1η Σειρά Ασκήσεων στο μάθημα

Σχεδιασμός Ενσωματωμένων Κυκλωμάτων

Ομάδα 29

Μέλη : Αριστοτέλης Γρίβας

Σάββας Λεβεντικίδης

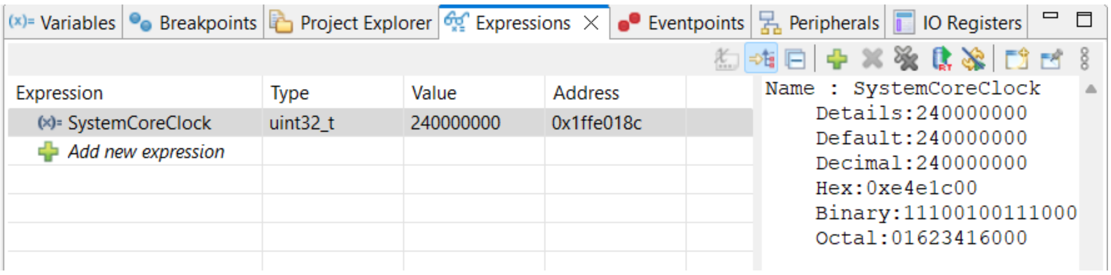
Άσκηση 1:

1)

Από το datasheet εξάγουμε ότι το Renesas S7G2 έχει SRAM μεγέθους 640 KB και δύο Flash μνήμες εκ των οποίων η μία είναι Code Flash Memory μεγέθους 4 MB και η άλλη είναι Data Flash Memory μεγέθους 64 KB

Το συγκεκριμένο board έχει flash cache memory η οποία συνεισφέρει στην βελτίωση των επιδόσεων μειώνοντας τις καθυστερήσεις λόγω προσβάσεων στην κύρια μνήμη.

Τη συχνότητα ρολογιού, την υπολογίζουμε βάζοντας ένα watch expression στο SystemCoreClock, όπως φαίνεται παρακάτω (240MHz) :



2)

Για τη μέτρηση του απαιτούμενου χρόνου, τροποποιούμε τη συνάρτηση hal_entry όπως φαίνεται παρακάτω. Συνοπτικά, αρχικοποιούμε τον register που μετρά τους κύκλους, εκτελούμε τον κώδικα, μετράμε τους κύκλους και τους αποθηκεύουμε στον πίνακα timer[], και στη συνέχεια βρίσκουμε το χρόνο και τον αποθηκεύουμε στον πίνακα cycles[], διαιρώντας τον πίνακα timer[] με τη συχνότητα. Συνεπώς, ο πίνακας cycles[], έχει τον απαιτούμενο χρόνο κάθε επανάληψη σε ms :

```
	float timer[10] = {0};
	float cycles[10] = {0};
void hal_entry(void) {
```

```c
    // Code to initialize the DWT->CYCCNT register
    CoreDebug->DEMCR |= 0x01000000;
       ITM->LAR = 0xC5ACCE55;
       DWT->CYCCNT = 0;
       DWT->CTRL |= 1;
       /* Initialize your variables here */
       int motion_vectors_x[N/B][M/B],
           motion_vectors_y[N/B][M/B], i, j;

      for (int i=0; i<10; i++){
        /* Add timer code here */
        phods_motion_estimation(current,previous,motion_vectors_x,motion_vectors_y);
        timer[i] = DWT->CYCCNT;
      }
      for (int i=1; i<10; i++){
         cycles[i] = (timer[i]-timer[i-1])/240000;
      }
      cycles[0] = timer[0]/240000;
      while(1){

      }
}
```

Πήραμε τα εξής αποτελέσματα :

Expression	Type	Value	Address
cycles[1]	float	1.843266...	0x1ffe01cc
cycles[0]	float	1.843279...	0x1ffe01c8
cycles[2]	float	1.843266...	0x1ffe01d0
cycles[1]	float	1.843266...	0x1ffe01cc
cycles[3]	float	1.843266...	0x1ffe01d4
cycles[4]	float	1.843266...	0x1ffe01d8
cycles[5]	float	1.843266...	0x1ffe01dc
cycles[6]	float	1.843266...	0x1ffe01e0
cycles[7]	float	1.843266...	0x1ffe01e4
cycles[8]	float	1.843266...	0x1ffe01e8
cycles[9]	float	1.843266...	0x1ffe01ec
Add new expressi			

Παρατηρούμε ότι δεν υπάρχει διαφοροποίηση στον χρόνο εκτέλεσης (σε msec) εκτός από την πρώτη φορά που τον τρέχουμε η οποία έχει μια ελάχιστη διαφοροποίηση. Πιθανώς να έχει να κάνει με κάποια initializations του πυρήνα.

3)

Στο σημείο αυτό, θα εφαρμόσουμε μετασχηματισμούς στον κώδικα ώστε να μειωθεί ο χρόνος εκτέλεσης. Αρχικά, θα ασχοληθούμε με τους μετασχηματισμούς global loop και συγκεκριμένα στις μεταβλητές i,k,l .

Ειδικότερα, παρατηρούμε πως μπορούμε να συγχωνεύσουμε τις εντολές μέσα στα loops :

for(i=-S; i<S+1; i+=S), for(k=0; k<B; k++) , for(l=0; l<B; l++)

και να τις ενσωματώσουμε σε μία επανάληψη των βρόχων αυτών. Επίσης παρατηρούμε τα εξής :

· Η εντολή ανάγνωσης p1=current[B*x+k][B*y+l]; εμφανίζεται δύο φορές μέσα στις εντολές του νέου βρόχου , οπότε διαγράφουμε την επανάληψη της εντολής.

· Για i=0 το p2 είναι ίσο με το q2 οπότε δεν είναι απαραίτητο να το υπολογίσουμε ξανά, οπότε εισάγουμε μια εντολή ελέγχου if(i==0) disty=distx; που αυτόματα παρακάμπτει τον υπολογισμό του disty όταν το i==0.

· Ο διπλός βρόχος αρχικοποίησης των μεταβλητών vectors_x και vectors_y, μπορεί να παραληφθεί και η αρχικοποίηση των πινάκων να γίνει μέσα στο loops των x,y.

Ο κώδικας μετα τους μετασχηματισμούς παρουσιάζεται παρακάτω :

```
#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.*/

      float timer[10] = {0};
      float cycles[10] = {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;
  /*For all blocks in the current frame*/
  for(x=0; x<N/B; x++)
  {
    for(y=0; y<M/B; y++)
    {
      vectors_x[x][y] = 0;
      vectors_y[x][y] = 0;
      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;
          disty = 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];

              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);
```

```
              if(i == 0) disty = distx;

              else{ // else opens


              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);
               } // else closes

            }
          }

          if(distx < min1)
          {
            min1 = distx;
            bestx = i;
          }

          if(disty < min2)
          {
            min2 = disty;
            besty = i;
          }
        }

        /*For all candidate blocks in Y dimension*/

        S = S/2;
        vectors_x[x][y] += bestx;
        vectors_y[x][y] += besty;
      }
    }
  }

}
```

```c
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 */
       int motion_vectors_x[N/B][M/B],
           motion_vectors_y[N/B][M/B], i, j;

      for (int i=0; i<10; i++){
         /* Add timer code here */
         phods_motion_estimation(current,previous,motion_vectors_x,moti
on_vectors_y);
         timer[i] = DWT->CYCCNT;
      }
      for (int i=1; i<10; i++){
          cycles[i] = (timer[i]-timer[i-1])/240000;
      }
      cycles[0] = timer[0]/240000;
      while(1){

      }
}
```

Τα αποτελέσματα που λαμβάνουμε είναι τα εξής :

Expression	Type	Value	Address
cycles[1]	float	1.369154...	0x1ffe01cc
cycles[0]	float	1.369166...	0x1ffe01c8
cycles[2]	float	1.369154...	0x1ffe01d0
cycles[1]	float	1.369154...	0x1ffe01cc
cycles[3]	float	1.369154...	0x1ffe01d4
cycles[4]	float	1.369154...	0x1ffe01d8
cycles[5]	float	1.369154...	0x1ffe01dc
cycles[6]	float	1.369154...	0x1ffe01e0
cycles[7]	float	1.369154...	0x1ffe01e4
cycles[8]	float	1.369154...	0x1ffe01e8
cycles[9]	float	1.369154...	0x1ffe01ec
Add new expressi			

Όπως αναμέναμε, οι χρόνοι έχουν μειωθεί σημαντικά.

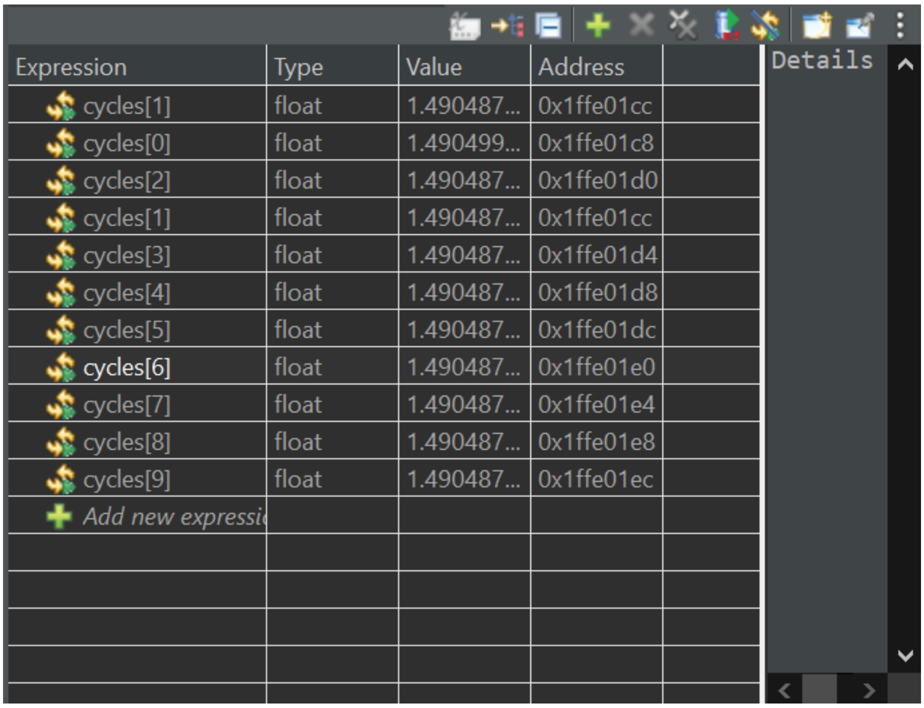
<u>Δοκιμή κάποιων έξτρα μετασχηματισμών (επαναχρησιμοποίησης δεδομένων), οι οποίοι, εν τέλη, δε βοήθησαν στη μείωση του χρόνου :</u>

Το επόμενο στάδιο της μεθοδολογίας είναι η εφαρμογή των μετασχηματισμών επαναχρησιμοποίησης δεδομένων. Αυτό που θέλουμε να πετύχουμε είναι να μειωθούν οι προσπελάσεις στους πίνακες δεδομένων current και previous οι οποίοι έχουν μεγάλο μέγεθος. Η μείωση θα γίνει με την εισαγωγή μικρότερου μεγέθους πινάκων δεδομένων για την προσωρινή αποθήκευση τμημάτων των αρχικών πινάκων.

Ακολουθώντας τη μεθοδολογία που υπάρχει και στο εργαστηριακό υλικό, θα δοκιμάσουμε τους εξής μετασχηματισμούς :

· Το μετασχηματισμό επαναχρησιμοποίησης δεδομένων με την εισαγωγή του πίνακα current_line.

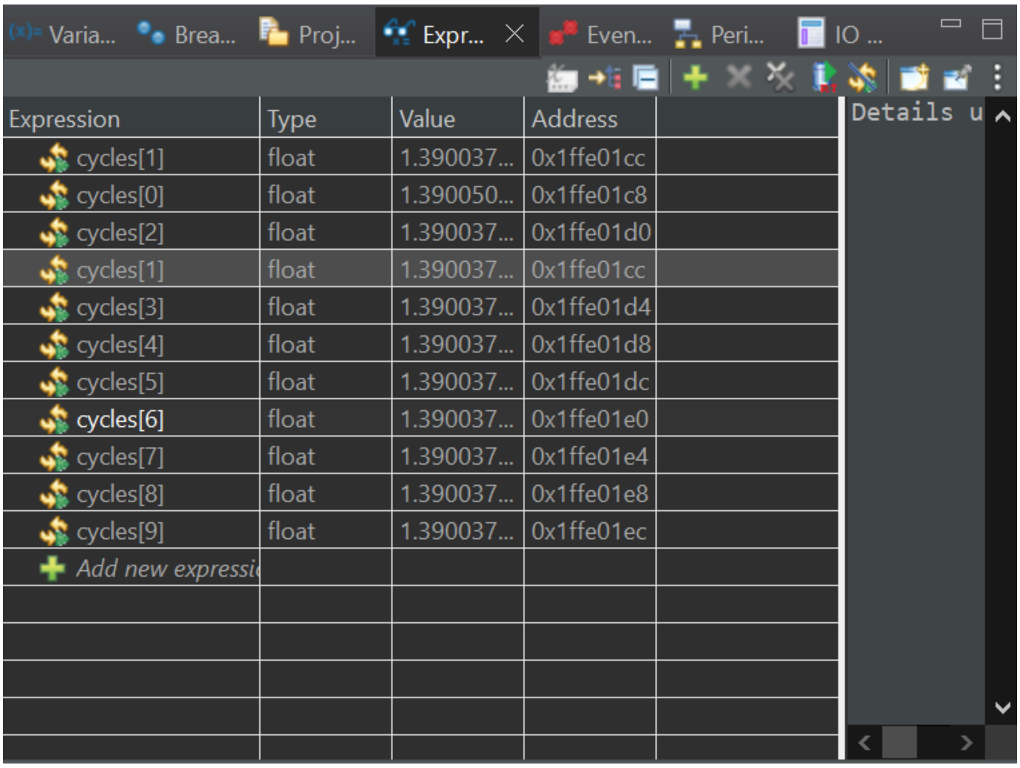
Έχουμε τα εξής αποτελέσματα :



Expression	Type	Value	Address
cycles[1]	float	1.490487...	0x1ffe01cc
cycles[0]	float	1.490499...	0x1ffe01c8
cycles[2]	float	1.490487...	0x1ffe01d0
cycles[1]	float	1.490487...	0x1ffe01cc
cycles[3]	float	1.490487...	0x1ffe01d4
cycles[4]	float	1.490487...	0x1ffe01d8
cycles[5]	float	1.490487...	0x1ffe01dc
cycles[6]	float	1.490487...	0x1ffe01e0
cycles[7]	float	1.490487...	0x1ffe01e4
cycles[8]	float	1.490487...	0x1ffe01e8
cycles[9]	float	1.490487...	0x1ffe01ec
Add new expressi			

· Το μετασχηματισμό επαναχρησιμοποίησης δεδομένων με την εισαγωγή του πίνακα block.

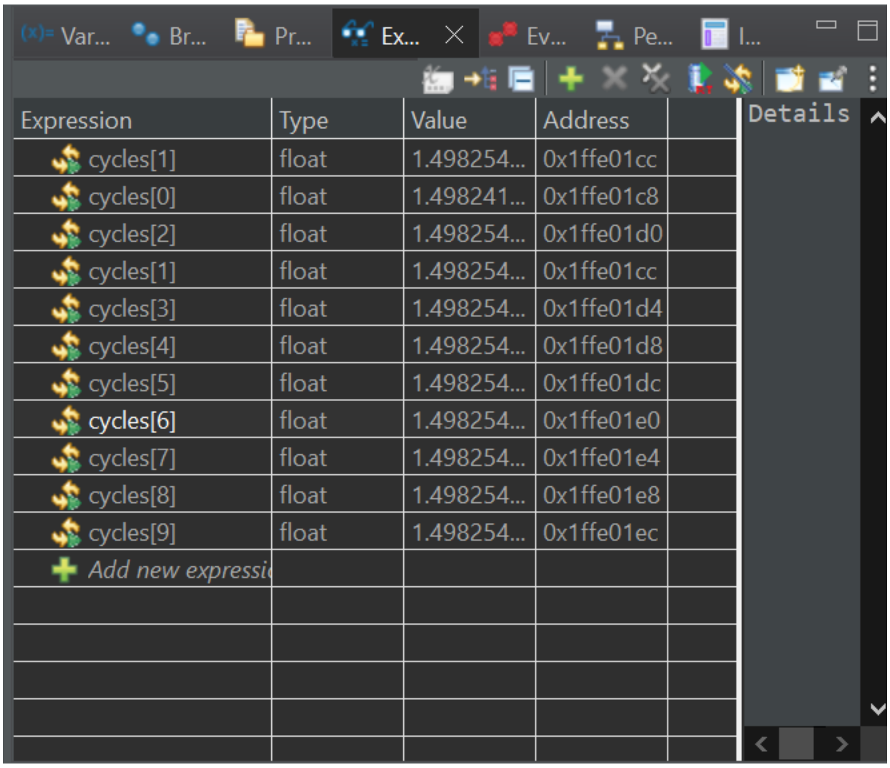
Έχουμε τα εξής αποτελέσματα :



Expression	Type	Value	Address
cycles[1]	float	1.390037...	0x1ffe01cc
cycles[0]	float	1.390050...	0x1ffe01c8
cycles[2]	float	1.390037...	0x1ffe01d0
cycles[1]	float	1.390037...	0x1ffe01cc
cycles[3]	float	1.390037...	0x1ffe01d4
cycles[4]	float	1.390037...	0x1ffe01d8
cycles[5]	float	1.390037...	0x1ffe01dc
cycles[6]	float	1.390037...	0x1ffe01e0
cycles[7]	float	1.390037...	0x1ffe01e4
cycles[8]	float	1.390037...	0x1ffe01e8
cycles[9]	float	1.390037...	0x1ffe01ec
Add new expressi			

· Το μετασχηματισμό επαναχρησιμοποίησης δεδομένων με την εισαγωγή των πινάκων current_line και block.

Έχουμε τα εξής αποτελέσματα :



Expression	Type	Value	Address
cycles[1]	float	1.498254...	0x1ffe01cc
cycles[0]	float	1.498241...	0x1ffe01c8
cycles[2]	float	1.498254...	0x1ffe01d0
cycles[1]	float	1.498254...	0x1ffe01cc
cycles[3]	float	1.498254...	0x1ffe01d4
cycles[4]	float	1.498254...	0x1ffe01d8
cycles[5]	float	1.498254...	0x1ffe01dc
cycles[6]	float	1.498254...	0x1ffe01e0
cycles[7]	float	1.498254...	0x1ffe01e4
cycles[8]	float	1.498254...	0x1ffe01e8
cycles[9]	float	1.498254...	0x1ffe01ec
Add new expressi			

Παρατηρούμε πως, γενικώς, έχουμε αύξηση του απαιτούμενου χρόνου για την εκτέλεση της εφαρμογής. Αυτό οφείλεται,κυρίως, στα έξτρα loops που έχουμε προσθέσει για την αρχικοποίηση των νέων πινάκων.

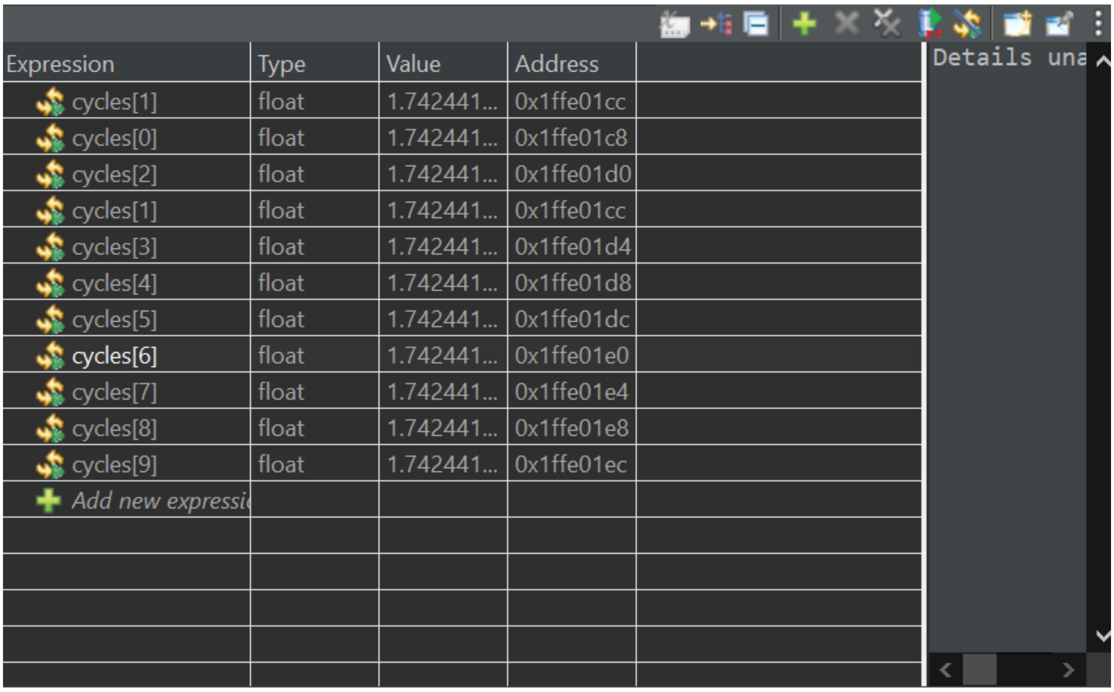
Συνεπώς, ο τελικός και βέλτιστος κώδικας, είναι αυτός που έχουμε παραθέσει παραπάνω (μετά τους μεγασχηματισμούς global loop) .

4)

Θα λάβουμε αποτελέσματα για τους κοινούς διαιρέτες των M και N επομένως για B με τιμές 1,2,5,10 .

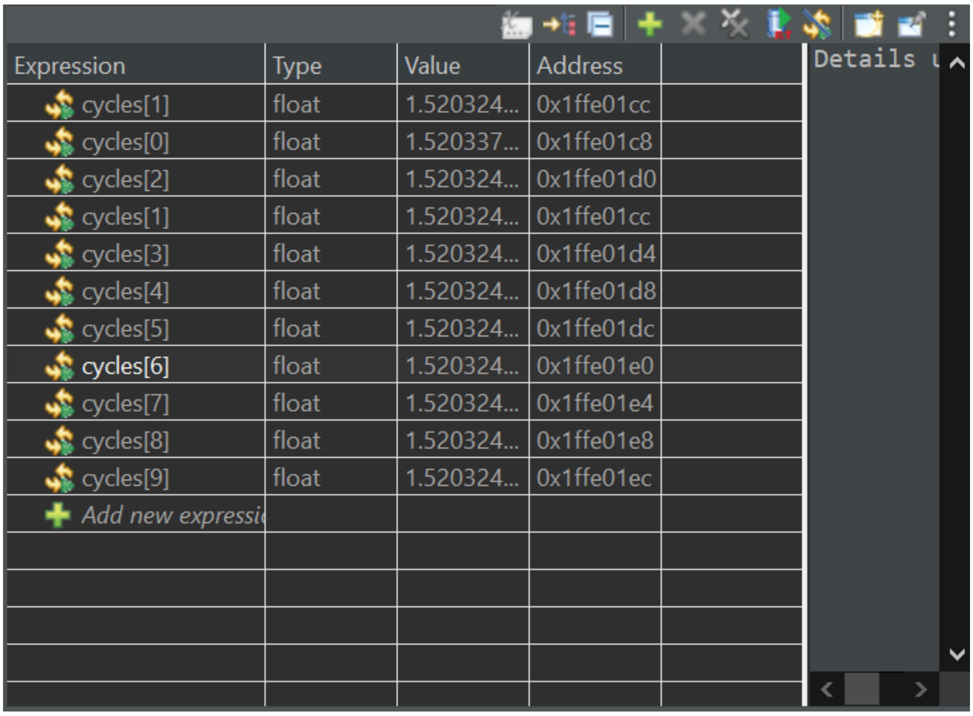
Για B = 5 τα αποτελέσματα είναι τα ίδια με το ερώτημα 3 εφόσον το τρέξαμε με B = 5

Για B = 1 λαμβάνουμε τα εξής αποτελέσματα :



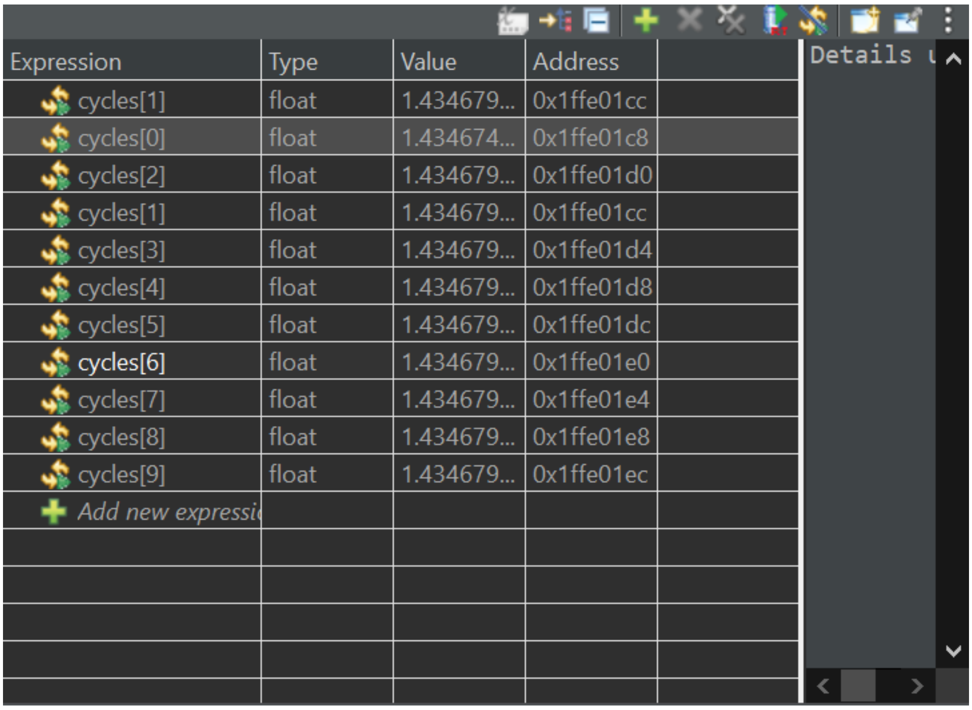
Expression	Type	Value	Address
cycles[1]	float	1.742441...	0x1ffe01cc
cycles[0]	float	1.742441...	0x1ffe01c8
cycles[2]	float	1.742441...	0x1ffe01d0
cycles[1]	float	1.742441...	0x1ffe01cc
cycles[3]	float	1.742441...	0x1ffe01d4
cycles[4]	float	1.742441...	0x1ffe01d8
cycles[5]	float	1.742441...	0x1ffe01dc
cycles[6]	float	1.742441...	0x1ffe01e0
cycles[7]	float	1.742441...	0x1ffe01e4
cycles[8]	float	1.742441...	0x1ffe01e8
cycles[9]	float	1.742441...	0x1ffe01ec
Add new expressi			

Για B = 2 λαμβάνουμε τα εξής αποτελέσματα :



Expression	Type	Value	Address
cycles[1]	float	1.520324...	0x1ffe01cc
cycles[0]	float	1.520337...	0x1ffe01c8
cycles[2]	float	1.520324...	0x1ffe01d0
cycles[1]	float	1.520324...	0x1ffe01cc
cycles[3]	float	1.520324...	0x1ffe01d4
cycles[4]	float	1.520324...	0x1ffe01d8
cycles[5]	float	1.520324...	0x1ffe01dc
cycles[6]	float	1.520324...	0x1ffe01e0
cycles[7]	float	1.520324...	0x1ffe01e4
cycles[8]	float	1.520324...	0x1ffe01e8
cycles[9]	float	1.520324...	0x1ffe01ec
Add new expressi			

Για B = 10 λαμβάνουμε τα εξής αποτελέσματα :



Expression	Type	Value	Address
cycles[1]	float	1.434679...	0x1ffe01cc
cycles[0]	float	1.434674...	0x1ffe01c8
cycles[2]	float	1.434679...	0x1ffe01d0
cycles[1]	float	1.434679...	0x1ffe01cc
cycles[3]	float	1.434679...	0x1ffe01d4
cycles[4]	float	1.434679...	0x1ffe01d8
cycles[5]	float	1.434679...	0x1ffe01dc
cycles[6]	float	1.434679...	0x1ffe01e0
cycles[7]	float	1.434679...	0x1ffe01e4
cycles[8]	float	1.434679...	0x1ffe01e8
cycles[9]	float	1.434679...	0x1ffe01ec
Add new expressi			

Συνοψίζοντας τα αποτελέσματα για τις διαφορετικές τιμές των B είναι τα εξής :

	B_1	B_2	B_5	B_10
Average Value	1,742441	1,5203253	1,3691552	1,4346785

Παρατηρούμε ότι ενώ για αύξηση του B μέχρι 5 ο χρόνος βελτιώνεται, πράγμα το οποίο είναι λογικό αφού για μικρό block size έχουμε γενικά αρκετά requests οπότε αυξάνεται το traffic και επομένως και ο χρόνος εκτέλεσης. Παρόλα αυτά παρατηρούμε ότι για Block size

10 ο χρόνος εκτέλεσης αυξήθηκε. Το αποτέλεσμα αυτό δεν είναι το αναμενόμενο αλλά κατά πάσα πιθανότητα οφείλεται στο γεγονός ότι χαλάμε το locality επομένως έχουμε παραπάνω requests και,συνεπώς, περισσότερους κύκλους.

5)

Λαμβάνουμε αποτελέσματα για όλους τους συνδυασμούς Bx και By με τα Bx και By να είναι 1,2,5,10.

Τα αποτελέσματα είναι τα εξής :

Bx_1_By_1	Bx_1_By_2	Bx_1_By_5	Bx_1_By_10
1,742441	1,5641442	1,3379633	1,3367374

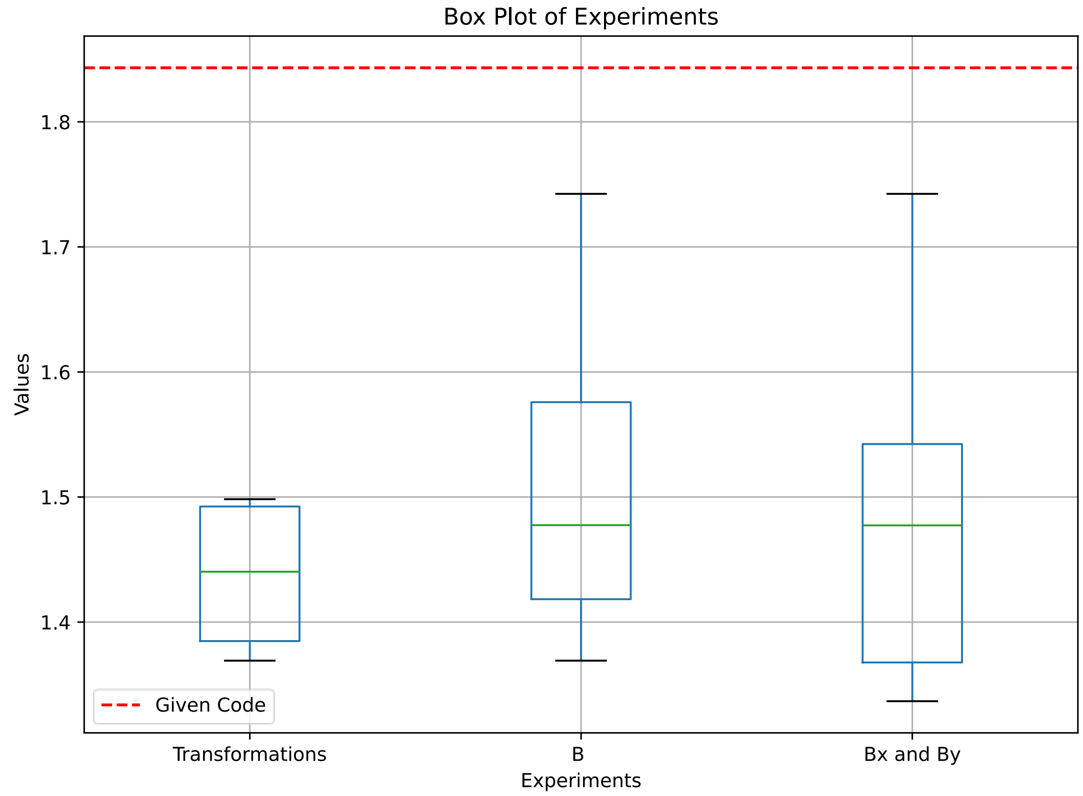
Bx_2_By_1	Bx_2_By_2	Bx_2_By_5	Bx_2_By_10
1,6035652	1,5203253	1,3593955	1,3634512

Bx_5_By_1	Bx_5_By_2	Bx_5_By_5	Bx_5_By_10
1,5408802	1,5199704	1,3691552	1,4027338

Bx_10_By_1	Bx_10_By_2	Bx_10_By_5	Bx_10_By_10
1,5468874	1,5327512	1,4123048	1,4346785

Σχετικά με τα block sizes και το traffic ισχύει ότι αναφέραμε και στο ερώτημα 4 Παρατηρούμε πως πετυχαίνουμε καλύτερο χρόνο για Bx=1,By=10 , ενώ χειρότερο χρόνο για Bx = By =1, παρόλα αυτά αξίζει να σημειωθεί ότι σύμφωνα με την θεωρία εφόσον η c είναι row major γλώσσα το performance αυξάνεται αν έχουμε row-wise accesses.

6)



<u>Άσκηση 2:</u>

1)

Έχοντας ολοκληρώσει την εγκατάσταση του gem5 αρχίζουμε την εκτέλεση των εντολών όπως ζητείται από την εκφώνηση.

Αρχικά, προσομοιώνουμε την εφαρμογή tables.exe με την πρώτη αρχιτεκτονική, η οποία αποτελείται μόνο από τον επεξεργαστή και την κύρια μνήμη , μέσω της εντολής :

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

Ανοίγοντας το φάκελο m5out, και το αντίστοιχο αρχείο stats.txt βλέπουμε πως οι κύκλοι που απαιτούνται για την εκτέλεση του προγράμματος είναι 857689669 , όπως φαίνεται παρακάτω :

```
system.cpu.numCycles                    857689669     # Number of cpu cycles simulated (Cycle)
```

2)

Στη συνέχεια, θα προσομοιώσουμε τη λειτουργία της εφαρμογής για τη δεύτερη αρχιτεκτονική, η οποία έχει επαυξηθεί με L1 Instruction, Data και L2 caches , μέσω της εντολής:

```
$ 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
```

Παρατηρούμε και πάλι τους κύκλους που απαιτούνται και έχουμε το εξής αποτέλεσμα :

```
system.cpu.numCycles                    59787087                 # Number of cpu cycles simulated (Cycle)
```

Βλέπουμε πως οι κύκλοι μειώθηκαν σημαντικά στους 59787087. Κάτι τέτοιο είναι λογικό, καθώς οι μνήμες cache , που είναι τοποθετημένες κοντά στον επεξεργαστή σε σύκγριση με την κύρια μνήμη, αξιοποιώντας την τοπικότητα (τόσο χωρική όσο και χρονική), εξασφαλίζουν την ταχύτερη μεταφορά δεδομένων και εντολών με αποτέλεσμα τη μείωση των απαιτούμενων κύκλων .

3)

Στο επόμενο βήμα, προσομοιώνουμε την εφαρμογή στην αρχιτεκτονική του ερωτήματος 2, αλλά αυτή τη φορά δοκιμάζουμε διάφορες τιμές για το unrolling factor(2,4,8,16,32) , μέσω της εντολής :

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

Για να πραγματοποιήσουμε την εξερεύνηση για όλες τις τιμές αυτές, δημιουργούμε το παρακάτω script σε γλώσσα python, το οποίο εκτελεί το πρόγραμμα OptimizationProblem, που δίνεται ήδη από το εργαστηριακό υλικό. Εκτελούμε το πρόγραμμα για διάφορες τιμές του unrolling factor(το οποίο αποτελεί το στοιχείο x[3] της συνάρτησης “_gem5Simulation(self, x)”, με x τον πίνακα των μεταβλητών για τα L1I,L1D,L2C,UF) .

Ο κώδικας :

```python
import numpy as np
import pandas as pd

from pymoo.algorithms.moo.nsga2 import NSGA2
from pymoo.operators.sampling.rnd import IntegerRandomSampling
from pymoo.operators.crossover.sbx import SBX
from pymoo.operators.mutation.pm import PM
from pymoo.operators.repair.rounding import RoundingRepair
```

```python
from pymoo.optimize import minimize
from pymoo.termination.default import DefaultMultiObjectiveTermination
import matplotlib.pyplot as plt

from pmodules.optimizationProblem import OptimizationProblem

# There are 4 input variables a) L1I cache size, b) L1D cache size, c)
L2 cache size, and d) the loop unrolling factor
nVar = 4
# Define the upper and lower bounds for our input variables
xU = np.asarray([5, 5, 3, 4])
xL = np.asarray([0, 0, 0, 0])


# Define the optimization problem
problem = OptimizationProblem(
    nVar,
    xU,
    xL
)



out = []
out = [0 for i in range(5)]
for l in range(5):
    x = np.asarray([2, 2, 0, l])
    metrics = problem._gem5Simulation(x)
    print(metrics[0])
    out[l] = metrics[0]

n = [i**2 for i in range(1,6)]
plt.plot(n, out, label="Unrolling factor dependance")
plt.legend()
plt.xlabel("Unrolling factor value")
plt.ylabel("latencyCC")
plt.title("Unrolling factor")
plt.savefig("Unrolling.png")
```

Οι κύκλοι οι οποίο χρειάστηκαν για κάθε τιμή του unrolling factor καταγράφονται παρακάτω, σε συνδιασμό με το line plot που περιγράφει τα αποτελέσματα :

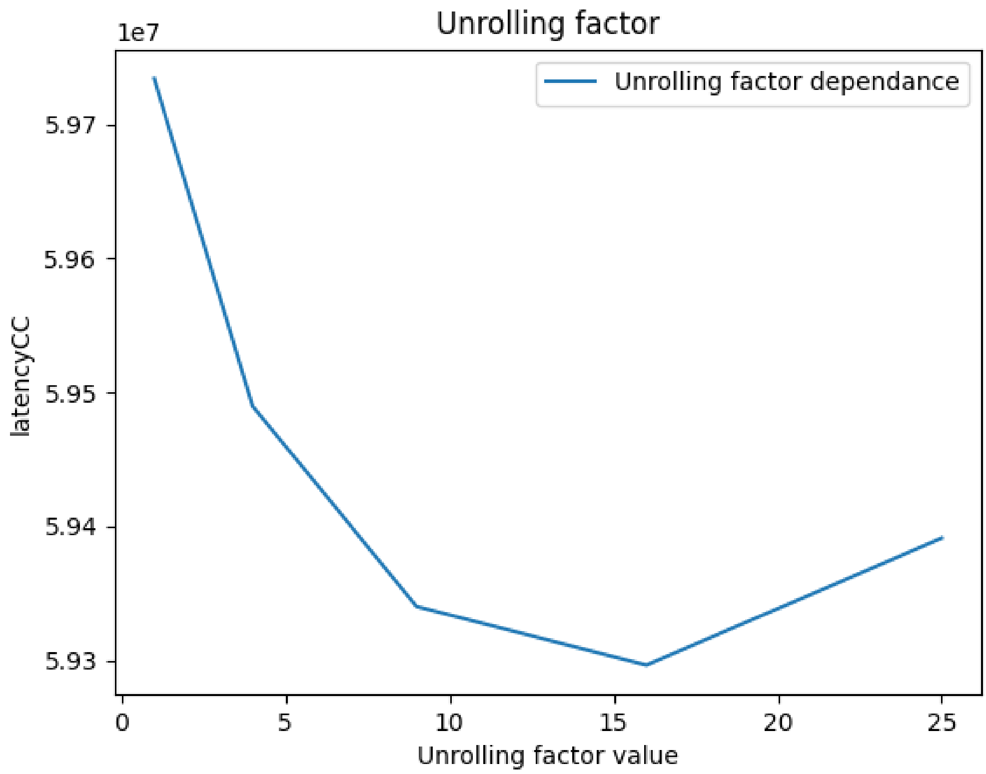
UF = 2 : 59734252.0

UF = 4 : 59489841.0

UF = 8 : 59340239.0

UF = 16 : 59296471.0

UF = 32 : 59391341.0



Επίσης εξερευνούμε την επίδραση διαφορετικών μεγεθών L1 Data cache (2kB, 4kB, 8kB, 16kB, 32kB και 64kB) για τη δεύτερη αρχιτεκτονική μέσω της εντολής :

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

Και πάλι δημιουργούμε ένα δικό μας python script, το οποίο κάνει την εξερεύνηση αυτόματα. Όπως πριν, χρησιμοιούμε το πρόγραμμα OptimizationProblem, στο οποίο ωστόσο θα αγνοήσουμε το unrolling factor,αλλάζοντας την εντολή :

gem5Command = "/gem5/build/X86/gem5.opt /gem5/configs/learning_gem5/part1/two_level.py /gem5/tables_UF/tables_uf" + unrollingFactorSTR + ".exe --l1i_size=" + L1ICacheSizeKBSTR + " --l1d_size=" + L1DCacheSizeKBSTR + " --l2_size=" + L2CacheSizeKBSTR σε :

gem5Command = "/gem5/build/X86/gem5.opt /gem5/configs/learning_gem5/part1/two_level.py /gem5/tables_UF/tables.exe --l1i_size=" + L1ICacheSizeKBSTR + " --l1d_size=" + L1DCacheSizeKBSTR + " --l2_size=" + L2CacheSizeKBSTR

Κώδικας Python για διαφορετικές τιμές L1D cache :

```
import numpy as np
import pandas as pd

from pymoo.algorithms.moo.nsga2 import NSGA2
from pymoo.operators.sampling.rnd import IntegerRandomSampling
from pymoo.operators.crossover.sbx import SBX
from pymoo.operators.mutation.pm import PM
from pymoo.operators.repair.rounding import RoundingRepair
from pymoo.optimize import minimize
from pymoo.termination.default import DefaultMultiObjectiveTermination
import matplotlib.pyplot as plt

from new import OptimizationProblem

# There are 4 input variables a) L1I cache size, b) L1D cache size, c) L2 cache size, and d) the loop unrolling factor
nVar = 4
# Define the upper and lower bounds for our input variables
xU = np.asarray([5, 5, 3, 4])
xL = np.asarray([0, 0, 0, 0])


# Define the optimization problem
problem = OptimizationProblem(
    nVar,
    xU,
    xL
)

out = []
out = [0 for i in range(6)]
for l in range(6):
    x = np.asarray([2, l, 0, 0])
    metrics = problem._gem5Simulation(x)
    print(metrics[0])
    out[l] = metrics[0]

n = [i**2 for i in range(1,7)]
plt.plot(n, out, label="L1 Data Cache dependance")
```

```
plt.legend()
plt.xlabel("L1 Data Cache size")
plt.ylabel("latencyCC")
plt.title("L1 Data Cache")
plt.savefig("L1DCache.png")
```

Οι κύκλοι οι οποίο χρειάστηκαν για κάθε τιμή της L1D cache καταγράφονται παρακάτω, σε συνδιασμό με το line plot που περιγράφει τα αποτελέσματα:

L1D cache = 2 kB : 59973394.0

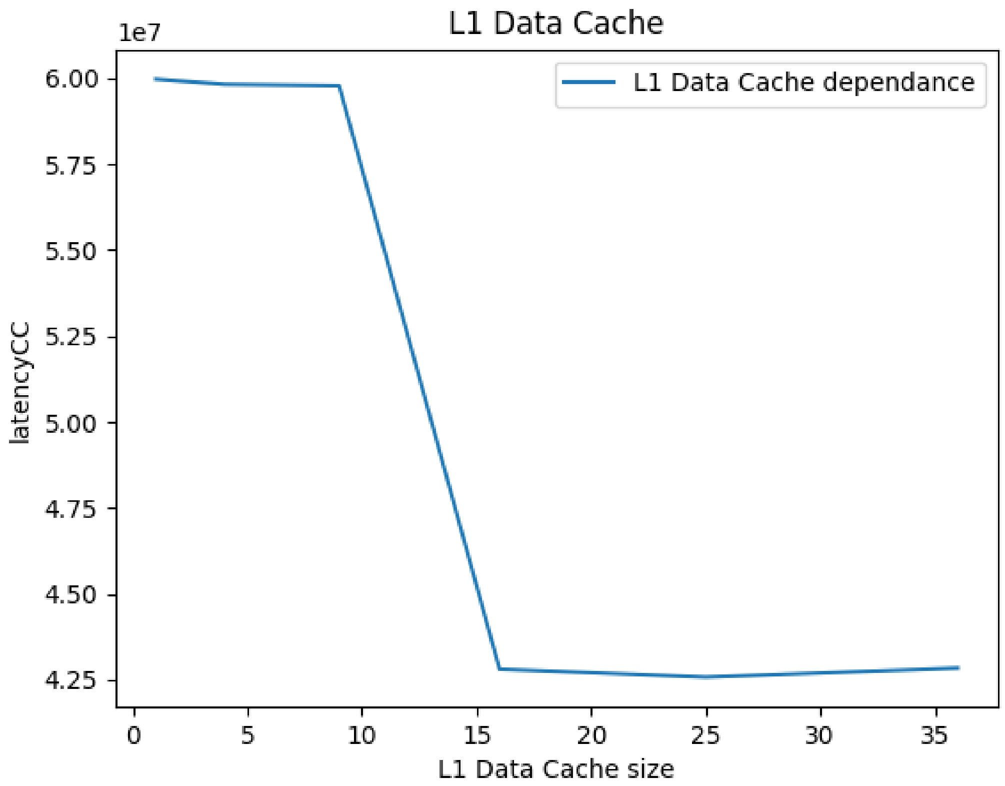
L1D cache = 4 kB : 59824719.0

L1D cache = 8 kB : 59787087.0

L1D cache = 16 kB : 42811763.0

L1D cache = 32 kB : 42580473.0

L1D cache = 64 kB : 42842142.0



Παρατηρήσεις :

Unrolling factor :

Όσον αφορά στο unrolling factor, παρατηρούμε πως γενικώς, μεγαλύτερη τιμή του unrolling factor οδηγεί σε μείωση των απαιτούμενων κύκλων για την εκτέλεση της εφαρμογής. Αυτό είναι αναμενόμενο, καθώς μεγαλύτερη τιμή του unrolling factor διασφαλίζει λιγότερες επαναλήψεις στα loops, και συνεπώς πιο γρήγορη εκτέλεση του προγράμματος. Ωστόσο, παρατηρούμε πως για τη μεγαλύτερη τιμή (UF = 32), η εκτέλεση του προγράμματος απαιτεί περισσότερους κύκλους. Κάτι τέτοιο μπορεί να οφείλεται σε διαφόρους λόγους που έχουν να κάνουν με την αρχιτεκτονική, όπως για παράδειγμα στην χρήση περισσότερων καταχωρητών σε κάθε iteration στο loop ώστε να αποθηκευτούν οι τοπικές μεταβλητές, με αποτέλεσμα τη μείωση της απόδοσης και την απαίτηση περισσότερων κύκλων ρολογιού.

<u>L1 Data Cache :</u>

Όσον αφορά στη data cache, παρατηρούμε και πάλι πως με αύξηση του μεγέθους της έχουμε βελτίωση της απόδοσης.Αυτό είναι λογικό, καθώς όσο μεγαλύτερη η data cache,τόσα περισσότερα είναι τα δεδομένα που μπορούμε να αποθηκεύσουμε και να έχουμε γρήγορη πρόσβαση από τον επεξεργαστή.

4)

Στο ερώτημα αυτό, καλούμαστε να βελτιστοποιήσουμε από κοινού την εφαρμογή και την αρχιτεκτονική στην οποία θα εκτελεστεί. Ο σκοπός μας είναι διπλός να **ελαχιστοποιήσουμε το πλήθος των κύκλων** που απαιτούνται για την εκτέλεση της εφαρμογής μας **αλλά και τη συνολική μνήμη που θα χρησιμοποιήσουμ**ε δηλ. το άθροισμα της L1D, της L1I και της L2 cache σε KB.

Ειδικότερα, θα εκτελέσουμε εξαντλητική αναζήτηση για κάθε συνδιασμό των χαρακτηριστικών, θα βρούμε το pareto front για ελάχιστη χρήση μνήμης και ελάχιστη απαίτηση κύκλων, και θα αποθηκεύσουμε τα αποτελέσματα σε ένα csv αρχείο. Οι τιμές στις οποίες θα γίνει οι αναζήτηση είναι οι εξής :

$$\begin{gathered}
\textbf{L1D cache size} \in [2\text{kB}, 4\text{kB}, 8\text{kB}, 16\text{kB}, 32\text{kB}, 64\text{kB}] \\
\textbf{L1I cache size} \in [2\text{kB}, 4\text{kB}, 8\text{kB}, 16\text{kB}, 32\text{kB}, 64\text{kB}] \\
\textbf{L2 cache size} \in [128\text{kB}, 256\text{kB}, 512\text{kB}, 1024\text{kB}] \\
\textbf{Unrolling factor} \in [2, 4, 8, 16, 32]
\end{gathered}$$

Μέσω της εντολής :

```
$ build/X86/gem5.opt configs/learning_gem5/part1/two_level.py
/gem5/tables_UF/tables_ufXXX.exe --l1i_size=XXX --l1d_size=XXX
-l2_size=XXX
```

Δημιουργούμε εκ νέου ένα python script, το οποίο είναι παρόμοιο με αυτό του ερωτήματος 3 για το unrolling factor, αλλά αυτή τη φορά μεταβάλουμε όλα τα χαρακτηριστικά, δοκιμάζοντας όλες τις πιθανές τιμές. Οι τιμές για τους κύκλους και τη μνήμη αποθηκεύονται στους πίνακες out1 (latencyCC), και out2(TotalMemoryUsed). Στη συνέχεια,σχεδιάζουμε ένα dataframe με τους πίνακες αυτούς, και βρίσκουμε το pareto front για το dataframe αυτό. Τέλος, τα αποτελέσματα αποθηκεύονται σε ένα αρχείο csv.

Ακολουθεί ο κώδικας :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from paretoset import paretoset

from pymoo.algorithms.moo.nsga2 import NSGA2
from pymoo.operators.sampling.rnd import IntegerRandomSampling
from pymoo.operators.crossover.sbx import SBX
from pymoo.operators.mutation.pm import PM
from pymoo.operators.repair.rounding import RoundingRepair
from pymoo.optimize import minimize
from pymoo.termination.default import DefaultMultiObjectiveTermination
import csv

from pmodules.optimizationProblem import OptimizationProblem

# There are 4 input variables a) L1I cache size, b) L1D cache size, c)
L2 cache size, and d) the loop unrolling factor
nVar = 4
# Define the upper and lower bounds for our input variables
xU = np.asarray([5, 5, 3, 4])
xL = np.asarray([0, 0, 0, 0])


# Define the optimization problem
problem = OptimizationProblem(
    nVar,
    xU,
    xL
)


l1i = ['2kB', '4kB', '8kB', '16kB', '32kB', '64kB']
l1d = ['2kB', '4kB', '8kB', '16kB', '32kB', '64kB']
l2c = ['128kB', '256kB', '512kB', '1024kB']
uf = [2, 4, 8, 16, 32]

out1 = []
out1 = [0 for i in range(720)]
out2 = []
out2 = [0 for i in range(720)]
l1ii = []
l1ii = [0 for i in range(720)]
l1dd = []
l1dd = [0 for i in range(720)]
l2cc = []
l2cc = [0 for i in range(720)]
```

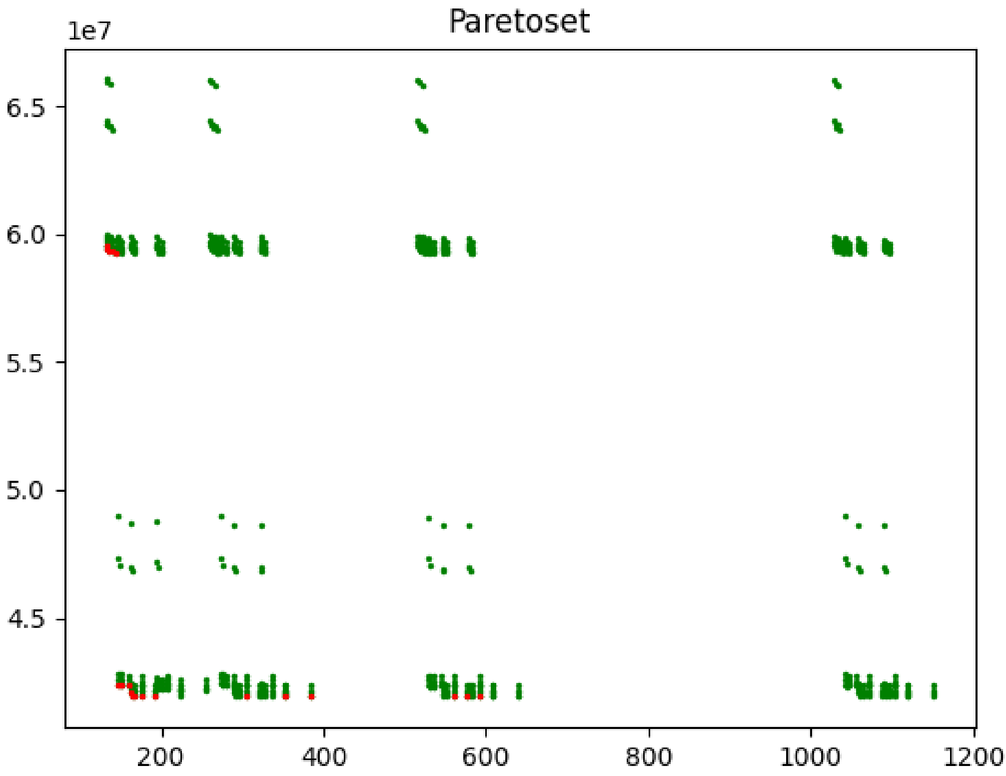
```python
uff = []
uff = [0 for i in range(720)]
a = 0
for i in range(6):
    for j in range(6):
        for k in range(4):
            for l in range(5):
                x = np.asarray([i, j, k, l])
                metrics = problem._gem5Simulation(x)
                out1[a] = metrics[0]
                out2[a] = metrics[1]
                l1ii[a] = l1i[i]
                l1dd[a] = l1d[j]
                l2cc[a] = l2c[k]
                uff[a] = uf[l]
                a = a+1
                print(a)

plt.scatter(out2, out1,s=2, c='green')

best = pd.DataFrame({"ExTimeCC": out1,
                       "TotalMem": out2})
mask = paretoset(best, sense=["min", "min"])
paretofront = np.array(best[mask])

l1iii = pd.DataFrame({"l1i": l1ii})
l1iiii = np.array(l1iii[mask])
l1ddd = pd.DataFrame({"l1d": l1dd})
l1dddd = np.array(l1ddd[mask])
l2ccc = pd.DataFrame({"l1c": l2cc})
l2cccc = np.array(l2ccc[mask])
ufff = pd.DataFrame({"uf": uff})
uffff= np.array(ufff[mask])

length = len(paretofront)

header = ['L1I Cache size', 'L1D Cache size', 'L2 Cache size',
'Unrolling Factor', 'latencyCC', 'totalMemoryKB']

with open('task4.csv', 'w', encoding='UTF8') as f:
    writer = csv.writer(f)
    writer.writerow(header)
    for i in range(0,length):
        print(paretofront[i][0], paretofront[i][1])
        data = [l1iiii[i][0], l1dddd[i][0], l2cccc[i][0], uffff[i][0],
paretofront[i][0] , paretofront[i][1]]
        plt.scatter(paretofront[i][1] , paretofront[i][0],s=2, c='red')
        writer.writerow(data)
```

```
plt.title("Paretoset")
plt.savefig("task4.png")
```

Και το csv αρχείο με το pareto front:

```
L1I Cache size,L1D Cache size,L2 Cache size,Unrolling Factor,latencyCC,totalMemoryKB
2kB,2kB,128kB,8,59581198.0,132.0
2kB,4kB,128kB,8,59433698.0,134.0
2kB,16kB,128kB,8,42416612.0,146.0
2kB,32kB,128kB,8,42078234.0,162.0
4kB,4kB,128kB,16,59343511.0,136.0
4kB,8kB,128kB,16,59306017.0,140.0
4kB,16kB,128kB,16,42409832.0,148.0
4kB,32kB,128kB,16,41969420.0,164.0
8kB,8kB,128kB,16,59296471.0,144.0
8kB,16kB,128kB,16,42403308.0,152.0
8kB,32kB,128kB,16,41969226.0,168.0
16kB,16kB,128kB,16,42387518.0,160.0
16kB,32kB,128kB,16,41945801.0,176.0
16kB,32kB,256kB,16,41933039.0,304.0
16kB,32kB,512kB,16,41923846.0,560.0
16kB,64kB,512kB,16,41922998.0,592.0
32kB,32kB,128kB,16,41936488.0,192.0
32kB,32kB,512kB,16,41923699.0,576.0
64kB,32kB,256kB,16,41928359.0,352.0
64kB,64kB,256kB,16,41925024.0,384.0
```

Μπορούμε να βάλουμε όλες τις τιμές σε ένα διάγραμμα για να επαληθεύσουμε πως το pareto front είναι έγκυρο. Ακολουθεί το διάγραμμα (με κόκκινο χρώμα παρουσιάζονται τα σημεία του pareto front), ωστόσο δεν είναι και το πιο ευανάγνωστο λόγω του μεγάλου πλήθους των σημείων (720 σημεία) :



5)

Στο τελευταίο βήμα της άσκησης, επαναλάμβάνουμε την παραπάνω εξερεύνηση με τη χρήση ενός γενετικού αλγορίθμου, μέσω της εντολής :

```
$ python3 genOptimizer.py
```

Εχούμε πολύ λιγότερα evaluations με χρήση του κώδικα,29 αντί για 720 που έκανε η εξαντλητική αναζήτηση, και παίρνουμε το εξής pareto front :

```
1 L1I_cache_size,L1D_cache_size,L2_cache_size,unrolling_factor,exec_time_cc,total_mem_kB
2 32kB,32kB,256kB,16,41934034,320
3 4kB,16kB,128kB,16,42409832,148
4 8kB,4kB,128kB,8,59378749,140
5 16kB,32kB,128kB,8,42007396,176
6 4kB,4kB,128kB,8,59384448,136
7 32kB,64kB,256kB,16,41929862,352
8 16kB,16kB,128kB,16,42387518,160
```

Παρατηρούμε πως το parent front στην περίπτωση του γενετικού αλγορίθμου είναι διαφορετικό από αυτό που βρήκαμε μέσω της εξαντλητικής εξερεύνησης. Έχει λιγότερα σημεία (7 σημεία) σε σύγκριση με αυτό από το 4^{ο} ερώτημα, με 4 από αυτά να είναι κοινά μεταξύ και των δύο pareto fronts, ενώ 3 ανήκουν μόνο σε αυτό του 5^{ου} ερωτήματος.

Η διαφοροποίηση αυτή είναι λογική και οφείλεται στο γεγονός, πως ο γενετικός αλγόριθμος, με σκοπό να εξοικονομήσει χρόνο εκτέλεσης, προσπερνά την εξέταση ορισμένων σημείων, κάνει πολύ λιγότερα evaluations από την εξαντλητική αναζήτηση (η οποία δοκιμάζει όλες τις 720 τιμές), και επιστρέφει ένα pareto front με αρκετή, αλλά όχι απόλυτη ακρίβεια.