บทที่ 3

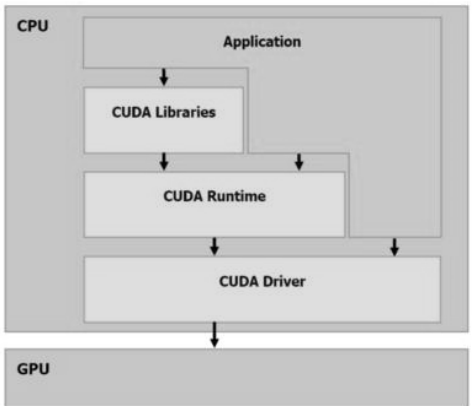
วิธีการดำเนินงานวิจัย

วิทยานิพนธ์ฉบับนี้พัฒนาระบบที่เรียกว่า **เวอร์ชวลคูด้า** มีวัตถุประสงค์เพื่อออกแบบการใช้ทรัพยากรจีพียูร่วมกันจากเครื่องที่ให้บริการทรัพยากรจีพียู (Server) ได้อย่างมีประสิทธิภาพ และวิธีการจัดสรรทรัพยากรจีพียูให้เพียงพอต่อการใช้งานสำหรับคูด้าแอพพลิเคชันโดยไม่จำกัดว่าผู้ที่ขอรับบริการใช้งานจีพียู (Client) นั้น จะเป็นคูด้าแอพพลิเคชันบนเวอร์ชวลแมชชีนหรือคูด้าแอพพลิเคชันบนเครื่องจริงก็ได้

โดยทั่วไป การเข้าถึงและสั่งงานจีพียูสำหรับการประมวลผลการทำงานของคูด้าแอพพลิเคชันนั้น แอพพลิเคชันจะเรียกใช้งานผ่านคูด้าไดรเวอร์สำหรับการเข้าถึงและสั่งงานไปยังอุปกรณ์จีพียู ไม่ว่าจะเรียกใช้งานในระดับต่ำหรือที่เรียกว่า CUDA driver API คือในส่วนของ CUDA Driver หรือระดับสูงที่เรียกว่า CUDA runtime API คือในส่วนของ CUDA Libraries และ CUDA Runtime ดังภาพที่ 3.1 แต่ทั้งนี้ในระบบคอมพิวเตอร์ที่มีการใช้งานเวอร์ชวลแมชชีนนั้น การเรียกใช้งานผ่านคูด้าไดรเวอร์เพื่อเข้าถึงและสั่งงานไปยังจีพียูจะไม่สามารถเรียกได้จากแอพพลิเคชันที่ทำงานอยู่บนเวอร์ชวลแมชชีน เนื่องจากเกสโอเปอร์เรติงซิสเต็มของเวอร์ชวลแมชชีนไม่สามารถเข้าถึงจีพียูได้โดยตรง

วิทยานิพนธ์ฉบับนี้จึงได้นำเสนอออกแบบและพัฒนาระบบเวอร์ชวลคูด้าเพื่อสนับสนุนการเข้าถึงและเรียกใช้งานจีพียูขึ้น โดยมีรายละเอียดการออกแบบดังต่อไปนี้

ภาพที่ 3.1 Compute Unified Device Architecture Software Stack

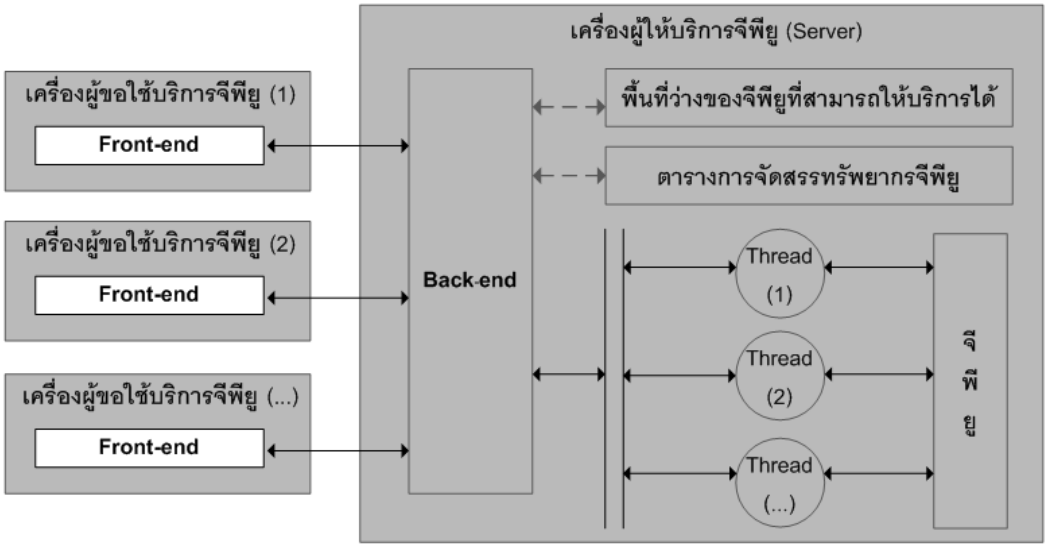


ที่มา : “NVIDIA CUDA Compute Unified Device Architecture” โดย nVidia,2008,น.12

3.1 การออกแบบระบบ

วิทยานิพนธ์ฉบับนี้ได้ออกแบบวิธีการเข้าถึงทรัพยากรจีพียูและวิธีการติดต่อสื่อสารระหว่างเครื่องที่ต้องการใช้ทรัพยากรจีพียูกับเครื่องที่ให้บริการทรัพยากรจีพียูโดยได้ออกแบบสถาปัตยกรรมการทำงานของระบบเป็นภาพโดยรวม ดังภาพที่ 3.2

ภาพที่ 3.2 สถาปัตยกรรมภาพรวมของระบบเวอร์ชวลคูด้า



การทำงานของระบบจากภาพที่ 3.2 สามารถอธิบายการลำดับขั้นตอนในทำงานได้ดังนี้

1. คูด้าแอพพลิเคชันบนเครื่องผู้ขอรับบริการจีพียูจะเรียกใช้งานจีพียูผ่านมาทางฟรอนท์เอนด์ไลบรารี (Front-end ดังภาพ) ที่จำลองคูด้าเอพีไอ บนเครื่องของตน

2. เมื่อคำร้องขอใช้จีพียูได้ถูกส่งมาถึงเครื่องผู้ให้บริการจีพียู เครื่องของผู้ให้บริการจะมีแบคเอ็นด์ซอฟต์แวร์ (Back-end ดังภาพ) คอยจัดการคำร้องขอการใช้ทรัพยากรจีพียู

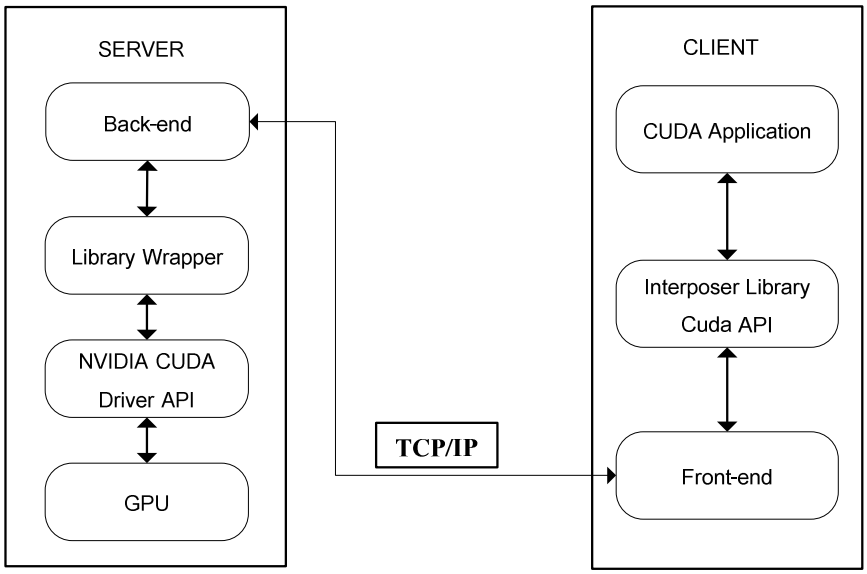
3. แบคเอ็นด์จะสร้างเทรด (Thread) การทำงานของผู้ขอรับใช้บริการจีพียูโดยจะสร้าง 1 เทรดการทำงานต่อ 1 ผู้ขอรับบริการ เพื่อให้สามารถเรียกใช้งานจีพียูพร้อม ๆ กันได้ครั้งละหลายผู้ขอรับบริการจีพียู

4. ระหว่างการให้บริการจีพียู แบ็คเอ็นด์จำเป็นต้องคอยตรวจสอบว่าทรัพยากรจีพียูมีหน่วยความจำสำหรับการให้บริการเพียงพอหรือไม่จากตารางการจัดการทรัพยากรที่แบ็คเอ็นด์สร้างขึ้น

3.1.1 การทำงานของผู้ให้บริการทรัพยากรจีพียู (Server) และผู้ขอใช้บริการทรัพยากรจีพียู (Client)

จากการอธิบายการทำงานภาพรวมสถาปัตยกรรมการทำงานของระบบเวอร์ชวลคูด้าสามารถจำแนกการทำงานของระบบเวอร์ชวลคูด้าออกเป็น 2 ส่วนการทำงานดังแสดงในภาพที่ 3.3 คือ 1) ส่วนการทำงานของผู้ให้บริการทรัพยากรจีพียู (Server) ซึ่งมีส่วนประกอบในการทำงานของ แบคเอ็นด์ (Back-end) และ Library Wrapper 2) ส่วนการทำงานของผู้ขอใช้ทรัพยากรจีพียู (Client) มีส่วนประกอบในการทำงานของ Interposer Library CUDA API และ ฟรอนท์เอนด์ (Front-end) โดยมีรายละเอียดในการทำงานของ Server และ Client แสดงในภาพที่ 3.3 ดังต่อไปนี้

ภาพที่ 3.3 การทำงานระหว่างเครื่องผู้ขอใช้บริการจีพียู (Client) กับเครื่องที่ให้บริการจีพียู (Server)



3.1.1.1 การทำงานของผู้ให้บริการทรัพยากรจีพียู (Server) มีรายละเอียดหน้าที่ในการทำงานดังนี้

1. แบคเอ็นด์ (Back-end) ทำหน้าที่เป็นตัวกลางผสานงานและจัดการ การเข้าถึงจีพียูของแต่ละคำสั่งตามคูด้าแอพพลิเคชันที่ถูกส่งมาจากฟรอนท์เอนด์ของเครื่องผู้ขอรับบริการจีพียู โดยนำคำสั่งที่ได้รับไปประมวลผลและเรียกใช้งานจีพียูให้ประมวล หลังจากประมวลผลเสร็จสิ้น แบคเอ็นด์จะนำผลที่ได้ส่งกลับไปยังฟรอนท์เอนด์ของเครื่องที่ขอใช้บริการจีพียูของแต่ละเครื่อง

สำหรับการเรียกใช้งานคูด้าแอพพลิเคชันของแบคเอ็นด์นั้นจำเป็นต้องมีการสร้างแอพพลิเคชันสำหรับเรียกใช้งานไว้ที่ผู้ให้บริการจีพียูสำหรับการเรียกใช้งานบนจีพียู โดยมีรายละเอียดดังต่อไปนี้

ก. เริ่มจากนำโปรแกรมสำหรับประมวลผลการปฏิบัติงานบนจีพียูแล้วนำมาแปลโปรแกรม (Compile) ด้วย NVCC เพื่อให้ได้ Cubin File โดยใช้ตัวเลือกให้ออกมาเป็น PTX Code

ข. นำ Cubin File ดังกล่าวไปวางไว้บนเครื่องของผู้ให้บริการทรัพยากรจีพียูสำหรับแบคเอ็นด์เรียกใช้งานโดยผ่าน Library Wrapper

2. Library Wrapper ทำหน้าที่แปลงคำสั่งจากฟรอนท์เอนด์ที่ต้องการเรียกใช้งานจีพียูด้วยคำสั่งของ CUDA Driver API ผ่าน NVIDIA CUDA Library ที่ถูกติดตั้งไว้บนเครื่องผู้ให้บริการจีพียูให้อยู่ในรูปของคำสั่ง CUDA Driver API ตามปกติ สำหรับใช้ในการติดต่อสื่อสารกับอุปกรณ์จีพียู เพื่อผสานการทำงานและประมวลผลการทำงานตามแต่ละคำสั่งที่ได้รับจากฟรอนท์เอนด์

3.1.1.2 การทำงานของผู้ขอใช้ทรัพยากรจีพียู (Client) มีรายละเอียดในการทำงานดังนี้

1. การทำงานของผู้ขอใช้ทรัพยากรจีพียูเริ่มต้นเมื่อมีการเรียกใช้งานคูด้าแอพพลิเคชันโดยส่วนที่เป็นคูด้าโค้ดที่ต้องการเข้าถึงจีพียูเพื่อใช้จีพียูในการประมวลผลการทำงาน ซึ่งโดยปกติแล้วคำสั่ง CUDA Driver API จะถูกเรียกใช้งานผ่าน NVIDIA CUDA Library แต่ในกรณีที่ผู้ขอใช้ทรัพยากรจีพียูไม่มีอุปกรณ์จีพียูเป็นของตนเองจึงจำเป็นต้องเข้าถึงจีพียู โดยผ่าน Interposer Library CUDA API สำหรับเรียกใช้งานคำสั่ง CUDA Driver API ที่อยู่บนเครื่องของผู้ให้บริการทรัพยากรจีพียู

2. Interposer Library CUDA API ทำหน้าที่เป็นตัวกลางผสานงานจัดการ การเรียกใช้งาน NVIDIA CUDA Library จากคูด้าแอพพลิเคชันของผู้ขอรับบริการจีพียู โดยไลบรารีที่ทำหน้าที่เป็นตัวกลางนี้จะถูกนำไปวางไว้บนเครื่องของผู้ขอรับบริการจีพียู และเมื่อไลบรารีถูกเรียกใช้งานจะทำการเตรียมข้อมูลที่ต้องส่งไปให้แบ็คเอ็นด์ เช่น ตัวแปร ขนาดของข้อมูล และประเภทคำสั่งของคูด้า นำทุกอย่างรวมกันเป็นแพคเกจแล้วจึงส่งแพคเกจนั้น ๆ ต่อให้ฟรอนท์เอนด์ทำงานต่อไป

3. ฟรอนท์เอนด์ (Front-end) ทำหน้าที่จัดการ การติดต่อสื่อสารระหว่างเครื่องผู้ให้บริการจีพียูกับเครื่องผู้ขอรับบริการจีพียู โดยเริ่มจากการสร้างช่องทางการติดต่อด้วยซ็อกเก็ต (Socket) ผ่านโปโตคอลทีซีพี/ไอพี (TCP/IP) แล้วจึงจัดส่งแพคเกจที่ได้รับจาก Interposer Library CUDA API ส่งไปตามช่องทางที่ถูกสร้างขึ้นไปที่แบคเอ็นด์ของเครื่องผู้ให้บริการทรัพยากรจีพียู ซึ่งการเลือกใช้โปโตคอลทีซีพีสำหรับฟรอนท์เอนด์นั้น เนื่องจากทีซีพีมีกลไกในการตรวจสอบการสูญหายของข้อมูลระหว่างการรับส่งข้อมูลต่างจากยูดีพี (UDP) ทำให้เชื่อมั่นได้ว่าข้อมูลที่ผ่านการรับส่งผ่านทีซีพีจะไม่เกิดการสูญหายระหว่างทาง

สำหรับฟังก์ชั่นการทำงานของ CUDA Driver API ที่อยู่ภายใน Library Wrapper และ Interposer Library CUDA API งานวิจัยนี้ได้รวบรวมฟังก์ชั่นพื้นฐานที่สำคัญของ CUDA Driver API ที่นำมาใช้สำหรับการประมวลผลปฏิบัติงานของคูด้าแอพพลิเคชัน โดยส่วนที่เป็นฟังก์ชั่นสำหรับประมวลผลปฏิบัติงานบนจีพียูนั้นจะถูกแปลโปรแกรม (Compile) ให้อยู่ในรูปของ Cubin File และวางอยู่ในส่วนของผู้ให้บริการทรัพยากรจีพียูสำหรับรอการถูกเรียกใช้งานจากแบคเอ็นด์ ฟังก์ชั่นพื้นฐานที่สำคัญของ CUDA Driver API ที่นำมาใช้ได้แก่

- cuInit
- cuDeviceGet
- cuCtxCreate
- cuModuleLoad
- cuModuleGetFunction
- cuMemAlloc
- cuMemcpyHtoD/ cuMemcpyDtoH
- cuParamSetv
- cuParamSetSize
- cuFuncSetBlockShape
- cuLaunchGrid
- cuCtxDetach
- cuMemFree

โดยมีเหตุผลในการเลือกใช้ฟังก์ชั่นพื้นฐานที่สำคัญของ CUDA Driver API ทั้ง 14 ฟังก์ชั่นนี้ เนื่องจากเป็นฟังก์ชั่นที่สำคัญพื้นฐานสำหรับการสั่งงานให้จีพียูสามารถประมวลผลการทำงานได้ และเวลาทำงานวิจัยที่มีอยู่อย่างจำกัด

3.1.2 การจัดสรรการเข้าถึงทรัพยากรจีพียู

ส่วนนี้กล่าวถึงการออกแบบการจัดสรรการเข้าถึงจีพียูของผู้ให้บริการจีพียูโดยแสดงขั้นตอนการทำงานโดยรวมดังภาพที่ 3.4 , 3.5 และ 3.6 ซึ่งการออกแบบการจัดสรรทรัพยากรจีพียูของระบบเวอร์ชวลคูด้า ได้ออกแบบให้มีตารางสำหรับการจัดสรรทรัพยากรเพื่อหลีกเลี่ยงการเกิดปัญหาการติดตาย อันเนื่องมาจากเหตุการณ์ที่แบ็คเอ็นด์ทำการจองพื้นที่บนจีพียูทันทีหลังจากได้รับคำสั่งจากฟรอนท์เอนด์ และคำสั่งที่รับมาพร้อม ๆ กันจากหลายเวอร์ชวลแมชชีนโดยไม่ใช่คำสั่งสำหรับสั่งให้จีพียูประมวลผลซึ่งอาจจะทำให้เกิดการรอคำสั่งประเภทดังกล่าวจนเกิดการติดตาย หรือเกิดการยกเลิกการประมวลผลของคูด้าแอพพลิเคชันกรณีมีพื้นที่ไม่เพียงพอต่อการประมวลผล อีกทั้งระบบเวอร์ชวลคูด้ายังได้ให้ความสำคัญกับลำดับในการเข้าถึงทรัพยากรจีพียูจากการร้องขอใช้ทรัพยากร และคำนึงถึงการเกิดปัญหาการติดตาย (Deadlock)

วิทยานิพนธ์ฉบับนี้จึงได้ออกแบบอัลกอลิทึมสำหรับจัดสรรการเข้าถึงทรัพยากรจีพียูออกเป็น 3 ขั้นตอน คือ 1) การจัดสรรทรัพยากรให้เพียงพอต่อการเรียกใช้งาน 2) การจัดสิทธิในการเข้าถึงทรัพยากรจากลำดับการขอเข้าใช้งานจีพียู 3) ตรวจสอบคำสั่งการปฏิบัติงาน cuLaunchGrid กรณีเรียกใช้งานซ้ำ โดยอธิบายรายละเอียดแต่ละขั้นตอนดังนี้

3.1.2.1 การจัดสรรทรัพยากรให้เพียงพอต่อการเรียกใช้งาน

การจัดสรรทรัพยากรจีพียูสำหรับผู้ขอใช้บริการทรัพยากรจีพียูนั้น วิทยานิพนธ์นี้ได้นำอัลกอริทึมของนายธนาคาร (Banker's Algorithm) เข้ามาประยุกต์ในการจัดสรรทรัพยากรเพื่อใช้ในการตรวจสอบทรัพยากรว่ามีเพียงพอต่อการให้บริการหรือไม่ อีกทั้งยังสามารถหลีกเลี่ยงปัญหาการติดตาย (Deadlock) ซึ่งการเลือกอัลกอริทึมของนายธนาคารมาใช้เนื่องจาก อัลกอริทึมของนายธนาคาร เป็นอัลกอริทึมที่ใช้หลีกเลี่ยงการเกิดปัญหาการติดตายแต่ไม่ใช่อัลกอริทึมเพื่อแก้ปัญหาการติดตายเหมือนกับ Detection Algorithm จึงเหมาะสมกับระบบเวอร์ชวลคูด้าที่จัดทำขึ้น โดยมีขั้นตอนการจัดสรรทรัพยากรดังนี้

1. เมื่อแบ็คเอ็นด์รับคำสั่งคูด้าแอพพลิเคชันมาจากเครื่องผู้ขอรับบริการแบ็คเอ็นด์จะทำหน้าที่ตรวจสอบก่อนว่าคำสั่งคูด้านั้นมีหน้าที่ใด กรณีเป็นคำสั่งเพื่อจองพื้นที่บนจีพียู

(cuMemAlloc) แบ็คเอ็นด์จะนำตัวแปร ขนาดของตัวแปร และหมายเลขเครื่องผู้ขอรับบริการจีพียู ไปเก็บไว้ยังตารางการจัดสรรทรัพยากร (ดังตารางที่ 3.1) เพื่อรอและนำไปใช้งานต่อไป อีกทั้งนำข้อมูลที่ถูกส่งมาจากฟรอนท์เอนด์ด้วยคำสั่ง cuMemcpyHtoD พักไว้บนแบ็คเอ็นด์สำหรับรอเรียกใช้งานในการถ่ายโอนข้อมูลไปยังจีพียูเมื่อมีการสั่งให้ปฏิบัติงานด้วยคำสั่ง cuLaunchGrid เกิดขึ้นจากการสั่งงานของฟรอนท์เอนด์ การนำทุกคำสั่งคูด้าแอพพลิเคชันจากฟรอนท์เอนด์มาเก็บในตารางการจัดสรรทรัพยากร เนื่องจากการทำงานด้วยวิธีนี้สามารถป้องกันการเกิดปัญหาติดตายและแอพพลิเคชันที่ประมวลผลไม่สำเร็จเนื่องจากพื้นที่สำหรับการใช้ประมวลผลไม่เพียงพอต่อการใช้งาน

2. กรณีที่แบ็คเอ็นด์รับคำสั่งคูด้าแอพพลิเคชัน ซึ่งส่งมาจากเครื่องผู้ขอรับบริการ และเป็นคำสั่งเพื่อเรียกใช้งานในฟังก์ชั่นของคูด้าแอพพลิเคชัน ซึ่งคำสั่งดังกล่าวคือคำสั่ง cuLaunchGrid ในส่วนการทำงานของแบ็คเอ็นด์จะทำการตรวจสอบก่อนว่าตัวแปรที่ฟังก์ชั่นต้องนำไปใช้นั้นมีอะไรบ้างและเมื่อทราบแล้ว แบ็คเอ็นด์จะรวมพื้นที่ทั้งหมดของตัวแปรดังกล่าวและนำไปตรวจสอบกับพื้นที่ของจีพียูที่ยังเหลืออยู่ว่ามีขนาดเพียงพอกับเนื้อที่ทั้งหมดของตัวแปรที่จะนำไปใช้หรือไม่ หากเพียงพอแบ็คเอ็นด์จะจัดการเรียกการทำงานผ่าน NVIDIA CUDA Library เพื่อจองพื้นที่ให้กับตัวแปรทั้งหมดพร้อมกันทำสัญลักษณ์ให้กับตัวแปรในตารางการจัดสรรทรัพยากรว่าตัวแปรดังกล่าวได้ถูกจองพื้นที่ไว้แล้วในหน่วยความจำของจีพียู การทำสัญลักษณ์ว่ามีการจองพื้นให้กับตัวแปรไปแล้วในตารางการจัดสรรทรัพยากรนี้มีวัตถุประสงค์เพื่อ เมื่อมีการเรียกใช้งานตัวแปรเดิมซ้ำแบ็คเอ็นด์จะไม่นำตัวแปรดังกล่าวไปจองพื้นที่ในหน่วยความจำจีพียูซึ่งตัวแปรทุกตัวจะถูกยกเลิกออกจากตารางการจัดสรรทรัพยากรและหน่วยความจำจีพียูด้วยคำสั่ง cuMemFree

3.1.2.2 การจัดสิทธิในการเข้าถึงทรัพยากรจากลำดับการขอเข้าใช้งานจีพียู

ส่วนนี้กล่าวถึงการออกแบบการจัดการการเข้าถึงจีพียูของผู้ให้บริการจีพียู โดยการจัดสิทธิสำหรับการเข้าถึงจีพียูจากลำดับการขอเข้าใช้งานจีพียูซึ่งการจัดการดังกล่าวนี้ จะจัดการในระดับคำสั่งของคูด้า ตามการเรียกใช้ของผู้ขอรับบริการจีพียู ด้วยวิธีมาก่อนจะได้รับสิทธิในการประมวลผลก่อน (FIFO) ซึ่งการจัดลำดับความสำคัญนี้จะให้ความสำคัญที่คำสั่ง cuLaunchGrid โดยมีรายละเอียดตามขั้นตอนดังนี้

1. เมื่อผู้ให้บริการจีพียูได้รับคำสั่ง cuLaunchGrid ในส่วนการทำงานของแบ็คเอ็นด์จะทำหน้าที่ให้ลำดับความสำคัญในการประมวลและหมายเลขเครื่องผู้ขอรับบริการจีพียู

2. หลังจากให้ลำดับความสำคัญในการประมวลผลแล้ว แบ็คเอ็นด์จะทำการสำรวจคำสั่งการให้ปฏิบัติงานของ cuLaunchGrid จากตารางการจัดสรรทรัพยากรและสั่งให้ cuLaunchGrid ที่มีลำดับการปฏิบัติงานสูงที่สุดปฏิบัติงาน โดยต้องผ่านการตรวจสอบใน 3.1.2.1) เป็นที่เรียบร้อยแล้ว

3. เมื่อการประมวลผลตามคำสั่งการปฏิบัติงานของ cuLaunchGrid เสร็จสิ้น หรือแบ็คเอ็นด์ได้รับคำสั่งให้ยกเลิกการจองพื้นที่ของตัวแปรในจีพียู แบ็คเอ็นด์จำเป็นต้องสำรวจตารางการจัดสรรทรัพยากรโดยทำตาม ขั้นตอนที่ 2) สำหรับการจัดสิทธิในการเข้าถึงทรัพยากรจากลำดับการขอเข้าใช้งานจีพียู จนกระทั่งไม่มีคำสั่ง cuLaunchGrid เหลืออยู่ในตารางจัดสรรทรัพยากร

3.1.2.3 ตรวจสอบคำสั่งการปฏิบัติงาน cuLaunchGrid กรณีเรียกใช้งานซ้ำ

ส่วนนี้กล่าวถึงการตรวจสอบคำสั่งปฏิบัติงาน cuLaunchGrid กรณีเรียกใช้งานซ้ำจากผู้ขอรับบริการจีพียู ซึ่งสั่งงานผ่านมาทางฟรอนท์เอนด์ และเมื่อคำสั่งดังกล่าวมาถึงแบ็คเอ็นด์ แบ็คเอ็นด์มีหน้าที่ต้องตรวจสอบ ตามรายละเอียดดังนี้

1. ตรวจสอบข้อมูลตัวแปรแต่ละตัวที่คำสั่ง cuLaunchGrid ต้องการเรียกใช้งานเพื่อนำไปประมวลผลว่ามีการจองพื้นที่แล้วบนจีพียูแล้วหรือไม่ กรณีที่มีการจองพื้นที่แล้วแบ็คเอ็นด์จะไม่ทำการจองพื้นที่ซ้ำอีกครั้ง

2. ตรวจสอบข้อมูลตัวแปรที่เป็น Input แต่ละตัวที่คำสั่ง cuLaunchGrid ต้องการเรียกใช้งานว่ามีการคัดลอกข้อมูลไปยังจีพียูแล้วหรือไม่ กรณีที่มีการคัดลอกแล้ว แบ็คเอ็นด์จะไม่ทำการคัดลอกซ้ำลงไปอีก

เมื่อทำการตรวจสอบตามข้อ 1. และ 2. เรียบร้อย และผลปรากฏว่าเป็นไปตามเงื่อนไข คือตัวแปรแต่ละตัวได้ถูกจองพื้นที่บนจีพียูแล้ว และตัวแปรที่เป็น Input ได้ถูกคัดลอกข้อมูลไปยังจีพียูแล้ว นั่นแสดงว่า cuLaunchGrid เคยถูกเรียกใช้งานมาก่อนและข้อมูลต่าง ๆ ที่อยู่บนจีพียูยังไม่เคยถูกล้างออกไป ซึ่งจะเป็นเหตุผลให้แบ็คเอ็นด์เรียกใช้งาน cuLaunchGrid ได้ทันทีโดยที่ไม่ต้องไปเริ่มทำคำสั่ง cuInit ถึง cuMemcpyHtoD ใหม่อีกครั้ง จึงเป็นการลดเวลาในการทำงานในขั้นตอนดังกล่าวสำหรับแบ็คเอ็นด์ หรือแม้แต่ฟรอนท์เอนด์หากต้องการเรียกใช้งาน cuLaunchGrid เดิมซ้ำ ก็ไม่จำเป็นต้องสั่งงาน cuInit ถึง cuMemcpyHtoD มียังแบ็คเอ็นด์อีก ซึ่งจะเป็นการลดเวลาในการถ่ายโอนข้อมูลมายังแบ็คเอ็นด์เป็นอย่างมาก

3.1.2.4 ตัวอย่างการจัดสรรการเข้าถึงทรัพยากรจีพียู

ตารางที่ 3.1 ตัวอย่างตารางการจัดสรรทรัพยากรตามคำสั่งคูด้าแอพพลิเคชัน

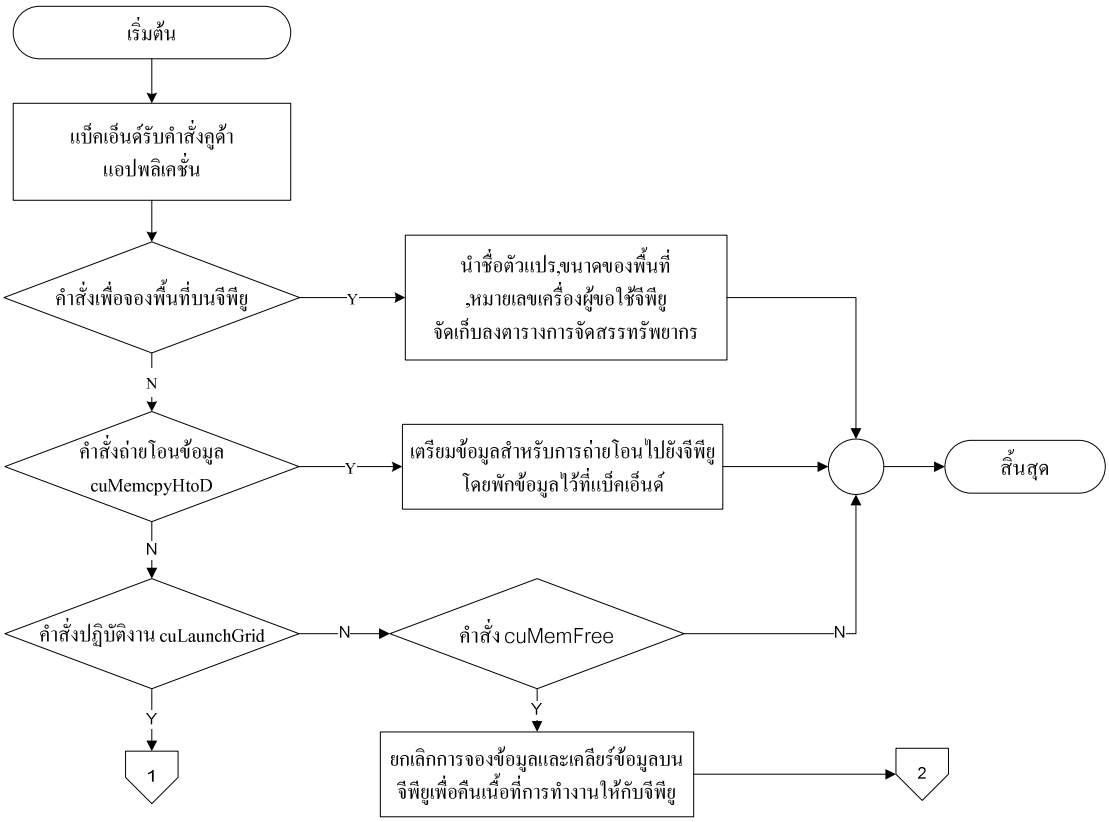
คำสั่ง	ชื่อตัวแปร	ขนาดข้อมูล	เครื่องผู้ขอใช้จีพียู	ลำดับการทำงาน	พื้นที่ถูกจอง
cuMemcpyHtoD	A	20MB	1	-	ไม่
cuMemcpyHtoD	A	30MB	2	-	ใช่
cuMemcpyHtoD	B	20MB	2	-	ใช่
cuMemcpyHtoD	B	50MB	1	-	ไม่
cuLaunchGrid	-	0	1	1	-
cuLaunchGrid	-	0	2	2	-

จากตารางที่ 3.1 สามารถอธิบายตัวอย่างการทำงานจัดสรรการเข้าถึงทรัพยากรจีพียูได้ดังนี้ คือตารางที่ 3.1 เป็นตารางการจัดสรรทรัพยากรจีพียูที่ฝังอยู่บนแบคเอ็นด์ของเครื่องผู้ให้บริการจีพียู โดยมีเหตุการณ์สมมุติดังนี้ เมื่อมีการเรียกใช้งานคำสั่งคูด้าฟังก์ชั่นในการจองพื้นที่สำหรับตัวแปร A จากเครื่องหมายเลข 1 ขนาดของข้อมูล 20 MB และมีประเภทการจองพื้นที่เป็น cuMemcpyHtoD คือขอจองพื้นที่เพื่อคัดลอกข้อมูลจากซีพียูไปยังจีพียู จากฟรอนท์เอนด์ของเครื่องเวอร์ชวลแมชชีน แบคเอ็นด์จะทำการตรวจสอบคำสั่งที่ส่งมาก่อนว่าเป็นคำสั่งปฏิบัติงาน cuLaunchGrid หรือไม่ กรณีตรวจแล้วไม่ใช่จะนำคำสั่ง cuMemcpyHtoD ชื่อตัวแปร A หมายเลขเครื่องและขนาดของข้อมูลไปเก็บไว้ในตารางการจัดสรร ดังแสดงในตารางที่ 3.1 และทำแบบเดียวกันนี้ไปเรื่อย ๆ จนกว่าคำสั่งที่ส่งมาจากฟรอนท์เอนด์จะเป็นคำสั่ง cuLaunchGrid ซึ่งในตารางที่ 3.1 ได้เก็บไว้ 4 ค่าคือ A,B ของเครื่องที่ 1 และ 2 กรณีตรวจสอบแล้วเป็นคำสั่ง cuLaunchGrid จริง แบคเอ็นด์จะยังไม่ทำการรันคูด้าแอพพลิเคชันแต่จะตรวจสอบพื้นที่ก่อนว่ามีเพียงพอสำหรับการประมวลผลหรือไม่ โดยจะรวมค่าตัวแปรทั้งหมดที่ต้องใช้สำหรับ cuLaunchGrid นั้น ๆ กรณีมีพื้นที่เพียงพอแบคเอ็นด์จะทำการประมวลผลคูด้าแอพพลิเคชันนั้น ๆ ทันที แต่ถ้าหากมีพื้นที่ไม่เพียงพอจะนำคำสั่ง cuLaunchGrid ไปเก็บไว้ในตารางการจัดสรรทรัพยากรจีพียู และให้ลำดับในการรอประมวลผลจนกว่ามีพื้นที่ว่างพอให้ประมวลผล จากตัวอย่างกำหนดให้พื้นที่ในการประมวลผลจีพียูเหลือเพียง 50 MB เพราะฉะนั้นจะเห็นได้ว่าเมื่อมีคำสั่ง

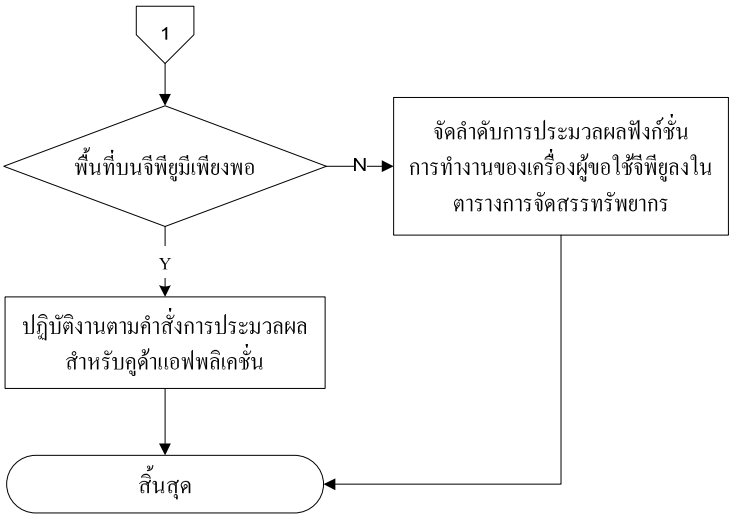
cuLaunchGrid ที่ส่งมาจากฟรอนท์เอนด์ จากตารางที่ 3.1 แถวที่ 5 ซึ่งเป็นของเครื่องหมายเลขที่ 1 พบว่าเมื่อแบคเอ็นด์ทำการตรวจสอบพื้นที่ต้องใช้ในการประมวลผลสำหรับเครื่องหมายเลข 1 นั้น พื้นที่ของตัวแปร A บวกกับพื้นที่ของ ตัวแปร B สำหรับใช้ในการประมวลผลคูด้าแอพพลิเคชันสำหรับคำสั่ง cuLaunchGrid นั้นมีพื้นที่มากเกินกว่าที่จีพียูจะสามารถให้บริการได้ แบคเอ็นด์จึงนำคำสั่ง cuLaunchGrid ของเครื่องหมายเลข 1 ไปเก็บไว้ในตารางการจัดสรรทรัพยากรจีพียู พร้อมกับระบุลำดับของสิทธิที่ได้รับสำหรับการทำงานเมื่อจีพียูมีพื้นที่เพียงพอ และต่อมาแบคเอ็นด์ได้รับคำสั่ง cuLaunchGrid ของเครื่องหมายเลข 2 ปรากฏว่าจากการตรวจสอบพบว่ามีพื้นที่เหลือเพียงพอในการประมวลผลจึงปล่อยให้ cuLaunchGrid ของเครื่องหมายเลข 2 ทำงาน และเมื่อมีพื้นที่ว่างจากการที่คูด้าแอพพลิเคชันก่อนหน้าประมวลผลเสร็จ cuLaunchGrid ของเครื่องหมายเลข 1 จะถูกทำงานทันทีเช่นกัน (ดังภาพที่ 3.4 , 3.5 และ 3.6)

จากตารางที่ 3.1 สามารถอธิบายตัวอย่างตรวจสอบคำสั่งการปฏิบัติงาน cuLaunchGrid กรณีเรียกใช้งานซ้ำได้ดังนี้ คือ กรณีที่เครื่องผู้ขอใช้จีพียูหมายเลข 2 เข้าไปสั่งงานจีพียูตามคำสั่ง cuLaunchGrid แสดงว่ามีพื้นที่บนจีพียูเพียงพอสำหรับให้บริการ ทำให้ตัวแปร A และ B สามารถจองพื้นที่ และคัดลอกข้อมูลลงไปที่จีพียูได้ ขณะที่ตัวแปร A และ B จองพื้นที่ และคัดลอกข้อมูลลงไปที่จีพียูได้นั้น แบคเอ็นด์จะทำสัญลักษณ์การจองพื้นที่ไว้บนตารางการจัดสรรว่าตัวแปรดังกล่าวเคยจองพื้นที่ และคัดลอกข้อมูลลงไปที่จีพียูแล้ว โดยจะถูกยกเลิกข้อมูลก็ต่อเมื่อพบคำสั่ง cuMemFree ทั้งนี้ทำให้เมื่อเครื่องผู้ขอใช้จีพียูหมายเลข 2 สั่ง cuLaunchGrid อีกครั้ง แบคเอ็นด์จะไปตรวจสอบข้อมูลตัวแปร A และ B ที่ตารางการจัดสรร ซึ่งจะพบว่าข้อมูลทั้งสองตัวแปรเคยถูกจองพื้นที่และมีข้อมูลอยู่บนจีพียูแล้ว ซึ่งจะทำให้แบคเอ็นด์สามารถสั่ง cuLaunchGrid ได้ทันทีโดยที่ไม่ต้องจองพื้นที่และคัดลอกข้อมูลลงไปที่จีพียูซ้ำอีกครั้ง (ดังภาพที่ 3.7)

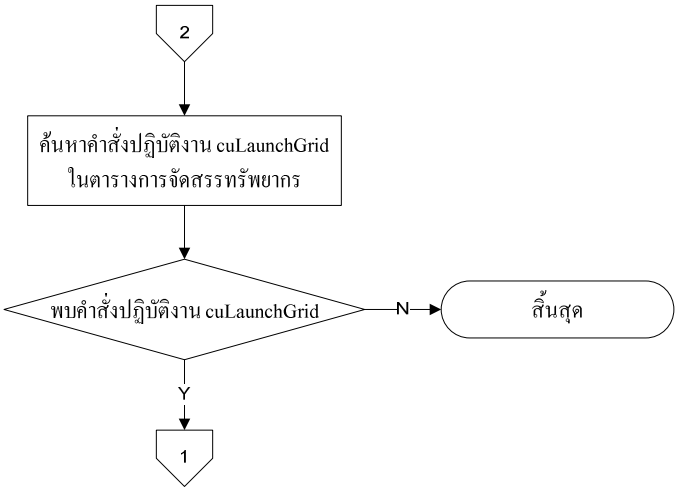
ภาพที่ 3.4 ภาพรวมการจัดสรรการเข้าถึงทรัพยากรจีพียูของระบบเวอร์ชวลคูด้า สำหรับขั้นตอนการตรวจสอบแต่ละประเภทคำสั่ง



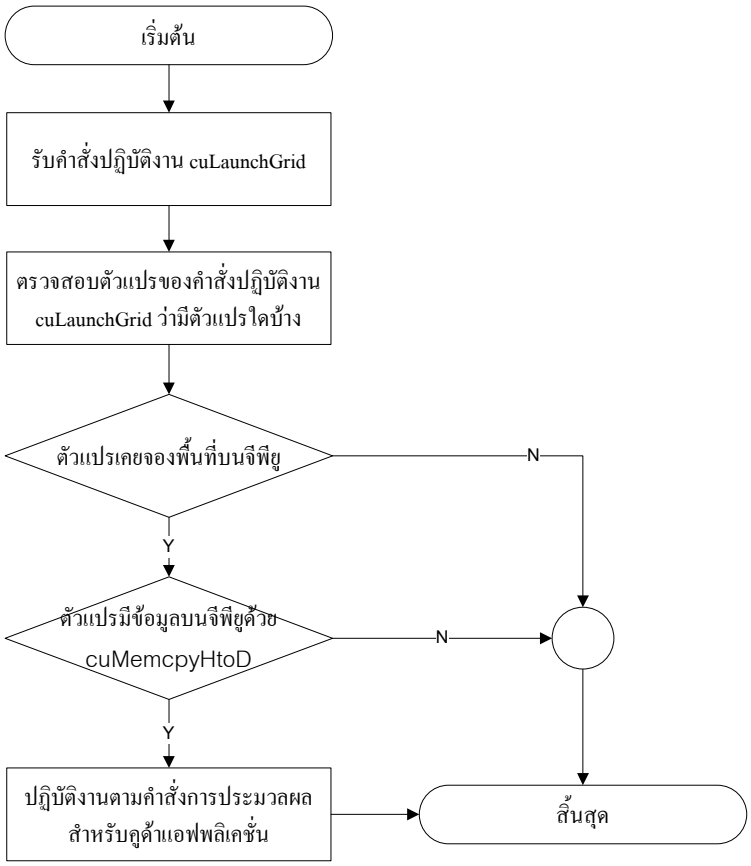
ภาพที่ 3.5 ภาพรวมการจัดสรรการเข้าถึงทรัพยากรจีพียูของระบบเวอร์ชวลคูด้า สำหรับขั้นตอนการตรวจสอบพื้นที่บนจีพียูเพื่อประมวลผล



ภาพที่ 3.6 ภาพรวมการจัดสรรการเข้าถึงทรัพยากรจีพียูของระบบเวอร์ชวลคูด้า สำหรับขั้นตอนการตรวจสอบคำสั่งปฏิบัติงานในตารางจัดสรร



ภาพที่ 3.7 ภาพตรวจสอบคำสั่งการปฏิบัติงาน cuLaunchGrid กรณีเรียกใช้งานซ้ำของระบบเวอร์ชวลคูด้า



3.1.3 ตัวอย่างโปรแกรมสำหรับเรียกใช้งานจีพียูด้วยคูด้าแอพพลิเคชันโดยผ่านระบบเวอร์ชวลคูด้า

ระบบเวอร์ชวลคูด้าที่ได้พัฒนาขึ้นมานั้น จะทำงานโดยเข้าถึงจีพียูด้วยคำสั่งของคูด้าแบบ CUDA driver API โดยจะใช้ฟังก์ชั่นพื้นฐานที่ได้กล่าวไว้ในข้อ 3.1.1.2 โดยแสดงการเรียกใช้งาน ดังนี้

```
#include "functioncuda_client.h"
```

```
void matrixMul(int argc, char** argv)
{
    int sockfd=connectServer();

   // set seed for rand()
   srand(2006);

   unsigned int uiWA, uiHA, uiWB, uiHB, uiWC, uiHC;
   int iSizeMultiple = 1;
   uiWA = WA * iSizeMultiple;
   uiHA = HA * iSizeMultiple;
   uiWB = WB * iSizeMultiple;
   uiHB = HB * iSizeMultiple;
   uiWC = WC * iSizeMultiple;
   uiHC = HC * iSizeMultiple;

   unsigned int size_A = uiWA * uiHA;
   unsigned int mem_size_A = sizeof(float) * size_A;
   float* h_A = (float*)malloc(mem_size_A);
   unsigned int size_B = uiWB * uiHB;
   unsigned int mem_size_B = sizeof(float) * size_B;
   float* h_B = (float*)malloc(mem_size_B);

   // initialize host memory
   randomInit(h_A, size_A);
```

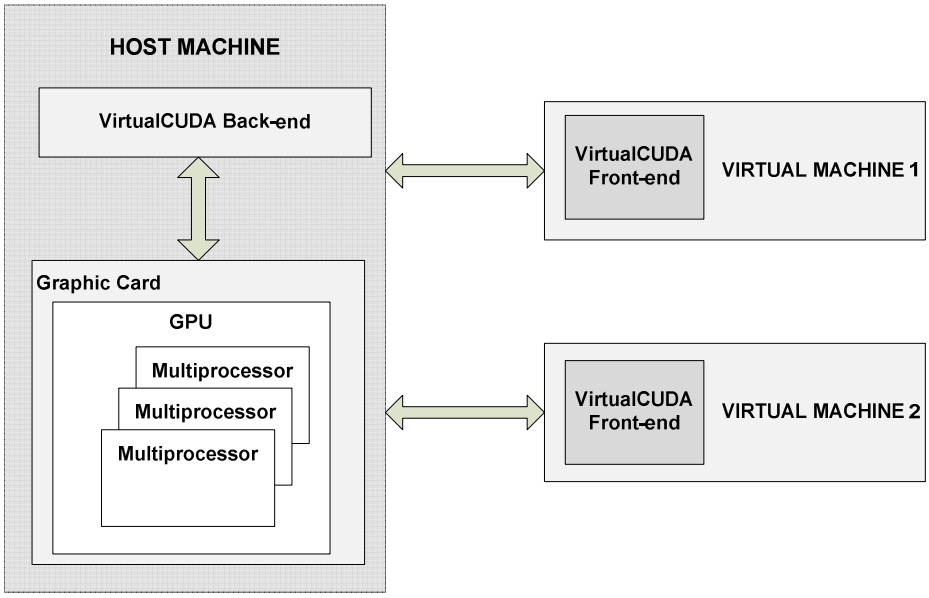
```
randomInit(h_B, size_B);

cuInitialize(sockfd);
cuLoadFunction("matrixMul",sockfd);

unsigned int size_C = uiWC * uiHC;
unsigned int mem_size_C = sizeof(float) * size_C;

cudaMalloc((void**) &d_C, mem_size_C);*/
cuMemAllocDevice(mem_size_C,"d_C",sockfd);
cuMemAllocDevice(mem_size_A,"d_A",sockfd);
cuMemAllocDevice(mem_size_B,"d_B",sockfd);

cuMemcpy("d_A",h_A,mem_size_A,HtoD,sockfd);
cuMemcpy("d_B",h_B,mem_size_B,HtoD,sockfd);

cudaCallFunction(sockfd,BLOCK_SIZE,BLOCK_SIZE,1,GridX,GridY);
cuMemcpy("d_C",h_C,mem_size_C,DtoH,sockfd));

free(h_A);
free(h_B);
free(h_C);
cudaMemFree("d_A",sockfd);
cudaMemFree("d_B",sockfd);
cudaMemFree("d_C",sockfd);
}
```

3.2 วิธีการทดลอง

3.2.1 เครื่องมือที่ใช้ในการทดลอง

ในการทดลองนี้เรากำหนดให้มีเวอร์ชวลแมชชีนสองเครื่องสามารถเรียกใช้งานทรัพยากรจีพียูบนเครื่องของโฮสแมชชีน โดยผ่านระบบเวอร์ชวลคูด้า ดังภาพที่ 3.8 และสองเวอร์ชวลแมชชีน ดังกล่าวจะถูกรันอยู่บนเครื่องโฮสเครื่องเดียวกัน ซึ่งแสดงรายละเอียดของ โฮสแมชชีน เวอร์ชวลแมชชีน และ Graphic Card ดังต่อไปนี้

ภาพที่ 3.8 การทำงานของโฮสแมชชีนและสองเวอร์ชวลแมชชีน



1. โฮสแมชชีนสำหรับแบ็คเอ็นด์ มีรายละเอียดดังนี้
   - Architecture : x86_64
   - Processor : Intel(R) Xeon(R) CPU E5530 @ 2.40GHz
   - CPU : 4
   - Memory capacity : 8GB
   - OS : Ubuntu 10.10
   - Kernel : 2.6.35-24-server
2. เวอร์ชวลแมชชีนแบบ KVM เครื่องที่ 1 มีรายละเอียดดังนี้

- Architecture : x86_64
- Processor : QEMU Virtual CPU version 0.12.5
- Memory capacity : 1GB
- OS : Ubuntu 9.04
- Kernel: 2.6.28-13-generic

3. เวอร์ชวลแมชชีนแบบ KVM เครื่องที่ 2 มีรายละเอียดดังนี้
   - Architecture : x86_64
   - Processor : QEMU Virtual CPU version 0.12.5
   - Memory capacity : 1GB
   - OS : Ubuntu 9.04
   - Kernel : 2.6.28-13-generic
4. ไฮเปอร์ไวเซอร์ kvm-88
5. Graphic Card
   - รุ่น : NVIDIA GeForce 8800GTS
   - CUDA Driver Version : 3.2
   - CUDA Cores : 96
   - Core Clock (MHz) : 500
   - Shader Clock (MHz) : 1200
   - Memory Clock (MHz) : 800
   - Memory Amount : 640MB
   - Memory Interface : 320-bit
   - Memory Bandwidth (GB/sec) : 64

3.2.2 การออกแบบการทดลอง

ในส่วนนี้อธิบายถึงการออกแบบการทดลองสำหรับการขอรับบริการใช้งานทรัพยากรจีพียูของเวอร์ชวลแมชชีนโดยแบ่งการทดลองออกเป็น 3 แบบคือ 1) การทดลองประมวลผลผ่านซีพียูของเวอร์ชวลแมชชีนเปรียบเทียบกับการประมวลผลผ่านเวอร์ชวลคู่ค้า 2) การทดลองเพื่อวัด

ระยะเวลาที่เรียกใช้งานจีพียูผ่านระบบเวอร์ชวลคูด้า 3) การทดลองการใช้งานจีพียูร่วมกันของเวอร์ชวลแมชชีนผ่านระบบเวอร์ชวลคูด้า โดยมีรายละเอียดในการทดลองแต่ละแบบดังต่อไปนี้

3.2.2.1 แบบที่ 1 การทดลองประมวลผลผ่านซีพียูของเวอร์ชวลแมชชีนเปรียบเทียบกับการประมวลผลผ่านเวอร์ชวลคูด้า มีรายละเอียดดังต่อไปนี้

• <u>วิธีการทดลอง</u>

1. ทำการรันแอพพลิเคชันบนเครื่องเวอร์ชวลแมชชีนโดยที่แอพพลิเคชันดังกล่าวประมวลผลบนซีพียู จากนั้นทำการจับเวลาทั้งหมดที่ใช้ไปสำหรับการประมวลผล

2. นำเอาแอพพลิเคชันเดียวกันที่เคยประมวลผลบนซีพียูมาดัดแปลงให้เป็นคูด้าแอพพลิเคชันเพื่อให้สามารถประมวลผลบนจีพียูได้ โดยการประมวลผลของคูด้าแอพพลิเคชันดังกล่าวนี้จำเป็นต้องประมวลผลผ่านระบบเวอร์ชวลคูด้า จากนั้นทำการจับเวลาทั้งหมดที่ใช้ไปสำหรับการประมวลผล

3. นำเวลาที่ได้จากการประมวลผลทั้งในข้อ 1 และข้อ 2 มาเปรียบเทียบกัน

• <u>เป้าหมายการทดลอง</u>

1. เพื่อวัดประสิทธิภาพในการประมวลผลและการใช้ประโยชน์ (Utilization) ที่เพิ่มมากขึ้น เมื่อนำจีพียูเข้ามาช่วยในการประมวลผล

2. เพื่อแสดงให้เห็นความสามารถในการทำงานสำหรับการเข้าถึงจีพียูของระบบเวอร์ชวลคูด้าจากเวอร์ชวลแมชชีน

• <u>สมมติฐานการทดลอง</u>

การประมวลผลผ่านคูด้าแอพพลิเคชันเพื่อเรียกใช้งานจีพียูผ่านเวอร์ชวลคูด้าการประมวลผลมีประสิทธิภาพที่ดีกว่าการประมวลผลผ่านซีพียูของเวอร์ชวลแมชชีน

3.2.2.2 แบบที่ 2 การทดลองเพื่อวัดระยะเวลาที่เรียกใช้งานจีพียูผ่านระบบเวอร์ชวลคูด้า มีรายละเอียดดังต่อไปนี้

• <u>วิธีการทดลอง</u>

1. รันคูด้าแอพพลิเคชันสำหรับเรียกใช้งานจีพียูบนเวอร์ชวลแมชชีนผ่านระบบเวอร์ชวลคูด้า

2. วัดระยะการทำงานของแบ็คเอ็นด์ และฟรอนท์เอนด์ ตั้งแต่ Initialization (คำสั่ง cuInit) ถึง Function Call (คำสั่ง cuLaunchGrid) จากนั้นทำการเปรียบเทียบระยะเวลาที่ได้ดังกล่าวระหว่างแบ็คเอ็นด์ กับฟรอนท์เอนด์

3. วัดระยะการทำงานของฟังก์ชั่นการคัดลอกข้อมูลแบบ Host to Device (จากซีพียูไปจีพียู) ของแบ็คเอ็นด์ และฟรอนท์เอนด์ จากนั้นทำการเปรียบเทียบระยะเวลาที่ได้ดังกล่าวระหว่างแบ็คเอ็นด์ กับฟรอนท์เอนด์

4. วัดระยะการทำงานของฟังก์ชั่นการคัดลอกข้อมูลแบบ Device to Host (จากจีพียูไปซีพียู) ระหว่างแบ็คเอ็นด์ และฟรอนท์เอนด์ จากนั้นทำการเปรียบเทียบระยะเวลาที่ได้ดังกล่าวระหว่างแบ็คเอ็นด์ กับฟรอนท์เอนด์

• <u>เป้าหมายการทดลอง</u>

1. เพื่อวัดผลและวิเคราะห์โอเวอร์เฮดการประมวลผลของคูด้าแอพพลิเคชันสำหรับแบ็คเอ็นด์ และฟรอนท์เอนด์กรณีเรียกใช้งานจีพียูผ่านระบบ เวอร์ชวลคูด้า

2. เพื่อวัดผลและวิเคราะห์โอเวอร์เฮดการถ่ายโอนข้อมูลระหว่างแบ็คเอ็นด์และฟรอนท์เอนด์ กรณีเรียกใช้งานจีพียูผ่านระบบเวอร์ชวลคูด้า

• <u>สมมติฐานการทดลอง</u>

1. ระยะเวลาการประมวลผลของแบ็คเอ็นด์ให้ผลที่ดีกว่าการประมวลผลของฟรอนท์เอนด์

2. การถ่ายโอนข้อมูลไปมาระหว่างแบ็คเอ็นด์ และฟรอนท์เอนด์มีผลทำให้ระยะเวลาในการประมวลของคูด้าแอพพลิเคชันบนเวอร์ชวลแมชชีนเพิ่มมากขึ้นตามขนาดของข้อมูลที่ถูกถ่ายโอน

3.2.2.3 แบบที่ 3 การทดลองการใช้งานจีพียูร่วมกันของเวอร์ชวลแมชชีนผ่านระบบเวอร์ชวลคูด้า มีรายละเอียดดังต่อไปนี้

• <u>วิธีการทดลอง</u>

1. ทำการรันคูด้าแอพพลิเคชันจากเครื่องเวอร์ชวลแมชชีนผ่านระบบเวอร์ชวลคูด้า เพื่อร้องขอใช้บริการจีพียูมายังเครื่องผู้ให้บริการ กรณีเวอร์ชวลแมชชีนมากกว่า 1 เครื่อง แต่แบ็คเอ็นด์สามารถขอใช้บริการจีพียูได้เพียงครั้งละ 1 เวอร์ชวลแมชชีน โดยที่แบ็คเอ็นด์ไม่แบ่งเทรดการทำงานให้กับเวอร์ชวลแมชชีนแต่ละเครื่องที่เข้ามาขอใช้บริการจีพียีทำให้เวอร์ชวลแมชชีนต้องต่อคิวเพื่อเข้าใช้งานจีพียู

2. เพิ่มประสิทธิแก่ระบบเวอร์ชวลคูด้าในการให้บริการจีพียู สำหรับการรันคูด้าแอพพลิเคชันบนเครื่องเวอร์ชวลแมชชีน ด้วยการเพิ่มความสามารถให้ กับแบ็คเอ็นด์ในฝั่งของผู้ให้บริการจีพียูให้สามารถรองรับการทำงานของเวอร์ชวลแมชชีน มากกว่า 1 เครื่อง กรณีขอเข้าใช้งานจีพียูพร้อมกันเพื่อประโยชน์ในการใช้ทรัพยากรจีพียูแบบเต็มประสิทธิภาพมากที่สุด

3. กำหนดให้คูด้าแอพพลิเคชันที่ใช้ในการประมวลผลทั้งในข้อ 1 และข้อ 2 เป็นคูด้าแอพพลิเคชันเดียวกัน

• <u>เป้าหมายการทดลอง</u>

1. เพื่อวัดประสิทธิภาพและเปรียบเทียบการทำงานในข้อ 1 และข้อ 2 ว่ามีการใช้ประโยชน์ (Utilization) สำหรับการเรียกใช้งานจีพียูกรณีใช้งานร่วมกันเพิ่มขึ้นหรือไม่

2. เพื่อวัดผลและวิเคราะห์โอเวอร์เฮดสำหรับเวลาที่ใช้ไปในการประมวลผลของคูด้าแอพพลิเคชัน กรณีที่ขอใช้ทรัพยากรจีพียูร่วม

• <u>สมมติฐานการทดลอง</u>

1. เวอร์ชวลคูด้าสามารถรองรับเวอร์ชวลแมชชีนกรณีเรียกใช้งานจีพียูพร้อมกันสองเครื่องได้

2. การเข้าใช้งานจีพียูพร้อมกันของสองเวอร์ชวลแมชชีนได้โดยผ่านเวอร์ชวลคูด้ามีประสิทธิภาพในการทำงานที่ดีกว่าการเข้าใช้งานจีพียูโดยผ่านเวอร์ชวลคูด้าครั้งละหนึ่งเวอร์ชวลแมชชีน

3.2.3 **การวิเคราะห์ข้อมูลและการวัดผล**

1. วิเคราะห์ประสิทธิภาพเปรียบเทียบระหว่างการประมวลผลบนซีพียูและบนจีพียูผ่านเวอร์ชวลคูด้าของเวอร์ชวลแมชชีน

2. วัดผลและวิเคราะห์โอเวอร์เฮดของการใช้งานคูด้าแอพพลิเคชัน สำหรับแบ็คเอ็นด์ และฟรอนท์เอนด์

3. วิเคราะห์ประสิทธิภาพเปรียบเทียบระหว่างการใช้งานจีพียูที่แบ็คเอ็นด์อนุญาตให้เข้าถึงได้เพียงครั้งละ 1 เวอร์ชวลแมชชีนกับการใช้งานจีพียูพร้อมกันครั้งละมากกว่า 1 เวอร์ชวลแมชชีน

4. วัดผลและวิเคราะห์โอเวอร์เฮดระหว่างการใช้งานจีพียูที่ 1 เวอร์ชวลแมชชีนกับการใช้งานจีพียูพร้อมกัน 2 เวอร์ชวลแมชชีน