1η ΑΣΚΗΣΗ ΠΡΟΗΓΜΕΝΑ ΘΕΜΑΤΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Ακ. έτος 2017-2018, 8ο Εξάμηνο, Σχολή ΗΜ&ΜΥ

Τελική Ημερομηνία Παράδοσης: **25/03/2018**

1. Εισαγωγή

Στα πλαίσια της παρούσας άσκησης θα χρησιμοποιήσετε το εργαλείο *"PIN"* για να μελετήσετε την επίδραση διαφόρων παραμέτρων της ιεραρχίας μνήμης στην απόδοση ενός συνόλου εφαρμογών. Στόχος της άσκησης είναι η εξοικείωση με το εργαλείο PIN καθώς και με την διαδικασία διεξαγωγής πειραματικών μετρήσεων με σκοπό την εξαγωγή χρήσιμων συμπερασμάτων.

2. Το εργαλείο PIN

Το PIN είναι ένα εργαλείο το οποίο αναπτύσσεται και συντηρείται από την Intel και χρησιμοποιείται για την ανάλυση εφαρμογών. Χρησιμοποιείται για dynamic binary instrumentation, δηλαδή για την εισαγωγή κώδικα δυναμικά (την στιγμή που εκτελείται η εφαρμογή) ανάμεσα στις εντολές της εφαρμογής με σκοπό την συλλογή πληροφοριών σχετικά με την εκτέλεση (π.χ. cache misses, total instructions κλπ.).

Περισσότερες πληροφορίες σχετικά με το PIN καθώς και εγχειρίδια χρήσης μπορείτε να βρείτε εδώ:

https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool

3. Λήψη και εγκατάσταση του PIN

Την τελευταία έκδοση του PIN μπορείτε να την κατεβάσετε από εδώ προκειμένου να την εγκαταστήσετε στο σύστημα σας:

https://software.intel.com/en-us/articles/pintool-downloads

Το PIN εξαρτάται άμεσα από τον πυρήνα του λειτουργικού συστήματος, οπότε υπάρχει πιθανότητα κάποιες εκδόσεις του PIN να έχουν ασυμβατότητα με συγκεκριμένες εκδόσεις του πυρήνα Linux. Τα βήματα που παρουσιάζονται παρακάτω για την εκτέλεση του PIN έχουν δοκιμαστεί επιτυχώς στους παρακάτω συνδυασμούς PIN και Linux:

- PIN 71313, Ubuntu Linux 14.04 με έκδοση πυρήνα 3.13
- PIN 81205, Ubuntu Linux 16.04 με έκδοση πυρήνα 4.4
- PIN 97554, Ubuntu Linux 14.04 με έκδοση πυρήνα 3.13
- PIN 97554, Ubuntu Linux 16.04 με έκδοση πυρήνα 4.4

Αφού ολοκληρωθεί η λήψη του αρχείου θα πρέπει να το αποσυμπιέσετε δίνοντας σε ένα τερματικό την παρακάτω εντολή:

```
$ tar xvfz pin-3.6-97554-g31f0a167d-gcc-linux.tar.gz
```

Τώρα μπορείτε να περιηγηθείτε στα περιεχόμενα του PIN:

```
$ cd pin-3.6-97554-g31f0a167d-gcc-linux
$ ls -aF
./  ../  doc/  extlicense/  extras/  ia32/  intel64/  LICENSE  pin  README
redist.txt source/
```

Το **pin** είναι το εκτελέσιμο που θα χρησιμοποιήσετε για την εκτέλεση των εφαρμογών στα πειράματά σας. Για να δείτε τον τρόπο χρήσης του pin.sh μπορείτε να το εκτελέσετε χωρίς ορίσματα:

```
$ ./pin

E: Missing application name
Pin: pin-3.6-97554-31f0a167d
Copyright (c) 2003-2017, Intel Corporation. All rights reserved.

Usage: pin [OPTION] [-t <tool> [<toolargs>]] -- <command line>
Use -help for a description of options
```

Με το όρισμα **–t** λέτε στο PIN ποιό pintool να χρησιμοποιήσει ενώ ώς **<command line>** δίνεται το εκτελέσιμο το οποίο θα αναλυθεί από το PIN καθώς και τα ορίσματά του. Ένα παράδειγμα εκτέλεσης του pin δίνεται παρακάτω:

```
$ ./pin -t ./source/tools/ManualExamples/obj-intel64/inscount0.so \
                                -o ls.inscount0.output -- /bin/ls -aF
./ ../ doc/ extras/ ia32/ intel64/ LICENSE pin README redist.txt source/
$ cat ls.inscount0.output
Count 453337
```

Στο παραπάνω παράδειγμα χρησιμοποιήθηκε το pintool **inscount0.so** το οποίο αθροίζει το σύνολο των εντολών που εκτελούνται. Τα pintools είναι προγράμματα γραμμένα σε C++ που χρησιμοποιούνται από το PIN και επικοινωνούν με αυτό μέσω του API του για να κατευθύνουν την ανάλυση των εφαρμογών. Μπορείτε να γράψετε τα δικά σας pintools αλλά υπάρχουν και αρκετά που παρέχονται μαζί με το PIN. Θα τα βρείτε στον φάκελο **pin-3.6-97554-g31f0a167d-gcc-linux /source/tools/**. Για να τα μεταγλωττίσετε μπορείτε να εκτελέσετε την εντολή make στον φάκελο που σας ενδιαφέρει.

4. Μετροπρογράμματα

Το PIN μπορεί να χρησιμοποιηθεί για την εκτέλεση οποιασδήποτε εφαρμογής. Στα πλαίσια της παρούσας άσκησης θα χρησιμοποιήσετε τα PARSEC benchmarks για τα οποία μπορείτε να βρείτε περισσότερες πληροφορίες εδώ:

http://parsec.cs.princeton.edu/

Κατεβάστε την έκδοση 3.0 της σουίτας από το παρακάτω link:

http://parsec.cs.princeton.edu/download/3.0/parsec-3.0-core.tar.gz

Επίσης, θα χρειαστείτε τα αρχεία εισόδου για τα benchmarks τα οποία μπορείτε να κατεβάσετε από εδώ:

http://parsec.cs.princeton.edu/download/3.0/parsec-3.0-input-sim.tar.gz

Στη συνέχεια αποσυμπιέστε τα ληφθέντα αρχεία:

```
$ tar xvfz parsec-3.0-core-tar.gz
$ tar xvfz parsec-3.0-input-sim.tar.gz
```

Η σουίτα περιλαμβάνει 13 benchmarks και διάφορα αρχεία εισόδου. Για τους σκοπούς της άσκησης θα χρησιμοποιήσετε τα παρακάτω 10 benchmarks με τα simlarge αρχεία εισόδου:

1. blackscholes
2. bodytrack
3. canneal
4. facesim
5. ferret
6. fluidanimate
7. freqmine
8. raytrace
9. swaptions
10. streamcluster

Στον κώδικα κάθε benchmark έχουν οριστεί οι περιοχές του κώδικα που παρουσιάζουν το μεγαλύτερο ενδιαφέρον (Regions of Interest, ROI). Τα ROI ορίζονται από τις κλήσεις των συναρτήσεων **__parsec_roi_begin()** και **__parsec_roi_end()**.

Για την διαχείριση (μεταγλώττιση, εκτέλεση κ.λ.π.) των PARSEC benchmarks παρέχεται το script bin/parsecmgmt του οποίου οι σημαντικότερες παράμετροι είναι:

- -a action : η ενέργεια που θέλουμε να γίνει, π.χ. build, run, status
- -p benchmark : το όνομα του benchmark που θέλουμε να μεταγλωττίσουμε ή να εκτελέσουμε
- -c config-file : το αρχείο ρυθμίσεων που θα χρησιμοποιηθεί για την μεταγλώττιση. Παρέχονται μεταξύ άλλων:
  - gcc-serial: μεταγλώττιση σειριακών benchmarks
  - gcc-pthreads: μεταγλώττιση παράλληλων benchmarks με χρήση των pthreads
  - gcc-hooks: μεταλώττιση παράλληλων benchmarks με χρήση pthreads και hooks που ορίζουν τα ROI

Επειδή δεν παρέχεται κάποιο config αρχείο που να μεταγλωττίζει τη σειριακή έκδοση των benchmarks με ταυτόχρονη χρήση των hooks για τα ROI θα χρειαστεί να τροποποιηθούν τα config αρχεία της έκδοσης gcc-serial. Το script **cslab_process_parsec_benchmarks.sh** που παρέχεται με τον βοηθητικό κώδικα της άσκησης κάνει όλες τις κατάλληλες τροποποιήσεις και θα πρέπει να το εκτελέσετε μέσα στον φάκελο parsec-3.0.

Για να μεταγλωττίσετε τα benchmarks που θα χρησιμοποιηθούν στο πειραματικό σκέλος της άσκησης εκτελέστε τις παρακάτω εντολές:

```
$ ~/advcomparch-2017-18-ex1-helpcode/cslab_process_parsec_benchmarks.sh
$ sudo apt-get update
$ sudo apt-get install make g++ libx11-dev libxext-dev libxaw7 \
x11proto-xext-dev libglu1-mesa-dev libxi-dev libxmu-dev
```

```
$ ./bin/parsecmgmt -a build -c gcc-serial -p blackscholes bodytrack
canneal facesim ferret fluidanimate freqmine raytrace swaptions
streamcluster
```

Στη συνέχεια εκτελέστε το script **cslab_create_parsec_workspace.sh** το οποίο θα δημιουργήσει έναν φάκελο *parsec_workspace* που θα περιέχει όλα τα εκτελέσιμα που θα χρειαστείτε καθώς και όλα τα αρχεία εισόδου για τα benchmarks. Τέλος, στον βοηθητικό κώδικα σας δίνεται το αρχείο **cmds_simlarge.txt** που περιέχει τις εντολές για την εκτέλεση κάθε benchmark. Έχουμε επιλέξει να μην χρησιμοποιηθεί το script parsecgmgt για την εκτέλεση των benchmarks, καθώς με τη χρήση του δημιουργούνται επιπλέον διεργασίες κατά την εκτέλεση των benchmarks, γεγονός που μπορεί να επηρεάσει τις μετρήσεις σας.

```
$ ~/advcomparch-2017-18-ex1-helpcode/cslab_create_parsec_workspace.sh
$ cd parsec_workspace
$ cp ~/advcomparch-2017-18-ex1-helpcode/cmds_simlarge.txt
$ cat cmds_simlarge.txt
./executables/blackscholes 1 inputs/in_64K.txt prices.txt
./executables/bodytrack inputs/sequenceB_4 4 4 4000 5 0 1
./executables/canneal 1 15000 2000 inputs/400000.nets 128
./executables/facesim -timing -threads 1
./executables/ferret inputs/corel lsh inputs/queries 10 20 1 output.txt
./executables/fluidanimate 1 5 inputs/in_300K.fluid out.fluid
./executables/freqmine inputs/kosarak_990k.dat 790
./executables/rtview inputs/happy_buddha.obj -automove -nthreads 1 -frames
3 -res 1920 1080
./executables/streamcluster 10 20 128 16384 16384 1000 none output.txt 1
./executables/swaptions -ns 64 -sm 40000 -nt 1
```

Πριν την εκτέλεση οποιουδήποτε benchmark *θα πρέπει* να έχετε ορίσει την μεταβλητή περιβάλλοντος LD_LIBRARY_PATH ώστε να δείχνει στο PATH που περιέχει την βιβλιοθήκη των hooks:

```
$ export LD_LIBRARY_PATH=~/parsec-3.0/pkgs/libs/hooks/inst/amd64-
linux.gcc-serial/lib
```

5. PINTOOL: simulator

Στον βοηθητικό κώδικα της άσκησης θα βρείτε το pintool **simulator.cpp**, το οποίο θα χρησιμοποιήσετε για την προσομοίωση της εκτέλεσης εφαρμογών σε ένα περιβάλλον με έναν in-order επεξεργαστή, κρυφή μνήμη δύο επιπέδων (L1-Data + L2 caches) και μνήμη μετάφρασης διευθύνσεων (TLB). Το simulator.cpp είναι γραμμένο έτσι ώστε να ενεργοποιείται μόνο κατά την διάρκεια των PARSEC ROI. Για την μεταγλώττισή του, κατευθυνθείτε στο directory του pintool που σας παρέχεται:

```
$ cd advcomparch-2017-18-ex1-helpcode/pintool
```

Τροποποιήστε κατάλληλα το αρχείο `makefile` ώστε το PIN_ROOT να δείχνει στο path του pin, δηλαδή:

PIN_ROOT ?= `/path/to/pin-3.6-97554-g31f0a167d-gcc-linux`

Και εκτελέστε:

```
$ make clean; make
```

Το simulator pintool δέχεται τα παρακάτω ορίσματα:

- -o <filename> : το αρχείο εξόδου όπου θα αποθηκευτούν οι παράμετροι και τα στατιστικά της προσομοίωσης
- -L1c : το μέγεθος της L1 (KB)
- -L1a : το associativity της L1
- -L1b : το μέγεθος του block της L1 (bytes)
- -L2c : το μέγεθος της L2 (KB)
- -L2a : το associativity της L2
- -L2b : το μέγεθος του block της L2 (bytes)
- -L2prf : ο αριθμός των blocks που γίνονται prefetched στην L2 (η default τιμή 0 απενεργοποιεί το prefetching)
- -TLBe : το μέγεθος σε καταχωρίσεις (entries) του TLB
- -TLBa : το associativity του TLB
- -TLBp : το μέγεθος της σελίδας μνήμης (pagesize) που χρησιμοποιεί το TLB (bytes)

Με τις κατάλληλες τροποποιήσεις, το simulator pintool μπορεί επίσης να προσομοιώσει διαφορετικές στρατηγικές ως προς το write allocation ή διαφορετικές πολιτικές αντικατάστασης στις caches και στο TLB.

Παράδειγμα χρήσης του simulator pintool:

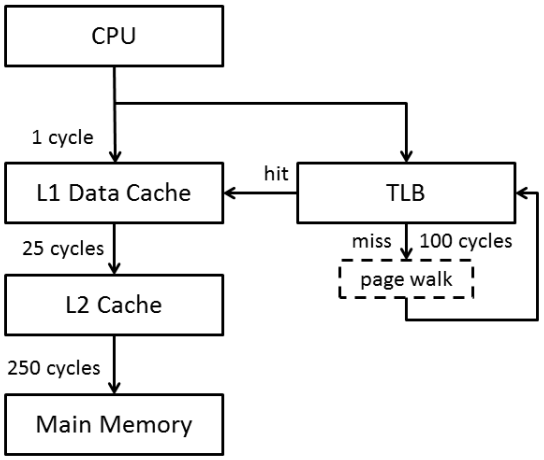
```
$ /path/to/pin-3.6-97554-g31f0a167d-gcc-linux/pin \
 -t /path/to/advcomparch-2017-18-ex1-helpcode/pintool/obj-
intel64/simulator.so \
-o my_output.out –L1c 64 –L1a 8 –L1b 64 –L2c 256 –L2a 8 –L2b 64 \
  -TLBe 64 –TLBa 4 –TLBp 4096 -- \
 /path/to/parsec_workspace/executables/blackscholes \
  1 /path/to/parsec_workspace /inputs/in_64K.txt prices.txt
```

6. Προσομοίωση Ιεραρχίας μνήμης

Η ιεραρχία κρυφής μνήμης που θα χρησιμοποιήσουμε στα πλαίσια αυτής της άσκησης απεικονίζεται στο παρακάτω σχήμα. Πιο συγκεκριμένα, το simulator pintool προσομοιώνει έναν in-order επεξεργαστή με την inclusive ιεραρχία μνήμης και μετάφραση διευθύνσεων που φαίνεται στο παραπάνω σχήμα. Για τον υπολογισμό της επίδοσης των εφαρμογών που χρησιμοποιούνται στις προσομοιώσεις, χρησιμοποιείται ένα απλό μοντέλο, όπου θεωρούμε ότι κάθε εντολή απαιτεί 1 κύκλο για την εκτέλεσή της (IPC=1).

Επιπρόσθετα, οι εντολές που πραγματοποιούν πρόσβαση στη μνήμη (είτε load είτε store) προκαλούν επιπλέον καθυστερήσεις ανάλογα με το αν οι μεταφράσεις διευθύνσεων βρίσκονται στο TLB και με το πού βρίσκονται τα δεδομένα τους. Θεωρούμε ότι η L1 είναι υλοποιημένη ως **Virtually indexed, Physically tagged (VIPT)** cache, δηλαδή οι προσβάσεις στο TLB και στην L1 cache επικαλύπτονται και πραγματοποιούνται παράλληλα. Αν η ζητούμενη μετάφραση δεν βρίσκεται στο TLB (TLB miss),

τότε εισάγεται μία καθυστέρηση 100 κύκλων που προσομοιώνει την καθυστέρηση ανάκλησης της μετάφρασης διεύθυνσης από την μνήμη (page walk), και στη συνέχεια εκτελείται η πρόσβαση στην ιεραρχία μνήμης μέσω της L1 cache.



Το TLB module του simulator χρησιμοποιείται μόνο για την μελέτη της επίδρασης της μετάφρασης διευθύνσεων στην επίδοση των προγραμμάτων όσον αφορά τον χρόνο εκτέλεσης, και όχι για να παρέχει πραγματική μετάφραση διευθύνσεων στον φυσικό χώρο. Δηλαδή, κατά την προσομοίωση το cache module χρησιμοποιεί εικονικές διευθύνσεις για την πρόσβαση στην μνήμη δεδομένων και τον έλεγχο των cache hits/misses.

Συνεπώς, όπως μπορείτε να διακρίνετε και στο παραπάνω σχήμα έχουμε τις εξής περιπτώσεις:

1. TLB hit: 0 cycles (η πρόσβαση πραγματοποιείται παράλληλα με την L1 cache)
2. TLB miss: 100 cycles
3. L1 hit: 1 cycle
4. L2 hit: 25 cycles
5. Main memory access: 250 cycles

Οι παραπάνω τιμές κύκλων μπορούν να δοθούν ως παράμετροι κατά την αρχικοποίηση της ιεραρχίας μνήμης στο simulator pintool.

Συνολικά, ο αριθμός των κύκλων υπολογίζεται ως:

Cycles = Inst + TLB_Misses*TLB_miss_cycles + L1_Accesses * L1_hit_cycles + L2_Accesses * L2_hit_cycles + Mem_Accesses * Mem_acc_cycles

7. 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.

7.1 Μελέτη επίδρασης παραμέτρων ιεραρχίας μνήμης στην απόδοση της εφαρμογής

7.1.1 L1 cache

Για όλες τις περιπτώσεις που εξετάζονται στο πείραμα αυτό, οι παράμετροι της L2 cache και του TLB θα διατηρηθούν σταθερές και συγκεκριμένα ίσες με:

- L2 size = 1024 KB
- L2 associativity = 8
- L2 block size = 128 B
- TLB size = 64 entr.
- TLB associativity = 4
- TLB page size = 4096 B

Εκτελέστε τα benchmarks για τις παρακάτω L1 caches:

L1 size	L1 associativity	L1 cache block size
16KB	4	32B, 64B, 128B
32KB	4	32B, 64B, 128B
32KB	8	64B

L1 size	L1 associativity	L1 cache block size
64KB	4	32B, 64B, 128B
64KB	8	64B
128KB	8	64B

7.1.2 L2 cache

Για όλες τις περιπτώσεις που εξετάζονται στο πείραμα αυτό, οι παράμετροι της L1 cache και του TLB θα διατηρηθούν σταθερές και συγκεκριμένα ίσες με:

- L1 size = 32 KB
- L1 associativity = 8
- L1 block size = 64 B
- TLB size = 64 entr.
- TLB associativity = 4
- TLB page size = 4096 B

Εκτελέστε τα benchmarks για τις παρακάτω L2 caches:

L2 size	L2 associativity	L2 cache block size
256 KB	4	128B
512 KB	4	128B
512 KB	8	64B, 128B, 256B
1024 KB	8	64B, 128B, 256B
1024 KB	16	128B
2048 KB	8	64B, 128B, 256B
2048KB	16	128B

7.1.3 TLB

Για όλες τις περιπτώσεις που εξετάζονται στο πείραμα αυτό, οι παράμετροι της L1 και L2 cache θα διατηρηθούν σταθερές και συγκεκριμένα ίσες με:

- L1 size = 32 KB
- L1 block size = 64 B
- L1 associativity = 8
- L2 size = 1024 KB
- L2 associativity = 8
- L2 block size = 128 B

Εκτελέστε τα benchmarks για τα παρακάτω TLB:

TLB size	TLB associativity	TLB page size
8	4	4096B
16	4	4096B
32	4	4096B
64	1	4096B
64	2	4096B
64	4	4096B

TLB Size	TLB associativity	TLB page size
64	8	4096B
64	16	4096B
64	32	4096B
64	64	4096B
128	4	4096B
256	4	4096B

7.1.4 Προανάκληση (prefetching)

Σε όλες τις προηγούμενες προσομοιώσεις, η προανάκληση (prefetching) είναι απενεργοποιημένη. Επεκτείνετε κατάλληλα τον κώδικα τώρα, ώστε το σύστημα να προσομοιώνει prefetching στη L2 cache. Πιο συγκεκριμένα, για κάθε L2 miss θα πρέπει η cache να φέρνει εκτός από το ζητούμενο block και τα επόμενα n blocks, όπου n ο αριθμός που ορίζεται από την παράμετρο L2prf.

Εκτελέστε τα benchmarks για n = 1, 2, 4, 8, 16, 32, 64. Οι παράμετροι των L1, L2, TLB θα διατηρηθούν σταθερές και συγκεκριμένα ίσες με:

- L1 size = 32 KB
- L1 block size = 64 B
- L1 associativity = 8
- L2 size = 1024KB
- L2 block size = 128 B
- L2 associativity = 8
- TLB size = 64 entr.
- TLB associativity = 4
- TLB page size = 4096 B

7.1.5 Ζητούμενο

Σαν βασική μετρική επίδοσης 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 μια από τις παραπάνω περιπτώσεις μελετήστε τις μεταβολές στο IPC και στην επίδοση της cache της οποίας τις παραμέτρους μεταβάλλετε. Παρουσιάστε σε γραφικές παραστάσεις τις μεταβολές αυτές για κάθε περίπτωση. Συνοψίστε τα συμπεράσματα των προηγούμενων ερωτημάτων. Ποιες από τις παραμέτρους που εξετάσατε έχουν τη μεγαλύτερη επίδραση στην απόδοση;

7.2 Μελέτη μεταβολής μετρικών απόδοσης στο χρόνο

Στα προηγούμενα ερωτήματα εξετάσατε τη συμπεριφορά κάθε εφαρμογής για ολόκληρη την περιοχή ενδιαφέροντος όπως αυτή έχει οριστεί στον κώδικα της κάθε εφαρμογής. Στο ερώτημα αυτό καλείστε να μελετήσετε τη δυναμική συμπεριφορά των εφαρμογών, εξετάζοντας τον τρόπο που οι διάφορες μετρικές απόδοσης μεταβάλλονται στο χρόνο.

Τροποποιήστε κατάλληλα τον κώδικα που σας έχει δοθεί ώστε να αποθηκεύετε στατιστικά κάθε 10M εντολές. Διαλέξτε ένα configuration (π.χ. L1 32KB, 8-way, block size 64B & L2 1024KB, 8-way,

block size 128B, write-allocate ιεραρχία με LRU πολιτική αντικατάστασης, TLB 64 entries, 4-way, 4096 page size) και μελετήστε τη δυναμική συμπεριφορά των μετρικών επίδοσης για τα benchmarks. Για κάθε μία μετρική επίδοσης χρησιμοποιήστε ένα διάγραμμα που θα έχει στον x άξονα το χρόνο (στην ουσία τον αριθμό των εντολών στο κάθε σημείο που αποθηκεύτηκαν αποτελέσματα) 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 7.1 για την ίδια παραμετροποίηση της ιεραρχίας μνήμης, ανταποκρίνονται στην εκτέλεση των πρώτων 10M ή των πρώτων 100M εντολών; Αντιπροσωπεύουν οι πιο «σύντομες» αυτές προσομοιώσεις τη δυναμική συμπεριφορά κάθε εφαρμογής;

Παραδοτέο της άσκησης θα είναι ένα ηλεκτρονικό κείμενο (**pdf**, **docx** ή **odt**). *Στο ηλεκτρονικό κείμενο να αναφέρετε στην αρχή τα στοιχεία σας (Όνομα, Επώνυμο, ΑΜ).*

Η άσκηση θα 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 όλων των προσομοιώσεων, ξεκινήστε αμέσως!