**1ο μέρος εργασίας του μαθήματος**

**«Σχεδιασμός Ενσωματωμένων Συστημάτων»**

**Ομάδα 34**

**Μαρουλάκος-Σεφεριάδης Πάρης 57128**

**Μπαντήλας Παναγιώτης 57138**

**4. Loop Tiling**

**Θεωρία**

Ουσιαστικά αυτό που κάνουμε είναι να χωρίζουμε νοητά έναν πίνακα σε μικρότερους υποπίνακες (πλακάκια – tiles) και να τα επεξεργαζόμαστε κατά αυτόν τον τρόπο. Αυτό σαφώς αλλάζει και τον τρόπο με τον οποίο μεταβιβάζονται τα δεδομένα από την κύρια μνήμη στην μνήμη cache. Καθώς τα δεδομένα τα οποία χρησιμοποιούμε δεν ακολουθούν το συνηθισμένο μοτίβο προσπέλασης.

Υπάρχουν δύο τρόποι με τους οποίος μπορεί να επιτευχθεί το tiling. Οπτικά στον ένα τρόπο οι προσπελάσεις σχηματίζουν πολλά νοητά Ζ (δηλαδή ο πιο εσωτερικός εμφωλευμένος βρόγχος μετράει τις στήλες) και στον δεύτερο να σχηματίζουν πολλά αντίστροφα Ν (δηλαδή ο πιο εσωτερικός εμφωλευμένος βρόγχος μετράει τις γραμμές).

Όσον αφορά την πρώτη περίπτωση (ο πιο εσωτερικός εμφωλευμένος βρόγχος μετράει τις στήλες), χωρίσαμε τους δισδιάστατους πίνακες σε tiles διαστάσεων:

• 288 x 22

• 288 x 32

• 288 x 44

• 288 x 88

• 288 x 176

Όσον αφορά τη δεύτερη περίπτωση (ο πιο εσωτερικός εμφωλευμένος βρόγχος μετράει τις γραμμές), χωρίσαμε τους δισδιάστατους πίνακες σε tiles διαστάσεων:

• 24 x 352

• 32 x 352

• 36 x 352

• 72 x 352

• 96 x 352

• 144 x 352

Armulator:

1η Περίπτωση:

**Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα**

Παρατηρούμε ότι για κάθε ξεχωριστό tile του συγκεκριμένου τρόπου προσέγγισης προκύπτει ακριβώς ο ίδιος πίνακας δεδομένων. Επίσης αυξήθηκε λιγο το μέγεθος του κώδικα και τα ZI Data.

2η Περίπτωση:**Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα**

Παρατηρούμε ότι για κάθε ξεχωριστό tile του συγκεκριμένου τρόπου προσέγγισης προκύπτει ακριβώς ο ίδιος πίνακας δεδομένων. Επίσης αυξήθηκε λίγο το μέγεθος του κώδικα και τα ZI Data (400 Bytes περίπου).

Debugger Internals Statistics:

1η Περίπτωση:

288x22:

Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

288x32:

Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

288x44:

Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

288x88:

Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

288x176:

Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

2η Περίπτωση:

24x352:

Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

32x352:

Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

36x352:

Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

72x352:

Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

96x352:

Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

144x352:

Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

Παρατηρούμε σε όλα αυξήθηκε ο συνολικός αριθμός των κύκλων. Η πρώτη περίπτωση έδωσε καλύτερα αποτελέσματα από τη δεύτερη.

Από την πρώτη περίπτωση την καλύτερη επίδοση είχε το tile 288 x 88, όπου ο ο συνολικός αριθμός των κύκλων αυξήθηκε ελαχιστα περίπου 0.01%

Από τη δεύτερη περίπτωση την καλύτερη επίδοση είχε το tile 72 x 352, όπου ο συνολικός αριθμός των κύκλων αυξήθηκε περίπου 1%

Επίσης, η αύξηση του μεγέθους του κώδικα είναι προφανής, γιατί στους βρόγχους που εφαρμόστηκε το loop tiling χρησιμοποιήθηκαν τρίς και όχι δύο εμφωλευμένες for.

Τα ΖΙ data αυξήθηκαν κατά 4 Bytes, επειδή προστέθηκαν στον κώδικα οι ακέραιες (int) μεταβλητές jj (στην πρώτη περίπτωση) και ii (στη δεύτερη περίπτωση) αντίστοιχα.

**5. Loop Collapsing**

Θεωρία:

Ορισμένοι εμφωλευμένοι βρόχοι μπορούν να συρρικνωθούν σε έναν ενιαίο βρόχο για να μειωθεί το loop overhead(επιβάρυνση βρόχου) και να βελτιωθεί η απόδοση χρόνου εκτέλεσης.

Το loop collapsing μπορεί να βελτιώσει τις ευκαιρίες για περαιτέρω βελτιστοποιήσεις, όπως το loop unrolling.

Armulator:

Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

Μιεώθηκε και ο κώδικας και τα ZI Data

Debugger Internals Statistics:

Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

Παρατηρουμε ότι οι συνολικοί κύκλοι μειώθηκαν αρκετά(περίπου στο 1/3)

Σημείωση για το loop collapsing:

Δυστυχώς είναι το μόνο κομμάτι των βελτιστοποιήσεων που εικόνα στο τέλος αν και πολύ κοντά στην αρχική έχε μεταβολές που δεν είμαστε σίγουροι σε τι οφείλονται και άρα δεν μπορούμε να βγάλουμε πολλά συμπεράσματα για αυτή την τεχνική.

Ωστόσο η μείωση του κώδικα μπορεί να αποδοθεί στην μείωση των instructions, αφού κάποιοι εμφωλευμένοι βρόχοι, οι οποίο χρησιμοποιούσαν δύο δείκτες, άρα κάναν διπλούς ελέγχους για το αν ο δείκτης έχει ξεπεράσει το όριο, αντικαταστάθηκαν από βρόχους με έναν δείκτη.

Επίσης, η μεγάλη μείωση των ZI data μπορεί να αποδοθεί στο γεγονός, ότι ο δισδιάστατος πίνακας current\_y αντιπροσωπεύεται από ένα integer δείκτη \*c.

**6. Loop Inversion**

Θεωρία:

Tο loop inversion αποτελεί βελτιστοποίηση και μετασχηματισμός του loop, στον οποίο, ένας βρόχος while αντικαθίσταται από ένα μπλοκ if που περιέχει ένα βρόχο do...while. Όταν χρησιμοποιείται σωστά, μπορεί να βελτιώσει την απόδοση λόγω του instruction pipelining. Οι σύγχρονες CPU χρησιμοποιούν instruction pipeline. Εκ φύσεως. κάθε άλμα στον κώδικα προκαλεί pipeline stall, πράγμα το οποίο είναι επιζήμιο για την απόδοση. Επιπρόσθετα, το loop inversion επιτρέπει ασφαλή κίνηση κώδικα με αμετάβλητο βρόχο.

Σημείωση για το loop inversion: Αρχικά υλοποιήσαμε τον κώδικά μας με λούπες while αντί για for. Στη συνέχεια αντικαταστήσαμε τις while με λούπες do...while εφαρμόζοντας την τεχνική loop inversion

Armulator:

Περίπτωση Do While:

Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

Περίπτωση While:

Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

Παρατηρούμε ότι στην περίπτωση του do While\* μειώθηκε ο συνολικός κώδικας κατά 30 bytes ενώ στην περίπτωση του while αυξήθηκε κατά 90 Bytes.Οι αλλαγές στο μέγεθος του κώδικα είναι πολύ μικρές και αυτό μπορεί να οφείλεται σε σημαντικό βαθμό σε ορισμένες μικροδιαφορές όσον αφορά την ανάπτυξη του εκάστοτε κώδικα.

\*Στην περίπτωση του do while δεν είμαστε σίγουροι αν έχει γίνει κάποιο λάθος γιατί ενώ στο dev++ τρέχει κανονικά ο κώδικας στο code warrior μας βγάζει error το οποίο επισυνάπτεται παρακάτω και φαίνεται και όταν το κάνουμε run:

Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

Debugger Internals Statistics:

Περίπτωση Do While:

Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

Περίπτωση While:

Εικόνα που περιέχει στιγμιότυπο οθόνης

Περιγραφή που δημιουργήθηκε αυτόματα

Παρατηρούμε ότι στην περίπτωση του While οι συνολικοί κύκλοι μειώνονται κατά 0.02%

Η διαφορά στον αριθμό των συνολικών κύκλων μπορεί να οφείλεται στο γεγονός ότι κάθε λούπα επανάληψης μεταφράζεται με διαφορετικό τρόπο σε assembly.

Βιβλιογραφία

Διαφάνειες εργαστηρίου από το μάθημα «Σχεδιασμός Ενσωματωμένων Συστημάτων» Σημειώσει από το μάθημα «Σχεδιασμός Ενσωματωμένων Συστημάτων» https://en.wikipedia.org/wiki/Loop\_unrolling https://en.wikipedia.org/wiki/Loop\_fission\_and\_fusion https://en.wikipedia.org/wiki/Loop\_interchange http://www.nullstone.com/htmls/category/collapse.htm https://en.wikipedia.org/wiki/Loop\_inversion