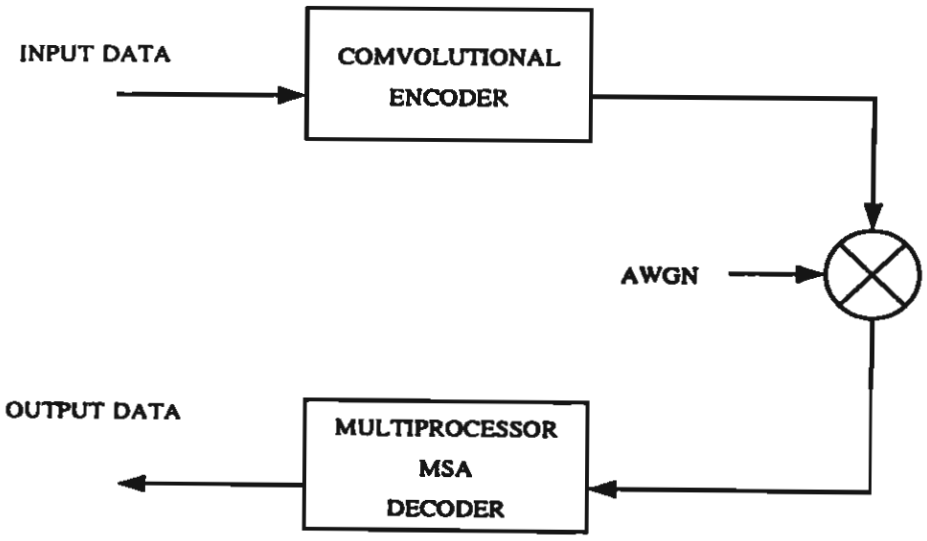
บทที่ 4
การทดสอบและผลการทดสอบ

4.1 บทนำ

บทนี้เป็นการนำเสนอผลการจำลองระบบการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ ตามแบบโครงสร้างที่ได้ศึกษาในบทที่ 3 เนื้อหาภายในบทนี้เริ่มจากการวิเคราะห์พารามิเตอร์ที่เหมาะสม เริ่มจากการหาจำนวนโนดที่เหมาะสมที่ย้ายระหว่างตัวประมวลผล (T') ซึ่ง T' เป็นพารามิเตอร์ตัวใหม่ที่แตกต่างจากระบบการถอดรหัสแบบ MSA ที่ใช้ตัวประมวลผลเดียว และได้ทดสอบพารามิเตอร์อื่น ๆ ที่มีผลต่อการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ เพื่อเปรียบเทียบประสิทธิภาพในอัตราความผิดพลาดบิตกับการถอดรหัสแบบ MSA ที่ใช้ตัวประมวลผลเดียว

4.2 การจำลองระบบสื่อสารและสมมุติฐานที่ใช้ในการทดสอบ

ในการทดสอบจะใช้แบบจำลองระบบตามภาพที่ 4.1 ที่มีการเข้ารหัสสัญญาณแบบคอนโวลูชัน อัตราการเข้ารหัส 1/2 ผ่านช่องสื่อสารเป็นแบบ AWGN (Additive White Gaussian noise) และนำสัญญาณมาถอดรหัสที่ภาครับสัญญาณด้วยการถอดรหัสแบบซีเควนเชียลโดยใช้อัลกอริทึมแบบ MSA ที่ใช้ตัวประมวลผลเดียวและ อัลกอริทึมแบบ MSA โดยมัลติโพรเซสเซอร์



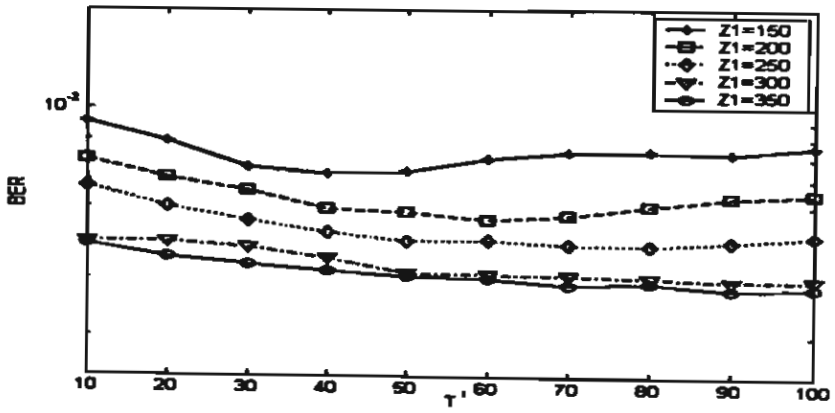
ภาพที่ 4.1 บล็อกไดอะแกรมการเข้ารหัสและถอดรหัสสำหรับการทดสอบ

เพื่อทดสอบประสิทธิภาพของหลักการที่ได้นำเสนอในงานวิจัยนี้ ผู้วิจัยได้จำลองระบบโดยการเขียนโปรแกรมคอมพิวเตอร์ภาษาซี บนระบบปฏิบัติการลินุกซ์ พารามิเตอร์ที่ใช้ในการจำลองที่สำคัญมีดังต่อไปนี้คือ ช่องสื่อสารเป็นแบบ AWGN (Additive White Gaussian noise) อัตราการเข้ารหัสคอนโวลูชันเป็น 1/2 ค่าของความยาวคอนสเตรนท์ของตัวเข้ารหัสมี 2 ค่า คือ $m = 5$ โดยมีเมตริกซ์ของตัวกำเนิด (Generator matrices) $G_1 = 110101$, $G_2 = 101111$ และ $m = 7$ ซึ่งใช้ Generator matrices $G_1 = 10111101$, $G_2 = 11001011$ จำนวนบิตต่อเฟรม K เป็น 60 บิต 120 บิตและ 180 บิตซึ่งมีจำนวนบิตทดสอบรวมเป็น 1,200,000 บิตเท่ากันในทุกกรณี $C_{\lim} = 2500$ จำนวนโนดที่ย้ายระหว่างสแตคหลักและสแตครองภายในแต่ละตัวประมวลผล T เป็น 2 ขนาดของสแตครอง Z เป็น 12 จำนวนสแตครองของระบบตัวประมวลเดียว n_z เป็น 300 และของตัวประมวลผลหลายตัว n_{ZP} เป็นไปตามสมการที่ (3.1)

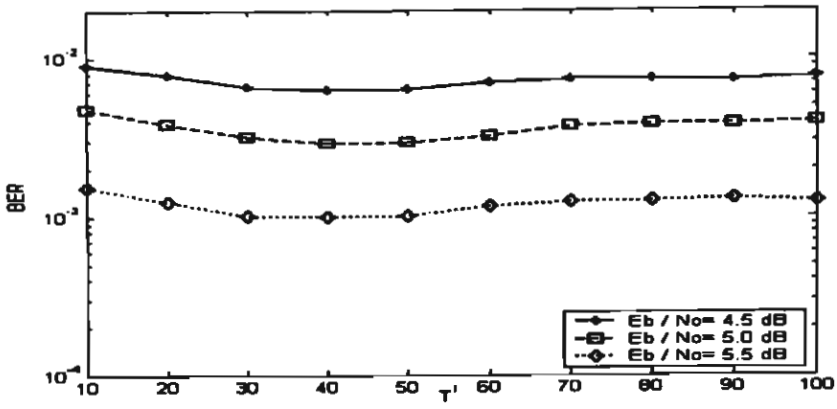
4.3 ผลการทดสอบประสิทธิภาพกับค่าพารามิเตอร์ต่าง ๆ

4.3.1 ผลจำนวนโนดที่ย้ายระหว่างตัวประมวลผล (T')

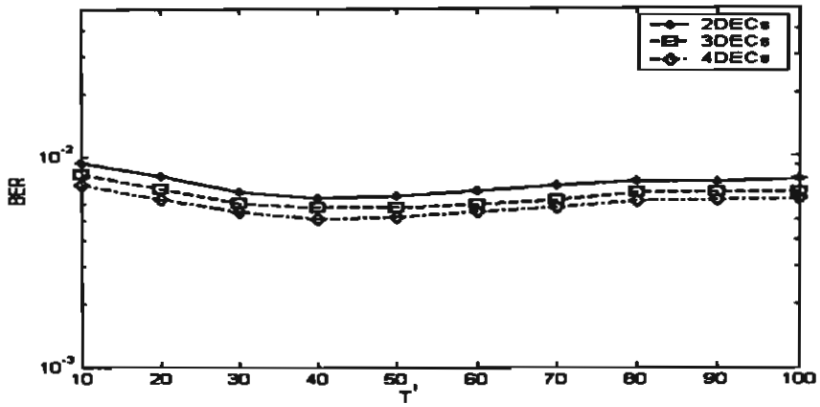
การทดสอบหาจำนวนโนดที่เหมาะสมที่ย้ายระหว่างตัวประมวลผลสำหรับงานวิจัย ผลการทดสอบ ภาพที่ 4.2 แสดงค่าอัตราความผิดพลาดบิต (Bit error rate, BER) เทียบกับจำนวนโนดที่ย้ายจาก สแตคหลักของ DEC #1 ไปยังสแตคหลักของ DEC # j ($j = 2,\ldots,n$) เมื่อกำหนดค่า $E_b / N_o = 4.5$ dB จะเห็นว่าค่า T' ที่ให้ค่า BER ต่ำที่สุดจะมากขึ้นตามค่าของ Z_1 (สังเกตจากกราฟแต่ละเส้นที่ให้ค่า BER ต่ำสุด) และเมื่อกำหนดให้ขนาดของสแตคหลักคงที่ที่ค่า $Z_1 = 150$ โดยปรับค่า E_b / N_o จาก 4.5-5.5 dB ให้ค่า BER เมื่อใช้ตัวประมวลผล 2 ตัวดังแสดงในภาพที่ 4.3 สำหรับภาพที่ 4.4 เป็นผลการจำลองเพื่อดูค่า BER เมื่อใช้ตัวประมวลผล 2 ตัว 3 ตัว และ 4 ตัว โดยกำหนดให้ $Z_1 = 150$ คงที่เช่นเดิม ภาพที่ 4.5 แสดงผลอัตราความผิดพลาดบิตที่ค่า m และ K ต่างๆกันจะเห็นว่าเมื่อใช้จำนวน m และ K มากขึ้นค่า T' ที่ทำให้ BER ต่ำที่สุดมีค่าลดลง จากผลการทดสอบข้างต้นจะเห็นว่าที่ค่า $Z_1 = 150$ ค่า T' ที่เหมาะสมคือค่าตั้งแต่ 40 ถึง 50 ดังนั้นจึงได้เลือกค่า $Z_1 = 150$ สำหรับ $m = 5$ และ $T' = 50$ เพื่อใช้ทดสอบประสิทธิภาพในบทความนี้



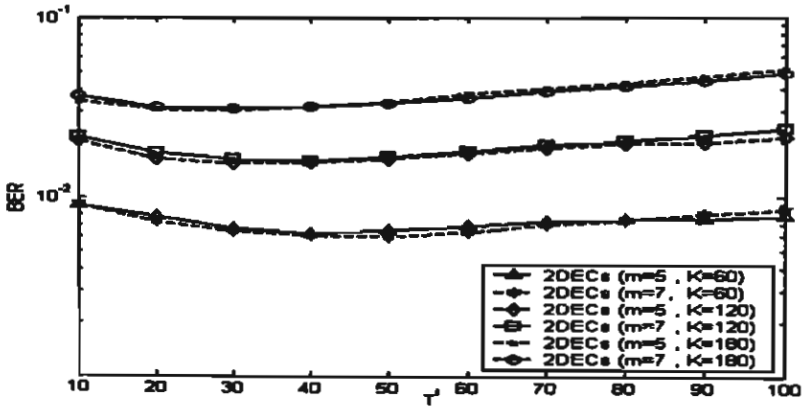
ภาพที่ 4.2 ค่า BER กับ T' เมื่อเปลี่ยนจำนวน Z_1



ภาพที่ 4.3 ค่า BER กับ T' ที่ค่า E_b / N_o ต่างๆกัน



ภาพที่ 4.4 ค่า BER กับ T' เมื่อใช้จำนวนตัวประมวลผลต่างกัน



ภาพที่ 4.5 ค่า BER กับ T' เมื่อใช้ ค่าของ m และ K ต่างกัน

ตารางที่ 4.1

จำนวนบล็อกที่ถอดรหัสเสร็จในสแตคหลักของ DEC# 1

จำนวนตัวประมวลผล	จำนวนบล็อกที่ถอดรหัสเสร็จในสแตคหลักของ DEC #1					
	$T = 20$	$T = 30$	$T = 40$	$T = 50$	$T = 60$	$T = 70$
1 DEC	18,507*					
2 DECs	18,548	18,578	18,582	18,639	18,682	18,745
3 DECs	18,569	18,635	18,692	18,763	18,832	18,903
4 DECs	18,594	18,691	18,790	18,895	18,915	19,045

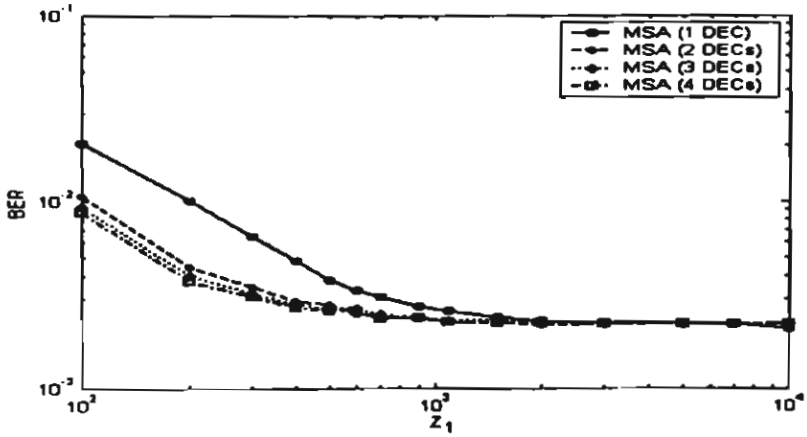
* ตัวประมวลผลตัวเดียวไม่มีการย้ายโนดข้อมูลระหว่างตัวถอดรหัส

ตารางที่ 4.1 แสดงผลจำนวนบล็อกของการถอดรหัสซึ่งสิ้นสุดภายในสแตคหลักของตัวประมวลผลตัวแรก การเพิ่ม T จะทำให้ได้จำนวนบล็อกของการถอดรหัสที่สิ้นสุดในสแตคหลักของ DEC#1 เพิ่มมากขึ้นด้วย แต่เส้นทางเดินของการถอดรหัสที่การถอดรหัสที่สิ้นสุดภายในสแตคหลักของ DEC#1 (หลังการย้าย T) นั้นอาจจะไม่ได้เป็นเส้นทางเดินของการถอดรหัสที่มีค่าแมตริกซ์ที่ดีที่สุด

4.3.2 การทดสอบผลของขนาดสแตคหลัก (Z_1) ต่อประสิทธิภาพ

โดยหลักการแล้วประสิทธิภาพในอัตราความผิดพลาดบิตของการถอดรหัสแบบ MSA ระบบตัวประมวลผลเดียวจะสามารถทำให้ดีขึ้นได้โดยการเพิ่มขนาดของสแตคหลัก ในงานวิจัยนี้ได้ทดสอบผลของขนาดของ สแตคหลักสำหรับการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ และเปรียบเทียบประสิทธิภาพในอัตราความผิดพลาดบิตกับค่าที่ได้จากการถอดรหัสแบบ MSA ระบบตัวประมวลผลเดียว จากผลการทดลองการเพิ่มขนาดของ Z_1 มีผลต่ออัตราความผิดพลาดบิตเช่นเดียวกันทั้งสองระบบ ภาพที่ 4.6 เป็นผลการทดสอบประสิทธิภาพ BER เมื่อเปลี่ยน Z_1 ค่าต่าง ๆ ที่ค่า E_b / N_o เป็น 4.5 dB เมื่อเพิ่มขนาดของสแตคหลักขึ้นเรื่อย ๆค่าอัตราความผิดพลาดบิตของการถอดรหัสแบบ MSA ทั้งสองระบบจะลดลงเรื่อย ๆตามลำดับเช่นกัน ด้วยเหตุผลเป็นไปตามสมการที่ (2.11) จนกระทั่งการเพิ่มขนาดสแตคหลักจะไม่มีผลต่ออัตราความผิดพลาดบิตของการถอดรหัสแบบ MSA ทั้งสองระบบ เมื่อเปรียบเทียบทั้งสองระบบจะเห็นว่าค่า BER ของการถอดรหัสระบบ MSA ซึ่งใช้ตัวประมวลผลตัวเดียวจะเริ่มคงที่เมื่อขนาด Z_1 เป็น 2000 และได้ค่า BER เท่ากับ $2.28 \cdot 10^{-3}$ ในขณะที่การถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ จะมีค่าคงที่เมื่อ Z_1 เป็น 1100 และได้ค่า BER เท่ากับ $2.24 \cdot 10^{-3}$ แสดงให้เห็นว่าการใช้จำนวนตัวประมวลผลมากขึ้นสามารถให้ค่าอัตราความผิดพลาดบิตที่เท่ากับหรือใกล้เคียงกับค่าอัตราความผิดพลาดบิตที่ได้จากการถอดรหัสแบบMSA ใช้ตัวประมวลผลตัวเดียว โดยใช้ขนาดของสแตคหลักเล็กกว่า แต่อย่างไรก็ตามการเพิ่มจำนวนตัวประมวลผลในระบบการถอดรหัส MSA โดยมัลติโพรเซสเซอร์ จะ

มีผลต่ออัตราความผิดพลาดบิตน้อยลงเมื่อขนาดของ Z_1 มากกว่า 700 และการเพิ่มขนาดสแตคหลักจะไม่มีผลต่ออัตราความผิดพลาดบิตของการถอดรหัสแบบ MSA ทั้งสองระบบ เมื่อค่า Z_1 มากว่า 2000 ซึ่งมีค่า BER ประมาณ $2.20*10^{-3}$ เนื่องจากการถอดรหัสแบบ MSA โดยระบบมัลติโพรเซสเซอร์สามารถถอดรหัสจำนวนบล็อกข้อมูลเสร็จภายในสแตคหลักของตัวประมวลผลตัวแรก DEC #1 ได้เพิ่มขึ้น ทำให้จำนวนบล็อกที่ถูกถอดรหัสภายใน สแตครองมีจำนวนน้อยลงและจะเป็นบล็อกที่มีสัญญาณรบกวนมาก ๆ เท่านั้น



ภาพที่ 4.6 ค่าอัตราความผิดพลาดบิตเมื่อเปลี่ยนจำนวน Z_1

4.3.3 การทดสอบผลค่าลิมิตของการคำนวณ (C_{lim}) ต่อประสิทธิภาพ

การถอดรหัสแบบซีเควนเชียลจะมีความแปรปรวนในการคำนวณ (Computational variability) คือค่าของการคำนวณของการถอดรหัสจะขึ้นอยู่กับปริมาณของสัญญาณรบกวน สำหรับการถอดรหัสแบบ MSA นั้นการกำหนดค่าลิมิตของการคำนวณเพื่อให้การถอดรหัสไม่มีอิเรชัว จะต้องกำหนดให้ C_{lim} มีค่ามากกว่าค่าวิกฤตต่ำสุดของการคำนวณ C_{crit} ตามสมการที่ 2.6 การทดสอบหาต่ำสุดของการคำนวณ C_{min} ของระบบการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์เพื่อให้การถอดรหัสไม่มีอิเรชัวนั้น แสดงตัวอย่างในตารางที่ 4.2 ผลการทดสอบค่า C_{min} ของการถอดรหัสแบบ MSA ที่ใช้ตัวประมวลผลเดียวเท่ากับ 569 ในขณะที่ค่า C_{crit} ที่ได้จากการคำนวณตามสมการที่ 2.6 เท่ากับ 897 ซึ่งการที่ค่า C_{min} ต่ำกว่าค่า C_{crit} เนื่องจากการกำหนดให้ขนาดของสแตครองมีขนาดเล็กกว่าขนาดของสแตคหลักมาก ๆ (Chevillat และ Costello ,1976) และการกำหนดค่า T เป็น 2 ผลการทดสอบเมื่อใช้ตัวประมวลผลหลายตัว ค่า C_{min} ของการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ มีค่าเท่ากับ 640, 655 และ 655 เมื่อใช้ตัวประมวลผล 2 ,3 และ 4 ตัวตามลำดับ การที่ค่า C_{min} ของระบบโดยมัลติโพรเซสเซอร์มีค่ามากกว่าค่า C_{min} ของการถอดรหัสแบบ MSA ที่ใช้ตัวประมวลผลเดียว เนื่องจากการย้ายจำนวนโนดระหว่างตัวประมวลผล T' ทำให้สแตคหลักของตัวประมวลผลตัวแรกมีเนื้อที่ในการคำนวณเพิ่มขึ้นเท่ากับ T' แต่ค่า C_{min} จะไม่เปลี่ยนแปลงมากนักเมื่อเพิ่มจำนวนตัวประมวลผลเนื่องจากการถอดรหัสสามารถถึงค่า *การตัดสินใจเบื้อง*

ต้นอันดับแรก (First Tentative decision,TD) ภายในสแตคแรกของตัวประมวลผลอันดับก่อนหน้า ภาพที่ 4.7 แสดงค่าอัตราความผิดพลาดบิตเนื่องจากผลการเพิ่มค่า C_{lim} ขึ้นเรื่อย ๆ เมื่อเพิ่มค่า C_{lim} ค่าอัตราความผิดพลาดบิตจะมีค่าลดลง เนื่องจากการเพิ่มค่า C_{lim} จะทำให้โอกาสที่การถอดรหัสจะถึงโนดสุดท้ายในแผนภูมิต้นไม้แทนรหัสมากขึ้น และที่ค่า C_{lim} เท่ากัน และการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ให้ค่าอัตราความผิดพลาดบิตที่ต่ำกว่า ในภาพที่ 4.8 แสดงจำนวนครั้งเฉลี่ยที่กระบวนการถอดรหัสถึงโนดสุดท้ายในแผนภูมิต้นไม้แทนรหัสเทียบกับค่า C_{lim} จะเห็นว่าการประยุกต์ใช้วิธีการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ให้จำนวนครั้งที่ถึงโนดสุดท้ายได้มากกว่า MSA ที่ใช้ตัวประมวลผลเดียว และนั่นหมายถึงโอกาสที่จะได้ทางเดินของการถอดรหัสที่ดีกว่าเมื่อใช้ตัวประมวลผลหลายตัว อย่างไรก็ตามจำนวนครั้งของการถึงโนดสุดท้ายในภาพที่ 4.8 ไม่ได้หมายถึงจำนวนครั้งของการบันทึกค่าใหม่ของ TD เสมอไป ภาพที่ 4.9 ได้แสดงให้เห็นถึงจำนวนครั้งเฉลี่ยที่มีการบันทึกค่าทางเดินใหม่ใน TD ซึ่งผลที่ได้สอดคล้องกับภาพที่ 4.8 นั่นคือระบบการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ให้ค่าเฉลี่ยของจำนวนครั้งในการบันทึกค่าใหม่ที่สูงขึ้นและจะสูงกว่าเดิมเมื่อเพิ่มจำนวนตัวประมวลผลในการถอดรหัสมากขึ้น

ตารางที่ 4.2

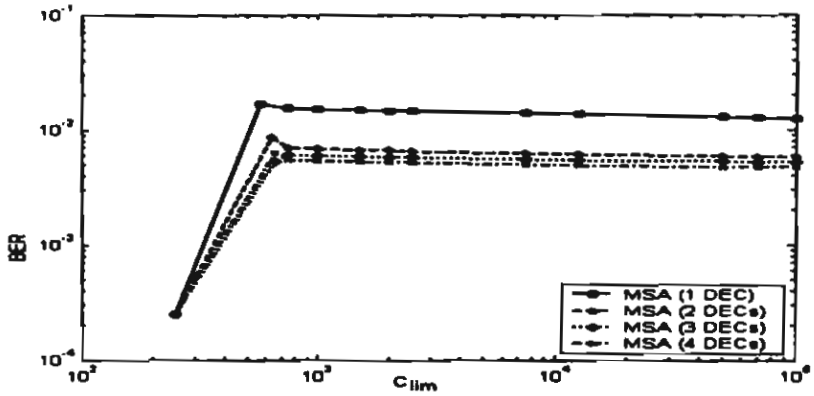
ค่าจำนวนการคำนวณต่ำสุดที่ทำให้การถอดรหัสสมบูรณ์ (erasure free)

จำนวนตัวประมวลผล	$Z_1 = 150$	$Z_1 = 300$
	C_{min}	C_{min}
1 DEC	569	855
2 DECs	640	942
3 DECs	655	956
4 DECs	655	954

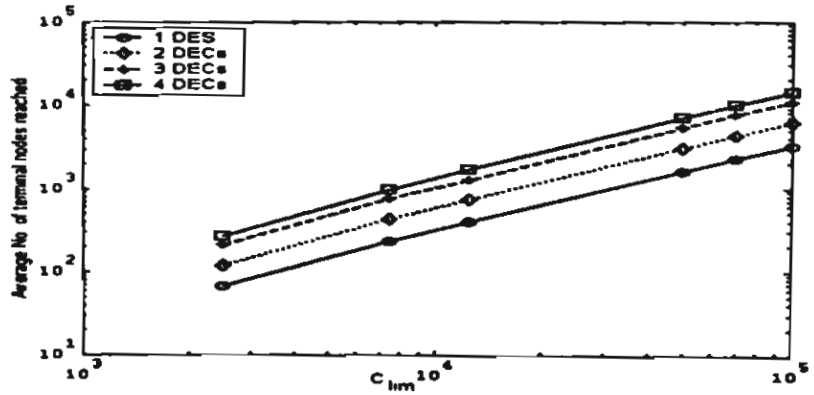
4.3.4 การทดสอบผลของขนาดของสแตครอง (Z) ต่อประสิทธิภาพ

ขนาดของสแตครองเป็นปัจจัยหนึ่งที่มีผลต่อการถอดรหัสแบบ MSA ซึ่งการกำหนดขนาดของ สแตครองจะมีผลต่ออิเรชัวได้คือถ้ากำหนดขนาดของสแตครองให้มีขนาดเล็กกว่าขนาดของสแตคหลักมาก ๆ จะทำให้การถอดรหัสสามารถถึงโนดสุดท้ายของเส้นทางตามแผนภูมิต้นไม้รหัสเร็วขึ้น และมีค่าของการคำนวณที่ต่ำลงด้วย ภาพที่ 4.10 แสดงค่าอัตราความผิดพลาดบิตเมื่อใช้ระบบการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์เมื่อทำการเปลี่ยนขนาดของสแตครองผลการทดสอบแสดงให้เห็นว่ามีอัตราความผิดพลาดบิตที่ต่ำกว่าระบบการถอดรหัสแบบ MSA เดิม แต่การเปลี่ยนขนาดของสแตครองมีผล

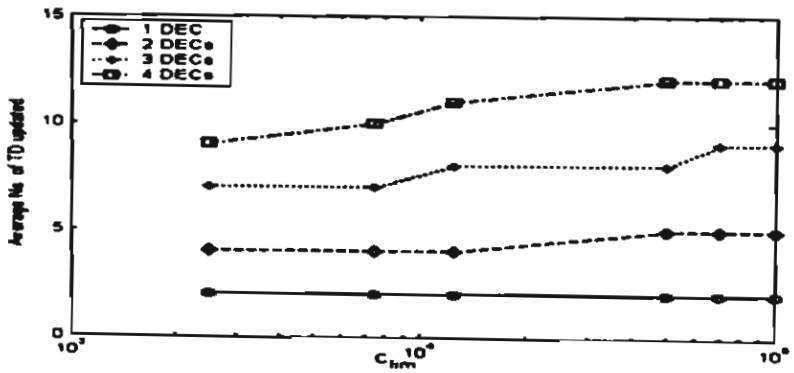
ต่ออัตราความผิดพลาดบิตที่น้อยมากทั้งสองระบบ และเมื่อเปรียบเทียบกับระบบ MSA เดิม ที่ขนาดของสแตคหลักเป็นสองเท่า (300) ค่าอัตราความผิดพลาดบิตที่ได้จากระบบการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ก็ยังมีอัตราความผิดพลาดบิตที่ต่ำกว่าซึ่งแสดงว่าการเพิ่มจำนวนตัวประมวลผลให้ประสิทธิภาพที่ดีกว่าการเพิ่มขนาดของสแตคหลักในขณะที่ปัจจัยอื่น ๆเหมือนเดิม



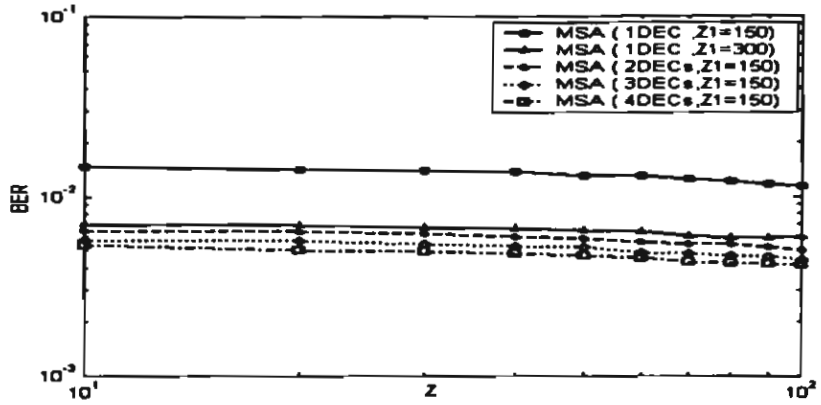
ภาพที่ 4.7 ค่าอัตราความผิดพลาดบิตเมื่อเปลี่ยนจำนวน C_{lim}



ภาพที่ 4.8 จำนวนครั้งเฉลี่ยที่การถอดรหัสถึงโนดสุดท้ายในแผนภูมิต้นไม้แทนรหัส



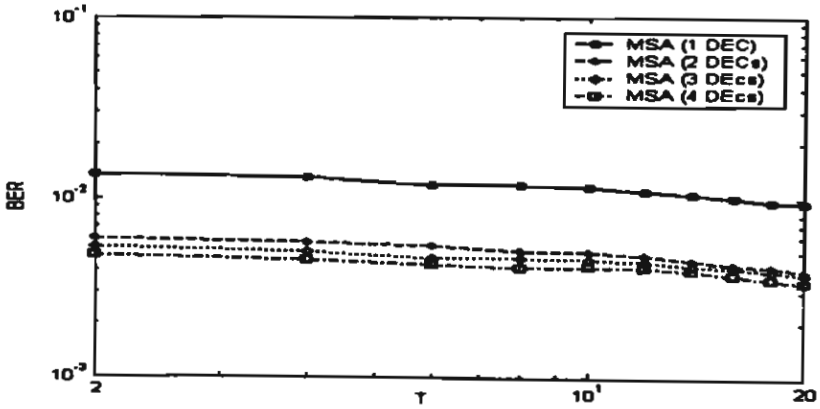
ภาพที่ 4.9 จำนวนครั้งเฉลี่ยที่มีการบันทึกค่า TD ใหม่



ภาพที่ 4.10 ค่าอัตราความผิดพลาดบิตเมื่อเปลี่ยนจำนวน Z

4.3.5 การทดสอบผลจำนวนโนด (T) ต่อประสิทธิภาพ

จำนวนโนดที่ย้ายระหว่างสแตคเป็นอีกปัจจัยหนึ่งที่มีผลต่อการถอดรหัสแบบ MSA ซึ่งการกำหนดจำนวนโนดจะมีผลต่อจำนวนสแตครองและมีผลต่ออิเรชัวคือถ้ากำหนดจำนวนโนดข้อมูลที่ถูกย้ายมากจำนวนสแตครองที่ใช้ในการถอดรหัสก็เพิ่มมากขึ้น(ในขณะที่ขนาดของสแตครองคงที่) และทำให้การถอดรหัสสามารถถึงโนดสุดท้ายของเส้นทางตามแผนภูมิต้นไม้รหัสช้าและมีค่าของการคำนวณเพิ่มขึ้นด้วย ภาพที่ 4.11 แสดงค่าอัตราความผิดพลาดบิตเมื่อเปลี่ยนจำนวนโนดข้อมูลที่ถูกย้าย ในขณะที่ T เท่ากันระบบการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์จะมีค่าอัตราความผิดพลาดบิตที่ต่ำกว่าระบบการถอดรหัสแบบ MSA เดิม และการเพิ่มจำนวนโนดข้อมูลที่ย้ายระหว่างสแตคของตัวประมวลผลแต่ละตัวจะสามารถลดค่าอัตราความผิดพลาดบิตด้วย



ภาพที่ 4.11 ค่าอัตราความผิดพลาดบิตเมื่อเปลี่ยนจำนวน T

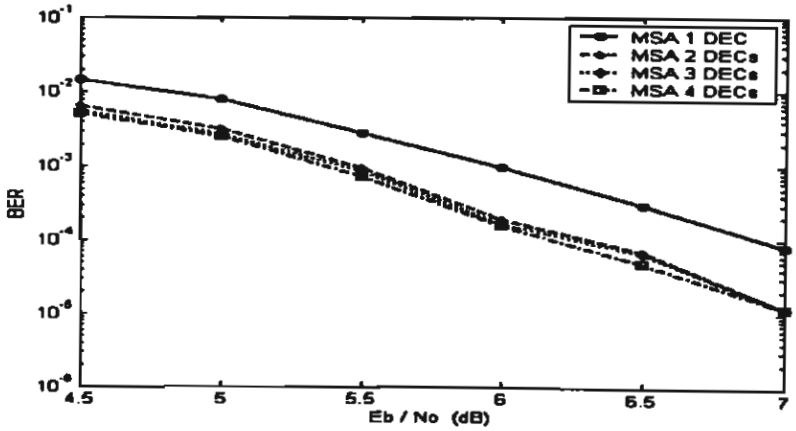
4.3.6 ค่าอัตราความผิดพลาดบิตที่ค่าสัญญาณรบกวนต่าง ๆ

ผลการทดสอบการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ที่ค่าสัญญาณรบกวนค่าต่าง ๆเพื่อเปรียบเทียบประสิทธิภาพในอัตราความผิดพลาดบิตกับการถอดรหัสแบบ MSA ที่ใช้ตัวประมวลผลเดียวภาพที่ 4.12 เป็นการทดสอบประสิทธิภาพของ BER เมื่อเปลี่ยนค่า E_b / N_o ในช่วง $4.5-7.0\,dB$ (ค่าความยาวคอน สเตรนท์ของตัวเข้ารหัส $m=5$, Generator matrices $G_1=110101$, $G_2=101111$) จะเห็นว่าค่า BER ในการถอดรหัสเมื่อใช้วิธีการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์มีค่าต่ำกว่า BER ของระบบ MSA เดิมซึ่งใช้ตัวประมวลผลตัวเดียวและค่า BER จะมีค่าลดลงต่อไปอีกเมื่อเพิ่มตัวประมวลผล

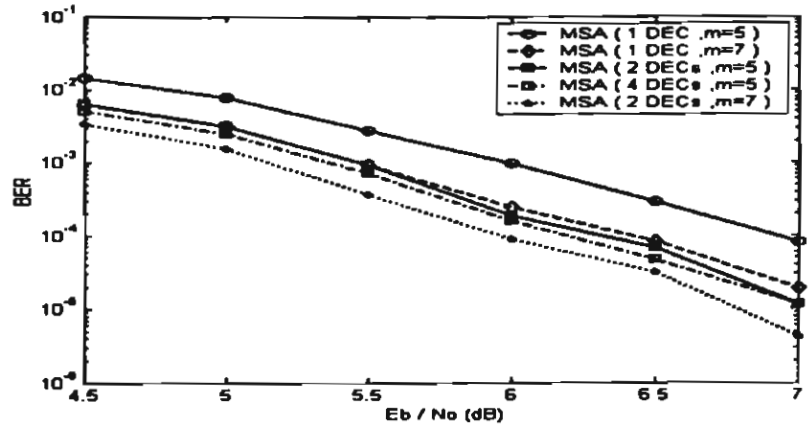
ภาพที่ 4.12 เป็นการทดสอบประสิทธิภาพของ BER เมื่อเปลี่ยนค่า E_b / N_o ในช่วง 4.5-7.0 dB จะเห็นว่าค่า BER ในการถอดรหัสเมื่อใช้วิธีการถอดรหัสแบบขนานสำหรับ MSA มีค่าต่ำกว่า BER ของระบบ MSA เดิมซึ่งใช้ตัวประมวลผลตัวเดียว อย่างไรก็ตามเมื่อเพิ่มจำนวนตัวประมวลผลเป็น 3 และ 4 ตัว ไม่ช่วยให้ประสิทธิภาพในอัตราความผิดพลาดบิตของการถอดรหัสดีขึ้นจากเดิมมากนักในขณะที่ขนาดของ Z_1 ใหญ่มาก

ภาพที่ 4.13 เป็นการทดสอบประสิทธิภาพของ BER เมื่อค่าความยาวคอนสเตรนท์ของตัวเข้ารหัสเป็น $m=7$ Generator matrices $G_1=10111101$, $G_2=11001011$ จะเห็นว่าค่า BER ในการถอดรหัสเมื่อใช้วิธีการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ ที่ค่า $m=5$ สามารถให้ค่า BER ต่ำกว่าค่า BER ที่ได้จากการถอดรหัสระบบ MSA เดิมซึ่งใช้ค่าความยาวคอนสเตรนท์มากกว่า($m=7$)

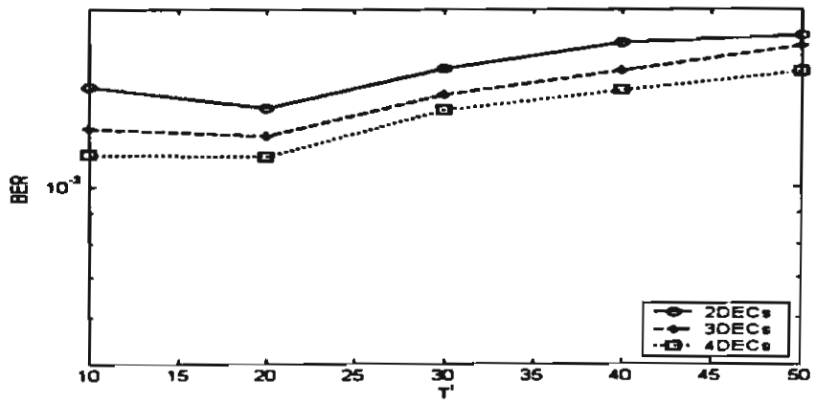
ภาพที่ 4.14 เป็นการทดสอบประสิทธิภาพของ BER ที่ค่า T' ต่าง ๆ เมื่อทดสอบในขณะที่ขนาดของสแตคหลัก $Z_1=75$ และใช้ $m=7, K=120$ จะเห็นว่าค่า T' ที่เหมาะสมคือ $T'=20$ ซึ่งให้ค่า BER ต่ำที่สุดในทุกกรณีของการถอดรหัสแบบขนานเมื่อใช้จำนวนตัวประมวลผลต่างกัน และเมื่อทดสอบที่ค่า E_b / N_o ต่าง ๆกัน ดังภาพที่ 4.15 ผลที่ได้แสดงให้เห็นว่าเมื่อเพิ่มค่า m และ K ในขณะที่ลดขนาดของ Z_1 ลงจาก 150 เป็น 75 ตัวประมวลผลที่เพิ่มขึ้นสามารถปรับปรุงประสิทธิภาพของการถอดรหัสในอัตราความผิดพลาดบิตได้เพิ่มขึ้นเล็กน้อยเมื่อเทียบกับรภาพที่ 4.12



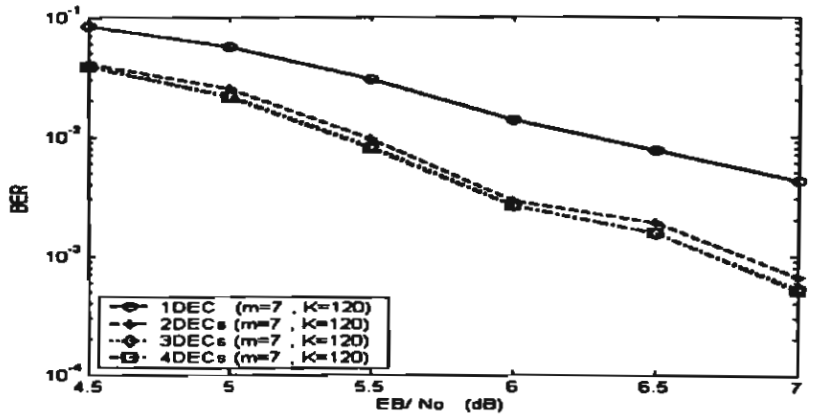
ภาพที่ 4.12 ค่า BER ที่ค่า E_b / N_o ต่างๆ ($m=5, K=60, Z_1=150, T'=50$)



ภาพที่ 4.13 ค่า BER ที่ค่า E_b / N_o ต่างๆ ($m = 5$, $m = 7$ $K = 60$, $Z_1 = 150$, $T' = 50$)



ภาพที่ 4.14 ค่า BER กับ T' เมื่อใช้ ค่าของ $m = 7$ และ $K = 120$ $Z_1 = 75$



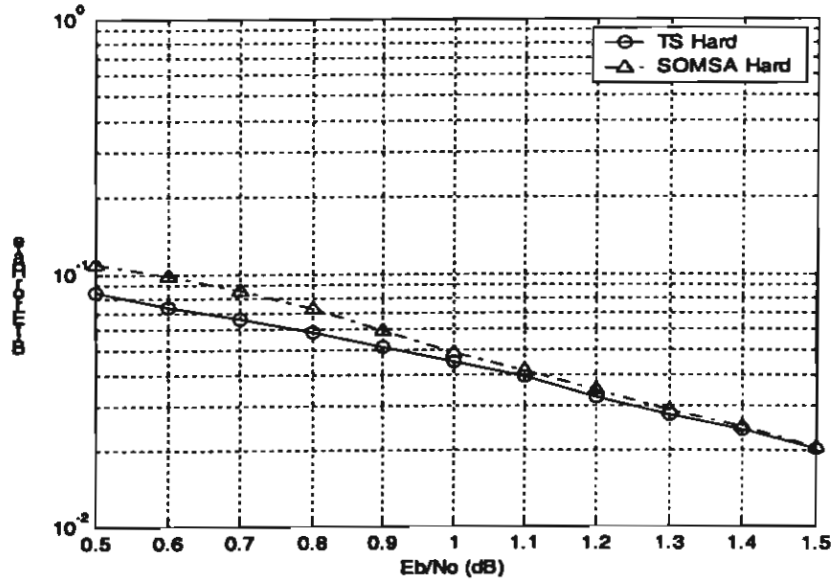
ภาพที่ 4.15 ค่า BER ที่ค่า E_b / N_o ต่างๆ ($m = 7$, $K = 120$, $Z_1 = 75$, $T' = 20$)

4.4 ผลการทดสอบประสิทธิภาพการถอดรหัสแบบ MSA ในรหัสแบบรีเคอร์ซีพซิสเต็มเมติกคอนโวลูชัน

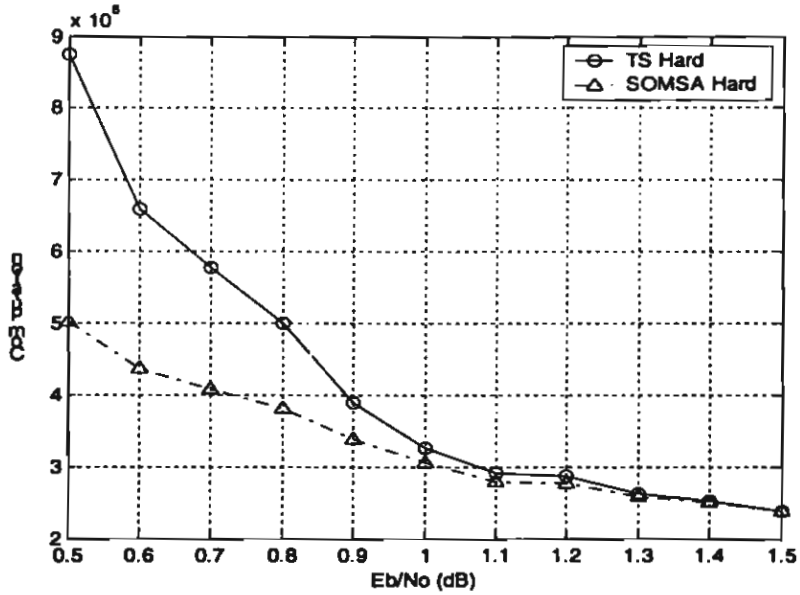
หัวข้อนี้เป็นการแสดงผลการทดสอบประสิทธิภาพของการถอดรหัสแบบ M-MSA เปรียบเทียบกับการถอดรหัสแบบ TS ซึ่งในการทดสอบจะใช้แบบจำลองระบบที่มีการเข้ารหัสสัญญาณแบบ RSC ผ่านช่องสื่อสารแบบ AWGN โดยใช้โปรแกรมคอมพิวเตอร์ภาษาซีเพื่อทำการจำลองตัวเข้ารหัส ตัวถอดรหัส และช่องสื่อสารในระบบปฏิบัติการลินุกซ์ พารามิเตอร์ที่สำคัญซึ่งใช้ในการทดสอบมีดังต่อไปนี้

- จำนวนข้อมูลที่ส่ง คือ 1,000,000 บิต โดยแบ่งออกเป็นบล็อก บล็อกละ 100 บิต
- เข้ารหัสแบบ RSC โดยมีเมตริกของตัวสร้างรหัส $G=\left[1 \quad \frac{67}{51}\right]$ และค่าความยาวคอนสเตรนท์เท่ากับ 5
- ใช้การตัดสินใจแบบซอฟท์ 8 ระดับ
- E_s / N_o จาก 0.5 ถึง 1.5 dB

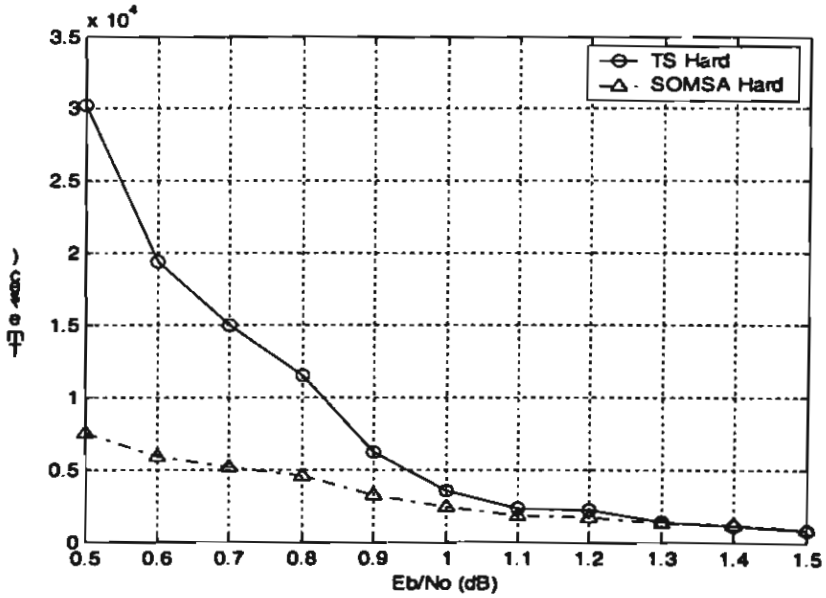
ภาพที่ 4.16 แสดงการเปรียบเทียบอัตราความผิดพลาดบิตของการถอดรหัสแบบ M-MSA และ TS จากรูปจะเห็นว่าการถอดรหัสแบบTS ให้อัตราความผิดพลาดบิตต่ำกว่าการถอดรหัสแบบ M-MSA เล็กน้อย ในช่วงที่ค่า E_s / N_o ต่ำๆ และจะค่อยๆ ใกล้เคียงกันเมื่อ E_s / N_o มีค่ามากขึ้น แต่เมื่อพิจารณาทดสอบเปรียบเทียบในจำนวนรอบของการคำนวณในการถอดรหัส และเวลาในการถอดรหัส ดังแสดงในภาพที่ 4.17 และ 4.18 พบว่าการถอดรหัสแบบ M-MSA มีจำนวนรอบของการคำนวณและเวลาในการถอดที่น้อยกว่าการถอดรหัสแบบ TS ในช่วงที่ค่า E_s / N_o ต่ำๆ และจะค่อยๆ ใกล้เคียงกันเมื่อ E_s / N_o มีค่ามากขึ้น



ภาพที่ 4.16 การเปรียบเทียบอัตราผิดพลาดบิตของการถอดรหัส



ภาพที่ 4.17 การเปรียบเทียบการคำนวณของการถอดรหัส



ภาพที่ 4.18 การเปรียบเทียบเวลาในการถอดรหัส

บทที่ 5
สรุปงานวิจัย

งานวิจัยนี้ได้นำเสนอการประยุกต์ใช้วิธีการถอดรหัสแบบขนานโดยใช้ตัวประมวลผลหลายตัวกับอัล กอริทึมการถอดรหัสแบบ MSA สำหรับรหัสคอนโวลูชัน เพื่อปรับปรุงประสิทธิภาพในอัตราความผิดพลาดบิตของการถอดรหัสแบบ MSA เดิมให้ดีขึ้น ซึ่งปัญหาในการถอดรหัสแบบ MSA เดิมนั้น คือ บล็อกที่มีปริมาณของสัญญาณรบกวนมากไม่สามารถสิ้นสุดกระบวนการถอดรหัสในสแตคหลักได้และเมื่อทำการย้ายโนดบางส่วนในสแตคหลักมาทำการถอดรหัสต่อในสแตครอง โอกาสที่การถอดรหัสจะสามารถกลับมาสิ้นสุดที่แสตคหลักนั้นมีค่าน้อยมาก ดังนั้นการถอดรหัสมักจะสิ้นสุดโดยค่าของ C_{lim} ซึ่งให้ค่าอัตราความผิดพลาดบิตสูง ในงานวิจัยนี้ได้ทำการประยุกต์การถอดรหัสแบบขนานกับ MSA เพื่อปรับปรุงประสิทธิภาพในอัตราความผิดพลาดบิตของ MSA เดิมให้ดีขึ้น ที่ค่า C_{lim} และปริมาณหน่วยความจำที่เท่ากัน ผลการจำลองด้วยคอมพิวเตอร์ ปรากฏว่าการถอดรหัสแบบขนานโดยใช้ตัวประมวลผลหลายตัวสามารถทำให้ค่าอัตราความผิดพลาดบิตใน MSA ลดลงได้อย่างมาก

ต่อมาได้ทำการประยุกต์วิธีการการถอดรหัสแบบหลายสแตคชนิดดัดแปลงเพื่อใช้ถอดรหัสสัญญาณแบบรีเดอร์ซีพซิสเต็มมาติกคอนโวลูชัน โดยการประยุกต์วิธีการถอดรหัสแบบทวินสแตคมาใช้เพื่อสร้างผลลัพธ์แบบฮาร์ดและผลลัพธ์แบบซอฟท์ ผลการทดสอบประสิทธิภาพในอัลกอริทึมการถอดรหัสแบบหลายสแตคชนิดดัดแปลง พบว่า สามารถที่จะให้อัตราความผิดพลาดบิตที่ใกล้เคียงกับกรถอดรหัสแบบทวินสแตค แต่มีข้อที่น่าสนใจคือ เวลาที่ใช้ในการถอดรหัสและจำนวนรอบในการคำนวณในกระบวนการถอดรหัส โดยเฉลี่ยมีค่าลดลงอย่างมากสำหรับค่า E_b/N_o ต่ำๆ ปัจจุบันงานวิจัยนี้กำลังอยู่ในระหว่างการดำเนินการสร้างผลลัพธ์แบบซอฟท์โดยใช้ผลลัพธ์แบบฮาร์ดที่ได้ เพื่อที่จะนำไปประยุกต์ใช้ในวิธีการถอดรหัสแบบวนซ้ำสำหรับถอดรหัสแบบเทอร์โบต่อไป

เอกสารอ้างอิง

Forney, G.D. 1974. Convolutional Codes III: Sequential Decoding. Inf. Control, Vol. 25, pp. 267-297

Lin, S., and Costello, D. J. Jr. 1983. Error Control Coding: Fundamentals and Applications. Prentice-Hall

Haccoun, D., and Begin, G. 1989. High-rate punctured convolutional codes for Viterbi and sequential decoding. IEEE Trans. Commun. Vol. 37 No. 11, pp. 1113-1125

Forney, Jr., G.D., and Bower, E.K. 1971. A High-Speed Sequential Decoder Prototype Design and Test. IEEE Transaction on Communication Technology, COM-19(5), 821-835.

Belanger, N., Haccoun, D., andSavaria, Y. 1994.A Multiprocessor Architecture for Multiple Path Sequential Decoders. IEEE Transactions on Communications, 42(2/3/4), 951-957.

Chivillat, P.R., and Costello, Jr., D.J. 1976. A Multiple Stack Algorithm for Erasurefree Decoding of Convolutional Codes, IEEE Transactions on Commun., COM-25(12), 1460-1470

Jelinek, F. 1969 A fast sequential decoding using a stack, IBM J. Res. Dev., Vol.13, pp. 675-685

Lin, Y., and Tu, S.H. 1997. Modified multiple stack algorithm for decoding convolutional codes. IEE Pro.-Commun, Vol. 44, No. 4, August, 221-228.

Kaiping, L., and Samir, K. 1999. A Bidirectional Multiple Stack Algorithm. IEEE Transactions on Communications, Vol. 47, No. 1, January, 6-9.

ภาคผนวก

Parallel Decoding Scheme for Multiple Stack Algorithm

Journal:	*IEE Proc. Communications*
Manuscript ID:	COM-2005-0036.R1
Manuscript Type:	Research Paper
Date Submitted by the Author:	25-May-2005
Keyword:	CHANNEL CODING, CONVOLUTIONAL CODES, MULTIPLE STACK ALGORITHM

Parallel Decoding Scheme for Multiple Stack Algorithm

Virasit Imtawil*, Ph.D. and Wanlop Surakampontorn**, Ph.D.

*Department of Electrical Engineering, Faculty of Engineering,

Khon Kaen University, Khonkaen, THAILAND 40002

Corresponding author email address: virasit@kku.ac.th

Department of Electronics, Faculty of Engineering,

King Mongkut's Institute of Technology Ladkrabang, Bangkok, THAILAND 10520

Abstract In order to improve the error performance particularly in a very noisy block which requires excessive tree searches, a parallel decoding scheme for a multiple stack algorithm (MSA) is proposed in this paper. In this scheme, apart from the main decoder working on the main stack, a scalable set of multiple decoders working in parallel on their multiple stacks are incorporated. All the decoders are working almost independently with the help of the control processor, where the decoding process and the termination rules are outlined. Two cases, 4-processor MSA and 16-processor MSA, are employed as examples to investigate the performance of the proposed scheme. Comparing with the conventional MSA, extensive computer simulations show that the bit error probabilities (BER) are significantly improved.

Indexing terms: Multiple stack algorithm; Parallel decoding; Convolutional codes

1. Introduction

Convolutional codes are widely used for error correction in both satellite and terrestrial communications. There are two main techniques for decoding convolutional codes namely the Viterbi algorithm (VA) and the sequential decoding [1]. For the sequential decoding, although it is not optimal, arbitrarily low error probabilities can be achieved since its decoding effort does not depend on the code constraint length [2-3]. In addition, increasing the code constraint length means to improve the error correcting capability of the code. However, the most undesirable attribute of the sequential decoding is that noisy blocks may require a large number of

computations and decoding times can occasionally exceed the computational limit (C_{lim}), causing the information to be lost or erased (marked as an erasure). Chevillat and Costello [4] proposed an algorithm for erasurefree sequential decoding called the multi stack algorithm (MSA). The MSA is developed from the single stack algorithm (SSA) [5] by splitting the stack into many stacks, where the first stack is usually large and the second and higher order stacks with typically all the same sizes are relatively small. The basic idea of the MSA is to make use of higher order stacks for blocks that require an excessive number of computations to be decoded. The MSA decoding process will be terminated when either a terminal node is found in the main stack or the computation limit C_{lim} is reached. However, decoding termination with C_{lim} usually gives very high error rate. Later Y.Lin and S.H. Tu [6] proposed a modified multiple stack algorithm that rearranged the memory used to implement stacks as a ring-like structure. The main advantage of their work is the flexibility in defining the size of the top transferred nodes between stacks (T) and the size of higher order stacks (Z) to be as large as desired in order to improve the bit error performance. However, any increase in T or Z requires more computation and thereby a long delay may be occurred. Kallel and Li proposed a bidirectional multiple stack algorithm (BMSA) in 1999 [7]. Their technique was to apply the bi-directional tree search to the MSA. It has been shown that the performance of the conventional MSA is significantly improved with the use of a bidirectional decoding scheme.

Recently, in order to achieve high-throughput rate or to reduce decoding delay, parallel decoding has gained some interest from many researchers in the coding community [8-13]. In this study, a parallel decoding scheme for the MSA is proposed. The main idea is to exploit scalable multiple decoders working in parallel to search the correct path in the code tree. The scheme comprises of a main decoder extending paths on the main stack and many scalable sub-decoders extending paths on the sub-stacks. All the decoders work independently in parallel with the help of the control processor. It is found by extensive computer simulations that the error performance of our proposed scheme is better than that of the conventional MSA. The paper is organized as follows. Background on the MSA decoding is briefly discussed in Section 2. In Section 3, the proposed parallel decoding scheme for the MSA is described, where description of the scheme,

decoding process and the termination rules are given. In Section 4, performance of the proposed scheme is presented by computer simulation results in comparison with the conventional MSA. Finally, Section 5 is the conclusion of this paper.

2. Background on the MSA

In the MSA, when the first stack fills up, the top T nodes with best metrics are transferred to a second stack. Decoding then continues in the second stack using only these T transferred nodes. If a path in the second stack reaches a terminal node before the stack fills up, the path is stored in a special register called the *tentative decision* (TD). The decoder then deletes the remaining nodes in the second stack and returns to the first stack where decoding continues. Since T nodes were removed there are now exactly T free places in the first stack. If the decoder reaches a terminal node before the first stack fills up again, the metric of the new path is compared to that of the TD. The path with the better metric is retained and becomes the final decoding decision. It can be shown that a decision made in this way is as good as that made by the SSA with an arbitrarily large stack. However, if the first stack fills up again before the end of the tree is reached, a new second stack is formed by transferring from the top T nodes of the first stack. Additional stacks are formed in a similar manner until a path to a terminal node is found. The decoder compares this path with the TD and retains the path with the best metric. The rest of the nodes in the current stack are then deleted and decoding proceeds in the previous stack. Occasionally, the decoder may not reach a terminal node in the first stack after the number of node extensions (computations) exceeds the computational limit C_{lim}. In this case, the TD becomes the final decoding decision. To guarantee erasurefree decoding C_{lim} must be carefully chosen. For the case $T = 1$ and a rate of $1/w$ convolutional code (where w is the codeword length in bits) with the constraint length ν the value of C_{lim} must be at least equal to the critical value C_{crit} given by

$$C_{crit} = \sum_{i=1}^{k-1} (Z_i - 1) + 2(\nu - 1) \qquad (1)$$

where Z_i is the size of stack i, and k the number of tree branches without tail (the information frame size). The decoding termination with C_{lim} in the MSA usually happens with very noisy blocks that require excessive tree searches. The decoded bit error rate in this case is very high. Although, the error performance of the MSA can be improved by increasing the value of C_{lim} or increasing the size of the first stack, this choice may not be possible in some applications where decoding delays are not acceptable or the memory storage is limited with a single-processor decoder. In such situations we propose the use of a Parallel decoding process.

3. A Parallel Decoding Scheme for the MSA

Our main goal is to apply a scalable set of multiple decoders concurrently searching the correct path in the code tree. Therefore, the probability of finding the correct path is raised up with our proposed scheme.

3.1 Description of the proposed scheme

The proposed parallel decoding scheme for the MSA is shown in Fig. 1. The arrangement and the processing of the scheme are as follows.

(1) It consists of the following components: the main (first) decoder (DEC#1), scalable multiple sub-decoders (DEC#2, DEC#3,..., DEC#n), the control processor (CP), the main (first) stack (Z_1), $n-1$ primary sub-stacks (Z_2), which equal to the number of sub-decoders and several secondary sub-stacks (Z).

(2) The main decoder extends paths of the main stack only based on the SSA.

(3) Each sub-decoder is identical. One sub-decoder extends paths of its primary sub-stack and secondary sub-stacks based on the conventional MSA.

(4) The CP is used for the following tasks: (a) Stop the decoding when a decoding termination rule is found (b) notify the main decoder whenever there is a sub-decoder available, compare and update the best path with best metric as the TD in a special register whenever a decoder finds a terminal node

(5) The main stack Z_1 is usually large. The primary sub-stacks Z_2 and the secondary sub-stacks Z are very small and all have equal size.

3.2 Decoding of a data block

The decoding begins with placing the root node in the main stack where the DEC#1 extends paths is based on the SSA. When the first stack is full, the CP is informed by the DEC#1. The CP then checks if there is one sub-decoder available. If it finds one, the top T nodes of the main stack are transferred to the corresponding primary sub-stack where the available sub-decoder will extend paths based on the conventional MSA. After that the main decoder and the sub-decoder will simultaneously extend the top paths of their own stacks. This operation is different from the conventional MSA in which the node extensions in the first stack cannot be continued when the first stack is full. With this scheme, it is the task of the sub-decoder to extend those transferred nodes of its own sub-stacks. Then the first decoder can continue the node extension in the first stack straight away without having to wait for the sub-stacks to be deleted. The node extension in the first stack cannot continue only when there is no sub-decoder available. The probability of the first stack to be suspended is decreased with the number of sub-decoders increases. The advantage of this scheme is that whenever the first stack is full, some top nodes are transferred to a sub-stack as long as there is at least one sub-decoder available. All the decoders are working in parallel on their own stacks to search for the most likely path in the code tree.

For a sub-decoder, a new secondary sub-stack will be created whenever the primary sub-stack is full. Then the top T nodes will be transferred to the current secondary sub-stack as in the conventional MSA. The decoding will proceed this way until a decoding termination rule is found.

The following example shows the contents of some stacks during the decoding in progress. Assuming that two sub-decoders ($n = 3$) are employed, Fig. 2(a) shows the stacks contents at the beginning stage at which only the main decoder is busy with the node extensions in the main stack while the two sub-decoders are standstill. When the first stack is full (see Fig. 2(b)), the top T nodes are transferred to the primary sub-stack of DEC#2 (see Fig. 3(a)). Then DEC#1 and DEC#2 will extend the top nodes of their stacks simultaneously while DEC#3 is still doing nothing (see

Fig. 3(a)). Fig. 3(b) shows the situation when the first stack is full for the second time. At this point the top T nodes are transferred from the first stack to the primary sub-stack of DEC#3 (see Fig. 4(a)), after that all the decoders are working in parallel.

Fig. 4(b) shows the situation when the first stack and the primary sub-stack of DEC#2 are full. At this point the first stack is in suspension stage as there is no sub-decoder available, DEC#2 creates a secondary sub-stack and then transfers the top T nodes from the primary sub-stack to the current secondary sub-stack and DEC#3 continues extending paths of its primary sub-stack (see Fig. 5(a)).

Fig. 5(b) shows the situation when a terminal node is found in the primary sub-stack of DEC#3. This terminal node will be kept in a special register as the TD. Then the primary sub-stack of DEC#3 is deleted and DEC#3 returns to standstill (see Fig. 6(a)). After that DEC#1 is informed of the availability of DEC#3 by the CP and then the top T nodes are transferred to the primary sub-stack of DEC#3 for the second time. All the decoders are now working in parallel again (see Fig. 6(b)). The decoding process continues in this way until one of the termination rules is found.

3.3 Termination rules

The decoding process will be terminated when one of the following rules is found:

(1) The computational limit, C_{lim}, is reached (same as the conventional MSA).

(2) If a terminal node is found in the first stack **and** all the sub-decoders return to be standstill.

This decoding process is different from the conventional MSA in which the decoding is terminated when a terminal is found in the first stack. The reason of not stopping the decoding straight away when a terminal node is found in the first stack is that the correct path may be hidden in one of the sub-stacks where a sub-decoder is working on.

3.4 Discussion

Note that in our process the CP allows all the decoders to work at their full speeds without having to exchange information with each other and the usual communications bottleneck is thereby kept

at minimum. The advantage of having several sub-decoders working on the sub-stacks can be summarized as follows.

(1) The probability of finding the correct path is raised as the availability of sub-decoders helps the main decoder extend the nodes of sub-stacks allowing the main decoder to continue the node extensions in the first stack straight away without having to wait for the deletion of the primary sub-stacks as happened in the conventional MSA

(2) More terminal nodes can be found as several paths are extended simultaneously. Therefore better TD is made with the same value of C_{lim} compared to the conventional MSA.

4. Performance comparison with the MSA

By computer simulation, the performance comparison of our proposed scheme with the MSA is investigated by comparing bit error rate (BER), where 160 000 frames of length $L = 250$ were transmitted through an additive white Gaussian noise (AWGN) channel. The simulator was written in the C language and was run on a LINUX operating system. The cases of 4 and 16 were mostly investigated in this paper. The 4-processor scheme comprises of one main decoder and the other three sub-decoders. Similarly, the case of 16-processor scheme comprises of one main decoder and fifteen sub-decoders. For convenience, we name 4-processor and 16-processor cases as 4MSA and 16MSA, respectively. The code used is a nonsystematic rate $R = 1/2$ code with the code constraint length $\nu = 15$ and generators used are in octal 65231 and 43677 (optimum d_{free} code [2]). In order to make a fair comparison between the conventional MSA and our proposed scheme, the amounts of memory storage capacity of both systems are kept the same.

Fig 7 shows the BER curves as a function of the computational limit C_{lim} at the signal to noise ratio of $E_b / N_0 = 5.5\,\text{dB}$ with $Z_1 = 800, Z = 11, T = 3$ for all the three algorithms. As it can be seen from the Figure, the BER performance of 16MSA is lowest among the three. For example, at $C_{\text{lim}} = 20000$ the MSA and 4MSA reach BER values of 5.9×10^{-5} and 5.6×10^{-5} respectively, while the 16MSA reaches a BER of value 4.1×10^{-5}. Also as expected, the value of C_{erf} (the value which guarantees erasurefree decoding) is lowest for the 16MSA. This is due to the multiple path extensions by the main decoder and sub-decoders. From the Figure, the value of C_{erf} for the

16MSA is approximately 2200, while the values of C_{crit} for the cases of 4MSA and MSA are approximately 2800 and 3000, respectively. It should be noted that the value of C_{crit} is approximately determined from the value of C_{lim} at which the BER reaches the peak. Keeping the parameters the same as those used for Fig 7 except changing Z_1 to 1100, the similar results are shown in Fig 8. The humps of the curves for the 16MSA in Fig 7 and Fig 8 can be explained as follows. When increasing C_{lim} up to C_{crit} the decoding process begins to perform erasure-free decoding. After the value of C_{lim} is further increased the decoding process is done in parallel by the main decoder and the sub-decoders. The probability of finding the correct path is then increased, resulting in a further improvement in the BER performance when compared to the BER at the value of C_{crit}.

Fig. 9 shows the BER curves with $Z=11, T=3$, the computational limit $C_{lim}=10000$ as a function of Z_1 at $E_b/N_0=5$ dB. From the Figure we can see that the 4MSA has lower BER than the MSA. The improvement in the decoded BER is more pronounced when increasing the number of sub-decoders from 3 (4MSA) to 15 (16MSA). When the first stack size increases, the decoded BER of the three algorithms tend to be closer. This is because more frames finished the decoding in the first stack before it fills up for the first time when Z_1 is increased. Fig. 10 shows BER curves as a function of Z_1 at $E_b/N_0=5.5$ dB with the same parameters used for Fig. 9. The results are similar for both Figures.

Next we investigated the influence of size of Z on the decoded BER at $E_b/N_0=5.5$ dB. Fig. 11 shows the BER curves with $Z_1=700, T=3, C_{lim}=10000$ for various values of Z. It is seen from the Figure that no significant influence of Z on the BER for all the MSA, the 4MSA and the 16MSA. Similar results are shown in Fig. 12 for the BER curves as a function of Z when increasing Z_1 from 700 to 800.

Fig 13 displays the BER curves as a function of T at $E_b/N_0=5.5$ dB with $Z_1=800, Z=11, C_{lim}=10000$. As shown, an increase in T has no noticeable improvement in the BER for the MSA and 4MSA while the BER performance of the 16MSA is somewhat improved when increasing T. This is due to larger value of T and more sub-decoders allow the main stack

to continue the node extension more frequently after it fills up. For example, the BER reduces from 4.3×10^{-5} to 3.7×10^{-5} when changing T from 2 to 5. Similar results are shown in Fig 14 for the BER curves as a function of T when changing Z from 11 to 30. As expected, the BER performance of the 16MSA is the best among the three.

Next we considered the influences of the number of the top nodes transferred from Z_1 to Z_2 and the size of Z_2. Let T' be the number of the nodes transferred from Z_1 to Z_2 when Z_1 fills up. Fig 15 displays the BER performance of the 16MSA as a function of T' for various values of Z_2. The parameters used for the Figure are $Z_1 = 800, Z = 20$, $T = 3$, and $C_{lim} = 10000$ at $E_b / N_0 = 5.5$ dB. We can see that any increase in Z_2 obtains the improvement in the BER performance. This is because the larger value of Z_2 allows more node extensions in the primary sub-stack thereby raising the probability of finding a terminal node in the primary sub-stack. When a terminal is found in a primary sub-stack, this sub-stack will be deleted and then the top nodes from the main stack can be transferred. This allows the node extension in the main stack to be continued therefore the probability of finding the correct path in the main stack is raised. The Figure also shows that any increase in T' tends to improve the BER for a given value of Z_2

Figure 16 shows the comparison of the BER performance for various values of signal to noise ratio (E_b / N_0) for the MSA, the 4MSA, the 8MSA, the 16MSA, and the 32MSA. The parameters used are $Z_1 = 1000, Z = 20, T = 8$ and $C_{lim} = 20000$ for all the curves and $Z_2 = 300, T' = 50$ for the 4MSA, the 8MSA, the 16MSA, and the 32MSA. As expected, the BER performance is improved with our proposed parallel decoding scheme. For example, at signal to noise ratio of 5 dB, the BER of the MSA is 2.5×10^{-4}, the BER of the 4-MSA is 1.8×10^{-4}, the BER of the 8-MSA is 1.5×10^{-4}, the BER of the 16-MSA is 8×10^{-5} and the BER of the 32-MSA is 6×10^{-5}. It is clearly shown that when increasing the number of processors the BER is significantly decreased.

From the simulation results presented above, a guideline for parameter selection in parallel decoding scheme can be given as follows.

1. The value of C_{lim} should be large enough to guarantee erasure-free decoding.

2. The size of Z_1 should be made as large as possible. This depends on the available memory storage.
3. The size of Z can be independently selected. Therefore small size of Z is recommended.
4. The sizes of T and T' should be as large as possible.
5. The size of Z_2 should also be chosen as large as possible according to the results on Figure 15.

5. Conclusions

A parallel decoding scheme for the MSA is proposed and investigated in this paper. The idea of parallelism in the MSA mainly aims at the improvement of the bit error probability of the MSA for some blocks, which requires excessive searches. In the conventional MSA when the number of tree searches is limited with the value of C_{lim} but a terminal node is not yet found in the main stack, the decoder will usually give very poor error performance. In this paper we investigated the application of parallel decoding for the MSA in which multiple paths in the code tree are simultaneously extended by a set of scalable decoders. Simulation results show that the average bit error rate of the MSA is significantly improved with our proposed parallel decoding scheme. In addition, the BER performance is further improved when the number of processors is increased. Parallel decoding for MSA appears to be an interesting scheme for decoding convolution codes especially in applications where low probabilities are required.

6. Acknowledgement

The authors would like to thank Professor P.G. Farrell of the University of Lancaster for his very useful comments. Many thanks also to the reviewers for their patience and their many helpful suggestions. Lastly, the authors would like to acknowledge the Thailand Research Funds Agency for its support under the grant MRG4680124.

7. References

1. FORNEY, G.D.: 'Convolutional codes III: sequential decoding', *Inf. Control*, 1974, **25**, pp. 267-297

2. LIN, S., and COSTELLO, D. J. Jr.: 'Error control coding: fundamentals and applications' (Prentice-Hall, New Jersy, 1983)

3. HACCOUN, D., and BEGIN, G.: 'High-rate punctured convolutional codes for Viterbi and sequential decoding', *IEEE Trans. Commun.*, 1989, **37**, (11), pp. 1113-1125

4. CHEVILLAT, P.R., and COSTELLO, D.J. Jr.: 'A multiple stack algorithm for erasurefree decoding of convolutional codes', *IEEE Trans. on Commun.*, 1976, **25**, (12), pp. 1460-1470

5. JELINEK, F.: 'A fast sequential decoding using a stack', *IBM J. Res. Dev.*, 1969, **13**, pp. 675-685

6. LIN, Y., and TU, S.H.: 'Modified multiple stack algorithm for decoding convolutional codes', *IEE Pro. Commun.*, 1997, **44**, (4), pp. 221-228

7. KALLEL, S., and LI, K.: 'A bidirectional multiple stack algorithm', *IEEE Trans. on Commun.*, 1999, **47**, (1), pp. 6-9

8. FAIRHURST, G., PANG, S.L., and WAN, P.S.: 'Smart codec: an adaptive packet data link', *IEE Pro. Commun.*, 1998, **145**, (3), pp. 180-185

9. IMTAWIL, V.: 'Bidirectional sequential decoding: a multiprocessor approach', Ph.D. thesis, the University of Manchester, U.K., 1999

10. KWAK, J., and LEE, K.: 'Design of dividable interleaver for parallel decoding in turbo codes' Electronics Letters, 2002, **38**, (22), pp. 1362 – 1364

11. ZHANG, X., ZHAO, M., ZHOU, S., and WANG, J.: 'Parallel decoding of turbo product codes for high data rate communication', IEEE Vehicular Technology Conference, VTC 2003, pp. 2372 – 2375

12. UMANESAN, G., and FUJIWARA, E.: 'Parallel decoding cyclic burst error correcting codes' Proceedings of IEEE International Symposium on Information Theory, ISIT 2003, pp. 420 – 420

13. NIKARA, J., VASSILIADIS, S., TAKALA, J. and LIUHA, P.: 'Multiple-symbol parallel decoding for variable length codes' *IEEE Trans. on VLSI*, 2004, **12**, (7), pp. 676 – 685

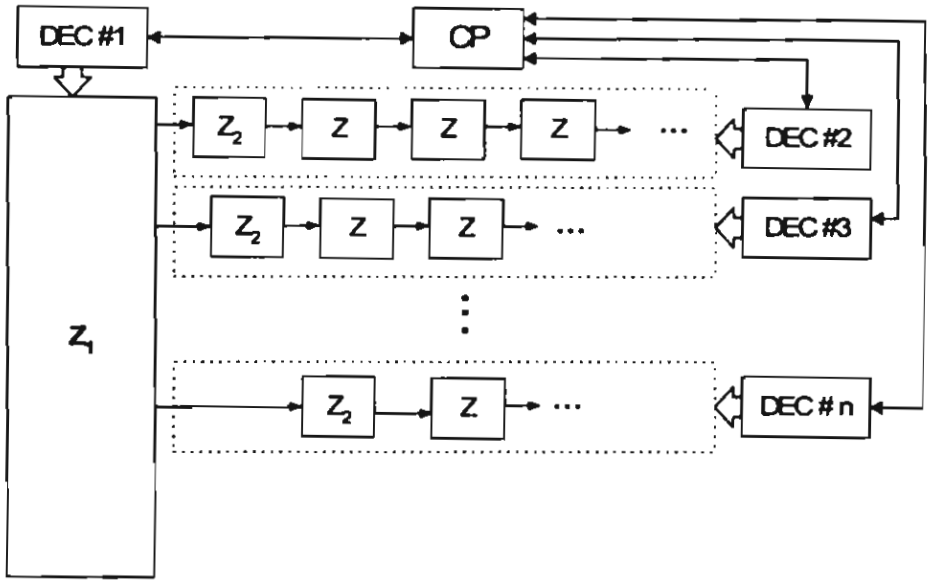


Fig. 1 *A parallel decoding scheme for the MSA*

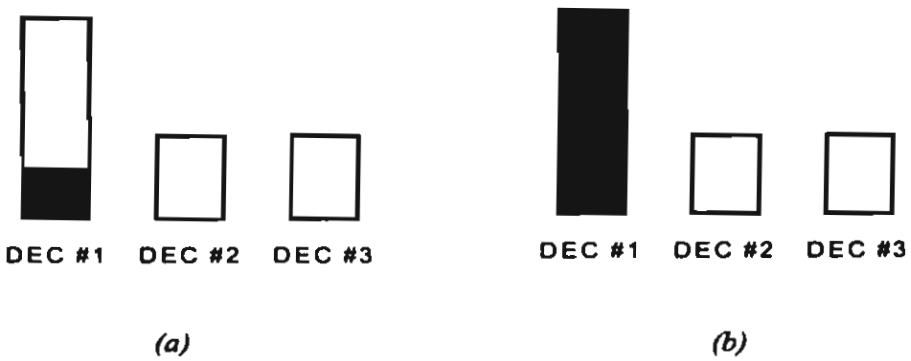


Fig. 2 *Stacks contents at the beginning stage until the first stack is full*

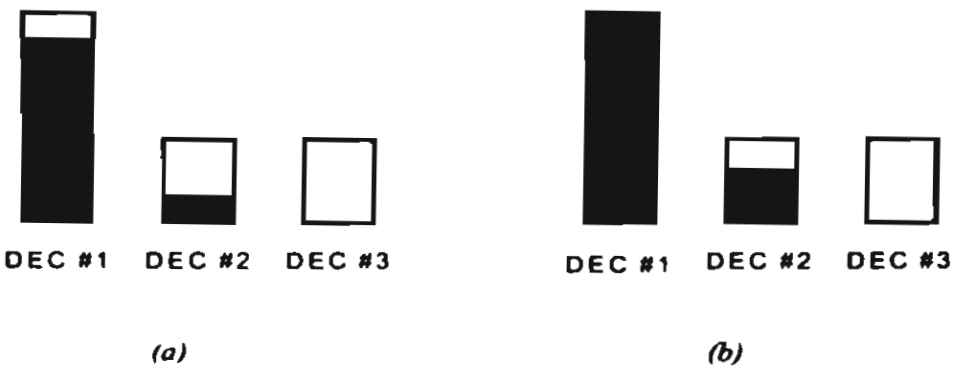


Fig. 3 *Top T nodes in the first stack are transferred to the primary sub-stack of DEC#2*

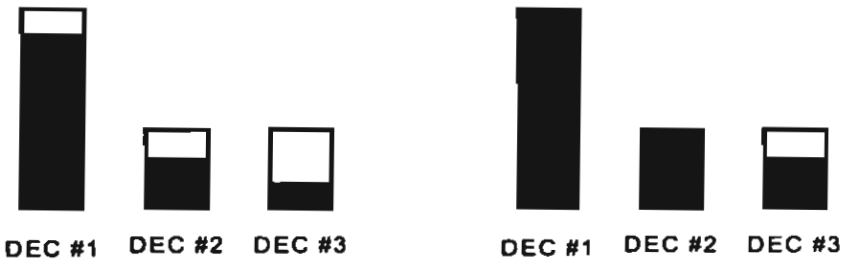


Fig. 4 *(a) All decoders are working in parallel*

(b) First stack and primary sub-stack of DEC#2 are full

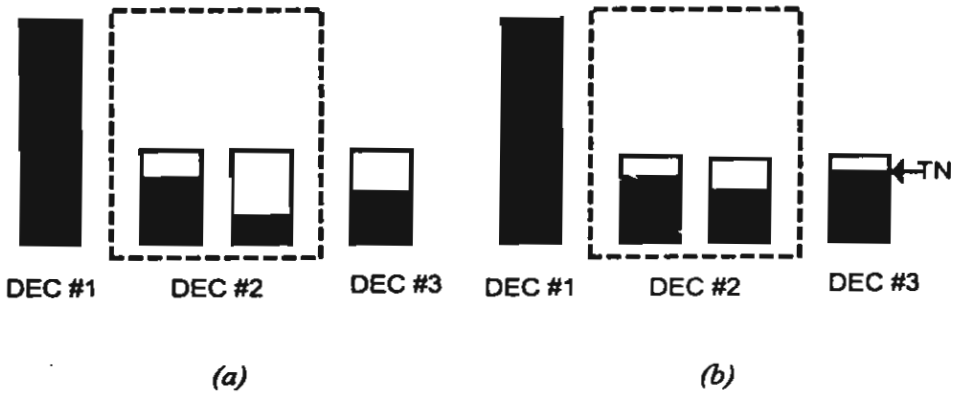


Fig. 5 *(a) First stack is suspended (b) A terminal node is found by DEC#3*

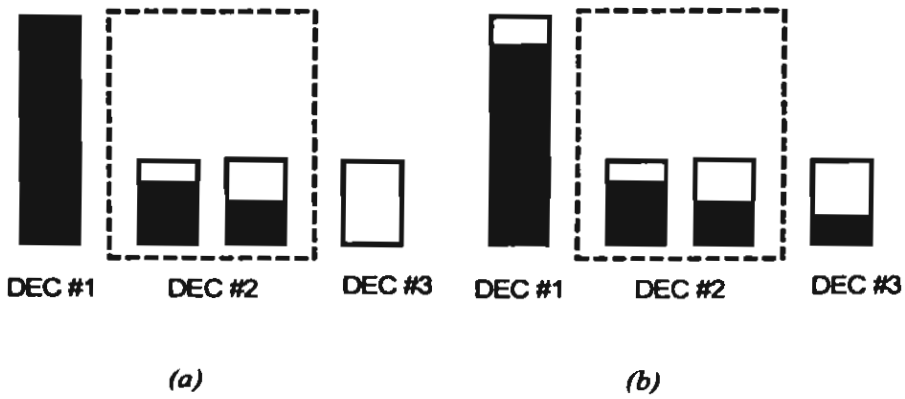


Fig. 6 *(a) The first sub-stack of DEC#3 is deleted.*

(b) All the decoders are working in parallel again

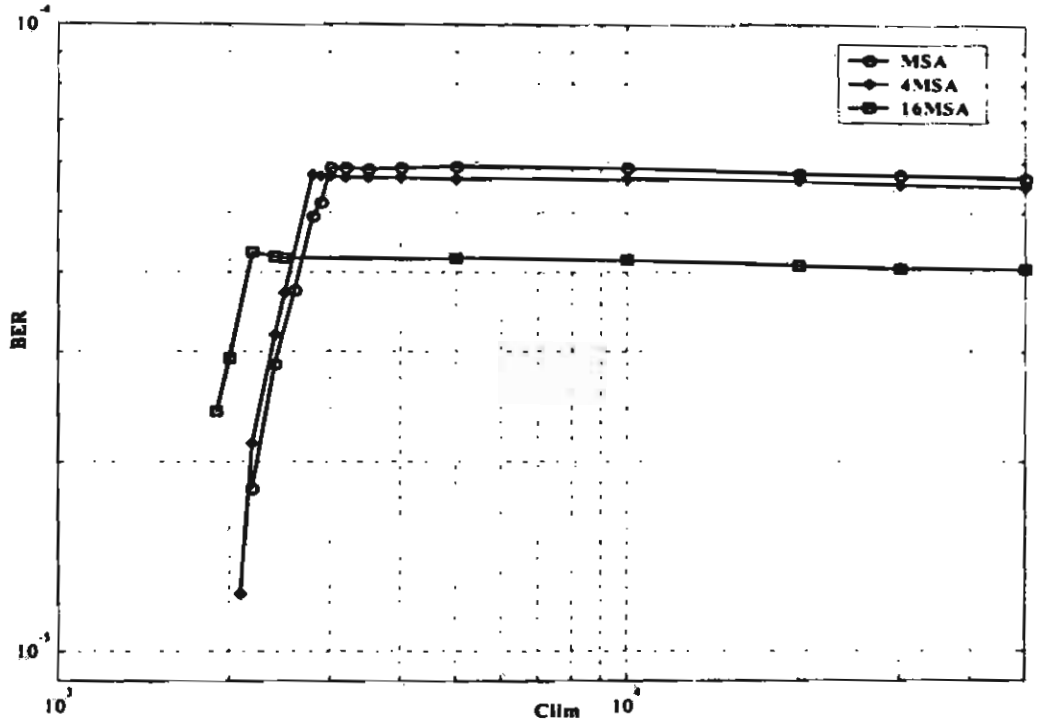


Fig. 7 *BER as a function of* C_{lim} *for* $Z_1 = 800$

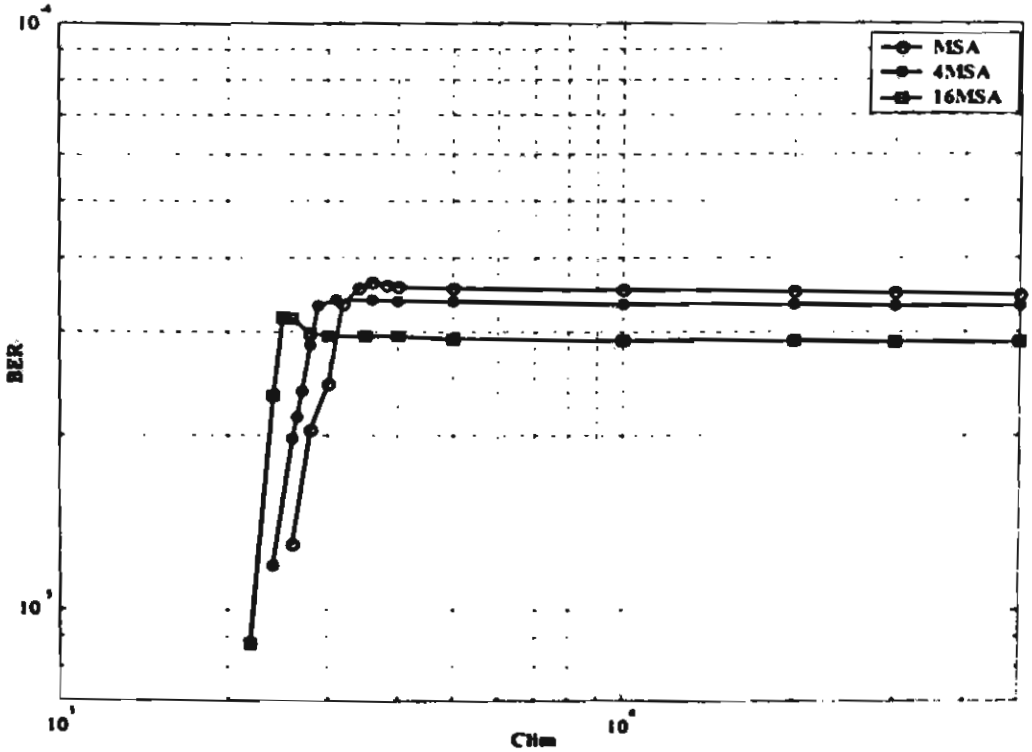


Fig. 8 *BER as a function of* C_{lim} *for* $Z_1 = 1100$

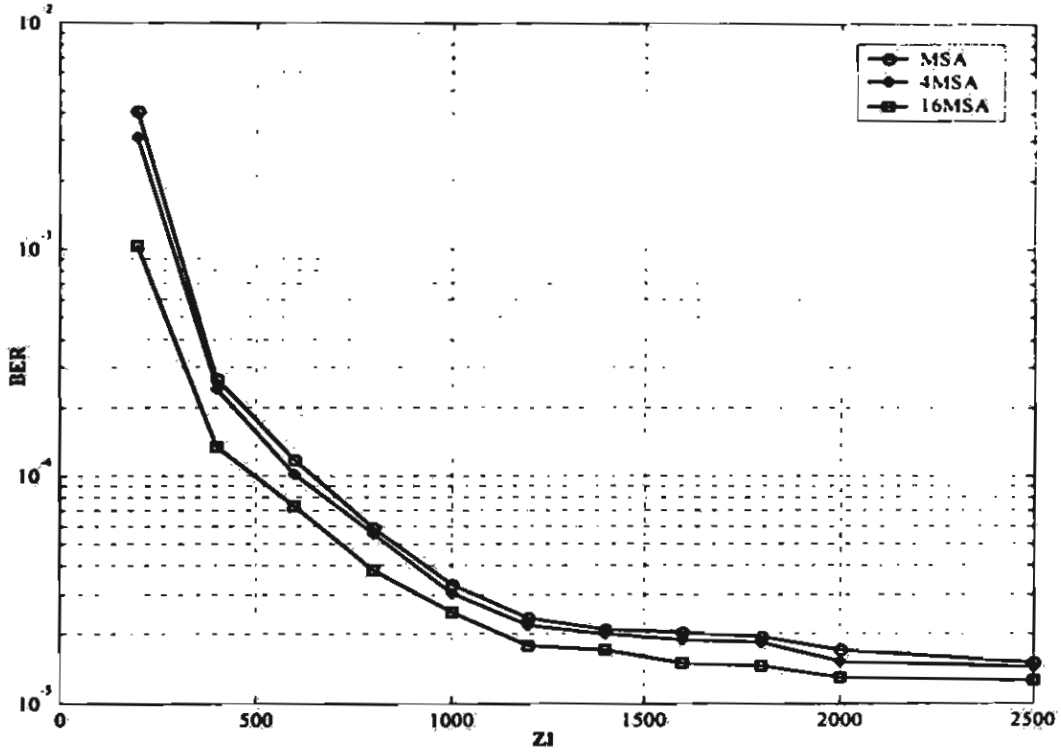


Fig. 9 *BER as a function of* Z_1 *at* $E_b / N_0 = 5$ *dB*

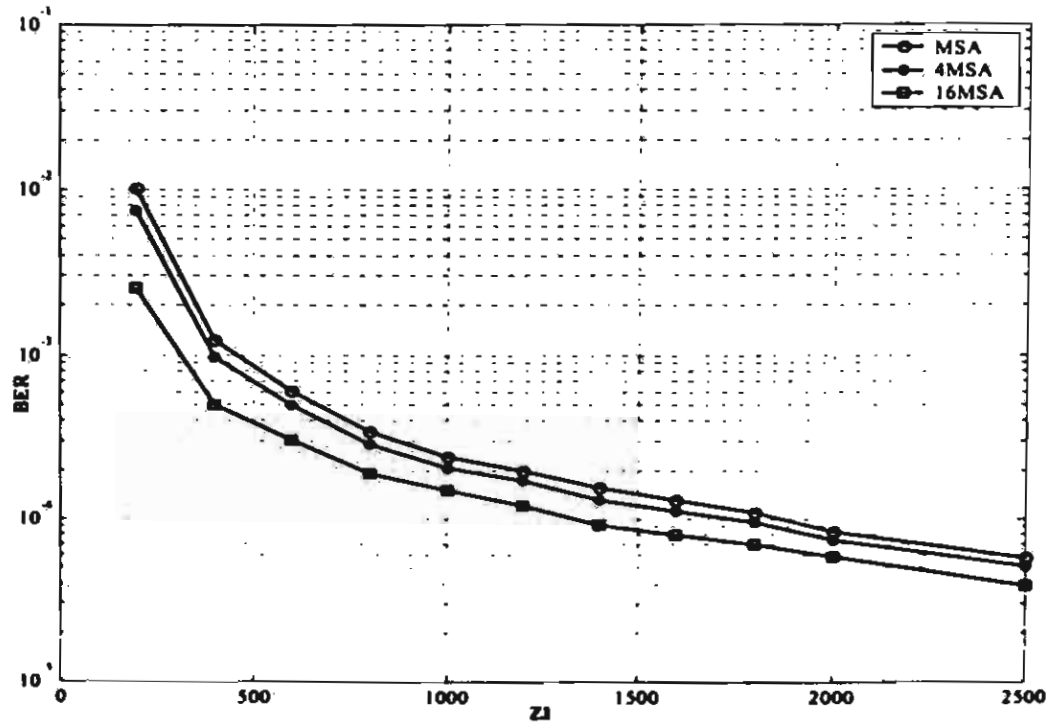


Fig. 10 *BER as a function of* Z_1 *at* $E_b / N_0 = 5.5$ *dB*

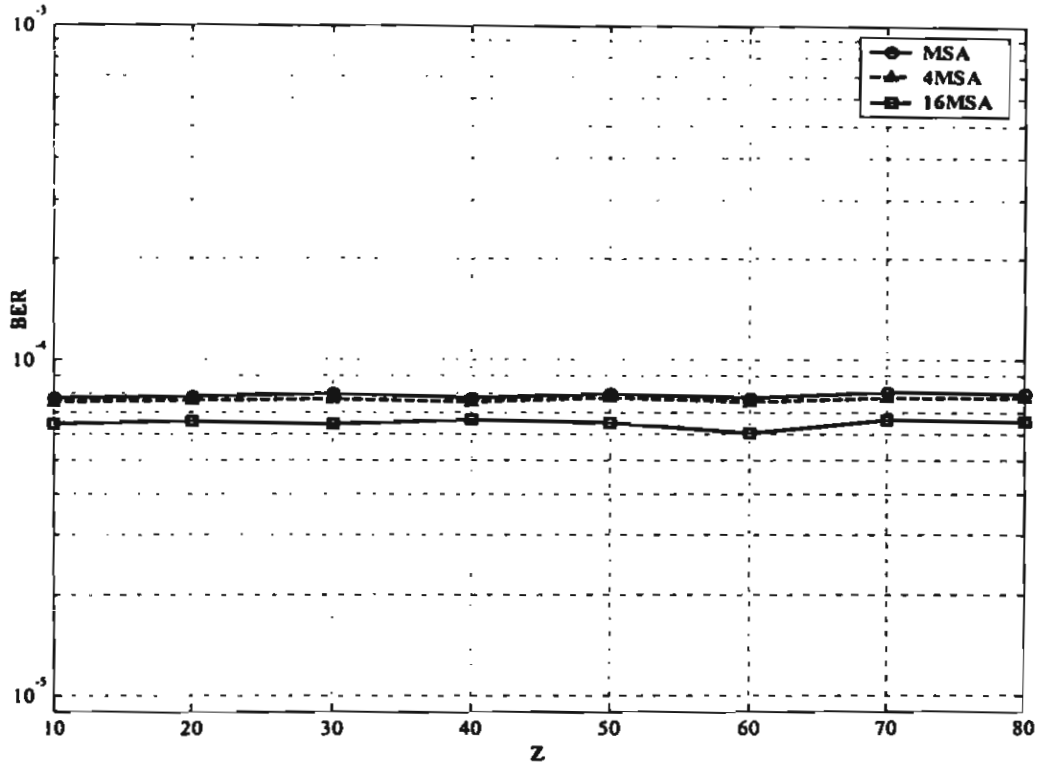


Fig. 11 *BER as a function of Z with* $Z_1 = 700$

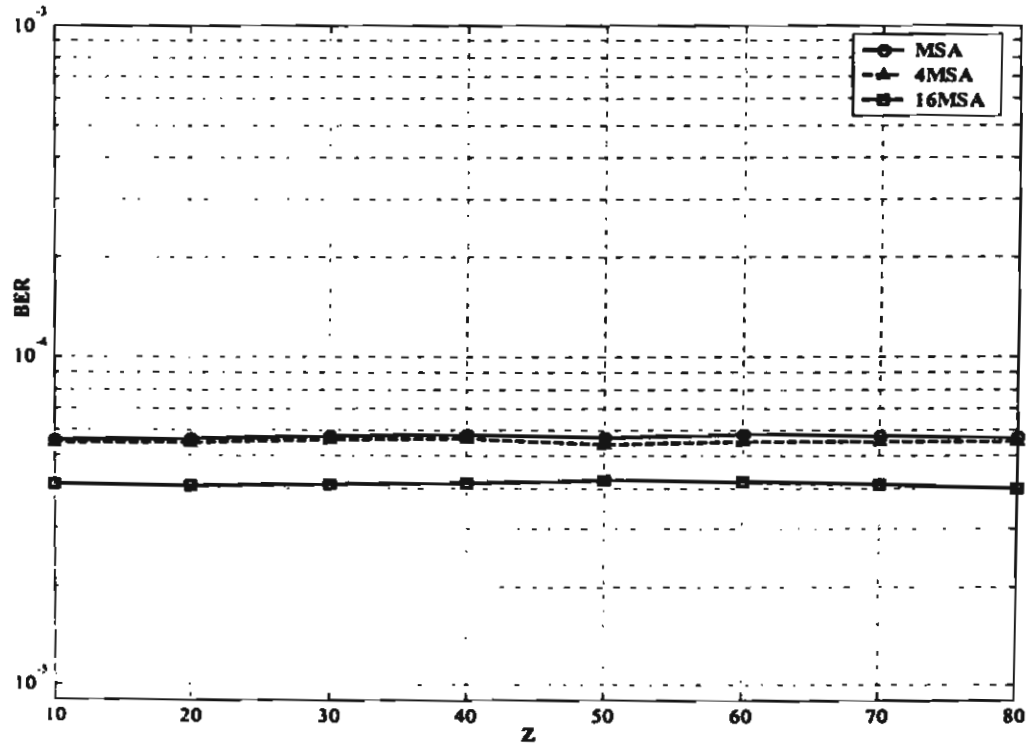


Fig. 12 *BER as a function of Z with* $Z_1 = 800$

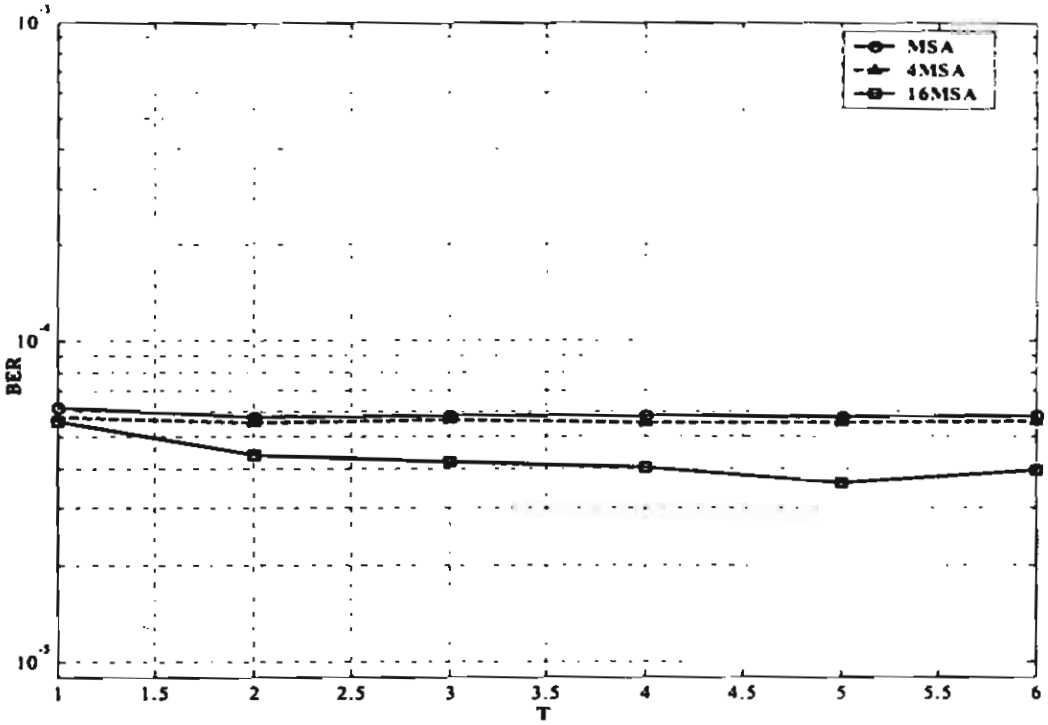


Fig. 13 *BER as a function of* T *for* $Z = 11, C_{im} = 10000$

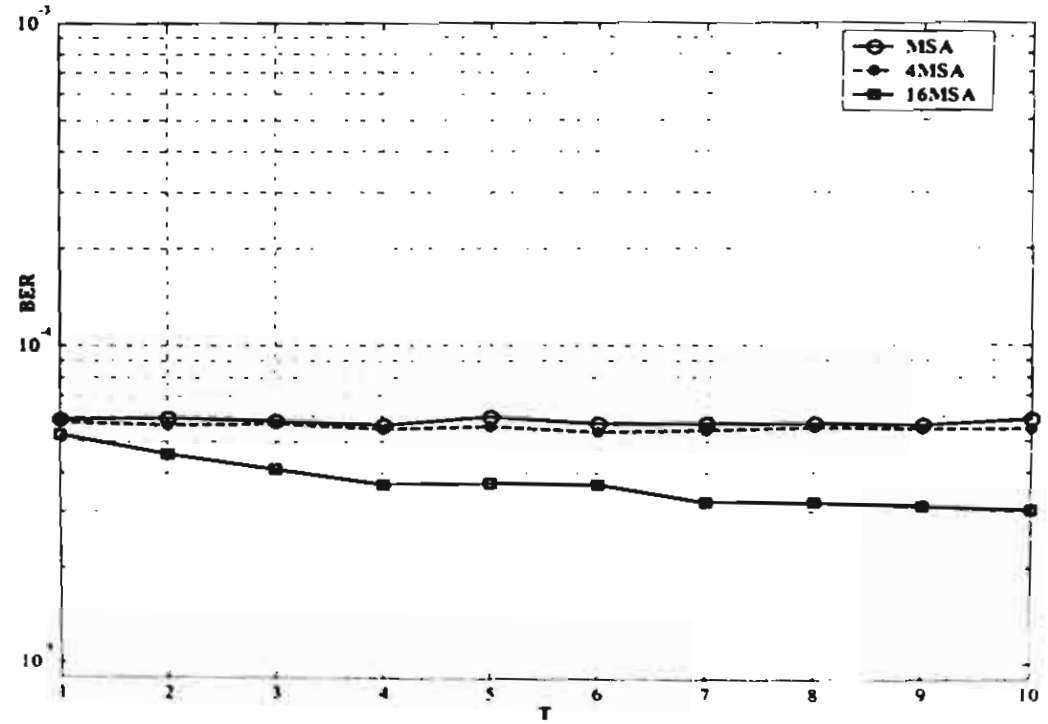


Fig. 14 *BER as a function of* T *for* $Z = 30, C_{im} = 10000$

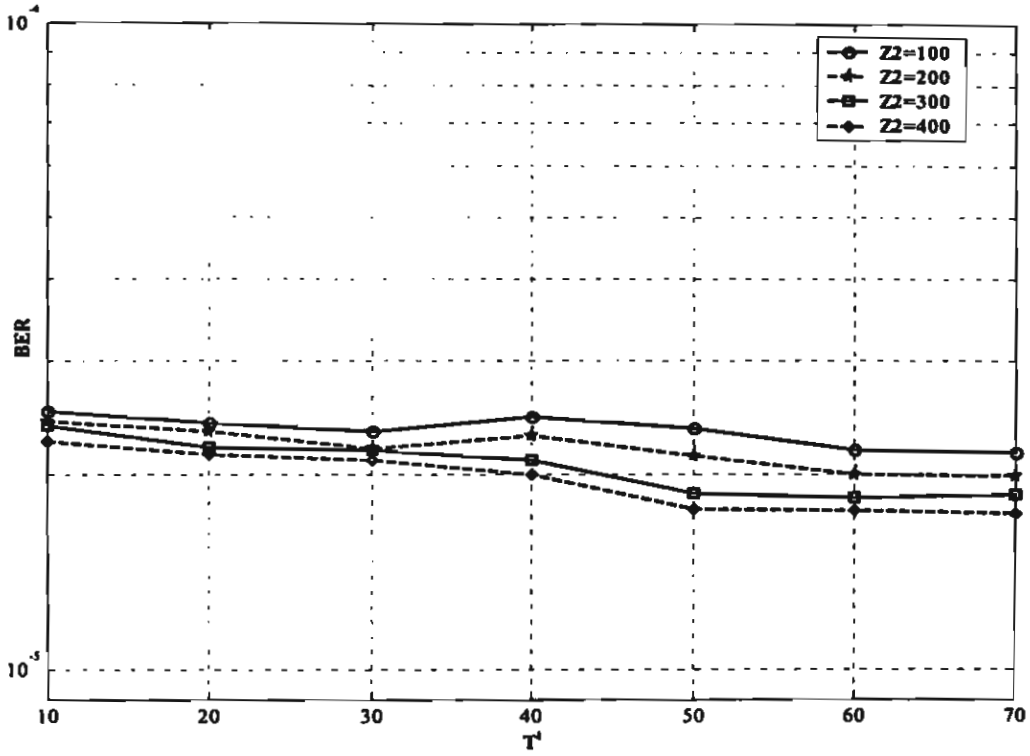


Fig. 15 *BER of the 16MSA as a function of T' for various values of* Z_2

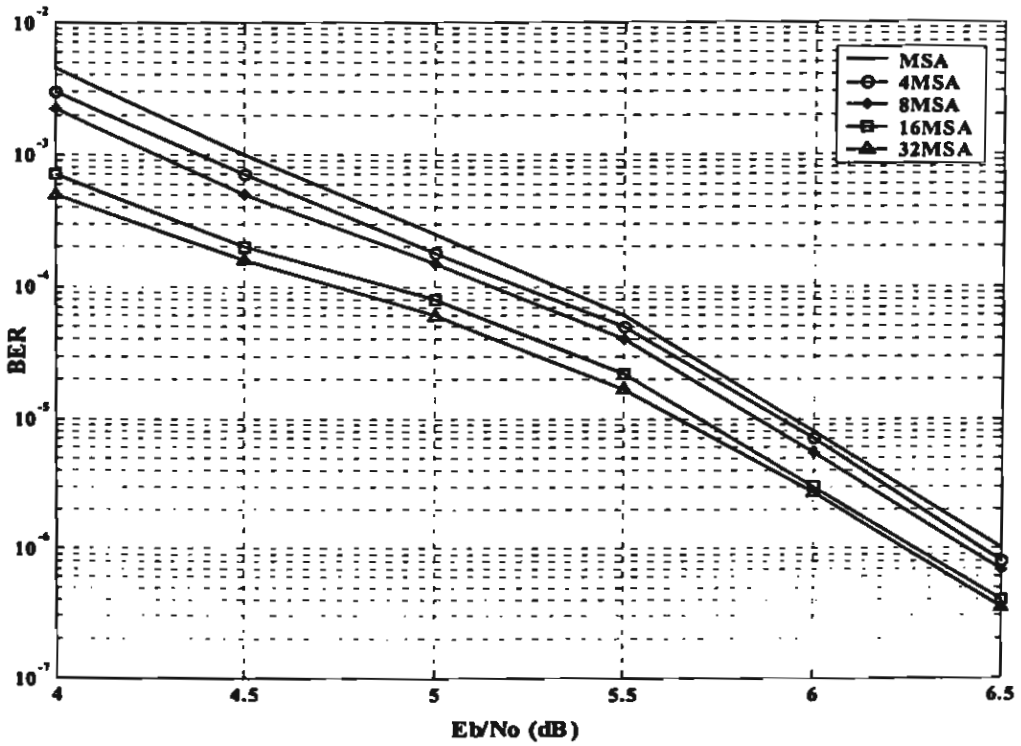


Fig. 16 Comparison on BER with the MSA

A Scalable multiprocessor-based multiple stack algorithm

Virasit Imtawil*, Ph.D. and Wanlop Surakampontorn, Ph.D.

Department of Electrical Engineering, Faculty of Engineering,

Khon Kaen University, Khonkaen, THAILAND 40002

Abstract— A parallel decoding scheme for the multiple stack algorithm (MSA) is proposed in this paper. The proposed scheme is composed of a main decoder working on the main stack and a scalable set of multiple decoders working on their multiple stacks. All the decoders are working almost independently with the help of the control processor. Two cases, 4-processor MSA and 16-processor MSA, are employed to investigate the performance of the proposed scheme. Extensive computer simulations show that the bit error probability of the MSA is improved. We also demonstrate that parallel decoding for the MSA appears to be an interesting scheme for decoding convolutional codes especially in applications where low error probabilities are required.

Keywords— Multiple stack algorithm; Parallel decoding; Convolutional codes

Output

1. V. Imtawil and W. Surakampontorn, submitted to *IEE Proceedings – Communications*

* Corresponding author Tel. 043 202353, 06 715 8082, email: virasit@kku.ac.th