Simulating Branch Prediction with PIN

Χαράλαμπος Παπαδόπουλος
03120199

Απρίλιος 2025

1 Εισαγωγή

Ζητούμενο της άσκησης είναι η κατασκευή και σε συνέχεια η σύγκριση διάφορων branch predictors χρησιμοποιώντας το εργαλείο προσομοίωσης PIN.

2 Ανάλυση εντολών άλματος

Αρχικά, συλλέγουμε δεδομένα σχετικά με τα benchmarks που θα χρησιμοποιήσουμε (τόσο για τα train όσο και για τα ref) χρησιμοποιώντας το pintool cslab_branch_stats.so. Λαμβάνουμε, λοιπόν τα εξής αποτελέσματα:

ΠΟΛΛΑ ΔΙΑΓΡΑΜΜΑΤΑ

3 N-bit predictors

Σε αυτό το ερώτημα πλέον ξεκινάμε την ανάλυση κάποιων branch predictors.

3.1

Διατηρώντας σταθερό τον αριθμό των BHT entries και ίσο με 16K, προσομοιώνουμε τους n-bit predictors, για N = 1, 2, 3, 4. Τα n-bits υλοποιούν ένα saturating up-down counter (cslab_branch.cpp) όπως είδαμε στις διαλέξεις.

Τροποιούμε κατάλληλα τον βοηθητικό κώδικα προσθέτοντας τον εξής κώδικα

```
for (int i=1; i <= 4; i++) {
    NbitPredictor *nbitPred = new NbitPredictor(14, i);
    branch_predictors.push_back(nbitPred);
}
```

Οι παράμετροι που δίνουμε στο object `NbitPredictor` είναι:

- `index_bits = 14` : Ορίζει το πλήθος των bits που χρησιμοποιούνται για την κατασκευή του πίνακα προβλέψεων. Δηλαδή, το μέγεθος του πίνακα θα είναι 2^{14} καταχωρήσεις.

- `cntr_bits` : Ορίζει το πλήθος των bits κάθε μετρητή στον πίνακα. Όσο περισσότερα bits, τόσο περισσότερες καταστάσεις μπορεί να περιγράψει ένας μετρητής.

Καταλήγουμε, λοιπόν, με τα εξής διαγραμμάτα:
ΠΟΛΛΑ ΔΙΑΓΡΑΜΜΑΤΑ
ΣΥΜΠΕΡΑΣΜΑΤΑ

3.2

Από το paper *"Optimal 2-Bit Branch Predictors"* (R. Nair, 1995) βλέπουμε τα εξής πιθανά FSM:

TABLE VI
BEST FIVE MACHINES IGNORING STARTING STATE

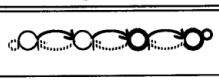
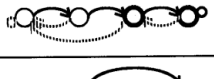
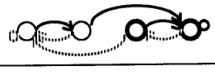
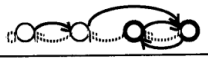
Rank	State diagram for machine
1.	
2.	
3.	
4.	
5.	

Figure 1: FSMs according to R. Nair.

όπου με **bold** απεικονίζονται οι μεταβάσεις μετά από σωστή πρόβλεψη και `dotted` οι μεταβάσεις ύστερα από αποτυχημένη.

Βασιζόμενοι σε αυτά τα διαγράμματα δημιουργούμε τους αντίστοιχους predictors στο αρχείο branch_predictor.h:

```
class TwobitPredictor_FSM2 : public BranchPredictor
{
public:
    TwobitPredictor_FSM2(unsigned index_bits_ = 14, unsigned cntr_bits_
        = 2)
    \\same as before ...

    virtual void update(bool predicted, bool actual, ADDRINT ip,
        ADDRINT target) {
        unsigned int ip_table_index = ip % table_entries;
        if (actual) {
            if (TABLE[ip_table_index] < COUNTER_MAX)
                TABLE[ip_table_index]++;
        } else {
            if (TABLE[ip_table_index] == 2)
                TABLE[ip_table_index] -= 2;
            else if (TABLE[ip_table_index] > 0)
```

```
                TABLE[ip_table_index]--;
        }

        updateCounters(predicted, actual);
    };

    virtual string getName() {
        std::ostringstream stream;
        stream << "FSM2_" << pow(2.0,double(index_bits)) / 1024.0 << "K
            -" << cntr_bits;
        return stream.str();
    }
```

Κατά αντίστοιχο τρόπο κατασκευάζουμε και τις υπόλοιπες κλάσεις τις οποίες μετά καλούμε στο cslab_branch.cpp

```
new TwobitPredictor_FSM1();
new TwobitPredictor_FSM2();
new TwobitPredictor_FSM3();
new TwobitPredictor_FSM4();
new TwobitPredictor_FSM5();
```

ΔΙΑΓΡΑΜΜΑΤΑ
ΣΥΜΠΕΡΑΣΜΑΤΑ

3.3

Προκειμένου να ορίσουμε το hardware ως 32K χρειάζεται να καλέσουμε τους constructor των κλάσεων με index_bits * cntr_bits = 32K. Οπότε, παίρνουμε τα ζεύγη (15,1), (14,2), (13,3).

```
    // 32K hardware
    new TwobitPredictor_FSM1();
    new TwobitPredictor_FSM2();
    new TwobitPredictor_FSM3();
    new TwobitPredictor_FSM4();
    new TwobitPredictor_FSM5();

    new NbitPredictor(15, 1);
    new NbitPredictor(13, 4);
```

ΔΙΑΓΡΑΜΜΑΤΑ
ΣΥΜΠΕΡΑΣΜΑΤΑ

4 Μελέτη του BTB