่อ k คือจำนวนบิตของเลขประจำหน่วยประมวลผลแบบไบนารี และสามารถปรับการเชื่อมต่อให้มีจำนวนเส้นเชื่อมหรือลิงค์ได้มาก เช่นเดียวกับเครือข่ายแบบไฮเปอร์คิว ที่มีขนาดเท่ากัน ($N = 2^k$) ดังแสดงได้ดังรูปที่ 4.3



รูปที่ 4.3 แสดงฟังก์ชันการเชื่อมต่อระหว่างหน่วยประมวลผลแบบหนึ่งบนเครือข่ายแบบมัลติสแตจที่มีจำนวน 4 หน่วยประมวลผลเมื่อ $k = 2$

4.2.1 การคำนวณหาจำนวนเส้นเชื่อมทั้งหมดของการติดต่อสื่อสารแบบไดนามิก

เนื่องจากจำนวนเส้นเชื่อมหรือลิงค์ (Link) ระหว่างหน่วยประมวลผลมีจำนวน L_k เส้น สำหรับ $N = 2^k$ หน่วยประมวลผลซึ่งจะสามารถคำนวณหาจำนวนเส้นเชื่อมทั้งหมดบนการติดต่อสื่อสารแบบมัลติสแตจ โดยพิจารณาเช่นเดียวกับการติดต่อสื่อสารแบบไฮเปอร์คิวได้ดังสมการที่ 4.2

$$L_k = 2L_{k-1} + 2^k \quad (4.2)$$

หรือ $L_k = kN$ เมื่อ $N = 2^k$

เมื่อแต่ละลิงค์ใด ๆ มีการติดต่อสื่อสารแบบสองทิศทาง เอกสารนี้เป็นเอกสารที่เผยแพร่เพื่อวัตถุประสงค์ในการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตัวอย่างที่ 4.1 แสดงการคำนวณเส้นเชื่อมทั้งหมดของการติดต่อสื่อสารแบบไดนามิก โดยใช้สมการที่ 4.2 เมื่อ k มีค่าดังต่อไปนี้

ในกรณีที่ $k = 1$

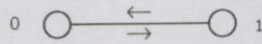
$$L_k = 2L_{k-1} + 2^k$$

$$L_1 = 2L_0 + 2^1$$

$$= 2(0) + 2$$

$$= 2$$

ซึ่งจะแสดงจำนวนเส้นเชื่อมทั้งหมดดังรูปที่ 4.4



รูปที่ 4.4 แสดงจำนวนเส้นเชื่อมจำนวน 2 เส้น ที่ $N = 2$ และ $k = 1$

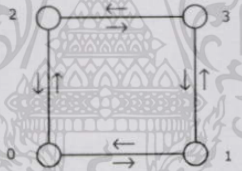
ในกรณีที่ $k = 2$

$$L_k = 2L_{k-1} + 2^k$$

$$= 2(2) + 4$$

$$= 8$$

ซึ่งจะแสดงจำนวนเส้นเชื่อมทั้งหมดดังรูปที่ 4.5



รูปที่ 4.5 แสดงจำนวนเส้นเชื่อมจำนวน 8 เส้น ที่ $N = 4$ และ $k = 2$

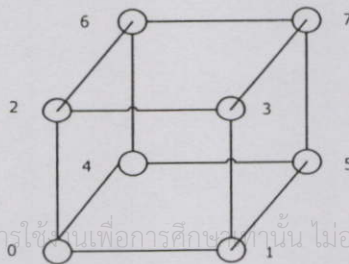
ในกรณีที่ $k = 3$

$$L_k = 2L_{k-1} + 2^k$$

$$= 2(8) + 8$$

$$= 24$$

ซึ่งจะแสดงจำนวนเส้นเชื่อมทั้งหมดดังรูปที่ 4.6



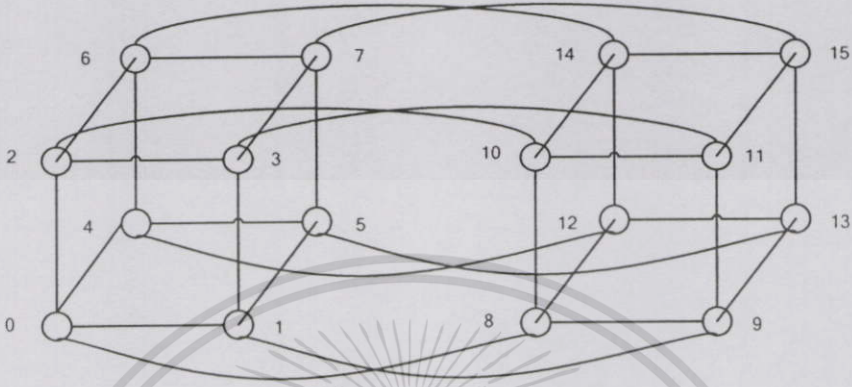
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้ภายในเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 4.6 แสดงจำนวนเส้นเชื่อมจำนวน 24 เส้นทาง ที่ $N = 8$ และ $k = 3$

ในกรณีที่ $k = 4$

$$\begin{aligned} L_k &= 2L_{k-1} + 2^k \\ &= 2(24) + 16 \\ &= 64 \end{aligned}$$

ซึ่งจะแสดงจำนวนเส้นเชื่อมทั้งหมดดังรูปที่ 4.7



รูปที่ 4.7 แสดงจำนวนเส้นเชื่อมจำนวน 64 เส้น ที่ $N = 16$ และ $k = 4$

4.2.2 การคำนวณจำนวนของต้นไม้แบบทอดข้ามที่แตกต่างที่มีจำนวนมากที่สุดบนเครือข่ายแบบไดนามิก

ถ้าให้จำนวนของต้นไม้แบบทอดข้ามที่แตกต่างจำนวนมากที่สุดมีจำนวน k ต้นบนระบบเครือข่ายแบบไดนามิก

เนื่องจากต้นไม้แต่ละต้นจะมีจำนวนเส้นเชื่อมที่แตกต่างทั้งหมด $N - 1$ เส้น ดังนั้นต้นไม้ k ต้น จะต้องการจำนวนเส้นเชื่อมที่แตกต่างทั้งหมด $k(N - 1)$ เส้น

เมื่อ $k(N - 1) < L_k \leq (k + 1)(N - 1)$ ดังนั้น k ก็คือจำนวนมากที่สุดของต้นไม้แบบทอดข้ามที่แตกต่างที่สามารถสร้างได้จากเครือข่ายขนาด $N = 2^k$ และจำนวนเส้นเชื่อมที่แตกต่าง $L_k = kN$ ดังแสดงในสมการที่ 4.2

ดังนั้นจะสามารถคำนวณหาจำนวนของเส้นเชื่อมต่อในต้นไม้แบบทอดข้ามที่แตกต่างทุก ๆ ต้น (L_{ISTs}) ได้จากสมการที่ 4.3

$$L_{ISTs} = n_{IST} \times L_T \quad (4.3)$$

เมื่อ n_{IST} แทนจำนวนของต้นไม้แบบทอดข้ามที่แตกต่าง โดยที่ $n_{IST} = k$

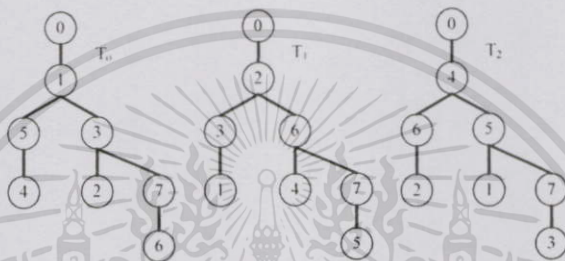
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับกรณีศึกษาเท่านั้น ไม่สามารถนำไปใช้เพื่อวัตถุประสงค์อื่นได้
 แทนจำนวนของเส้นเชื่อมในต้นไม้แบบทอดข้ามที่แตกต่างในแต่ละต้น
 โดยที่ $L_T = N - 1$

ตัวอย่างที่ 4.2 แสดงการคำนวณจำนวนของต้นไม้แบบทอดข้ามที่แตกต่าง (ISTs) ที่มีจำนวนมากที่สุด k ต้น และเส้นเชื่อมของทุกๆ ต้นบนการติดต่อสื่อสารแบบไดนามิก โดยใช้สมการที่ 4.3 เมื่อ k และ N มีค่าดังต่อไปนี้

ในกรณีที่ $k = 3$ และ $N = 8$ ดังนั้น $n_{IST} = 3$ และ $L_T = 8 - 1 = 7$

$$\begin{aligned} L_{ISTs} &= n_{IST} \times L_T \\ &= 3 \times 7 \\ &= 21 < L_k = 24 \end{aligned}$$

ซึ่งแสดงดังรูปที่ 4.8

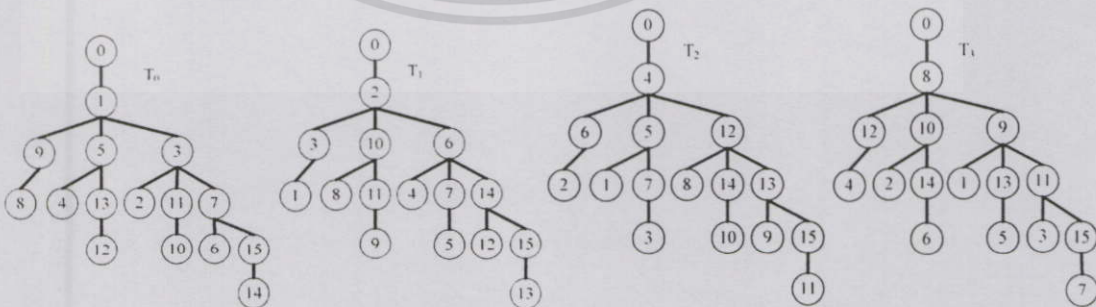


รูปที่ 4.8 แสดงเส้นเชื่อมต่อระหว่างหน่วยประมวลผลหรือโหนดจำนวน 21 เส้นที่ $N = 8$ ในต้นไม้แบบทอดข้ามที่แตกต่าง 3 ต้น

ในกรณีที่ $k = 4$ และ $N = 16$ ดังนั้น $n_{IST} = 4$ และ $L_T = 16 - 1 = 15$

$$\begin{aligned} L_{ISTs} &= n_{IST} \times L_T \\ &= 4 \times 15 \\ &= 60 < L_k = 64 \end{aligned}$$

ซึ่งแสดงดังรูปที่ 4.9



รูปที่ 4.9 แสดงเส้นเชื่อมต่อระหว่างหน่วยประมวลผลหรือโหนดจำนวน 60 เส้น ที่ $N = 16$ ในต้นไม้แบบทอดข้ามที่แตกต่าง 4 ต้น

4.3 การคำนวณหาเส้นทางที่แตกต่างกันของต้นไม้แบบทอดข้ามที่แตกต่างกัน บนระบบเครือข่ายแบบไดนามิก

ในการคำนวณหาเส้นทางที่แตกต่างกันของต้นไม้แบบทอดข้ามที่แตกต่างกัน บนระบบเครือข่ายแบบไดนามิกจะสามารถพิจารณาได้เช่นเดียวกับที่แสดงบนเครือข่ายแบบไฮเปอร์คิว เมื่อเราทำการสร้างต้นไม้แบบทอดข้ามที่แตกต่างกันจากขั้นตอนวิธีการได้ครบทุกๆ ต้นแล้ว ดังนั้นจะสังเกตเห็นว่าเส้นทางจากรากของ T_i ไปยังโหนดอื่นๆ จะมีเส้นทางที่ไม่ซ้ำกัน ซึ่งจะสามารถค้นหาเส้นทางที่ไม่ซ้ำกันได้โดยการเชื่อม X และ 0 ใน T_i ได้ 2 กรณีดังต่อไปนี้โดย

สมมติให้ : X แทนเลขประจำโหนดใดๆ แบบไบนารีดังนี้

$$X = (b_{k-1}b_{k-2} \dots b_1 \dots b_2b_1b_0)$$

และให้ Y เป็นโหนดพ่อแม่ของ X ($parent(x)$)

กรณีที่ 1 ถ้า $b_i = 1$ ดังนั้น โหนด Y ที่อยู่ในเส้นทางที่ไม่ซ้ำกันระหว่างโหนด X และ โหนดรากที่ 0 คือการสลับบิตที่ j โดยที่ $j = pred(i)$ ของ X ในเมตริกซ์ $I_E(x)$

$$\begin{aligned} X(b_{k-1}b_{k-2} \dots b_i \dots b_2b_1b_0) &\rightarrow Y(b_{k-1}b_{k-2} \dots \bar{b}_i \dots b_2b_1b_0) \\ &\rightarrow \dots \\ &\rightarrow 0 \end{aligned}$$

เมื่อเมตริกซ์ $I_E(x)$ มีการออกแบบไว้ดังต่อไปนี้

$$I_E(x) = \begin{bmatrix} i_0 & i_1 & \dots & i_{t-3} & i_{t-1} \\ i_1 & i_2 & \dots & i_{t-1} & i_0 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ i_{t-2} & i_{t-1} & \dots & i_{t-4} & i_{t-3} \\ i_{t-1} & i_0 & \dots & i_{t-3} & i_{t-2} \end{bmatrix}$$

ซึ่งแสดงการคำนวณไว้ในหัวข้อที่ 3.2 ในบทที่ 3

กรณีที่ 2 ถ้า $b_i = 0$ ดังนั้น โหนด Y ที่อยู่ในเส้นทางที่ไม่ซ้ำกันระหว่างโหนด X และ โหนดรากที่ 0 คือการสลับบิต ณ ตำแหน่งของบิตที่ i ของ X ใน T_i ดังนี้

$$\begin{aligned} X(b_{k-1}b_{k-2} \dots b_i \dots b_2b_1b_0) &\rightarrow Y(b_{k-1}b_{k-2} \dots \bar{b}_i \dots b_2b_1b_0) \\ &\rightarrow \dots \\ &\rightarrow 0 \end{aligned}$$

เอกสารนี้เป็นเอกสารที่สงวนลิขสิทธิ์กับสถาบันเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตัวอย่างที่ 4.3 พิจารณาเส้นทางที่ไม่ซ้ำกันสำหรับทุกๆ โหนดในเครือข่ายแบบมัลติสแตจขนาด $N = 8$ เช่น $1(001)$, $2(010)$, $3(011)$, $4(100)$, $5(101)$, $6(110)$, $7(111)$ ใน Q , โดยจะแสดงเส้นทางที่ไม่ซ้ำกันจากโหนดต่างๆ ดังกล่าวไปยัง 0 ในทุกๆ ต้นไม้แบบทอดข้าม T_i ดังต่อไปนี้ ดังแสดงในรูปที่ 4.8

$$T_0[1, 0] \quad 1(001) \rightarrow 0$$

$$T_1[1, 0] \quad 1(001) \rightarrow 3(110) \rightarrow 2(010) \rightarrow 0$$

$$T_2[1, 0] \quad 1(001) \rightarrow 5(101) \rightarrow 4(100) \rightarrow 0$$

$$T_0[2, 0] \quad 2(010) \rightarrow 3(011) \rightarrow 1(001) \rightarrow 0$$

$$T_1[2, 0] \quad 2(010) \rightarrow 0$$

$$T_2[2, 0] \quad 2(010) \rightarrow 6(110) \rightarrow 4(100) \rightarrow 0$$

$$T_0[3, 0] \quad 3(011) \rightarrow 1(001) \rightarrow 0$$

$$T_1[3, 0] \quad 3(011) \rightarrow 2(010) \rightarrow 0$$

$$T_2[3, 0] \quad 3(011) \rightarrow 7(111) \rightarrow 5(101) \rightarrow 4(100) \rightarrow 0$$

$$T_0[4, 0] \quad 4(100) \rightarrow 5(101) \rightarrow 1(001) \rightarrow 0$$

$$T_1[4, 0] \quad 4(100) \rightarrow 6(110) \rightarrow 2(010) \rightarrow 0$$

$$T_2[4, 0] \quad 4(100) \rightarrow 0$$

$$T_0[5, 0] \quad 5(101) \rightarrow 1(001) \rightarrow 0$$

$$T_1[5, 0] \quad 5(101) \rightarrow 7(111) \rightarrow 6(110) \rightarrow 2(010) \rightarrow 0$$

$$T_2[5, 0] \quad 5(101) \rightarrow 4(100) \rightarrow 0$$

$$T_0[6, 0] \quad 6(110) \rightarrow 7(111) \rightarrow 3(011) \rightarrow 1(001) \rightarrow 0$$

$$T_1[6, 0] \quad 6(110) \rightarrow 2(010) \rightarrow 0$$

$$T_2[6, 0] \quad 6(110) \rightarrow 4(100) \rightarrow 0$$

$$T_0[7, 0] \quad 7(111) \rightarrow 3(011) \rightarrow 1(001) \rightarrow 0$$

$$T_1[7, 0] \quad 7(111) \rightarrow 6(110) \rightarrow 2(010) \rightarrow 0$$

$$T_2[7, 0] \quad 7(111) \rightarrow 5(101) \rightarrow 4(100) \rightarrow 0$$

เอกสารนี้เป็นเอกสารที่สร้างโดยระบบอัตโนมัติของห้องสมุดดิจิทัลของมหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี เพื่อใช้ในการศึกษาและวิจัยเท่านั้น ไม่สามารถนำเอกสารนี้ไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งยังห้ามนำเอกสารนี้ไปเผยแพร่หรือใช้เพื่อวัตถุประสงค์อื่นใดโดยไม่ได้รับอนุญาตจากห้องสมุดดิจิทัลของมหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี

ตัวอย่างที่ 4.4 พิจารณาเส้นทางที่ไม่ซ้ำกันสำหรับทุกๆ โหนดในเครือข่ายแบบมัลติสเตจ ดังต่อไปนี้
 $1(0001), 2(0010), 3(0011), 4(0100), 5(0101), 6(0110), 7(0111), 8(1000), 9(1001), 10(1010),$
 $11(1011), 12(1100), 13(1101), 14(1110), 15(1111)$ ใน Q_4 และ $N = 16$ โดยจะแสดงเส้นทางที่ไม่ซ้ำ
 กันจาก 13 (และ 6) ไปยัง 0 ในทุกๆ ต้นไม้แบบทอดข้าม T_i ดังต่อไปนี้ ดังแสดงในรูปที่ 4.9

$$T_0[1, 0] \quad 1(0001) \rightarrow 0$$

$$T_1[1, 0] \quad 1(0001) \rightarrow 3(0110) \rightarrow 2(0010) \rightarrow 0$$

$$T_2[1, 0] \quad 1(0001) \rightarrow 5(0101) \rightarrow 4(0100) \rightarrow 0$$

$$T_3[1, 0] \quad 1(0001) \rightarrow 9(1001) \rightarrow 8(1000) \rightarrow 0$$

$$T_0[2, 0] \quad 2(0010) \rightarrow 3(0011) \rightarrow 1(0001) \rightarrow 0$$

$$T_1[2, 0] \quad 2(0010) \rightarrow 0$$

$$T_2[2, 0] \quad 2(0010) \rightarrow 6(0110) \rightarrow 4(0100) \rightarrow 0$$

$$T_3[2, 0] \quad 2(0010) \rightarrow 10(1010) \rightarrow 8(1000) \rightarrow 0$$

$$T_0[3, 0] \quad 3(0011) \rightarrow 1(0001) \rightarrow 0$$

$$T_1[3, 0] \quad 3(0011) \rightarrow 2(0010) \rightarrow 0$$

$$T_2[3, 0] \quad 3(0011) \rightarrow 7(0111) \rightarrow 5(0101) \rightarrow 4(0100) \rightarrow 0$$

$$T_3[3, 0] \quad 3(0011) \rightarrow 11(1011) \rightarrow 9(1001) \rightarrow 8(1000) \rightarrow 0$$

$$T_0[4, 0] \quad 4(0100) \rightarrow 5(0101) \rightarrow 1(0001) \rightarrow 0$$

$$T_1[4, 0] \quad 4(0100) \rightarrow 6(0110) \rightarrow 2(0010) \rightarrow 0$$

$$T_2[4, 0] \quad 4(0100) \rightarrow 0$$

$$T_3[4, 0] \quad 4(0100) \rightarrow 12(1100) \rightarrow 8(1000) \rightarrow 0$$

$$T_0[5, 0] \quad 5(0101) \rightarrow 1(0001) \rightarrow 0$$

$$T_1[5, 0] \quad 5(0101) \rightarrow 7(0111) \rightarrow 6(0110) \rightarrow 2(0010) \rightarrow 0$$

$$T_2[5, 0] \quad 5(0101) \rightarrow 4(0100) \rightarrow 0$$

$$T_3[5, 0] \quad 5(0101) \rightarrow 13(1101) \rightarrow 9(1001) \rightarrow 8(1000) \rightarrow 0$$

เอกสารนี้เป็นเอกสารที่สงวนลิขสิทธิ์ไว้สำหรับใช้ภายในเท่านั้น ไม่อนุญาตให้เผยแพร่โดยไม่ได้รับอนุญาต
 ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

$$T_0[6, 0] \quad 6(0110) \rightarrow 7(0111) \rightarrow 3(0011) \rightarrow 1(0001) \rightarrow 0$$

$$T_1[6, 0] \quad 6(0110) \rightarrow 2(0010) \rightarrow 0$$

$$T_2[6, 0] \quad 6(0110) \rightarrow 4(0100) \rightarrow 0$$

$$T_3[6, 0] \quad 6(0110) \rightarrow 14(1110) \rightarrow 10(1010) \rightarrow 8(1000) \rightarrow 0$$

$$T_0[7, 0] \quad 7(0111) \rightarrow 3(0011) \rightarrow 1(0001) \rightarrow 0$$

$$T_1[7, 0] \quad 7(0111) \rightarrow 6(0110) \rightarrow 2(0010) \rightarrow 0$$

$$T_2[7, 0] \quad 7(0111) \rightarrow 5(0101) \rightarrow 4(0100) \rightarrow 0$$

$$T_3[7, 0] \quad 7(0111) \rightarrow 15(1111) \rightarrow 11(1011) \rightarrow 9(1001) \rightarrow 8(1000) \rightarrow 0$$

$$T_0[8, 0] \quad 8(1000) \rightarrow 9(1001) \rightarrow 1(0001) \rightarrow 0$$

$$T_1[8, 0] \quad 8(1000) \rightarrow 10(1010) \rightarrow 2(0010) \rightarrow 0$$

$$T_2[8, 0] \quad 8(1000) \rightarrow 12(1100) \rightarrow 4(0100) \rightarrow 0$$

$$T_3[8, 0] \quad 8(1000) \rightarrow 0$$

$$T_0[9, 0] \quad 9(1001) \rightarrow 1(0001) \rightarrow 0$$

$$T_1[9, 0] \quad 9(1001) \rightarrow 11(1011) \rightarrow 10(1010) \rightarrow 2(0010) \rightarrow 0$$

$$T_2[9, 0] \quad 9(1001) \rightarrow 13(1101) \rightarrow 12(1100) \rightarrow 4(0100) \rightarrow 0$$

$$T_3[9, 0] \quad 9(1001) \rightarrow 8(1000) \rightarrow 0$$

$$T_0[10, 0] \quad 10(1010) \rightarrow 11(1011) \rightarrow 3(0011) \rightarrow 1(0001) \rightarrow 0$$

$$T_1[10, 0] \quad 10(1010) \rightarrow 2(0010) \rightarrow 0$$

$$T_2[10, 0] \quad 10(1010) \rightarrow 14(1110) \rightarrow 12(1100) \rightarrow 4(0100) \rightarrow 0$$

$$T_3[10, 0] \quad 10(1010) \rightarrow 8(1000) \rightarrow 0$$

$$T_0[11, 0] \quad 11(1011) \rightarrow 3(0011) \rightarrow 1(0001) \rightarrow 0$$

$$T_1[11, 0] \quad 11(1011) \rightarrow 10(1010) \rightarrow 2(0010) \rightarrow 0$$

$$T_2[11, 0] \quad 11(1011) \rightarrow 15(1111) \rightarrow 13(1101) \rightarrow 12(1100) \rightarrow 4(0100) \rightarrow 0$$

$$T_3[11, 0] \quad 11(1011) \rightarrow 9(1001) \rightarrow 8(1000) \rightarrow 0$$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมีเหตุดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

$$T_0[12, 0] \quad 12(1100) \rightarrow 13(1101) \rightarrow 5(0101) \rightarrow 1(0001) \rightarrow 0$$

$$T_1[12, 0] \quad 12(1100) \rightarrow 14(1110) \rightarrow 6(0110) \rightarrow 2(0010) \rightarrow 0$$

$$T_2[12, 0] \quad 12(1100) \rightarrow 4(0100) \rightarrow 0$$

$$T_3[12, 0] \quad 12(1100) \rightarrow 8(1000) \rightarrow 0$$

$$T_0[13, 0] \quad 13(1101) \rightarrow 5(0101) \rightarrow 1(0001) \rightarrow 0$$

$$T_1[13, 0] \quad 13(1101) \rightarrow 15(1111) \rightarrow 14(1110) \rightarrow 6(0110) \rightarrow 2(0010) \rightarrow 0$$

$$T_2[13, 0] \quad 13(1101) \rightarrow 12(1100) \rightarrow 4(0100) \rightarrow 0$$

$$T_3[13, 0] \quad 13(1101) \rightarrow 9(1001) \rightarrow 8(1000) \rightarrow 0$$

$$T_0[14, 0] \quad 14(1110) \rightarrow 15(1111) \rightarrow 7(0111) \rightarrow 3(0011) \rightarrow 1(0001) \rightarrow 0$$

$$T_1[14, 0] \quad 14(1110) \rightarrow 6(0110) \rightarrow 2(0010) \rightarrow 0$$

$$T_2[14, 0] \quad 14(1110) \rightarrow 12(1100) \rightarrow 4(0100) \rightarrow 0$$

$$T_3[14, 0] \quad 14(1110) \rightarrow 10(1010) \rightarrow 8(1000) \rightarrow 0$$

$$T_0[15, 0] \quad 15(1111) \rightarrow 7(0111) \rightarrow 3(0011) \rightarrow 1(0001) \rightarrow 0$$

$$T_1[15, 0] \quad 15(1111) \rightarrow 14(1110) \rightarrow 6(0110) \rightarrow 2(0010) \rightarrow 0$$

$$T_2[15, 0] \quad 15(1111) \rightarrow 13(1101) \rightarrow 12(1100) \rightarrow 4(0100) \rightarrow 0$$

$$T_3[15, 0] \quad 15(1111) \rightarrow 11(1011) \rightarrow 9(1001) \rightarrow 8(1000) \rightarrow 0$$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทสรุปและแนวทางการพัฒนางานวิจัย

เนื้อหาในบทนี้เป็นบทสรุปของงานวิจัย โดยสรุปว่าในงานวิจัยนี้มีการออกแบบขั้นตอนวิธีการแบบขนาน บนระบบเครือข่ายแบบไดนามิก ที่มีการพัฒนาปรับปรุงจากขั้นตอนวิธีเดิมบนระบบเครือข่ายแบบสแตติก ทำให้ได้ขั้นตอนวิธีใหม่บนระบบเครือข่ายแบบไดนามิกที่มีความยืดหยุ่นมากกว่า และเสนอแนวทางในการพัฒนางานวิจัย

5.1 บทสรุป

ในงานวิจัยนี้ได้นำเสนอขั้นตอนวิธีการแบบขนานที่มีความยืดหยุ่น สำหรับการสร้างต้นไม้แบบทอดข้ามที่แตกต่าง บนระบบเครือข่ายแบบไดนามิก อาทิ เครือข่ายแบบคิว เครือข่ายแบบโอเมก้า และเครือข่ายแบบฟิลิป โดยที่มีโหนดรากที่ r และให้ $r=0$ ดังนั้นโหนดรากของต้นไม้แบบทอดข้ามที่แตกต่างคือ โหนดรากที่ 0 ซึ่งงานวิจัยนี้จะใช้พื้นฐานของการเพิ่มขึ้นของเส้นทางบนเมตริกซ์จัตุรัสลาติน (Increasing Exchange Path Latin Square Matrix ($I_E(x)$)) โดยการใช้ฟังก์ชันการเรียงสับเปลี่ยน หรือ ฟังก์ชันเส้นทางสับสวิตช์ โดยการพิจารณาค่าของ X ใดๆ แบบไบนารี ที่มีตำแหน่งบิตที่ i ของ X มีค่าเป็น 1 นำตำแหน่งดังกล่าวมาสร้างเมตริกซ์จัตุรัสลาติน (Latin Square Matrix) ให้สามารถสร้างฟังก์ชันเส้นทางบนระบบเครือข่ายแบบไดนามิก แล้วจึงนำฟังก์ชันเส้นทางดังกล่าวมาใช้สำหรับสร้างต้นไม้แบบทอดข้ามที่แตกต่างด้วยวิธีการแบบขนาน ซึ่งเมื่อได้ต้นไม้แบบทอดข้ามที่แตกต่างแล้วจะได้เส้นทางจากโหนดใดๆ ที่ไม่เป็น 0 ไปยังโหนดรากที่เป็น 0 ไม่ซ้ำกันในทุกๆ ต้นไม้แบบทอดข้ามที่แตกต่าง ลักษณะของระบบเครือข่ายแบบไดนามิกคือมีความยืดหยุ่นโดยผู้ใช้สามารถปรับสวิตช์ให้สามารถเชื่อมต่อกันระหว่างหน่วยประมวลผลได้ ดังนั้นการสร้างต้นไม้แบบทอดข้ามที่แตกต่าง บนระบบเครือข่ายแบบไดนามิก จะสามารถสร้างด้วยความซับซ้อนด้านเวลาเท่ากับ $O(k^2)$ โดยที่ k คือจำนวนของบิตของเลขประจำหน่วยประมวลผลหรือโหนด บนระบบที่มีขนาด $N = 2^k$ สำหรับจำนวนที่มากที่สุดของต้นไม้แบบทอดข้ามที่แตกต่างจะขึ้นอยู่กับขนาดของระบบ และประเภทของการเชื่อมต่อ โดยเราจะได้ค่าเฉลี่ยความยาวของเส้นทางของแต่ละต้นไม้แบบทอดข้ามที่แตกต่างน้อยที่สุด เหมือนๆ กับการสร้างต้นไม้แบบทอดข้ามที่แตกต่างบนเครือข่ายแบบไฮเปอร์คิว

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.2 แนวทางการพัฒนางานวิจัย

- ลดความซับซ้อนด้านเวลาของการสร้างต้นไม้แบบทอดข้ามที่แตกต่าง บนระบบเครือข่ายแบบไดนามิก
- ลดความสูงของต้นไม้แบบทอดข้ามที่แตกต่าง บนระบบเครือข่ายแบบไดนามิก เพื่อให้เส้นทางระหว่างโหนดรากไปยังโหนด X ใดๆ มีค่าลดลง
- ลดข้อจำกัดของการสร้างต้นไม้แบบทอดข้ามที่แตกต่าง บนระบบเครือข่ายแบบไดนามิก ที่ให้โหนดรากที่ r เป็นโหนด 0 เท่านั้น ให้สามารถสร้างต้นไม้แบบทอดข้ามที่แตกต่างโดยมีโหนดรากที่ r ใดๆ โดยที่ $r \neq 0$



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เอกสารอ้างอิง

- [1] Bao F., Igarashi Y., and Ohring S.R., **"Reliable broadcasting in product networks,"** Discrete Applied Mathematics, Vol. 83, 1998, pp. 3-20.
- [2] El-Rewini H., and Abd-El-Barr M., **Advanced computer architecture and parallel processing,** Jonh Wiley & Sons, Inc., 2005.
- [3] Hwang K. **"Advanced Computer Architecture: Parallelism, Scalability, Programmability,"** New York: McGraw-Hill, 1993.
- [4] Huck A., **"Independent trees in graphs,"** Graphs and Combinatorics, Vol. 10, 1994, pp. 29-45.
- [5] Huck A., **"Independent trees in planar graphs,"** Graphs and Combinatorics, Vol. 15, 1999, pp. 29-77.
- [6] Iwasaki Y., Kajiwar Y., Obokata K., and Igarashi Y., **"Independent spanning trees of chordal rings,"** Information processing Letters, Vol. 39, 1999, pp. 155-160.
- [7] Kim S., and Chung I., **"Application of the special Latin square to a parallel routing algorithm on a recursive circulant network,"** Information Processing Letters, Vol. 66, 1998, pp. 141-147.
- [8] Kumar V. **Introduction to Parallel Computing: Design and Analysis of Algorithms.** Redwood City, California. Benjamin-Cummings. 1994.
- [9] Obokata K., Iwasaki Y., Boa F., and Igarashi Y., **"Independent spanning trees of product graphs,"** Lecture Springer Berlin / Heidelberg, 2006.
- [10] Petagon P., and Werapun J. **"Parallel Construction of Independent Spanning Trees on Dynamic Interconnection Networks."** International Joint conference on computer science and Software Engineering (JCSSE 2008). Vol. 1: International Sessions, 2008, pp. 187-192.
- [11] Quinn M.J. **Parallel Computing: Theory and Practice.** 2nd ed. New York : McGraw-Hill. 1994.
- [12] Quinn M.J., **"Parallel Programming in C with MPI and OpenMP,"** International Edition เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับอาจารย์และเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า 2003, Singapore: McGraw Hill, 2003.

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- [13] Shen X., Yang F., and Pan Y., "Equivalent permutation capabilities between time-division optical omega networks and non-optical extra-stage omega networks," IEEE/ACM Transaction on Networking, Vol. 9, No. 4, 2001, pp.518-524.
- [14] Tang S.-M., Yang J.-S., Chang J.-M., and Wang Y.-L., "Parallel construction of independent spanning trees on multidimensional tori," Proceedings of The 24th workshop on combinatorial mathematics and computation theory (CMCT'07), Nantou, Taiwan, Nantou, 2007, pp. 085-093.
- [15] Wu C.-L., and Feng T.-Y., "On a class of multistage interconnection networks," IEEE Transaction on Computers, Vol. 29, No. 8, 1980, pp.694-702.
- [16] Yang J.-S., Chang J.-M., Tang S.-M. and Wang Y.-L., "On the independent spanning trees problem in a hypercube," ICS, 2006, pp. 734-737.
- [17] Yang J.-S., Chang J.-M., Tang S.-M., and Wang Y.-L., "Reducing the height of independent spanning trees in chordal rings," IEEE Transaction on Parallel and Distributed Systems, Vol. 18, No. 5, 2007, pp. 644-657.
- [18] Yang J.-S., Chang J.-M., Tang S.-M., and Wang Y.-L., "Parallel construction of optimal independent spanning trees on hypercubes," ScienceDirect Parallel Computing, Vol. 33, No. 1, 2007, pp. 73-79.
- [19] จุรีพร บุญนิม, "การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคบนระบบพีซีคลัสเตอร์." วิทยานิพนธ์ วิทยาศาสตร์มหาบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์ บัณฑิตวิทยาลัย, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2548.
- [20] ชัยพฤกษ์ ภักดีพรหมมา. "การเรียงลำดับข้อมูลแบบขนานด้วยวิธีผสานบนระบบเครือข่ายแบบตาข่ายรีคอนฟิกิวเรเบิล." วิทยานิพนธ์วิทยาศาสตร์มหาบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์ บัณฑิตวิทยาลัย, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2550.
- [21] สรรพวรรณ ก้นตะบุตร. "เอกสารประกอบการสอนวิชาการประมวลผลแบบขนาน." เชียงใหม่: ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยเชียงใหม่. 2547.
- [22] สุวารี แก้วปรารณา. "ขั้นตอนวิธีการหาคอนเวกซ์ฮัลสองมิติแบบขนาน บนระบบเครือข่ายแบบตาข่ายรีคอนฟิกิวเรเบิล." วิทยานิพนธ์วิทยาศาสตร์มหาบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์ บัณฑิตวิทยาลัย, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า 2550. ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



ภาคผนวก ผลงานที่ได้รับการตีพิมพ์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Parallel Construction of Independent Spanning Trees on Dynamic Interconnection Networks

Roselin Petagon and Jeeraporn Werapun

Department of Mathematics and Computer Science, Faculty of Science,
King Mongkut's Institute of Technology Ladkrabang (KMITL), Bangkok, Thailand 10520
Email: s9067512@kmitl.ac.th and ksjeerap@kmitl.ac.th

Abstract

This paper presents a flexible parallel algorithm based on the Latin square scheme to generate an optimal amount of ISTs (Independent Spanning Trees) on dynamic interconnection networks. Time complexity of our proposed parallel algorithm is $O(k^3)$, where k is a number of bits, representing the address of each processing element (PE) of all PEs ($N = 2^k$) in the dynamic networks. Our dynamic approach introduces the simple and flexible switch-setting rule and hence can easily be implemented on the fly on a variety of the dynamic interconnection networks such as Cube, Omega, Flip networks, etc. Furthermore, the average path length of each generated tree is minimized similar to that of the efficient parallel ISTs algorithm on static hypercube networks.

Keyword: Independent spanning trees (ISTs), dynamic interconnection networks, Latin square.

1. Introduction

In the considering graph $G = (V, E)$, a tree T is called a spanning tree of G if T is a sub-graph of G and contains all vertices (N) in V and some $N-1$ edge in E . The spanning trees of G are called independent spanning trees if these spanning trees share the same root (r) but all unique paths connecting vertices v and r for $v \neq r$ in the trees, are pair-wise internally disjoint. Therefore, in this study our focus is the construction of k independent spanning trees (ISTs) with an arbitrary root vertex (r).

The design of multiple ISTs has been utilized in many applications and a popular one is applied to reliable communication protocols [1]. For example, one rooted spanning trees in the underlying graph of a

network can be viewed as a fault-tolerant broadcasting scheme in data communication. Further, the fault-tolerance broadcasting scheme can be achieved by sending messages through k ISTs rooted at one source node of a k -channel graph G . If the source node is faultless, this scheme can tolerate up to $k-1$ faulty nodes and hence is more reliable for completing the broadcasting process.

The problem of parallel constructing ISTs has been studied since 1994 for specific static interconnection networks such as planar graphs [2, 3], product graphs [4], chordal rings [5, 6], hypercubes [7, 8], etc. For such static networks, the evolvement of parallel algorithms for constructing ISTs tends toward two research goals, which are efficient time first and then optimal path length. In the first approach (1994), the efficient algorithm design for constructing ISTs was linear-time complexity [2]. In that linear-time study, the authors presented the construction of ISTs on chordal rings, which can construct four ISTs rooted at the same vertex. Later in 1999 another improved approach [3] was introduced to reduce the heights of ISTs. That algorithm generated four ISTs from the chordal rings with reduced the height in each tree. In recently study (2006), Yang et al. [7] proposed an efficient-time and optimal-path-length algorithm for hypercube networks. They showed that the resulting ISTs generated from their algorithm were optimal in terms of the average path length. Later in 2007, Yang et al. [8] proposed the new algorithm that was modified from their previous study [7]. That study introduced the modified algorithm, based on Hamming distance Latin square scheme on k dimensional (k -D) hypercubes in $O(k)$ time. A Latin square of order n is an $n \times n$ array in which each of the n^2 cells contains a symbol from an alphabet of size n , such that each symbol in the alphabet occurs just once in each row and once in each column. That Latin square is applied to generate optimal k ISTs on k -D hypercubes.

เอกสารนี้เป็นเอกสารที่สงวนลิขสิทธิ์สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

In this paper, we introduce a flexible parallel construction of *ISTs* algorithm, modified from the static hypercube-based approach [8] for a number of dynamic interconnection networks with time complexity $O(k^2)$, where k is a number of bits, representing the address of each processor. Our flexible construction of *ISTs* algorithm is also simple to implement as needed on the fly because of using the flexible switch-setting rule with dynamically configured to match communication demand in user programs. Many dynamic interconnection networks covered are Cube, Omega, Flip networks, etc. In addition, the average path length of each of our generated tree is minimized similar to that of the static hypercube-based networks.

The remaining parts of this paper are organized as follows. Section 2 introduces the concept of dynamic interconnection networks and the construction of *ISTs*. Section 3 presents our proposed parallel algorithm along with its time complexity for constructing *ISTs* on dynamic interconnection network, especially Cube, Omega, and Flip networks. Finally, the last section contains the conclusion and our future work.

2. Related Work

2.1 Dynamic Interconnection Networks

Dynamic interconnection networks are established as required on the fly by user programs. The dynamic networks can be classified based on interconnection schemes as bus-based networks and switch-based networks. The bus-based dynamic networks can further be classified as a single bus or multiple buses. The switch-based dynamic networks can be classified according to the structure of the networks, called multistage interconnection networks (*MINs*) as shown in Figure 1. an example of *MINs* with $N=8$ processors and 2×2 switches, connected in three stages (S_0, S_1, S_2) and four switches per stage. Three basic interconnection topologies exist: Cube, Omega, and Flip networks. These *MINs* are used to connect among processors or processors and memory modules. The main building blocks of *MINs* [9], used in both *MIMD* and *SIMD* parallel computers with N processors, is the 2×2 cross-bar switch, which has two inputs, two outputs and each 2×2 switch can be set either straight or exchange.

On *MINs* with N processors, all possible mapping functions are N^N functions. However, if we limit the mappings only bijections (one-to-one and onto

mappings), a number of mapping functions are reduced to $N!$ functions. When a routing function f is executed in *MINs*, the data from PE_i is sent to $PE_{f(i)}$. This data routing occurs simultaneously for all active PE_i in the set ($0 \leq i \leq N-1$). An inactive PE may receive data from another PE when a routing function is executed, but it cannot transmit the data. For all *MINs*, it is necessary to assign an address to each PE and several addressing schemes may be used. For example, in one scheme each PE is addressed by using k -bit address field, where $N = 2^k$.

Let x be an address of any PE expressed by the k -bit address field: $x = (b_{k-1}, b_{k-2}, \dots, b_0)$. The dynamic interconnection can be set, according to permutations on binary representations of inputs. For example, the exchange permutation is defined as $E_i(x) = (b_{k-1}, \dots, \underline{b_i}, \dots, b_0)$, where $0 \leq i \leq k-1$, and the bit b_i is inverted. However, if one considers only bijection (one-to-one connections), the Cube, Omega, and Flip networks are functionally equivalent by some relabeling techniques [10, 11].

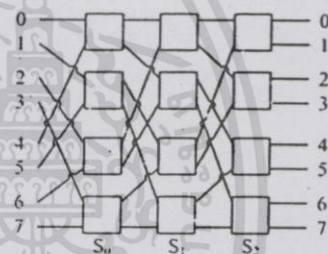


Figure 1. An example of *MINs* using 2×2 switches and $N = 8$ processors.

2.2 Independent Spanning Trees

For constructing *ISTs* with rooted $r = 0$ on dynamic networks, the parent of a vertex x ($x \neq r$), denoted by $\text{parent}(T, x)$, is a vertex adjacent to x in the path $\Pi[r, x]$. Figure 2 shows three *ISTs*, denoted by T_0, T_1 , and T_2 for $N = 8$ processors. Usually, for a k -bit address field, where $N = 2^k$ processors, we can generate k *ISTs* on the static hypercube network and also on the dynamic interconnection networks.

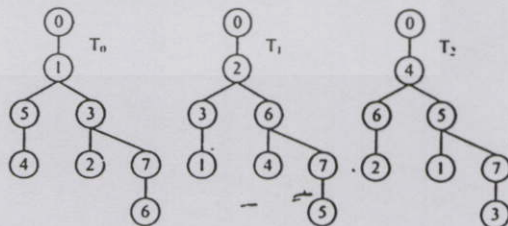


Figure 2. A set of independent spanning trees on a dynamic interconnection network with $N = 8$.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้เผยแพร่โดยไม่ได้รับอนุญาตจากสำนักพิมพ์

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3. Proposed Parallel ISTs Algorithm

In this section, we propose a parallel algorithm for constructing k ISTs rooted at address $r = 0$ on dynamic interconnection networks, especially $MINs$. Before presenting our proposed parallel ISTs algorithm, some notations and useful properties are illustrated as follows. First, we introduce the Latin square, adopted from the hypercube-based approach [8] and applied with the exchange path, to construct the optimal amount of ISTs on $MINs$.

Let $x = (b_{k-1}, b_{k-2}, \dots, b_0)$ be any address of a PE. $E_i(x)$ be an exchange permutation (defined in section 2.1) or called the exchange path in this section. Then, the following defined matrix is called the increasing exchange path Latin square (EPLS):

$$I_E(x) = \begin{bmatrix} i_0 & i_1 & \dots & i_{t-2} & i_{t-1} \\ i_1 & i_2 & \dots & i_{t-1} & i_0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ i_{t-2} & i_{t-1} & \dots & i_{t-4} & i_{t-3} \\ i_{t-1} & i_0 & \dots & i_{t-3} & i_{t-2} \end{bmatrix}$$

In the 1st row of EPLS ($i_0 < i_1 < \dots < i_{t-1}$), the element i_j represent the location of 1 in x , where $0 \leq j \leq k-1$ (i.e. 0 = min location, $k-1$ = max location) and t represent a number of 1-bit(s) in x .

In the EPLS matrix, the entry i_{j-1} is called the predecessor of i_j , denoted by $pred(i_j)$, excepted $pred(i_0) = i_j$ for the matrix size = 1×1 .

Example 1: For a system size $N = 8$ and $k = 3$, all possible binary addresses of 8 PEs are 000, 001, 010, 011, 100, 101, 110, 111. For instant, if we consider the address $s = 111_2$ (or 7), then the above generated matrix is

$$I_E(s) = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \end{bmatrix}$$

In other cases, if we consider address $u = 011_2$ (or 3), $v = 101_2$ (or 5) and $w = 110_2$ (or 6), then the generated EPLS matrices are

$$I_E(u) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, I_E(v) = \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix}, \text{ and } I_E(w) = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

Similarly, if we consider the address $x = 001_2$ (or 1), $y = 010_2$ (or 2), and $z = 100_2$ (or 4), then the generated EPLS matrices are

$$I_E(x) = [0], I_E(y) = [1], \text{ and } I_E(z) = [2]$$

The proposed parallel ISTs algorithm on any MIN is

Algorithm PARALLEL CONSTRUCTION OF ISTs

```

Begin
  For all PEs with address  $x$  ( $x \neq r$ ) do
    where  $x = b_{k-1}b_{k-2}\dots b_0$ 
    // Create  $k$  routing functions ( $R_i$ )
    For  $i=0$  to  $k-1$  do
      If ( $b_i=0$ )
        parent( $x$ ) =  $E_i(x)$ ;
         $R_i = R_i \cup \{(x, \text{parent}(x))\}$  for  $T_i$ 
      If ( $b_i=1$ )
        parent( $x$ ) =  $E_{pred(i)}(x)$ ;
         $R_{pred(i)} = R_{pred(i)} \cup \{(x, \text{parent}(x))\}$  for  $T_i$ 
      End for  $i$ 
    // Construct  $k$  ISTs using  $R_i$ 
    For  $i=0$  to  $k-1$  do
      Create partially  $T_0, T_1, \dots, T_{k-1}$  using  $R_i$ ;
    End for  $i$ 
  End for all  $x$ 
End
  
```

Example 2: For a MIN of size $N = 8$. In this case, $k = 3$ and hence there are three ISTs similar to those in Figure 2. The complete routing function (R_0, R_1, R_2) are computed for all three ISTs (T_0, T_1, T_2) as follows:

In the 1st iteration ($i = 0$), Table 1 shows the partial routing function (R_0, R_1, R_2) for constructing T_0 by $N-1$ PEs. In this case, if $b_0 = 0$ then $x \in \{010(2), 100(4), 110(6)\}$, $\text{parent}(x) = E_0(x)$ and $R_0 = R_0 \cup \{(x, \text{parent}(x))\} = R_0 \cup \{(2 \rightarrow 3), (4 \rightarrow 5), (6 \rightarrow 7)\}$. If $b_0 = 1$, one remaining case for R_0 is $x = 001(1)$, then $pred(0) = 0$ (referred $I_E(x)$) and $\text{parent}(1) = E_0(1) = 0$. The connection ($x, \text{parent}(x)$) is added in $R_0 = R_0 \cup \{(1 \rightarrow 0)\}$. Similarly, $x \in \{011(3), 101(5), 111(7)\}$ and hence $\text{parent}(x) = E_{pred(0)}(x)$, are computed as follows: $\text{parent}(3) = E_1(3) = 001$ (or 1) (referred $I_E(u)$), added in $R_1 = R_1 \cup \{(3 \rightarrow 1)\}$; $\text{parent}(5) = E_2(5) = 001$ (or 1) (referred $I_E(v)$), added in $R_2 = R_2 \cup \{(5 \rightarrow 1)\}$; $\text{parent}(7) = E_2(7) = 011$ (or 3) (referred $I_E(s)$) added in $R_2 = R_2 \cup \{(7 \rightarrow 3)\}$. Thus for $i = 0$, $R_0 = \{(1 \rightarrow 0), (2 \rightarrow 3), (4 \rightarrow 5), (6 \rightarrow 7)\}$; $R_1 = \{(3 \rightarrow 1)\}$ and $R_2 = \{(5 \rightarrow 1), (7 \rightarrow 3)\}$.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table 1.

The partial routing function (R_0, R_1, R_2) for T_0 ($i = 0$).

x	Binary string	$pred(i)$	$E_{pred(i)}$	$parent(x)$	R_i	$R_{pred(i)}$
1	001	0	0	0	0	-
2	010	-	-	3	0	-
3	011	1	1	1	-	1
4	100	-	-	5	0	-
5	101	2	1	1	-	2
6	110	-	-	7	0	-
7	111	2	3	3	-	2

In the 2nd iteration ($i = 1$), Table 2 shows the partial routing function (R_0, R_1, R_2) for constructing of T_1 by N-1 PEs. In this case, if $b_0 = 0$ then $x \in \{001(1), 100(4), 101(5)\}$, $parent(x) = E_1(x)$ and $R_1 = R_1 \cup \{(x, parent(x))\} = R_1 \cup \{(1 \rightarrow 3), (4 \rightarrow 6), (5 \rightarrow 7)\}$. If $b_0 = 1$, one remaining case for R_1 is $x = 010(2)$, then $pred(1) = 0$ (referred $E_0(v)$) and $parent(2) = E_0(2) = 0$. The connection $(x, parent(x))$ is added in $R_1 = R_1 \cup \{(2 \rightarrow 0)\}$. Similarly, $x \in \{011(3), 101(6), 111(7)\}$ and hence $parent(x) = E_{pred(i)}(x)$, are computed as follows: $parent(3) = E_1(3) = 010$ (or 2) (referred $E_1(u)$), added in $R_0 = R_0 \cup \{(3 \rightarrow 2)\}$; $parent(6) = E_2(6) = 010$ (or 2) (referred $E_2(w)$), added in $R_2 = R_2 \cup \{(6 \rightarrow 2)\}$; $parent(7) = E_1(7) = 110$ (or 6) (referred $E_1(s)$), added in $R_0 = R_0 \cup \{(7 \rightarrow 6)\}$. Thus for $i = 1$, $R_0 = R_0$ for $T_0 \cup \{(3 \rightarrow 2), (7 \rightarrow 6)\}$; $R_1 = R_1$ for $T_0 \cup \{(1 \rightarrow 3), (4 \rightarrow 6), (5 \rightarrow 7)\}$ and $R_2 = R_2$ for $T_0 \cup \{(6 \rightarrow 2)\}$.

Table 2.

The partial routing function (R_0, R_1, R_2) for T_1 ($i = 1$).

x	Binary string	$pred(i)$	$E_{pred(i)}$	$parent(x)$	R_i	$R_{pred(i)}$
1	001	-	-	3	1	-
2	010	1	0	0	-	1
3	011	0	2	2	-	0
4	100	-	-	6	1	-
5	101	-	-	7	1	-
6	110	2	2	2	-	2
7	111	0	6	6	-	0

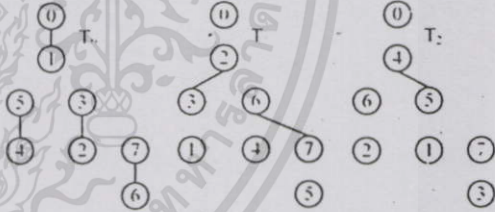
In the 3rd iteration ($i = 2$), Table 3 shows the partial routing function (R_0, R_1, R_2) for constructing T_2 by N-1 PEs. In this case, if $b_0 = 0$ then $x \in \{001(1), 010(2), 011(3)\}$, $parent(x) = E_2(x)$ and $R_2 = R_2 \cup \{(x, parent(x))\} = \{(1 \rightarrow 5), (2 \rightarrow 6), (3 \rightarrow 7)\}$. If $b_0 = 1$, one

remaining case for R_2 is $x = 100(4)$, then $pred(2) = 2$ (referred $E_2(z)$) and $parent(4) = E_2(4) = 000$ (or 0). The connection $(x, parent(x))$ is added in $R_2 = R_2 \cup \{(4 \rightarrow 0)\}$. Similarly, $x \in \{101(5), 110(6), 111(7)\}$ and hence $parent(x) = E_{pred(i)}(x)$ are computed as follows: $parent(5) = E_1(5) = 100$ (or 4) (referred $E_1(v)$), added in $R_0 = R_0 \cup \{(5 \rightarrow 4)\}$; $parent(6) = E_1(6) = 100$ (or 4) (referred $E_1(w)$), added in $R_1 = R_1 \cup \{(6 \rightarrow 4)\}$; $parent(7) = E_1(7) = 101$ (or 5) (referred $E_1(s)$), added in $R_1 = R_1 \cup \{(7 \rightarrow 5)\}$. Thus for $i = 2$, $R_0 = R_0$ for T_0 and $T_1 \cup \{(5 \rightarrow 4)\}$; $R_1 = R_1$ for T_0 and $T_1 \cup \{(6 \rightarrow 4), (7 \rightarrow 5)\}$ and $R_2 = R_2$ for T_0 and $T_1 \cup \{(1 \rightarrow 5), (2 \rightarrow 6), (3 \rightarrow 7), (4 \rightarrow 0)\}$.

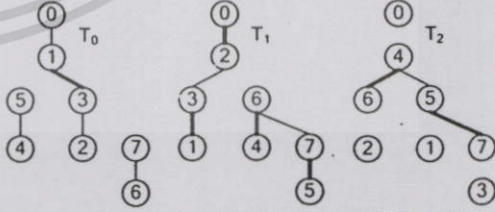
Table 3.

The partial routing function (R_0, R_1, R_2) for T_2 ($i = 2$).

x	Binary string	$pred(i)$	$E_{pred(i)}$	$parent(x)$	R_i	$R_{pred(i)}$
1	001	-	-	5	2	-
2	010	-	-	6	2	-
3	011	-	-	7	2	-
4	100	2	0	0	-	2
5	101	0	4	4	-	0
6	110	1	4	4	-	1
7	111	1	5	5	-	1



(a) The 1st routing function:
 $R_0 = \{(1 \rightarrow 0), (2 \leftrightarrow 3), (4 \leftrightarrow 5), (6 \leftrightarrow 7)\}$



(b) The 2nd routing function:
 $R_1 = \{(2 \rightarrow 0), (3 \leftrightarrow 4), (5 \leftrightarrow 6), (7 \leftrightarrow 8)\}$

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

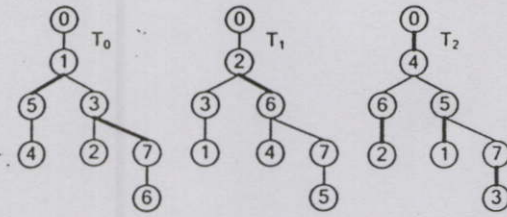


Figure 3. shows 3 routing functions for constructing ISTs on MINs ($N = 8$).

Time complexity of our parallel algorithm is $O(k^2)$ since it performs k routing functions and each routing function (R_i , $i = 0, 1, 2, \dots, k-1$) requires k stages to complete.

In addition, the average path length of each generated tree by using our approach is minimized similar to that of the efficient parallel algorithm on static hypercube networks [8]. Next in the following sections, we present our parallel construction ISTs, according to the increasing exchange path Latin square (EPLS), on three MINs, which are Cube, Omega, Flip networks.

3.1 Construction of ISTs on Cube Networks

The construction of k ISTs on a Cube interconnection network for $N = 8$ and $k = 3$ is described in Figure 4, which created ISTs (T_0 , T_1 , T_2), as shown in Figure 3.

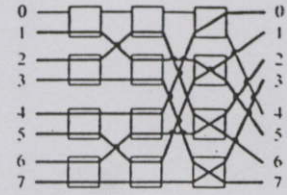
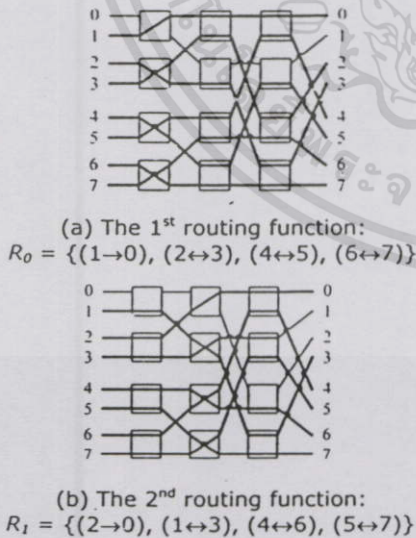


Figure 4. shows 3 routing functions for constructing ISTs on a Cube network ($N = 8$).

3.2 Construction of ISTs on Omega Networks

The construction of k ISTs on an Omega interconnection network for $N = 8$ and $k = 3$ is described in Figure 5, which created ISTs (T_0 , T_1 , T_2), as shown in Figure 3.

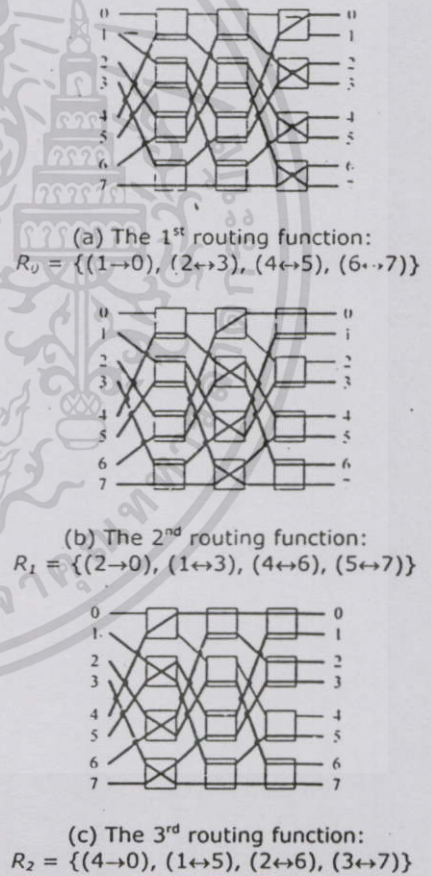


Figure 5. shows 3 routing functions for constructing ISTs on an Omega network ($N = 8$).

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.3 Construction of *ISTs* on Flip Networks

The construction of k *ISTs* on a Flip interconnection network for $N = 8$ and $k = 3$ is described in Figure 6, which created *ISTs* (T_0 , T_1 , T_2), as shown in Figure 3.

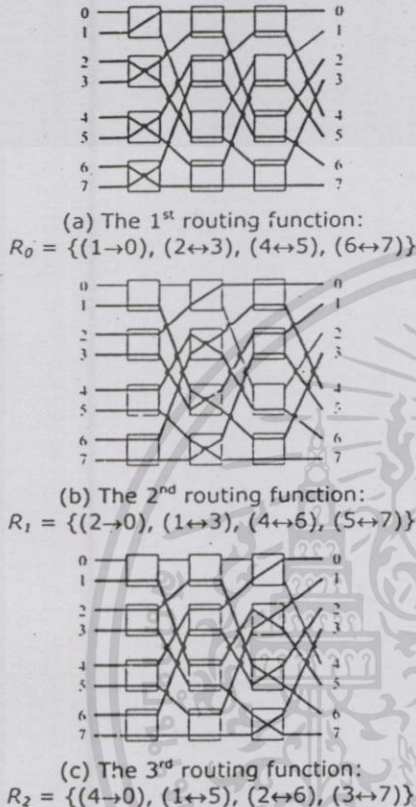


Figure 6. shows 3 routing functions for constructing *ISTs* on a Flip network ($N = 8$).

4. Conclusion and future work

In this paper, we proposed a parallel algorithm for optimal constructing *ISTs* on dynamic interconnection networks such as Cube, Omega, and Flip networks. Our *ISTs* are generated, based on the increasing exchange path Latin square matrix $I_E(x)$. Our dynamic parallel approach can be performed in $O(k^2)$ time, where k is a number of bits, represented the address of each processing element of the system size $N = 2^k$. The average path length of each of our *ISTs* is also minimized similar to that of the static hypercube-based algorithm.

In our future study, we are interesting to reduce the time complexity and the height of tree on the dynamic interconnection networks.

Acknowledgments

The authors are thankful to King Mongkut's Institute of Technology Ladkrabang (KMUTL), Bangkok, Thailand for the academic support and the research financial support.

5. References

- [1] F. Bao, Y. Igarashi, and S.R. Ohring, "Reliable broadcasting in product networks," Discrete Applied Mathematics, Vol. 83, 1998, pp. 3-20.
- [2] A. Huck, "Independent trees in graphs," Graphs and Combinatorics, Vol. 10, 1994, pp. 29-45.
- [3] A. Huck, "Independent trees in planar graphs," Graphs and Combinatorics, Vol. 15, 1999, pp. 29-77.
- [4] K. Obokata, Y. Iwasaki, F. Boa, and Y. Igarashi, "Independent spanning trees of product graphs. Lecture Springer Berlin / Heidelberg, 2006.
- [5] Y. Iwasaki, Y. Kajiura, K. Obokata, and Y. Igarashi, "Independent spanning trees of chordal rings," Information processing Letters, Vol. 39, 1999, pp. 155-160.
- [6] J.-S. Yang, J.-M. Chang, S.-M. Tang, and Y.-L. Wang, "Reducing the height of independent spanning trees in chordal rings," IEEE Transaction on Parallel and Distributed Systems, Vol. 18, No. 5, 2007, pp. 644-657.
- [7] J.-S. Yang, S.-M. Tang, J.-M. Chang and Y.-L. Wang, "On the independent spanning trees problem in a hypercube," ICS, 2006, pp. 734-737.
- [8] J.-S. Yang, J.-M. Chang, S.-M. Tang, and Y.-L. Wang, "Parallel construction of optimal independent spanning trees on hypercubes," ScienceDirect Parallel Computing, Vol. 33, No. 1, 2007, pp. 73-79.
- [9] H. El-Rewini, and M. Abd-El-Barr, Advanced computer architecture and parallel processing, John Wiley & Sons, Inc., 2005.
- [10] C.-L. Wu, and T.-Y. Feng, "On a class of multistage interconnection networks," IEEE Transaction on Computers, Vol. 29, No. 8, 1980, pp.694-702.
- [11] X. Shen, F. Yang, and Y. Pan, "Equivalent permutation capabilities between time-division optical omega networks and non-optical extra-stage omega networks," IEEE/ACM Transaction on Networking, Vol. 9, No. 4, 2001, pp.518-524.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
 ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ประวัติผู้เขียน

ชื่อ – สกุล นางสาวสลิน เพตะกร
 วัน เดือน ปีเกิด 28 พฤศจิกายน 2525
 ที่อยู่ 59 หมู่ 1 บ้านคอนแก้ว ตำบลพระธาตุ
 อำเภอเขียงกลาง จังหวัดน่าน 55160

ประวัติการศึกษา

2547 จบการศึกษาปริญญาครุศาสตรบัณฑิต
 สาขาคอมพิวเตอร์ศึกษา มหาวิทยาลัยราชภัฏเชียงใหม่
 2549- ปัจจุบัน กำลังศึกษาวิทยาศาสตรมหาบัณฑิต
 สาขาวิทยาการคอมพิวเตอร์
 ภาควิชาคณิตศาสตร์และวิทยาศาสตร์ คณะวิทยาศาสตร์
 สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ประวัติการทำงาน

2548 - ปัจจุบัน ตำแหน่งอาจารย์ สาขาวิชาคอมพิวเตอร์
 ภาควิชาวิทยาศาสตร์ประยุกต์
 คณะวิทยาศาสตร์และเทคโนโลยี
 มหาวิทยาลัยราชภัฏเชียงใหม่

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
 ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้