

Σχεδιασμός Ενσωματωμένων Συστημάτων

Εργασία 2^η

Μαζαράκης Περικλής Α.Μ.: 57595

Παπαδόπουλος Αριστείδης Α.Μ.: 57576

Γενικά:

Σε ορισμένες for-loops, που είχαν εμφωλευμένες μερικές switch-cases, ορίστηκε η αρχή της loop να ξεκινάει από το επόμενο νούμερο και να τερματίζει στο προτελευταίο, προκειμένου να μην γίνεται έλεγχος των ορίων της εικόνας. Η βελτιστοποίηση αυτή αφορά περιπτώσεις όπου υπολογίζεται γινόμενο συνέλιξης. Στην προηγούμενη εργασία θεωρήσαμε ιδανική μνήμη και βέλτιστους χρόνους ανάγνωσης/εγγραφής/μεταφοράς δεδομένων. Όμως, με την τοποθέτηση μνήμης, αναμένουμε σημαντική αύξηση των κύκλων ρολογιού καθώς υπάρχει αυτό το overhead χρόνου.

```
void gauss_filter(){
    printf("Create Gaussian Image...\n");
    for (i = 0; i < N; i++)
    {
        switch (i) {
            default:{
                for (j = 0; j < M; j=j+3)
                { switch (j) {
                    default:{ //convolve gauss filter in grayscale image, grayscale is the Y Channel
                        gauss_img[i][j] = current_y[i-1][j-1] * gauss[0][0] + current_y[i-1][j] * gauss[0][1] + current_y[i-1][j+1] * gauss[0][2]
                        + current_y[i][j-1] * gauss[1][0] + current_y[i][j] * gauss[1][1] + current_y[i][j+1] * gauss[1][2]
                        + current_y[i+1][j-1] * gauss[2][0] + current_y[i+1][j] * gauss[2][1] + current_y[i+1][j+1] * gauss[2][2];
                        gauss_img[i][j+1] = current_y[i-1][j] * gauss[0][0] + current_y[i-1][j+1] * gauss[0][1] + current_y[i-1][j+2] * gauss[0][2]
                        + current_y[i][j] * gauss[1][0] + current_y[i][j+1] * gauss[1][1] + current_y[i][j+2] * gauss[1][2]
                        + current_y[i+1][j] * gauss[2][0] + current_y[i+1][j+1] * gauss[2][1] + current_y[i+1][j+2] * gauss[2][2];
                        gauss_img[i][j+2] = current_y[i-1][j+1] * gauss[0][0] + current_y[i-1][j+2] * gauss[0][1] + current_y[i-1][j+3] * gauss[0][2]
                        + current_y[i][j+1] * gauss[1][0] + current_y[i][j+2] * gauss[1][1] + current_y[i][j+3] * gauss[1][2]
                        + current_y[i+1][j+1] * gauss[2][0] + current_y[i+1][j+2] * gauss[2][1] + current_y[i+1][j+3] * gauss[2][2];
                    }
                    case 0:continue;
                    case M-1:continue;}
                }
            case 0:continue;
            case M-1:continue;}
        }
    }
}
```

Εικόνα 1 Κώδικας πριν την αλλαγή

```
void gauss_filter(){
    int gauss[3][3] = {{1, 2, 1},{2, 4, 2},{1, 2, 1}};
    printf("Create Gaussian Image...\n");
    for (i = 1; i < N-1; i++)
    {
        for (j = 0; j < M; j=j+3)
        { switch (j) {
            default:{ //convolve gauss filter in grayscale image, grayscale is the Y Channel
                gauss_img[i][j] = current_y[i-1][j-1] * gauss[0][0] + current_y[i-1][j] * gauss[0][1] + current_y[i-1][j+1] * gauss[0][2]
                + current_y[i][j-1] * gauss[1][0] + current_y[i][j] * gauss[1][1] + current_y[i][j+1] * gauss[1][2]
                + current_y[i+1][j-1] * gauss[2][0] + current_y[i+1][j] * gauss[2][1] + current_y[i+1][j+1] * gauss[2][2];
                gauss_img[i][j+1] = current_y[i-1][j] * gauss[0][0] + current_y[i-1][j+1] * gauss[0][1] + current_y[i-1][j+2] * gauss[0][2]
                + current_y[i][j] * gauss[1][0] + current_y[i][j+1] * gauss[1][1] + current_y[i][j+2] * gauss[1][2]
                + current_y[i+1][j] * gauss[2][0] + current_y[i+1][j+1] * gauss[2][1] + current_y[i+1][j+2] * gauss[2][2];
                gauss_img[i][j+2] = current_y[i-1][j+1] * gauss[0][0] + current_y[i-1][j+2] * gauss[0][1] + current_y[i-1][j+3] * gauss[0][2]
                + current_y[i][j+1] * gauss[1][0] + current_y[i][j+2] * gauss[1][1] + current_y[i][j+3] * gauss[1][2]
                + current_y[i+1][j+1] * gauss[2][0] + current_y[i+1][j+2] * gauss[2][1] + current_y[i+1][j+3] * gauss[2][2];
            }
            case 0:continue;
            case M-1:continue;}
        }
    }
}
```

Εικόνα 2 Κώδικας μετά την αλλαγή

Ιεραρχία Μνήμης 1:

Η ιεραρχία μνήμης που ορίσαμε περιλαμβάνει μία μνήμη ROM 512KB, μία μνήμη RAM(DRAM) 3.56MB και μία μνήμη SRAM που λειτουργεί σαν cache 1MB. Οι μνήμες ROM, RAM είναι off-chip, ενώ η SRAM on-chip. Η RAM επιλέχθηκε να είναι σχετικά μεγάλη, καθώς οι πράξεις της συνέλιξης απαιτούν την χρήση πολλών στοιχείων. Ομοίως, η μνήμη SRAM επιλέχθηκε να είναι σχετικά μεγάλη γιατί οι μεταβλητές-πίνακες που προσπελάζονται πιο συχνά είναι μεγάλοι σε μέγεθος. Επομένως, η δομή που χρησιμοποιήθηκε είναι η εξής(στο αρχείο *memory.map*):

Αρχική διεύθυνση μνήμης	Μέγεθος μνήμης (hex)	Όνομα μνήμης	Μέγεθος διαύλου (Byte)	Χρήση Μνήμης	Χρόνος read N/S	Χρόνος write N/S
0x0	0x0080000	ROM	4	R(=read)	150/100	150/100
0x0080000	0x0390C3C	RAM	4	RW(read,write)	50/25	50/25
0x0410C3C	0x0100000	SRAM	4	RW(read,write)	1/1	1/1

Οι χρόνοι επιλέχθηκαν σχεδόν αυθαίρετα, με τους χρόνους της ROM να είναι οι πιο μεγάλοι καθώς είναι πιο αργή αλλά και βρίσκεται off-chip, ενώ οι χρόνοι της RAM είναι πιο μικροί γιατί παρόλο που βρίσκεται off-chip είναι πιο γρήγορη σε σχέση με την ROM. Επιπλέον, επειδή η SRAM είναι ταχύτερη και από τις 2 μνήμες, αλλά και επειδή είναι on-chip οι χρόνοι της θεωρήθηκαν ιδανικοί γι' αυτό και είναι μικρότεροι από όλους. Τέλος, επειδή οι σειριακές κλήσεις στην μνήμη είναι ταχύτερες, καθώς τα στοιχεία βρίσκονται κοντά μεταξύ τους τοπικά, ο χρόνος των σειριακών κλήσεων είναι μικρότερος από αυτόν των μη σειριακών κλήσεων.

Επιπρόσθετα, υπάρχει η δομή του stack/heap, που αποτελεί μέρος της μνήμης RAM. Τροποποιήθηκε έτσι ώστε να χωράει τις τοπικές μεταβλητές και άλλα στοιχεία, χωρίς όμως να επικαλύπτεται με τις άλλες μνήμες. Επομένως για να μην υπάρχει επικάλυψη άλλαξε η αρχική της διεύθυνση(0x0041043C) και η τελική της (0x00410C3C), δηλαδή η διαφορά μεταξύ heap και stack είναι 2KB περίπου.

Αντίστοιχα, το αρχείο scatter που καθορίζει το είδος των δεδομένων που αποθηκεύονται σε κάθε μνήμη, έχει αυτήν την μορφή:

```
ROM 0x0 0x00080000
{
ROM 0x0 0x00080000
{
*.o ( +RO )
}
RAM 0x00080000 0x00390C3C
{
```

```

* ( ram )
* ( +ZI, +RW )
}
SRAM 0x00410C3C 0x00100000
{
*( sram )
}
}

```

Δοκιμές:

Αρχικά, όλα οι μεταβλητές τοποθετήθηκαν στην μνήμη DRAM, με αποτέλεσμα την τεράστια αύξηση των κύκλων ρολογιού, διότι πλέον εμφανίζονται `wait_cycles`, όπου ο επεξεργαστής είναι σε αδράνεια/αναμονή γιατί πιθανόν να περιμένει να του προσκομιστούν δεδομένα από τις μνήμες. Στην συνέχεια, τοποθετήσαμε τις μεταβλητές `i,j` που χρησιμοποιούνται στις `for loops` στην μνήμη SRAM, αφού προσπελάζονται συχνά (σε κάθε `loop`), άρα αναμένουμε μείωση των κύκλων. Πράγματι, επιτεύχθηκε μια μείωση, ωστόσο δοκιμάσαμε να βάλουμε και μερικούς πίνακες δεδομένων (τους πιο συχνά χρησιμοποιούμενους) στην μνήμη αυτή.

Ειδικότερα, βάζοντας τους πίνακες `lx,ly` που απεικόνιζαν τις συνιστώσες της `gradient image`, αλλά και με την προσθήκη των μεταβλητών `max,min` παρατηρήθηκε περαιτέρω πτώση των `clock cycles`. Έπειτα, μεταφέρθηκαν οι πίνακες `dl,theta` στην μνήμη SRAM, αφού αφαιρέθηκαν οι πίνακες `lx,ly` και διαπιστώθηκε επιπλέον ελάττωση των κύκλων ρολογιού. Τέλος, αποθηκεύοντας τις μεταβλητές `max,min` στην SRAM επιτεύχθηκε το βέλτιστο αποτέλεσμα. Στον πίνακα που ακολουθεί, γίνεται η σύγκριση μεταξύ των δοκιμών.

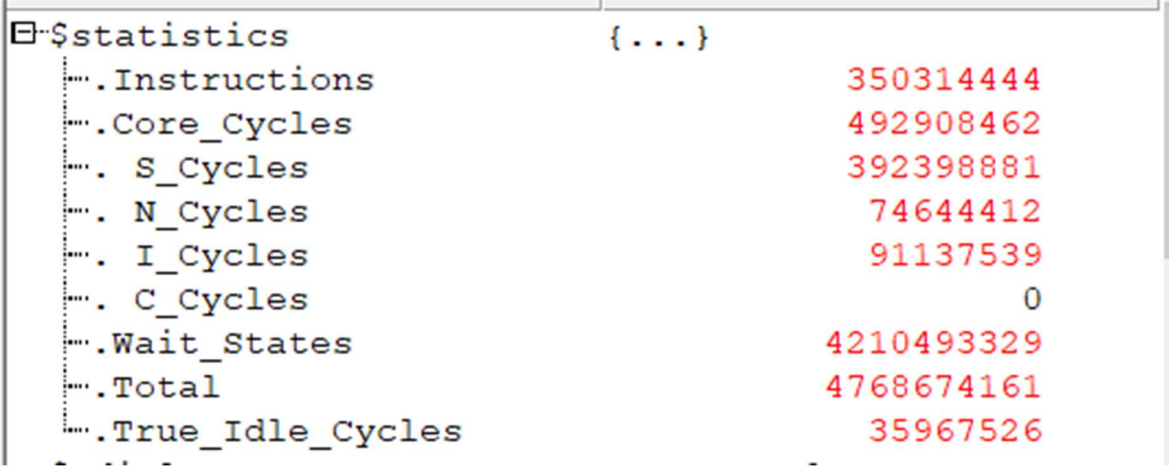
Δοκιμή	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	Wait_States	Total
Όλες οι μεταβλητές στην DRAM	350314447	492908467	392398885	74644413	91137539	2047332140	2605512977
<code>i,j</code> στην SRAM	350314444	492908462	392398881	74644412	91137539	2039900279	2598081111
<code>i,j,lx,ly</code> στην SRAM	350314444	492908462	392398881	74644412	91137539	2038038791	2596219623
<code>i,j,lx,ly,max,min</code> στην SRAM	350314444	492908462	392398881	74644412	91137539	2037805211	2595986043
<code>i,j,dl,theta</code> στην SRAM	350314444	492908462	392398881	74644412	91137539	2037892789	2596073621
<code>i,j,dl,theta,max,min</code> στην SRAM	350314444	492908462	392398881	74644412	91137539	2037659209	2595840041

Στην **πράσινη γραμμή** φαίνεται το βέλτιστο αποτέλεσμα. Ωστόσο η χρήση SRAM με μέγεθος 1MB είναι αδόκιμη καθώς είναι πολύ μεγάλη και επομένως έπρεπε να μειωθεί,

ξέροντας πως κάτι τέτοιο θα έχει αρνητικές επιπτώσεις στους κύκλους του ρολογιού, καθώς θα αυξηθούν τα wait states.

Αλλαγή διαύλου:

Μια δοκιμή που επιχειρήθηκε ήταν η αλλαγή του μεγέθους του διαύλου(bus). Συγκεκριμένα, στην προηγούμενη ενότητα χρησιμοποιήθηκε δίαυλος 4byte(32bit), ενώ τώρα δοκιμάστηκε δίαυλος 2byte(16bit) όπου αναμένουμε σχεδόν διπλασιασμό των κύκλων ρολογιού καθώς μια μεταφορά 4byte χρειάζεται πλέον δύο κύκλους(αντί για έναν).



Statistics { ... }	
.Instructions	350314444
.Core_Cycles	492908462
. S_Cycles	392398881
. N_Cycles	74644412
. I_Cycles	91137539
. C_Cycles	0
.Wait_States	4210493329
.Total	4768674161
.True_Idle_Cycles	35967526

Εικόνα 3 Αποτελέσματα αλλαγής διαύλου

Όντως , οι κύκλοι ρολογιού αυξήθηκαν, καθώς διπλασιάστηκαν τα wait_states. Με παρόμοιο τρόπο αν χρησιμοποιηθεί δίαυλος 8byte(64bit) τότε θα υποδιπλασιαστούν τα wait states, με αποτέλεσμα μείωση των συνολικών κύκλων ρολογιού. Ωστόσο, κάτι τέτοιο είναι αδύνατο και δεν μπορεί να υλοποιηθεί.

Ιεραρχία Μνήμης 2:

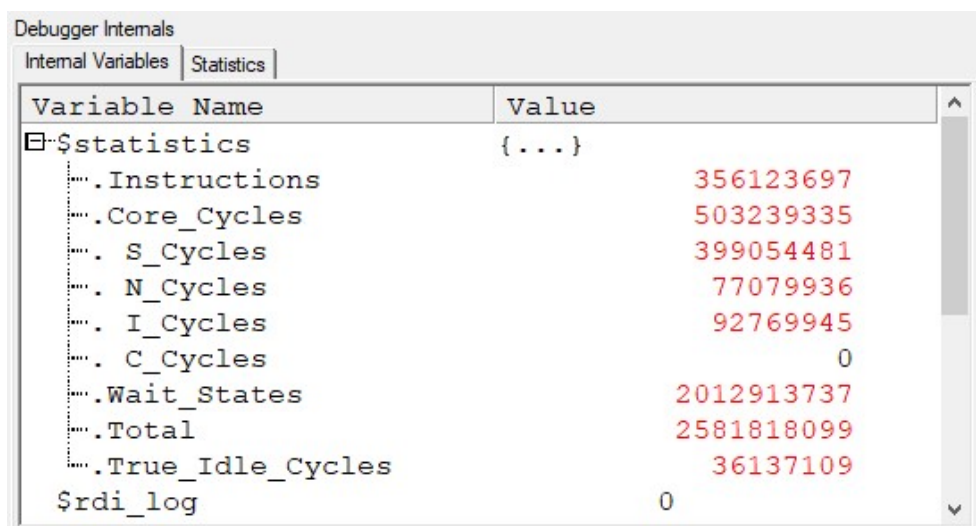
Επομένως, το μέγεθος της SRAM πλέον είναι 64KB, με τις υπόλοιπες μνήμες να μένουν ως έχει. Δοκιμάζοντας πιθανές βελτιστοποιήσεις, καταλήξαμε στο συμπέρασμα πως πλέον δεν χωράει κανένας πίνακας δεδομένων, παραμόνο οι μεταβλητές i, j, max, min . Στον παρακάτω πίνακα φαίνονται τα αποτελέσματα.

Δοκιμή	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	Wait_States	Total
Όλες οι μεταβλητές στην DRAM	350314447	492908467	392398885	74644413	91137539	2047332140	2605512977
i, j στην SRAM	350314444	492908462	392398881	74644412	91137539	2039900279	2598081111
i, j, max, min στην SRAM	350080923	492207899	392165360	74410891	90904018	2036397405	2593877674
i, j στην SRAM με 2byte bus	350314444	492908462	392398881	74644412	91137539	4213854934	4772035766
i, j, max, min στην SRAM με 4byte bus	350080923	492207899	392165360	74410891	90904018	4207900060	4765380329

Οι βέλτιστες επιδόσεις απεικονίζονται στην πράσινη γραμμή. Επίσης, από την στιγμή που χωράνε μόνο οι μεταβλητές i, j, max, min η μνήμη SRAM μπορεί να ελαχιστοποιηθεί και να έχει μέγεθος 12B, καθώς οποιαδήποτε προσπάθεια εισαγωγής πίνακα απαιτεί 114KB (χρησιμοποιώντας την εικόνα car).

Loop Tiling Revisited:

Όπως είναι ήδη γνωστό, η βελτιστοποίηση loop-tiling δουλεύει καλύτερα σε δομές με ιεραρχία μνήμης. Ωστόσο, όταν την δοκιμάσαμε στην 1^η ιεραρχία μνήμης, τα αποτελέσματα δεν ήταν τα αναμενόμενα, καθώς οι κύκλοι αυξήθηκαν αντί να μειωθούν.



Variable Name	Value
\$statistics	{...}
.Instructions	356123697
.Core_Cycles	503239335
.S_Cycles	399054481
.N_Cycles	77079936
.I_Cycles	92769945
.C_Cycles	0
.Wait_States	2012913737
.Total	2581818099
.True_Idle_Cycles	36137109
\$rdi_log	0

Εικόνα 4 Αποτελέσματα μετά το loop tiling

Όπου η τεχνική loop-tiling χρησιμοποιήθηκε στις συναρτήσεις color & magn_theta_calc.

Συμπεράσματα:

Δεδομένου του μεγέθους του προβλήματος που μας έχει ανατεθεί (υλοποίηση 3 συνελίξεων, πολλαπλές if για την εύρεση της κατεύθυνσης των εικονοστοιχείων κτλ.), απαιτείται η χρήση αρκετών πόρων. Φυσικό και επόμενο λοιπόν, να χρησιμοποιείται RAM 3.5MB και SRAM 1MB στην 1^η ιεραρχία μνήμης που δοκιμάστηκε, που πιθανόν να θεωρούνται μεγάλες (και ειδικά για την SRAM και ακριβές σε πιθανή υλοποίηση). Σε κάθε περίπτωση, με την εισαγωγή της ιεραρχίας μνήμης, μεγαλώνουν οι κύκλοι ρολογιού, διότι αυξάνεται ο αριθμός των wait_states. Αυτό οφείλεται στο γεγονός ότι η προσπέλαση της μνήμης διαρκεί πολύ περισσότερο από πριν. Επομένως, το ιδανικό σενάριο θα ήταν η τοποθέτηση όλων των δεδομένων στην SRAM on-chip μνήμη, για να εκμηδενιστούν οι χρόνοι μεταφοράς/αναμονής κτλ., που όμως δεν είναι εφικτό λόγω των προαναφερθέντων περιορισμών.

\$statistics	{...}	
...Instructions		350314447
...Core_Cycles		492908465
...S_Cycles		392398885
...N_Cycles		74644412
...I_Cycles		91137538
...C_Cycles		0
...Wait_States		0
...Total		558180835
...True_Idle_Cycles		35967526
\$rdi_log		0
\$target_fpu		1
\$image_cache_enable		0
\$clock		11163616

Εικόνα 5 Ιδανική περίπτωση μνήμης

Όπου φαίνεται πως οι κύκλοι στους οποίους ο επεξεργαστής είναι σε αναμονή έχουν μηδενιστεί, καθώς ουσιαστικά πρόκειται για τα αποτελέσματα της προηγούμενης εργασίας.

Τα αρχεία που χρησιμοποιήθηκαν για την 1^η ιεραρχία μνήμης είναι τα:

- mymem1.map
- myscatter1.txt
- stack1.c
- optimized.c

Τα αρχεία που χρησιμοποιήθηκαν για την 2^η ιεραρχία μνήμης είναι τα:

- mymem2.map
- myscatter2.txt
- stack2.c
- optimized2.c

Τα αρχεία που χρησιμοποιήθηκαν για την ιδανική ιεραρχία μνήμης είναι τα:

- ideal.map
- ideal_scatter.txt
- ideal_stack.c
- ideal.c