ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
www.cslab.ece.ntua.gr

ΠΡΟΗΓΜΕΝΑ ΘΕΜΑΤΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣΥΠΟΛΟΓΙΣΤΩΝ

Ακ. έτος 2009-2010, 8ο εξάμηνο, Σχολή ΗΜ&ΜΥ

3η ΕΡΓΑΣΙΑ

Τελική Ημερομηνία Παράδοσης: 6 Ιουνίου *(δεν θα δοθεί παράταση)*

Σκοπός αυτής της άσκησης είναι η εξοικείωση με το περιβάλλον προσομοίωσης του Simics και η μελέτη της επίδρασης διαφόρων παραμέτρων της ιεραρχίας της μνήμης στην απόδοση των εφαρμογών .

1. Ορισμοί και συμβάσεις

Θα χρησιμοποιούμε τον όρο ***host*** για να αναφερόμαστε στο πραγματικό μηχάνημα όπου εκτελείται ο Simics. Θα χρησιμοποιούμε τον όρο ***target*** για να αναφερόμαστε στο ιδεατό μηχάνημα που προσομοιώνει ο Simics. Επιπλέον, κάνουμε τις εξής συμβάσεις όσον αφορά το υπόλοιπο του κειμένου:

- Η είσοδος που δίνουμε στο command line του host, θα αναγράφεται ως εξής:

  ```
  host$ some command
  ```

- Η είσοδος που δίνουμε στην κονσόλα του Simics, θα αναγράφεται ως εξής:

  ```
  simics> some command
  ```

- Η είσοδος που δίνουμε στο command line του target, θα αναγράφεται ως εξής:

  ```
  target# some command
  ```

2. Εγκατάσταση του Simics και διαμόρφωση περιβάλλοντος εργασίας

2.1 Εγκατάσταση του Simics

Για να εγκαταστήσετε τον Simics πρέπει πρώτα να κατεβάσετε το κατάλληλο tar αρχείο. Ανάλογα με την αρχιτεκτονική του συστήματος σας (32bit ή 64bit) εκτελέστε:

```
host$ wget http://www.cslab.ece.ntua.gr/courses/advcomparch/files/simics-pkg-20-3.0.31-linux.tar
```

ή

```
host$ wget http://www.cslab.ece.ntua.gr/courses/advcomparch/files/simics-pkg-20-3.0.31-linux64.tar
```

Στη συνέχει θα πρέπει να εκτελέσετε τις παρακάτω εντολές:

```
host$ cp simics-pkg-20-3.0.31-linuxXX.tar /tmp
host$ tar xvf /tmp/simics-pkg-20-3.0.31-linuxXX.tar
host$ cd /tmp/simics-3.0-install
host$ sudo ./install-simics.sh
```

Ο installer θα ζητήσει ένα decryption key, το οποίο είναι το:

60d22ec5d936a5290e6f1d6e31216792

Σε αυτά που ζητούνται κατά την εγκατάσταση δίνετε τις default τιμές (πατάτε Enter). Αν όλα πάνε καλά, στο τέλος την εγκατάστασης θα δείτε ένα μήνυμα σαν και το ακόλουθο:

Simics 3.0.31 was successfully installed as /opt/virtutech/simics-3.0.31/

Τέλος, εκτελείτε:

```
host$ rm -rf /tmp/simics-3.0-install
```

2.2 Δημιουργία workspace

Για την εκτέλεση των προσομοιώσεων απαιτείται η δημιουργία ενός φακέλου workspace. Στο εξής, υποθέτουμε ότι ο κατάλογος αυτός θα είναι ο ~/simics-workspace και θα βρίσκεται στο home σας.

```
host$ /opt/virtutech/simics-3.0.31/bin/workspace-setup ~/simics-workspace
```

2.3 OS images

Για την εκτέλεση του λειτουργικού απαιτείται η χρήση κάποιων images, τα οποία απεικονίζουν δίσκους όπου έχει ήδη γίνει η εγκατάσταση του OS. Τα images αυτά είναι θεμιτό να βρίσκονται σε κάποια συγκεκριμένη τοποθεσία.

```
host$ sudo mkdir /opt/virtutech/images
host$ cd /opt/virtutech/images/
host$ sudo wget http://www.simics.net/pub/simics/import/x86/tango1-fedora5.craff
```

Ο φάκελος αυτός πρέπει να δηλωθεί στο configuration του Simics. Για αυτό προσθέτουμε στο αρχείο **/opt/virtutech/simics-3.0.31/config/startup-commands** την ακόλουθη γραμμή:

add-directory /opt/virtutech/images

2.4 License server και support

Για την εκτέλεση του Simics χρειάζεται να επικοινωνείτε με τον license server. Αυτό επιτυγχάνεται θέτοντας την παρακάτω μεταβλητή περιβάλλοντος (environment variable):

```
host$ export VTECH_LICENSE_FILE=@scirouter.cslab.ece.ntua.gr
```

Για να μη θέτετε κάθε φορά την μεταβλητή αυτή, προτείνεται να την εντάξετε στο *.bash_profile* σας. Υπενθυμίζετε, ότι ο license server αναγνωρίζει μόνο μηχανήματα με ip Πολυτεχνείου (147.102.xxx.xxx) και για αυτό το λόγο απαιτείται η επικοινωνία μέσω κάποιου vpn δικτύου.

Για support, εκτός από τη mailing list του μαθήματος, μπορείτε να απευθύνεστε και στο forum της Virtutech, το οποίο βρίσκεται στη διεύθυνση https://www.simics.net/mwf/forum_show.pl . Για να χρησιμοποιήσετε το forum αυτό, απαιτείται η εγγραφή σας ως φοιτητές δηλώνοντας **ΥΠΟΧΡΕΩΤΙΚΑ** email Πολυτεχνείου (δηλαδή κάποιο email που τελειώνει σε @xxxx.ntua.gr).

2.5 Εκκίνηση λειτουργικού στον Simics

Την πρώτη φορά που θα εκτελέσετε τον Simics θα πρέπει να επιβεβαιώσετε ότι δέχεστε το license. Απλά πατάτε Enter και yes όπου χρειάζεται. Στη συνέχεια για να εκκινήσετε το λειτουργικό που θα χρησιμοποιηθεί στην άσκηση αυτή, εκτελείτε τις παρακάτω εντολές:

```
host$ cd ~/simics-workspace
host$ ./simics
simics> $text_console="yes"
simics> run-command-file targets/x86-440bx/tango-common.simics
```

Οι εντολές αυτές θα δημιουργήσουν ένα μηχάνημα *tango*, το οποίο θα χρησιμοποιεί το disk image που κατεβάσατε προηγουμένως (Fedora 5) και θα έχει ένα text console. Σε αυτό το σημείο λοιπόν, ο Simics θα αρχίσει την προσομοίωση, εμφανίζοντας ένα νέο παράθυρο, το οποίο θα είναι η κονσόλα του target μηχανήματος. Η προσομοίωση ξεκινάει με την κονσόλα του target μηχανήματος "παγωμένη". Για να συνεχιστεί η προσομοίωση, και να αρχίσει το target μηχάνημα να φορτώνει το λειτουργικό, τυπώνουμε:

```
simics> continue
```

ή απλά

```
simics> c
```

Μετά από αρκετή ώρα, και αφού περάσουν τα συνήθη στάδια αρχικοποίησης, εμφανίζεται το prompt του Linux. Δίνετε username *root* και password *simics* και είστε πλέον σε θέση να χρησιμοποιήσετε το target ως ένα κανονικό σύστημα Linux, με τα συνηθισμένα command-line utilities.

Δίνοντας **`[Ctrl-C]`** στο τερματικό όπου εκτελείται ο Simics, σταματά προσωρινά η προσομοίωση και επιστρέφετε στο simics prompt. Όσο η προσομοίωση είναι σταματημένη, δεν είστε σε θέση να αλληλεπιδράσετε με το target μηχάνημα. Για να επανεκκινήσετε την προσομοίωση, δίνετε την εντολή continue στο simics prompt, όπως αναφέραμε προηγουμένως.

2.6 Μετάβαση σε single-user mode

Στα πλαίσια της αξιολόγησης της απόδοσης εφαρμογών, είναι καλή πρακτική να οδηγούμε το σύστημα σε single-user mode, ώστε περιορίσουμε στο ελάχιστο τις διεργασίες που τρέχουν στο background και να δημιουργήσουμε ένα όσο το δυνατόν πιο απομονωμένο περιβάλλον εκτέλεσης για τα πειράματα. Για αυτό, δίνουμε στην κονσόλα του target την εντολή:

```
target# telinit 1
```

2.7 Mount του συστήματος αρχείων του host

Το επόμενο βήμα είναι η μεταφορά αρχείων από το host στο target μηχάνημα. Η πρόσβαση στο σύστημα αρχείων του host μηχανήματος γίνεται μέσω του utility simicsfs, το οποίο είναι ήδη εγκατεστημένο στο disk image που χρησιμοποιείτε. Για να κάνετε λοιπόν mount το σύστημα αρχείων του host, εκτελείτε:

```
simics> hfs0.root /home/username
simics> c
target# mount /host
```

Η πρώτη εντολή λέει στον simics σε ποιο σημείο του host file-system θα κάνει mount. Σε περίπτωση που η εντολή αυτή δεν εκτελεστεί, τότε το mount θα γίνει στο / του host, εγείροντας έτσι ζητήματα ασφαλείας. Τελικά, το host file-system, μπορεί να προσπελαστεί από τον κατάλογο /host του target μηχανήματος.

Στο σημείο αυτό μπορείτε να αντιγράψετε χρησιμοποιώντας την εντολή cp αρχεία από το /host directory tree στον κατάλογο /root ή /home ή οπουδήποτε αλλού μέσα στο προσομοιωμένο μηχάνημα. Αφού

πραγματοποιήσετε τις απαιτούμενες μεταφορές αρχείων, καλό είναι να κάνετε umount το host file-system.

```
target# umount /host
```

2.8 Δημιουργία και χρήση checkpoints

Στο σημείο αυτό έχετε φορτώσει το OS και έχετε μεταφέρει τα απαιτούμενα αρχεία στο δίσκο του target μηχανήματος. Η διαδικασία αυτή (εκκίνηση OS + μεταφορά αρχείων) είναι κοινή για τις περισσότερες προσομοιώσεις και μπορεί να αποφεύγεται στο μέλλον με τη χρήση chekcpoints.

Για να δημιουργήσετε ένα checkpoint πρέπει πρώτα από όλα να σιγουρευτείτε πως έχετε κάνει umount το host file-system. Στην κονσόλα του simics δίνετε την εντολή:

```
simics> write-configuration /path/to/checkpoints/name.check
```

Η εντολή αυτή θα δημιουργήσει ένα checkpoint με όνομα name.check στον κατάλογο που δείχνει το /path/to/checkpoints. Αυτός ο κατάλογος μπορεί να βρίσκεται είτε μέσα στο simics-workspace σας είτε στο /opt/virtutech/images. Αυτή η διαδικασία λέγεται *checkpointing*, και το σύνολο των δεδομένων που προσδιορίζουν την κατάσταση του προσομοιωμένου μηχανήματος εκείνη τη στιγμή (π.χ. Περιεχόμενα RAM, περιεχόμενα δίσκου, κ.λπ.) λέγεται checkpoint.

Για να χρησιμοποιήσετε το checkpoint στο μέλλον, δίνετε:

```
host$ ./simics -c /path/to/checkpoints/name.check
```

ή εναλλακτικά

```
host$ ./simics
simics> read-configuration /path/to/checkpoints/name.check
```

Σημαντική λεπτομέρεια: Κάθε φορά που αποθηκεύετε ένα checkpoint, ο Simics αποθηκεύει τις διαφορές ανάμεσα στην τρέχουσα κατάσταση του target και του προηγούμενου checkpoint. Αυτό βοηθάει πολύ στην εξοικονόμηση αποθηκευτικού χώρου. Το πρόβλημα είναι ότι αν αποθηκεύσουμε ένα checkpoint, εκτελέσουμε κάποια πράγματα, στη συνέχεια αποθηκεύσουμε ένα δεύτερο checkpoint, και μετά διαγράψουμε το πρώτο, τότε το δεύτερο checkpoint δε θα δουλεύει πια.

2.9 Χρήσιμες εντολές του Simics

Στο σημείο αυτό είστε σε θέση να εκτελείτε πλέον προσομοιώσεις, οι οποίες μπορούν να ελέγχονται μέσω διαφόρων εντολών. Η επανεκκίνηση και η διακοπή της προσομοίωσης γίνεται δίνοντας **`continue`** και [**`Ctrl-C`**], αντίστοιχα. Με τις εντολές **`step-instruction`** και **`step-cycle`**, μπορούμε να εκτελούμε την προσομοίωση για συγκεκριμένο αριθμό εντολών και κύκλων, αντίστοιχα (η λειτουργικότητα των εντολών αυτών είναι ίδια, όταν το CPI για την αρχιτεκτονική που προσομοιώνουμε είναι ίσο με 1), τυπώνοντας τις εντολές που εκτελούνται στην κονσόλα του simics:

```
simics> step-instruction 100_000
simics> step-cycle 100_000
```

Στα αριθμητικά ορίσματα μπορούμε να χρησιμοποιούμε το **`_`** προκειμένου να ομαδοποιούμε τα αριθμητικά ψηφία για μεγαλύτερη σαφήνεια.

Άλλες εντολές του Simics τυπώνουν πληροφορίες σχετικά με την τρέχουσα κατάσταση του προσομοιωμένου συστήματος. Για παράδειγμα, η εντολή **`pregs`** τυπώνει τα περιεχόμενα των καταχωρητών. Οι εντολές **`read-reg`** και **`write-reg`** χρησιμοποιούνται για την ανάγνωση και τροποποίηση, αντίστοιχα, κάποιου συγκεκριμένου καταχωρητή. Οι εντολές **`get`** και **`set`** κάνουν το ίδιο πράγμα αλλά για θέσεις μνήμης. Η εντολή **`ptime`** τυπώνει τον χρόνο που πέρασε από την αρχή της προσομοίωσης, ενώ η **`pstats`** τυπώνει στατιστικά για τον επεξεργαστή. Γενικά, δίνοντας **`help`**

<instruction-name> στην κονσόλα του simics μπορείτε να βρίσκετε πληροφορίες για τη σύνταξη και τη λειτουργία της κάθε εντολής.

Τέλος, βάζοντας τον χαρακτήρα **!** μπροστά από την εντολή που δίνετε στην κονσόλα του simics, έχει σαν αποτέλεσμα η εντολή αυτή να εκτελείται εξωτερικά, στο shell του host μηχανήματος και όχι στην κονσόλα του simics. Άλλες χρήσιμες εντολές και οδηγίες αναφορικά με την προσομοίωση μπορείτε να βρείτε στο Simics User Guide (στο /opt/virtutech/simics-3.0.31/doc).

3. Προσομοίωση Ιεραρχίας Μνήμης

3.1 Εισαγωγή

Ο συνηθέστερος τρόπος χρήσης του Simics είναι για λειτουργική προσομοίωση του συνόλου εντολών μιας αρχιτεκτονικής, και όχι τόσο η προσομοίωση της απόδοσής της. Ενώ παρέχει λοιπόν στο λειτουργικό σύστημα και στις εφαρμογές μία πλήρη, ιδεατή πλατφόρμα εκτέλεσης, στην πραγματικότητα πολλές από τις λειτουργίες της πλατφόρμας αυτής είναι εξιδανικευμένες. Για παράδειγμα, μια εντολή που προσπελάζει τη μνήμη εμφανίζεται σαν να παίρνει έναν κύκλο μηχανής για να εκτελεστεί, που για ένα πραγματικό σύστημα είναι προφανώς μη ρεαλιστικό.

Παρόλα αυτά, είναι δυνατόν να προσαρτήσουμε κατάλληλο module στον Simics προκειμένου να εκτιμήσουμε τη συμπεριφορά ενός προγράμματος όσον αφορά την cache ή να μελετήσουμε την αποδοτικότητα μιας συγκεκριμένης ιεραρχίας μνήμης. Το module αυτό είναι το g-cache και περιλαμβάνεται στον Simics. Caches που προσομοιώνονται μέσω του g-cache module μπορούν να παραμετροποιηθούν και να διασυνδεθούν μεταξύ τους, σχηματίζοντας μια πλήρη ιεραρχία μνήμης. Η προδιαγραφή των παραμέτρων και των διασυνδέσεων γίνεται με χρήση scripts του Simics, όπως θα δούμε σε επόμενη ενότητα. Τα simics scripts δεν είναι τίποτα παραπάνω από αρχεία με ακολουθίες εντολών του simics, ενώ μερικές φορές περιέχουν εντολές της Python (έχουν πρόθεμα το @), ή εντολές του Unix shell (με πρόθεμα το !).

Εκτελώντας τον simics με το command line όρισμα -stall, μπορούμε να προσομοιώσουμε τις όποιες καθυστερήσεις στην εκτέλεση των εντολών, ανάλογα με το αν τα δεδομένα που χρειάζονται οι εντολές βρίσκονται ή όχι σε κάποιο επίπεδο της ιεραρχίας μνήμης. Την ίδια στιγμή, τα cache modules καταγράφουν στατιστικά όσον αφορά τις αιτήσεις μνήμης που κλήθηκαν να εξυπηρετήσουν οι αντίστοιχες caches. Με το -stall, πραγματοποιούμε σαφώς πιο λεπτομερή προσομοίωση της συμπεριφοράς του συστήματος, γι' αυτό και η ταχύτητα της προσομοιούμενης πλατφόρμας μειώνεται αισθητά.

Μία επιπλέον λεπτομέρεια είναι ότι όταν οι caches προσαρτώνται για πρώτη φορά στον simics, είναι αρχικά άδειες. Οποιαδήποτε στατιστικά επομένως καταγράφονται αρχικά από αυτές, θα αφορούν πιο πολύ τα compulsory misses τα οποία συμβαίνουν καθώς αυτές γεμίζουν. Για αυτό το λόγο, είναι απαραίτητο να “θερμάνουμε” (warm-up) τις caches εκτελώντας το benchmark που μας ενδιαφέρει για αρκετές εντολές, πριν αρχίσουμε να συλλέγουμε στατιστικά τα οποία θα χρησιμοποιήσουμε για να εκτιμήσουμε την πραγματική συμπεριφορά του benchmark.

Γι' αυτό, η γενική μεθοδολογία προσομοίωσης που ακολουθούμε είναι η εξής:

0. Φορτώνουμε το λειτουργικό, δημιουργούμε checkpoints κ.λπ., όπως περιγράψαμε στην ενότητα 2.

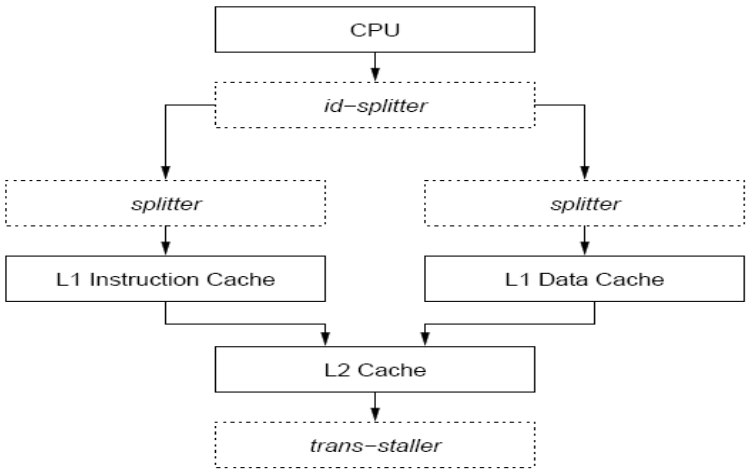
1. Ξεκινάμε τον simics χρησιμοποιώντας τα checkpoints, και επιπλέον χρησιμοποιώντας το όρισμα **-stall**

2. Φορτώνουμε τις caches χρησιμοποιώντας το κατάλληλο simics script
3. Εκτελούμε το benchmark για το οποίο ενδιαφερόμαστε, "θερμαίνοντας" τις caches
4. Όταν η εκτέλεση του benchmark φτάσει σε "ώριμο" σημείο (π.χ. όταν ολοκληρώσει τα στάδια αρχικοποίησης καθώς και τις αρχικές επαναλήψεις), διακόπτουμε την προσομοίωση
5. Μηδενίζουμε τα στατιστικά για όλες τις caches
6. Συνεχίζουμε την προσομοίωση για να συλλέξουμε τα *πραγματικά* στατιστικά για τις caches

Τα δεδομένα από τα cache modules μπορούν να ανακτηθούν με διάφορους τρόπους. Δίνοντας **`<cache>.info`** στην κονσόλα του simics παίρνουμε της παραμέτρους μιας συγκεκριμένης cache. Με **`<cache>.status`** παίρνουμε τα τρέχοντα περιεχόμενα για όλες τις γραμμές της cache. Με **`<cache>.statistics`** παίρνουμε διάφορα στατιστικά για την απόδοση της cache. Με **`<cache>.add-profiler`** προσθέτουμε ένα profiler module στην cache το οποίο κατηγοριοποιεί τα misses ανά εντολή ή ανά διεύθυνση.

3.2 Ορισμός της ιεραρχίας μνήμης

Η ιεραρχία κρυφής μνήμης που θα χρησιμοποιήσουμε στα πλαίσια αυτής της άσκησης βασίζεται στο σύστημα κρυφής μνήμης του σχήματος της σελίδας 200 στο Simics User Guide, το οποίο φαίνεται παρακάτω. Αποτελείται από κρυφή μνήμη δεδομένων 1ου επιπέδου (dc), κρυφή μνήμη εντολών 1ου επιπέδου (ic) και ενοποιημένη μνήμη εντολών και δεδομένων 2ου επιπέδου (l2c). Οι παράμετροι των επιμέρους caches, καθώς και ο τρόπος που διασυνδέονται τα επιμέρους components, μπορούν να αποθηκευτούν σε ένα configuration αρχείο (simics script), το οποίο θεωρούμε ότι είναι το cache-hierarchy.simics, τα περιεχόμενα του οποίου δίνονται στο Παράρτημα Α. Για περισσότερες λεπτομέρειες σχετικά με τη λειτουργία των επιμέρους components μπορείτε να ανατρέξετε στην ενότητα 18.4 (και γενικότερα στο κεφάλαιο 18) του Simics User Guide. Για αναλυτική περιγραφή των παραμέτρων ενός αντικειμένου τύπου g-cache, μπορείτε να ανατρέξετε στη σελίδα 1103 του Simics Reference Manual.



3.3 Μοντέλο απόδοσης

Όπως μπορείτε να παρατηρήσετε στις παραμέτρους της ιεραρχίας μνήμης στο Παράρτημα Α, τα penalties των προσβάσεων στη μνήμη έχουν οριστεί ίσα με 0. Η επιλογή αυτή έγινε προκειμένου να μειωθεί ο απαιτούμενος χρόνος προσομοίωσης. Καθώς οι caches έχουν μηδενικούς χρόνους πρόσβασης, ο αριθμός των κύκλων που δίνει ο simics για την εκτέλεση της περιοχής μας ενδιαφέρει δεν είναι σωστός. Για αυτό τον λόγο απαιτείται ένα μοντέλο απόδοσης το οποίο θα προσεγγίζει με μεγαλύτερη ακρίβεια τον πραγματικό αριθμό απαιτούμενων κύκλων.

Το μοντέλο του simics για τις x86 αρχιτεκτονικές είναι ένας in-order επεξεργαστής με IPC=1. Εμείς θεωρούμε ότι οι εντολές πρόσβασης στις caches πρώτου επιπέδου δεν προκαλούν καθυστέρηση εφόσον είναι hits. Κάθε miss στις L1 στοιχίζει 10 κύκλους και κάθε miss στη L2 200 κύκλους αντίστοιχα.

Επομένως, ο συνολικός αριθμός των κύκλων μπορεί να προσεγγιστεί ως εξής :

Cycles = Instructions + (L1D_misses+L1I_misses) * L1_penalty + L2_misses * L2_penalty

3.4 Διαδικασία συλλογής στατιστικών

Τα benchmarks τα οποία θα χρησιμοποιήσετε στα πλαίσια της άσκησης προέρχονται από την σουίτα SPEC CPU2000. Τα εκτελέσιμα καθώς και τα αρχεία εισόδου μπορείτε να τα κατεβάσετε με τον παρακάτω τρόπο:

```
host$ wget

       http://www.cslab.ece.ntua.gr/courses/advcomparch/files/bench_ask3.tgz
```

Για κάθε ένα από τα benchmarks:

1. Ξεκινήστε τον simics σε stall mode και φορτώστε το κατάλληλο checkpoint:

```
host$ ./simics -stall -c <checkpoint-name>
```

2. Εκτελέστε τις επόμενες εντολές για να εξασφαλίσετε τη σωστή λειτουργία των benchmarks και των caches:

```
simics> instruction-fetch-mode instruction-fetch-trace
simics> istc-disable
simics> dstc-disable
simics> magic-break-enable
```

Με την τελευταία εντολή, δίνουμε τη δυνατότητα στον simics να αντιλαμβάνεται τα "magic breakpoints" κατά τη διάρκεια της προσομοίωσης. Τα magic breakpoints είναι ο τρόπος επικοινωνίας μιας εφαρμογής με τον προσομοιωτή. Στην ουσία είναι ειδικές εντολές τις οποίες μπορούμε να εισάγουμε στον κώδικα των προγραμμάτων, ώστε κάθε φορά που ο simics εκτελεί μια τέτοια εντολή να διακόπτει την προσομοίωση, να επιστρέφει στην κονσόλα του, και έτσι να μπορούμε εμείς να κάνουμε τις ενέργειες που επιθυμούμε προτού ξεκινήσουμε και πάλι την προσομοίωση.

Κάθε ένα από τα benchmarks που σας δίνουμε εμπεριέχει 2 magic breakpoints, τα οποία στην ουσία καθορίζουν την περιοχή ενδιαφέροντος του προγράμματος που θέλουμε να μελετήσουμε. Το πρώτο breakpoint βρίσκεται συνήθως μετά από τη φάση αρχικοποίησης του benchmark (π.χ. διάβασμα αρχείων, δέσμευση μνήμης, κ.λπ.), έτσι ώστε να μπορούμε να αρχίζουμε την λεπτομερειακή (και πιο αργή) προσομοίωση αφού γίνουν όλες οι αρχικοποιήσεις για τις οποίες δεν μας ενδιαφέρει η συλλογή

στατιστικών. Το δεύτερο breakpoint βρίσκεται συνήθως στο τέλος του benchmark, έτσι ώστε όταν η προσομοίωση γίνεται μέχρι το τέλος της περιοχής ενδιαφέροντος (και όχι απλά για κάποια εκατομμύρια αρχικών εντολών) ο έλεγχος να επιστρέφεται στον simics. Σημειώνουμε ότι σε κανένα από τα πειράματα που ζητώνται στην άσκηση η προσομοίωση δε φτάνει μέχρι το δεύτερο breakpoint.

3. Εκτελέστε στην κονσόλα του target κάποιο από τα benchmarks. Ο τρόπος που εκτελείται το καθένα είναι διαφορετικός:

```
target# ./art -scanfile c756hel.in -trainfile1 a10.img -trainfile2 hc.img -stride 2
-startx 110 -starty 200 -endx 160 -endy 240 -objects 10

target# ./crafty < inp.in.crafty.train

target# ./mcf ./inp.in.mcf.train
```

Τα input αρχεία είναι επίσης διαφορετικά για το κάθε benchmark.

4. Γρήγορα το benchmark θα φτάσει στο πρώτο breakpoint, οπότε η προσομοίωση θα διακοπεί και ο έλεγχος θα επιστρέψει στην κονσόλα του simics. Σε αυτό το σημείο, φορτώνουμε το module για τις caches:

```
simics> run-command-file cache-hierarchy.simics
```

Δείτε χαρακτηριστικά, πώς επιστρέφει ο simics τις παραμέτρους των caches, συγκρίνοντας τα αποτελέσματα που επιστρέφονται στην κονσόλα με τα περιεχόμενα του αρχείου cache-hierarcy.simics:

```
simics> dc.info
simics> ic.info
simics> l2c.info
```

Σημαντική παρατήρηση: Όπως αναφέραμε προηγουμένως, η λεπτομερής προσομοίωση της ιεραρχίας μνήμης και η συλλογή στατιστικών στοιχείων μας ενδιαφέρει για το τμήμα της εκτέλεσης του προγράμματος μετά το 1ο breakpoint (μετά τη φάση αρχικοποίησης). Αυτό σημαίνει ότι, για κάθε μία εφαρμογή, θα μπορούσατε να πάρετε ένα δεύτερο checkpoint τη στιγμή που η εκτέλεσή της φτάνει στο 1ο breakpoint, και στα επόμενα πειράματα να ξεκινάτε από το checkpoint αυτό (αφού πρώτα φορτώσετε το επιθυμητό cache configuration) χωρίς να χρειάζεται να επαναλάβετε τη φάση αρχικοποίησης. Ένα λεπτό σημείο σε αυτήν την περίπτωση είναι ότι, για να ληφθούν υπόψη από τον simics οι επόμενες εντολές που δίνετε σαν όρισμα στην εντολή continue (βλ. βήμα 5), θα πρέπει να έχετε απενεργοποιήσει τα breakpoints (δηλαδή στο βήμα 2 δεν χρειάζεται η εκτέλεση της εντολής magic-break-enable).

5. Συνεχίστε την προσομοίωση για 100,000,000 εντολές, προκειμένου να θερμανθούν επαρκώς οι caches. Αυτό θα πάρει αρκετό χρόνο, καθώς τώρα η προσομοίωση εκτελείται με πολύ μεγαλύτερη λεπτομέρεια:

```
simics> c 100_000_000
```

6. Καθαρίστε τα στατιστικά που κατέγραψαν οι caches κατά τη διάρκεια της warm-up περιόδου, και συνεχίστε την προσομοίωση για 1,000,000,000 εντολές:

```
simics> dc.reset-statistics
simics> ic.reset-statistics
simics> l2c.reset-statistics
simics> ptime
simics> pstats
simics> c 1_000_000_000
```

7. Συλλέξτε τα στατιστικά που καταγράφηκαν για τις caches και για τον επεξεργαστή:

```
simics> dc.statistics
simics> ic.statistics
simics> l2c.statistics
simics> pstats
```

```
simics> ptime
```

Την παραπάνω διαδικασία την επαναλαμβάνετε για κάθε benchmark. Προκειμένου να αυτοματοποιήσετε τη διαδικασία και να μην χρειάζεται να εισάγετε κάθε φορά τις ίδιες εντολές, μπορείτε να τις εισάγετε σε ένα δεύτερο simics script. Ένα παράδειγμα τέτοιου script παρουσιάζεται στο Παράρτημα Β. Το script αυτό αφορά την εκτέλεση μιας εφαρμογής εξ' αρχής, με όλες τις απαραίτητες αρχικοποιήσεις. Για την εκτέλεση της ίδιας εφαρμογής σε επόμενα πειράματα, αφού έχετε πάρει checkpoint μετά το 1ο breakpoint της (σύμφωνα με όσα είπαμε στο βήμα 4), μπορείτε να χρησιμοποιήσετε ένα script αντίστοιχο με αυτό του Παραρτήματος Γ.

4. Πειραματική Αξιολόγηση

Στα πλαίσια της εργασίας αυτής, θα διερευνηθεί αρχικά η επίδραση των βασικότερων παραμέτρων ιεραρχίας κρυφής μνήμης στην απόδοση της εφαρμογής. Σε δεύτερη φάση, θα διερευνηθεί για μία συγκεκριμένη παραμετροποίηση της ιεραρχίας μνήμης ο τρόπος που μεταβάλλονται διάφορες μετρικές απόδοσης στο χρόνο. Τα πειράματα που θα χρειαστεί να εκτελέσετε παρουσιάζονται στη συνέχεια.

4.1 Μελέτη επίδρασης παραμέτρων ιεραρχίας μνήμης στην απόδοση της εφαρμογής

4.1.1 L1 cache δεδομένων

Για όλες τις περιπτώσεις που εξετάζονται στο πείραμα αυτό, οι παράμετροι της L2 cache θα διατηρηθούν σταθερές και ίσες με τις default παραμέτρους (όπως δίνονται στο Παράρτημα Α).

- Για τις dc και ic θα χρησιμοποιήσετε το configuration που δίνεται στο Παράρτημα Α, αλλάζοντας κατάλληλα το associativity και τον αριθμό των γραμμών ώστε να εκτελέστε τα 3 benchmarks για τις παρακάτω περιπτώσεις:

size (dc/ic)	associativity (dc/ic)
16K	2
16K	4
32K	4
32K	8
64K	4
64K	8

- Εκτελέστε και τα 3 benchmarks για τους παρακάτω συνδυασμούς size και associativity για line size 32 και 128. Προσοχή, θα πρέπει να θέσετε το αντίστοιχο line_size και στους αντίστοιχους splitters. Επίσης για κάθε line_size θα πρέπει να υπολογίσετε και τον σωστό αριθμό γραμμών.

size (dc/ic)	associativity (dc/ic)
16K	4
32K	4
32K	8
64K	4
64K	8

4.1.2 L2 cache

Για όλες τις περιπτώσεις που θα εξεταστούν εδώ θα χρησιμοποιήσετε για ic, dc το configuration που δίνεται στο Παράρτημα Α, μεταβάλλοντας απλώς τον αριθμό γραμμών ώστε να αντιπροσωπεύουν L1 caches μεγέθους 32Kb με associativity 4.

- Για τη L2 θα χρησιμοποιήσετε το configuration που δίνεται στο Παράρτημα Α, αλλάζοντας κατάλληλα το associativity και τον αριθμό των γραμμών ώστε να εκτελέστε τα 3 benchmarks για τις παρακάτω περιπτώσεις:

size	associativity
256K	4
512K	4
512K	8
1024K	8
1024K	16
2048K	8
2048K	16

- Εκτελέστε και τα 3 benchmarks για τους παρακάτω συνδυασμούς size και associativity για line_size 64 και 256. Προσοχή, θα πρέπει να θέσετε το αντίστοιχο line_size και στους αντίστοιχους splitters. Επίσης για κάθε line_size θα πρέπει να υπολογίσετε και τον σωστό αριθμό γραμμών.

size	associativity
512K	8
1024K	8
2048K	8

4.1.3 Ζητούμενο

Προσομοιώστε όλες τις προηγούμενες περιπτώσεις για 1,000,000,000 εντολές, αφού πρώτα "θερμάνετε" τις caches για 100,000,000 εντολές (όπως ακριβώς περιγράφεται στην ενότητα 3.4).

Σαν βασική μετρική απόδοσης θα χρησιμοποιήσετε το IPC (Instructions Per Cycle). Με την προϋπόθεση ότι ο κύκλος μηχανής και ο εκτελούμενος αριθμός εντολών παραμένουν σταθεροί κάθε φορά, μεγαλύτερες τιμές στο IPC υποδεικνύουν καλύτερη απόδοση (*σημείωση: αυτό ισχύει μόνο στα πλαίσια της προσομοίωσης. Στην πράξη, οι διάφορες τροποποιήσεις στα μικροαρχιτεκτονικά χαρακτηριστικά του επεξεργαστή επιφέρουν συνήθως αλλαγές και στην διάρκεια του κύκλου ρολογιού*).

Για κάθε ένα από τα παραπάνω ερωτήματα μελετήστε τις μεταβολές στο IPC και στο miss rate της cache της οποίας τις παραμέτρους μεταβάλλετε. Παρουσιάστε σε γραφικές παραστάσεις τις μεταβολές αυτές για κάθε περίπτωση. Για παράδειγμα, στο πείραμα 4.1.a, ζητούνται στην ουσία 2 γραφικές παραστάσεις, 1 για IPC και 1 για τα miss rates. Κάθε μία από αυτές τις γραφικές παραστάσεις θα έχει στον άξονα x την παράμετρο που μεταβάλλεται (στην περίπτωση αυτή size, associativity) και θα περιέχει τα αποτελέσματα και για τα 3 benchmarks.

Συνοψίστε τα συμπεράσματα των προηγούμενων ερωτημάτων. Ποιες από τις παραμέτρους που εξετάσατε έχουν τη μεγαλύτερη επίδραση στην απόδοση;

4.2 Μελέτη μεταβολής μετρικών απόδοσης στο χρόνο

Στα προηγούμενα ερωτήματα εξετάσατε τη συμπεριφορά της απόδοσης κάθε εφαρμογής για ένα συγκεκριμένο χρονικό διάστημα, δηλαδή για τις πρώτες 1,000,000,000 εντολές της περιοχής ενδιαφέροντος. Τα αποτελέσματα αυτά ωστόσο ενδέχεται να μην είναι αντιπροσωπευτικά του συνολικού προφίλ εκτέλεσης της εφαρμογής, αφού αφενός αφορούν ένα (μικρό) μέρος της εκτέλεσής της, και αφετέρου δεν αποτυπώνουν με κάποιον τρόπο τη δυναμική της συμπεριφορά.

Σε αυτό το ερώτημα καλείστε να μελετήσετε τη δυναμική συμπεριφορά των εφαρμογών, εξετάζοντας τον τρόπο που οι διάφορες μετρικές απόδοσης μεταβάλλονται στο χρόνο. Συγκεκριμένα, διατηρήστε σταθερές τις παραμέτρους της ιεραρχίας μνήμης στις εξής τιμές:

L1D/L1I size: 32KB
L1D/L1I associativity: 4
L1 miss latency: 10 κύκλοι
L2 size: 1MB
L2 associativity: 8
L2 miss latency: 200 κύκλοι

και εκτελέστε κάθε μία από τις εφαρμογές για 5,000,000,000 εντολές. Ανά 100,000,000 εντολές, καταγράφετε το πλήθος των L1 misses, L2 misses και κύκλων που αφορούν τη συγκεκριμένη φάση εκτέλεσης. Στο Παράρτημα Δ παρατίθεται ένα τμήμα simics script μέσω του οποίου μπορείτε να διακόπτετε την προσομοίωση περιοδικά (ανά 100,000,000 εντολές), και να καταγράφετε τα στατιστικά για την εκάστοτε φάση σε ξεχωριστό αρχείο εξόδου.

Για κάθε μία μετρική απόδοσης, χρησιμοποιήστε ένα διάγραμμα που θα έχει στον x άξονα το χρόνο (στην ουσία, το id της κάθε φάσης) και στον y την τιμή της μετρικής, προκειμένου να δείξετε τη δυναμική μεταβολή της συγκεκριμένης μετρικής στο χρόνο.

Τι συμπεράσματα βγάζετε; Κατά πόσο τα αποτελέσματα που πήρατε στο ερώτημα 4.1 για την ίδια παραμετροποίηση της ιεραρχίας μνήμης, α) ανταποκρίνονται στην εκτέλεση των 5,000,000,000 εντολών, β) αντιπροσωπεύουν τη δυναμική συμπεριφορά κάθε εφαρμογής;

Παραδοτέο της άσκησης θα είναι ένα ηλεκτρονικό κείμενο (pdf, doc ή odt). Στο ηλεκτρονικό κείμενο να αναφέρετε στην αρχή τα στοιχεία σας (Όνομα, Επώνυμο, ΑΜ). Η άσκηση να παραδοθεί ηλεκτρονικά στην ιστοσελίδα:

http://www.cslab.ece.ntua.gr/courses/advcomparch/submit

Δουλέψτε ατομικά. Έχει ιδιαίτερη αξία για την κατανόηση του μαθήματος να κάνετε μόνοι σας την εργασία. Μην προσπαθήσετε να την αντιγράψετε από άλλους συμφοιτητές σας.

Μην αφήσετε την εργασία για το τελευταίο Σαββατοκύριακο, απαιτεί αρκετό χρόνο για την εκτέλεση των benchmarks, ξεκινήστε αμέσως!

Παράρτημα Α: Simics script with cache configuration

```
#
# Transaction staller for memory
#
@staller = pre_conf_object('staller', 'trans-staller')
@staller.stall_time = 0

#
# l2 cache: 1Mb
#
@l2c = pre_conf_object('l2c', 'g-cache')
@l2c.cpus = conf.cpu0
@l2c.config_line_number = 8192
@l2c.config_line_size = 128
@l2c.config_assoc = 8
@l2c.config_virtual_index = 0
@l2c.config_virtual_tag = 0
@l2c.config_write_back = 0
@l2c.config_write_allocate = 0
@l2c.config_replacement_policy = 'lru'
@l2c.penalty_read = 0
@l2c.penalty_write = 0
@l2c.penalty_read_next = 0
@l2c.penalty_write_next = 0
@l2c.timing_model = staller

#
# instruction cache: 32Kb
#
@ic = pre_conf_object('ic', 'g-cache')
@ic.cpus = conf.cpu0
@ic.config_line_number = 512
@ic.config_line_size = 64
@ic.config_assoc = 4
@ic.config_virtual_index = 0
@ic.config_virtual_tag = 0
@ic.config_replacement_policy = 'lru'
@ic.penalty_read = 0
@ic.penalty_write = 0
@ic.penalty_read_next = 0
@ic.penalty_write_next = 0
@ic.timing_model = l2c

#
# data cache: 32Kb
#
@dc = pre_conf_object('dc', 'g-cache')
@dc.cpus = conf.cpu0
@dc.config_line_number = 512
@dc.config_line_size = 64
@dc.config_assoc = 4
@dc.config_virtual_index = 0
@dc.config_virtual_tag = 0
@dc.config_replacement_policy = 'lru'
@dc.penalty_read = 0
@dc.penalty_write = 0
@dc.penalty_read_next = 0
@dc.penalty_write_next = 0
@dc.timing_model = l2c

#
```

```
# transaction splitter for instruction cache
#
@ts_i = pre_conf_object('ts_i', 'trans-splitter')
@ts_i.cache = ic
@ts_i.timing_model = ic
@ts_i.next_cache_line_size = 64

#
# transaction splitter for data cache
#
@ts_d = pre_conf_object('ts_d', 'trans-splitter')
@ts_d.cache = dc
@ts_d.timing_model = dc
@ts_d.next_cache_line_size = 64

#
# instruction-data splitter
#
@id = pre_conf_object('id', 'id-splitter')
@id.ibranch = ts_i
@id.dbranch = ts_d

@SIM_add_configuration([staller, l2c, ic, dc, ts_i, ts_d, id], None)

@conf.phys_mem0.timing_model = conf.id
```

Παράρτημα Β: Ενδεικτικό simics script για την εκτέλεση μιας εφαρμογής (πρώτη εκτέλεση, προσομοίωση εξ' αρχής)

```
#Invoke as follows:  simics -stall -x runbenchmark.simics
#Load checkpoint
read-configuration /home/user/simics-workspace/checkpoints/after-boot.conf
hfs0.root /home/user/

magic-break-enable
instruction-fetch-mode instruction-fetch-trace
istc-disable
dstc-disable

#Mount host, cp benchmark to root folder, umount host, and start simulation
con0.input "bash\n"
con0.input "mount /host\n"
con0.input "cp /host/simics-workspace/bencmarks/mcf /\n"
con0.input "cp /host/simics-workspace/bencmarks/inp.in.mcf.train /\n"
con0.input "umount /host\n"
con0.input "cd /\n"
con0.input "./mcf inp.in.mcf.train \n"
c

#First breakpoint encountered

#NOTE: you may take checkpoint here for each different application, in order
#to omit the execution of its initialization phase in subsequent simulations (see
#Parartima C). E.g.:
#write-configuration /home/user/simics-workspace/checkpoints/after-first-break.conf

#Load cache hierarcy parameters
echo "Loading caches"
run-command-file cache-hierarchy.simics

#Continue simulation for 100,000,000 instructions to warm-up caches
echo "Warming up caches"
c 100_000_000

#Reset statistics
dc.reset-statistics
ic.reset-statistics
l2c.reset-statistics

pstats
ptime

#Continue simulation for 1,000,000,000 instructions
echo "Simulating..."
c 1000_000_000

#Collect statistics
dc.statistics
ic.statistics
l2c.statistics

pstats
ptime

exit
```

Παράρτημα Γ: Ενδεικτικό simics script για την εκτέλεση μιας εφαρμογής (εκτέλεση μετά το 1ο breakpoint)

```
#Invoke as follows:  simics -stall -x runbenchmark.simics
#Load checkpoint taken after the first breakpoint of the application
read-configuration /home/user/simics-workspace/checkpoints/after-first-break.conf

#We do not enable breakpoints
instruction-fetch-mode instruction-fetch-trace
istc-disable
dstc-disable

#We do not need to mount host and copy benchmarks to root folder. They are already
#copied to the checkpoint we saved.

#Here is the point right after the first breakpoint.

#Load cache hierarcy parameters
echo "Loading caches"
run-command-file cache-hierarchy.simics

#Continue simulation for 100,000,000 instructions to warm-up caches
echo "Warming up caches"
c 100_000_000

#Reset statistics
dc.reset-statistics
ic.reset-statistics
l2c.reset-statistics

pstats
ptime

#Continue simulation for 1,000,000,000 instructions
echo "Simulating..."
c 1000_000_000

#Collect statistics
dc.statistics
ic.statistics
l2c.statistics

pstats
ptime

exit
```

Παράρτημα Δ: Ενδεικτικό simics script για καταγραφή αποτελεσμάτων ανά φάσεις

```
#initialization actions as in previous examples
#Continue simulation for 100,000,000 instructions to warm-up caches
echo "Warming up caches"
c 100_000_000


$phase = 0

while $phase < 50 {

        #write results for current phase at different file
        $outputfile = ("output." + "phase"+ $phase)
        output-file-start $outputfile

        #We need to reset statistics at the beginning of each new phase
        #(without of course resetting the cache contents)
        dc.reset-statistics
        ic.reset-statistics
        l2c.reset-statistics

        pstats
        ptime

        #Continue simulation for 100,000,000 instructions
        echo "Simulating for phase " + $phase
        c 100_000_000

        #Collect statistics
        dc.statistics
        ic.statistics
        l2c.statistics

        pstats
        ptime

        output-file-stop $outputfile

        $phase += 1
}
exit
```