ภาคผนวก

ภาคผนวก ก

คู่มือการใช้งานระบบจำลองซิมิกส์และเครื่องมือวิเคราะห์พี-สปา

ซิมิกส์ (Simics) เป็นระบบจำลองแบบเต็มระบบ ที่จัดเตรียมอุปกรณ์ฮาร์ดแวร์เสมือน เช่น หน่วยประมวลผล แคช หน่วยความจำหลัก ดิสก์ อุปกรณ์เชื่อมต่อภายนอก และระบบเครือข่าย ไว้ให้ผู้ใช้ศึกษาการทำงานของสถาปัตยกรรมของคอมพิวเตอร์เต็มทั้งระบบ ผู้ใช้สามารถปรับแต่งค่าทางสถาปัตยกรรมของอุปกรณ์เหล่านี้ได้ตามต้องการ นอกจากนี้ผู้ใช้สามารถใช้ภาษาซี ไพธอน และดีเอ็มแอล (DML ย่อมาจาก Device Modeling Language) ซึ่งเป็นภาษาโปรแกรมของซิมิกส์ พัฒนาและกำหนดการทำงานของอุปกรณ์ฮาร์ดแวร์เสมือนขึ้นมาใหม่ได้อีกด้วย บนอุปกรณ์เสมือนเหล่านี้ผู้ใช้สามารถติดตั้งและรันระบบปฏิบัติการจริง คอมไพเลอร์ เบนช์มาร์ก และแอพพลิเคชันจริง ได้เหมือนการทำงานบนระบบคอมพิวเตอร์จริง

การใช้งานซิมิกส์บนระบบปฏิบัติการลีนิกซ์เบื้องต้น

ซิมิกส์ได้เตรียมระบบคอมพิวเตอร์ส่วนหนึ่ง เพื่อให้ผู้ใช้สามารถนำไปใช้งานได้ทันที หรือให้ผู้ใช้นำไปปรับตั้งค่าของระบบและเพิ่มเติมแอพพลิเคชันเพื่อใช้ทำงานต่อไป ระบบคอมพิวเตอร์ที่ ซิมิกส์ ได้จัดเตรียมไว้มีหลายแพลตฟอร์ม เช่น Enterprise (Redhat 7.3 on x86-p4) Tango (Fedora core 5 on x86-p4) Vasa (Redhat 7.1 on Itanium) และ Bagle (Suse 7.3 on UltraSPARC II) เป็นต้น ซึ่งแต่ละแพลตฟอร์มจะมีคู่มือการใช้งานไว้ให้ใน [simics]/doc

โดยคู่มือฉบับนี้ใช้ Enterprise เพื่อแสดงตัวอย่างการทำงานเบื้องต้นของซิมิกส์โดยต้องดาวน์โหลด image file (enterprise3-rh73.craff[1]) มาเพิ่มเติม ซึ่งเป็นไฟล์ที่ได้ลง Redhat 7.3 ไว้เรียบร้อยแล้วและพร้อมใช้งานได้ทันที โดยให้นำไฟล์ดังกล่าวไปไว้ที่ [simics]/targets/x86-440bx/images เพื่อใช้ทำงานต่อไป โดยสามารถเรียกใช้ระบบคอมพิวเตอร์เป้าหมายจากส่วนควบคุมของซิมิกส์

ส่วนควบคุมของซิมิกส์มีชุดคำสั่ง Command Line Interface (CLI) เป็นอินเตอร์เฟซเพื่อให้ผู้ใช้ควบคุม ป้อนคำสั่ง ปรับตั้งค่า และสั่งงานระบบคอมพิวเตอร์เป้าหมาย สำหรับการเรียกใช้ Enterprise ผู้ใช้ต้องเข้าไปใน [workspace] และสั่งให้ซิมิกส์เริ่มทำงานจาก [workspace]

---

[1] https://www.simics.net/mwf/vh?34

พร้อมทั้งเรียกใช้ไฟล์สคริปต์ enterprise-common. simics เพื่อสั่งให้ Enterprise เริ่มทำงาน ทั้งนี้วิธีการใช้งานซิมิกส์สามารถศึกษาได้จาก simics-user-guide-unix.pdf[2] และคู่มือจาก [simics]/doc

user@host $ **cd [workspace]**

user@host workspace $ **./simics targets/x86-440bx/enterprise-common.simics**

จากนั้นซิมิกส์จะแสดงหน้าต่างของโปรแกรมออกมาสองหน้าต่าง คือส่วนควบคุมของซิมิกส์ และ Enterprise window (ถ้าเป็นระบบคอมพิวเตอร์เป้าหมายระบบอื่น ก็จะเป็นอินเตอร์เฟซของระบบนั้น) หลังจากซิมิกส์แสดงทั้งสองหน้าต่างขึ้นมา โปรแกรมจะค้างไว้ที่ส่วนควบคุมของซิมิกส์ จากนั้นให้ป้อนคำสั่ง continue หรือ c เพื่อให้ ซิมิกส์ รัน Enterprise ต่อ

simics > **continue**

Enterprise ก็จะบูตระบบขึ้นมาจนกระทั่งมาหยุดแสดงที่ส่วนรอรับคำสั่ง เพื่อให้ผู้ใช้ป้อนคำสั่งการทำงานต่อไป ผู้ใช้สามารถสลับการใช้งานระหว่าง ส่วนควบคุมของซิมิกส์ และ Enterprise window ได้ด้วยการกดปุ่ม control+c ที่ส่วนควบคุมของซิมิกส์และคำสั่ง continue เพื่อกลับไปใช้งานใน Enterprise อีกครั้ง

[press control-c]

[cpu0] v:0xc0003d0c p:0x000003d0c beq- cr4,0xc0003d1c

simics > **continue**

[cpu0] v:0xc0003d0c p:0x000003d0c beq- cr4,0xc0003d1c เป็นสถานะของหน่วยประมวลผลของ ระบบคอมพิวเตอร์เป้าหมาย ประมวลผลคำสั่งสุดท้ายก่อนจะสลับมาทำงานที่ส่วนควบคุมของซิมิกส์ โดยที่ v:0xc0003d0c เป็น Virtual memory address และ p:0x000003d0c เป็น Physical memory address ของคำสั่งนั้น

ใน Enterprise window หรือระบบคอมพิวเตอร์เป้าหมายอื่นผู้ใช้สามารถใช้คำสั่งในระบบปฏิบัติการนั้นได้เหมือนการใช้งานจริง เช่น การเรียกดูไฟล์ใน Enterprise ด้วยคำสั่ง ls ซึ่ง Enterprise จะแสดงรายการของโฟลเดอร์และไฟล์ที่มีอยู่ในระบบ

---

[2] https://www.simics.net/mwf/vh?simics-user-guide-unix.pdf

root@target # **ls /**

การสร้างจุดตรวจสอบ

ซิมิกส์มีคำสั่งการเก็บสถานะการทำงานของระบบคอมพิวเตอร์เป้าหมายไว้ เพื่อให้ผู้ใช้กลับมาใช้งานต่อได้ทันที โดยไม่ต้องบูตระบบคอมพิวเตอร์เป้าหมายเริ่มต้นใหม่ทั้งหมด ในตัวอย่างแสดงการบันทึกไฟล์สถานะการทำงานไว้ชื่อ enterprise-common-after-boot.conf และเก็บไฟล์ดังกล่าวไว้ที่ [workspace]/targets/x86-440bx

[press control-c]

simics > **ptime**

processor	steps	cycles	time [s]
cpu0	16667617210	16667617210	166.676

simics > **write-configuration targets/x86-440bx/enterprise-common-after-boot.conf**

คำสั่ง ptime เป็นคำสั่งใช้เรียกดูเวลาการทำงานและจำนวนคำสั่งที่หน่วยประมวลผลของระบบคอมพิวเตอร์เป้าหมายที่ผ่านมาทั้งหมดตั้งแต่บูตระบบขึ้นมา ใช้เพื่อตรวจสอบเวลาเมื่อย้อนกลับทำงานระบบคอมพิวเตอร์เป้าหมายอีกครั้ง จากนั้นให้ออกจาก Simics ด้วยคำสั่ง quit หรือ q และเรียกใช้ไฟล์สถานะการทำงานเพื่อเรียกใช้ระบบคอมพิวเตอร์เป้าหมายต่อ

simics > **quit**

user@host workspace $

user@host workspace $ **./simics -c targets/x86-440bx/enterprise-common-after-boot.conf**

simics > **ptime**

processor	steps	cycles	time [s]
cpu0	16667617210	16667617210	166.676

simics > **continue**

root@target #

Enterprise จะเริ่มทำงานต่อทันทีในจุดที่เก็บสถานะการทำงานไว้ การเก็บสถานะการทำงานของระบบคอมพิวเตอร์เป้าหมายนี้ ช่วยอำนวยความสะดวกแก่ผู้ใช้สามารถย้อนกลับมาทำงานในจุดที่ต้องการได้ทันที

การหยุดโปรแกรมระหว่างประมวลผล

การหยุดการทำงานของระบบคอมพิวเตอร์เป้าหมายเพื่อกลับมาทำงานบน Simics console นอกจากใช้วิธีการกดปุ่ม control+c แล้ว Simics ยังมี magic instruction เพื่อใช้หยุดการทำงานของระบบคอมพิวเตอร์เป้าหมายแบบอัตโนมัติระหว่างการประมวลผล magic instruction เป็นชุดคำสั่งที่ผู้ใช้สามารถแทรกเข้าไปในโค้ดของโปรแกรมในจุดที่ต้องการศึกษาหรือวิเคราะห์ประสิทธิภาพ โดยคำสั่งที่แทรกเข้าไปจะไม่มีผลต่อการทำงานของโปรแกรม และการใช้ชุดคำสั่ง magic instruction ในแต่ละแพลตฟอร์มมีวิธีการใช้ไม่เหมือนกัน ขึ้นอยู่กับระบบคอมพิวเตอร์เป้าหมาย

ในตัวอย่างแสดงการแทรกคำสั่ง MAGIC_BREAKPOINT เข้าไปในโค้ดของโปรแกรม arrayloops.c เพื่อจับเวลาการทำงานเฉพาะช่วงการวนลูปเพราะคำนวณผลรวมค่าที่อยู่ในอาเรย์ สำหรับไฟล์ magic-instruction.h ที่ถูกเรียกใช้เพิ่มเข้าไปในโปรแกรม และได้ถูกเก็บไว้ที่ [simics]/src/include/simics ซึ่งไฟล์ดังกล่าวเก็บชุดคำสั่ง magic instruction ไว้ เพื่อให้ผู้ใช้เรียกใช้คำสั่งดังตัวอย่างโค้ดโปรแกรม arrayloops.c ต่อไปนี้

```
#include “magic-instruction.h”
int main()
{
    //another part of code
    MAGIC_BREAKPOINT;
    //a loop sums values in an array here.

    MAGIC_BREAKPOINT;
    //display sum values
    return 0;
}
```

จากนั้นให้คัดลอกไฟล์ arrayloops.c และ magic-instruction.h ไปที่ระบบคอมพิวเตอร์เป้าหมายเพื่อประมวลผลต่อไป โดยสามารถทำได้ดังขั้นตอนต่อไปนี้

```
root@target # cp /host/home/tu/arrayloops.c .
root@target # cp /host/[simic]/ src/include/simics/ magic-instruction.h .
root@target # gcc -o arrayloops arrayloops.c
root@target # ./arrayloops
simics >
```

เมื่อโปรแกรมถูกประมวลผลมาถึงคำสั่ง MAGIC_BREAKPOINT คำสั่งแรก Simics จะหยุดการทำงานของระบบคอมพิวเตอร์เป้าหมายและเปลี่ยนมาทำงานบน Simics console ซึ่ง ณ จุดนี้ ผู้ใช้สามารถปรับเปลี่ยนค่าของระบบหรือเรียกดูข้อมูลของระบบคอมพิวเตอร์เป้าหมายเพื่อไว้เปรียบเทียบเมื่อโปรแกรมประมวลผลถึงคำสั่ง MAGIC_BREAKPOINT ถัดไป

```
simics > ptime
processor                steps                cycles          time [s]
cpu0                190470390957         190470390957          9523.520
simics > c
```

หลังจากป้อนคำสั่ง c โปรแกรม arrayloops ในระบบคอมพิวเตอร์เป้าหมายจะถูกประมวลผลต่อ จนกระทั่งประมวลผลคำสั่ง MAGIC_BREAKPOINT คำสั่งที่สอง Simics ก็จะมาทำงานที่ Simics console อีกครั้ง

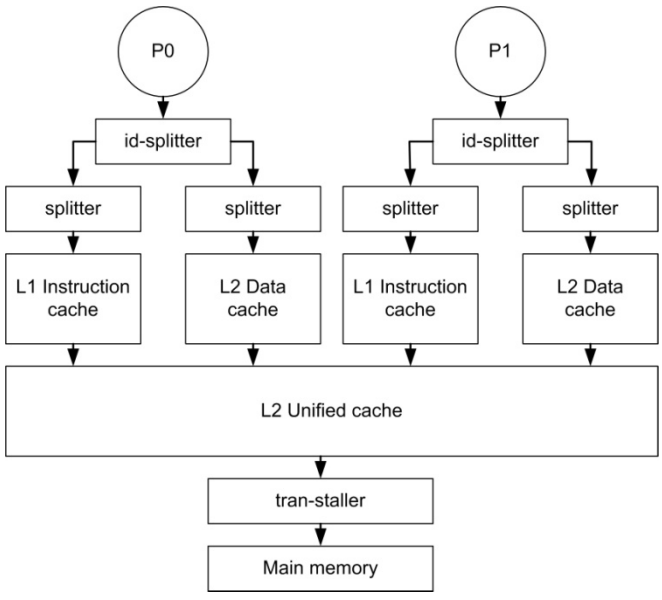
```
simics > ptime
processor                steps                cycles          time [s]
cpu0                190536364944         190536364944          9526.818
simics > c

[result of arrayloops program]
root@target #
```

เมื่อเปรียบเทียบเวลาการทำงานของหน่วยประมวลผลจากคำสั่ง ptime ทั้งสองครั้ง พบว่าการคำนวณผลรวมในอาเรย์ของโปรแกรม arrayloops ใช้เวลาการประมวลผล 3.298 วินาที และใช้จำนวนคำสั่ง 65,973,987 คำสั่ง ซึ่งเป็นเวลาและจำนวนคำสั่งบน ระบบคอมพิวเตอร์เป้าหมาย โดย steps หมายถึงจำนวนคำสั่งที่ประมวลผล และ cycles หมายถึงจำนวนรอบการทำงานของหน่วยประมวลผล ซึ่งค่าเริ่มต้นของ Simics หนึ่งคำสั่งใช้รอบการประมวลผลหนึ่งรอบ (1 step/ 1cycle)

การสร้างแคชจากโมดูลพี-แคช

โมดูลพี-แคชของพีสปาถูกพัฒนาเพิ่มเติมขึ้นบนโมดูลจี-แคช ซึ่งผู้ใช้สร้างแคชได้จากโมดูลพี-แคชโดยการเขียนไฟล์บทคำสั่งให้ซิมิกส์อ่านเมื่อสั่งให้ระบบคอมพิวเตอร์เป้าหมายเริ่มต้นทำงาน ในคู่มือนี้จะยกตัวอย่างไฟล์บทคำสั่งที่ใช้สร้างแคชบนชิปมัลติโพรเซสเซอร์ที่มี 2 หน่วยประมวลผล ดังที่แสดงในภาพภาคผนวกที่ 1



ภาพภาคผนวกที่ 1 โครงสร้างของชิปมัลติโพรเซสเซอร์ที่มี 2 หน่วยประมวลผลและใช้แคชระดับที่ 2 ร่วมกัน

จากภาพภาคผนวกที่ 1 ฮาร์ดแวร์ที่เกิดขึ้นในระบบจะมีทั้งหมดห้าส่วน คือ (1) id-splitter ทำหน้าที่แยกชนิดของการเข้าถึงพื้นที่หน่วยความจำออกเป็นการเข้าถึงชุดคำสั่งและการเข้าถึงข้อมูล (2) splitter ทำหน้าที่ส่งการเข้าถึงพื้นที่หน่วยความจำไปยังแคชระดับที่ 1 (3) แคชระดับที่ 1 ซึ่งในตัวอย่างจะแยกออกเป็นแคชชุดคำสั่งและแคชข้อมูล (4) แคชระดับที่ 2 ซึ่งหน่วย

ประมวลผลทั้งสองหน่วยใช้ร่วมกัน และ (5) trans-staller ทำหน้าที่ควบคุมเวลาการเข้าถึงพื้นที่หน่วยความจำในกรณีที่ข้อมูลไม่ได้อยู่ในแคช

จากโครงสร้างชิปมัลติโพรเซสเซอร์ในภาพภาคผนวกที่ 1 สามารถเขียนเป็นไฟล์บทคำสั่งดังนี้

```
#       Instruction L1 cache      : 2 x 32 KBytes, 8-way set associative, 64-byte line size
#       Data L1 cache             : 2 x 32 KBytes, 8-way set associative, 64-byte line size
#       with write-back, allocate ,latency 3 cycles
#       Shared Unified L2 cache   : 4098 KBytes, 16-way set associative, 64-byte line size
#         with write-back, allocate ,latency 10 cycles, penalty 165 cycles
#       Transaction staller for memory
@staller = pre_conf_object("staller", "trans-staller")
@staller.stall_time = 165

# Shared Unified L2 cache: 4 MB Write-back, allocate
@l2c = pre_conf_object("l2c", "p-cache")
@l2c.cpus = [conf.cpu0, conf.cpu1]
@l2c.config_cache_level = 22
@l2c.config_cache_type = 0
@l2c.config_reuse_dist = 1024
@l2c.config_line_number = 65536
@l2c.config_line_size = 64
@l2c.config_assoc = 16
@l2c.config_virtual_index = 1
@l2c.config_virtual_tag = 1
@l2c.config_write_back = 1
@l2c.config_write_allocate = 1
@l2c.config_replacement_policy = "lru"
@l2c.penalty_read = 10
@l2c.penalty_write = 10
@l2c.penalty_read_next = 0
@l2c.penalty_write_next = 0
@l2c.timing_model = staller
# CPU0
```

```
# instruction cache: 32 KB
@ic_cpu0 = pre_conf_object("ic_cpu0", "p-cache")
@ic_cpu0.cpus = conf.cpu0
@ic_cpu0.config_cache_level = 10
@ic_cpu0.config_cache_type = 2
@ic_cpu0.config_reuse_dist = 1024
@ic_cpu0.config_line_number = 512
@ic_cpu0.config_line_size = 64
@ic_cpu0.config_assoc = 8
@ic_cpu0.config_write_back = 1
@ic_cpu0.config_write_allocate = 1
@ic_cpu0.config_virtual_index = 1
@ic_cpu0.config_virtual_tag = 1
@ic_cpu0.config_replacement_policy = "lru"
@ic_cpu0.penalty_read = 3
@ic_cpu0.penalty_write = 0
@ic_cpu0.penalty_read_next = 0
@ic_cpu0.penalty_write_next = 0
@ic_cpu0.timing_model = l2c

# data cache: 32 KB Write-through, non-allocate
@dc_cpu0 = pre_conf_object("dc_cpu0", "p-cache")
@dc_cpu0.cpus = conf.cpu0
@dc_cpu0.config_cache_level = 10
@dc_cpu0.config_cache_type = 1
@dc_cpu0.config_reuse_dist = 1024
@dc_cpu0.config_line_number = 512
@dc_cpu0.config_line_size = 64
@dc_cpu0.config_assoc = 8
@dc_cpu0.config_write_back = 1
@dc_cpu0.config_write_allocate = 1
@dc_cpu0.config_virtual_index = 1
@dc_cpu0.config_virtual_tag = 1
@dc_cpu0.config_replacement_policy = "lru"
```

```
@dc_cpu0.penalty_read = 3
@dc_cpu0.penalty_write = 3
@dc_cpu0.penalty_read_next = 0
@dc_cpu0.penalty_write_next = 0
@dc_cpu0.timing_model = l2c

# transaction splitter for instruction cache
@ts_i_cpu0 = pre_conf_object("ts_i_cpu0", "trans-splitter")
@ts_i_cpu0.cache = ic_cpu0
@ts_i_cpu0.timing_model = ic_cpu0
@ts_i_cpu0.next_cache_line_size = 64
# transaction splitter for data cache
@ts_d_cpu0 = pre_conf_object("ts_d_cpu0", "trans-splitter")
@ts_d_cpu0.cache = dc_cpu0
@ts_d_cpu0.timing_model = dc_cpu0
@ts_d_cpu0.next_cache_line_size = 64
# instruction-data splitter
@ids_cpu0 = pre_conf_object("ids_cpu0", "id-splitter")
@ids_cpu0.ibranch = ts_i_cpu0
@ids_cpu0.dbranch = ts_d_cpu0

# CPU1
# instruction cache: 32 KB
@ic_cpu1 = pre_conf_object("ic_cpu1", "p-cache")
@ic_cpu1.cpus = conf.cpu1
@ic_cpu1.config_cache_level = 11
@ic_cpu1.config_cache_type = 2
@ic_cpu1.config_reuse_dist = 1024
@ic_cpu1.config_line_number = 512
@ic_cpu1.config_line_size = 64
@ic_cpu1.config_assoc = 8
@ic_cpu1.config_write_back = 1
@ic_cpu1.config_write_allocate = 1
@ic_cpu1.config_virtual_index = 1
```

```
@ic_cpu1.config_virtual_tag = 1
@ic_cpu1.config_replacement_policy = "lru"
@ic_cpu1.penalty_read = 3
@ic_cpu1.penalty_write = 0
@ic_cpu1.penalty_read_next = 0
@ic_cpu1.penalty_write_next = 0
@ic_cpu1.timing_model = l2c

# data cache: 32 KB Write-through, non-allocate
@dc_cpu1 = pre_conf_object("dc_cpu1", "p-cache")
@dc_cpu1.cpus = conf.cpu1
@dc_cpu1.config_cache_level = 11
@dc_cpu1.config_cache_type = 1
@dc_cpu1.config_reuse_dist = 1024
@dc_cpu1.config_line_number = 512
@dc_cpu1.config_line_size = 64
@dc_cpu1.config_assoc = 8
@dc_cpu1.config_write_back = 1
@dc_cpu1.config_write_allocate = 1
@dc_cpu1.config_virtual_index = 1
@dc_cpu1.config_virtual_tag = 1
@dc_cpu1.config_replacement_policy = "lru"
@dc_cpu1.penalty_read = 3
@dc_cpu1.penalty_write = 3
@dc_cpu1.penalty_read_next = 0
@dc_cpu1.penalty_write_next = 0
@dc_cpu1.timing_model = l2c

# transaction splitter for instruction cache
@ts_i_cpu1 = pre_conf_object("ts_i_cpu1", "trans-splitter")
@ts_i_cpu1.cache = ic_cpu1
@ts_i_cpu1.timing_model = ic_cpu1
@ts_i_cpu1.next_cache_line_size = 64
# transaction splitter for data cache
```

```
@ts_d_cpu1 = pre_conf_object("ts_d_cpu1", "trans-splitter")
@ts_d_cpu1.cache = dc_cpu1
@ts_d_cpu1.timing_model = dc_cpu1
@ts_d_cpu1.next_cache_line_size = 64
# instruction-data splitter
@ids_cpu1 = pre_conf_object("ids_cpu1", "id-splitter")
@ids_cpu1.ibranch = ts_i_cpu1
@ids_cpu1.dbranch = ts_d_cpu1

# Set cache hierarchy
@l2c.higher_level_caches = [ic_cpu0, dc_cpu0, ic_cpu1, dc_cpu1]
# Add cache to system
@SIM_add_configuration([staller, l2c, ic_cpu0, dc_cpu0, ts_i_cpu0, ts_d_cpu0, ids_cpu0,
ic_cpu1, dc_cpu1, ts_i_cpu1, ts_d_cpu1, ids_cpu1], None)
```

เมื่อผู้ใช้ต้องการเรียกดูสถิติการทำงานของแคชแต่ละส่วน สามารถใช้คำสั่งดังตัวอย่างการเรียกดูสถิติของแคชระดับที่ 2 ต่อไปนี้

```
simics > l2c.statistics
P-Cache statistics: l2c
------------------
```

Total number of transactions:	309977766
Device data reads (DMA):	0
Device data writes (DMA):	0
Uncacheable data reads:	214832129
Uncacheable data writes:	88357562
Uncacheable instruction fetches:	0
Data read transactions:	2249
Data read misses:	888
Compulsory:	471
Capacity:	70
Conflict:	347

Data read hit ratio:	60.52%
Instruction fetch transactions:	4969062
Instruction fetch misses:	5108
Instruction fetch hit ratio:	99.90%
Data write transactions:	1805775
Data write misses:	1812
Compulsory:	1770
Capacity:	3
Conflict:	39
Data write hit ratio:	99.90%
Copy back transactions:	0
Lost Stall Cycles:	18356730

simics > **c**

การวิเคราะห์คุณสมบัติพื้นที่เก็บข้อมูลด้วยพี-เทรซ

การเรียกใช้พี-เทรซเพื่อวิเคราะห์คุณสมบัติพื้นที่เก็บข้อมูล โดยเทรซไฟล์จะถูกบันทึกไว้ใน [workspace]/P-GRAPH/ ผู้ใช้สามารถเรียกดังตัวอย่างคำสั่งต่อไปนี้

simics > **new-P-SPA-tracer**

P-SPA-Trace object P_SPA_trace0 created. Enable tracing with P-GRAPH.

simics> **P_SPA_trace0.start**

P-SPA-tracing enabled. Writing P_SPA_trace0 output to P-GRAPH.

simics> **c**

เมื่อโปรแกรมประมวลผลเสร็จสิ้น ผู้ใช้สามารถสั่งให้พี-เทรซหยุดการทำงานด้วยคำสั่งต่อไปนี้

simics> **P_SPA_trace0.stop**

P-SPA-tracing disabled.

simics> **c**

ภาคผนวก ข

ตารางวิเคราะห์สถิติ

ตารางภาคผนวกที่ 1 ผลการวิเคราะห์ความถดถอยเวลาประมวลผลของการทดสอบความถูกต้องของพี-สปาในบทที่ 4.3

Model Summary(b)

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.957(a)	.915	.901	.191187

a Predictors: (Constant), p_spa_time
b Dependent Variable: measure_time

ANOVA(b)

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	2.372	1	2.372	64.894	.000(a)
	Residual	.219	6	.037		
	Total	2.591	7			

a Predictors: (Constant), p_spa_time
b Dependent Variable: measure_time

ตารางภาคผนวกที่ 2 ผลการวิเคราะห์ความถดถอยอัตราไม่พบข้อมูลในแคชระดับที่ 2 ของการทดสอบความถูกต้องของพี-สปาในบทที่ 4.3

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.714(a)	.510	.428	.00133829

a Predictors: (Constant), p_spa_miss

ANOVA(b)

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	.000	1	.000	6.234	.047(a)
	Residual	.000	6	.000		
	Total	.000	7			

a Predictors: (Constant), p_spa_miss
b Dependent Variable: measure_miss

ตารางภาคผนวกที่ 3 ผลการวิเคราะห์การวิเคราะห์สถิติแบบจำแนกทางเดียว อัตราไม่พบข้อมูลในแคชระดับที่ 2 ของชุดโปรแกรมเอ็นพีบี ในการทดลองบทที่ 5.2

ANOVA

cache_miss

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	19.262	1	19.262	18.408	.001
Within Groups	14.649	14	1.046		
Total	33.910	15			

ตารางภาคผนวกที่ 4 ผลการวิเคราะห์การวิเคราะห์สถิติแบบจำแนกทางเดียว อัตราไม่พบข้อมูลในแคชระดับที่ 1 ของโปรแกรมเอฟที ในการทดลองบทที่ 6.1

ANOVA

l1c_miss

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	.187	1	.187	.425	.550
Within Groups	1.763	4	.441		
Total	1.950	5			

ตารางภาคผนวกที่ 5 ผลการวิเคราะห์การวิเคราะห์สถิติแบบจำแนกทางเดียว อัตราไม่พบข้อมูลในแคชระดับที่ 2 ของโปรแกรมเอฟที ในการทดลองบทที่ 6.1

ANOVA

l2c_miss

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	2.112	1	2.112	316.840	.000
Within Groups	.027	4	.007		
Total	2.139	5			

ตารางภาคผนวกที่ 6 ผลการวิเคราะห์การวิเคราะห์สถิติแบบจำแนกทางเดียว เวลาประมวลผลของโปรแกรมเอฟที ในการทดลองบทที่ 6.1

ANOVA

Time

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	143480130698817.200	1	143480130698817.200	4.858	.092
Within Groups	118137873799642.600	4	29534468449910.670		
Total	261618004498459.900	5			