



ΕΘΝΙΚΟ ΜΕΤΣΟΒΕΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ
ΥΠΟΛΟΓΙΣΤΩΝ

Τομέας Τεχνολογίας Υπολογιστών και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

Σχεδιασμός Ενσωματωμένων Συστημάτων
9^ο Εξάμηνο ΗΜΜΥ

1η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΩΝ ΓΙΑ ΧΑΜΗΛΗ ΚΑΤΑΝΑΛΩΣΗ ΕΝΕΡΓΕΙΑΣ ΚΑΙ ΥΨΗΛΗ ΑΠΟΔΟΣΗ

Ημ/νια Παράδοσης : 15/11/2024

Ομάδα 22

Παναγιώτης Μπέλσης AM : 03120874

Θεοδώρα Εξάρχου AM : 03120865

1.2 Ζητούμενο 1^ο

Renesas S7G2 Microcontroller

Table 1.15 Functional comparison

		Part numbers					
Function		R7FS7G27H2A01CBD/ R7FS7G27G2A01CBD	R7FS7G27H2A01CBG/ R7FS7G27G2A01CBG	R7FS7G27H3A01CFC/ R7FS7G27G3A01CFC	R7FS7G27H2A01CLK/ R7FS7G27G2A01CLK	R7FS7G27H3A01CFB/ R7FS7G27G3A01CFB	R7FS7G27G3A01CFP
Pin count		224	176	176	145	144	100
Package		BGA	BGA	LQFP	LGA	LQFP	LQFP
Code flash memory		4/3 MB					3 MB
Data flash memory		64 KB					
SRAM		640 KB					
	Parity	608 KB					
	DED	32 KB					
Standby SRAM		8 KB					
System	CPU clock	240 MHz					
	Backup registers	512 bytes					

1.

Μέγεθος Flash μνήμης : up to 4-MB code flash memory, 64-KB data flash memory

Μέγεθος RAM μνήμης : 640-KB SRAM

2.

Οι **κρυφές μνήμες** και λειτουργίες είναι:

55.3 Flash Cache

55.3.1 Overview

The flash cache (FCACHE) speeds up read access from the bus master to the flash memory. The FCACHE includes:

- FCACHE1, for CPU instruction fetches
- FCACHE2, for CPU operand access and DMA
- FLPF, for the prefetch access in CPU instruction fetches.

Table 55.3 Flash cache overview

Parameter	Flash cache 1 (FCACHE1)	Flash cache 2 (FCACHE2)	Prefetch Buffer (FLPF)
Cache target region	0000 0000h - 003F FFFFh	0000 0000h - 003F FFFFh	0000 0000h - 003F FFFFh
Target bus master	CPU instruction fetch	CPU Operand Access and Access from other than CPU	FLPF
Capacity	256 bytes	16 bytes	32 bytes
Associativity	<ul style="list-style-type: none">• 8-way set associative• 128 bits/entry (128-bit aligned data)• 2 entries/way	<ul style="list-style-type: none">• Fully associative• 128 bits/entry (128-bit aligned data)• 1 entry for FCACHE2	<ul style="list-style-type: none">• Fully associative• 128 bits/entry (128-bit aligned data)• 2 entries
Access cycles	<ul style="list-style-type: none">• Cache hit: 0 waits• Cache miss: Number of waits set in Flash Wait Cycle Register	<ul style="list-style-type: none">• Cache hit: 0 waits• Cache miss: Number of waits set in Flash Wait Cycle Register	<ul style="list-style-type: none">• Cache hit: 0 waits• Cache miss: Number of waits set in Flash Wait Cycle Register

Flash Cache (FCACHE): Μνήμη cache για βελτίωση της απόδοσης.
Η FCACHE περιλαμβάνει την FCACHE1, την FCACHE2 και την FLPF.

3.

Συχνότητα ρολογιού στην οποία θα προγραμματιστεί το board (απο το e2studio ή μέσω του debugger βάζοντας ένα watch expression στο SystemCoreClock) : 240000000 Hz(240 MHz)

Expression	Type	Value
clock_frequency	volatile uint32_t	240000000
Add new expression		

```
// Code to initialize the DWT->CYCCNT register
CoreDebug->DEMCR |= 0x01000000;
ITM->LAR = 0xC5ACCE55;
DWT->CYCCNT = 0;
DWT->CTRL |= 1;
/* Initialize your variables here */
clock_frequency = SystemCoreClock;

⚠ unused variable 'j' [-Wunused-variable] | ⚠ unused variable 'i' [-Wunused-variable]
int motion_vectors_x[N/B][M/B], motion_vectors_y[N/B][M/B], i, j;

⚠ declaration of 'i' shadows a previous local [-Wshadow]
for(int i=0; i<1; i++){
    /* Add timer code here */
    ⚠ passing argument 2 of 'phods_motion_estimation' from incompatible pointer type
    phods_motion_estimation(current, previous, motion_vectors_x, motion_vectors_y);
}
breakpoint relocated to: hal_entry.c [line: 183]
```

1.2.2

Μετασχηματίσαμε τον κώδικα ώστε να μετράται ο χρόνος που χρειάζεται η συνάρτηση `phods_motion_estimation` για να εκτελεστεί.

Για εκτέλεση 1 φορά του Phods :

```
}
⚠ unused variable 'time' [-Wunused-variable]
float time=(float)cycle[0] / (float)clock_frequency;
⚠ Breakpoint relocated to: hal_entry.c [line: 187]
// printf("Time (second): %f\n", time);

while(1){
}
```

Expression	Type	Value	Address
clock_frequency	volatile uint32_t	240000000	0x1ffe01a0
time	float	0.00185546256	0x1ffe1a38
Add new expression			

Για εκτέλεση 10 φορές του Phods :

Expression	Type	Value	Address
clock_frequency	volatile uint32_t	240000000	0x1ffe01a0
> time	float [10]	0x1ffe19e0 <g_main_stack+19...	0x1ffe19e0
avgtime	float	0.0018554593	0x1ffe1a28
maxtime	float	0.00185546663	0x1ffe1a38
mintime	float	0.00185545837	0x1ffe1a30

Το κομμάτι του κώδικα που αλλάξαμε έτσι ώστε να εκτελούμε 10 φορές και να μετράμε το μέσο όρο, μέγιστο και ελάχιστο χρόνο εκτέλεσης.

```
void hal_entry(void) {
    // Code to initialize the DWT->CYCCNT register
    CoreDebug->DEMCR |= 0x01000000;
    ITM->LAR = 0xC5ACCE55;
    DWT->CYCCNT = 0;
    DWT->CTRL |= 1;
    /* Initialize your variables here */
    clock_frequency = SystemCoreClock;
    int motion_vectors_x[N/B][M/B], motion_vectors_y[N/B][M/B], i, j;
    float time[10];
    uint32_t cycle[10];
    for(int i=0; i<10; i++){
        /* Add timer code here */
        DWT->CYCCNT = 0;
        phods_motion_estimation(current, previous, motion_vectors_x, motion_vectors_y);
        cycle[i]=DWT->CYCCNT;
        time[i]=(float)cycle[i] / (float)clock_frequency;
    }
    float maxtime=0;
    float timecounter=0;
    float mintime=time[0];
    for(int i=0; i<10; i++){
        if(time[i]>maxtime)
            maxtime=time[i];
        if(time[i]<mintime)
            mintime=time[i];
        timecounter+=time[i];
    }
    float avgtime=timecounter/10;
    while(1){
    }
}
```

1.2.3

Εφαρμόσαμε τους εξής μετασχηματισμούς στον κώδικα ώστε να μειωθεί ο χρόνος εκτέλεσης.

1. Loop Fusion
2. Loop Unrolling for X-Y dimensions (distx, disty calculations)
3. Loop Fusion & Loop Unrolling
4. Loop Fusion + Computation in the Outer Loop

Παρακάτω θα αναλύσουμε τον κάθε μετασχηματισμό και θα παραθέσουμε μόνο τα σημεία που άλλαξαν στον κώδικα.

1) Loop Fusion

Χρόνοι για 10 εκτελέσεις:

Expression	Type	Value	Address
clock_frequency volatile u...		240000000	0x1ffe01...
> time	float [10]	0x1ffe19e0 <g_main_stack+...	0x1ffe19...
avgtime	float	0.0016536027	0x1ffe1a...
maxtime	float	0.00165362505	0x1ffe1a...
mintime	float	0.00165360002	0x1ffe1a...
Add new exp...			

Με την τεχνική του Loop Fusion συγχωνεύουμε τους υπολογισμούς των distx και disty στον ίδιο βρόχο. Οι τιμές p1, p2, q2 είναι ίδιες και για τους 2 υπολογισμούς, οπότε με την συγχώνευση πετυχαίνουμε καλύτερη αξιοποίηση της χωρικής τοπικότητας αφού ανακτώνται και χρησιμοποιούνται στο ίδιο iteration.

αλλαγές στον κώδικα:

```
for(i = -S; i < S + 1; i += S)
{
    distx = 0;
    disty = 0;
    // printf("Processing i = %d\n", i); // Debug output
    /* For all pixels in the block */
    for(k = 0; k < B; k++)
    {
        for(l = 0; l < B; l++)
        {
            p1 = current[B*x + k][B*y + l];
            // Calculate disty
            if((B*x + vectors_x[x][y] + k) < 0 ||
               (B*x + vectors_x[x][y] + k) > (N-1) ||
               (B*y + vectors_y[x][y] + i + 1) < 0 ||
               (B*y + vectors_y[x][y] + i + 1) > (M-1))
            {
                q2 = 0;
            }
            else
            {
                q2 = previous[B*x + vectors_x[x][y] + k][B*y + vectors_y[x][y] + i + 1];
            }
            if(p1 - q2 > 0)
            {
                disty += p1 - q2;
            }
        }
    }
}
```

```






else
{
    disty += q2 - p1;
}
// Calculate distx
if((B*x + vectors_x[x][y] + i + k) < 0 ||
    (B*x + vectors_x[x][y] + i + k) > (N-1) ||
    (B*y + vectors_y[x][y] + 1) < 0 ||
    (B*y + vectors_y[x][y] + 1) > (M-1))
{
    p2 = 0;
}
else
{
    p2 = previous[B*x + vectors_x[x][y] + i + k][B*y + vectors_y[x][y] + 1];

    if(p1 - p2 > 0)
    {
        distx += p1 - p2;
    }
    else
    {
        distx += p2 - p1;
    }
}
}
// Check if the current disty is the minimum
if(disty < min2)
{
    min2 = disty;
    besty = i;
}
// Check if the current distx is the minimum
if(distx < min1)
{
    min1 = distx;
    bestx = i;
}
}

```

2) Loop Unrolling for X-Y dimensions

Χρόνοι για 10 εκτελέσεις:

>  time	float [10]	0x1ffe19e0 <g_main_stack+...	0x1ffe19...
 avgtime	float	0.00179380155	0x1ffe1a...
 maxtime	float	0.00179381669	0x1ffe1a...
 mintime	float	0.00179380004	0x1ffe1a...
 Add new exp			

Με τη χρήση της τεχνικής loop unrolling αυξάνουμε την απόδοση των βρόχων με τη μείωση του αριθμού των επαναλήψεων και την εκτέλεση πολλαπλών υπολογισμών μέσα σε κάθε επανάληψη.

αλλαγές στον κώδικα

```

#include "hal_data.h"
#include "stdio.h"
#include "string.h"
#include "images.h"
#define N 10 /*Frame dimension for QCIF format*/
#define M 10 /*Frame dimension for QCIF format*/
#define B 5 /*Block size*/
#define p 7 /*Search space. Restricted in a [-p,p] region around the
            original location of the block.*/
volatile uint32_t clock_frequency = 0;
void phods_motion_estimation(const int current[N][M], const int previous[N][M],
    int vectors_x[N/B][M/B],int vectors_y[N/B][M/B])
{
    int x, y, i, j, k, l, p1, p2, q2, distx, disty, S, min1, min2, bestx, besty;
    distx = 0;
    disty = 0;
    /*Initialize the vector motion matrices*/
    for(i=0; i<N/B; i++)
    {
        for(j=0; j<M/B; j++)
        {
            vectors_x[i][j] = 0;
            vectors_y[i][j] = 0;
        }
    }
    /*For all blocks in the current frame*/
    for(x=0; x<N/B; x++)
    {
        for(y=0; y<M/B; y++)
        {
            S = 4;
            while(S > 0)
            {
                min1 = 255*B*B;
                min2 = 255*B*B;
                /*For all candidate blocks in X dimension*/
                for(i=-S; i<S+1; i+=S)
                {
                    distx = 0;
                    /*For all pixels in the block (Loop unrolling by 2 for k and l)*/
                    for(k = 0; k < B; k += 2) { // Unrolling by 2
                        for(l = 0; l < B; l += 2) { // Unrolling by 2
                            // Process the first pair (k, l)
                            p1 = current[B*x + k][B*y + l];
                            if((B*x + vectors_x[x][y] + i + k) < 0 ||
                                (B*x + vectors_x[x][y] + i + k) > (N-1) ||
                                (B*y + vectors_y[x][y] + l) < 0 ||
                                (B*y + vectors_y[x][y] + l) > (M-1))
                            {
                                p2 = 0;
                            } else {
                                p2 = previous[B*x + vectors_x[x][y] + i + k][B*y + vectors_y[x][y] + l];
                            }
                            distx += abs(p1 - p2);
                            // Process the second pair (k+1, l)
                            if (k + 1 < B) {
                                p1 = current[B*x + k + 1][B*y + l];
                                if((B*x + vectors_x[x][y] + i + k + 1) < 0 ||
                                    (B*x + vectors_x[x][y] + i + k + 1) > (N-1) ||
                                    (B*y + vectors_y[x][y] + l) < 0 ||
                                    (B*y + vectors_y[x][y] + l) > (M-1))
                                {
                                    p2 = 0;
                                } else {
                                    p2 = previous[B*x + vectors_x[x][y] + i + k + 1][B*y + vectors_y[x][y] + l];
                                }
                                distx += abs(p1 - p2);
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

// Process the first pair (k, l+1)
if (l + 1 < B) {
    p1 = current[B*x + k][B*y + l + 1];
    if((B*x + vectors_x[x][y] + i + k) < 0 ||
        (B*x + vectors_x[x][y] + i + k) > (N-1) ||
        (B*y + vectors_y[x][y] + l + 1) < 0 ||
        (B*y + vectors_y[x][y] + l + 1) > (M-1))
    {
        p2 = 0;
    } else {
        p2 = previous[B*x + vectors_x[x][y] + i + k][B*y + vectors_y[x][y] + l + 1];
    }
    distx += abs(p1 - p2);
}
// Process the second pair (k+1, l+1)
if (k + 1 < B && l + 1 < B) {
    p1 = current[B*x + k + 1][B*y + l + 1];
    if((B*x + vectors_x[x][y] + i + k + 1) < 0 ||
        (B*x + vectors_x[x][y] + i + k + 1) > (N-1) ||
        (B*y + vectors_y[x][y] + l + 1) < 0 ||
        (B*y + vectors_y[x][y] + l + 1) > (M-1))
    {
        p2 = 0;
    } else {
        p2 = previous[B*x + vectors_x[x][y] + i + k + 1][B*y + vectors_y[x][y] + l + 1];
    }
    distx += abs(p1 - p2);
}
}
}
if(distx < min1)
{
    min1 = distx;
    bestx = i;
}
}
/*For all candidate blocks in Y dimension*/
for(i = -S; i < S + 1; i += S) {
    disty = 0;
    /* For all pixels in the block (Loop unrolling by 2 for k and l) */
    for(k = 0; k < B; k += 2) { // Unrolling by 2
        for(l = 0; l < B; l += 2) { // Unrolling by 2
            // Process the first pair (k, l)
            p1 = current[B*x + k][B*y + l];
            if((B*x + vectors_x[x][y] + k) < 0 ||
                (B*x + vectors_x[x][y] + k) > (N - 1) ||
                (B*y + vectors_y[x][y] + i + l) < 0 ||
                (B*y + vectors_y[x][y] + i + l) > (M - 1)) {
                q2 = 0;
            } else {
                q2 = previous[B*x + vectors_x[x][y] + k][B*y + vectors_y[x][y] + i + l];
            }
            disty += abs(p1 - q2);
            // Process the second pair (k+1, l)
            if(k + 1 < B) {
                p1 = current[B*x + k + 1][B*y + l];
                if((B*x + vectors_x[x][y] + k + 1) < 0 ||
                    (B*x + vectors_x[x][y] + k + 1) > (N - 1) ||
                    (B*y + vectors_y[x][y] + i + l) < 0 ||
                    (B*y + vectors_y[x][y] + i + l) > (M - 1)) {
                    q2 = 0;
                } else {
                    q2 = previous[B*x + vectors_x[x][y] + k + 1][B*y + vectors_y[x][y] + i + l];
                }
                disty += abs(p1 - q2);
            }
            // Process the first pair (k, l+1)
            if(l + 1 < B) {

```

```

        p1 = current[B*x + k][B*y + l + 1];
        if((B*x + vectors_x[x][y] + k) < 0 ||
           (B*x + vectors_x[x][y] + k) > (N - 1) ||
           (B*y + vectors_y[x][y] + i + l + 1) < 0 ||
           (B*y + vectors_y[x][y] + i + l + 1) > (M - 1)) {
            q2 = 0;
        } else {
            q2 = previous[B*x + vectors_x[x][y] + k][B*y + vectors_y[x][y] + i + l + 1];
        }
        disty += abs(p1 - q2);
    }
    // Process the second pair (k+1, l+1)
    if(k + 1 < B && l + 1 < B) {
        p1 = current[B*x + k + 1][B*y + l + 1];
        if((B*x + vectors_x[x][y] + k + 1) < 0 ||
           (B*x + vectors_x[x][y] + k + 1) > (N - 1) ||
           (B*y + vectors_y[x][y] + i + l + 1) < 0 ||
           (B*y + vectors_y[x][y] + i + l + 1) > (M - 1)) {
            q2 = 0;
        } else {
            q2 = previous[B*x + vectors_x[x][y] + k + 1][B*y + vectors_y[x][y] + i + l + 1];
        }
        disty += abs(p1 - q2);
    }
}
}
if(disty < min2) {
    min2 = disty;
    besty = i;
}
}
S = S/2;
vectors_x[x][y] += bestx;
vectors_y[x][y] += besty;
}
}
}
}

```

3) Loop Fusion & Loop Unrolling

Χρόνοι για 10 εκτελέσεις:

Expression	Type	Value	Address
(x) clock_frequency	volatile unsigned int	240000000	0x1ffe01...
> 🕒 time	float [10]	0x1ffe19e0 <g_main_stack+...	0x1ffe19...
(x) avgtime	float	0.00175783841	0x1ffe1a...
(x) maxtime	float	0.00175784586	0x1ffe1a...
(x) mintime	float	0.00175783748	0x1ffe1a...
➕ Add new expression			

Δοκιμάσαμε να εφαρμόσουμε μαζί τις 2 προηγούμενες τεχνικές, όμως παρατηρήσαμε ότι οι χρόνοι εκτέλεσης που προκύπτουν είναι χειρότεροι από το αν εφαρμόσουμε αποκλειστικά Loop Fusion.

αλλαγές στον κώδικα

```
#include "hal_data.h"
#include "stdio.h"
#include "string.h"
#include "images.h"
#define N 10 /*Frame dimension for QCIF format*/
#define M 10 /*Frame dimension for QCIF format*/
#define B 5 /*Block size*/
#define p 7 /*Search space. Restricted in a [-p,p] region around the
            original location of the block.*/
volatile uint32_t clock_frequency = 0;
void phods_motion_estimation(const int current[N][M], const int previous[N][M],
    int vectors_x[N/B][M/B],int vectors_y[N/B][M/B])
{
    int x, y, i, j, k, l, p1, p2, q2, distx, disty, S, min1, min2, bestx, besty;
    distx = 0;
    disty = 0;
    /*Initialize the vector motion matrices*/
    for(i=0; i<N/B; i++)
    {
        for(j=0; j<M/B; j++)
        {
            vectors_x[i][j] = 0;
            vectors_y[i][j] = 0;
        }
    }
    /*For all blocks in the current frame*/
    for(x=0; x<N/B; x++)
    {
        for(y=0; y<M/B; y++)
        {
            S = 4;
            while(S > 0)
            {
                min1 = 255*B*B;
                min2 = 255*B*B;
                /* For all candidate blocks in X and Y dimensions */
                for(i = -S; i < S + 1; i += S) {
                    distx = 0;
                    disty = 0;
                    /* For all pixels in the block (Loop unrolling by 2 for k and l) */
                    for(k = 0; k < B; k += 2) { // Unrolling by 2 for k
                        for(l = 0; l < B; l += 2) { // Unrolling by 2 for l
                            // ----- Calculate disty -----
                            p1 = current[B*x + k][B*y + l];
                            if((B*x + vectors_x[x][y] + k) < 0 ||
                                (B*x + vectors_x[x][y] + k) > (N - 1) ||
                                (B*y + vectors_y[x][y] + i + 1) < 0 ||
                                (B*y + vectors_y[x][y] + i + 1) > (M - 1)) {
                                    q2 = 0;
                                } else {
                                    q2 = previous[B*x + vectors_x[x][y] + k][B*y + vectors_y[x][y] + i + 1];
                                }
                            disty += abs(p1 - q2);
                            // Process the second pair (k+1, l)
                            if(k + 1 < B) {
                                p1 = current[B*x + k + 1][B*y + l];
                                if((B*x + vectors_x[x][y] + k + 1) < 0 ||
                                    (B*x + vectors_x[x][y] + k + 1) > (N - 1) ||
                                    (B*y + vectors_y[x][y] + i + 1) < 0 ||
                                    (B*y + vectors_y[x][y] + i + 1) > (M - 1)) {
                                        q2 = 0;
                                    } else {
                                        q2 = previous[B*x + vectors_x[x][y] + k + 1][B*y + vectors_y[x][y] + i + 1];
                                    }
                                disty += abs(p1 - q2);
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```

// Process the first pair (k, l+1)
if(l + 1 < B) {
    p1 = current[B*x + k][B*y + l + 1];
    if((B*x + vectors_x[x][y] + k) < 0 ||
        (B*x + vectors_x[x][y] + k) > (N - 1) ||
        (B*y + vectors_y[x][y] + i + l + 1) < 0 ||
        (B*y + vectors_y[x][y] + i + l + 1) > (M - 1)) {
        q2 = 0;
    } else {
        q2 = previous[B*x + vectors_x[x][y] + k][B*y + vectors_y[x][y] + i + l + 1];
    }
    disty += abs(p1 - q2);
}
// Process the second pair (k+1, l+1)
if(k + 1 < B && l + 1 < B) {
    p1 = current[B*x + k + 1][B*y + l + 1];
    if((B*x + vectors_x[x][y] + k + 1) < 0 ||
        (B*x + vectors_x[x][y] + k + 1) > (N - 1) ||
        (B*y + vectors_y[x][y] + i + l + 1) < 0 ||
        (B*y + vectors_y[x][y] + i + l + 1) > (M - 1)) {
        q2 = 0;
    } else {
        q2 = previous[B*x + vectors_x[x][y] + k + 1][B*y + vectors_y[x][y] + i + l + 1];
    }
    disty += abs(p1 - q2);
}
// ----- Calculate distx -----
p1 = current[B*x + k][B*y + l];
if((B*x + vectors_x[x][y] + i + k) < 0 ||
    (B*x + vectors_x[x][y] + i + k) > (N - 1) ||
    (B*y + vectors_y[x][y] + l) < 0 ||
    (B*y + vectors_y[x][y] + l) > (M - 1)) {
    p2 = 0;
} else {
    p2 = previous[B*x + vectors_x[x][y] + i + k][B*y + vectors_y[x][y] + l];
}
distx += abs(p1 - p2);
// Process the second pair (k+1, l)
if(k + 1 < B) {
    p1 = current[B*x + k + 1][B*y + l];
    if((B*x + vectors_x[x][y] + i + k + 1) < 0 ||
        (B*x + vectors_x[x][y] + i + k + 1) > (N - 1) ||
        (B*y + vectors_y[x][y] + l) < 0 ||
        (B*y + vectors_y[x][y] + l) > (M - 1)) {
        p2 = 0;
    } else {
        p2 = previous[B*x + vectors_x[x][y] + i + k + 1][B*y + vectors_y[x][y] + l];
    }
    distx += abs(p1 - p2);
}
// Process the first pair (k, l+1)
if(l + 1 < B) {
    p1 = current[B*x + k][B*y + l + 1];
    if((B*x + vectors_x[x][y] + i + k) < 0 ||
        (B*x + vectors_x[x][y] + i + k) > (N - 1) ||
        (B*y + vectors_y[x][y] + l + 1) < 0 ||
        (B*y + vectors_y[x][y] + l + 1) > (M - 1)) {
        p2 = 0;
    } else {
        p2 = previous[B*x + vectors_x[x][y] + i + k][B*y + vectors_y[x][y] + l + 1];
    }
    distx += abs(p1 - p2);
}
// Process the second pair (k+1, l+1)
if(k + 1 < B && l + 1 < B) {
    p1 = current[B*x + k + 1][B*y + l + 1];
    if((B*x + vectors_x[x][y] + i + k + 1) < 0 ||
        (B*x + vectors_x[x][y] + i + k + 1) > (N - 1) ||

```

```

        (B*y + vectors_y[x][y] + 1 + 1) < 0 ||
        (B*y + vectors_y[x][y] + 1 + 1) > (M - 1)) {
            p2 = 0;
        } else {
            p2 = previous[B*x + vectors_x[x][y] + i + k + 1][B*y + vectors_y[x][y] + 1 + 1];
        }
        distx += abs(p1 - p2);
    }
}
// Check if the current disty is the minimum
if(disty < min2) {
    min2 = disty;
    besty = i;
}
// Check if the current distx is the minimum
if(distx < min1) {
    min1 = distx;
    bestx = i;
}
}
S = S/2;
vectors_x[x][y] += bestx;
vectors_y[x][y] += besty;
}
}
}
}

```

4) Loop Fusion + Computation in the Outer Loop

Expression	Type	Value	Address
(X) clock_freq	volatile u...	240000000	0x1ffe01...
> time	float [10]	0x1ffe19e0 <g_main_stack+...	0x1ffe19...
(X) avgtime	float	0.000868709176	0x1ffe1a...
(X) maxtime	float	0.000868716685	0x1ffe1a...
(X) mintime	float	0.000868708361	0x1ffe1a...
+ Add new exp			

Loop Fusion + υπολογισμός σε πιο εξωτερικό βρόχο των παρακάτω υπολογισμών:
(έξω από τα loop που είχαν τα k,l,S)

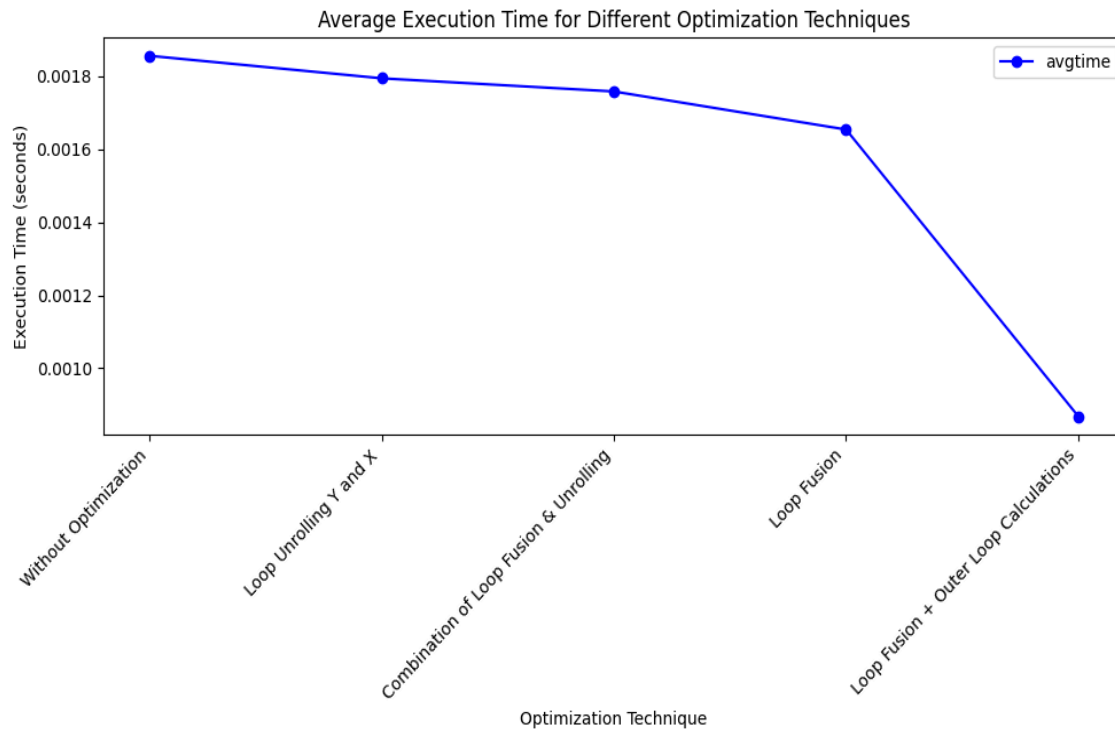
```

tempx=B*x + vectors_x[x][y];
tempy=B*y + vectors_y[x][y];

```

Με αυτόν τον τρόπο αποφεύγουμε να υπολογίζουμε κάθε φορά ένα πράγμα που επαναλαμβάνεται συνεχώς στον κώδικα. Έπειτα από αυτήν την αλλαγή παρατηρούμε μείωση του χρόνου εκτέλεσης μόνο με το loop fusion (~1.6sec) περίπου στο μισό (~0.8sec).

Διάγραμμα με κάθε optimization που εφαρμόσαμε συναρτήσει του execution time.



Παρατηρούμε ότι το καλύτερο optimization πραγματοποιήθηκε με το Loop fusion και Outer loop Calculations, με αρκετά μεγάλη απόκλιση σε σύγκριση με τους άλλους μετασχηματισμούς.

αλλαγές στον κώδικα

```
#include "hal_data.h"
#include "stdio.h"
#include "string.h"
#include "images.h"
#define N 10 /*Frame dimension for QCIF format*/
#define M 10 /*Frame dimension for QCIF format*/
#define B 5 /*Block size*/
#define p 7 /*Search space. Restricted in a [-p,p] region around the
original location of the block.*/
volatile uint32_t clock_frequency = 0;
void phods_motion_estimation(const int current[N][M], const int previous[N][M],
int vectors_x[N/B][M/B],int vectors_y[N/B][M/B])
{
int x, y, i, j, k, l, p1, p2, q2, distx, disty, S, min1, min2, bestx, besty;
distx = 0;
disty = 0;
/*Initialize the vector motion matrices*/
for(i=0; i<N/B; i++)
{
for(j=0; j<M/B; j++)
{
vectors_x[i][j] = 0;
vectors_y[i][j] = 0;
}
}
/*For all blocks in the current frame*/
for(x=0; x<N/B; x++)
{
for(y=0; y<M/B; y++)
{
```

```

S = 4;
int temp_x=B*x + vectors_x[x][y];
int temp_y=B*y + vectors_y[x][y];
while(S > 0)
{
    min1 = 255*B*B;
    min2 = 255*B*B;
    /*For all candidate blocks in X Y dimensionS*/
    for(i = -S; i < S + 1; i += S)
    {
        dist_x = 0;
        dist_y = 0;
        /* For all pixels in the block */
        for(k = 0; k < B; k++)
        {
            for(l = 0; l < B; l++)
            {
                p1 = current[B*x + k][B*y + l];
                // Calculate dist_y
                if((temp_x + k) < 0 ||
                   (temp_x + k) > (N-1) ||
                   (temp_y + i + l) < 0 ||
                   (temp_y + i + l) > (M-1))
                {
                    q2 = 0;
                }
                else
                {
                    q2 = previous[temp_x + k][temp_y + i + l];
                }
                if(p1 - q2 > 0)
                {
                    dist_y += p1 - q2;
                }
                else
                {
                    dist_y += q2 - p1;
                }
                // Calculate dist_x
                if((temp_x + i + k) < 0 ||
                   (temp_x + i + k) > (N-1) ||
                   (temp_y + l) < 0 ||
                   (temp_y + l) > (M-1))
                {
                    p2 = 0;
                }
                else
                {
                    p2 = previous[temp_x + i + k][temp_y + l];
                }
                if(p1 - p2 > 0)
                {
                    dist_x += p1 - p2;
                }
                else
                {
                    dist_x += p2 - p1;
                }
            }
        }
        // Check if the current dist_y is the minimum
        if(dist_y < min2)
        {
            min2 = dist_y;
            best_y = i;
        }
        // Check if the current dist_x is the minimum
        if(dist_x < min1)

```

```

        {
            min1 = distx;
            bestx = i;
        }
    }
    S = S/2;
    vectors_x[x][y] += bestx;
    vectors_y[x][y] += besty;
}
}
}

```

1.2.4

clock_frequency	volatile ui...	240000000	0x1ffe01...
> time	float [10]	0x1ffe19e4 <g_main_stack+...	0x1ffe19...
✓ avgtime	float [4]	0x1ffe197c <g_main_stack+...	0x1ffe19...
avgtime[0]	float	0.0015414001	0x1ffe19...
avgtime[1]	float	0.00111527485	0x1ffe19...
avgtime[2]	float	0.000975470874	0x1ffe19...
avgtime[3]	float	0.000948200002	0x1ffe19...
✓ maxtime	float [4]	0x1ffe19ac <g_main_stack+...	0x1ffe19...
maxtime[0]	float	0.00154139998	0x1ffe19...
maxtime[1]	float	0.00111527496	0x1ffe19...
maxtime[2]	float	0.000975470815	0x1ffe19...
maxtime[3]	float	0.000948200002	0x1ffe19...
✓ mintime	float [4]	0x1ffe199c <g_main_stack+...	0x1ffe19...
mintime[0]	float	0.00154139998	0x1ffe19...
mintime[1]	float	0.00111527496	0x1ffe19...
mintime[2]	float	0.000975470815	0x1ffe19...
mintime[3]	float	0.000948200002	0x1ffe19...
Add new expression			

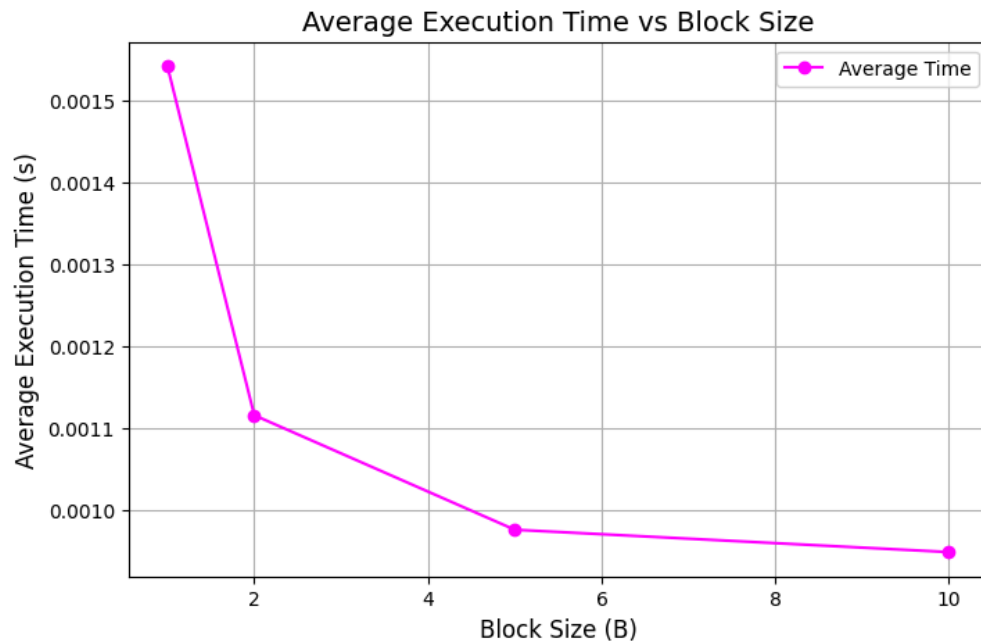
Τα παραπάνω αποτελέσματα είναι για BxB, όπου B= 1, 2, 5, 10

avgtime[0] → B=1, avgtime[1] → B=2, avgtime[2] → B=5, avgtime[3] → B=10 .

Παρατηρούμε ότι για Blocksize B=10 έχουμε τους πιο γρήγορους χρόνους. Όλο το block χωράει στην cache.

Επιπροσθέτως παρατηρούμε ότι για B=5 έχουμε ελάχιστα πιο αργό χρόνο σε σχέση με πριν (ερώτημα 1.2.3 avgtime=0.86ms), ενώ τώρα έχουμε για ίδιο size του B (avgtime=0.97ms).

Αυτή η διαφορά προέκυψε διότι καταργήσαμε το define =5 στην αρχή του κώδικα και αντι αυτού κάναμε έναν πίνακα int B[4] = {1,2,5,10}; που περιέχει τα επιθυμητά blocksizes. Επιπλέον στη συνάρτηση rhods{}, προσθέσαμε ένα ακόμη όρισμα int z. Έπειτα από δοκιμές καταλήξαμε στο ότι η καθυστέρηση του 0.1ms προέκυψε από αυτές τις αλλαγές.



αλλαγές στον κώδικα

```
#include "hal_data.h"
#include "stdio.h"
#include "string.h"
#include "images.h"
#define N 10 /*Frame dimension for QCIF format*/
#define M 10 /*Frame dimension for QCIF format*/
// #define B 5 /*Block size*/
#define p 7 /*Search space. Restricted in a [-p,p] region around the
original location of the block.*/
volatile uint32_t clock_frequency = 0;
int B[4] = {1, 2, 5, 10};
void phods_motion_estimation(int z, const int current[N][M], const int previous[N][M],
int vectors_x[N/B[z]][M/B[z]], int vectors_y[N/B[z]][M/B[z]])
{
int x, y, i, j, k, l, p1, p2, q2, distx, disty, S, min1, min2, bestx, besty;
distx = 0;
disty = 0;
// int B=B[z];
/*Initialize the vector motion matrices*/
for(i=0; i<N/B[z]; i++)
{
for(j=0; j<M/B[z]; j++)
{
vectors_x[i][j] = 0;
vectors_y[i][j] = 0;
}
}
/*For all blocks in the current frame*/
for(x=0; x<N/B[z]; x++)
{
for(y=0; y<M/B[z]; y++)
{
S = 4;
int temp_x=B[z]*x + vectors_x[x][y];
int temp_y=B[z]*y + vectors_y[x][y];
while(S > 0)
{
min1 = 255*B[z]*B[z];
min2 = 255*B[z]*B[z];
/*For all candidate blocks in X Y dimensionS*/
for(i = -S; i < S + 1; i += S)
```

```

{
    distx = 0;
    disty = 0;
    /* For all pixels in the block */
    for(k = 0; k < B[z]; k++)
    {
        for(l = 0; l < B[z]; l++)
        {
            p1 = current[B[z]*x + k][B[z]*y + l];
            // Calculate disty
            if((tempx + k) < 0 ||
               (tempx + k) > (N-1) ||
               (tempy + i + 1) < 0 ||
               (tempy + i + 1) > (M-1))
            {
                q2 = 0;
            }
            else
            {
                q2 = previous[tempx + k][tempy + i + 1];
            }

            if(p1 - q2 > 0)
            {
                disty += p1 - q2;
            }
            else
            {
                disty += q2 - p1;
            }
            // Calculate distx
            if((tempx + i + k) < 0 ||
               (tempx + i + k) > (N-1) ||
               (tempy + l) < 0 ||
               (tempy + l) > (M-1))
            {
                p2 = 0;
            }
            else
            {
                p2 = previous[tempx + i + k][tempy + l];
            }

            if(p1 - p2 > 0)
            {
                distx += p1 - p2;
            }
            else
            {
                distx += p2 - p1;
            }
        }
    }
    // Check if the current disty is the minimum
    if(disty < min2)
    {
        min2 = disty;
        besty = i;
    }
    // Check if the current distx is the minimum
    if(distx < min1)
    {
        min1 = distx;
        bestx = i;
    }
}
S = S/2;
vectors_x[x][y] += bestx;
vectors_y[x][y] += besty;
}

```



```

    }
}
}
/*****
*****/**
 * @brief Blinky example application
 *
 * Blinks all leds at a rate of 1 second using the software delay function provided by the BSP.
 * Only references two other modules including the BSP, IOPORT.
 *
 *****/
void hal_entry(void) {
    // Code to initialize the DWT->CYCCNT register
    CoreDebug->DEMCR |= 0x01000000;
    ITM->LAR = 0xC5ACCE55;
    DWT->CYCCNT = 0;
    DWT->CTRL |= 1;
    /* Initialize your variables here */
    clock_frequency = SystemCoreClock;
    float time[10];
    uint32_t cycle[10];
    float maxtime[4]={0,0,0,0};
    float mintime[4]={0,0,0,0};
    float timecounter[4]={0,0,0,0};
    float avgtime[4]={0,0,0,0};
    for(int z=0; z<4; z++){
        int motion_vectors_x[N/B[z]][M/B[z]],motion_vectors_y[N/B[z]][M/B[z]];
        for(int i=0; i<10; i++){
            /* Add timer code here */
            DWT->CYCCNT = 0;
            phods_motion_estimation(z,current,previous,motion_vectors_x,motion_vectors_y);
            cycle[i]=DWT->CYCCNT;
            time[i]=(float)cycle[i] / (float)clock_frequency;
        }
        mintime[z]=time[0];
        for(int i=0; i<10; i++){
            if(time[i]>maxtime[z])
                maxtime[z]=time[i];
            if(time[i]<mintime[z])
                mintime[z]=time[i];
            timecounter[z]+=time[i];
        }
        avgtime[z]=timecounter[z]/10;
    }
    while(1){
    }
}

```

1.2.5

AVGTIME

Expression	Type	Value
(*)= clock_frequency	volatile ui...	240000000
> time	float [10]	0x1ffe19f4 <g_main_stack+...
▼ avgtime	float [16]	0x1ffe18cc <g_main_stack+...
(*)= avgtime[0]	float	0.00156053412
(*)= avgtime[1]	float	0.00124843756
(*)= avgtime[2]	float	0.00106071238
(*)= avgtime[3]	float	0.000998037634
(*)= avgtime[4]	float	0.00131487485
(*)= avgtime[5]	float	0.00112412905
(*)= avgtime[6]	float	0.00100961246
(*)= avgtime[7]	float	0.000971420784
(*)= avgtime[8]	float	0.00116667908
(*)= avgtime[9]	float	0.0010492875
(*)= avgtime[10]	float	0.000978879281
(*)= avgtime[11]	float	0.000955420837
(*)= avgtime[12]	float	0.00111737091
(*)= avgtime[13]	float	0.0010244291
(*)= avgtime[14]	float	0.00096866244
(*)= avgtime[15]	float	0.000950087502
> maxtime	float [16]	0x1ffe198c <g_main_stack+...
> mintime	float [16]	0x1ffe194c <g_main_stack+...
+ Add new expression		

MAXTIME

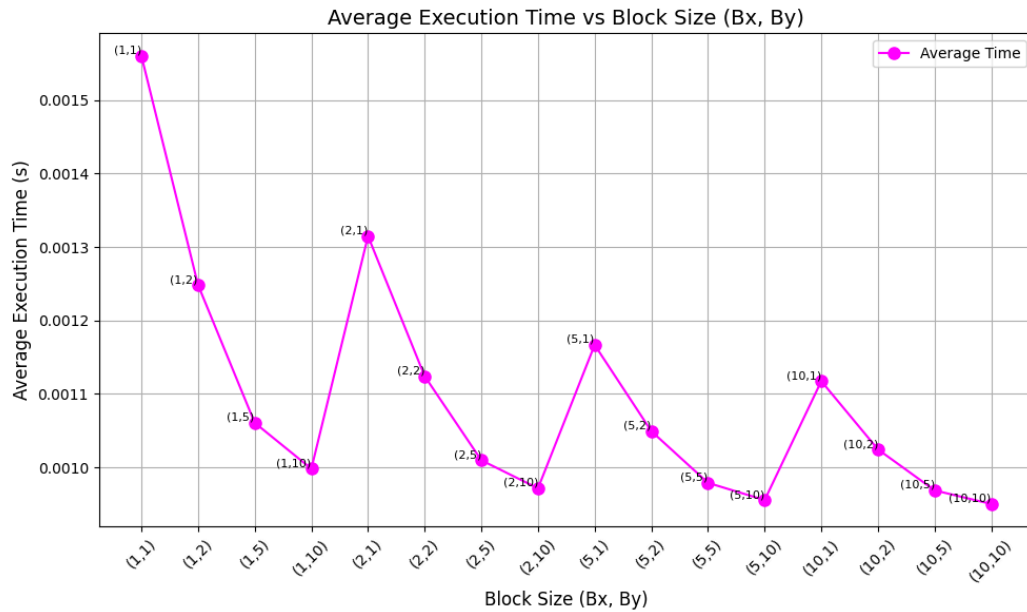
(*)= clock_frequency	volatile ui...	240000000
> time	float [10]	0x1ffe19f4 <g_main_stack+...
> avgtime	float [16]	0x1ffe18cc <g_main_stack+...
▼ maxtime	float [16]	0x1ffe198c <g_main_stack+...
(*)= maxtime[0]	float	0.00156054168
(*)= maxtime[1]	float	0.00124843745
(*)= maxtime[2]	float	0.0010607125
(*)= maxtime[3]	float	0.000998037518
(*)= maxtime[4]	float	0.00131487497
(*)= maxtime[5]	float	0.00112412916
(*)= maxtime[6]	float	0.00100961246
(*)= maxtime[7]	float	0.000971420843
(*)= maxtime[8]	float	0.0011666792
(*)= maxtime[9]	float	0.0010492875
(*)= maxtime[10]	float	0.000978879165
(*)= maxtime[11]	float	0.000955420837
(*)= maxtime[12]	float	0.00111737079
(*)= maxtime[13]	float	0.00102442922
(*)= maxtime[14]	float	0.000968662498
(*)= maxtime[15]	float	0.000950087502
> mintime	float [16]	0x1ffe194c <g_main_stack+...
+ Add new expression		

MINTIME

(*)= clock_frequency	volatile ui...	240000000
> time	float [10]	0x1ffe19f4 <g_main_stack+...
> avgtime	float [16]	0x1ffe18cc <g_main_stack+...
> maxtime	float [16]	0x1ffe198c <g_main_stack+...
▼ mintime	float [16]	0x1ffe194c <g_main_stack+...
(*)= mintime[0]	float	0.0015605333
(*)= mintime[1]	float	0.00124843745
(*)= mintime[2]	float	0.0010607125
(*)= mintime[3]	float	0.000998037518
(*)= mintime[4]	float	0.00131487497
(*)= mintime[5]	float	0.00112412916
(*)= mintime[6]	float	0.00100961246
(*)= mintime[7]	float	0.000971420843
(*)= mintime[8]	float	0.0011666792
(*)= mintime[9]	float	0.0010492875
(*)= mintime[10]	float	0.000978879165
(*)= mintime[11]	float	0.000955420837
(*)= mintime[12]	float	0.00111737079
(*)= mintime[13]	float	0.00102442922
(*)= mintime[14]	float	0.000968662498
(*)= mintime[15]	float	0.000950087502
+ Add new expression		

Mapping για τα avgtime, mintime, maxtime

[0] → Bx=1, By=1		[5] → Bx=2, By=2		[10] → Bx=5, By=5
[1] → Bx=1, By=2		[6] → Bx=2, By=5		[11] → Bx=5, By=10
[2] → Bx=1, By=5		[7] → Bx=2, By=10		[12] → Bx=10, By=1
[3] → Bx=1, By=10		[8] → Bx=5, By=1		[13] → Bx=10, By=2
[4] → Bx=2, By=1		[9] → Bx=5, By=2		[14] → Bx=10, By=5
				[15] → Bx=10, By=10



Για την εύρεση του βέλτιστου μεγέθους μπλοκ, θεωρούμε ορθογώνιο μπλοκ διαστάσεων B_x και B_y , όπου τα B_x, B_y είναι διαιρέτες των N, M . Η μέθοδος που χρησιμοποιείται για την εύρεση του βέλτιστου ζεύγους B_x, B_y είναι η εξαντλητική αναζήτηση, η οποία εξετάζει όλους τους δυνατούς συνδυασμούς για να βρει εκείνον που ελαχιστοποιεί τον χρόνο εκτέλεσης του προγράμματος. Από την αναζήτηση αυτή, καταλήξαμε στο συμπέρασμα ότι ο συνδυασμός που μας δίνει τον βέλτιστο χρόνο εκτέλεσης είναι $B_x=10$ και $B_y=10$, όπως φαίνεται και στο διάγραμμα. Δηλαδή, το μέγεθος του μπλοκ που καλύπτει την καλύτερη απόδοση είναι ένα ορθογώνιο μπλοκ με διαστάσεις 10×10 , το οποίο ταυτίζεται με το Frame dimension N και M αντίστοιχα. Οι επόμενοι καλύτεροι χρόνοι είναι οι συνδυασμοί με διαστάσεις $5 \times 10, 10 \times 5, 5 \times 5, 2 \times 10$.

αλλαγές στον κώδικα

```
#include "hal_data.h"
#include "stdio.h"
#include "string.h"
#include "images.h"
#define N 10 /*Frame dimension for QCIF format*/
#define M 10 /*Frame dimension for QCIF format*/
#define p 7 /*Search space. Restricted in a [-p,p] region around the original location of the block.*/
volatile uint32_t clock_frequency = 0;
// Asymmetric block sizes for height (Bx) and width (By)
int Bx[4] = {1, 2, 5, 10}; // Block heights
int By[4] = {1, 2, 5, 10}; // Block widths
void phods_motion_estimation(int z1,int z2, const int current[N][M], const int previous[N][M],
    int vectors_x[N/Bx[z1]][M/By[z2]], int vectors_y[N/Bx[z1]][M/By[z2]])
{
    int x, y, i, j, k, l, p1, p2, q2, distx, disty, S, min1, min2, bestx, besty;
    distx = 0;
    disty = 0;
    /*Initialize the vector motion matrices*/
    for(i = 0; i < N / Bx[z1]; i++)
    {
        for(j = 0; j < M / By[z2]; j++)
        {
            vectors_x[i][j] = 0;
            vectors_y[i][j] = 0;
        }
    }
}
```

```

}
/*For all blocks in the current frame*/
for(x = 0; x < N / Bx[z1]; x++)
{
    for(y = 0; y < M / By[z2]; y++)
    {
        S = 4;
        int tempx = Bx[z1] * x + vectors_x[x][y];
        int tempy = By[z2] * y + vectors_y[x][y];
        while(S > 0)
        {
            min1 = 255 * Bx[z1] * By[z2];
            min2 = 255 * Bx[z1] * By[z2];
            /*For all candidate blocks in X Y dimensions*/
            for(i = -S; i < S + 1; i += S)
            {
                distx = 0;
                disty = 0;
                /* For all pixels in the block */
                for(k = 0; k < Bx[z1]; k++)
                {
                    for(l = 0; l < By[z2]; l++)
                    {
                        p1 = current[Bx[z1] * x + k][By[z2] * y + l];
                        // Calculate disty
                        if((tempx + k) < 0 ||
                           (tempx + k) > (N - 1) ||
                           (tempy + i + 1) < 0 ||
                           (tempy + i + 1) > (M - 1))
                        {
                            q2 = 0;
                        }
                        else
                        {
                            q2 = previous[tempx + k][tempy + i + 1];
                        }
                        if(p1 - q2 > 0)
                        {
                            disty += p1 - q2;
                        }
                        else
                        {
                            disty += q2 - p1;
                        }
                        // Calculate distx
                        if((tempx + i + k) < 0 ||
                           (tempx + i + k) > (N - 1) ||
                           (tempy + l) < 0 ||
                           (tempy + l) > (M - 1))
                        {
                            p2 = 0;
                        }
                        else
                        {
                            p2 = previous[tempx + i + k][tempy + l];
                        }
                        if(p1 - p2 > 0)
                        {
                            distx += p1 - p2;
                        }
                        else
                        {
                            distx += p2 - p1;
                        }
                    }
                }
                // Check if the current disty is the minimum
                if(disty < min2)

```

```

        {
            min2 = disty;
            besty = i;
        }
        // Check if the current distx is the minimum
        if(distx < min1)
        {
            min1 = distx;
            bestx = i;
        }
    }
    S = S / 2;
    vectors_x[x][y] += bestx;
    vectors_y[x][y] += besty;
}
}
}
}

void hal_entry(void) {
    // Code to initialize the DWT->CYCCNT register
    CoreDebug->DEMCR |= 0x01000000;
    ITM->LAR = 0xC5ACCE55;
    DWT->CYCCNT = 0;
    DWT->CTRL |= 1;
    /* Initialize your variables here */
    clock_frequency = SystemCoreClock;
    float time[10]; // Store the times for each trial
    uint32_t cycle[10]; // Store the cycle counts for each trial
    // 1D arrays for storing the max, min, average times (flattened)
    float maxtime[16] = {0}; // Store the max time for each combination of Bx, By
    float mintime[16] = {100}; // Store the min time for each combination of Bx, By
    float timecounter[16] = {0}; // Store the total time for each combination
    float avgtime[16] = {0}; // Store the average time for each combination
    // Block height and width combinations
    int Bx[4] = {1, 2, 5, 10}; // Block heights
    int By[4] = {1, 2, 5, 10}; // Block widths
    int i=0;
    // Iterate through all combinations of Bx and By (16 combinations)
    for (int z1 = 0; z1 < 4; z1++) {
        for (int z2 = 0; z2 < 4; z2++) {
            int motion_vectors_x[N / Bx[z1]][M / By[z2]], motion_vectors_y[N / Bx[z1]][M / By[z2]];
            // Loop for 10 trials to calculate the timings
            for (i = 0; i < 10; i++) {
                /* Add timer code here */
                DWT->CYCCNT = 0;
                phods_motion_estimation(z1,z2, current, previous, motion_vectors_x, motion_vectors_y);
                cycle[i] = DWT->CYCCNT;
                time[i] = (float)cycle[i] / (float)clock_frequency;
            }
            mintime[z1 * 4 + z2] = time[0];
            maxtime[z1 * 4 + z2] = time[0];
            // Calculate the min, max, and total time for this combination
            for (i = 0; i < 10; i++) {
                if (time[i] > maxtime[z1 * 4 + z2]) {
                    maxtime[z1 * 4 + z2] = time[i];
                }
                if (time[i] < mintime[z1 * 4 + z2]) {
                    mintime[z1 * 4 + z2] = time[i];
                }
                timecounter[z1 * 4 + z2] += time[i];
            }
            // Calculate the average time
            avgtime[z1 * 4 + z2] = timecounter[z1 * 4 + z2] / 10;
        }
    }
}
while (1) {
}
}

```

1.3 Ζητούμενο 2ο

Βελτιστοποίηση της εφαρμογή *tables.c*

/gem5/tables_UF/tables.c

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i;
    int N = 100000;
    double *x1, *x2, *x3, *y;
    double a1 = 0.5;
    double a2 = 1;
    double a3 = 1.5;

    x1 = (double*) malloc(N * sizeof(double));
    x2 = (double*) malloc (N * sizeof(double));
    x3 = (double*) malloc (N * sizeof(double));
    y = (double*) malloc (N * sizeof(double));

    //Do not modify this loop
    for (i=0; i<=N-1; i++)
    {
        x1[i] = (double) i * 0.5;
        x2[i] = (double) i * 0.8;
        x3[i] = (double) i * 0.2;
        y[i] = 0;
    }

    for (i=0; i<=N-1; i++)
    {
        y[i] = y[i] + a1*x1[i] + a2*x2[i] + a3*x3[i];
    }

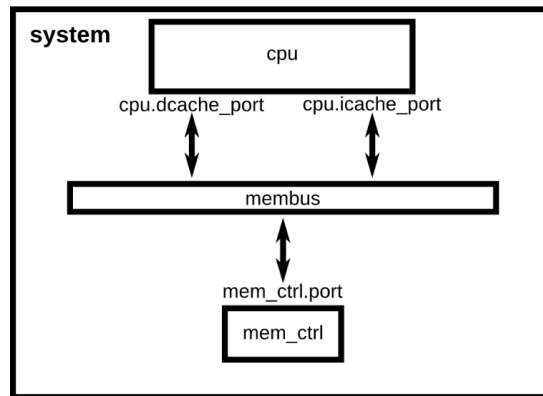
    return 0;
}
```

Καλούμαστε να εκτελέσουμε την παραπάνω εφαρμογή σε δύο διαφορετικές αρχιτεκτονικές x86 με τη βοήθεια του προσομοιωτή Gem5.

1. Η πρώτη αρχιτεκτονική αφορά ένα σύστημα το οποίο αποτελείται από τον επεξεργαστή και την κύρια μνήμη.
2. Η δεύτερη έχει επαυξηθεί με L1 Instruction, Data και L2 caches

1.3.1

Πρώτη αρχιτεκτονική (χωρίς cache): Το σύστημα περιλαμβάνει μόνο τον επεξεργαστή και την κύρια μνήμη



Τρέχουμε την παρακάτω εντολή

```
build/X86/gem5.opt configs/learning_gem5/part1/simple.py  
/gem5/tables_UF/tables.exe
```

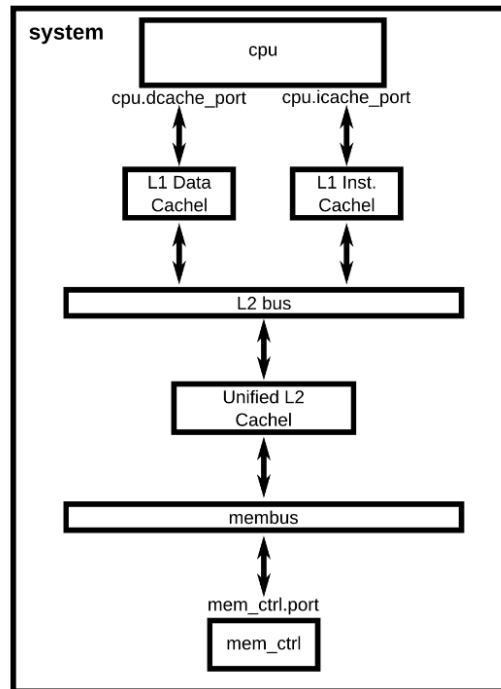
cd gem5/m5out/

```
grep "system.cpu.numCycles" stats.txt
```

```
root@28bcefe998c0:/gem5/m5out# grep "system.cpu.numCycles" stats.txt  
system.cpu.numCycles 857689669
```

1.3.2

Δεύτερη αρχιτεκτονική (με L1 και L2 cache):



Τρέχουμε την παρακάτω εντολή

```
build/X86/gem5.opt configs/learning_gem5/part1/two_level.py  
/gem5/tables_UF/tables.exe --l1i_size=8kB --l1d_size=8kB --l2_size=128kB
```

cd gem5/m5out/

```
grep "system.cpu.numCycles" stats.txt
```

```
root@28bcefe998c0:/gem5/m5out# grep "system.cpu.numCycles" stats.txt  
system.cpu.numCycles 59787087 # Number of cpu cycles simulated (Cycle)  
root@28bcefe998c0:/gem5/m5out#
```

simple.py → 857689669 CC

two_level.py (L1=8kB, L2=128kB) → 59787087 CC

Η εκτέλεση της εφαρμογής για την πρώτη αρχιτεκτονική, που αφορά ένα σύστημα το οποίο αποτελείται από τον επεξεργαστή και την κύρια μνήμη, απαιτεί 857,689,669 CC. Ενώ για τη δεύτερη, που αφορά ένα σύστημα το οποίο έχει επανυζηθεί με L1 Instruction, Data και L2 caches, απαιτεί 59,787,087 CC.

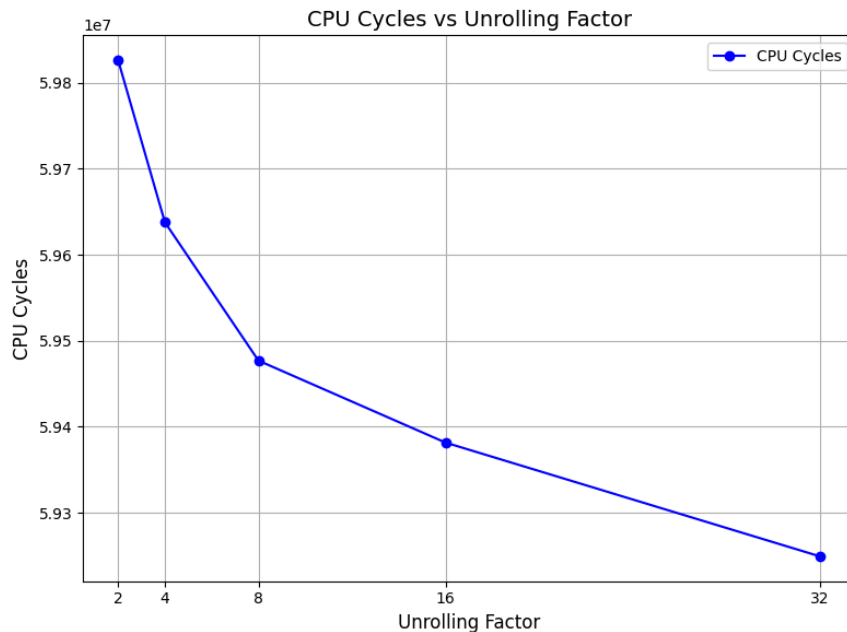
Παρατηρούμε ότι στη δεύτερη αρχιτεκτονική έχουμε μείωση των κύκλων.

Με την προσθήκη 2 επιπέδων cache (L1, L2) αποφεύγουμε τις πολύ συχνές προσβάσεις στην κύρια μνήμη. Οι L1 και L2 είναι αρκετά μεγάλες (8kB, 128kB αντίστοιχα) οπότε ένα μεγάλο μέρος των δεδομένων και των εντολών μένουν αποθηκευμένα πιο κοντά στην CPU. Οι προσβάσεις στην κύρια μνήμη είναι πολύ πιο αργές από τις προσβάσεις στην cache διότι συνεπάγονται τη χρήση περισσότερων stalls.

1.3.3 Α Μέρος

Αποτελέσματα που προέκυψαν από την προσομοίωση για διαφορετικά unrolling factors(2,4,8,16,32).

```
#tables_uf2.exe .....system.cpu.numCycles  59826244  CC  
#tables_uf4.exe .....system.cpu.numCycles  59637744  CC  
#tables_uf8.exe .....system.cpu.numCycles  59476823  CC  
#tables_uf16.exe.....system.cpu.numCycles  59381481  CC  
#tables_uf32.exe.....system.cpu.numCycles  59249179  CC
```



Από το παραπάνω plot παρατηρούμε ότι όταν αυξάνουμε το loop unrolling factor(2,4,8,16,32) πετυχαίνουμε λιγότερους κύκλους CPU και συνεπώς καλύτερα run time. Αυτό το συμπέρασμα είναι λογικό, αφού με το loop unrolling κάθε iteration του loop έχει παραπάνω εντολές, με αποτέλεσμα να εκτελούμε λιγότερα iterations.

Για unrolling factor 2 μέχρι unrolling factor 8 παρατηρείται η πιο ραγδαία αλλαγή.

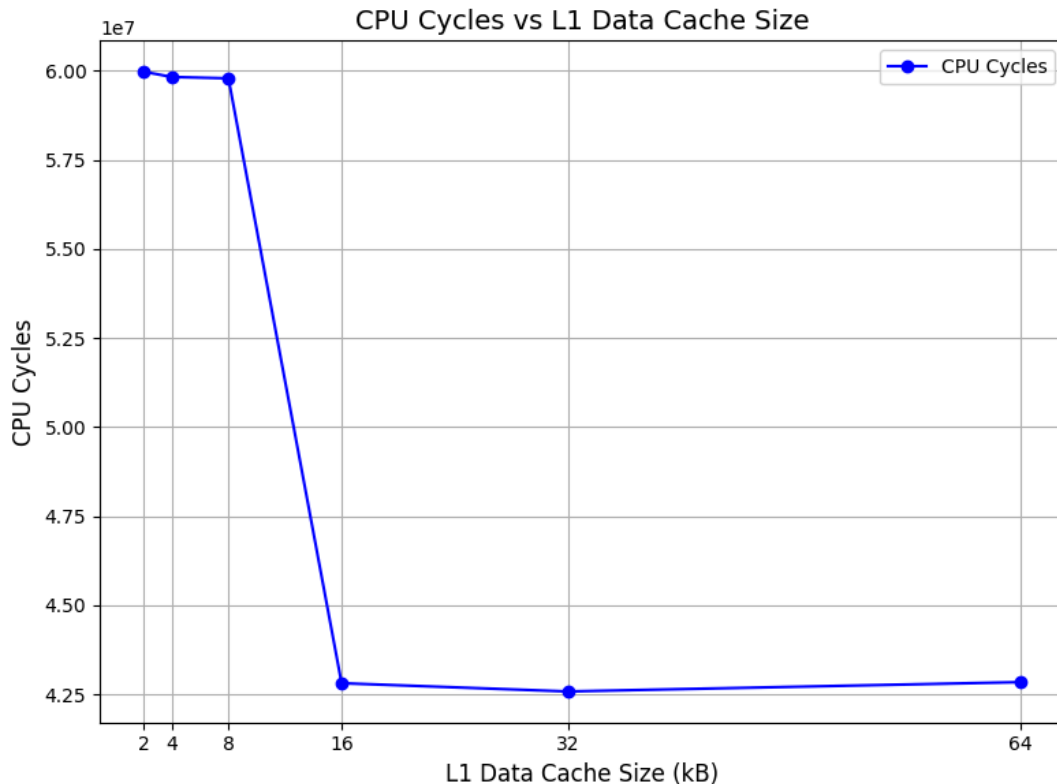
Για unrolling factor 16 και 32 παρατηρούμε μικρότερο κέρδος από το περαιτέρω loop unrolling.

Αυτο συμβαίνει διότι μετά από ένα σημείο το κόστος του loop overhead γίνεται συγκρίσιμο με το κόστος των operations που γίνονται μέσα στο loop, οπότε μπορεί να εκτελούμε ακόμα λιγότερα iterations του for loop σε σχέση με πριν, αλλά με πιά απαιτητικό computation το καθένα.

1.3.3 Β Μέρος

Αποτελέσματα που προέκυψαν από την προσομοίωση για διαφορετικά μεγέθη L1 Data cache.

2kB → 59973394 CC
4kB → 59824719 CC
8kB → 59787087 CC
16kB → 42811763 CC
32kB → 42580473 CC
64kB → 42842142 CC



Από το παραπάνω plot παρατηρούμε ότι όσο αυξάνεται το μέγεθος της L1 Data Cache, οι κύκλοι μειώνονται, ειδικά με τη μετάβαση από 8KB σε 16KB. Ωστόσο, μεταξύ 16KB και 64KB παρατηρείται μια σταθεροποίηση στους κύκλους. Το καλύτερο αποτέλεσμα σε σχέση με τον αριθμό των κύκλων καταγράφεται όταν το μέγεθος της cache είναι 32KB, καθώς στα 64KB υπάρχει μια μικρή αύξηση στους κύκλους, άρα παρόλο που η cache είναι μεγαλύτερη, δεν οδηγεί σε περαιτέρω βελτίωση της απόδοσης. Αυτές οι καθυστερήσεις οφείλονται σε παράγοντες όπως στη ανάγκη για μεγαλύτερη διαχείριση της μνήμης και η αυξημένη πολυπλοκότητα στην αναζήτηση δεδομένων μέσα στην Cache.

1.3.4

Ο σκοπός είναι να ελαχιστοποιήσουμε το πλήθος των κύκλων που απαιτούνται για την εκτέλεση της εφαρμογής μας αλλά και τη συνολική μνήμη που θα χρησιμοποιήσουμε δηλ. το άθροισμα της L1D, της L1I και της L2 cache σε KB.

Για τις ανάγκες του ερωτήματος δημιουργήσαμε ένα νέο script με όνομα `run_simulations.py` το οποίο

εφαρμόζει εξαντλητική αναζήτηση για κάθε πιθανό συνδυασμό των παρακάτω χαρακτηριστικών, καταγράφοντας παράλληλα το πλήθος των κύκλων που απαιτούνται για την εκτέλεση της εφαρμογής σε ένα αρχείο CSV.

L1D cache size \in [2kB,4kB,8kB,16kB,32kB,64kB]

L1I cache size \in [2kB,4kB,8kB,16kB,32kB,64kB]

L2 cache size \in [128kB,256kB,512kB,1024kB]

Unrolling factor \in [2,4,8,16,32]

κώδικας script *run_simulations.py* :

```
import subprocess
import csv
import os

# Define cache sizes and unrolling factors
L1I_SIZES = ["2kB", "4kB", "8kB", "16kB", "32kB", "64kB"]
L1D_SIZES = ["2kB", "4kB", "8kB", "16kB", "32kB", "64kB"]
L2_SIZES = ["128kB", "256kB", "512kB", "1024kB"]
UNROLL_FACTORS = [2, 4, 8, 16, 32]

# Output CSV file in the /gem5 directory
OUTPUT_FILE = "/gem5/simulation_results.csv"

# Write the header of the CSV
with open(OUTPUT_FILE, mode="w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(["L1I_Size", "L1D_Size", "L2_Size", "Unrolling_Factor", "CPU_Cycles"])

# Function to run the simulation and extract CPU cycles from stats.txt
def run_simulation(L1I_SIZE, L1D_SIZE, L2_SIZE, UNROLL_FACTOR):
    command = [
        "build/X86/gem5.opt",
        "configs/learning_gem5/part1/two_level.py",
        f"/gem5/tables_UF/tables_uf{UNROLL_FACTOR}.exe",
        f"--l1i_size={L1I_SIZE}",
        f"--l1d_size={L1D_SIZE}",
        f"--l2_size={L2_SIZE}"
    ]

    try:
        subprocess.run(command, capture_output=True, text=True, check=True)
        stats_file = "/gem5/m5out/stats.txt"

        # Check if the stats.txt file exists after running the simulation
        if os.path.isfile(stats_file):
            with open(stats_file, "r") as file:
                for line in file:
                    if "system.cpu.numCycles" in line:
                        # Extract CPU cycles from the line
                        cpu_cycles = int(line.split()[1])
                        return cpu_cycles
        else:
            print("Error: stats.txt not found in m5out directory.")
            return None

    except subprocess.CalledProcessError as e:
        print(f"Error running simulation for {L1I_SIZE}, {L1D_SIZE}, {L2_SIZE}, {UNROLL_FACTOR}: {e}")
        return None

# Loop through all combinations of cache sizes and unrolling factors
for L1I_SIZE in L1I_SIZES:
    for L1D_SIZE in L1D_SIZES:
```

```

        for L2_SIZE in L2_SIZES:
            for UNROLL_FACTOR in UNROLL_FACTORS:
                print(f"Running simulation for L1I: {L1I_SIZE}, L1D: {L1D_SIZE}, L2: {L2_SIZE}, Unroll: {UNROLL_FACTOR}")

                cpu_cycles = run_simulation(L1I_SIZE, L1D_SIZE, L2_SIZE, UNROLL_FACTOR)

            if cpu_cycles is not None:
                with open(OUTPUT_FILE, mode="a", newline="") as file:
                    writer = csv.writer(file)
                    writer.writerow([L1I_SIZE, L1D_SIZE, L2_SIZE, UNROLL_FACTOR, cpu_cycles])

        print("Simulation complete. Results saved to", OUTPUT_FILE)

```

Ο συνολικός χρόνος που απαιτήθηκε για την εκτέλεση του script ήταν : 03:15:00

Αφού ολοκληρώθηκε το exhaustive search πήραμε σαν output το αρχείο `simulation_results.csv` του οποίου η μορφή ήταν κάπως έτσι:

```

root@fd67424b0616: /gem5
L1I_Size,L1D_Size,L2_Size,Unrolling_Factor,CPU_Cycles
2kB,2kB,128kB,2,60081042
2kB,2kB,128kB,4,59891554
2kB,2kB,128kB,8,59725456
2kB,2kB,128kB,16,64577590
2kB,2kB,128kB,32,65964857
2kB,2kB,256kB,2,60070840
2kB,2kB,256kB,4,59876289
2kB,2kB,256kB,8,59728803
2kB,2kB,256kB,16,64566189
2kB,2kB,256kB,32,65945624
2kB,2kB,512kB,2,60044481

```

Με τη βοήθεια της βιβλιοθήκης `paretoset` της `python` εντοπίσαμε τρέχοντας το ακόλουθο script τις Pareto Optimal λύσεις.

```

import pandas as pd
from paretoset import paretoset

data = pd.read_csv('simulation_results.csv')

# Αφαιρέσι του kB από τις stiles
def convert_memory_size(size):
    return int(size.replace('kB', '').strip())

data['L1I_Size'] = data['L1I_Size'].apply(convert_memory_size)
data['L1D_Size'] = data['L1D_Size'].apply(convert_memory_size)
data['L2_Size'] = data['L2_Size'].apply(convert_memory_size)

# Compute the total memory
data['totalMemoryKB'] = data['L1I_Size'] + data['L1D_Size'] + data['L2_Size']

# Define the objectives: minimize both CPU_Cycles and totalMemoryKB
objectives = ['CPU_Cycles', 'totalMemoryKB']

# Find the Pareto frontier
pareto_mask = paretoset(data[objectives], sense=["min", "min"])

pareto_front = data[pareto_mask].sort_values(by=['CPU_Cycles', 'totalMemoryKB'], ascending=[True, True])

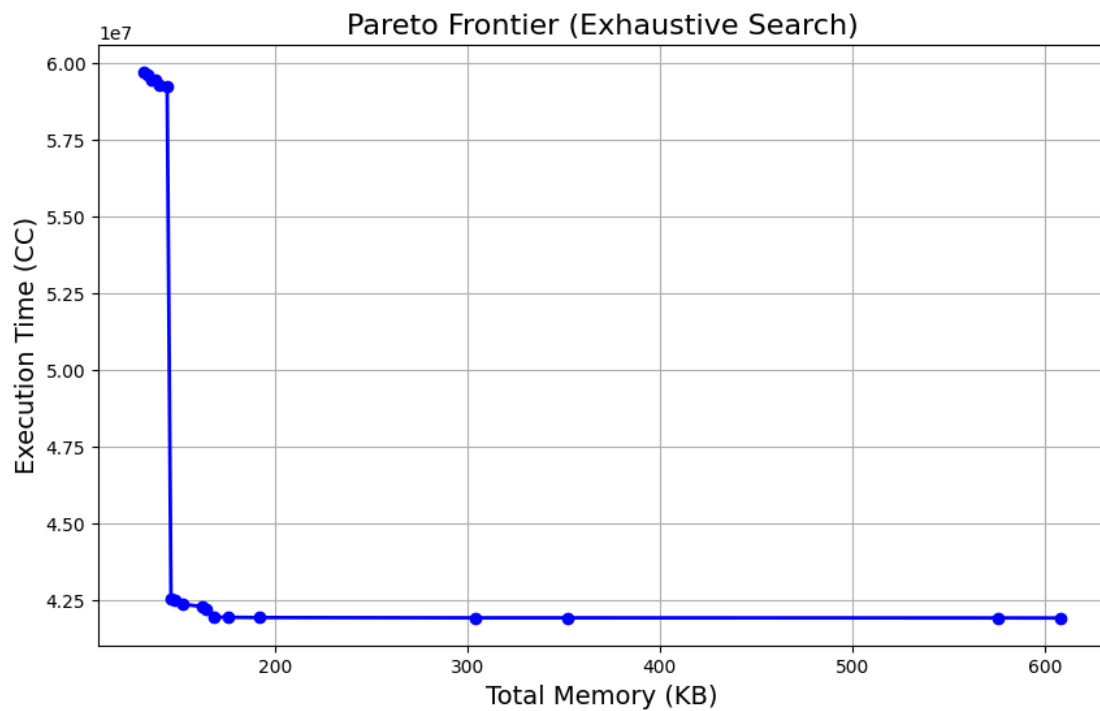
print("Pareto-optimal combinations:")
print(pareto_front)
pareto_front.to_csv('pareto_frontier.csv', index=False)

```

output από την εκτέλεση του προηγούμενου script.

Pareto-optimal combinations:						
	L1I_Size	L1D_Size	L2_Size	Unrolling_Factor	CPU_Cycles	totalMemoryKB
2	2	2	128	8	59725456	132
62	2	16	128	8	42539600	146
82	2	32	128	8	42278590	162
123	4	2	128	16	59608257	134
143	4	4	128	16	59477547	136
183	4	16	128	16	42480342	148
203	4	32	128	16	42185703	164
244	8	2	128	32	59447087	138
264	8	4	128	32	59287222	140
284	8	8	128	32	59249179	144
304	8	16	128	32	42348777	152
324	8	32	128	32	41936302	168
444	16	32	128	32	41924727	176
449	16	32	256	32	41908054	304
564	32	32	128	32	41915945	192
574	32	32	512	32	41904833	576
589	32	64	256	32	41907792	352
694	64	32	512	32	41901786	608

Plot με το Pareto Frontier:



1.3.5

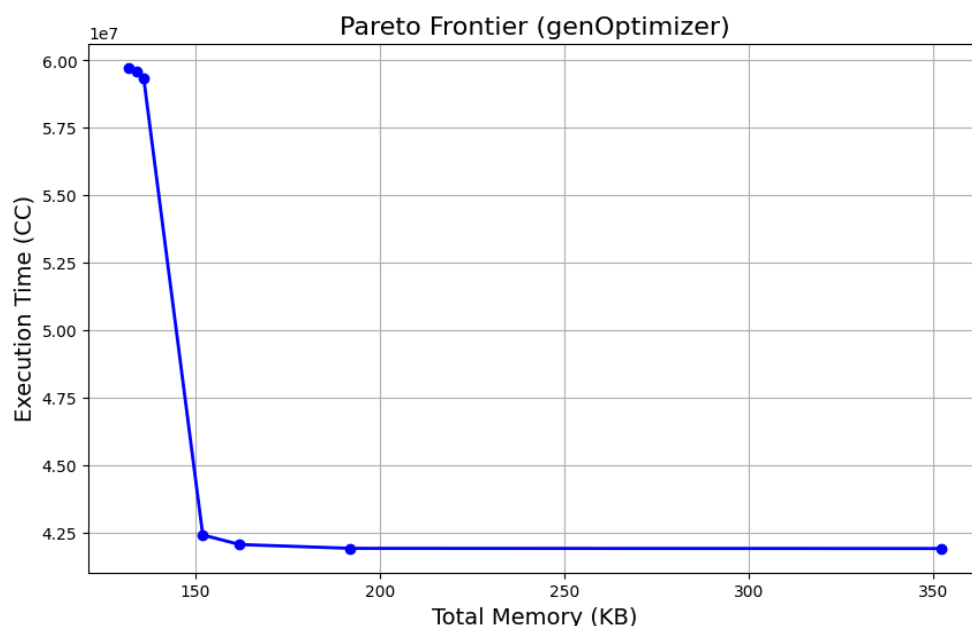
Pareto Optimal Configurations

- 1) L1I (KB) = 4 L1D (KB) = 4 L2 (KB) = 128 UF = 16 config. results in Execution Time (CC) = 59343511 and Total Memory (KB) = 136
- 2) L1I (KB) = 2 L1D (KB) = 4 L2 (KB) = 128 UF = 4 config. results in Execution Time (CC) = 59583529 and Total Memory (KB) = 134
- 3) L1I (KB) = 32 L1D (KB) = 64 L2 (KB) = 256 UF = 16 config. results in Execution Time (CC) = 41929862 and Total Memory (KB) = 352
- 4) L1I (KB) = 2 L1D (KB) = 32 L2 (KB) = 128 UF = 8 config. results in Execution Time (CC) = 42078234 and Total Memory (KB) = 162
- 5) L1I (KB) = 2 L1D (KB) = 2 L2 (KB) = 128 UF = 4 config. results in Execution Time (CC) = 59731958 and Total Memory (KB) = 132
- 6) L1I (KB) = 32 L1D (KB) = 32 L2 (KB) = 128 UF = 16 config. results in Execution Time (CC) = 41936488 and Total Memory (KB) = 192
- 7) L1I (KB) = 8 L1D (KB) = 16 L2 (KB) = 128 UF = 8 config. results in Execution Time (CC) = 42431214 and Total Memory (KB) = 152

Configuration	L1I (KB)	L1D (KB)	L2 (KB)	UF	Execution Time (CC)	Total Memory (KB)
1	4	4	128	16	59343511	136
2	2	4	128	4	59583529	134
3	32	64	256	16	41929862	352
4	2	32	128	8	42078234	162
5	2	2	128	4	59731958	132
6	32	32	128	16	41936488	192
7	8	16	128	8	42431214	152

Στο κώδικα που μας δόθηκε στο genOptimizer.py υλοποιείται ο γενετικός αλγόριθμος με τη διαδικασία του NSGA-II. Ο NSGA-II (Non-dominated Sorting Genetic Algorithm II) είναι ένας γενετικός αλγόριθμος πολλαπλών στόχων (multi-objective) που χρησιμοποιείται για την εύρεση των βέλτιστων λύσεων σε προβλήματα με περισσότερους από έναν αντικειμενικούς στόχους, στο συγκεκριμένο μας αφορά το πλήθος των κύκλων που απαιτούνται για την εκτέλεση της εφαρμογής μας και συνολική μνήμη. Στον υπάρχοντα κώδικα προστέθηκε μια εντολή για την εκτύπωση του συνολικού αριθμού των evaluations. Τα συνολικά evaluations που μετρήθηκαν είναι 30. Τα αποτελέσματα τα πήραμε σε πολύ μικρό χρονικό διάστημα (25 λεπτά) σε σύγκριση με την εξαντλητική εξερεύνηση, η οποία διήρκεσε πάνω από 3 ώρες. Με τον γενετικό αλγόριθμο βρέθηκαν 7 σημεία στο Pareto frontier.

Total evaluations: 30



Αφού έχουμε βρεί το pareto frontier και με εξαντλητική αναζήτηση αλλά και με χρήση γενετικού αλγορίθμου, παρατηρούμε ότι τα 2 plot παρουσιάζουν αρκετά παρεμφερή αποτελέσματα.

Με την χρήση εξαντλητικής αναζήτησης έχουμε πιο ακριβές plot (συγκεκριμένα 18 σημεία Pareto) αλλά ξοδεύουμε σχεδόν 10πλάσιο χρόνο στην εκτέλεση όλων των πιθανών αρχιτεκτονικών.

Με την χρήση γενετικού αλγορίθμου λιγότερο ακριβές plot (7 σημεία Pareto) σε πολύ λιγότερο χρόνο (25λεπτά).

Συμπερασματικά η εξαντλητική αναζήτηση μπορεί να είναι μια καλή λύση όταν έχουμε μικρότερο αριθμό παραμέτρων να ελέγχουμε. Όταν έχουμε να δοκιμάσουμε πάρα πολλές διαφορετικές αρχιτεκτονικές η εξαντλητική αναζήτηση δεν είναι καθόλου αποδοτική, αν και ακριβής. Σε αυτή την περίπτωση η χρήση ενός γενετικού αλγορίθμου μπορεί να προσφέρει πιο γρήγορη λύση, με κάποιο trade-off στην ακρίβεια των αποτελεσμάτων.