ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

«1η Εργαστηριακή Άσκηση»

Εργαστηριακή Αναφορά στο μάθημα

Σχεδιασμός Ενσωματωμένων Συστημάτων

των φοιτητών

Αβραμίδης Σταύρος Α.Μ. 17811

Λάζου Μαρία-Αργυρώ Α.Μ. 20192

Διδάσκοντες:

Σωτήριος Ξύδης, Δημήτριος Σούντρης, Σωτήριος Κοκόσης

Νοέμβριος 2024

Μέρος 1ο

Εισαγωγή

Στο 1ο μέρος της άσκησης ζητήθηκε η βελτιστοποίηση ενός αλγορίθμου Parallel Hierarchical One Dimensional Search (PHODS). Ο αλγόριθμος PHODS είναι ένας αλγόριθμος εκτίμησης κίνησης (Motion Estimation), ο οποίος έχει στόχο να ανιχνεύσειτη κίνηση των αντικειμένων μεταξύ δύο διαδοχικών εικόνων (frame) του βίντεο. Ο αλγόριθμος βελτιστοποιήθηκε για το S7G2, έναν μικροελεγκτή της Renesas.

Ζητούμενα

Μέσω του Datasheet και SpecSheet του dev-board εξάγουμε τις παρακάτω πληροφορίες:[1]

- 640KB SRAM
- 4MB Flash
- 240MHz CPU clock frequency
- Cortex-M4 CPU
- Flash Cache
  - Flash cache 1 - Instruction cache: 256 bytes
  - Flash cache 2 - CPU operands: 16 bytes
  - Prefetch Buffer: 32 bytes

Από την σελίδα 1763 του *«Groups User manual»* για τον εν λόγω μικροελεγκτή, μπορούμε να δούμε ότι η FCACHE είναι απενεργοποιημένη. Πάραυτα, το BSP ενεργοποιεί την cache, όταν καλείται η ρουτίνα του Reset Handle, κατά την αρχικοποίηση των ρολογιών (bps_clocks.c).

Υλοποίηση

Για τον σκοπό της άσκησης, η σημαία βελτιστοποιήσεων του μεταγλωττιστή τέθηκε σε *«-O0»*. Για τον λόγο αυτό, πέρα των loop transformation, που είναι ο κύριος σκοπός της άσκησης, έγιναν και μερικές περαιτέρω βελτιστοποιήσεις, πού ανώτερα επίπεδα βελτιστοποίησης θα μπορούσαν να εξαλείψουν. Την ίδια στιγμή, όμως, αποφεύχθηκαν βελτιστοποιήσεις, που θα άλλαζαν την βασική δομή του αλγορίθμου (λ.χ. μετατροπή int σε uint8_t), θεωρώντας τες εκτός του πλαισίου της άσκησης.

Αναλυτικότερα στον δοσμένο κώδικα, έγιναν οι εξής βελτιστοποιήσεις:

1. Loop Merging (x3) στις των loop i, k, l. (Ακόλουθο αυτού, sub-expression elimination του p1)
2. Loop Merging των 2 εξωτερικών loop (i,j) και (x,y) του κύριου αλγορίθμου και τις αρχικοποίησης των vectors_x και vectors_y.
3. Απαλοιφή των πολλαπλών memory accesses των vectors_x[x][y] και vectors_y[x][y].
4. Loop Unrolling του loop i από -S εώς S+1. *
   - Αυτό έχει ως αποτέλεσμα την απλοποίηση της περίπτωσης για i=0, αφού οι συνθήκες των if/else είναι κοινές και κατα συνέπεια p2 = q2 και κατ'επέκταση p1-q2 = p1-p2.

[1] «https://www.renesas.com/en/products/microcontrollers-microprocessors/renesas-synergy-platform-mcus/s7g2-240-mhz-arm-cortex-m4-cpu#documents»

5. SubExpression Elimination για όλες τις πράξεις που γίνονται πάνω από μια φορά εντός των εσωτερικών των loop (i, k, l).
6. Υλοποίηση των πολλαπλασιασμών στα x,y loop, ως running sum, αντί για πολλαπλασιασμό κάθε φορά.
7. Τέλος, παρατηρούμε ότι η εντολές των if/else που προσαυξάνουν τα distx και disty, πραγματοποιούν πρακτικά μια πράξη abs. Για τον λόγο αυτό, η πράξη της αφαίρεσης και αφαίρεσης του προσήμου υλοποιήθηκε με βέλτιστο τρόπο σε assembly. (Συνοπτική επεξήγηση, ακολουθεί παρακάτω).

* Δεν εξετάστηκε, αν αντί για unroll, ήταν ευνοϊκότερη αντ' αυτού η χρήση if-else για την περίπτωση του i=0, με την προϋπόθεση ότι το loop body θα χωράει στην instructions cache.

<u>Η ορθότητα του αλγορίθμου επαληθεύτηκε με την χρήση του randomized unit testing, στον προσωπικό μας υπολογιστή.</u>

Ο τελικός κώδικας, μετά τις βελτιστοποιήσεις, είναι ο εξής:

phods.h

```c
#include "inttypes.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. */


__attribute__((always_inline))
inline int32_t sub_and_abs(int32_t a, int32_t b) {
  int32_t result;
  __asm__("subs %0, %1, %2\n" // result = a - b, sets flags
          "it mi\n" // If result is negative (MI), execute next instruction
          "negmi %0, %0\n" // If negative, result = -result
          : "=r"(result)   // output
          : "r"(a), "r"(b) // inputs
          : "cc"           // clobbers the condition codes
  );
  return result;
}

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, k, l, p1, p2, q2, distx, disty, S, min1, min2, bestx, besty;

  distx = 0;
  disty = 0;

  /*For all blocks in the current frame*/
  int B_times_x = 0;

  for (x = 0; x < N / B; x++) {

    int B_times_y = 0;

    for (y = 0; y < M / B; y++) {

      /*Initialize the vector motion matrices*/
      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 and Y dimension*/

        const int vec_x_xy = vectors_x[x][y];
        const int vec_y_xy = vectors_y[x][y];
        const int B_times_x_plus_vec_x_xy = B_times_x + vec_x_xy;
        const int B_times_y_plus_vec_y_xy = B_times_y + vec_y_xy;

        {
          const int i = -S;
          distx = 0;
          disty = 0;

          const int B_times_x_plus_vec_x_xy_plus_i =
              B_times_x_plus_vec_x_xy + i;
          const int B_times_y_plus_vec_y_xy_plus_i =
              B_times_y_plus_vec_y_xy + i;

          /*For all pixels in the block*/
          for (k = 0; k < B; k++) {
            const int B_times_x_plus_vec_x_xy_plus_i_plus_k =
                B_times_x_plus_vec_x_xy_plus_i + k;

            const int B_times_x_plus_vec_x_xy_plus_k =
                B_times_x_plus_vec_x_xy + k;

            const int B_times_x_plus_k = B_times_x + k;

            for (l = 0; l < B; l++) {
              const int B_times_y_plus_vec_y_xy_plus_i_plus_l =
                  B_times_y_plus_vec_y_xy_plus_i + l;

              const int B_times_y_plus_vec_y_xy_plus_l =
                  B_times_y_plus_vec_y_xy + l;

              p1 = current[B_times_x_plus_k][B_times_y + l];

              if ((B_times_x_plus_vec_x_xy_plus_i_plus_k) < 0 ||
                  (B_times_x_plus_vec_x_xy_plus_i_plus_k) > (N - 1) ||
                  (B_times_y_plus_vec_y_xy_plus_l) < 0 ||
                  (B_times_y_plus_vec_y_xy_plus_l) > (M - 1)) {
                p2 = 0;
              } else {
                p2 = previous[B_times_x_plus_vec_x_xy_plus_i + k]
                             [B_times_y_plus_vec_y_xy_plus_l];
              }

              if ((B_times_x_plus_vec_x_xy_plus_k) < 0 ||
                  (B_times_x_plus_vec_x_xy_plus_k) > (N - 1) ||
                  (B_times_y_plus_vec_y_xy_plus_i_plus_l) < 0 ||
                  (B_times_y_plus_vec_y_xy_plus_i_plus_l) > (M - 1)) {
                q2 = 0;
              } else {
                q2 = previous[B_times_x_plus_vec_x_xy_plus_k]
                             [B_times_y_plus_vec_y_xy_plus_i_plus_l];
              }

              distx += sub_and_abs(p1, p2);
              disty += sub_and_abs(p1, q2);
            }
          }

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

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

        {
          const int i = 0;

          distx = 0;
          disty = 0;

          /*For all pixels in the block*/
          for (k = 0; k < B; k++) {
            const int B_times_x_plus_vec_x_xy_plus_k =
```

```
                B_times_x_plus_vec_x_xy + k;

            const int B_times_x_plus_k = B_times_x + k;

            for (l = 0; l < B; l++) {
              const int B_times_y_plus_vec_y_xy_plus_l =
                  B_times_y_plus_vec_y_xy + l;

              p1 = current[B_times_x_plus_k][B_times_y + l];

              if ((B_times_x_plus_vec_x_xy_plus_k) < 0 ||
                  (B_times_x_plus_vec_x_xy_plus_k) > (N - 1) ||
                  (B_times_y_plus_vec_y_xy_plus_l) < 0 ||
                  (B_times_y_plus_vec_y_xy_plus_l) > (M - 1)) {
                p2 = 0;
              } else {
                p2 = previous[B_times_x_plus_vec_x_xy_plus_k]
                             [B_times_y_plus_vec_y_xy_plus_l];
              }

              int res = sub_and_abs(p1, p2);
              distx += res;
              disty += res;
            }
          }

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

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

        {
          const int i = S;
          distx = 0;
          disty = 0;

          const int B_times_x_plus_vec_x_xy_plus_i =
              B_times_x_plus_vec_x_xy + i;
          const int B_times_y_plus_vec_y_xy_plus_i =
              B_times_y_plus_vec_y_xy + i;

          /*For all pixels in the block*/
          for (k = 0; k < B; k++) {
            const int B_times_x_plus_vec_x_xy_plus_i_plus_k =
                B_times_x_plus_vec_x_xy_plus_i + k;

            const int B_times_x_plus_vec_x_xy_plus_k =
                B_times_x_plus_vec_x_xy + k;

            const int B_times_x_plus_k = B_times_x + k;

            for (l = 0; l < B; l++) {
              const int B_times_y_plus_vec_y_xy_plus_i_plus_l =
                  B_times_y_plus_vec_y_xy_plus_i + l;

              const int B_times_y_plus_vec_y_xy_plus_l =
                  B_times_y_plus_vec_y_xy + l;

              p1 = current[B_times_x_plus_k][B_times_y + l];

              if ((B_times_x_plus_vec_x_xy_plus_i_plus_k) < 0 ||
                  (B_times_x_plus_vec_x_xy_plus_i_plus_k) > (N - 1) ||
                  (B_times_y_plus_vec_y_xy_plus_l) < 0 ||
                  (B_times_y_plus_vec_y_xy_plus_l) > (M - 1)) {
                p2 = 0;
              } else {
                p2 = previous[B_times_x_plus_vec_x_xy_plus_i + k]
                             [B_times_y_plus_vec_y_xy_plus_l];
              }

              if (B_times_x_plus_vec_x_xy_plus_k < 0 ||
                  B_times_x_plus_vec_x_xy_plus_k > (N - 1) ||
                  (B_times_y_plus_vec_y_xy_plus_i_plus_l) < 0 ||
                  (B_times_y_plus_vec_y_xy_plus_i_plus_l) > (M - 1)) {
                q2 = 0;
```

```
                } else {
                  q2 = previous[B_times_x_plus_vec_x_xy_plus_k]
                               [B_times_y_plus_vec_y_xy_plus_i_plus_l];
                }

                distx += sub_and_abs(p1, p2);
                disty += sub_and_abs(p1, q2);
              }
            }

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

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

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

        // Reduce B * y to addition
        B_times_y += B;
      }

      // Reduce B * x
      B_times_x += B;
    }
  }
```

hal_entry.c

```
#include "stdio.h"
#include "string.h"

#include "common_utils.h"
#include "hal_data.h"


#include "image.h"
#include "phods.h"

void hal_entry(void) {

  // Initialize your variables here
  int motion_vectors_x[N / B][M / B], motion_vectors_y[N / B][M / B], i, j;

  uint32_t timer;

  // Code to initialize the DWT->CYCCNT register
  CoreDebug->DEMCR |= 0x01000000;
  ITM->LAR = 0xC5ACCE55;
  DWT->CYCCNT = 0;
  DWT->CTRL |= 1;

  // Initial

  APP_PRINT("Embedded Systems - Lab1\n\n");
  APP_PRINT("Omada 24 - Maria Lazou, Avramidis Stavros\n\n");

  APP_PRINT("System Running @%u MHz\n\n", SystemCoreClock / 1000000);

  APP_PRINT("Running for B: %d\n", Bx, By);

  for (int i = 0; i < 10; i++) {
    timer = DWT->CYCCNT;
    phods_motion_estimation_naive(current, previous, motion_vectors_x,
                                  motion_vectors_y);
    timer = DWT->CYCCNT - timer;

    APP_PRINT("Time for naive loop[%d]: %u\n", i, timer);
  }

```

```
42    APP_PRINT("\n");
43
44    for (int i = 0; i < 10; i++) {
45      timer = DWT->CYCCNT;
46      phods_motion_estimation_opt(current, previous, motion_vectors_x,
47                                  motion_vectors_y);
48      timer = DWT->CYCCNT - timer;
49
50      APP_PRINT("Time for opt loop[%d]: %u\n", i, timer);
51    }
52
53    while (1)
54      ;
55  }
```

Όσον αφορά την υλοποίηση της πράξης της αφαίρεσης με βέλτιστο τρόπο σε assembly, έγινε η ακόλουθη υλοποίηση:

```
SUBS %0, %1, %2
TI
NEGMI %0, %0
```

Αρχικά, υλοποιούμε την αφαίρεση, η οποία ενημερώνει το αντίστοιχο flag, αν το αποτέλεσμα της είναι αρνητικό. Κατόπιν εισάγουμε ένα IT (If/Then) block[2], που θα εκτελεστεί μόνο αν η συνθήκη του Negative flag, έχει τεθεί. Έτσι, μπορούμε να κάνουμε τις πράξεις της αφαίρεσης και abs(), με μόνο 2 εντολές κι είναι και branchless, μη επιβαρύνοντας έτσι τον branch predictor.

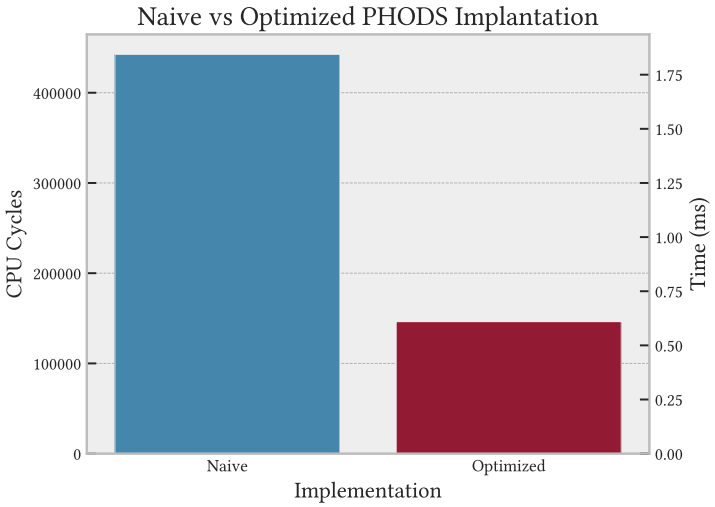
Χρόνοι εκτέλεσης

Ο αρχικός χρόνος εκτέλεσης του αλγορίθμου ήταν 442,530 κύκλοι ή 1.84ms. Μετά τις βελτιστοποιήσεις, ο χρόνος εκτέλεσης μειώθηκε στους 146,283 κύκλους ή 0.61ms.

Υλοποίηση	Κύκλοι	Χρόνος Εκτέλεσης (ms)
Naive	442530	1.84
Optimized	146283	0.61

Πίνακας 1: Χρονοι εκτέλεσης πριν και μετά τις βελτιστοποιήσεις

[2]Το IT block (If-Then block) στους επεξεργαστές ARM είναι ένα χαρακτηριστικό του instruction set της αρχιτεκτονικής ARM Thumb-2, που επιτρέπει την εκτέλεση υπό συνθήκη μιας ομάδας εντολών
https://developer.arm.com/documentation/den0042/a/Unified-Assembly-Language-Instructions/Instruction-set-basics/Conditional-execution



Σχήμα 1: Σύγκριση χρόνων εκτέλεσης

Καταλήγουμε λοιπόν να έχουν Speedup = 3.025.

Ενδιαφέρον επίσης παρουσιάζει η απενεργοποίηση της FCACHE, κι η σύγκριση του χρόνου του αλγορίθμου. Με την επίδοση να πέφτει στους 184285 κύκλους (26% Αύξηση!).

Μεταβολή του Block Size

Στην συνέχεια ζητήθηκε, να συγκριθεί και να αναλυθεί η επίδραση του Block Size (B) στον χρόνο εκτέλεσης του αλγορίθμου.

B	Κύκλοι	Χρόνος Εκτέλεσης (ms)
1	273050	1.14
2	179422	0.75
5	146283	0.61
10	139770	0.58

Πίνακας 2: Συγκριτικά αποτελέσματα ανά Block Size

Παρατηρούμε ότι όσο μεγαλύτερο είναι το Block Size, τόσο μειώνεται ο χρόνος εκτέλεσης του αλγορίθμου. Αυτό είναι λογικό, αφού στον αλγόριθμο έχουμε 2 ομάδες loop, τα εξωτερικά(x,y) που είναι αντιστρόφως ανάλογα του B και τα εσωτερικά (k,l), πού είναι ανάλογα του B. Ο συνολικός αριθμός επαναλήψεων συνεπώς μένει ίδιος, όμως όσο το φόρτο των υπολογισμών μετατοπίζεται προς το εσωτερικό, τόσο τα memory accesses, αλλά και διάφοροι υπολογισμοί μειώνονται. Την ίδια στιγμή δίνεται η ευκαιρία να «χτυπήσουμε» περισσότερο την instruction cache, αφού ο αλγόριθμός περνάει μεγαλύτερο χρονικό διάστημα στον ίδιο κώδικα.

Ορθογώνιο Παράθυρο

Στην συνέχεια, ζητήθηκε η υλοποίηση του αλγορίθμου με ορθογώνιο παράθυρο, διαστάσεων B_x, B_y. Η υλοποίηση τροποποιείται όπως επισημαίνεται παρακάτω:

phods_rect.h

```
#include "inttypes.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. */

#define Bx (5)
#define By (5)

__attribute__((always_inline))
inline int32_t sub_and_abs(int32_t a, int32_t b) {
  int32_t result;
  __asm__("subs %0, %1, %2\n" // result = a - b, sets flags
          "it mi\n" // If result is negative (MI), execute next instruction
          "negmi %0, %0\n" // If negative, result = -result
          : "=r"(result)   // output
          : "r"(a), "r"(b) // inputs
          : "cc"           // clobbers the condition codes
  );
  return result;
}

void phods_motion_estimation_opt(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, k, l, p1, p2, q2, distx, disty, S, min1, min2, bestx, besty;

  distx = 0;
  disty = 0;

  /*For all blocks in the current frame*/
  int B_times_x = 0;

  for (x = 0; x < N / Bx; x++) {

    int B_times_y = 0;

    for (y = 0; y < M / By; y++) {

      /*Initialize the vector motion matrices*/
      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 and Y dimension*/

        const int vec_x_xy = vectors_x[x][y];
        const int vec_y_xy = vectors_y[x][y];
        const int B_times_x_plus_vec_x_xy = B_times_x + vec_x_xy;
        const int B_times_y_plus_vec_y_xy = B_times_y + vec_y_xy;

        {
          const int i = -S;
          distx = 0;
          disty = 0;

          const int B_times_x_plus_vec_x_xy_plus_i =
              B_times_x_plus_vec_x_xy + i;
          const int B_times_y_plus_vec_y_xy_plus_i =
              B_times_y_plus_vec_y_xy + i;

          /*For all pixels in the block*/
```

```
            for (k = 0; k < Bx; k++) {
              const int B_times_x_plus_vec_x_xy_plus_i_plus_k =
                  B_times_x_plus_vec_x_xy_plus_i + k;

              const int B_times_x_plus_vec_x_xy_plus_k =
                  B_times_x_plus_vec_x_xy + k;

              const int B_times_x_plus_k = B_times_x + k;

              for (l = 0; l < By; l++) {
                const int B_times_y_plus_vec_y_xy_plus_i_plus_l =
                    B_times_y_plus_vec_y_xy_plus_i + l;

                const int B_times_y_plus_vec_y_xy_plus_l =
                    B_times_y_plus_vec_y_xy + l;

                p1 = current[B_times_x_plus_k][B_times_y + l];

                if ((B_times_x_plus_vec_x_xy_plus_i_plus_k) < 0 ||
                    (B_times_x_plus_vec_x_xy_plus_i_plus_k) > (N - 1) ||
                    (B_times_y_plus_vec_y_xy_plus_l) < 0 ||
                    (B_times_y_plus_vec_y_xy_plus_l) > (M - 1)) {
                  p2 = 0;
                } else {
                  p2 = previous[B_times_x_plus_vec_x_xy_plus_i + k]
                               [B_times_y_plus_vec_y_xy_plus_l];
                }

                if ((B_times_x_plus_vec_x_xy_plus_k) < 0 ||
                    (B_times_x_plus_vec_x_xy_plus_k) > (N - 1) ||
                    (B_times_y_plus_vec_y_xy_plus_i_plus_l) < 0 ||
                    (B_times_y_plus_vec_y_xy_plus_i_plus_l) > (M - 1)) {
                  q2 = 0;
                } else {
                  q2 = previous[B_times_x_plus_vec_x_xy_plus_k]
                               [B_times_y_plus_vec_y_xy_plus_i_plus_l];
                }

                distx += sub_and_abs(p1, p2);
                disty += sub_and_abs(p1, q2);
              }
            }

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

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

          {
            const int i = 0;

            distx = 0;
            disty = 0;

            /*For all pixels in the block*/
            for (k = 0; k < Bx; k++) {
              const int B_times_x_plus_vec_x_xy_plus_k =
                  B_times_x_plus_vec_x_xy + k;

              const int B_times_x_plus_k = B_times_x + k;

              for (l = 0; l < By; l++) {
                const int B_times_y_plus_vec_y_xy_plus_l =
                    B_times_y_plus_vec_y_xy + l;

                p1 = current[B_times_x_plus_k][B_times_y + l];

                if ((B_times_x_plus_vec_x_xy_plus_k) < 0 ||
                    (B_times_x_plus_vec_x_xy_plus_k) > (N - 1) ||
                    (B_times_y_plus_vec_y_xy_plus_l) < 0 ||
                    (B_times_y_plus_vec_y_xy_plus_l) > (M - 1)) {
                  p2 = 0;
                } else {
                  p2 = previous[B_times_x_plus_vec_x_xy_plus_k]
                               [B_times_y_plus_vec_y_xy_plus_l];
```

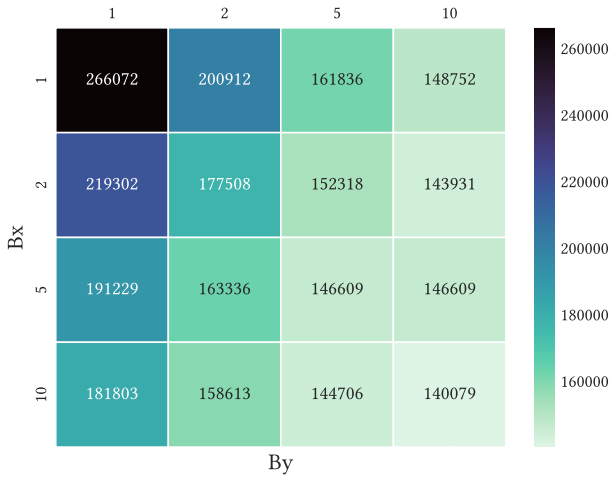
```
153                 }
154
155                 int res = sub_and_abs(p1, p2);
156                 distx += res;
157                 disty += res;
158               }
159             }
160
161             if (distx < min1) {
162               min1 = distx;
163               bestx = i;
164             }
165
166             if (disty < min2) {
167               min2 = disty;
168               besty = i;
169             }
170           }
171
172           {
173             const int i = S;
174             distx = 0;
175             disty = 0;
176
177             const int B_times_x_plus_vec_x_xy_plus_i =
178                 B_times_x_plus_vec_x_xy + i;
179             const int B_times_y_plus_vec_y_xy_plus_i =
180                 B_times_y_plus_vec_y_xy + i;
181
182             /*For all pixels in the block*/
183             for (k = 0; k < Bx; k++) {
184               const int B_times_x_plus_vec_x_xy_plus_i_plus_k =
185                   B_times_x_plus_vec_x_xy_plus_i + k;
186
187               const int B_times_x_plus_vec_x_xy_plus_k =
188                   B_times_x_plus_vec_x_xy + k;
189
190               const int B_times_x_plus_k = B_times_x + k;
191
192               for (l = 0; l < By; l++) {
193                 const int B_times_y_plus_vec_y_xy_plus_i_plus_l =
194                     B_times_y_plus_vec_y_xy_plus_i + l;
195
196                 const int B_times_y_plus_vec_y_xy_plus_l =
197                     B_times_y_plus_vec_y_xy + l;
198
199                 p1 = current[B_times_x_plus_k][B_times_y + l];
200
201                 if ((B_times_x_plus_vec_x_xy_plus_i_plus_k) < 0 ||
202                     (B_times_x_plus_vec_x_xy_plus_i_plus_k) > (N - 1) ||
203                     (B_times_y_plus_vec_y_xy_plus_l) < 0 ||
204                     (B_times_y_plus_vec_y_xy_plus_l) > (M - 1)) {
205                   p2 = 0;
206                 } else {
207                   p2 = previous[B_times_x_plus_vec_x_xy_plus_i + k]
208                                [B_times_y_plus_vec_y_xy_plus_l];
209                 }
210
211                 if (B_times_x_plus_vec_x_xy_plus_k < 0 ||
212                     B_times_x_plus_vec_x_xy_plus_k > (N - 1) ||
213                     (B_times_y_plus_vec_y_xy_plus_i_plus_l) < 0 ||
214                     (B_times_y_plus_vec_y_xy_plus_i_plus_l) > (M - 1)) {
215                   q2 = 0;
216                 } else {
217                   q2 = previous[B_times_x_plus_vec_x_xy_plus_k]
218                                [B_times_y_plus_vec_y_xy_plus_i_plus_l];
219                 }
220
221                 distx += sub_and_abs(p1, p2);
222                 disty += sub_and_abs(p1, q2);
223               }
224             }
225
226             if (distx < min1) {
227               min1 = distx;
228               bestx = i;
229             }
230
231             if (disty < min2) {
232               min2 = disty;
```

```
            besty = i;
          }
        }

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

      // Reduce B * y to addition
      B_times_y += By;
    }

    // Reduce B * x
    B_times_x += Bx;
  }
}
```

Οι χρόνοι εκτέλεσης του αλγορίθμου με ορθογώνιο παράθυρο, για διάφορες τιμές των (B_x, B_y), παρουσιάζονται στον παρακάτω πίνακα:

Bx\By	**1**	**2**	**5**	**10**
1	266072	200912	161836	148752
2	219302	177508	152318	143931
5	191229	163336	146609	146609
10	181803	158613	144706	140079

Πίνακας 3: Συγκεντρωτικά αποτελέσματα ανά \(B_x, B_y\)



Σχήμα 2: Χρόνοι εκτέλεσης ανά \(B_x, B_y\)

Για τους ίδιους λόγους που αναφέρθηκαν και παραπάνω, καλύτερο χρόνο επιτυγχάνει η υλοποίηση για $B_x = 10, B_y = 10$.

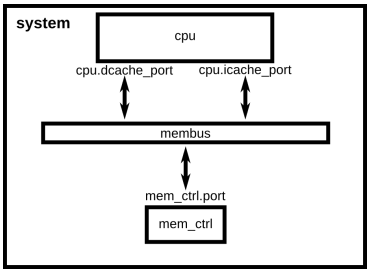
Ευριστικός Αλγόριθμος.

Μία ευριστική μέθοδος, για την ανίχνευση κινήσεων στα frames, θα ήταν η τυχαία αναζήτηση block, κι όχι η αναζήτηση όλων των block. Αν το block, εμφανίζει αλλαγές μπορούμε να κινηθούμε ανάλογα με αυτές. Πρακτικά, ο αριθμός ο αριθμός των τυχαίων block που επιλέγουμε σε κάθε βήμα, αποτελεί και το quality factor του αλγορίθμου. Τέλος, η δημιουργία sub-blocks, θα μπορούσε να βελτιώσει την ποιότητα της συμπίεσης (όπως λ.χ. το H.265/HEVC).

Μέρος 2ο

1.

Προσομοιώνουμε πρώτα στο gem5 την κάτωθι αρχιτεκτονική (CPU + main memory):



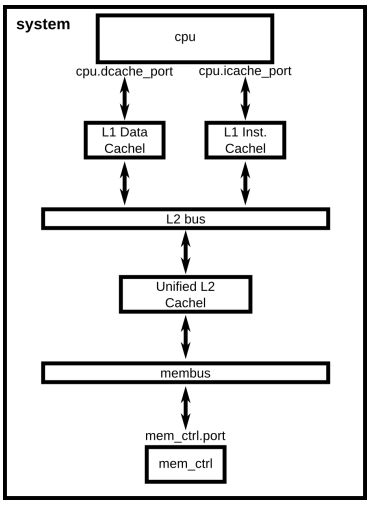
Σχήμα 3: Αρχιτεκτονική 1

και καταγράφουμε τους κύκλους που απαιτήθηκαν:

```
root@6262902d3b2f:/gem5# cat m5out/stats.txt | grep system.cpu.numCycles
system.cpu.numCycles                          858895958       # Number of cpu cycles simulated (Cycle)
```

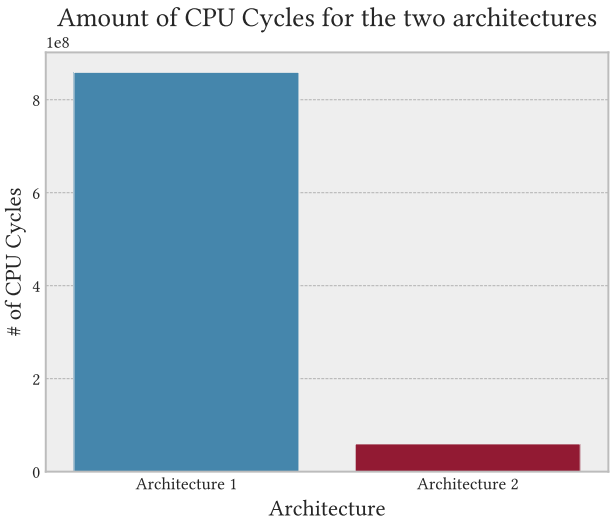
2.

Ομοίως για την ακόλουθη αρχιτεκτονική:



Σχήμα 4: Αρχιτεκτονική 2

```
root@6262902d3b2f:/gem5# cat m5out/stats.txt | grep system.cpu.numCycles
system.cpu.numCycles                         59723234      # Number of cpu cycles simulated (Cycle)
```



Σχήμα 5: Σύγκριση συνολικών κύκλων εκτέλεσης αρχιτεκτονικών

Είναι προφανές, ότι με την προσθήκη κρυφής μνήμης μειώνεται σημαντικά το latency. Τα δεδομένα βρίσκονται πιο κοντά στον επεξεργαστή και δεν χρειάζεται να πληρώνουμε το overhead για την μεταφορά τους από την κύρια μνήμη σε κάθε single operation παρά μόνο όταν έχουμε misses. Επιπρόσθετα, ο επεξεργαστής που προσημειώνουμε (X86TimingSimpleCPU[3]), δεν έχει pipeline και χρειάζεται να περιμένει την ολοκλήρωση των προηγούμενων εντολών πριν εκτελέσει τις επόμενες. Συνεπώς, η προσθήκη κρυφής μνήμης και κατ'επέκταση μείωση του χρόνου αναμονής των δεδομένων, μειώνει το stall time και τον συνολικό αριθμό κύκλων εκτέλεσης. Τέλος, η προσθήκη της Instruction L1 cache, είναι σίγουρα οφέλη, αφού το πρόγραμμα υπό εξέταση, απαρτίζεται κατα κύριο λόγο δύο for-loop, οπότε εκτελούνται συνεχώς τα ίδια κομμάτια κώδικα.

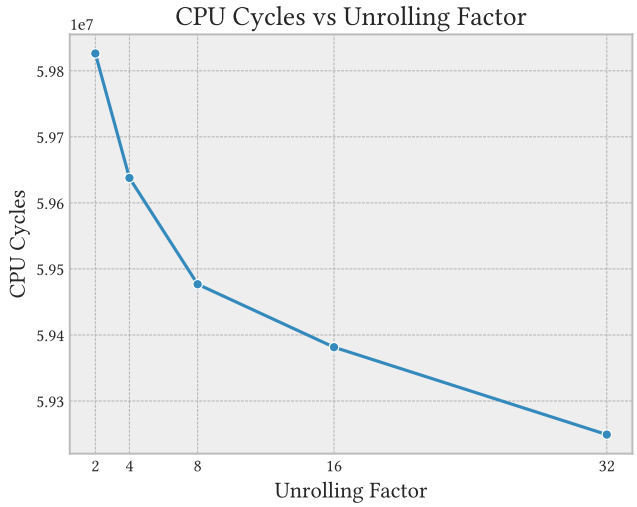
3.

Με το ακόλουθο script τρέχουμε τις προσομοιώσεις με τις διαφορετικές τιμές unrolling factor (2, 4, 8, 16, 32) και απεικονίζουμε τα αποτελέσματα:

unrolling.sh

```
#!/bin/bash

outfile="unrolling.txt"

rm -f $outfile
touch $outfile
echo -e "Unrolling Factor system.cpu.numCycles " >>$outfile
for i in 2 4 8 16 32; do
    build/X86/gem5.opt configs/learning_gem5/part1/two_level.py /gem5/tables_UF/tables_uf$i.exe --l1i_size=8kB --l1d_size=8kB --l2_size=128kB
    echo -ne "${i}:\t\t\t" >>$outfile
    cat m5out/stats.txt | grep system.cpu.numCycles | sed 's/[^0-9]*//g' >>$outfile
done
```

[3] https://www.gem5.org/documentation/general_docs/cpu_models/SimpleCPU#timingsimplecpu



Σχήμα 6: Συγκριτικό διάγραμμα για το Unrolling Factor

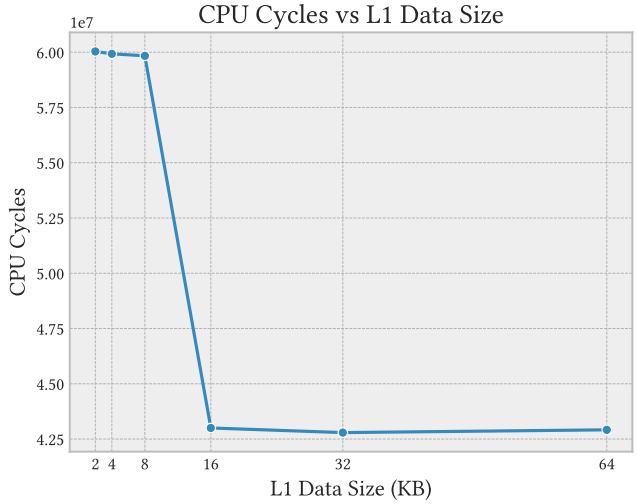
Ο αριθμός των κύκλων μειώνεται σημαντικά όσο το unrolling factor αυξάνεται. Δεδομένου ότι, έχουμε single threaded εκτέλεση σε επεξεργαστή χωρίς pipelining, η βελτίωση οφείλεται αποκλειστικά στην μείωση του overhead των μηχανισμών των loops (κόστος αύξησης loop counter, σύγκριση κλπ.)

4.

Με το ακόλουθο script τρέχουμε τις προσομοιώσεις με τα διαφορετικά μεγέθη L1 Data Cache (2kB, 4kB, 8kB, 16kB, 32kB, 64kB) και απεικονίζουμε τα αποτελέσματα:

cache.sh

```
#!/bin/bash

outfile="cache.txt"

rm -f $outfile
touch $outfile
echo -e "L1D Size(KB) system.cpu.numCycles " >>$outfile
for i in 2 4 8 16 32 64; do
        build/X86/gem5.opt  configs/learning_gem5/part1/two_level.py  /gem5/tables_UF/tables.exe  --l1i_size=8kB --l1d_size=${i}kB --l2_size=128kB
    echo -ne "${i}:\t\t\t" >>$outfile
    cat m5out/stats.txt | grep system.cpu.numCycles | sed 's/[^0-9]*//g' >>$outfile
done
```



Σχήμα 7: Συγκριτικό διάγραμμα για το L1 Data Cache Size

Αυξάνοντας το μέγεθος της L1 Data Cache παρατηρούμε μικρή βελτίωση από τα 2KB έως 8KB και ραγδαία επιτάχυνση από τα 8KB στα 16KB, ενώ μετά δεν υπάρχει κλιμάκωση. Ανατρέχοντας στην υλοποίηση της εν λόγω L1 cache[4], βρίσκουμε ότι πρόκειται για μία 2-way associative cache. Εικάζουμε, λοιπόν, πώς η αύξηση του μεγέθους της cache, επιτρέπει την αποθήκευση περισσότερων δεδομένων και την μείωση των conflict misses. Στα 16KB ολόκληρο το Data Set κατάφερε τελικά να χωρέσει στην κρυφή μνήμη πληρώνοντας πλέον μόνο τα compulsory misses, οπότε περαιτέρω αύξηση της L1 Data Cache επιφέρει μικρή βελτίωση.

Εξαντλητική Αναζήτηση.

Με την μέθοδο της εξαντλητικής αναζήτησης, τρέχουμε τις προσομοιώσεις για όλους τους πιθανούς συνδυασμούς των μνημών του συστήματος και του unrolling factor που προκύπτουν από τα σύνολα που δίνονται (συνολικά 6 x 6 x 4 x 5 = 720):

Parameters	Values
L1D Cache Size	2kB, 4kB, 8kB, 16kB, 32kB, 64kB
L1I Cache Size	2kB, 4kB, 8kB, 16kB, 32kB, 64kB
L2 Cache Size	128kB, 256kB, 512kB, 1024kB
Unrolling Factor	2, 4, 8, 16, 32

Πίνακας 4: Συνδυασμοί παραμέτρων

exhaustive.py

```
import sys
import subprocess
import csv

# search space values
L1D = [2, 4, 8, 16, 32, 64]
L1I = [2, 4, 8, 16, 32, 64]
L2 = [128, 256, 512, 1024]
unrolling = [2, 4, 8, 16, 32]
```

[4] https://github.com/gem5/gem5/blob/stable/configs/learning_gem5/part1/caches.py

```

outfile = "exhaustive.csv"

grep_cmd = f"cat m5out/stats.txt | grep system.cpu.numCycles | sed 's/[^0-9]*//g'"


with open(outfile, "w", newline="") as csvfile:
    csv_writer = csv.writer(csvfile)
    csv_writer.writerow(["L1D", "L1I", "L2", "Unrolling", "Cycles"])

    for i in L1D:
        for j in L1I:
            for k in L2:
                for u in unrolling:
                        cmd = f"build/X86/gem5.opt configs/learning_gem5/part1/two_level.py /gem5/
tables_UF/tables_uf{u}.exe --l1i_size={j}kB --l1d_size={i}kB --l2_size={k}kB"

                    print("Running: ", cmd)
                    try:
                        subprocess.run(cmd, shell=True, check=True)
                        result = subprocess.run(
                            grep_cmd,
                            shell=True,
                            check=True,
                            capture_output=True,
                            text=True,
                        )
                        cc = result.stdout.strip()
                        print(cc)
                        if cc is not None:
                            csv_writer.writerow(
                                [f"{i}", f"{j}", f"{k}", f"{u}", f"{cc}"]
                            )
                        else:
                            print(f"No cycles found for command: {cmd}")

                    except subprocess.CalledProcessError as e:
                        print(f"Error executing command: {cmd}\n{e}")
```

Για την εξαγωγή των pareto βέλτιστων σημείων του χώρου εξερεύνησης όσον αφορά στο συνολικό μέγεθος μνήμης (L1D + L1I + L2) και τον αριθμό κύκλων ταυτόχρονα εφαρμόζουμε την built-in μέθοδο paretoset με συνάρτηση ελαχιστοποίησης και στα 2 μεγέθη και καταγράφουμε τα αποτελέσματα:

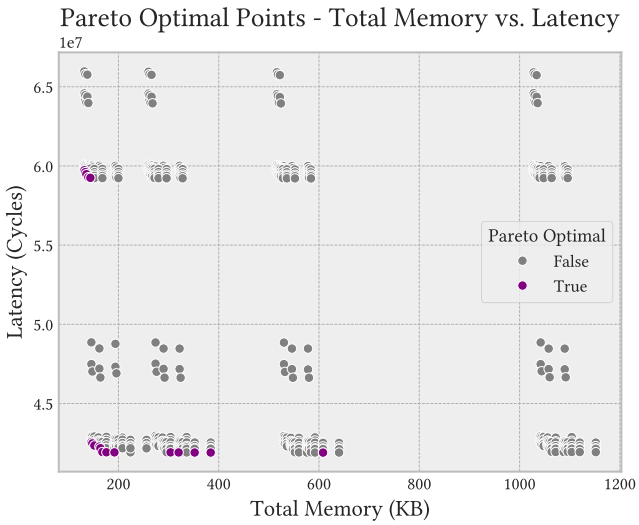
find_pareto.py

```
"""
@Authors: Maria Lazou, Stavros Avramidis
@Date: 11/11/2024
"""

from paretoset import paretoset
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("exhaustive.csv")
df["totalMemoryKB"] = df["L1D"] + df["L1I"] + df["L2"]
result_df = df[["totalMemoryKB", "Cycles"]].rename(columns={"Cycles": "Latency"})
print(result_df.head())

mask = paretoset(result_df, sense=["min", "min"])
# apply it on initial df
optimal = df[mask]
print(optimal)
optimal.to_csv("optimal_results.csv", index=False)

df["Pareto_optimal"] = mask

# Setup seaborn
sns.set_theme(style="whitegrid")
plt.style.use("bmh")
plt.rcParams["font.family"] = "Libertinus Serif"
plt.rcParams["font.size"] = 12
bmh_colors = plt.rcParams["axes.prop_cycle"].by_key()["color"]

```

```python
# Trick seaborn to bring pareto optimal points to the front
g = sns.scatterplot(
    data=df[df["Pareto_optimal"] == False],
    x="totalMemoryKB",
    y="Cycles",
    hue="Pareto_optimal",
    palette={True: bmh_colors[1], False: "darkgray"},
)
sns.scatterplot(
    data=df[df["Pareto_optimal"] == True],
    x="totalMemoryKB",
    y="Cycles",
    hue="Pareto_optimal",
    palette={True: bmh_colors[1], False: "darkgray"},
)
g.set_title("Pareto Optimal Points - Total Memory vs. Latency")
g.set_xlabel("Total Memory (KB)")
g.set_ylabel("Latency (Cycles)")
g.legend(title="Pareto Optimal")
plt.savefig("pareto_optimal.svg")
```

L1D	L1I	L2	Unrolling Factor	Total Memory (KB)	Latency(CC)
2	2	128	8	132	59725456
2	4	128	16	134	59608257
2	8	128	32	138	59447087
4	4	128	16	136	59477547
4	8	128	32	140	59287222
8	8	128	32	144	59249179
16	2	128	8	146	42547387
16	4	128	16	148	42480342
16	8	128	32	152	42348777
32	2	128	8	162	42278590
32	4	128	16	164	42176201
32	8	128	32	168	41936302
32	16	128	32	176	41922370
32	16	256	32	304	41915840
32	32	128	32	192	41915945
32	32	256	32	320	41910851
32	64	512	32	608	41901786
64	32	256	32	352	41906326
64	64	256	32	384	41904339

Πίνακας 5: Συγκεντρωτικός πίνακας των pareto-optimal σημείων



Σχήμα 8: Σύγκριση pareto-optimal σημείων

5.

Η ευριστική εξερεύνηση του ίδιου χώρου αναζήτησης ολοκληρώθηκε σε εκθετικά μικρότερο χρόνο και για τον ακριβή αριθμό των evaluations που έγιναν προσθέσαμε μια εκτύπωση της μεταβλητής *result.algorithm.evaluator.n_eval* στο genOptimizer.py η οπόια επέστρεψε αποτέλεσμα **30**.

Η συνάρτηση επέστρεψε πολύ λιγότερα pareto-optimal που είναι όμως υποσύνολο της εξαντλητικής αναζήτησης όπως φαίνεται στην συνέχεια:

L1I	L1D	L2	Unrolling Factor	Total Memory(KB)	Latency(CC)
4	4	128	16	136	59477547
16	32	128	32	176	41922370
4	32	128	8	164	42227791
2	4	128	4	134	59780093

Πίνακας 6: Συγκεντρωτικός πίνακας των ευριστικών σημείων

Παρατηρούμε ότι δεν βρήκε το βέλτιστο Latency που είναι 41901786CC ούτε την ελάχιστη μνήμη 132kB. Άλλοι συνδυασμοί που δεν εντοπίστηκαν διαφέρουν κυρίως στα unrolling factors που είναι ανεξάρτητο της αρχιτεκτονικής και την συνολικής μνήμης που καλείται ο αλγόριθμος να ελαχιστοποιήσει.