รายงานผลดำเนินการ

โครงงานวิจัยการพัฒนาโปรแกรมเข้ารหัสทั้งภาพและเสียงแบบ MPEG-4 บนโทรศัพท์เคลื่อนที่ยุคที่ 3 ชนิดประหยัดพลังงาน

(Development of Low-Power MPEG-4 Encoder on 3G Cellular/Mobile Phones)

ผศ.ดร.สุรินทร์ กิตติธรกุล

ปีงบประมาณ 2550

รายงานผลดำเนินการโครงงานวิจัยการพัฒนาโปรแกรมเข้ารหัสทั้งภาพและเสียงแบบ MPEG-4 บนโทรศัพท์เคลื่อนที่ยุคที่ 3 ชนิดประหยัดพลังงาน

(Development of Low-Power MPEG-4 Encoder on 3G Cellular/Mobile Phones)

1. ความสำคัญและที่มาของปัญหาที่ทำการวิจัย

จากเมื่อก่อนที่การสนทนากันทางโทรศัพท์ด้วยเสียง จนกระทั่งเทคโนโลยีต่างๆได้พัฒนาขึ้นจนกระทั่งปัจจุบันโทรศัพท์เคลื่อนที่ได้เข้ามามีบทบาทในชีวิตประจำวันมากขึ้น ดังจะเห็นได้จากวิวัฒนาการของโทรศัพท์เคลื่อนที่ที่ได้มีการพัฒนาไปมาก จากอดีตที่มีการทำงานด้วยระบบอนาล็อก (Analog) มาสู่ยุคปัจจุบันที่ทำงานด้วยระบบดิจิตอล (Digital) หรือยุคของ 2จี (2G), 2.5จี (2.5G) และยังมีแนวโน้มจะก้าวไปสู่ยุค 3จี (3G) ในเวลาอันใกล้นี้ จึงทำให้โทรศัพท์เคลื่อนที่ไม่ได้เป็นแค่เพียงอุปกรณ์ที่ใช้สำหรับการสื่อสารด้วยเสียงเพียงอย่างเดียวเท่านั้น แต่ยังสามารถใช้การสื่อสารด้วยภาพเพิ่มเติมเข้าไปด้วย ช่วยให้สามารถสื่อสารและทำความเข้าใจกับคู่สนทนาได้ดียิ่งขึ้น สามารถสนทนากันได้ยินเสียงและภาพพร้อมกันทั้งบนโทรศัพท์มือถือ, พีดีเอ และระบบมัลติมีเดียส่วนตัว ซึ่งเทคโนโลยี MPEG-4 วีดีโอดีโค้ดเดอร์และเอ็นโค้ดเดอร์ ได้รับความนิยมอย่างแพร่หลายในการบีบอัดข้อมูลในการสื่อสารแบบเรียลไทม์ (Real-time) เนื่องด้วยความสามารถในการบีบอัดข้อมูลที่มีความสามารถเพียงพอในการลดขนาดของภาพให้สามารถผ่านช่องทางที่มี bandwidth ที่จำกัดเช่นเครือข่ายไร้สายอย่างโทรศัพท์มือถือ แต่อย่างไรก็ตามวีดีโอ MPEG-4 บน RISC โพรเซสเซอร์ ยังต้องคำนวณมากและใช้พลังงานจากแบตเตอร์รี่มาก RISC โพรเซสเซอร์ในระบบพวกนี้ที่ได้รับความนิยมมากคือ ARM เนื่องด้วยกินพลังงานน้อยและขนาดเล็ก [10] โดยเฉพาะอย่างยิ่งในมือถือที่มีพลังงานในการคำนวณที่จำกัดและมีผลกระทบโดยตรงกับเวลาในการใช้งานแบตเตอร์รี่

ดังนั้นจึงมีความจำเป็นต้องออพติไมซ์ (Optimize) พลังงานและความเร็วของโคเดค (เอ็นโค้ดเดอร์และดีโค้ดเดอร์) MPEG-4 เป็นงานที่จำเป็น ในงานวิทยานิพนธ์ชิ้นนี้จะมุ่งเน้นการออพติไมเซชั่นเพื่อลดการใช้พลังงานและและเพิ่มความเร็วสำหรับวีดีโอ MPEG-4 simple profile ที่ทำงานบน ARM โพรเซสเซอร์

2. วัตถุประสงค์ของโครงการวิจัย

เพื่อพัฒนาต่อยอดโครงการพัฒนาโปรแกรมถอดรหัสวีดีโอแบบ MPEG-4 บนโทรศัพท์เคลื่อนที่ยุคที่ 3 ชนิดประหยัดพลังงานซึ่งได้รับการสนับสนุนในปีงบประมาณ 2548 โดยพัฒนาเข้ารหัส MPEG-4 บนโทรศัพท์เคลื่อนที่สมาร์ต (Smart Mobile Phones) ให้ใช้พลังงานต่ำ (Low Power)

3. ทฤษฎี สมมติฐานหรือกรอบแนวความคิด (Conceptual Framework) ของโครงการวิจัย

3.1 หลักการของมาตรฐาน MPEG-4 วีดีโอเข้ารหัสถอดรหัส (Codec)

การพัฒนาวิธีการบีบอัดข้อมูลวีดีโอ (Video Compression Algorithm) ด้วยมาตรฐาน MPEG นั้น เริ่มประมาณปี 1988 โดยคำว่า MPEG ย่อมาจาก Motion Pictures Expert Group (กลุ่มของผู้เชี่ยวชาญ ด้านภาพเคลื่อนไหว หรือ ภาพยนตร์) ซึ่งเป็น องค์กรภายใต้ ISO Study Group 29 ที่ทำการ พัฒนา การผลิต และสร้างโปรแกรมอิสระเพื่อใช้เป็น มาตรฐาน สำหรับ การบีบอัดข้อมูลวิดีโอ โดยมาตรฐานแรก ที่ออกมา นั่นคือ MPEG-1 ที่มีการพัฒนา และเปิดตัว ในเดือนสิงหาคม ปี 1993 โดยมีอัตราข้อมูล(bit rate)ที่ 1.5Mb/s และในปี1990 MPEGได้เริ่มพัฒนามาตรา MPEG-2 โดย MPEG-1 มีเป้าหมายที่โปรแกรมเหมาะสำหรับงานสร้างสรรค์ เพื่อความบันเทิง ภายในครอบครัวเท่านั้น เพราะ MPEG-1 มีข้อจำกัด เรื่องความละเอียด ของภาพเพียง 352 x 288 pixel อัตราข้อมูล (bit rate)ระหว่าง 1.5 Mb/s เท่านั้น ในขณะที่ MPEG-2 มีคุณภาพที่สูงทั้งระบบภาพ ระบบเสียง เนื่องจากอัตราข้อมูลที่สูงกว่า 2Mb/s จนถึง 30Mb/s

จากการที่คาดการณ์เกี่ยวกับธุรกิจการสื่อสาร, คอมพิวเตอร์ และธุรกิจทีวี/ภาพยนตร์ ISO จึงได้เริ่มพัฒนามาตรฐาน MPEG-4 ขึ้นในปี 1994 และออกมาเป็นมาตรฐานในปี 1998 โดย MPEG-4 ให้ความสำคัญต่อการบีบอัดข้อมูล, ความเป็นสากล, และความเหมาะสมตามสภาพที่ใช้งานจริง MPEG-4จะมีอัตราข้อมูลที่ต่างกัน เช่น ในการใช้งานบนโทรศัพท์เคลื่อนที่ใช้อัตราข้อมูล(bit rate)5-64 Kb/s แต่การใช้งานสำหรับทีวี/ภาพยนตร์ อาจต้องการอัตราข้อมูลถึง 2Mb/s [Sikora, 1997] เป็นต้น ปัญหาสำคัญ ในการจัดการ กับไฟล์ภาพวิดีโอก็คือ ขนาดของไฟล์ภาพ ที่มักจะมีขนาดใหญ่เกินไป โดยในแต่ละวินาทีนั้น อาจจะต้องใช้ ไฟล์ภาพถึง 30 ภาพ ซึ่งแต่ละภาพ กินเนื้อที่ราว 1 เมกะไบต์ (ไม่รวมเสียง) จากที่ MPEG-2 ซึ่งเทคโนโลยีนี้ ใช้วิธีอัดไฟล์ภาพ โดยการเปรียบเทียบ ข้อมูลภาพ ในแต่ละเฟรม หากส่วนใด ของเฟรมนั้น ที่คงเป็น ภาพเดิม ส่วนนั้น ก็จะไม่ถูก บันทึกไว้ ซึ่งนั่นทำให้ ขนาดของไฟล์ภาพ เล็กลงกว่า แต่อย่างไรก็ตาม ปัญหาของ MPEG-2 ก็คือ ยังเป็นการจัดการกับข้อมูลภาพ ทั้งเฟรมอยู่ ทำให้ไฟล์ภาพยังมีขนาดค่อนข้างใหญ่ ถึงแม้จะมีการตัดข้อมูลภาพ ส่วนที่ซ้ำกัน ออกไปมากแล้วก็ตาม เพื่อประหยัดเนื้อที่ MPEG-4 จึงมีการจัดการโดยมองเป็นวัตถุ (Object) ในภาพแทน การจัดการกับวัตถุในเฟรม (กรอบ) ของ MPEG-4 ช่วยประหยัดเนื้อที่การจัดเก็บได้มากกว่าไฟล์ MPEG-2 ถึง 8 - 12 เท่า [Sikora, 1997] ซึ่งนับว่าช่วยประหยัดเนื้อที่ได้อย่างมาก

การส่งสัญญาณภาพแบบไร้สายลงไปที่ เครื่องคอมพิวเตอร์ อุปกรณ์พกพา ส่วนบุคคล หรือโทรศัพท์มือถือ ยังสามารถทำได้ เพราะ MPEG-4 จะช่วยให้ มีการปรับขนาดของไฟล์ภาพ ให้

พอเหมาะ กับขนาดของอุปกรณ์พกพาซึ่งมีขนาดค่อนข้างเล็ก และ ด้วยอัตราการถ่ายโอนข้อมูล (bit rate) ที่ค่อนข้างต่ำ แต่ได้คุณภาพระดับสูงของภาพวิดีโอ ทำให้จะสามารถ ดูภาพยนตร์ บนจอเล็กๆ ของโทรศัพท์มือถือ ได้อย่างสะดวกสบาย โดยอาศัย เทคโนโลยี MPEG-4 นี้

- การเข้ารหัส (encoding) MPEG-4 ประกอบด้วยงานหลัก คือ

การประเมินการเคลื่อนไหน(motion estimation, ME) และการชดเชยการเคลื่อนไหน(motion compensation, MC) โดยใช้เพียงการเข้ารหัสเฟรมชนิด I (inter-coded frames)

การคำนวณ Discrete Cosine Transform coefficients และ inverse DCT (DCT/IDCT)

ทำการ quantizationและ inverse quantization (Q/IQ)

ทำรหัสที่มีความยาวเปลี่ยนแปลงได้ (Variable-length coefficient coding, VLC)

การเข้ารหัสรูปร่าง (Shape encoding) ต้องการใช้สำหรับ object–based coding

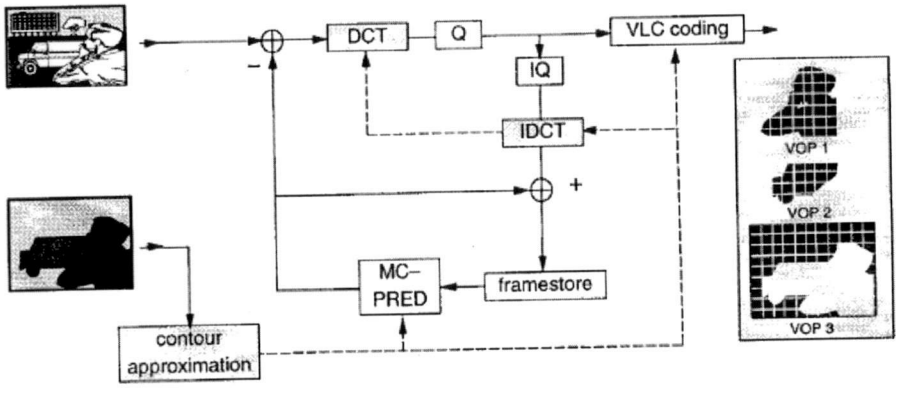
- การถอดรหัส (decoding) MPEG-4 ประกอบด้วยงานหลัก คือ

ทำรหัสที่มีความยาวเปลี่ยนแปลงได้ (Variable-length coefficient coding, VLC)

inverse quantization (IQ)

การชดเชยการเคลื่อนไหน(motion compensation, MC) ใช้สำหรับการเข้ารหัสเฟรมชนิด I (inter-coded frames)

การถอดรหัสรูปร่าง (Shape decoding) ต้องการใช้สำหรับ object–based coding [Cavali et.al, 2002]



รูปที่ 1 แสดงขั้นตอนการ Encode และ Decode ของ MPEG4

3.2 สถาปัตยกรรม ARM920T โปรเซสเซอร์

ARM โปรเซสเซอร์เป็นโปรเซสเซอร์ที่ได้รับความนิยมอย่างมากในระบบสมองกลฝังตัว โดยเฉพาะอย่างยิ่งในมือถือและพีดีเอ ซึ่ง ARM โปรเซสเซอร์จะแบ่งออกเป็นหลายซีรี่เพื่อรองรับความต้องการที่หลากหลาย งานวิจัยเรามุ่งเน้นที่จะใช้ได้จริงบนมือถือและโค้ดของเราทำงานบนระบบปฏิบัติการซิมเบียน ซึ่งมือถือที่ใช้ระบบปฏิบัติการนี่จะใช้ ARM ซีรี่9 ซึ่งใน ARM ซีรี่ 9 มีสองตัวที่ใช้ในมือถือและพีดีคือ ARM920T และ ARM922T ทั้งสองตัวนี้มีสถาปัตยกรรมการทำงานที่เหมือนกันต่างกันตรงที่ขนาดของแคช (cache) ที่ ARM920T มีขนาด 16 กิโลไบต์ ส่วน ARM922T ขนาด 8 กิโลไบต์ ในงานวิจัยเราได้เลือกศึกษา ARM920T เพราะมีแคชขนาดที่ใหญ่กว่า ARM922T เพื่อที่จะได้เห็นผลในการทดลองที่ชัดเจน

3.2.1 เกี่ยวกับ ARM920T

ARM920T เป็นสมาชิกหนึ่งในตระกูล ARM9TDMT ที่เป็น microprocessor ที่ใช้ในงานทั่วไป ซึ่งประกอบด้วย:

1. ARM9TDMI (core)
2. ARM940T (core บวก cache และ protection unit)
3. ARM920T และ ARM922T (core บวก cache และ MMU)

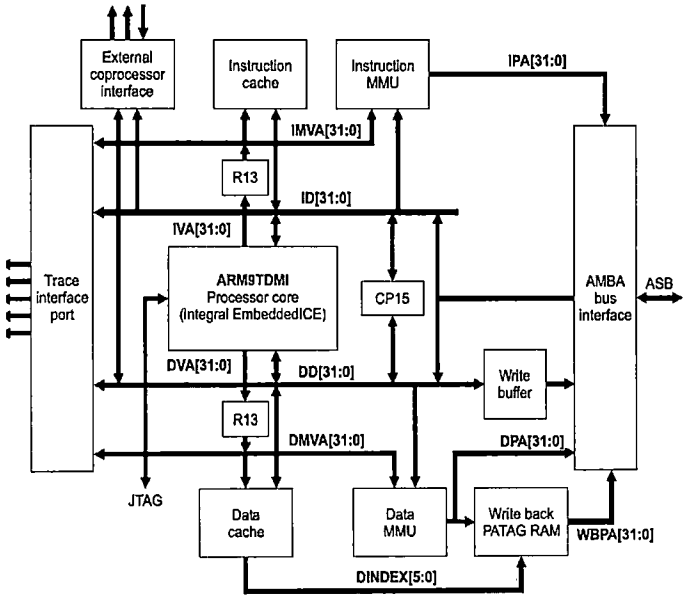
ARM9TDMI โพรเซสเซอร์ใช้สถาปัตยกรรม Harvard ที่มีห้า stage ของ pipeline ประกอบด้วย Fetch, Decode, Execute, Memory และ Write. ARM9TDMI สามารถใช้เป็น standalone core ได้นั้นคือสามารถที่จะฝังไปในอุปกรณ์ที่ซับซ้อนได้ โดยการใช้เป็น standalone core จะมี simple bus interface ที่อนุญาตให้ออกแบบระบบ cache และ memory ได้

ตระกูล ARM9TDMI microprocessor รองรับทั้งชุดคำสั่ง 32 bits ARM และ 16 bit Thumb ทำให้เราสามารถเลือกได้ว่าจะเอาโค้ดที่มีประสิทธิภาพสูงหรือจะเอาโค้ดที่มีขนาดเล็ก

ARM920T โพรเซสเซอร์ใช้สถาปัตยกรรม Harvard cache จุดประสงค์ใช้กับ multiprogramer application ที่มี memory management, ประสิทธิภาพสูง และใช้พลังงานน้อย มีการแยก instruction cache และ data cache ขนาด 16 KB ที่มี line ละ 8 word ARM920T มีการสร้าง enhanced ARM architecture v4 MMU ที่มีการจัดการการตรวจสอบสิทธิในการ access และ translation สำหรับตำแหน่ง instruction และ data

3.2.2 Block diagram ของฟังก์ชันการทำงานของโพรเซสเซอร์

รูป 2 แสดง Block diagram ของฟังก์ชันการทำงานของ ARM920T โพรเซสเซอร์



รูปที่ 2. ARM920T ไดอะแกรม

สถาปัตยกรรม ARM920T เป็น RISC โปรเซสเซอร์ขนาด 32 บิตที่ใน ARM920T เป็นโพรเซสเซอร์ภายในประกอบด้วย ARM9TDMI โพรเซสเซอร์กับ:

-แคชของชุดคำสั่งและแคชของข้อมูลขนาด 16 กิโลไบต์

-MMUหรือหน่วยจัดการหน่วยความจำคำสั่งและข้อมูล (Instruction and data Memory Management Units)

-ปัฟเฟอร์ก่อนเขียนลงหน่อยความจำ (Write buffer)

-AMBA™ (Advanced Microprocessor Bus Architecture) บัสอินเตอร์เฟส

-บัสในการส่งข้อมูลและคำสั่งขนาด 32 บิต

-Embedded Trace Macrocell (ETM) อินเตอร์เฟส

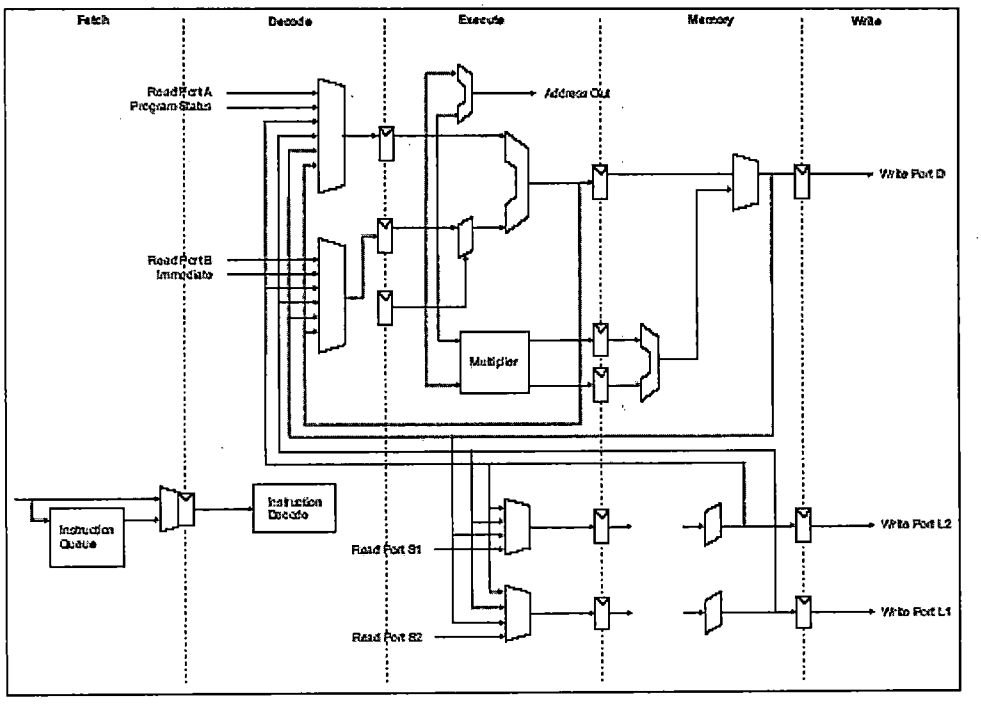
3.2.3 ARM920T Macrocell

ARM920T Macrocell อยู่บนพื้นฐานสถาปัตยกรรม ARM9TDMI Harvard โดยมีไปป์ไลนขนาดห้าขั้น: เฟ็ตช์ (fetch), ดีโค้ด (decode), ประมวลผล (execute), เข้าใช้หน่อยความจำ (memory access) และเขียนรีจีสเตอร์ (write register back) แสดงดังรูปที่ 3.

เพื่อลดแบนด์วิดท์และความหนาแน่นข้อมูลที่หน่วยความจำหลัก ARM920T จึงมี

– แคชชุดคำสั่ง

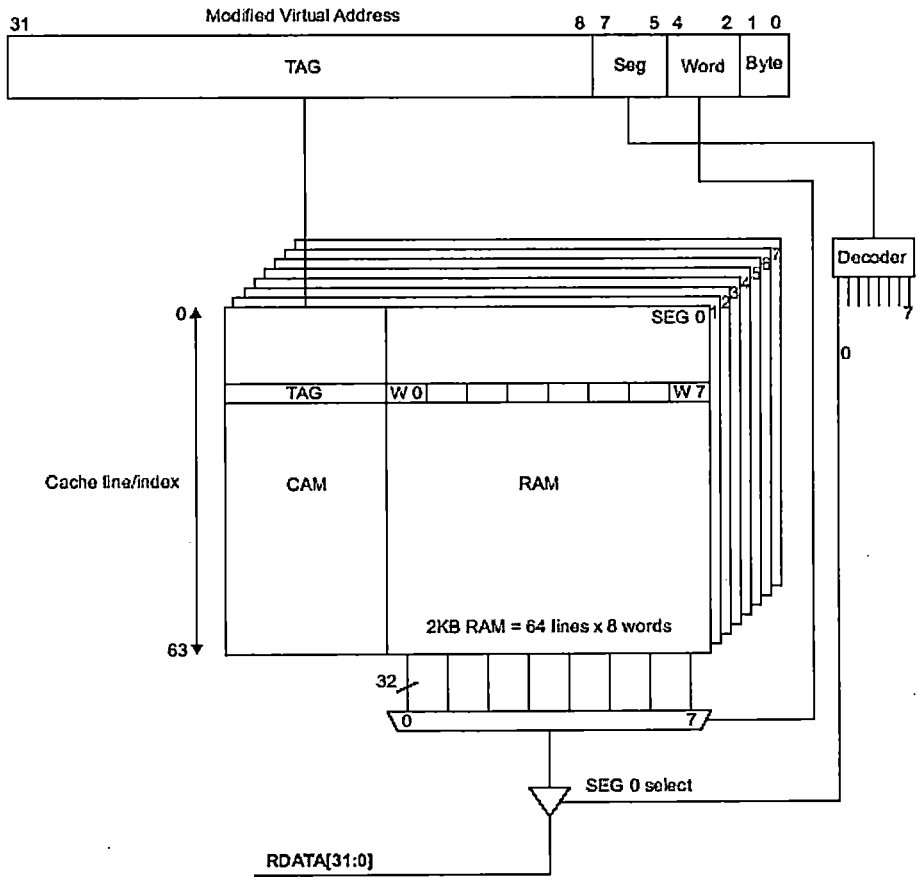
- แคชข้อมูล
- MMU
- TLBs
- Write buffer



รูปที่ 3. ไปป์ไลนของ ARM9TDMI

3.2.4 แคชของ ARM920T

ทั้งแคชของคำสั่งและของข้อมูลทั้งสองมีขนาด 16 กิโลไบต์ มีขนาดรายละ 8 word และเป็น 64-way set associative [5] จึงมี 8 Segments และมีบัสข้อมูลขนาด 32 บิตต่อทุกแคชไปที่ ARM9TDMI และเป็นจะ allocate ข้อมูลบนแคชเมื่ออ่านไม่เจอในแคช และยังใช้นโยบาย Write thought และ Non-allocate เมื่อการเขียนข้อมูลแล้วไม่เจอในแคช (Write miss) ส่วนการเขียนข้อมูลที่มีในแคช (Write hit) จะใช้ Write back



รูปที่ 4 โครงสร้างการจัดการของแคช

3.2.5 แอพพลิเคชั่นที่ใช้บน ARM920T

ระบบปฏิบัติการ

-EPOC (พวก Symbian OS)

-Linux

-WindowsCE

แอพพลิเคชั่นไร้สายประสิทธิ์ภาพสูง

-สมาร์โฟน

-พีดีเอ

เนตเวิร์กกิ้งแอพพลิเคชั่น

การเข้ารหัสและคล้ายรหัสของเสียงและภาพ

3.3 วิธีการออพติไมเซชั่นเพื่อลดการบริโภคพลังงานของ Encoder บน ARM920T

โครงงานนี้คือการทำให้ ระบบทั้งหมด ทำงานอย่างมีประสิทธิภาพดีที่สุด (Optimal Performance) ภายใต้เงื่อนไข (Constraints) คือเริ่มจากการรับภาพจากกล้อง มาใส่ buffer แล้วทำการแปลงจาก RGB เป็น YUV แล้วส่งต่อให้ตัว Encoder ทำการบีบอัดข้อมูลแล้วส่งข้อมูลไป comunication จากนั้น comunication จะติดต่อกับ GPRS/EDGE หรือ Bluetooth โดยที่จะลดการบริโภคพลังงานจากการพยายามทำการลดการเข้าใช้หน่อยความจำด้วยการปรับปรุงการทำงานของ cache โดยใช้ภาษาระดับสูง

จุดประสงค์คือการลดการใช้พลังงาน (low power) โดยการลดการใช้งานหน่วยความจำ (Memory Usage) ซึ่งนับจาก จำนวนครั้งในการอ่านเขียนข้อมูล และ ปริมาณข้อมูลที่ใช้ ในระบบจากการศึกษาการทำงานเกี่ยวกับระบบหน่วยความจำ (Memory System) และระบบบัสจะใช้พลังงานมาก (ระหว่าง 50-80%) [Palkovic et al., 2002] ดังนั้นวิธีออพติไมซ์เพื่อลดพลังงานที่เราใช้จะมุ่งไปการลดจำนวนการอ้างหน่วยความจำโดยทำออพติไมเซชั่นในภาษาระดับสูง

3.3.1 โมเดลกำลังและพลังงาน(Power/Energy Model)

ในระบบมัลติมีเดียสมองกลฝังตัว การเข้าใช้หน่วยความจำจะกินพลังงานถึง 50 – 80 % ของทั้งหมด การประเมินพลังงานที่ใช้ไปง่ายๆแสดงในสมการ (1) และ (2):

$$P_{Tr} = E_{Tr} * \frac{\#Transfers}{Second} \qquad (1)$$

$$E_{Tr} = f(\#words, \#bits) \qquad (2)$$

โดย P_{Tr} เป็นพลังงานที่ให้ต่อวินาที และ E_{Tr} เป็นพลังงานที่ใช้ต่อหนึ่งการส่งข้อมูลซึ่งขึ้นกับขนาดบิต สมมุติฐานของเราคือ E_{Tr} เป็นค่าคงในทุกการเข้าใช้หน่วยความจำ และผลของพลังงานที่ใช้จะแปรผันตรงจำนวนการส่งข้อมูล

$$E_{Total} \propto \#Transfers \qquad (3)$$

3.3.2 อัลกอริธึมในออพติไมเซชั่นเพื่อลดการใช้พลังงาน

กรณีการเขียนข้อมูลแล้วไม่เจอในแคช (Write miss) ในโพรเซสเซอร์ ARM920T ใช้นโยบาย Write thought และ Non-allocate ส่วนการเขียนข้อมูลที่มีในแคช (Write hit) ในโพรเซสเซอร์นี้จะใช้ Write back และเขียนใน Write back บัฟเฟอร์ อัลกอริธึมในออพติไมเซชั่นของเรา minimizing buffer และ read before write สามารถลดจำนวนการเข้าใช้หน่วยความจำโดยปรับปรุงเปลี่ยนแปลงโปรแกรมเพื่อประโยชน์ต่อแคชมากขึ้น โดยผลที่ออกมาจะมีจำนวน Cache miss น้อยลงซึ่งก็คือลดเหตุการณ์การใช้หน่วยความจำใน ARM ไมโครโพรเซสเซอร์

3.3.2.1 วิธี Minimizing Buffer Allocationในการพัฒนา C++ บนระบบปฏิบัติการซิมเบียน ใน subroutine จะมีบ่อยๆที่ใช้บัฟเฟอร์ในการเก็บผลก่อนในระหว่างการคำนวณ จากที่สังเกตดูบัฟเฟอร์พวกนี้จะ allocate แบบ local แล้ว assign(written) ต่อมาจะเอาค่าไปใช้ (read) ท้ายสุดทำลายทิ้ง (deallocate) เมื่อ subroutine ถูกเรียกทำงาน ตำแหน่งหน่วยความจำจะถูก allocate ในตำแหน่งที่แตกต่างกันในแต่ล่ะครั้ง ซึ่งคือปัญหาที่ยากในการจัดการกับแคชข้อมูลที่มีขนาดจำกัด วิธีของเราในการแก้ปัญหานี้โดยการประกาศบัฟเฟอร์ข้อมูลเป็น Attribute ของ class ทำให้สามารถเข้าสโคปที่ใหญ่ขึ้น โดยผลที่ได้จากที่บัฟเฟอร์ข้อมูล allocate ครั้งเดียวแล้วใช้มันอีกเรื่อยๆเป็นผล ให้ cache miss ลดลงดังในรูปที่ 5

```
void Cdecoder::ConstructL()
{
    int32_t roundtab1[16]=
    { 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2 };
        •
        •
}
void Cdecoder::decoder_mbinter(DECODER * dec, ...)
{
    int16_t *block,*data;
    block = new int16_t[6*64];
    data = new int16_t[6*64];
    uint32_t stride = dec->edged_width;
    uint32_t stride2 = stride >>1;
        •
        •
}
void Cdecoder::decoder_mbintra(DECODER * dec, ...)
{
    int16_t *block,*data;
    block = new int16_t[6*64];
     data  = new int16_t[6*64];
    uint32_t stride = dec->edged_width;
    uint32_t stride2 = stride / 2;
        •
        •
}
```

(a)

```
void Cdecoder::ConstructL()
{
    block = new int16_t[6*64];
    data = new int16_t[6*64];
    int32_t roundtab1[16]=
    { 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2 };
        •
        •
}

void Cdecoder::decoder_mbinter(DECODER * dec, ...)
{
    uint32_t stride = dec->edged_width;
    uint32_t stride2 = stride >>1;
        •
        •
}

void Cdecoder::decoder_mbintra(DECODER * dec, .. )
{
    uint32_t stride = dec->edged_width;
    uint32_t stride2 = stride / 2;
        •
        •
}
```

(b)

รูปที่ 5 โค้ด C++ ของ XviD MPEG-4 วีดีโอดีโค้ดเดอร์บนซิมเบียน (a) ก่อนและ (b) หลังใช้วิธี Minimizing Buffer Allocation

ในวิธีนี้จะทำให้บัฟเฟอร์เข้าไปอยู่ในแคช แม้ว่าในสถาปัตยกรรมของ ARM จะมีความสามารถทำ lock down ข้อมูลในแคช แต่มันยากสำหรับการเขียนโปรแกรมบน C/C++ให้มีประสิทธิภาพ

3.3.2.2 วิธี Read before Write เราพบว่าบัฟเฟอร์ที่ใช้เป็น temp จะถูกเขียน (ข้อมูลในระหว่างการคำนวณ) ครั้งแรกและจะถูกอ่านเอาข้อมูลไปใช้ต่อ ในการเขียนข้อมูลลงบัฟเฟอร์เป็นเหตุของการ write miss จึงเขียนลงหน่วยความจำเลย (write thought) ต่อมาจะเรียกใช้ข้อมูลในบัฟเฟอร์อีกก็จะไม่พบในแคช (read miss) โค้ดลักษณะแบบนี้เป็นโค้ดที่ไม่มีประสิทธิ์ภาพในการบริโภคพลังงานคือตอนแรกเขียนก็ไม่เจอในแคชต่อมาจะใช้ก็ไม่พบบนแคชทำให้เกิดการ data cache miss สองครั้งและเกิดการเข้าใช้หน่วยความจำที่ไม่จำเป็น

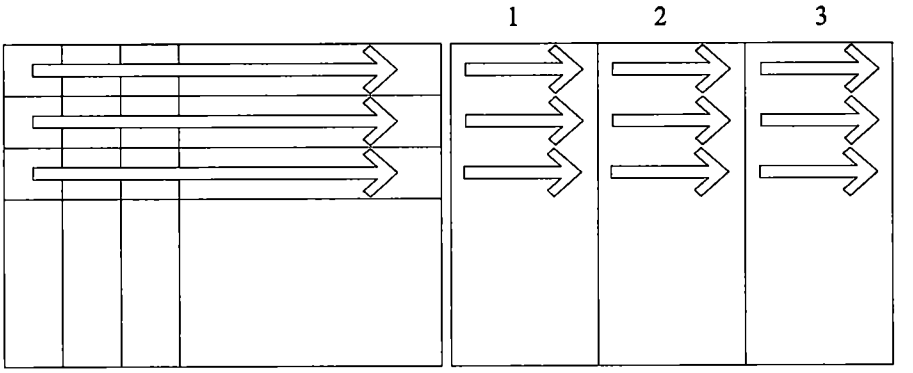
เราจึงเสนอทางแก้ง่ายคือทำการเริ่มตอนก็อ่านบัฟเฟอร์เพื่อให้บัฟเฟอร์เข้าไปอยู่ในแคชข้อมูลก่อนจะใช้งาน ดังนั้นจำนวนของการเข้าใช้หน่วยความจำจะลดลงเหมือนที่กินพลังงานน้อยลงแสดงในรูปที่ 6

```
//Load block to Dcache
int count;
for(count=0;count<6*64;count++)
{       tempdata+ = block[count];
        tempdata+ = data[count];   }
if (dec->quant_type == 0)
{       for (i = 0; i < 6; i++)
        {       if (cbp & (1 << (5 - i)))       // coded
                {Mem::Fill(&block[i * 64], 64 * sizeof(int16_t), 0);
                 ..
                }
        }
}
else
{       for (i = 0; i < 6; i++)
        {       if (cbp & (1 << (5 - i)))       // coded
                {Mem::Fill(&block[i * 64], 64 * sizeof(int16_t), 0);
                ..
                }
        }
}
```

รูปที่ 6 ส่วนโค้ดจาก MPEG-4 วีดีโอดีโค้ดเดอร์ที่ออฟติไมต์โดยวิธี read before write

3.3.2.3 วิธี Cache-Conscious Motion Estimation Pattern: Algorithms ในการ Encoding จะใช้พลังงานมากในการทำ motion estimation (ME) แม้ในหลายวิธีการ fast ME algorithm จะทำให้ลดการคำนวณและการเข้าใช้หน่วยความจำลงไป (ลดการใช้พลังงาน) แล้ว อย่างไรก็ตามขนาด cache และ cache replacement policy จะมีผลกระทบต่อการลดการคำนวณและการเข้าใช้หน่วยความจำลงไป วีดีโอ Encoder โดยส่วนมากจะเข้าใช้ ME ตามแนวแถวจากบนลงล่าง ดังรูป 7(a) ในการทำ motion estimation MB ที่อยู่ขวาไกลอาจถึงแทนที่ pixel ก่อนหน้าเนื่องด้วยความกว้างของ frame ที่กว้างและ cache replacement policy ของ ARM920T ที่เป็น FIFO

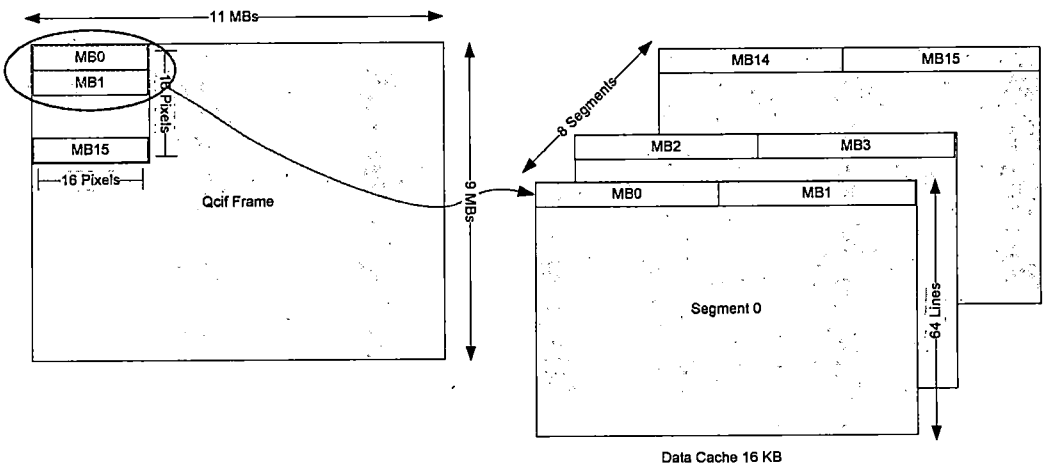
และ pattern ใหม่ที่แสดงในรูป 7(b) คือ ME จะเริ่มจาก MB ซ้ายไปขวาและบนลงล่างแต่ทำตาม column ก่อน ซึ่งทำให้ pixel ของ reference frame ใน data cache ถูกใช้มากขึ้นก่อนถูกไล่ออกไป โดยเราไม่ต้องออพติไมซ์ cache replacement policy ดังนั้นเราจึงเรียกวิธีนี้ว่า **Cache-Conscious Motion Estimation Pattern**



รูปที่ 7 ลำดับการเข้าใช้ MBs ในการทำ motion estimation (a) รูปแบบเก่า (b) รูปแบบใหม่

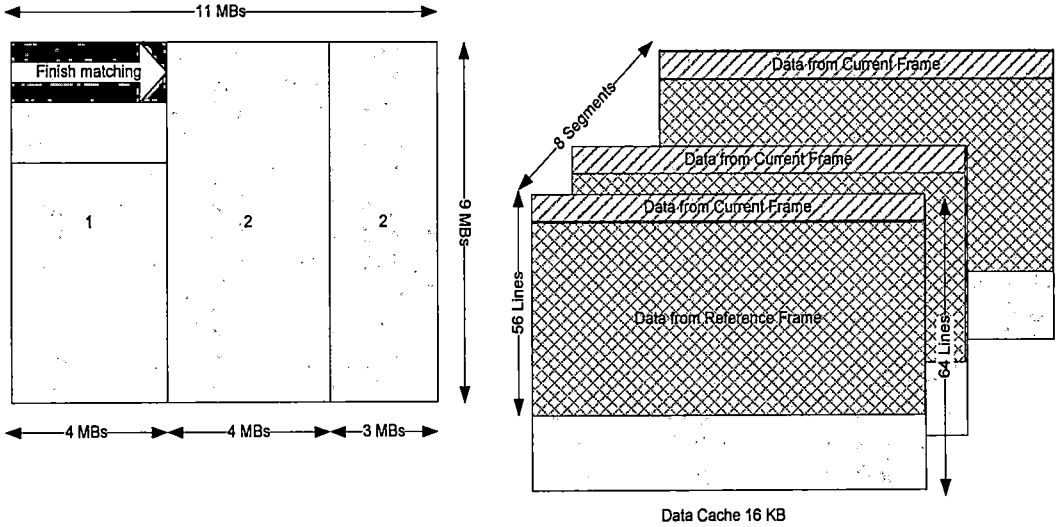
หลักการแบ่ง Pattern ของ Cache-Conscious Motion Estimation Pattern

วีดีโอที่เราใช้ในการทดลองคือ QCIF มีขนาด 176 * 144 Pixels คิดเป็น 11 * 9 MB โดย MB มีขนาด 16 * 16 Pixels เราใช้ pattern เป็น 4-4-3 ทำ Motion Estimation เริ่มคอลัมน์แรกจาก 4 MB คอลัมน์สอง 4 MB และคอลัมน์ท้าย 3 MB เหตุผลที่ใช้ pattern นี่ เราเฉลี่ยการทำ motion matching MB คือการไปหา MB ใดใน reference frame ใกล้เคียงกับ MB ใน current frame ทำไม่เกิน 4 ครั้ง หรือ 4 MB ซึ่งมีการเคลื่อนไหวอยู่ในขอบเขตวงกลมรัศมี 4 MB หรือรัศมี 64 Pixels สำหรับบนโทรศัพท์มือถือเวลาใช้งานจะเป็นภาพวีดีโอจากการถ่ายจากกล้องมือถือ ซึ่งถือว่าขอบเขตขนาดนี้ครอบคลุมการเคลื่อนไหวของภาพจากโทรศัพท์มือถือ โครงสร้างสถาปัตยกรรมของ Data Cache ใน ARM920T ที่มี 8 segments ในหนึ่ง segment มี 64 line โดย line ละ 8 word หรือ 32Bytes ให้หนึ่ง MB เวลา load ลง Data Cache จะตั้งแต่ segment 0 ถึง segment 7 ใช้ segment ละ 1 line ดังในรูปที่ 8



รูปที่ 8 แสดงข้อมูล MB ที่ถูกเก็บอยู่ใน Data Cache ของ ARM920T

ดังนั้นใน Motion matching ที่ใช้ใน XviD MPEG-4 video encoder คือ PMVfast ในตอนแรกจะทำการทำนายการเคลื่อนไหวจาก MB รอบตัวแล้วจากนั้นทำการ search แบบ diamond ทำการ search 4 ครั้งต้องอ้าง MB ใน reference frame 13 MB และ current frame 1 MB รวมต่อ 1 Motion matching อ้าง 14 MB การที่ทำ motion estimation แบบรูป 7(a) ที่ทำ motion matching ไป 11 MB ทางแนวนอนต้องอ้าง 11 * 14 MB เท่ากับ 154 MB แล้วกลับมาทำในแถวต่อมาข้อมูล reference frame ที่จำเป็นในการทำ motion matching เคยอยู่ใน Data Cache จากการอ้างในแถวก่อนหน้านี้ จะโดยข้อมูลอื่น load เข้าไปแทนที่อย่างแน่นอน เราใช้ pattern 4-4-3 ทำ motion matching ไป 4 MB มีการอ้าง 4 * 14 เท่ากับ 56 MB คือใช้ไป 56 line ในแต่ล่ะ segment เมื่อกลับมาทำ motion matching แถวต่อไปข้อมูล MB ของ reference frame ก็จะยังอยู่ใน Data Cache ดังรูปที่ 9

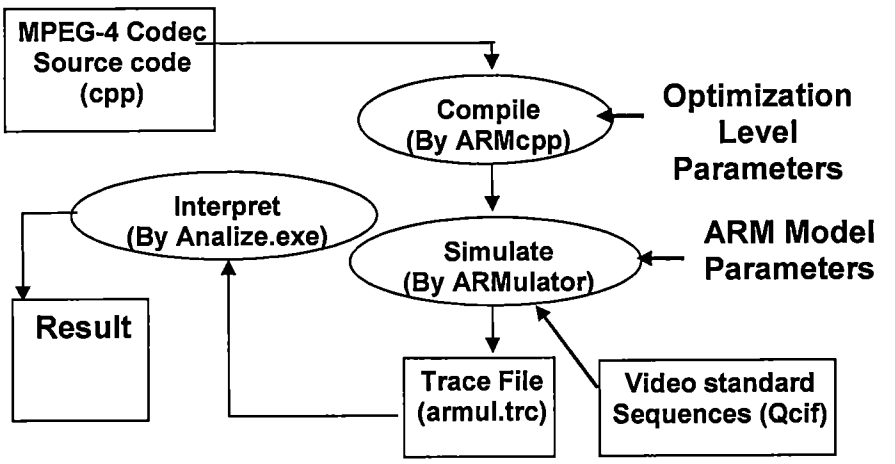


รูปที่ 9 แสดงข้อมูลที่อยู่ใน Data Cache เมื่อทำ Motion matching MB แล้ว 4 MB

4. การทดลองและผลการทดลอง

4.1 วิธีการทดลอง

เพื่อแสดงถึงประสิทธิภาพที่เราเสนอ รูปที่ 10 แสดงวิธีการทดลอง เราจะใช้ simple-profile simple-level MPEG-4 วีดีโอ Codec ของ XviD ที่ทำการแปลงโค้ดไปทำงานบนระบบปฏิบัติการซิมเบียน ใช้ ARM คอมไพเลอร์ (ARMcpp) ซึ่งมีความสามารถออพติไมซ์ได้สูงคือมีโค้ดขนาดเล็กและมีสามลำดับชั้นในการออพติไมซ์ความเร็ว Opt0, Opt1 และ Opt2



รูปที่ 10 แสดงขั้นตอนการทดลองของเรา

หลังจากคอมไพล์เสร็จได้ Binary ไฟล์แล้วนำไปจำลองการทำงานโดย ARMulator ซึ่งสามารถจำลองการทำงานหลายเวอร์ชันของสถาปัตยกรรม ARM instruction set สามารถนับจำนวนสัญญาณนาฬิกา, การเข้าใช้หน่วยความจำ และ เหตุการณ์พิเศษ (เช่น แคชข้อมูลหรือชุดคำสั่ง, write back stall) เราใช้ ARMulator จำลองการทำงานของ ARM920T โดยไม่มีหน่วย floating-point

ในการจำลองการทำงาน MPEG-4 video codec เราใช้สอง QCIF วีดีโอในการทดลองมี Foreman และ News โดยบีบอัดที่ 15 เฟรมต่อวินาทีและ 24 กิโลบิตต่อวินาที เพียง 10 วีดีโอเฟรม (หนึ่ง I และเก้า P เฟรม) ที่ดีโค้ดเดอร์

หลังจากการจำลองจะได้ trace ไฟล์คือ armul.trc โดยภายในประกอบด้วยเหตุการณ์ของโพรเซสเซอร์ต่างเช่นการเข้าใช้หน่วยความจำ, การไม่เจอข้อมูลในแคชและข้อมูลที่เกี่ยวข้อง เราจะแปลข้อมูล trace ไฟล์โดยใช้โปรแกรมวิเคราะห์ที่เราพัฒนาขึ้นคือ Analize.exe เพื่อเอาข้อมูลที่เกี่ยวกับการเข้าใช้หน่วยความจำและเหตุการณ์ที่เกี่ยวกับแคช ผลลัพธ์ที่ได้จากการวิเคราะห์โดย Analize.exe จะได้จำนวนการอ้างหน่วยความจำทั้งแบบการอ่านและการเขียน จำนวนการเกิด Data

Cache Misses และ จำนวนการเกิด Instruction Cache Misses และนำเอาข้อมูลเหล่านี้ไปแสดงเป็นกราฟ และตาราง เราจะวัดผลการทดลองของ MPEG-4 video codec ก่อนและหลังการทดลองแสดงในส่วนของผลการทดลอง ในรูปที่ 11 แสดงบางส่วนของ armul.trc

```
Date: Sat May 07 18:47:11 2005
Source: Armul
Options: Trace Events

E 00008000 00000000 10005
BSR4O___ A0000000 00000C1E
E 00008000 00000000 10002
BSR8O___ 00008018 E240B001 E242C001
IT 00008000 e28f8090 ADD r8,pc,#0x90 ; #0x8098
IT 00008004 e898000f LDMIA r8,{r0-r3}
BNR4O___ A0000000 00000C1E
BNR8O___ 00008098 00007804 00007828
BSR8O___ 00008080 10844009 E3C44003
BSR8O___ 00008088 E2555004 24847004
```

รูปที่ 11 แสดงตัวอย่างข้อมูลใน "armul.trc"

ต่อไปเราจะแสดงวิธีการแปลข้อมูลการเข้าใช้หน่วยความจำและเหตุการณ์ต่างๆใน trace ไฟล์

1) Memory Bus Trace (Bline): บรรทัดที่เริ่มต้นด้วย B แสดงถึงการเข้าใช้บัสของหน่วยความจำ โดยจะมีรูปแบบดังนี้ [12]

B<type><rw><size>[O][L][S] <address> <data>

โดย

<type> แสดงถึงชนิดของสัญญาณนาฬิกา

S sequential

N nonsequential

<rw> แสดงถึงการอ่านหรือการเขียน

R การอ่าน (Read)

W การเขียน (Write)

<size> แสดงถึงขนาดในการเข้าใช้บัสหน่วยความจำ

4 word (32 บิต)

2 halfword (16 บิต)

1 ไบต์ (8บิต)

O คือ Opcode fetch (instruction fetch)

L คือ locked access (SWR instruction)

S คือ speculative instruction fetch.

<address> จะอยู่ในรูปแบบเลขฐาน 16 ตัวอย่างเช่น 00008008

<data> จะแสดงตามนี้:

ค่าจากการอ่านหรือเขียน เช่น EB00000C

2) Events Trace (E lines): รูปแบบของบรรทัดเหตุการณ์ (E) เป็นดังนี้ [12]:

E <word1> <word2> <event_number>

ตัวอย่างเช่น E 00000048 00000000 10001

โดย

<word1> แสดงเป็น word บอกเช่นค่า PC

<word2> แสดงเป็น word บอกเช่นตำแหน่งที่เกิด event

<event_number> คือตัวเลขของเหตุการณ์ ตัวอย่างเช่น 0x10001 คือ MMUEvent_DLineFetch โดยเหตุการณ์ทั้งหมดจะอธิบายในตารางที่ 1

ตารางที่ 1 เหตุการณ์พิเศษต่างๆที่ตรวจได้จาก ARMulator

Event name (MMUEvent_)	**Word 1**	**Word 2**	**Event number**
DLineFetch	Miss address	Victim address	0x10001
ILineFetch	Miss address	Victim address	0x10002

รูปที่ 2 แสดงฟังก์ชันการทำงานของ ARM920T [5] โดย ARMulator ตรวจจับข้อมูลจาก trace interface port มันถูกกำหนดให้ตรวจจับการเข้าใช้บัสของหน่วยความจำในการอ่านและเขียนจาก AMBA 32-bit bus interface port

วิธีการทดลองการเพิ่มประสิทธิภาพด้านการลดการใช้พลังงาน

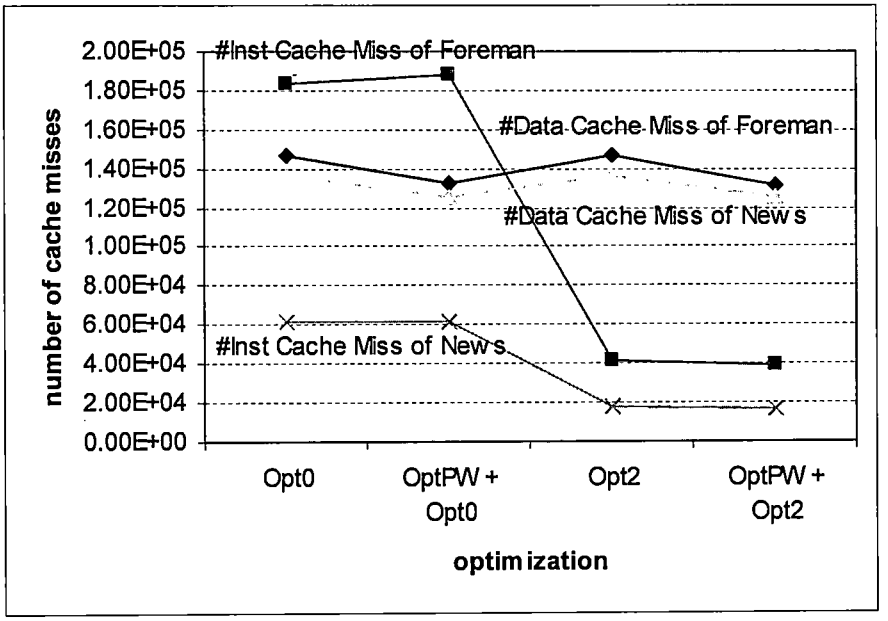
ออพติไมซ์สี่แบบที่แตกต่างกันคือ Opt0, Opt2, Opt0+OptPW และ Opt2+OptPW โดย OptPW คือวิธีการออพติไมซ์พลังงานสองวิธีที่เราเสนอไปในบทที่ 3.2 เราใช้สองวีดีโอ Foreman และ News จำนวน 10 frame (1 I-frame และ 9 P-frames) โดยใช้ทำการ Encode ที่ 15 fps ที่ 24 bps

เราจะหาประสิทธิภาพของวิธีการออพติไมเซชั่นพลังงาน(OptPW) ของเราจากตัวแปลพวกนี้: การไม่พบข้อมูลในแคช (cache miss), การเข้าใช้บัสของหน่วยความจำ (memory/bus access), เวลาในการทำงาน (execution time) และจำนวนพลังงานที่ลดลง

4.2 ผลการทดลองในการเพิ่มประสิทธิภาพด้านการลดพลังงานเอ็นโค้ดเดอร์ (Encoder)

เราจะหาประสิทธิภาพของวิธีการออพติไมเซชั่นพลังงาน(OptPW) ของเราจากตัวแปลพวกนี้: การไม่พบข้อมูลในแคช (cache miss), การเข้าใช้บัสของหน่วยความจำ (memory/bus access), เวลาในการทำงาน (execution time) และจำนวนพลังงานที่ลดลง โดยผลการทดลองที่ได้ได้ผลดังนี้:

1) การไม่พบข้อมูลในแคช (cache miss): ในเอ็นโค้ดเดอร์ส่วนมากจะเป็นการบีบอัดภาพ และการคำนวณการชัดเชยการเคลื่อนไหนซึ่งเป็นการคำนวณส่วนมาก ดังนั้นการเข้าใช้หน่วยความจำส่วนชุดคำสั่ง (instruction) จะบริโภคพลังงานมากกว่าส่วนชุดข้อมูล (data)



รูปที่ 12 จำนวนของการไม่พบข้อมูลในแคชสำหรับทุกการออพติไมซ์ของเอ็นโค้ดเดอร์ (24 กิโลบิตต่อวินาที, 15 เฟรมต่อวินาที, 10 วีดีโอเฟรมต่อวินาที)

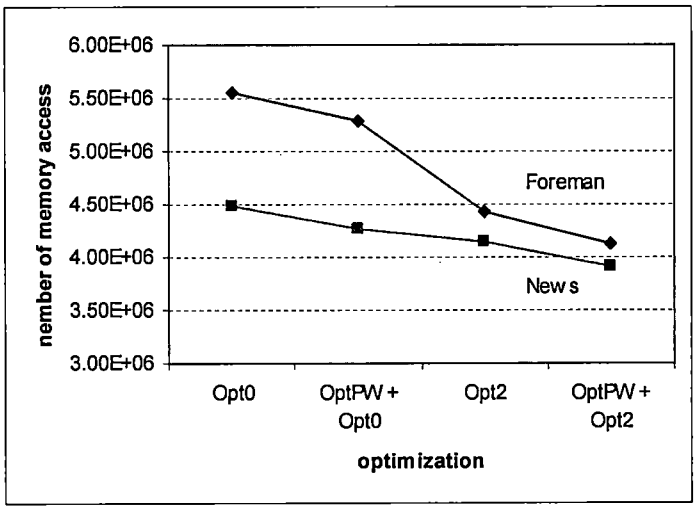
ดังในรูปที่ 12 แสดงถึงในเอ็นโค้ดเดอร์ จำนวนการไม่พบคำสั่งในแคชคำสั่งเยอะมากใน Opt0 แต่การออพติไมเซชั่นให้แคชชุดคำสั่งทำงานดีขึ้นมีการพัฒนามานานมากมายหลายวิธี และถูกใช้อย่างมากมายใน compiler ต่างๆ แต่วิธีการออพติไมเซชั่นพลังงานของเราที่ได้เสนอไปใน เป็นวิธีที่จะปรับปรุงการทำงานของแคชของข้อมูลให้ดีขึ้น วิธีการออพติไมซ์พลังงานของเรามีประสิทธิภาพในการลดจำนวนการไม่พบข้อมูลในแคชข้อมูล เพียง Opt2 อย่างเดียวลดได้แต่เพียงการไม่พบข้อมูลคำสั่งในแคชชุดคำสั่งและสามารถลดจำนวนการไม่พบข้อมูลในแคชข้อมูลได้น้อย รูปนี้แสดงให้เห็นว่าวิธีการของเรามีประสิทธิภาพทั้งที่แคชคำสั่งและแคชข้อมูลมี 16 กิโลไบต์ใน ARM 920T

2) การเข้าใช้หน่วยความจำ: ตารางที่ 2 แสดงจำนวนของการเข้าใช้หน่วยความจำ จำนวนการเข้าใช้หน่วยความจำทั้งหมดรวมกันจากจำนวนการอ่านข้อมูลในแคชคำสั่ง, จำนวนการอ่านข้อมูลในแคชข้อมูล และจำนวนการเขียนข้อมูลลงในแคชข้อมูลสำหรับสองวีดีโอทดสอบ จะเห็นว่าวิธีการออพติไมซ์ของเราสามารถลดจำนวนลดการเข้าใช้หน่วยความจำได้ทั้งการอ่านและเขียนข้อมูล แต่จะเพิ่มการอ่านชุดคำสั่งมากขึ้นนิดหน่อย ซึ่งคุ้มค่าเมื่อเทียบกับจำนวนการเข้าถึงหน่วยความจำทั้งหมดลดลงอย่างที่เห็น

ตารางที่ 2 จำนวนและชนิดของการเข้าใช้หน่วยความจำของทุกการออพติไมเซชั่นของเอ็นโค้ด-เดอร์ (15 เฟรมต่อวินาที, 10วีดีโอเฟรม)

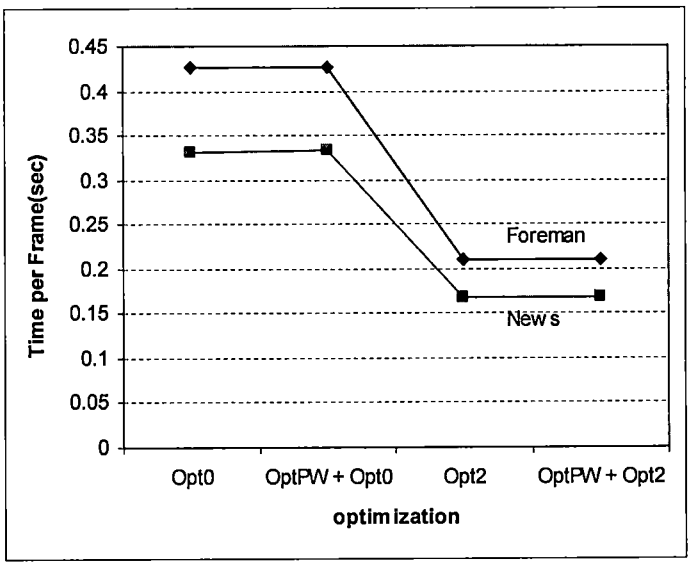
Video	Type	Opt0	OptPW+ Opt0	Opt2	OptPW+ Opt2
Foreman	Inst mem Read	1,465,461	1,537,685	326,756	335,052
	Data mem Read	1,176,691	984,378	1,176,723	1,009,866
	Data mem Write	2,917,379	2,770,846	2,923,459	2,784,276
	Total mem Access	5,559,531	5,292,909	4,426,938	4,129,194
News	Inst mem Read	490,949	516,623	142,852	154,463
	Data mem Read	1,090,907	968,382	1,090,476	971,573
	Data mem Write	2,905,996	2,787,725	2,912,728	2,790,617
	Total mem Access	4,487,852	4,272,730	4,146,056	3,916,653

รูปที่ 13 แสดงจำนวนการเข้าใช้หน่วยความจำทั้งหมดของทุกการออพติไมซ์สำหรับสองวีดีโอทดสอบ จากรูปนี้เราจะเห็นว่าวิธีการออพติไมซ์ของเรามีประสิทธิภาพในการลดจำนวนการเข้าใช้หน่วยความจำ และเมื่อใช้กับ opt2 ยังทำให้ลดการเข้าถึงหน่วยความจำลงไปอีก



รูปที่ 13 จำนวนของการเข้าใช้หน่วยความจำของทุกการออพติไมซ์ของเอ็นโค้ดเดอร์ (15 เฟรมต่อวินาที, 10 วีดีโอเฟรม)

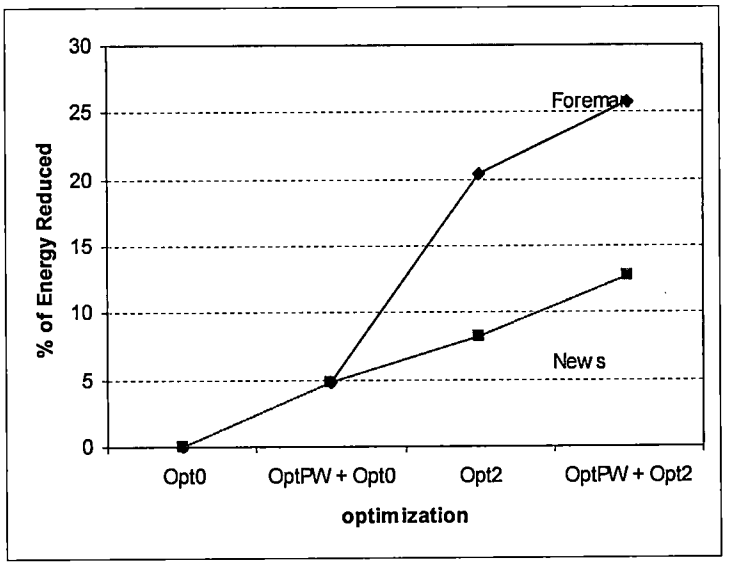
3) เวลาการทำงาน (Execution Time): รูปที่ 14 แสดงเวลาเฉลี่ยต่อหนึ่งเฟรมในการเอ็นโค้ดของแต่ละวิธีการออพติไมซ์ จากการสังเกตจะเห็นว่าวิธีของเรายังทำงานได้ดีกับคอมไพเลอร์ ไม่ได้ทำให้ใช้เวลามากขึ้นแต่อย่างไร



รูปที่ 14 เวลาในการทำงานต่อเฟรมสำหรับทุกการออพติไมซ์ของเอ็นโค้ดเดอร์ (24กิโลบิตต่อวินาที, 15เฟรมต่อวินาที, 10วีดีโอเฟรม)

4) เปอร์เซ็นต์การลดพลังงาน (Reduced Energy Percentage): จากสมการที่ 3 เราคำนวณเปอร์เซ็นต์การลดพลังงานโดยวิธีของเรากับวิธี Opt0 ผลที่ได้อยู่ในรูปที่ จะเห็นว่าวิธีการออพติไมซ์ของเรา

สามารถลดได้ 4.75 เปอร์เซ็นต์ของการใช้พลังงาน และเมื่อใช้กับ opt2 เพิ่มลดได้เพิ่มถึง 12.70 ถึง 25.72 ดังรูปที่ 15



รูปที่ 15 เปอร์เซ็นต์การลดการบริโภคพลังงานของทุกการออพติไมซ์ของเอ็นโค้ดเดอร์ (15 เฟรมต่อวินาที, 10 วีดีโอเฟรม)

5 สรุป

5.1สรุปงานวิจัยที่นำเสนอ

งานวิจัยนี้ได้ทำการเพิ่มประสิทธิภาพของ MPEG-4 Video encoder บนระบบปฏิบัติการซิมเบียน (Symbian OS) ซึ่งทำการแปลง source code มาจาก XviD MPEG-4 Video Codec มาทำงานบนระบบปฏิบัติการซิมเบียน โดยแต่เดิม XviD MPEG-4 Video encoder ทำงานบนพีซีซึ่งเป็น Open Source ของ XviD ปัญหาของ MPEG-4 Video Codec ที่ทำงานบนโทรศัพท์เคลื่อนที่คือหนึ่งมีโพรเซสเซอร์ที่มีความสามารถในการประมวลผลต่ำและมีแบตเตอรี่ที่จำกัด ดังนั้นเราจะทำการเพิ่มประสิทธิภาพความเร็วในการทำงานและลดการใช้พลังงานในการทำงาน โดยได้เสนอวิธีเพิ่มประสิทธิภาพซึ่งนำมาแก้ไข MPEG-4 Video Codec บนระบบปฏิบัติการซิมเบียนดังต่อไปนี้

การเพิ่มประสิทธิภาพด้านลดการใช้พลังงาน

- Cache-Conscious Motion Estimation Pattern: cache replacement policy จะมีผลกระทบต่อการลดการคำนวณและการเข้าใช้หน่วยความจำลงไป วีดีโอ Encoder โดยส่วนมากจะเข้าใช้ ME ตามแนวแถวจากบนลงล่าง ดังรูป 7(a) ในการทำ motion estimation MB ที่อยู่ขวาไกลอาจถึงแทนที่ pixel ก่อนหน้าเนื่องด้วยความกว้างของ frame ที่กว้างและ cache replacement policy ของ ARM920T ที่เป็น FIFO และ pattern ใหม่ที่แสดงในรูป 7(b) คือ ME จะเริ่มจาก MB ซ้ายไปขวาและบนลงล่างแต่ทำตาม column ก่อน ซึ่งทำให้ pixel ของ reference frame ใน data cache ถูกใช้มากขึ้นก่อนถูกไล่ออกไป โดยเราไม่ต้องออพติไมซ์ cache replacement policy

- Minimizing Buffer Allocation ในการพัฒนา C++ บนระบบปฏิบัติการซิมเบียนใน subroutine จะมีบ่อยๆที่ใช้บัฟเฟอร์ในการเก็บผลก่อนในระหว่างการคำนวณ จากที่สังเกตดูบัฟเฟอร์พวกนี้จะ allocate แบบ local แล้ว assign(written) ต่อมาจะเอาค่าไปใช้ (read) ท้ายสุดทำลายทิ้ง (deallocate) เมื่อ subroutine ถูกเรียกทำงาน ตำแหน่งหน่วยความจำจะถูก allocate ในตำแหน่งที่แตกต่างกันในแต่ล่ะครั้ง ซึ่งคือปัญหาที่ยากในการจัดการกับแคชข้อมูลที่มีขนาดจำกัด
วิธีของเราในการแก้ปัญหานี้โดยการประกาศบัฟเฟอร์ข้อมูลเป็น Attribute ของ class ทำให้สามารถเข้าสโคปที่ใหญ่ขึ้น โดยผลที่ได้จากที่บัฟเฟอร์ข้อมูล allocate ครั้งเดียวแล้วใช้มันอีกเรื่อยๆเป็นผลให้ cache miss ลดลง

- Read before Write เราพบว่าบัฟเฟอร์ที่ใช้เป็น temp จะถูกเขียน (ข้อมูลในระหว่างการคำนวณ) ครั้งแรกและจะถูกอ่านเอาข้อมูลไปใช้ต่อ ในการเขียนข้อมูลลงบัฟเฟอร์เป็นเหตุของการ write miss จึงเขียนลงหน่วยความจำเลย (write thought) ต่อมาจะเรียกใช้ข้อมูลในบัฟเฟอร์อีกก็จะไม่พบในแคช (read miss) โค้ดลักษณะแบบนี้เป็นโค้ด

ที่ไม่มีประสิทธิ์ภาพในการบริโภคพลังงานคือตอนแรกเขียนก็ไม่เจอในแคชต่อมาจะใช้ก็ไม่พบบนแคชทำให้เกิดการ data cache miss สองครั้งและเกิดการเข้าใช้หน่วยความจำที่ไม่จำเป็น

เราจึงเสนอทางแก้ง่ายคือทำการเริ่มตอนก็อ่านบัฟเฟอร์เพื่อให้บัฟเฟอร์เข้าไปอยู่ในแคชข้อมูลก่อนจะใช้งาน ดังนั้นจำนวนของการเข้าใช้หน่วยความจำจะลดลงเหมือนที่กินพลังงานน้อยลง

ซึ่งจากผลการทดลองนี้แสดงให้เห็นว่าจากการเพิ่มประสิทธิภาพลดการบริโภคพลังงาน ในด้านการลดพลังงานสามารถลดได้ถึง 4.5-5.7% ของ Encoder และถึงแม้เราจะใช้วิธีการเพิ่มประสิทธิภาพกับการออตไมเซชั่นของ compiler เช่น ARMCpp ที่ opt2 ก็ยังมีประสิทธิภาพอยู่

นอกจากนี้ผู้วิจัยได้นำหลักการนี้ไปนำเสนอในงานประชุมวิชาการ International Symposium on Wireless Pervasive Computing และได้รับการติดพิมพ์ดังข้างล่างนี้

P Pekdeepaiboonpol and S Kittitornkun, "Low Energy Optimization for MPEG-4 Video Encoder on ARM-based Mobile Phones", International Symposium on Wireless Pervasive Computing 2006, Thailand 2006

5.2 ปัญหาและอุปสรรค

1. ในปัจจุบัน MPEG-4 video codec บนโทรศัพท์มือถือไม่สามารถดาวน์โหลด source code มาทำวัดเปรียบเทียบความเร็วและพลังงานที่ใช้ในการทำงานได้โดยใช้ ARM developer suite เราจึงเปรียบเทียบให้ดู MPEG-4 video codec ก่อนและหลังการเพิ่มประสิทธิภาพ

2. จากด้วยข้อจำกัดทางเวลาในการวิจัยโครงงานนี้ ทางผู้วิจัยได้ดำเนินทำการพัฒนา encoder สำหรับบีบอัดเสียงแบบ MPEG4 แต่เดียวความซับซ้อนของโค๊ด MPEG4 audio encoder และการวิจัยหาวิธีลดพลังงานสำหรับ MPEG4 video encoder ที่มีจำนวนปริมาณงานเยอะ ทำให้ผู้วิจัยไม่สามารถพัฒนาได้เสร็จสิ้นทันในงานวิจัยนี้

5.3 แนวทางการพัฒนาต่อ

ในอนาคตเราจะพัฒนาเสียงเพิ่มเข้าไปใน MPEG-4 video codec ที่ทำงานบนระบบปฏิบัติการซิมเบียน โดยจะทำการ synchronization ระหว่างภาพและเสียง

6. เอกสารอ้างอิง (Reference) ของโครงการวิจัย

[P Pekdeepaiboonpol and S Kittitornkun, 2004] P Pekdeepaiboonpol and S Kittitornkun, "Performance Optimization of MPEG-4 Video Decoder on Symbian OS", Electrical Engineering Conference 27, Thailand 2004

[P Pekdeepaiboonpol and S Kittitornkun, 17 Nov 2005] P Pekdeepaiboonpol and S Kittitornkun, "ENERGY OPTIMIZATION FOR MOBILE MPEG-4 VIDEO DECODER", IEE MOBILITY CONFERENCE 2005, 15-17 November 2005, Guangzhou, China

[P Pekdeepaiboonpol and S Kittitornkun, 27-28 Oct 2005] P Pekdeepaiboonpol and S Kittitornkun, "Low Power Optimization Algorithms for ARM-based MPEG-4 Video Decoder" The 9th National Computer Science and Engineering Conference, Bangkok, THAILAND

[P Pekdeepaiboonpol and S Kittitornkun, 20-21 Oct 2005] P Pekdeepaiboonpol and S Kittitornkun, "Energy Optimization for MPEG-4 Video Decoder", Electrical Engineering Conference 28, Bangkok, Thailand 2005

[P Pekdeepaiboonpol and S Kittitornkun, 6-7 Dec 2005] P Pekdeepaiboonpol and S Kittitornkun, "POWER OPTIMIZATION FOR MOBILE MPEG-4 VIDEO DECODER", the Fifth International Conference on Information, Communications and Signal Processing 2005, Bangkok, Thailand 2005

[Wu et.al, 2000] D. Wu, Y.Hou, W. Zhu, H.-J. Lee, T. Chiang, Y.-Q. Zhang, and H.J. Chao,On End-to-End Architecture for Transporting MPEG-4 Video Over the Internet, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 10, NO. 6, SEPTEMBER 2000, pp. 923-41

[Yoon et. al, 2001] C.-W. Yoon, R. Woo, J. Kook, S.-J. Lee, K. Lee, and H.-J. Yoo, An 80/20-MHz 160-mW Multimedia Processor Integrated With Embedded DRAM, MPEG-4 Accelerator, and 3-D Rendering Engine for Mobile Applications, IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. 36, NO. 11, NOVEMBER 2001, pp.1758-67

[Budagavi et.al, 2000] M. Budagavi, W.R. Heinzelman, J. Webb, and R. Talluri, Wireless MPEG-4 Wideo COMMUNICATIONation on DSP Chips, IEEE Signal Processing Magazine, Jan. 2000, pp. 36-53

[Cavali et.al, 2002] F. Cavali, R. Cucchiara, M. Piccardi, and A. Prati,Performance Analysis of MPEG-4 Decoder and Encoder, 4th EURASIP-IEEE Region 8 Inter. Symp. on Video/Image

Processing and Multimedia COMMUNICATIONations (VIPromCom-2002), June 2002, pp.227-31

[Palkovic et.al, 2002] M. Palkovic M. Miranda K. Denolf P. Vos F. Catthoor, Systematic Address and Control Code Transformations for Performance Optimisation of a MPEG-4 Video Decoder, Proc. 15th Inter. Conf. on VLSI Design (VLSID'02), 2002

[Sikora, 1997] T. Sikora,The MPEG-4 Video Standard Verification Model, IEEE Trans. On Circuit and Systems for Video Technology, Feb., 1997, pp. 19-31

Low energy optimization for MPEG-4 video encoder on ARM-based mobile phones

Pakdeepaiboonpol, P. Kittitornkun, S.
Fac. of Eng., King Mongkut's Inst. of Technol., Bangkok, Thailand;

This paper appears in: Wireless Pervasive Computing, 2006 1st International Symposium on
Publication Date: 16-18 Jan. 2006
On page(s): 5 pp.-
ISBN: 0-7803-9410-0
INSPEC Accession Number: 9053900
Digital Object Identifier: 10.1109/ISWPC.2006.1613618
Posted online: 2006-04-10 08:28:58.0

Abstract
Most compiler optimization techniques concern most about speed. In this paper, we present two high-level power/energy optimization methods for ARM-based battery-powered embedded multimedia systems, e.g. mobile phones, pocket PCs, personal multimedia systems, etc. The experiments using MPEG-4 simple profile level 0 (SP@L0) video encoder on ARM920T with two QCIF video sequences 15 fps, 24 kbps show that the proposed techniques can complement the existing speed-oriented ones to achieve lower energy/power consumption up to 5.7% relative to all ARM C++ optimization levels despite the 16-KB instruction and 16-KB data caches of ARM 920T core.

Index Terms
Available to subscribers and IEEE members.

References
Available to subscribers and IEEE members.

Citing Documents
Available to subscribers and IEEE members.

Low Energy Optimization for MPEG-4 Video Encoder on ARM-based Mobile Phones

P. Pakdeepaiboonpol and S. Kittitornkun
Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Thailand 10520

Abstract— **Most compiler optimization techniques concern most about speed. In this paper, we present three high-level power/energy optimization methods for ARM-based battery-powered embedded multimedia systems, e.g. mobile phones, pocket PCs, personal multimedia systems, etc. The experiments using MPEG-4 simple profile level 0 (SP@L0) video encoder on ARM920T with two QCIF video sequences 15 fps, 24 kbps show that the proposed techniques can complement the existing speed-oriented ones to achieve lower energy/power consumption up to 5.7% relative to all ARM C++ optimization levels despite the 16-KB instruction and 16-KB data caches of ARM 920T core.**

Index Terms—**Low Power, Power Optimization, Video Coding, ARM, Cache**

I. INTRODUCTION

Real time MPEG-4 video applications have become commonplace for mobile smart phones, PDAs and personal multimedia systems. However, an MPEG-4 software video encoder on low-power RISC processors still requires significant amount of computational as well as battery power. The most popular RISC processor in these portable systems is ARM because of its low power consumption and smaller size [10]. Especially in mobile smart phones, the amount of processing power available is limited and has direct impact on the battery life.

Therefore, power/energy optimization of MPEG-4 encoder is an essential task for achieving effective low power consumption. However, most commercial compiler optimization techniques concern most about speed. This paper describes two simple yet effective energy optimization techniques for MPEG-4 simple profile video encoder on ARM920T processor.

The paper is organized as follows. Section 2 reviews some important low power optimization methods at high-level programming language. Later, our proposed methods are described in section 3. Extensive experimental results on commercial ARM simulator, ARMulator V.1.2, are illustrated and discussed in section 4. Finally, the paper is concluded and future works are suggested in section 5.

II. LITERATURE REVIEW

A. Low Power Optimization

Advanced multimedia systems and applications intrinsically have a high memory cost, making the design of high performance, low-power solutions a real challenge. [1] present high level optimization algorithm of video decoder on embedded cores. These approaches are proposed for optimizing data storage and transfer organization. They selected two target architectures: ARM7 RISC processor core and custom standard cell designs. Their approaches can reduce both clock cycles and instruction memory accesses, however, have little effects on ARM processor with big caches (e.g. ARM920T). Reference [4] and [9] advocate a methodology and tool that involve high-level platform independent optimizations called DTSE (Data Transfer and Storage Exploration). It is applied to an MPEG-4 video decoder, leading to high performance, reusable C code. But they did the experiments on PC platform. This may not work well on ARM processors.

B. ARM9 Architecture

ARM9 architecture is targeted at low-power 32 bit RISC processor for battery-powered smart phones and PDAs. To maximize instruction throughput, the integer pipeline consists of five stages: fetch, decode, execute, memory access and write register back. We picked the ARM920T because it is used in many mobile smart phones such as Nokia phones with Symbian Series 60 user interface. The ARM920T is a processor macrocell combining an ARM9TDMI™ processor core with:

- 16KB instruction and 16KB data caches
- Instruction and data Memory Management Units (MMUs)
- Write buffer
- An AMBA™ (Advanced Microprocessor Bus Architecture) bus interface
- 32bit data and instruction bus transferred.
- An Embedded Trace Macrocell (ETM) interface.

Both instruction and data caches are 64-way set associative [5].

III. PROPOSED OPTIMIZATION TECHNIQUES

Based on the well established power/energy model [4], our proposed methods are focused on reducing the number of memory/bus accesses at high-level language optimization.

A. Power/Energy Model

In embedded multimedia systems, it is confirmed that memory and bus accesses consume most of the energy between 50% and 80% [6] A simple estimation of the power consumption is shown in Eq. (1) and (2) [4]:

$$P_{Tr} = E_{Tr} * \frac{\#Transfers}{Second} \qquad (1)$$

$$E_{Tr} = f(\#words, \#bits) \quad (2)$$

where P_{Tr} is the amount of energy consumed per second and E_{Tr} is the amount of energy due to transferred data depending on the word size in bits. We assume that E_{Tr} is constant for each memory/bus transfer. As a result, the energy consumed through out the execution is proportional to the number of transfers occurred.

$$E_{Total} \propto \#Transfers \quad (3)$$

B. Proposed Optimization Methods

On cache write misses, ARM 920T processor uses write through and non-allocate policies. On cache write hits, the processor uses write back with a write back buffer. In addition, cache replacement policy of ARM920T is either FIFO or random.

Our proposed optimization algorithms, namely *cache-conscious motion estimation pattern, minimizing buffer allocation* and *read before write* methods, can reduce the number of memory/bus accesses by changing the program behavior to utilize caches more. This will result in less cache miss events that will eventually decrease memory/bus events on ARM microprocessors.

1) Cache-Conscious Motion Estimation Pattern: All video encoding algorithms spend most of the computing power on motion estimation (ME). Several fast ME algorithms have been proposed to save the number of computations and memory accesses, hence the amount of energy. However, cache size and cache replacement policy depend on the processor implementation. Most video encoder performs ME row by row from top to bottom just like Fig. 1(a). ME in the far right macroblocks may replace previous pixels in the data cache because the frame width is large and ARM920T's cache replacement policy is FIFO.

With the new pattern as shown in Fig. 1(b), ME starts from left to right MB and top to bottom but column by column. Reference frame pixels in the data cache are reused more and replaced less due to nonoptimal cache replacement policy. Therefore, we call it *Cache-Conscious Motion Estimation Pattern.*

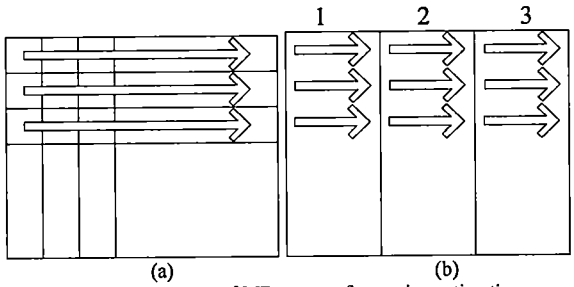


Fig. 1 Sequence of MBs access for motion estimation.
(a) Original pattern (b) New pattern

2) Minimizing Buffer Allocation Method [14]: In C++ implementations on Symbian operating system, some frequently called class methods (subroutines) utilize a number data buffers for temporary storage of intermediate results. We can observe that most of these buffers are locally allocated, assigned (written), accessed (read) and finally deallocated. Whenever the subroutine is called, different memory locations can be allocated to. This can make the problem even worse due to the fact data cache size is limited.

We propose a solution to this problem by declaring data buffers as class attributes accessible at a larger scope. As a result, these data buffers are allocated once and reused again and again causing less data cache misses as shown in Fig. 2

```
Void Cencoder::ConstructL()
{       mbtrans = new  Cmbtransquant();
        mbtrans->ConstructL();
        motion_comp = new  Cmotion_comp();
        motion_comp->ConstructL();
                •
                •
}

int Cencoder::FrameCodeP(Encoder * pEnc,, ...)
{       float fSigma;
        int16_t *dct_codes,*qcoeff;
        dct_codes = new int16_t[6*64];
        qcoeff = new int16_t[6*64];
                •
                •
}

Cencoder::FrameCodeI(Encoder * pEnc, ...)
{       int16_t *dct_codes,*qcoeff;
        dct_codes = new int16_t[6*64];
        qcoeff = new int16_t[6*64];
        uint16_t x, y;
        pEnc->iFrameNum = 0;
        pEnc->mbParam.m_rounding_type = 1;
                •
                •
}
```

(a)

```
Void Cencoder::ConstructL()
{       mbtrans = new  Cmbtransquant();
        mbtrans->ConstructL();
        motion_comp = new  Cmotion_comp();
        motion_comp->ConstructL();
        dct_codes = new int16_t[6*64];
        qcoeff = new int16_t[6*64];
                •
                •
}

int Cencoder::FrameCodeP(Encoder * pEnc,, ...)
{       float fSigma;
                •
                •
}

Cencoder::FrameCodeI(Encoder * pEnc, ...)
{       uint16_t x, y;
        pEnc->iFrameNum = 0;
        pEnc->mbParam.m_rounding_type = 1;
                •
                •
}
```

(b)

Fig. 2 Symbian C++ XviD MPEG-4 video encoder (a) before and (b) after Minimizing Buffer Allocation method.

This solution can entirely or partially retain the buffer in the data cache. Although ARM architecture has the feature called cache lock down [5], it is difficult for the C/C++ programmer to efficiently utilize.

3) Read before Write Method [14]: Furthermore, we have found that these temporarily allocated buffers are initialized (written) first and used (read) later. The initialization causes write misses and results in memory write through. Later accesses to the buffer will cause read misses. This naive coding sequence is inefficient in terms of energy consumption. Due to non-allocate write policy, these

temporary buffers cause cache write misses first and read misses again later which eventually results in twice data cache misses and memory accesses.

We propose a simple solution to this common pitfall. Initial accesses (reads) to the buffer can bring a cache line into data cache before the actual initialization. Therefore, the number of memory (bus) accesses can be reduced as well as the energy as shown in Fig. 3.

```
bool Cmotion_est::MotionEstimation(MBParam * const pParam,
                FRAMEINFO * const current,
                FRAMEINFO * const reference,
                const IMAGE * const pRefH,
                const IMAGE * const pRefV,
                const IMAGE * const pRefHV,
                const uint32_t iLimit)
{
    const uint32_t iWcount = pParam->mb_width;
    const uint32_t iHcount = pParam->mb_height;
    MACROBLOCK *const pMBs = current->mbs;
    MACROBLOCK *const prevMBs = reference->mbs;
    const IMAGE *const pCurrent = &current->image;
    const IMAGE *const pRef = &reference->image;

//load data in cache
for (y = 0; y < iHcount; y++)    {
    for (x = 0; x < iWcount; x ++)  {
    pMB = &pMBs[x + y * iWcount];
}}
                .
                .
```

Fig. 3 Piece of code from the MPEG-4 video encoder optimized with Read before Write method.

IV. EXPERIMENT SET UP AND RESULTS

A. Experiment Set Up

To demonstrate the effectiveness of our proposed methods, we use a simple-profile simple-level MPEG-4 video decoder. Fig. 4 shows our experiment set up. Our experiments are based on a XviD Coder/Decoder version 0.9 [7] ported to run on the Symbian operating system. ARM C++ compiler, ARMcpp, highly optimizes by default to ensure a small code size with three levels of speed optimization, Opt0 to Opt2.

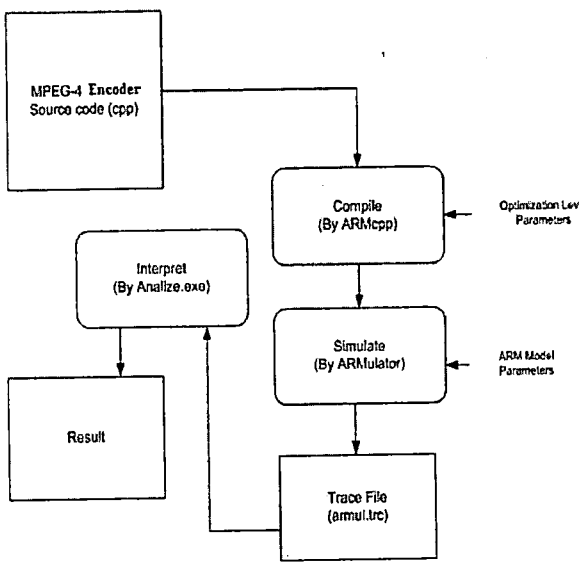


Fig. 4 Our experiment set up.

ARMulator [11] can model of several versions of ARM instruction set architecture. It can simulate, count the number of clock cycles, trace memory accesses and special events (i.e. D cache or I cache miss event, write back stall event). We used the ARMulator to simulate ARM920T with clock speed of 104MHz without floating-point unit. This configuration is similar to Nokia 6600.

Our two test QCIF video sequences including *Foreman* and *News* are encoded at 15-frame per second and 24 kilobits per second. Only ten video frames (one I and nine P frames) are decoded. Four different optimizations, Opt0, Opt2, Opt0+Ours, and Opt2+Ours, are experimented.

After each simulation, a trace file, *armul.trc*, containing processor events, e.g. memory/bus access, cache miss data and related information of the execution is output. We can interpret the trace files using our analysis program, *Analize.exe*, to extract information about memory bus accesses and cache events. Fig. 5 shows some part of armul.trc.

```
Date: Sat May 07 18:47:11 2005
Source: Armul
Options: Trace Events

E 00008000 00000000 10005
BSR4O___A0000000 00000C1E
E 00008000 00000000 10002
BSR8O___00008018 E240B001 E242C001
IT 00008000 e28f8090 ADD r8,pc,#0x90 ; #0x8098
IT 00008004 e898000f LDMIA r8,{r0-r3}
BNR4O___A0000000 00000C1E
BNR8O___00008098 00007804 00007828
BSR8O___00008080 10844009 E3C44003
BSR8O___00008088 E2555004 24847004
```

Fig. 5 Excerpt of "armul.trc"

We will show how to interpret memory bus accesses and special events.

1) *Memory Bus Trace (B lines):* The lines started with B indicate memory bus accesses. They have the following format for general memory accesses [12]:

B<type><rw><size>[O][L][S] <address> <data>

Where:
<type> indicates the cycle type:
S sequential
N nonsequential.
<rw> indicates either a read or a write operation:
R read
W write
<size> indicates the size of the memory bus access:
4 word (32 bits)
2 halfword (16 bits)
1 byte (8 bits)
O indicates an opcode fetch (instruction fetch).
L indicates a locked access (SWP instruction).
S indicates a speculative instruction fetch.
<address> gives the address in hexadecimal format, for example 00008008.
<data> gives the read/written value, for example EB00000C

2) *Events Trace (E lines):* The format of the event (E) lines is as follows [12]:

E <word1> <word2> <event_number>

For example: E 00000048 00000000 10001

where:
<word1> gives the first of a pair of words, such as the pc value.
<word2> gives the second of a pair of words, such as the aborting address.
<event_number> gives an event number, for example 0x10001. This is MMUEvent_DLineFetch. All Events are described in Table 1

TABLE I
Special events from tracing by ARMulator

Event name (MMUEvent_)	Word 1	Word 2	Event number
DLineFetch	Miss address	Victim address	0x10001
ILineFetch	Miss address	Victim address	0x10002

Fig. 6 shows the function diagram of ARM920T [5]. ARMulator traces information from trace interface port. It is configured to trace memory bus read and write from AMBA 32-bit bus interface.

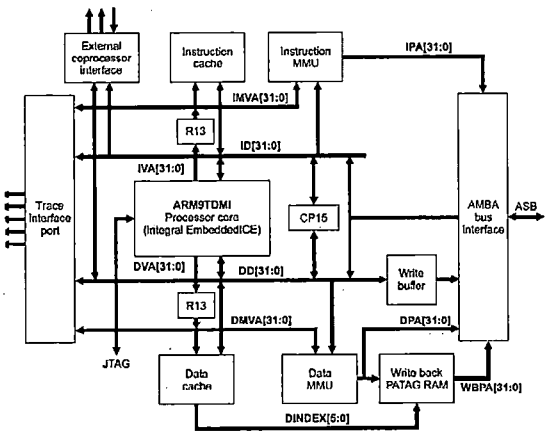


Fig. 6 ARM920T core diagram from ARM Limited [5]

B. Results and Discussions

We can evaluate the performance of our methods using the following parameters: cache misses, memory/bus accesses, execution time and the amount of energy reduced.

1) *Cache Misses:* In encoder, data memory accesses as well as instruction memory accesses consume more power.

Fig. 7 illustrates that our optimization methodologies can effectively reduce the total number of data cached misses while the number of instruction cache misses are about the same. Only Opt2 alone can reduce only instruction cache misses. This figure shows that our methods are efficient in spite of the fact that both data and instruction caches are of 16 KB each in ARM 920T.

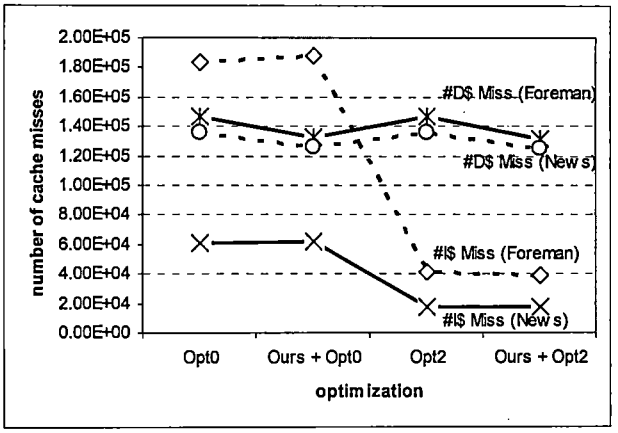


Fig. 7 Number of cache misses for each optimization (24 kbps, 15 fps, 10 video frames).

2) *Memory Accesses:* Table 2 shows the number of memory accesses in various categories. The total memory accesses combine instruction memory reads, data memory reads and data memory writes of each optimization for two video sequences. It can be observed that our optimization methods can reduce the number of memory accesses in almost all categories. As a result, the total number of memory accesses can be reduced accordingly.

TABLE II
Number and types of memory accesses for each optimization (15 fps, 10 video frames)

Video	Type	Opt0	Opt2	Ours+ Opt0	Ours+ Opt2
Foreman	Inst mem Read	1,465,461	326,756	1,537,685	335,052
	Data mem Read	1,176,691	1,176,723	984,378	1,009,866
	Data mem Write	2,917,379	2,923,459	2,770,846	2,784,276
	Total mem Access	5,559,531	4,426,938	5,292,909	4,129,194
News	Inst mem Read	490,949	142,852	516,623	154,463
	Data mem Read	1,090,907	1,090,476	968,382	971,573
	Data mem Write	2,905,996	2,912,728	2,787,725	2,790,617
	Total mem Access	4,487,852	4,146,056	4,272,730	3,916,653

Fig. 8 shows the total number of memory accesses of each optimization for two video sequences. It shows that our optimization methodologies can effectively reduce memory accesses; moreover our methods with ARMcpp Opt2 optimization can reduce memory accesses more then only ARMcpp Opt2 optimization.

3) *Execution Time:* Fig. 9 shows the average execution time per frame of the encoder at different optimization methods. It can be noticed that our methods work well with the commercial compiler.

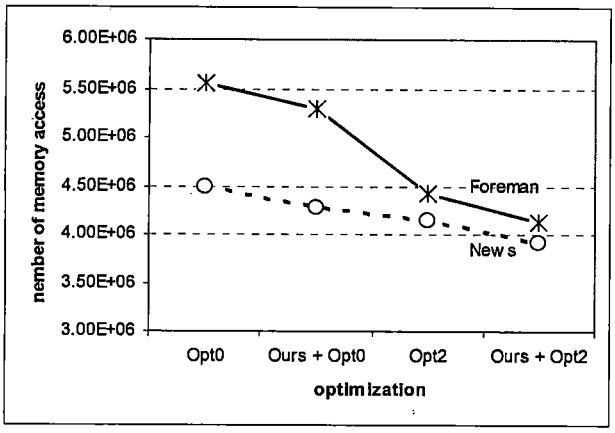


Fig. 8 Number of memory accesses of each optimization (15 fps, 10 video frames)

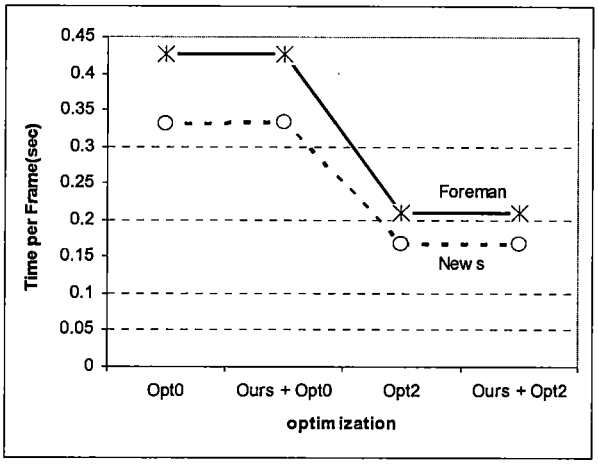


Fig. 9 Execution time per frame for each optimization (24 kbps, 15 fps, 10 video frames)

4) *Reduced Energy Percentage:* From Eq. (3), we can calculate reduced energy percentage optimized by our methods relative to Opt0 method using the following equation.

$$\%\ of\ Energy\ Reduced = \left(\frac{\#Tranfer_{opt0} - \#Tranfer_{new}}{\#Tranfer_{opt0}}\right) * 100 \quad (4)$$

The results are plotted in Fig. 10. It can be seen that our optimization can reduce 4-5-5.7 percents of energy consumption.

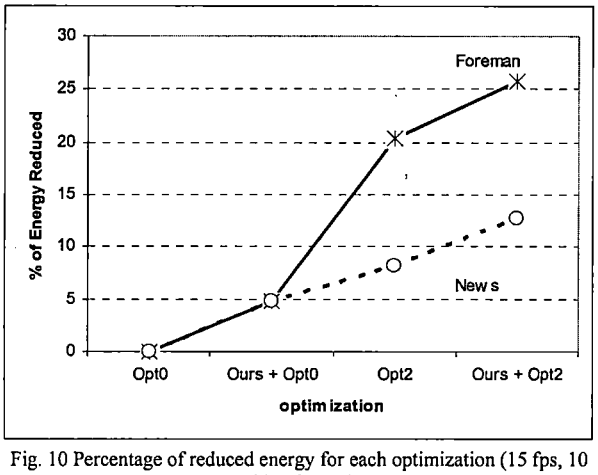


Fig. 10 Percentage of reduced energy for each optimization (15 fps, 10 video frames)

V. Conclusions

Most commercial compiler optimization techniques concern more about speed. In this paper, we have presented two high-level power/energy optimization methods for battery-powered ARM-based multimedia systems, e.g. mobile smart phones, PDAs, personal multimedia systems, etc. The experiments show that the proposed techniques can optimize a simple-profile simple-level MPEG-4 encoder to achieve lower energy/power consumption up to 5.7% relative to all ARM C++ optimization levels despite the 16-KB instruction and 16-KB data caches of ARM 920T core.

References

[1] L. Nachtergaele, D. Moolenaar, B. Vanhoof, F. Catthoor, and H. De Man, "System-level power optimization of video codecs on embedded cores : a systematic approach," Journal of VLSI Signal Processing Systems, vol. 18, no. 2, 1998, pp. 89-109

[2] T. Sikora, "The MPEG-4 video standard verification model", *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 7, No. 1, pp. 19-31, Feb 1997.

[3] M. Palkovic, M. Miranda, K. Denolf, P. Vos, F. Catthoor, "Systematic Address and Control Code Transformations for Performance Optimisation of a MPEG-4 Video Decoder", *ASP-DAC/VLSI Design 2002*, Jan 2002, pp. 547.

[4] K. Denolf, P. Vos, J. Bormans, and I. Bolsens. "Cost-efficient C-Level Design of anMPEG-4 Video Decoder", *PATMOS 2000*, Germany, Sep 2000, pp. 233-242

[5] ARM Limited, 20 April 2001, "ARM920T (Rev1) Technical Reference Manual"

[6] R. Gonzales and M. Horowitz, "Energy dissipation in general-purpose microprocessors". *IEEE J. of Solid-state Circ.*, SC-31(9):1277-1283, 1996

[7] XviD, http://www.xvid.org

[8] ARM Limited, 21 Nov 2001, "ARM Developer Suite - Version 1.2 - AXD and armsd Debuggers Guide"

[9] J. Bormans, K. Denolf, S. Wuytack, L. Nachtergaele, I. Bolsens, "Integrating system-level low power methodologies into a real-life design flow", *PATMOS'99 Ninth International Workshop Power and Timing Modeling, Optimization and Simulation*, pp. 19-28, Oct 6-8, 1999

[10] K. Ramkishor and V. Gunashree, "Real time implementation of MPEG-4 video decoder on ARM7TDMI," in IEEE Proceedings of International Symposium on Intelligent Multimedia, Video and Speech Processing, pp. 3653, 2001.

[11] K. Tatas, K. Siozios, D. Soudris, K. Masselos, K. Potamianos, S. Blionas, and A. Thanailakis, "Power Optimization Methodology for Multimedia Applications Implementation on Reconfigurable Platforms", PATMOS 2003, LNCS 2799, pp. 430–439, 2003.

[12] ARM Limited, 21 Nov 2001, "ARM Developer Suite - Version 1.2 - Debug Target Guide".

[13] ARM Limited, "ARM Programming Technique"

[14] P. Pakdeepaiboonpol and S. Kittitornkun, "ENERGY OPTIMIZATION FOR MOBILE MPEG-4 VIDEO DECODER," IEE MOBILITY CONFERENCE 2005, 15-17 Nov 2005.