

Δημοκρίτειο Πανεπιστήμιο Θράκης  
Πολυτεχνική Σχολή Ξάνθης  
Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Η/Υ

Σχεδιασμός Ενσωματωμένων Κυκλωμάτων

Εργασία 1η

Στα πλαίσια του μαθήματος "Σχεδιασμός Ενσωματωμένων Συστημάτων", μας ζητήθηκε η εργασία για "Περιγραφή ενός αλγορίθμου αύξησης οξύτητας grayscale εικόνων εφαρμόζοντας την τεχνική unsharp masking" είναι εκείνη που σας ανατίθεται. Σκοπός της εργασίας είναι να υλοποιηθεί σε γλώσσα C ο προτεινόμενος αλγόριθμος και στη συνέχεια να εφαρμοστεί η μεθοδολογία βελτιστοποίησης μεταφοράς και αποθήκευσης δεδομένων στον αλγόριθμο αυτό.

(α) Αρχικά να εφαρμοστούν οι μετασχηματισμοί βρόχων για την κανονικοποίηση της δομής του αλγορίθμου αλλά και να εφαρμοστεί οποιοσδήποτε άλλος μετασχηματισμός για τη βελτίωση της απόδοσης του αλγορίθμου.

(β) Στη συνέχεια να βρεθεί το μέγεθος του πίνακα δεδομένων και ο αριθμός των προσπελάσεων που γίνονται σε αυτόν.

(γ) Ολοκληρώνοντας τις δυνατές εκτελέσεις-βελτιστοποιήσεις να πραγματοποιηθούν μετρήσεις για την ταχύτητα εκτέλεσης του αλγορίθμου με τη βοήθεια του προσομοιωτή ARMulator του επεξεργαστή ARM.

Περιγραφή ενός αλγορίθμου αύξησης οξύτητας grayscale εικόνωνεφαρμόζοντας την τεχνική unsharp masking

**Βήμα 1ο :** Ορισμός χαρακτηριστικών του φίλτρου Gauss (μέγεθος και τιμή της τυπικής απόκλισης) για να προκύψει η κατάλληλη μάσκα συνέλιξης. Τα βάρη της ζητούμενης μάσκας διαστάσεων nxn και τυπικής απόκλισης σ2 προκύπτουν από την εξίσωση :



Όπου c είναι μια σταθερά κανονικοποίησης και i και j παίρνουν τιμές στο διάστημα [-floor(n/2), floor(n/2)]. Για τον προσδιορισμό της μεταβλητής c, θέτουμε g[n,n]=1 και λύνουμε ως προς την μεταβλητή. Εκτελώντας τη συνέλιξη οι τιμές των εικονοστοιχείων πρέπει να κανονικοποιηθούν με το άθροισμα των βαρών της μάσκας.

**Βήμα 2ο :** Εφαρμογή την μάσκας Gauss στην αρχική εικόνα μέσω συνέλιξης για την εξαγωγή την εξομαλυμένης εικόνας.

**Βήμα 3ο :** Αφαίρεση της εικόνας Ιs από την αρχική εικόνα Ι, ώστε να προκύπτει η Ιmask

Ιmask=Ι - Ιs

**Βήμα 3ο :** Υπολογισμός της τελικής εικόνας με βάση τη σχέση

Ιunsharp =Ι – kΙmask

Όπου k>0. Όταν ισχύει , εμφανίζεται αύξηση της οξύτητας, ένω όταν k<1 έχουμε εξομάλυνση εικόνας. Συνήθως ο συντελεστής k παίρνει ακέραιες τιμές στο διάστημα .

Αρχικός Κώδικας

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

#define im\_rows 288 //image size

#define im\_collumns 352

#define padded\_rows 292 // padded image size

#define padded\_collumns 356 // added two zeros on each side (for the case where N=5)

#define N 5 // filter class

#define STD 0.88 // standard deviation

#define K 3 // constant

#define filename "image\_in.yuv" //input file name define

#define finalfilename "image\_out.yuv" //output file name define

// Global Variables

int image[im\_rows ][im\_collumns]; //the image we are filtering

int padded\_image[padded\_rows][padded\_collumns]={0}; // padded image

double g[N][N]; // Gauss filter mask/kernell NXN size

double g\_flipped[N][N]; //needed for convolution

double conv\_out[im\_rows ][im\_collumns]; //output of convolution

//conv\_out is the same size as the initial image

// end initialization

//Image Read

void read() //image read

{

int i,j; //indexes //error when initialized inside for

FILE \*frame\_c;

if((frame\_c=fopen(filename,"rb"))==NULL)

{

printf("current frame doesn't exist\n");

exit(-1);

}

for(i=0;i<im\_rows;i++) // extracts lines

{

for(j=0;j<im\_collumns;j++) //extracts collumns

{

image[i][j] =fgetc(frame\_c);

}

}

fclose(frame\_c);

}

// end read

// Gauss Filter Kernell

void gauss(){

int i,j;

int x,y; //needed for gauss meshgrid

double c; //gauss filter coefficient

int sum=0; //used for normalization

x =-N/2;

c =exp(N\*N/(STD\*STD));

for(i=0;i<N;i++){

y =-N/2;

for(j=0;j<N;j++){

g[i][j] =c\*exp(-(pow (x,2)+pow(y,2))/(2\*STD\*STD)); // more efficient/faster to implement without pow

sum =sum+g[i][j];

y++;

}

x++;

}

for( i=0;i<N;i++){

for( j=0;j<N;j++){ // can be combined with the above for-loop fusion

g[i][j] = g[i][j]/501187233627271,5;

}

}

}//end gauss

// Kernell Flip(needed for convolution)

void Kernell\_Flip(){

int i,j; //indexes //error when initialized inside for

for(i=0; i<N; i++){

for( j=0; j<N; j++){

g\_flipped[i][j] =g[N-i-1][N-j-1];

}

}

}

//end Kernell\_Flip

// convolution 2D

void convolution2D(){

int i,j; //indexes //error when initialized inside for

int center = N/2; //automata kovei dekadika dld floor

int left = center-1;

int top = center - 1;

int x,y;

Kernell\_Flip();

//padding

for( i=top+1;i<=im\_rows+top;i++){

for( j=1+left;j<=im\_collumns+left;j++){

padded\_image[i][j]= image[i-top-1][j-left-1];

}

}

//padding x,y define the center pixel

for(x=0;x<im\_rows;x++){//more efficient to read per row

for( y=0;y<im\_collumns;y++){ //kernell is applied on each pixel

for(i=0;i<N;i++){ //center of the kernel is centered on the pixel we are reading

for(j=0;j<N;j++){

conv\_out[x][y]=conv\_out[x][y]+( padded\_image[i+x][j+y] \*g\_flipped[i][j] );

}//mask is applied(multiplication) on the surrounding pixels of the centered one

} // then the results of each mult. are summed together

}

}

}

//end convolution 2D

// final steps of algorithm

void final\_steps(){

int i,j; //indexes //error when initialized inside for

int Imask[im\_rows ][im\_collumns];

// image values range from 0 to 255 thus we need clipping

for(i=0;i<im\_rows;i++){

for(j=0;j<im\_collumns;j++){ // Imask=I-Is , Iunsharp=I+K\*Imask

Imask[i][j] =(image[i][j]-conv\_out[i][j]);

}

}

for(i=0;i<im\_rows;i++){

for(j=0;j<im\_collumns;j++){ // Imask=I-Is , Iunsharp=I+K\*Imask

image[i][j] =image[i][j] + K\*(Imask[i][j]);

if(image[i][j]>255){image[i][j] =255;}// maximum pixel intensity is 255

else if(image[i][j]<0){image[i][j] =0;}// minimum pixel intensity is 0

}

}

}

//end final\_steps

//write file

void write(){

int i,j; //indexes //error when initialized inside for

FILE \*frame\_y;

frame\_y=fopen(finalfilename,"wb");

for(i=0;i<im\_rows ;i++)

{

for(j=0;j<im\_collumns;j++)

{

fputc(image[i][j],frame\_y);

}

}

fclose(frame\_y);

}

// end write

int main(){

read();

gauss();

convolution2D();

final\_steps();

write();

return(0);

}

Initial results

|  |  |
| --- | --- |
| Instructions | 384558902 |
| Core\_Cycles | 547832789 |
| S\_Cycles | 425929562 |
| N\_Cycles | 91745781 |
| I\_Cycles | 101567858 |
| C\_Cycles | 0 |
| Total | 642292153 |
| Grand Totals | |
| Code | 22976 |
| RO Data | 606 |
| RW Data | 0 |
| ZI Data | 1633020 |
| Debug | 14096 |
|  |  |

Optimize 0

|  |  |  |  |
| --- | --- | --- | --- |
|  | Αρχικό | Βελτιστοποιημένο | Απόδοση(Μείωση) |
| Instructions | 384558902 | 360626373 | 6.2234 |
| Core\_Cycles | 547832789 | 511141516 | 6.6975 |
| S\_Cycles | 425929562 | 405166652 | 4.8747 |
| N\_Cycles | 91745781 | 80032326 | 12.7673 |
| I\_Cycles | 101567858 | 96349596 | 5.1377 |
| C\_Cycles | 0 | 0 | 0 |
| Total | 642292153 | 638240783 | 0.6308 |
| Grand Totals | | | |
| Code | 22976 | 18156 | 20.9784 |
| RO Data | 606 | 470 | 20.9784 |
| RW Data | 0 | 0 | 0 |
| ZI Data | 1633020 | 1633020 | 0 |
| Debug | 14096 | 13316 | 5.5335 |

Φροντίσαμε στον πρώτος μας κώδικα να αλλάξουμε τις άσκοπες συναρτήσεις, αφού μπορούμε να το πετύχoυμε με πιο απλό τρόπο, ώστε να γλυτώσουμε κύκλους εκτελέσεων. Αποφύγαμε να χρησιμοποιήσουμε την εντολή pow και την υλοποιήσαμε χειροκίνητα με πολλαπλασιασμό. Επίσης αποφύγαμε την χρήση συναρτήσεων floor και ceil αφού έχουμε Integer οι όποιοι κόβουν αυτόματα τα δεκαδικά και κάνουμε στρογγυλοποίηση. Ακόμη στα τελευταία κομμάτια του αλγορίθμου τα οποία είναι η εφαρμογή της μάσκας και η αφαίρεση της από την αρχική φροντίσαμε να υλοποιηθούν στην ίδια for και όχι σε ξεχωριστές for, εφαρμόζοντας την λογική loop fusion (finalsteps()).

Optimize 1

|  |  |  |  |
| --- | --- | --- | --- |
|  | Προηγούμενο | Βελτιστοποιημένο | Απόδοση(Μείωση) |
| Instructions | 360626373 | 360622112 | 0.002% |
| Core\_Cycles | 511141516 | 511134946 | 0.002% |
| S\_Cycles | 405166652 | 405161720 | 0.003% |
| N\_Cycles | 80032326 | 80031139 | 0.002% |
| I\_Cycles | 96349596 | 96349149 | ~0%(10^-4) |
| C\_Cycles | 0 | 0 | 0% |
| Total | 638240783 | 634912444 | 0.52% |
| Grand Totals | | | |
| Code | 18156 | 18028 | 0.71 |
| RO Data | 470 | 470 | 0% |
| RW Data | 0 | 0 | 0% |
| ZI Data | 1633020 | 1633020 | 0% |
| Debug | 13316 | 13232 | 0.63 |

* Στα Gauss αντί να κάνω άλλη for για το normalization των βαρών το βάζω κατευθείαν από την πρώτη loopfusion (optimize\_1.)
* Στα Gauss λόγω του μεταβλητού Ν δεν μπορώ να κάνω unroll, γιατί μπορεί να βγω εκτός ορίων. Αν έβαζα if για να κάνω unroll το κόστος if σε κάθε επανάληψη είναι χειρότερο, άρα δεν κερδίζω από το unroll.
* Με i++ είναι πιο efficient από το i=i+1, καθώς αντιστοιχεί σε παραπάνω εντολές το τελευταίο.

Πιο συγκεκριμένα απαιτεί λιγότερες εντολές στην assembly και λιγότερη επικοινωνία με την μνήμη.

Optimize 2

|  |  |  |  |
| --- | --- | --- | --- |
|  | Προηγούμενο | Βελτιστοποιημένο | Απόδοση |
| Instructions | 360622112 | 360621963 | ~0%(10^-4) |
| Core\_Cycles | 511134946 | 511134622 | ~0%(10^-4) |
| S\_Cycles | 405161720 | 405161564 | ~0%(10^-4) |
| N\_Cycles | 80031139 | 80030978 | ~0%(10^-4) |
| I\_Cycles | 96349149 | 96349142 | ~0%(10^-4) |
| C\_Cycles | 0 | 0 | 0% |
| Total | 634912444 | 581541684 | 8.41% |
| Grand Totals | | | |
| Code | 18028 | 18012 | 0.1 |
| RO Data | 470 | 470 | 0% |
| RW Data | 0 | 0 | 0% |
| ZI Data | 1633020 | 1632820 | 0.02 |
| Debug | 13232 | 13048 | 1.39 |

* Χρησιμοποιούμε δυο μάσκες την κανονική και την flipped που χρησιμοποιείται στο convolution το οποίο είναι μια σπάταλη στην μνήμη για τα ενσωματωμένα συστήματα, το όποια έχουν μικρή χωρητικότητα σε μνήμη. Για να το επιτύχουμε αυτό μετακινούμε την συνάρτηση kernel flip μέσα στο Gauss ώστε το g να γίνει τοπική μεταβλητή και να διαγράφει μετά την χρήση της. Παρατηρείτε ότι αυτή η ενσωμάτωση την kernel flip μαζί με την εξοικονόμηση χώρου της μνήμη είχε δραστική μείωση στου κύκλους.
* Η kernel flip την καλούσαμε μέσα από μια συνάρτηση, άρα με το την ενσωματώσουμε στην συνάρτηση Gauss γλυτώνουμε όλες τις εντολές τις assembly που χρειάζονται για την κλήση της συνάρτησης. Πιο συγκεκριμένα για να θέση τον link register, να μεταφέρει όλα τα δεδομένα των καταχωρητών στην στοίβα και τις ανάλογες εντολές που χρειάζονται για την επιστροφή στην συνάρτηση (ΒX LR).
* Εδώ θεωρούμε ότι η μνήμη είναι ιδανική, άρα δεν έχουμε wait cycle θα γλυτώναμε όλους του κύκλους που αφορούν το φόρτωμα και το «ξεφόρτωμα» από και προς την μνήμη. Επίσης εκμεταλλευόμαστε ότι οι μεταβλητές του g θα υπήρχαν στους καταχωρητές ή στην cache αν υπήρχε και θα γλυτώναμε wait cycles από την μνήμη.

Optimize 3\_2

|  |  |  |  |
| --- | --- | --- | --- |
|  | Προηγούμενο | Βελτιστοποιημένο | Απόδοση |
| Instructions | 360621963 | 360470474 | 0.05 |
| Core\_Cycles | 511134622 | 510882331 | 0.05 |
| S\_Cycles | 405161564 | 404959387 | 0.05 |
| N\_Cycles | 80030978 | 79980577 | 0.07 |
| I\_Cycles | 96349142 | 96349429 | 0% |
| C\_Cycles | 0 | 0 | 0% |
| Total | 581541684 | 581289393 | 0.05 |
| Grand Totals | | | |
| Code | 18012 | 18036 | -0.1332 |
| RO Data | 470 | 470 | 0% |
| RW Data | 0 | 0 | 0% |
| ZI Data | 1632820 | 1632820 | 0 |
| Debug | 13048 | 13060 | -0.1 |

* Εδώ κάνουμε την τεχνική loop unroll και ουσιαστικά γραφούμε χειροκίνητα 2 step της εντολής for και μεταβάλλουμε το βήμα κατά 2. Με αυτόν τον τρόπο μπορεί να αυξάνουμε τις γραμμές του κώδικα, αλλά μειώνουμε τις επαναλήψεις στην for, αλλά μειώνουμε τους ελέγχους που γίνονται αφού έχουμε λιγότερες επαναλήψεις, άρα και λιγότερους απαιτούμενους κύκλους.

Optimize 3\_4

|  |  |  |  |
| --- | --- | --- | --- |
|  | Προηγούμενο | Βελτιστοποιημένο | Απόδοση |
| Instructions | 360470474 | 360394442 | 0.03 |
| Core\_Cycles | 510882331 | 510755611 | 0.02 |
| S\_Cycles | 404959387 | 404858011 | 0.04 |
| N\_Cycles | 79980577 | 79955233 | 0.03 |
| I\_Cycles | 96349429 | 96349429 | 0% |
| C\_Cycles | 0 | 0 | 0% |
| Total | 581289393 | 581162673 | 0.11 |
| Grand Totals | | | |
| Code | 18036 | 18076 | -0.22 |
| RO Data | 470 | 470 | 0% |
| RW Data | 0 | 0 | 0% |
| ZI Data | 1632820 | 1632820 | 0% |
| Debug | 13060 | 13060 | 0% |

* Όπως και πριν κάνουμε την τεχνική loop unroll, αλλά εδώ έχουμε 4 step της εντολής for και μεταβάλλουμε το βήμα κατά 4. Με αυτόν τον τρόπο μπορεί να αυξάνουμε τις γραμμές του κώδικα, αλλά μειώνουμε τις επαναλήψεις στην for και μειώνουμε τους ελέγχους που γίνονται αφού έχουμε λιγότερες επαναλήψεις, άρα και λιγότερους απαιτούμενους κύκλους.

Optimize 3\_8

|  |  |  |  |
| --- | --- | --- | --- |
|  | Προηγούμενο | Βελτιστοποιημένο | Απόδοση |
| Instructions | 360394442 | 360356426 | 0.01 |
| Core\_Cycles | 510755611 | 510692251 | 0.02 |
| S\_Cycles | 404858011 | 404807323 | 0.01 |
| N\_Cycles | 79955233 | 79942561 | 0.02 |
| I\_Cycles | 96349429 | 96349429 | 0% |
| C\_Cycles | 0 | 0 | 0% |
| Total | 581162673 | 581099313 | 0.32 |
| Grand Totals | | | |
| Code | 18076 | 18156 | -0.44 |
| RO Data | 470 | 470 | 0% |
| RW Data | 0 | 0 | 0% |
| ZI Data | 1632820 | 1632820 | 0% |
| Debug | 13060 | 13064 | -0.03 |

* Όπως και πριν κάνουμε την τεχνική loop unroll, αλλά εδώ έχουμε 8 step της εντολής for και μεταβάλλουμε το βήμα κατά 8. Εδώ όμως το overhead των παραπάνω εντολών που γράψαμε στον κώδικα για το loop unroll είναι πιο έντονο. Όμως δικαιολογείται από την μείωση που είδαμε στους κύκλους.

Optimize 3\_16

|  |  |  |  |
| --- | --- | --- | --- |
|  | Προηγούμενο | Βελτιστοποιημένο | Απόδοση |
| Instructions | 360356426 | 360337418 | 0.05 |
| Core\_Cycles | 510692251 | 509646817 | 0.21 |
| S\_Cycles | 404807323 | 404781980 | 0.01 |
| N\_Cycles | 79942561 | 79429348 | 0.64 |
| I\_Cycles | 96349429 | 95842551 | 0.53 |
| C\_Cycles | 0 | 0 | 0% |
| Total | 581099313 | 580053879 | 0.15 |
| Grand Totals | | | |
| Code | 18156 | 18324 | -0.91 |
| RO Data | 470 | 470 | 0% |
| RW Data | 0 | 0 | 0% |
| ZI Data | 1632820 | 1632820 | 0% |
| Debug | 13064 | 13084 | -0.92 |

* Όπως και πριν κάνουμε την τεχνική loop unroll, αλλά εδώ έχουμε 16 step της εντολής for και μεταβάλλουμε το βήμα κατά 16. Εδώ όμως το overhead των παραπάνω εντολών που γράψαμε στον κώδικα για το loop unroll δεν δικαιολογεί την βελτιστοποίηση στα core cycles σε σύγκριση με το προηγούμενο του που είναι το loop unroll για 8 step.

Optimize 4,5,6

Μέχρι στιγμής όσες βελτιστοποίησης κάναμε επέτρεπαν την αλλαγή της τάξης Ν του φίλτρου Gauss, όμως για τα παραπάνω optimizations εκτελέσαμε loop unroll για Ν=5 σε όλα τα loop που εξαρτιόντουσαν από το ορό Ν. Τελικός όμως παρατηρήθηκε ότι δεν συνέφερε διότι το κέδρος (μείωση) στα core cycles ήταν της τάξης ~0.03, ενώ το overhead ήταν της τάξης 1% και συνδυάζοντας όλα αυτά μαζί το αποτέλεσμα ήταν ακόμα χειρότερο.

Optimize 7

|  |  |  |  |
| --- | --- | --- | --- |
|  | Προηγούμενο | Βελτιστοποιημένο | Απόδοση |
| Instructions | 360337418 | 360357291 | -0.0055 |
| Core\_Cycles | 509646817 | 510693690 | -0.2054 |
| S\_Cycles | 404781980 | 404809188 | -0.0067 |
| N\_Cycles | 79429348 | 79942840 | -0.6465 |
| I\_Cycles | 95842551 | 96349160 | -0.5286 |
| C\_Cycles | 0 | 0 | 0 |
| Total | 580053879 | 581100752 | -0.1805 |
| Grand Totals | | | |
| Code | 18324 | 18172 | 0.8295 |
| RO Data | 470 | 470 | 0 |
| RW Data | 0 | 0 | 0 |
| ZI Data | 1632820 | 1632820 | 0 |
| Debug | 13064 | 13048 | 0.1225 |

* Στη συγκεκριμένη βελτίωση στα final steps χρησιμοποιούμαι μια temp μεταβλητή ώστε να γλυτώσουμε κύκλους μνήμης. Πιο συγκεκριμένα δεδομένου ότι έχουμε load/store αρχιτεκτονική αν το αφήναμε χωρίς τη temp μεταβλητή θα έπρεπε πρώτα να αποθηκεύσει την τιμή πίσω στη μνήμη και μετά να την ξανατραβήξει. Ενώ με την προσωρινή μεταβλητή γλυτώνω όλους αυτούς τους κύκλους μνήμης.
* Στο συγκεκριμένο παράδειγμα επειδή έχουμε ιδανική μνήμη αυτή η βελτίωση δεν φαίνεται αλλά σε έναν πραγματικό επεξεργαστή οι κύκλοι μνήμης αποτελούν το μεγαλύτερο overhead άρα θα είχαμε σημαντική βελτίωση.

Σύγκριση με την αρχική έκδοση

|  |  |  |  |
| --- | --- | --- | --- |
|  | Προηγούμενο | Βελτιστοποιημένο | Απόδοση |
| Instructions | 360626373 | 360356426 | 0.07 |
| Core\_Cycles | 511141516 | 510692251 | 0.09 |
| S\_Cycles | 405166652 | 404807323 | 0.09 |
| N\_Cycles | 80032326 | 79942561 | 0.12 |
| I\_Cycles | 96349596 | 96349429 | 0.01 |
| C\_Cycles | 0 | 0 | 0 |
| Total | 638240783 | 581099313 | 9.96 |
| Grand Totals | | | |
| Code | 18156 | 18156 | 0 |
| RO Data | 470 | 470 | 0 |
| RW Data | 0 | 0 | 0 |
| ZI Data | 1633020 | 1632820 | 0.02 |
| Debug | 13316 | 13064 | 2 |
|  |  |  |  |

Παρατηρείται ότι το optimization που κάναμε για την μείωση των εντολών και περιττών δεδομένων κατάφερε να αναιρέσει το αρνητικό overhead λόγο των loop unroll. Άρα είχαμε μονό θετικές βελτιώσεις καθώς μειωθήκαν οι κύκλοι εντολών του κώδικα της τάξης του ~10% που οφείλονται κυρίως λόγο του optimization 2.

Για την τελική έκδοση χρησιμοποιήσαμε το optimization 3\_8

ΠΙΝΑΚΕΣ ΚΑΙ ΠΡΟΣΠΕΛΑΣΕΙΣ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Global | Bit | Διαστάσεις | Μέγεθος | Προσπελάσεις |
| image | 32 | 288x352 | 396ΚΒ | 405504 |
| padded\_image | 32 | 292x356 | 406.0625ΚΒ | 2635776 |
| g\_flipped | 64 | 5x5 | 200Β | 2534425 |
| conv\_out | 64 | 288x352 | 792ΚΒ | 5170176 |
| g | 64 | 5x5 | 200Β | 75 |

Λόγω της αρχιτεκτονικής load/store έχουμε access στο πίνακα των ανάλογων δεδομένων τόσο στο διάβασμα καθώς πρέπει να φορτωθεί στους καταχώρητες όσο και στην εγγραφή αφού πρέπει να μεταφερθεί από τους καταχωρητές στην μνήμη.

Ο πρώτος πίνακας αντιστοιχεί στο optimization 7 οπού έχουμε μείωση της επικοινωνίας με την μνήμη.

Στο παρακάτω πίνακα έχουμε το optimization 3\_8 (final) το οποίο έχει 101379 προσπελάσεις παραπάνω από ότι η 7 αλλά λιγότερους κύκλους.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Global | Bit | Διαστάσεις | Μέγεθος | Προσπελάσεις |
| image | 32 | 288x352 | 396ΚΒ | 506880 |
| padded\_image | 32 | 292x356 | 406.0625ΚΒ | 2635776 |
| g\_flipped | 64 | 5x5 | 200Β | 2534425 |
| conv\_out | 64 | 288x352 | 792ΚΒ | 5170176 |
| g | 64 | 5x5 | 200Β | 75 |

Η εικόνα μας πριν και μετά



Τελικός κώδικας

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

#define im\_rows 288 //image size

#define im\_collumns 352

#define padded\_rows 292 // padded image size

#define padded\_collumns 356 // added two zeros on each side (for the case where N=5)

#define N 5 // filter class

#define STD 0.88 // standard deviation

#define K 3 // constant

#define gauss\_const 501187233627271,5

#define filename "image\_in.yuv" //input file name define

#define finalfilename "image\_out.yuv" //output file name define

// Global Variables

int image[im\_rows ][im\_collumns]; //the image we are filtering

int padded\_image[padded\_rows][padded\_collumns]={0}; // padded image

double g\_flipped[N][N]; // Gauss filter mask/kernell NXN size

double conv\_out[im\_rows ][im\_collumns]; //output of convolution

//conv\_out is the same size as the initial image

// end initialization

//Image Read

void read() //image read

{

int i,j; //indexes //error when initialized inside for

FILE \*frame\_c;

if((frame\_c=fopen(filename,"rb"))==NULL)

{

printf("current frame doesn't exist\n");

exit(-1);

}

for(i=0;i<im\_rows;i++) // extracts lines

{

for(j=0;j<im\_collumns;j++) //extracts collumns

{

image[i][j] =fgetc(frame\_c);

}

}

fclose(frame\_c);

}

// end read

// Gauss Filter Kernell

void gauss(){

int i,j; //indexes //error when initialized inside for

int x,y; //needed for gauss meshgrid

double c; //gauss filter coefficient

double g[N][N]; // Gauss filter mask/kernell NXN size

//int sum=0; //used for normalization

x =-N/2;

c =exp(N\*N/(STD\*STD));

for(i=0;i<N;i++){

y =-N/2;

for(j=0;j<N;j++){ //we avoided using the pow function, in order to make it

g[i][j] =c\*exp(-(x\*x+y\*y)/(2\*STD\*STD)); // more efficient/faster

g[i][j] = g[i][j]/gauss\_const;

y++;

}

x++;

}

for(i=0; i<N; i++){

for( j=0; j<N; j++){

g\_flipped[i][j] =g[N-i-1][N-j-1];

}

}

// convolution 2D

void convolution2D(){

int i,j; //indexes //error when initialized inside for

int center =N/2; //automata kovei dekadika dld floor

int left =center-1;

int top =center-1;

int x,y;

//padding

for( i=top+1;i<=im\_rows+top;i++){

for( j=1+left;j<=im\_collumns+left;j=j+8){

padded\_image[i][j]= image[i-top-1][j-left-1];

padded\_image[i][j+1]= image[i-top-1][j+1-left-1];

padded\_image[i][j+2]= image[i-top-1][j+2-left-1];

padded\_image[i][j+3]= image[i-top-1][j+3-left-1];

padded\_image[i][j+4]= image[i-top-1][j+4-left-1];

padded\_image[i][j+5]= image[i-top-1][j+5-left-1];

padded\_image[i][j+6]= image[i-top-1][j+6-left-1];

padded\_image[i][j+7]= image[i-top-1][j+7-left-1];

}

}

//padding x,y define the center pixel

for(x=0;x<im\_rows;x++){//more efficient to read per row

for( y=0;y<im\_collumns;y++){ //kernell is applied on each pixel

for(i=0;i<N;i++){ //center of the kernel is centered on the pixel we are reading

for(j=0;j<N;j++){

conv\_out[x][y]=conv\_out[x][y]+( padded\_image[i+x][j+y] \*g\_flipped[i][j] );

}//mask is applied(multiplication) on the surrounding pixels of the centered one

} // then the results of each mult. are summed together

}

}

}

//end convolution 2D

// final steps of algorithm

void final\_steps(){

int i,j; //indexes //error when initialized inside for

// image values range from 0 to 255 thus we need clipping

for(i=0;i<im\_rows;i++){

for(j=0;j<im\_collumns;j++){ // Imask=I-Is , Iunsharp=I+K\*Imask

image[i][j] =image[i][j] + K\*(image[i][j]-conv\_out[i][j]);

if(image[i][j]>255){image[i][j] =255;}// maximum pixel intensity is 255

else if(image[i][j]<0){image[i][j] =0;}// minimum pixel intensity is 0

}

}

}

//end final\_steps

//write file

void write(){

int i,j; //indexes //error when initialized inside for

FILE \*frame\_y;

frame\_y=fopen(finalfilename,"wb");

for(i=0;i<im\_rows ;i++)

{

for(j=0;j<im\_collumns;j++)

{

fputc(image[i][j],frame\_y);

}

}

fclose(frame\_y);

}

// end write

void main(){

read();

gauss();

convolution2D();

final\_steps();

write();

}