Παράλληλος Προγραμματισμός 2017
Προγραμματιστική Εργασία #1

Παπαχρήστος Γεώργιος
ΑΜ Π2010035

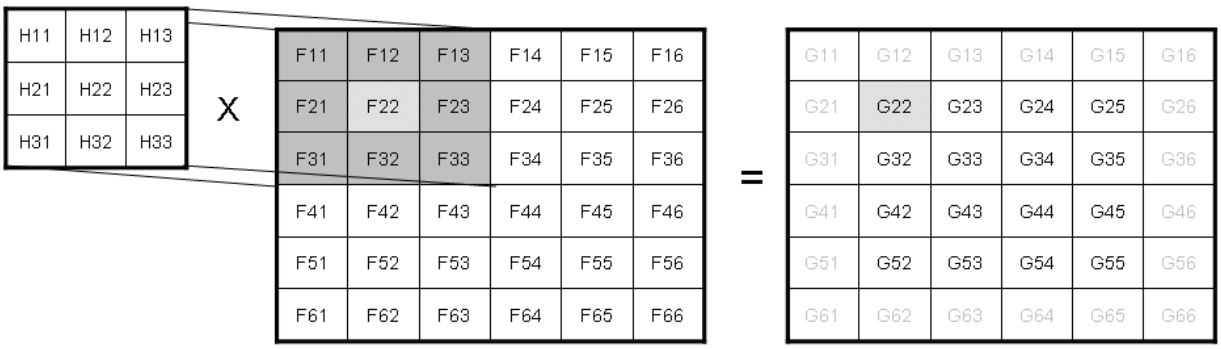
Στην επεξεργασία εικόνας συχνά εμφανίζεται ο μετασχηματισμός, κατά τον οποίο κάθε pixel αντικαθίσταται από μια νέα τιμή υπολογισμένη από τα 8 γειτονικά pixels (και το ίδιο). Τεχνικές που το χρησιμοποιούνε είναι η Soblel, Prewwitt, Canny ανίχνευση ακμών.

Κατά τον μετασχηματισμό, κάθε αρχικό pixel πολλαπλασιάζεται με μια σταθερά float.

Για τα γειτονικά pixels
P0 P1 P2
P3 **P4** P5
P6 P7 P8
το P4 είναι το κεντρικό pixel με τιμή από 0 έως 255.

Έχουμε και τις σταθερές μετασχηματισμού:
K0 K1 K2
K3 **K4** K5
K6 K7 K8

με τιμές όλα 0.5 εκτός από το K4 που είναι ίσο με 5.0.



Σχήμα 1

Κάθε κεντρικό pixel θα υπολογιστεί από την φόρμουλα
newP4 = P0*K0+P1*K1+P2*K2+P3*K3+P4*K4+P5*K5+P6*K6+P7*K7+P8*K8
όπως φαίνεται και στο παραπάνω σχήμα.

Αλγόριθμος no-sse.c

Θεωρούμε τον πίνακα
float K[3][3] = { {0.5,0.5,0.5}, {0.5,5.0,0.5}, {0.5,0.5,0.5}};
για τις σταθερές μετασχηματισμού.

Και

```
float **imageOriginal = malloc(sizeof *imageOriginal * N);
float **imageFinal = malloc(sizeof *imageFinal * N);

if (imageOriginal)
{
  for (i = 0; i < N; i++)
  {
    imageOriginal[i] = malloc(sizeof *imageOriginal[i] * M);
  }
}
if (imageFinal)
{
  for (i = 0; i < N; i++)
  {
    imageFinal[i] = malloc(sizeof *imageFinal[i] * M);
  }
}
```

δέσμευση μνήμης για την αρχική και τελική παραγόμενη εικόνα, ανα γραμμή και στήλη.

Αρχικοποίηση των εικόνων

```
for (i = 0; i < N; i++)
{
        for (j = 0; j < M; j++)
        {
                imageOriginal[i][j] = (float) ( rand() % 255);
                imageFinal[i][j] =  imageOriginal[i][j];
        }
}
```

Στην τελική εικόνα θέλουμε να έχει τα ίδια εικονοστοιχεία στα άκρα.
Αρχικοποιούμε και αυτόν τον πίνακα για να μεταφερθεί στην κρυφή μνήμη.

Ο τελικός αλγόριθμος που θέλουμε να χρονομετρήσουμε

```
for (i = 1; i < N-1; i++)
{
        for (j = 1; j < M-1; j++)
        {
                imageFinal[i][j] = K[0][0]*imageOriginal[i-1][j-1] +
                K[0][1]*imageOriginal[i][j-1] +
                K[0][2]*imageOriginal[i+1][j-1] +
                K[1][0]*imageOriginal[i-1][j] +
                K[1][1]*imageOriginal[i][j] +
                K[1][2]*imageOriginal[i+1][j] +
                K[2][0]*imageOriginal[i-1][j+1] +
                K[2][1]*imageOriginal[i][j+1] +
                K[2][2]*imageOriginal[i+1][j+1] ;
        }
}
```

Στο τέλος αποδεσμεύουμε τους πίνακες

```
 for(i = 0; i < N; i++)
 {
        free(imageOriginal[i]);
        free(imageFinal[i]);
 }

 free(imageOriginal); free(imageFinal);
```

Αλγόριθμος sse.c

Δήλωση δύο μεταβλητών για τις σταθερές 0.5 και 5.0

```
   __m128 const_p05 = _mm_set1_ps(0.5f);
  __m128 const_p50 = _mm_set1_ps(5.0f);
```

Μετατροπή τους σε m128 για να μπορούν να χρησιμοποιηθούν στις εντολές SIMD.

Δέσμευση μνήμης για αρχική, τελική εικόνα ευθυγραμμισμένες στα 16 bytes.

```
imageOriginal = (float *)_aligned_malloc(N * M * sizeof(float),16);
imageFinal = (float *)_aligned_malloc(N * M * sizeof(float),16);
```

Χρησιμοποιήθηκε η _aligned_malloc αντί της posix_memalign λόγω ότι τα προγράμματα υλοποιήθηκαν σε Windows 7.

Αποθήκευση αρχικού δείκτη για την τελική εικόνα, ώστε όταν την αναγνώσουμε να ξεκινήσουμε από την αρχή.

```
float* screen = imageFinal + g_image_width*1;
```

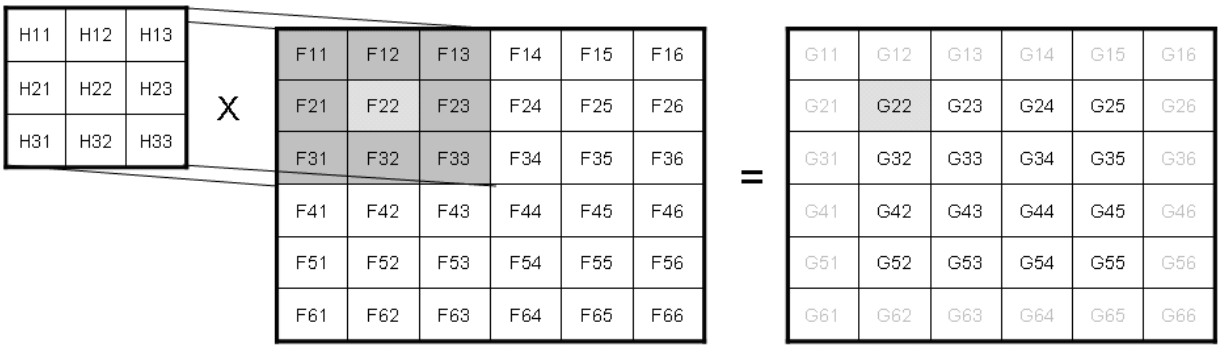
Αρχικοποίηση πινάκων

```
for (i = 0; i < N; i++)
        {
                for (j = 0; j < M; j++)
                {
                        *(imageOriginal + i*M + j) = (float) ( rand() % 255);
                        *(imageFinal + i*M + j) = *(imageOriginal + i*M + j);
                }
        }
```

Στο σημείο αυτό εισάγουμε τρείς δείκτες για κάθε γραμμή του πίνακα F, για το κεντρικό pixel, όπως φαίνεται και στο Σχήμα 2, για το F22.

```
float* image_0 = imageOriginal + g_image_width * 0;
float* image_1 = imageOriginal + g_image_width * 1;
float* image_2 = imageOriginal + g_image_width * 2;
```

Την πρώτη φορά θα ξεκινήσουμε από το F22, και οι δείκτες (η αρχή) των image_0, image_1, image_2 θα είναι F11,F21,F31 αντίστοιχα.
Σκοπός μας είναι να εκτελέσουμε τον μετασχηματισμό παράλληλα για τα F22,F23,F24,F25 και ούτω καθεξής.



Σχήμα 2

Αρχικοποιώ τις μεταβλητές με 0.0.

```
__m128 current_0 = _mm_set1_ps(0.0f);
__m128 current_1 = _mm_set1_ps(0.0f);
__m128 current_2 = _mm_set1_ps(0.0f);
```

Σε αυτές τις μεταβλητές θα τοποθετήσουμε τα στοιχεία της κάθε γραμμής σε πίνακα 4 float στοιχείων, δηλ τύπου __m128.
Το x λαμβάνει τιμές ανά 4 (pixels).

```
current_0 = _mm_loadu_ps(image_0+x-1);
current_1 = _mm_loadu_ps(image_1+x-1);
current_2 = _mm_loadu_ps(image_2+x-1);
```

Ετοιμάζουμε τα pixels του πίνακα F11-F33 (Σχήμα 2) και λόγω του __m128 θα έχουμε παράλληλα και τα F12 με F34, F13 με F35, F14 με F36

```
__m128 image_00 = current_0;
__m128 image_01 =  _mm_set1_ps(current_0[1]);
__m128 image_02 = _mm_set1_ps(current_0[2]);
__m128 image_10 = current_1;
__m128 image_11 = _mm_set1_ps(current_1[1]);
__m128 image_12 = _mm_set1_ps(current_1[2]);
__m128 image_20 = current_2;
__m128 image_21 = _mm_set1_ps(current_2[1]);
__m128 image_22 = _mm_set1_ps(current_2[2]);
```

Η εντολή σε SIMD που αναπαριστά την κλασική εντολή
newP4 = P0*K0+P1*K1+P2*K2+P3*K3+P4*K4+P5*K5+P6*K6+P7*K7+P8*K8

```
__m128 result = _mm_add_ps( _mm_mul_ps(image_00,const_p05),
_mm_add_ps( _mm_mul_ps(image_01,const_p05),
_mm_add_ps( _mm_mul_ps(image_02,const_p05),
_mm_add_ps( _mm_mul_ps(image_10,const_p05),
_mm_add_ps( _mm_mul_ps(image_11,const_p50),
_mm_add_ps( _mm_mul_ps(image_12,const_p05),
_mm_add_ps( _mm_mul_ps(image_20,const_p05),
_mm_add_ps( _mm_mul_ps(image_21,const_p05),
_mm_mul_ps(image_22,const_p05)))))))));
```

Οι εντολές _mm_add_ps και _mm_mul_ps είναι γνωστές για την παράλληλη πρόσθεση και πολλαπλασιασμό που εκτελούν 4 floats σε ένα aligned πίνακα.

Σώζουμε το αποτέλεσμα κάθε φορά στο αντίστοιχο κομμάτι που κάναμε επεξεργασία
_mm_storeu_ps((imageFinal+x),result);
Προσδιορίζοντας το με το x.

Στο τέλος της κάθε γραμμής μετακινούμαστε την επόμενη γραμμή με

```
image_0 += g_image_width;
image_1 +=  g_image_width;
image_2 +=  g_image_width;
imageFinal +=  g_image_width;
```

Στο τέλος αν θέλουμε να εκτυπώσουμε κάποια από τα στοιχεί του νέου πίνακα θα πρέπει να χρησιμοποιήσουμε τον screen δείκτη.

```
for (i = 0; i < 6; i++)
        {
                for (j = 0; j < 6; j++)
                {
                        printf("%2d,%2d -  %2.2f \n",i,j,*(screen + i*M + j));
                }
        }
```

Αποδέσμευση πίνακα
_aligned_free(imageOriginal);

Ένα πρόβλημα που συνάντησα είναι ότι το πρόγραμμα διακόπτεται με error όταν εκτελώ την εντολή _aligned_free(imageFinal);.

Αποτελέσματα

Τα αποτελέσματα μόνο συγκριτικά θα μπορούσαν να αξιολογηθούν.
Τα μεμονωμένα αποτελέσματα εξαρτώνται από τον κάθε υπολογιστή που χρησιμοποιείται.
Επίσης έχει αναφερθεί ότι η εντολή gettimeofday επηρεάζεται από τις υπόλοιπες λειτουργίες του λειτουργικού συστήματος.
Για καλύτερα αποτελέσματα πρέπει να πραγματοποιηθούν πολλά πειράματα και με ίδια ακριβώς δεδομένα εικόνας και να αποδοθούν οι μέσοι όροι.
Οπότε αναφέρουμε ότι το acceleration ratio ή αλλιώς το speedup factor είναι

speedup factor = (execution time sse) / (execution time no-sse)
= 6000/4000 = 1.5

Αρα έχουμε περίπου 1.5 φορές ταχύτερη εκτέλεση πράξεων μετασχηματισμού.