2η ΑΣΚΗΣΗ ΠΡΟΗΓΜΕΝΑ ΘΕΜΑΤΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Ακ. έτος 2019-2020, 8ο Εξάμηνο, Σχολή ΗΜ&ΜΥ

Τελική Ημερομηνία Παράδοσης: **10/05/2020**

1. Εισαγωγή

Στα πλαίσια της παρούσας άσκησης θα χρησιμοποιήσετε το εργαλείο *"PIN"* για να μελετήσετε την επίδραση διαφορετικών συστημάτων πρόβλεψης εντολών άλματος καθώς και η αξιολόγηση τους με δεδομένο το διαθέσιμο χώρο πάνω στο τσιπ.

2. PINTOOL

Στον βοηθητικό κώδικα της άσκησης θα βρείτε τα pintools **cslab_branch_stats.cpp** και **cslab_branch.cpp**. Αφού τροποποιήσετε το PIN_ROOT path στο αρχείο makefile, για την μεταγλώττισή τους δώστε:

```
$ cd advcomparch-2019-2-ex2-helpcode/pintool
$ make clean; make
```

Το *cslab_branch_stats.cpp* χρησιμοποιείται για την εξαγωγή στατιστικών σχετικά με τις εντολές αλμάτων που εκτελούνται από την εφαρμογή. Ένα παράδειγμα χρήσης δίνεται παρακάτω:

```
$ cd /path/to/advcomparch-2019-20-ex2-
helpcode/spec_execs_train_inputs/434.zeusmp
$ /path/to/pin-3.6-97554-g31f0a167d-gcc-linux/pin \
-t/path/to/advcomparch-2017-18-ex2-helpcode/pintool/obj-
intel64/cslab_branch_stats.so \
-o my_output.out -- \
./zeusmp_base.amd64-m64-gcc42-nn 1> zeusmp.out 2> zeusmp.err
$ cat my_output.out
Total Instructions: 103056416223

Branch statistics:
  Total-Branches: 7550195084
  Conditional-Taken-Branches: 3286815852
  Conditional-NotTaken-Branches: 3570524450
  Unconditional-Branches: 692782516
  Calls: 36135
  Returns: 36131
```

Το *cslab_branch.cpp* χρησιμοποιείται για την αξιολόγηση τεχνικών πρόβλεψης άλματος ενώ για τις εντολές επιστροφής από διαδικασίες προσομοιώνει διαφορετικά μεγέθη στοίβας διεύθυνσης επιστροφής (RAS). Ένα παράδειγμα χρήσης δίνεται παρακάτω:

```
$ /path/to/pin-3.6-97554-g31f0a167d-gcc-linux/pin \
-t /path/to/advcomparch-2017-18-ex2-helpcode/pintool/obj-
intel64/cslab_branch.so \
-o my_output.out -- \
./zeusmp_base.amd64-m64-gcc42-nn 1> zeusmp.out 2> zeusmp.err
```

Στο αρχείο **branch_predictor.h** ορίζουμε τους διαφορετικούς branch predictors. Για την προσθήκη ενός branch predictor απαιτείται η δημιουργία μίας νέας υπο-κλάσης της κλάσης **BranchPredictor** και ο ορισμός τριών μεθόδων **predict()**, **update()** και **getName()**. Η πρώτη συνάρτηση δέχεται ως ορίσματα το PC της εντολής και τη διεύθυνση προορισμού και καλείται να προβλέψει αν το άλμα θα εκτελεστεί ή όχι (Taken / Not Taken). Η δεύτερη μέθοδος καλείται να αποθηκεύσει τις πληροφορίες εκείνες που απαιτούνται για τις μελλοντικές προβλέψεις. Τα ορίσματα της είναι η πρόβλεψη που έκανε ο predictor, το πραγματικό αποτέλεσμα της εντολής διακλάδωσης, το PC της εντολής και η διεύθυνση προορισμού. Τέλος, η μέθοδος **getName()** χρησιμοποιείται για την εκτύπωση των αποτελεσμάτων του branch predictor στο αρχείο εξόδου του pintool.

3. Μετροπρογράμματα

Το PIN μπορεί να χρησιμοποιηθεί για την εκτέλεση οποιασδήποτε εφαρμογής. Στα πλαίσια της παρούσας άσκησης θα χρησιμοποιήσετε τα SPEC_CPU2006 benchmarks. Πιο συγκεκριμένα θα χρησιμοποιήσετε τα παρακάτω 12 benchmarks:

1. 403.gcc
2. 429.mcf
3. 434.zeusmp
4. 436.cactusADM
5. 445.gobmk
6. 450.soplex
7. 456.hmmer
8. 458.sjeng
9. 459.GemsFDTD
10. 471.omnetpp
11. 473.astar
12. 483.xalancbmk

Στον βοηθητικό κώδικα της άσκησης σας δίνεται ο φάκελος *spec_execs_train_inputs* ο οποίος περιέχει τα εκτελέσιμα και τα απαραίτητα αρχεία εισόδου για τα παραπάνω benchmarks.

4. Πειραματική Αξιολόγηση

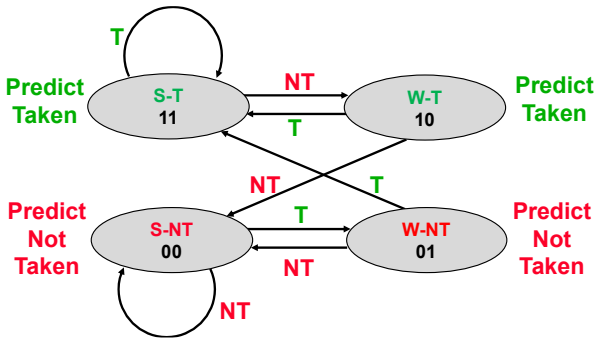
4.1 Μελέτη εντολών άλματος

Στο πρώτο κομμάτι της πειραματικής αξιολόγησης ο στόχος είναι η συλλογή στατιστικών για τις εντολές άλματος που εκτελούνται από τα benchmarks. Χρησιμοποιήστε το cslab_branch_stats.cpp και για κάθε benchmark δώστε ένα διάγραμμα που να δείχνει τον αριθμό των εντολών άλματος που εκτελέστηκαν και το ποσοστό αυτών που ανήκουν σε κάθε κατηγορία (conditional-taken, conditional-nottaken κλπ.).

4.2 Μελέτη των N-bit predictors

Μελετήστε την απόδοση των n-bits predictors χρησιμοποιώντας την υλοποίηση τους στο cslab_branch.cpp.

(i) Διατηρώντας σταθερό τον αριθμό των BHT entries και ίσο με 16K, προσομοιώστε τους n-bit predictors, για N=1, 2, 3, 4. Τα n-bits υλοποιούν ένα saturating up-down counter όπως είδαμε στις διαλέξεις. Για N=2 υλοποιήστε επιπλέον και το παρακάτω εναλλακτικό FSM (ως 2β). Συγκρίνετε τους 5 predictors χρησιμοποιώντας τη μετρική direction Mispredictions Per Thousand Instructions (direction MPKI).



(ii) Στο προηγούμενο ερώτημα η αύξηση του αριθμού των bits ισοδυναμεί με αύξηση του απαιτούμενου hardware, αφού ο αριθμός των entries του BHT παραμένει σταθερός. Διατηρώντας τώρα σταθερό το hardware και ίσο με 32K bits, εκτελέστε ξανά τις προσομοιώσεις για τα 12 benchmarks, θέτοντας N=1, 2, 2β, 4 και τον κατάλληλο αριθμό entries. Δώστε το κατάλληλο διάγραμμα και εξηγείστε τις μεταβολές που παρατηρείτε. Ποιον predictor θα διαλέγατε ως την βέλτιστη επιλογή;

4.3 Μελέτη του BTB

Υλοποιήστε έναν BTB και να μελετήστε την ακρίβεια πρόβλεψής του για τις ακόλουθες περιπτώσεις:

btb entries	btb associativity
512	1, 2
256	2, 4
128	4
64	8

Προσομοιώστε για τα benchmarks που παρέχονται και δώστε όπως και πριν τα κατάλληλα διαγράμματα. Υπενθυμίζεται ότι για τον BTB υπάρχουν 2 περιπτώσεις misses. Η πρώτη είναι direction misprediction και η δεύτερη target misprediction στην περίπτωση ενός direction hit. Πώς θα εξηγούσατε τη διαφορά επίδοσης ανάμεσα στις διαφορετικές περιπτώσεις; Διαλέξτε την καλύτερη οργάνωση για το BTB.

4.4 Μελέτη του RAS

Χρησιμοποιώντας την υλοποίηση της RAS (**ras.h**) που σας δίνεται, μελετήστε το ποσοστό αστοχίας για τις ακόλουθες περιπτώσεις:

Αριθμός εγγραφών στη RAS
4
8
16
24

Αριθμός εγγραφών στη RAS
32
48
64

Προσομοιώστε για τα benchmarks που παρέχονται και δώστε όπως και πριν τα κατάλληλα διαγράμματα εξηγώντας τις μεταβολές που παρατηρείτε. Επιλέξτε το κατάλληλο μέγεθος για το RAS.

4.5 Σύγκριση διαφορετικών predictors

Στο κομμάτι αυτό θα συγκρίνετε τους παρακάτω predictors (οι predictors σε **bold** *δε δίνονται* και πρέπει να τους υλοποιήσετε εσείς):

- **Static AlwaysTaken**
- **Static BTFNT (BackwardTaken-ForwardNotTaken)**
- Ο n-bit predictor που επιλέξατε στο 4.2 (ii)
- Pentium-M predictor (δίνεται ότι το hardware overhead είναι περίπου 30K)
- **Local-History two-level predictors** (βλ. διαφάνειες μαθήματος) με τα εξής χαρακτηριστικά :
  - PHT entries = 8192
  - PHT n-bit counter length = 2
  - BHT entries = X
  - BHT entry length = Z

  Υπολογίστε το Z ώστε το απαιτούμενο hardware να είναι σταθερό και ίσο με 32K, όταν X=2048 και X=4096
- **Global History two-level predictors** με τα εξής χαρακτηριστικά:
  - PHT entries = Z
  - PHT n-bit counter length = X
  - BHR length = 5, 10

  Υπολογίστε το Z ώστε το απαιτούμενο hardware να είναι σταθερό και ίσο με 32K όταν X=2 και X = 4. Το κόστος του Branch History Register (5 και 10 bits) θεωρείται αμελητέο
- **Alpha 21264 predictor** (βλ. διαφάνειες μαθήματος – hardware overhead 29K)
- **Tournament Hybrid predictors** (βλ. διαφάνειες μαθήματος) με τα εξής χαρακτηριστικά:
  - Ο meta-predictor M είναι ένας 2-bit predictor με 1024 ή 2048 entries (το overhead του μπορείτε να το αγνοήσετε στην ανάλυση σας)
  - Οι P_0, P_1 μπορούν να είναι n-bit, local-history ή global-history predictors
  - Οι P_0, P_1 έχουν overhead 16K ο καθένας
  - Υλοποιήστε *τουλάχιστον* 4 διαφορετικούς tournament predictors

Προσομοιώστε για τα benchmarks που παρέχονται και συγκρίνετε τους παραπάνω (*τουλάχιστον 15*) predictors. Δώστε τα κατάλληλα διαγράμματα. Ποιον predictor θα διαλέγατε τελικά να υλοποιήσετε στο σύστημα σας;

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!