3Η ΕΡΓΑΣΙΑ ΣΤΟ ΜΑΘΗΜΑ ΣΧΕΔΙΑΣΜΟΣ ΕΝΣΩΜΑΤΩΜΕΝΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΟΜΑΔΑ 45

Γιώργος Καλιωράκης ΑΜ:57069

Κωνσταντίνος Μερτζάνης ΑΜ:57131

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ …… 2

1ος τρόπος - Lines ……..... 3

Πριν την βελτιστοποίηση(filter-lines)……..…….. 3

Μετά τη βελτιστοποίηση(unroll-lines)…….. 5

Παρατηρήσεις…….. 6

2ος τρόπος - Blocks……. 6

Πριν την βελτιστοποίηση(filter-block)…..…….. 7

Μετά την βελτιστοποίηση(unroll-block)……. 8

Παρατηρήσεις…… 9

3ος τρόπος – Combination…….. 9

Πριν την βελτιστοποίηση(filter-combination)…….. 10

Μετά την βελτιστοποίηση(unroll-combination)…….. 11

Παρατηρήσεις…….. 13

Συμπέρασμα…… 13

ΠΗΓΕΣ…….. 13

ΕΙΣΑΓΩΓΗ

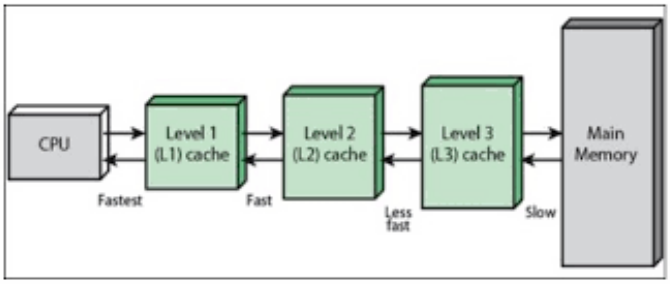
Σε αυτή την εργασία όπως και στο 2ο σκέλος της επικεντρωνόμαστε στην ανάλυση και χρήση διαφορετικών τεχνικών ιεραρχίας μνήμης, πάνω στη λογική βελτιστοποίησης της λειτουργίας του επεξεργαστή ως προς την επαναχρησιμοποίηση δεδομένων. Το θεωρητικό υπόβαθρο και η ανάλυση των περιοχών μνήμης που χρησιμοποιήθηκαν, συμβαδίζουν με τις γνώσεις και τις τεχνικές που υλοποιήθηκαν στο 2ο σκέλος της εργασίας μας.

Η λογική των μετασχηματισμών επαναχρησιμοποίησης δεδομένων, άπτονται στην εύρεση του βέλτιστου μηχανισμού που θα την υλοποιεί με αποτέλεσμα την ταχύτερη λειτουργία του επεξεργαστή μας. Όπως και στην προηγούμενη εργασία, αξίζει να αναφερθεί πως υπάρχει η SRAM (μνήμη cache) η οποία λόγω της κατασκευής της (πιο προηγμένη τεχνολογία από την DRAM επειδή διατηρεί τις τιμές των δεδομένων της χωρίς να χρειάζεται refresh), αλλά και λόγω της θέσης της πάνω στο ολοκληρωμένο κύκλωμα του επεξεργαστή που της παρέχει αμεσότερη «επικοινωνία» με τον CPU, αποθηκεύει και στέλνει δεδομένα σ αυτόν με μεγαλύτερη ταχύτητα από ότι οι άλλες μνήμες. Αυτό έχει ως επακόλουθο να εκμεταλλευτούμε αυτό το χαρακτηριστικό της, και να διαπεράσουμε μειονέκτημα της μικρής χωρητικότητάς της αποθηκεύοντας σε αυτήν «στοχευμένα» κομμάτια των δεδομένων που ο CPU θα απαιτείται να τα χρησιμοποιήσει ξανά και ξανά για την επεξεργασία της εικόνας μας.

Ο σκοπός μας λοιπόν, είναι να μειώσουμε τις περιττές προσπελάσεις στη μνήμη, εξασφαλίζοντας το βέλτιστο αποτέλεσμα. Πως υλοποιείται αυτό; εισάγοντας μικρότερου μεγέθους μνήμες διάφορων επιπέδων(cache 1, cache 2 , cache 3) για την προσωρινή αποθήκευση δεδομένων που μπορεί να αποθηκευτούν πιο κοντά στον επεξεργαστή. Στα επίπεδα αυτά τοποθετούνται τα η επαναχρησιμοποίηση των οποίων απαιτείται συχνά για την επεξεργασία της εικόνας μας. Με αυτό τον τρόπο η μεταφορά δεδομένων από τη μνήμη στον επεξεργαστή -και αντίστροφα- γίνεται στα επίπεδα πλησίον του επεξεργαστή. Έτσι η μεταφορά δεδομένων είναι πιο γρήγορη (αφού μειώνονται οι προσπελάσεις στην εξωτερική μνήμη).

Το 3ο και τελευταίο κομμάτι της εργασίας μας, χωρίζεται σε 3 διαφορετικά σκέλη. Στο 1ο σκέλος ακολουθώντας τη λογική του αντίστοιχου 3ο της προηγούμενης εργασίας χρησιμοποιήσαμε 3 buffers(1X354) με βηματική προσπέλαση των δεδομένων των πινάκων των εικονοστοιχείων μας. Στο 2ο σκέλος χρησιμοποιήσαμε ένα block (3X3) το οποίο βηματικά προσπέλαυνε τα εικονοστοιχεία του πίνακα. Και στο 3ο σκέλος χρησιμοποιήσαμε τον συνδυασμό των 2 μεθόδων. Συμβαδίζοντας με το αρχείο «Ασκήσεις Ενσωματωμένων Συστημάτων» υλοποιήσαμε τις τρεις αυτές τεχνικές επαναχρησιμοποίησης δεδομένων τόσο για τον αρχικό πίνακα φιλτραρίσματος της εικόνας μας (FILTER.C), όσο και για την βελτιστοποίηση βρόγχου που μας έδωσε τα βέλτιστα αποτελέσματα όσον αφορά τη λειτουργία του επεξεργαστή (UNROLL.C).

Για κάθε μέθοδο και κώδικα της εργασίας μας (εκτός της υλοποίησης unroll-lines, όπου έχουμε έτοιμα τα απαιτούμενα όρια μνήμης από την 2η εργασία) αρχικά υποθέσαμε μεγέθη ROM, DRAM, SRAM (και SRAM2 όπου χρειάστηκε) αρκετά μεγαλύτερα από τα απαιτούμενα, έτσι ώστε να κάνουμε την κατάλληλη υλοποίηση μνήμης «τόση μνήμη – όσα ακριβώς δεδομένα απαιτούνται» αφού θα βλέπαμε τα image component sizes για τις απαιτήσεις σε ROM, DRAM, SRAM.



1ος τρόπος - Lines

Σε αυτό το στάδιο θα γίνουν δύο δοκιμές μία στον αλγόριθμο πριν τη βελτιστοποίησή του και μια στον αλγόριθμο ύστερα από αυτήν. Σε αυτή τη περίπτωση θα χρησιμοποιήσουμε τρεις μονοδιάστατους buffer όπως ακριβώς και στο τελευταίο ερώτημα της προηγούμενης εργασίας. Η χρησιμότητα της διαδικασίας αυτής έγγυται στην αδυναμία υπολογιστικών συστημάτων να προσφέρουν μνήμες cache αρκετά μεγάλες, για να «χωρέσουν» το σύνολο των δεδομένων προς επεξεργασία. Παρόλα αυτά, στην δικιά μας περίπτωση λόγω απουσίας ενός τόσο μεγάλου όγκου δεδομένων, αναμένουμε να μην εμφανίσει κάποια ιδιαίτερη βελτίωση

```
int current_y[N][M];
int A[N+2][M+2];
int newA[N+2][M+2];
#pragma arm section zidata="sram"
int buffer1[M+2];
int buffer2[M+2];
int buffer3[M+2];
```

Πριν την βελτιστοποίηση(FILTER-LINES)

Παρακάτω φαίνονται τα αποτελέσματα τόσο από το make που προκύπτει απτό Armulator όσο και το memory map και scatter file που χρησιμοποιήσαμε για την δοκιμή αυτή. Τέλος και πιο σημαντικό από όλα είναι η διαδικασία του Debug το οποίο ουσιαστικά μας ενημερώνει για όλους τους κύκλους που απαιτούνται για την ολοκλήρωση της διαδικασίας.

Memory map

```
00000000 00080000 ROM 4 R 1/1 1/1
00080000 00800000 DRAM 4 RW 250/50 250/50
00880000 00008000 SRAM 4 RW 1/1 1/1
```

Scatter file

```
LOAD_ROM 0x0 0x00080000
{
        EXEC_ROM 0x0 0x00080000
        {
                *.o  (+RO)
        }
        DRAM 0x00080000 0x00800000
        {
                * (+ZI, +RW)
        }
        SRAM 0x00880000 0x00008000
        {
                *(sram)
        }
}
```

Make

```
Errors & Warnings
0   0   13   Errors and warnings for "filter-lines-bad.mcp"
==============================================================================
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1100         60          0    1231044       6300   Object Totals
     10128        314          0        300       4472   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11228        374          0    1231344      10772   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                11602 (  11.33kB)
    Total RW  Size(RW Data + ZI Data)           1231344 (1202.48kB)
    Total ROM Size(Code + RO Data + RW Data)      11602 (  11.33kB)
==============================================================================
```

Όπως παρατηρούμε αυτό το μέγεθος μνήμης δεν θα μας φέρει τα βέλτιστα αποτελέσματα λόγω της «παραπάνω από όσο χρειάζεται» χωρητικότητάς του, καθυστερώντας τον επεξεργαστή με προσπέλαση κενών θέσεων μνήμης όπως εξηγήσαμε και στην εργασία 2. Τα μεγέθη που χρειαζόμαστε είναι αυτά που παρατίθενται στο image component sizes.

Ας υπενθυμίσουμε, για οικονομία χώρου μιας και έχει εξηγηθεί λεπτομερώς, μόνο σε αυτό το παράδειγμα το πως προκύπτει η χωρητικότητα των μνημών για το memory map και το scatter file.

Τα Total RO Data μας δίνουν την συνολική απαίτηση μνήμης ROM που δεσμέυεται κατά την φόρτωση και την εκτέλεση του προγράμματός μας. Στην προκειμένη 11602 Bytes. Υπενθυμίζουμε πως το μέγεθος μνήμης πρέπει να συμβαδίζει(να διαιρείται τέλεια) με το εύρος διαύλου που έχουμε ορίσει- δηλαδή τα 4 Bytes. Επομένως 11602 → 11604 Bytes = $(2D54)_{16}$

Τα Total RW Data μας δίνουν την συνολική απαίτηση μνήμης DRAM και SRAM που χρειαζόμαστε κατά την εκτέλεση του προγράμματός μας. Στην προκειμένη 1231344 bytes. Από τη στιγμή που στην SRAM θα στοιβάξουμε τους 3 buffers, γνωρίζουμε ακριβώς τις απαιτήσεις σε SRAM(3x3x354 = 4248bytes) →$(1098)_{16}$. Με μια απλή αφαίρεση προκύπτει πως οι απαιτήσεις σε DRAM είναι 1227096 $(12B958)_{16}$.

Τέλος το όριο της διεύθυνσης από το οποίο ξεκινάει η SRAM προκύπτει με μια απλή πρόσθεση των χωρητικωτήτων της ROM,DRAM →1238700 = $(12E6AC)_{16}$

Memory map

```
1    00000000 00002D54 ROM 4 R 1/1 1/1
2    00002D54 0012B958 DRAM 4 RW 250/50 250/50
3    0012E6AC 00001098 SRAM 4 RW 1/1 1/1
```

Scatter file

```
LOAD_ROM 0x0 0x00002D54
{
        EXEC_ROM 0x0 0x00002D54
        {
                *.o  (+RO)
        }
        DRAM 0x00002D54 0x0012B958
        {
                * (+ZI, +RW)
        }
        SRAM 0x0012E6AC 0x00001098
        {
                *(sram)
        }
}
```

Debug

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
$statistics	12349166	23775092	11537690	9384928	3080003	0	39103530	63106151	23814

Μετά τη βελτιστοποίηση(unroll-lines)

Παρακάτω φαίνονται τα αποτελέσματα τόσο από το make που προκύπτει απτό Armulator όσο και το memory map και scatter file που χρησιμοποιήσαμε για την δοκιμή αυτή. Τέλος και πιο σημαντικό από όλα είναι η διαδικασία του Debug το οποίο ουσιαστικά μας ενημερώνει για όλους τους κύκλους που απαιτούνται για την ολοκλήρωση της διαδικασίας. Εδώ αναμένουμε να έχουμε μείωση των κύκλων του επεξεργαστή λόγω της χρήσης της βελτιστοποίησης βρόγχου.

Memory map

```
00000000 00002E6C ROM 4 R 1/1 1/1
00002E6C 0012B958 DRAM 4 RW 250/50 250/50
0012E7C4 00001098 SRAM 4 RW 1/1 1/1
```

Scatter file

```
LOAD_ROM 0x0 0x00002E6C
{
        EXEC_ROM 0x0 0x00002E6C
        {
                *.o  (+RO)
        }
        DRAM 0x00002E6C 0x0012B958
        {
                * (+ZI, +RW)
        }
        SRAM 0x0012E7C4 0x00001098
        {
                *(sram)
        }
}
```

Make

```
==============================================================================
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1380         60          0    1231044       6328   Object Totals
     10128        314          0        300       4472   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11508        374          0    1231344      10800   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                11882 (  11.60kB)
    Total RW  Size(RW Data + ZI Data)           1231344 (1202.48kB)
    Total ROM Size(Code + RO Data + RW Data)      11882 (  11.60kB)
==============================================================================
```

Debug

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
$statistics	10541528	20516726	9831428	8163694	2749133	0	32386122	53130377	23814

<u>Παρατηρήσεις</u>

Παρατηρούμε πως από το debug του αρχικού κώδικα και το debug του κώδικα με τη βελτιστοποίηση βρόγχου πως οι συνολικοί κύκλοι έχουν μειωθεί, αυτό είναι και το αναμενόμενο αποτέλεσμα καθώς με τη μέθοδο του unrolling διασπώντας τις μεγάλες σε αριθμό επαναλήψεις σε αρκετά λιγότερες μειώνεται ο αριθμός των ελέγχων που απαιτείται.

2ος τρόπος - Blocks

Σε αυτό το στάδιο όπως είναι φανερό αν ρίξουμε μία γρήγορη ματιά στον κώδικα αντιλαμβανόμαστε πως για τις δοκιμές που θα κάνουμε στην θέση όπου στον προηγούμενο τρόπο υπήρχαν τρεις πίνακες με 1 γραμμή και 354 στήλες αυτή τη στιγμή υπάρχει ένας πίνακας block με 3 γραμμές και 3 στήλες. Αυτό που συμβαίνει σε αυτή τη διαδικασία διαφορετικά από την προηγούμενη είναι πως ο πίνακας αυτός(ο οποίος είναι εσκεμμένα στο ίδιο μέγεθος με το φίλτρο που χρησιμοποιούμε) κάνει στην αρχή επεξεργασία κατά στήλη, μεταβαίνοντας κάθε φορά σε μία καινούρια στήλη(τριών νέων στοιχείων) και μόλις ολοκληρώσει την επεξεργασία των στηλών «κατεβαίνει» μια γραμμή και επαναλαμβάνει την ίδια διαδικασία. Η μνήμη στην οποία προσωρινά αποθηκεύονται στην προκειμένη περίπτωση τα στοιχεία του πίνακα block είναι αρκετά μικρότερη από αυτήν που χρησιμοποιήσαμε στο προηγούμενο ερώτημα. Συνεπώς ο συγκεκριμένος τρόπος είναι χρήσιμος αν έχουμε μεγάλο περιορισμό στην μνήμη SRAM. Στην δικιά μας περίπτωση αναμένουμε αύξηση των κύκλων του επεξεργαστή, καθώς δεν έχουμε τόσο ισχυρό περιορισμό όσον αφορά την cache και «αναγκάζουμε» το πρόγραμμα να κάνει αναίτιες - περιττές προσπελάσεις στη μνήμη.

```
int current_y[N][M];
int A[N+2][M+2];
int newA[N+2][M+2];
#pragma arm section zidata="sram"
int block[3][3];

#pragma arm section
```

<u>Πριν την βελτιστοποίηση(filter-block)</u>

Παρακάτω φαίνονται τα αποτελέσματα τόσο από το make που προκύπτει απτό Armulator όσο και το memory map και scatter file που χρησιμοποιήσαμε για την δοκιμή αυτή. Τέλος και πιο σημαντικό από όλα είναι η διαδικασία του Debug το οποίο ουσιαστικά μας ενημερώνει για όλους τους κύκλους που απαιτούνται για την ολοκλήρωση της διαδικασίας.

Το αρχικό make με την αχρείαστα μεγάλη μνήμη, μας δίνει τις απαιτήσεις του συστήματος για την επεξεργασία της εικόνας μας με την μέθοδο block . Τα image component sizes μας δίνουν τις ακόλουθες πληροφορίες

Make

```
0    0    13    Errors and warnings for "FILTER-BLOCK-GOOOOD.mcp"
==========================================================================
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1056         60          0    1226832       6264   Object Totals
     10128        314          0        300       4472   Library Totals
==========================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11184        374          0    1227132      10736   Grand Totals
==========================================================================

    Total RO  Size(Code + RO Data)                11558 (  11.29kB)
    Total RW  Size(RW Data + ZI Data)           1227132 (1198.37kB)
    Total ROM Size(Code + RO Data + RW Data)      11558 (  11.29kB)
==========================================================================
```

Επομένως προσαρμόζουμε με τον τρόπο που είδαμε στο πρώτο ερώτημα τα μεγέθη της εκάστοτε μνήμης έτσι ώστε τα δεδομένα να χωράνε ακριβώς.

Memory map

```
1    00000000 00002D8C ROM 4 R 1/1 1/1
2    00002D8C 0012B958 DRAM 4 RW 250/50 250/50
3    0012E6E4 00000024 SRAM 4 RW 1/1 1/1
```

Scatter file

```
LOAD_ROM 0x0 0x00002D8C
{
        EXEC_ROM 0x0 0x00002D8C
        {
                *.o  (+RO)
        }
        DRAM 0x00002D8C 0x0012B958
        {
                * (+ZI, +RW)
        }
        SRAM 0x0012E6E4 0x00000024
        {
                *(sram)
        }
}
```

Debug

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
$statistics	16874493	31189654	15382177	12181116	4192892	0	46844898	78601083	59711

<u>Μετά τη βελτιστοποίηση(unroll-block)</u>

Παρακάτω φαίνονται τα αποτελέσματα τόσο από το make που προκύπτει απτό Armulator όσο και το memory map και scatter file που χρησιμοποιήσαμε για την δοκιμή αυτή. Τέλος και πιο σημαντικό από όλα είναι η διαδικασία του Debug το οποίο ουσιαστικά μας ενημερώνει για όλους τους κύκλους που απαιτούνται για την ολοκλήρωση της διαδικασίας.

Make

```
0   0   13   Errors and warnings for "UNROLL-BLOCK-GOOD.mcp"
==============================================================================
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1584         60          0    1226832       6388   Object Totals
     10128        314          0        300       4472   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11712        374          0    1227132      10860   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                12086 (  11.80kB)
    Total RW  Size(RW Data + ZI Data)           1227132 (1198.37kB)
    Total ROM Size(Code + RO Data + RW Data)      12086 (  11.80kB)
==============================================================================
```

Memory map

memory.map

```
1   00000000 00002F38 ROM 4 R 1/1 1/1
2   00002F38 0012B958 DRAM 4 RW 250/50 250/50
3   0012E890 00000024 SRAM 4 RW 1/1 1/1
```

Scatter file

```
LOAD_ROM 0x0 0x00002F38
{
        EXEC_ROM 0x0 0x00002F38
        {
                *.o  (+RO)
        }
        DRAM 0x00002F38 0x0012B958
        {
                * (+ZI, +RW)
        }
        SRAM 0x0012E890 0x00000024
        {
                *(sram)
        }
}
```

Debug

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
$statistics	15349341	28350922	13865097	11138856	3910346	0	42637434	71551733	59277

<u>Παρατηρήσεις</u>

Στην περίπτωση που χρησιμοποιούμε block 3x3 παρατηρούμε πως οι κύκλοι έχουν αυξηθεί εμφανώς. Αυτό συμβαίνει επειδή τα δεδομένα που αποθηκεύονται στην μνήμη είναι λιγότερα και συνεπώς η συνεχής προσπέλασή τους μέσω των σε μεγάλο επαναλήψεων προκαλεί την αύξηση των κύκλων. Για τον ίδιο λόγο με την προηγούμενη περίπτωση παρατηρούμε πως οι κύκλοι στον κώδικα με τη μέθοδο unroll έχουν μειωθεί σε σχέση με τον αρχικό κώδικα.

Ταυτόχρονα σε όλους τους κώδικες που δεν έχουν την μέθοδο unroll παρατηρούμε μειωμένο μέγεθος στα RO sizes σε σχέση με αυτους που έχουν και αυτό οφείλεται στο ότι κώδικες που έχουν τη μέθοδο unroll έχουν αρκετές σειρές κώδικα παραπάνω λόγω της ξεδίπλωσης των επαναλήψεων.

3ος τρόπος - Combination

Τελευταίος τρόπος είναι ο συνδυασμός και των δύο παραπάνω. Σε αυτή την περίπτωση έχουμε τους 3 buffers της πρώτου τρόπου στους οποίους αποθηκεύονται οι 3 πρώτες γραμμές ολόκληρου του πίνακα A. Στην συνέχεια χρησιμοποιούμε το 3x3 block του προηγούμενου τρόπου για να κάνουμε γρήγορη προσπέλαση στα στοιχεία των πρώτων τριών γραμμών του πίνακα κ.ο.κ . Ουσιαστικά είναι σαν να χρησιμοποιούμε συνδυαστικά μια γρήγορη μνήμη προσωρινής αποθήκευσης των buffers και μία ακόμη γρηγορότερη και μικρότερη μνήμη στην οποία αποθηκεύεται το 3x3 block.

```
int current_y[N][M];
int A[N+2][M+2];
int newA[N+2][M+2];
#pragma arm section zidata="sram1"
int block[3][3];
#pragma arm section

#pragma arm section zidata="sram2"
int buffer1[M+2];
int buffer2[M+2];
int buffer3[M+2];
#pragma arm section
```

<u>Πριν την βελτιστοποίηση(filter-combination)</u>

Παρακάτω φαίνονται τα αποτελέσματα τόσο από το make που προκύπτει απτό Armulator όσο και το memory map και scatter file που χρησιμοποιήσαμε για την δοκιμή αυτή. Τέλος και πιο σημαντικό από όλα είναι η διαδικασία του Debug το οποίο ουσιαστικά μας ενημερώνει για όλους τους κύκλους που απαιτούνται για την ολοκλήρωση της διαδικασίας.

Make

```
0    0    13    Errors and warnings for "OULALA.mcp"
==============================================================================
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1048         80          0    1231080       6196   Object Totals
     10128        314          0        300       4472   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11176        394          0    1231380      10668   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                11570 (  11.30kB)
    Total RW  Size(RW Data + ZI Data)           1231380 (1202.52kB)
    Total ROM Size(Code + RO Data + RW Data)      11570 (  11.30kB)
==============================================================================
```

Memory map

```
00000000 00002D34 ROM 4 R 1/1 1/1
00002D34 0012B958 DRAM 4 RW 250/50 250/50
0012E68C 00000024 SRAM1 4 RW 1/1 1/1
0012E6B0 00001098 SRAM2 4 RW 1/1 1/1
```

Scatter file

```
LOAD_ROM 0x0 0x00002D34
{
        EXEC_ROM 0x0 0x00002D34
        {
                *.o  (+RO)
        }
        DRAM 0x00002D34 0x0012B958
        {
                * (+ZI, +RW)
        }
        SRAM1 0x0012E68C 0x00000024
        {
                *(sram1)
        }
        SRAM2 0x0012E6B0 0x00001098
        {
                *(sram2)
        }
}
```

Debug

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
$statistics	17085454	32269385	14918344	13576743	4001186	0	48177150	80673423	23748

<u>Μετά τη βελτιστοποίηση(αλγόριθμος με unroll)</u>

Παρακάτω φαίνονται τα αποτελέσματα τόσο από το make που προκύπτει απτό Armulator όσο και το memory map και scatter file που χρησιμοποιήσαμε για την δοκιμή αυτή. Τέλος και πιο σημαντικό από όλα είναι η διαδικασία του Debug το οποίο ουσιαστικά μας ενημερώνει για όλους τους κύκλους που απαιτούνται για την ολοκλήρωση της διαδικασίας

.

Make

```
==============================================================================

Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1784         80          0    1231080       6552   Object Totals
     10128        314          0        300       4472   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11912        394          0    1231380      11024   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                12306 (  12.02kB)
    Total RW  Size(RW Data + ZI Data)           1231380 (1202.52kB)
    Total ROM Size(Code + RO Data + RW Data)      12306 (  12.02kB)
==============================================================================
```

Memory map

memory.map

```
00000000 00003014 ROM 4 R 1/1 1/1
00003014 0012B958 DRAM 4 RW 250/50 250/50
0012E96C 00000024 SRAM1 4 RW 1/1 1/1
0012E990 00001098 SRAM2 4 RW 1/1 1/1
```

Scatter file

```
LOAD_ROM 0x0 0x00003014
{
        EXEC_ROM 0x0 0x00003014
        {
                *.o  (+RO)
        }
        DRAM 0x00003014 0x0012B958
        {
                * (+ZI, +RW)
        }
        SRAM1 0x0012E96C 0x00000024
        {
                *(sram1)
        }
        SRAM2 0x0012E990 0x00001098
        {
                *(sram2)
        }
}
```

Debug

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
$statistics	17915024	35465880	15201024	15461469	5030596	0	46283514	81976603	23781

Παρατηρήσεις

Η αύξηση των κύκλων είναι αναμενόμενη καθώς χρησιμοποιούμε δύο μνήμες προσωρινής αποθήκευσης. Έτσι ο αριθμός επαναλήψεων τόσο για την αποθήκευση των τιμών στους 3 buffers της μεθόδου lines όσο και αυτός της αποθήκευσης ενός μέρους των τιμών των buffers στο block 3x3 προφανώς αυξάνεται.

Συμπέρασμα

Παρατηρούμε πως οι λιγότεροι συνολικοί κύκλοι επιτυγχάνονται στην περίπτωση των 3 buffer 1x354 με τον κώδικα να έχει υποστεί βελτιστοποίηση βρόγχου(unroll – line). Στις άλλες 2 περιπτώσεις όπως έχει ήδη αναφερθεί, κάνουμε περιττές προσπελάσεις στη μνήμη με αποτέλεσμα να αυξάνονται οι κύκλοι του επεξεργαστή. Αυτό είναι λογικό καθώς η περίπτωση του μπλοκ θα μας βοηθούσε μόνο αν είχαμε έναν πολύ σημαντικό περιορισμό στη μνήμη cache.

ΠΗΓΕΣ

- Ασκήσεις Ενσωματωμένων Συστημάτων.pdf
- Πηγές εργασίας 2