8ο Εργαστήριο Αρχιτεκτονικής Η/Υ: Προσομοιωτής κρυφής μνήμης

Α. Ευθυμίου

Παραδοτέο: Παρασκευή 22 Μάη 2015, 23:00

Ο σκοπός αυτής της άσκησης είναι η κατανόηση της οργάνωσης και λειτουργίας κρυφών μνημών καθώς και της σημασίας που έχουν οι βασικές παράμετροί τους στις επιδόσεις τους.

Θα γράψετε ένα πρόγραμμα σε Java που προσομοιώνει κρυφές μνήμες και παρουσιάζει δεδομένα επίδοσης σχετικά με τον αριθμό αστοχιών τους. Ο προσομοιωτής δέχεται ως παραμέτρους το μέγεθος γραμμής, τη συσχετιστικότητα (associativity), τη συνολική χωρητικότητα μιας κρυφής μνήμης και την πολιτική εγγραφής write-allocate ή write-no-allocate. Μετά διαβάζει από ένα αρχείο έναν αριθμό προσπελάσεων μνήμης (είδος προσπέλασης και διευθύνση) και προσομοιώνει την λειτουργία της κρυφής μνήμης. Στο τέλος της προσομοίωσης, το πρόγραμμα εμφανίζει πληροφορίες όπως ο αριθμός ευστοχιών, αστοχιών ξεχωριστά για αναγνώσεις και εγγραφές καθώς και το συνολικό ποσοστό αστοχίας.

Θα πρέπει να έχετε μελετήσει τα μαθήματα σχετικά με κρυφές μνήμες που αντιστοιχούν στις ενότητες 5.1 - 5.3 και 5.5 του συγγράμματος. Για την υλοποίηση του προγράμματος θα χρησιμοποιήσετε απλές δομές δεδομένων, όπως πίνακες και λίστες.

1 Σκελετός άσκησης και τρόπος εκτέλεσης

Για να ξεκινήσετε θα χρειαστείτε τα αρχεία του εργαστηρίου δίνοντας τις εξής εντολές:

```
git remote add lab08_starter https://github.com/UoI-CSE-MYY402/lab08_starter.git
git fetch lab08_starter
git merge lab08_starter/master -m "Fetched lab08 starter files"
```

Στο αρχείο Lab08.java θα βρείτε το σκελετό του προγράμματος που μεταγλωτίζεται και τρέχει, αλλά δεν εκτελεί σωστά την προσομοίωση.

Μετά τη μεταγλώτιση, το πρόγραμμα τρέχει με την εντολή: `java Lab08 8 1 4096 A traceFile`. Προσέξτε ότι υπάρχει ένα επιπλέον όρισμα σε σχέση με την προηγούμενη άσκηση.

Ο πρώτος αριθμός της παραπάνω εντολής είναι το μέγεθος της γραμμής κρυφής μνήμης σε bytes. Ο επόμενος αριθμός είναι η συσχετιστικότητα (associativy ή number of ways) και ο τρίτος αριθμός είναι η συνολική χωρητικότητα (δεδομένων) της κρυφής μνήμης σε bytes. Το επόμενο όρισμα είναι ένα χαρακτηριστικό της «πολιτικής έγγραφής»: καθορίζει αν σε μία εγγραφή που αστοχεί, θα προσκομίζεται η γραμμή από τη κύρια μνήμη και μετά θα γίνεται η εγγραφή (τιμή ορίσματος: A, write-allocate) ή αν η άστοχη εγγραφή απλά γίνεται στη κύρια μνήμη (τιμή ορίσματος: N, write-no-allocate). Η κρυφή μνήμη είναι write-through και ο αλγόριθμος αντικατάστασης είναι LRU - least recently used.

Το τελευταίο όρισμα στην εντολή (traceFile) είναι ένα όνομα αρχείου, που περιέχει τις προσπελάσεις μνήμης ενός προγράμματος, μία σε κάθε γραμμή: το γράμμα R ή W (για ανάγνωση ή εγγραφή αντίστοιχα) ακολουθούμενο από μία διεύθύνση των 48bit. Αυτού του είδους αρχεία ονομάζονται traces.

Επειδή τα αρχεία αυτά είναι αρκετά μεγάλα, αντί για το gitHub έχουν έχουν αναρτηθεί στο `http://www.cs.uoi.gr/~myy402/traces.tar.gz`. Θα τα βρείτε επίσης στον κατάλογο `~myy402/lab08/` στους υπολογιστές του εργαστηρίου. Κατεβάστε τα στον υπολογιστή σας αλλά **μην τα προσθέσετε σε στιγμιότυπα (commits) στο gitHub** γιατί ο χώρος που θα καταλάβουν στο GitHub πολλαπλασιάζεται με τον αριθμό των φοιτητών του μαθήματος!

2 Τεχνική προσομοίωσης κρυφής μνήμης

Επειδή ο σκοπός της προσομοίωσης είναι η μέτρηση γεγονότων (events) όπως, για παράδειγμα, αριθμός αστοχιών μνήμης, δεν μας ενδιαφέρουν οι τιμές των δεδομένων στη μνήμη. Γι'αυτό το λόγο οι προσπελάσεις εγγραφής στο αρχείο δεν δίνουν τα δεδομένα που εγγράφονται, ούτε δίνονται οι αρχικές τιμές

δεδομένων στη μνήμη. Το μόνο που χρειάζεται είναι η πληροφορία των αποθηκευμένων tags και τα valid bits για όλες τις γραμμές τής κρυφής μνήμης.

Επίσης, επειδή ο προσομοιωτής δεν έχει σκοπό να περιγράψει ακριβώς την λειτουργία του υλικού, μπορούν να χρησιμοποιηθούν δομές δεδομένων που δεν είναι πρακτικά υλοποιήσιμες σε υλικό. Για παράδειγμα, για την υλοποίηση του αλγόριθμου LRU μεταξύ των γραμμών ενός σετ, μπορεί να χρησιμοποιήθεί μια συνδεδεμένη λίστα όπου το πρώτο στοιχείο της είναι η πιο πρόσφατα χρησιμοποιημένη γραμμή (και το τελευταίο το λιγότερο πρόσφατο).

Θα πρέπει να επιλέξετε κατάλληλες δομές δεδομένων που να μπορούν να χρησιμοποιηθούν είτε στην περίπτωση direct-mapped οργάνωσης είτε σε fully-associative (και φυσικά σε ενδιάμεσες way-assocative οργανώσεις). Η βασική δομή είναι ένας πίνακας από cache set. Το κάθε set θα περιέχει έναν αριθμό από «γραμμές», μόνο που αντί για δεδομένα θα περιέχουν μόνο τα tags και τα valid bits. Το πώς θα οργανώσετε τις γραμμές μέσα σε set, είναι δική σας επιλογή. Παραπάνω έγινε μια νύξη για ένα πιθανό τρόπο υλοποίησης.

3 Διαδικασία υλοποίησης του προσομοιωτή

Ανοίξτε το αρχείο Lab08.java και, αρχικά, προσθέστε τον δικό σας κώδικα από την προηγούμενη άσκηση στα κατάλληλα σημεία. Υπάρχουν σχόλια στον κώδικα που σας καθοδηγούν. Προσοχή, μην ξεχάσετε κάποιο τμήμα από το (δικό σας) Lab07 και μην σβήσετε γραμμές που είναι διαφορετικές και χρειάζονται μόνο σε αυτή την άσκηση!

Μετά περιηγηθείτε στον υπάρχοντα κώδικα. Η κλάση Lab08 περιέχει τη μέθοδο main που δημιουργεί ένα αντικείμενο της κλάσης Cache. Μετά διαβάζει το αρχείο και για κάθε προσπέλαση καλεί την μέθοδο `access`. Στο τέλος της προσομοίωσης καλεί την `report` που εμφανίζει τους αριθμούς προσπελάσεων, ευστοχιών κλπ. Δεν χρειάζεται να αλλάξετε τον κώδικα στην κλάση Lab08, ούτε τη μέθοδο report.

Όλες οι αλλαγές θα γίνουν στην κλάση Cache. Ο constructor είναι σε μεγάλο βαθμό ίδιος με αυτόν της προηγούμενης άσκησης. Στο τέλος του θα πρέπει να προσθέσετε τον δικό σας κώδικα από το Lab07, αλλά προσοχή μη σβήσετε την κλήση της μεθόδου `initCache`. Επίσης θα πρέπει να μεταφέρετε τον κώδικά σας για τις μεθόδους getTag, getIndex, getBoff καθώς και ό,τι δηλώσεις μεταβλητών είχατε (π.χ. για αριθμό σετ).

Η μέθοδος initCache έχει ως σκοπό την αρχικοποίηση δομών δεδομένων που θα απαιτηθούν.

Το κύριο βάρος της δουλειάς θα γίνεται στη μέθοδο access. Αρχικά, γράψτε ψευτοκώδικα που περιγράφει την προσπέλαση μιας κρυφής μνήμης. Σκεφτείτε όλες τις περιπτώσεις: η συσχετιστικότητα (associativity) μπορεί να είναι από 1 (direct-mapped) μέχρι και fully-associative. Οι εγγραφές, αν αστοχούν μπορεί να προκαλούν προσκόμιση γραμμών από την κύρια μνήμη, αν στη γραμμή εντολών ο χρήστης δώσει την επιλογή A. Αλλά πρέπει να υποστηρίζετε και την αντίθετη περίπτωση. Οι αστοχίες αναγνώσεων πάντα προκαλούν προσκόμιση γραμμής από την κύρια μνήμη. Επίσης σκεφτείτε πώς θα κρατάτε τη σειρά LRU κάθε σετ.

Μετά τον ψευτοκώδικα, σκεφτείτε ποιές είναι οι πιο κατάλληλες δομές δεδομένων για να κρατούν τα tags, valid bits ώστε να είναι και ευέλικτες ώστε να υποστηρίζουμε μεταβλητή associativity και αποδοτικές. Σίγουρα θα χρειαστείτε κάποιου είδους πίνακες ή λίστες, π.χ. ArrayList, LinkedList, Vector. Είστε ελεύθεροι να χρησιμοποιήσετε έτοιμες δομές όπως οι παραπάνω (generics στην Java, π.χ. ArrayList<CacheLine> όπου CacheLine μια δική σας κλάση) να προσθέσετε δικές σας κλάσεις κλπ. **Προσοχή. Μή ξεχάσετε να μετρήσετε τα διάφορα συμβάντα, π.χ. αριθμός ευστοχιών εγγραφής - writeHits.** Έχουν δηλωθεί στις γραμμες 29-34.

4 Πειραματισμός με τον προσομοιωτή και έλεγχος

Μπορείτε να κάνετε ένα βασικό έλεγχο της υλοποιησής σας κοιτώντας αν ο συνολικός αριθμός προσπελάσεων που αναφέρει στο τέλος είναι ίδιος με τον αριθμό γραμμών των αρχείων. Ο λεπτομερής έλεγχος είναι αρκετά δύσκολος γιατί για κάθε οργάνωση κρυφής μνήμης θα πρέπει να κατασκευάζεται ένα αρχείο προσπελάσεων που να ελέγχει τον αλγόριθμο αντικατάστασης, κλπ.

Τέλος τρέξτε μερικά πειράματα μεταβάλοντας τις βασικές παραμέτρους της κρυφής μνήμης. Παίρνει γύρω στα δύο λεπτά για κάθε προσομοίωση. Για παράδειγμα, κρατώντας το συνολικό μέγεθος σταθερό στα 4KB και το μέγεθος γραμμής στα 8 bytes, δοκιμάστε να μεταβάλετε την associativity, διπλασιάζοντάς την σε κάθε βήμα και παρατηρήστε πως μεταβάλεται το ποσοστό αστοχίας. Κάντε ανάλογα πειράματα για το μέγεθος γραμμής, συνολική χωρητικότητα και την πολιτική εγγραφής. Δοκιμάστε και τα 2 αρχεία με traces γιατί τα αποτελέσματα γενικά διαφέρουν ανάλογα με το προγραμμα.

Δεν υπάρχει κάποιο παραδοτέο σχετικό με τα παραπάνω πειράματα.

5 Παραδοτέο

Το παραδοτέο της άσκησης είναι το αρχείο `Lab08.java`. Η παράδοση θα γίνει μέσω GitHub, όπως πάντα.