ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός Ενσωματωμένων Συστήματων

Άσκηση 1η

Αναστάσιος Στέφανος Αναγνώστου
03119051

Σαββίνα Νάστου
03119146

28 Νοεμβρίου 2023

Περιεχόμενα

Μέρος I
Ζητούμενο 1

Ερώτημα 1

Τα χαρακτηριστικά της πλακέτας είναι τα εξής:

1. RAM: 640 KB
2. Flash μνήμες: 4 MB Code, 64 KB Data, 1 Cache
3. Clock Frequency: 240 MHz

Στις flash μνήμες αποθηκεύονται δεδομένα μόνο για ανάγνωση, δηλαδή όσα δηλώνονται με const στο πρόγραμμα.

Ερώτημα 2

Η μέτρηση του χρόνου έγινε με την υποδομή που φαίνεται στην main στον κώδικα 1.

Ο αρχικός κώδικας χωρίς καμμία βελτιστοποίηση εκτελείται σε χρόνους:

1. min time: 1.64756 ms
2. max time: 1.64755 ms
3. mean time: 1.647555 ms

Ερώτημα 3

Εφαρμόστηκαν διαδοχικά διάφορες βελτιστοποίησεις και καταγράφηκαν οι ενδιάμεσες μετρήσεις.

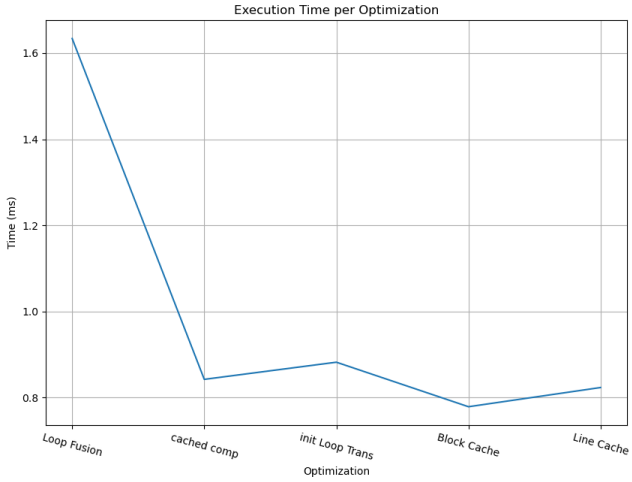
Αρχικά, εφαρμόσαμε loop fusion στους βρόχους `for(i=-S? i<S+1?  i+=S)` και παρατηρήθηκε μείωση του χρόνου εκτέλεσης, αλλά μικρή, τάξεως 1%.

Στην συνέχεια, παρατηρήθηκε ότι οι υπολογισμοί `B * x + vectors_x[x][y]` και `B * y + vectors_y[x][y]` βρίσκονταν σε πιο εσωτερικό βρόχο από ότι ήταν απαραίτητο και μπορούσαν να μετακινηθούν πιο έξω, μειώνοντας τους περιττούς υπολογισμούς. Έτσι, η επιτάχυνση ήταν μεγάλη, αφού παρατηρήθηκε υποδιπλασιασμός του χρόνου εκτέλεσης. Έπειτα, παρατηρήθηκε ότι οι πίνακες `vectors_x` και `vectors_y` αρχικοποιούνταν σε ξεχωριστό `for loop` από τον υπόλοιπο κώδικα, χωρίς να είναι ανάγκη. Επομένως, συγχωνεύθηκαν οι βρόχοι ((λοοπ φυσιον». Ο χρόνος εκτέλεσης παρουσίασε βελτίωση τάξεως 10%.

Τέλος, χρησιμοποιήθηκαν διάφορα προσωρινά buffers για να περιοριστεί η πρόσβαση του προγράμματος στον μεγάλο πίνακα της εικόνας, `current`, βελτιώνοντας την τοπικότητα των δεδομένων. Δοκιμάστηκαν δύο είδη buffer, ένα το οποίο φύλαγε το τρέχον block επεξεργασίας της εικόνας και ήταν μεγέθους $B \times B$ και ένα άλλο το οποίο φύλαγε ολόκληρη την τρέχουσα γραμμή επεξεργασίας της εικόνας και είχε μέγεθος $B \times M$. Το μεν ελάττωσε σημαντικά τον χρόνο εκτέλεσης ενώ το δε όχι ιδιαίτερα.

Συμπερασματικά, ο χρόνος ελαττώθηκε κατά πολύ.

Στο σχήμα 1 φαίνονται οι μετρήσεις ανά απόπειρα βελτίωσης του κώδικα.

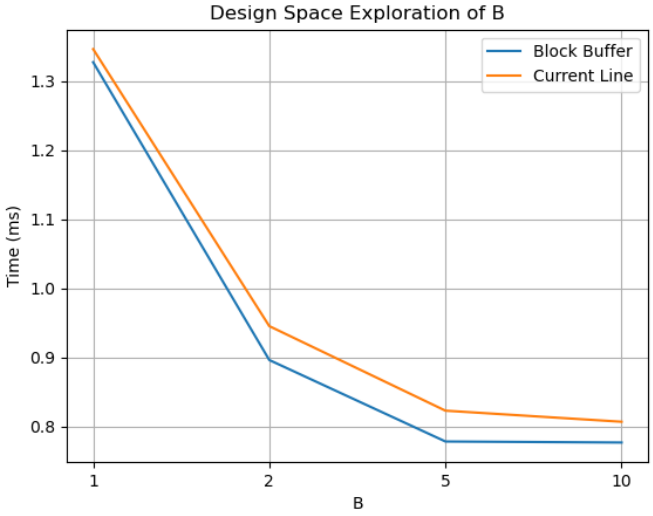


Σχήμα 1: Παρουσίαση Μετρήσεων

Ερώτημα 4

Το Design Space Exploration έγινε 'χειροκίνητα', λόγω του μικρού χώρου αναζήτησης. Συγκεκριμένα, αφού οι τιμές των παραμέτρων ήταν $N = M = 10$ και αναζητούνται τιμές του B ώστε να είναι διαιρέτης και των δύο, δοκιμάζονται μόνο οι τιμές $B = 1$, $B = 2$, $B = 5$ και $B = 10$.

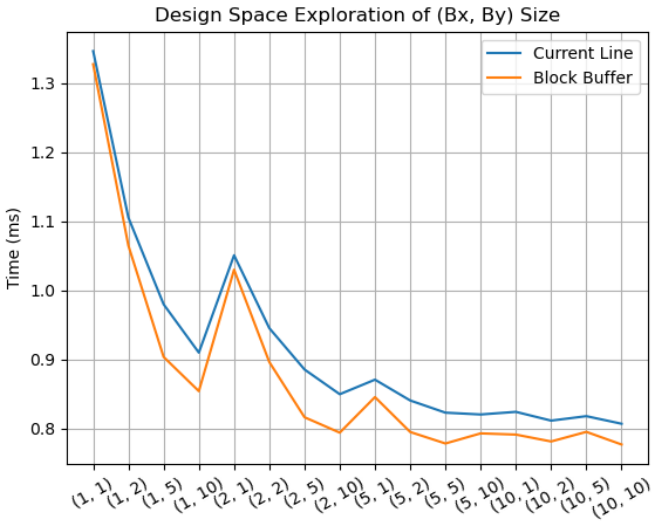
Σε κάθε περίπτωση buffer, ο καλύτερος χρόνος λήφθηκε για $B = 5$ και $B = 10$.



Σχήμα 2: Χρόνοι εκτέλεσης ανά B

Ερώτημα 5

Στο σχήμα 3 φαίνονται οι μετρήσεις για τις διάφορες τιμές των BX BΥ.



Σχήμα 3: Χρόνοι εκτέλεσης ανά (BX, BΥ)

Γενικά, η τάση είναι να επιταχύνεται η εκτέλεση όσο οι τιμές των B μεγαλώνουν. Αλλά, παρατηρείται ότι ο πιο σημαντικός παράγοντας είναι να είναι μεγάλη η τιμή του BΥ. Αυτό συμβαίνει επειδή BΥ είναι το μήκος της γραμμής ακεραίων η οποία θα αποθηκευτεί σε μία γραμμή της cache. Άρα, δεν οφελείται κανείς έχοντας πολλές γραμμές μικρού μήκους, αλλά οφελείται έχοντας οσεσδήποτε γραμμές μεγάλους μήκους, αρκεί να χωράνε στην γραμμή της cache. Αυτό θα μπορούσε να είναι ένα καλό κριτήριο.

Μέρος II
Παράρτημα

Listing 1: final.c

```
#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 BX 5       /*Block size*/
#define BY 2       /*Block size*/
#define p 7        /*Search space. Restricted in a [-p,p] region around the
                     original location of the block.*/

void phods_motion_estimation(const int current[N][M], const int previous[N][M], int
    vectors_x[N/BX][M/BY],int vectors_y[N/BX][M/BY])
{
    int x, y, i, j, k, l, p1, p2, q2, distx, disty, S, min1, min2, bestx, besty;
    int tempx, tempy;
    //int block[N][BY];
    int current_line[BX][M];
    int min_init = 255*BX*BY;
    int cached_x = cached_y = 0;
    distx = 0;
    disty = 0;
    /*For all blocks in the current frame*/
    for(x=0; x<N/BX; x++)
    {
        for(k = 0; k < BX; ++k)
            for(l = 0; l < M; ++l)
                current_line[k][l] = current[BX*x+k][l];
        cached_x = BX*x;
        for(y=0; y<M/BY; y++)
        {
            /*Initialize the vector motion matrices*/
            vectors_x[x][y] = 0;
            vectors_y[x][y] = 0;
            //for (k = 0; k < BX; ++k)
            //    for (l = 0; l < BY; ++l)
            //        block[k][l] = current[BX*x+k][BY*y+l];
            S = 4;
            cached_y = BY*y;
            while(S > 0)
            {
                min1 = min_init;
                min2 = min_init;
                /*For all candidate blocks in X dimension*/
                for(i=-S; i<S+1; i+=S)
                {
                    distx = 0;
                    disty = 0;
                    tempx = cached_x + vectors_x[x][y];
                    tempy = cached_y + vectors_y[x][y];
                    /*For all pixels in the block*/
                    for(k=0; k<BX; k++)
                    {
                        for(l=0; l<BY; l++)
                        {
                            //p1 = block[k][l];
                            p1 = current_line[k][cached_y+l];
```

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

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 */
    const int NUM_MEASUREMENTS=10, CLOCK_FREQ = 240000000;
    double exec_time[NUM_MEASUREMENTS];
    double mean_time = 0.0, total_time = 0.0;
    double max_time = 0.0, min_time = (double) CLOCK_FREQ;
    int motion_vectors_x[N/BX][M/BY], motion_vectors_y[N/BX][M/BY], i, j;

    while(1){
        /* Add timer code here */
        for (i = 0; i < NUM_MEASUREMENTS; ++i)
        {
            DWT->CYCCNT = 0;
            phods_motion_estimation(current,previous,motion_vectors_x,motion_vectors_y);
            exec_time[i] = ((double) (DWT->CYCCNT)) / ((double) CLOCK_FREQ) ;
            total_time += exec_time[i];
            min_time = exec_time[i] < min_time ? exec_time[i] : min_time;
            max_time = exec_time[i] > max_time ? exec_time[i] : max_time;
        }
        mean_time = ((double) total_time) / ((double) NUM_MEASUREMENTS);
    }
    while(1){}
}
```

Μέρος III
Ζητούμενο 2

Ερώτημα 1

```
system.cpu.numCycles 857669639 # Number of cpu cycles simulated (Cycle)
```

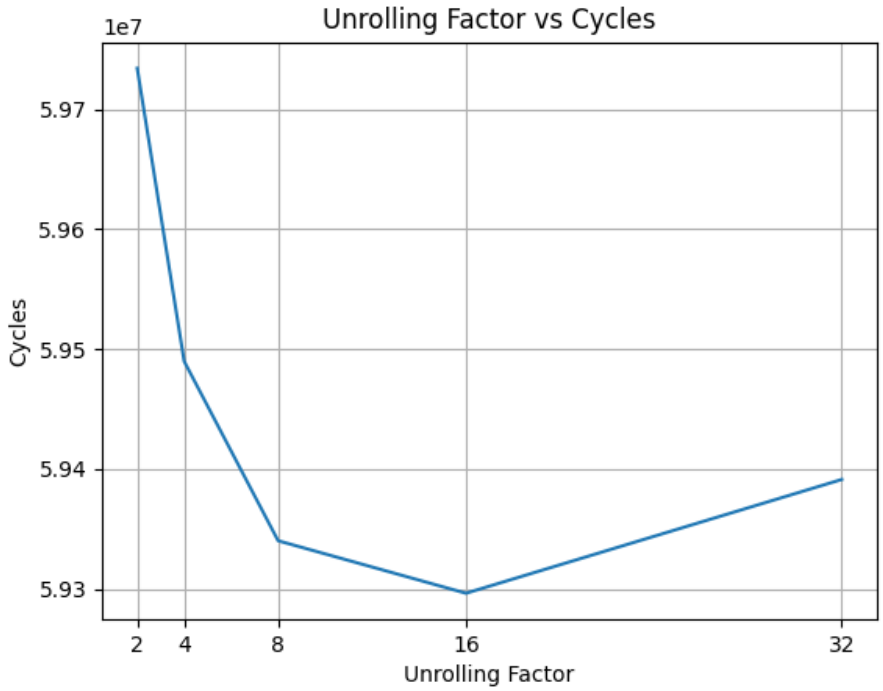
Ερώτημα 2

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

Στη δεύτερη αρχιτεκτονική παρατηρείται μείωση των κύκλων. Αυτό είναι κάτι αναμενόμενο αφού αυτή η αρχιτεκτονική διαθέτει caches στις οποίες μπορούν να αποθηκευτούν προσωρινά δεδομένα πιο κοντά στον επεξεργαστή.

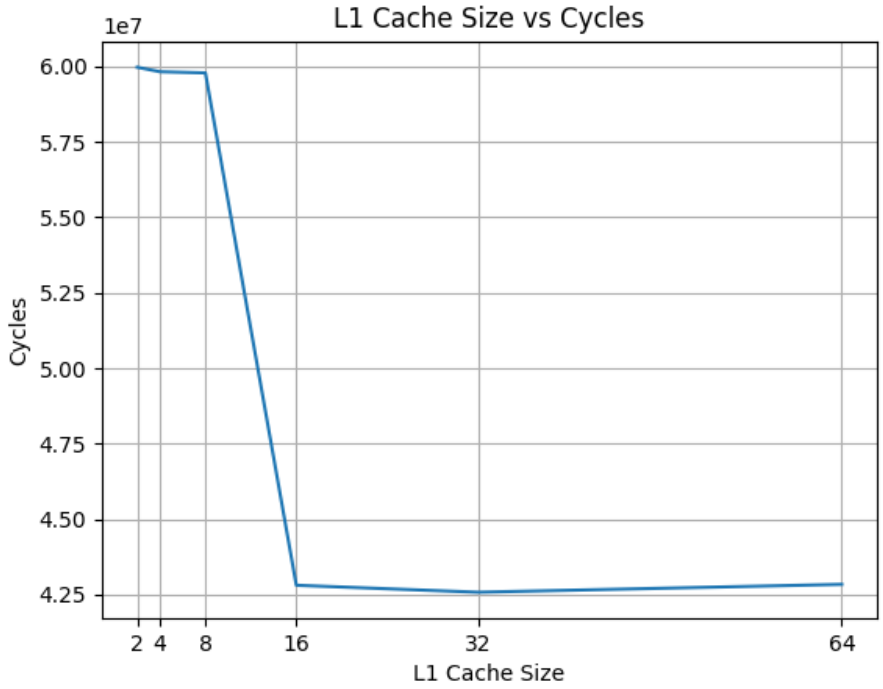
Ερώτημα 3

Οι μετρήσεις παρήχθησαν από τα bash scripts 2 και 3.



Σχήμα 4: Διάγραμμα μετρήσεων για το unrolling factor

Από το διάγραμμα παρατηρούμε ότι ο μικρότερος αριθμός κύκλων επιτυγχάνεται με unrolling factor ίσο με 16. Για unrolling factor ίσο με 32 ο αριθμός κύκλων αυξάνεται. Αυτό συμβαίνει γιατί χαλάει η τοπικότητα λόγω περιορισμένης χωρητικότητας της cache.



Σχήμα 5: Διάγραμμα μετρήσεων για το μέγεθος της L1 cache

Παρατηρούμε ότι με αύξηση του μεγέθους της cache μειώνονται οι κύκλοι. Ειδικότερα παρατηρούμε δραματική μείωση στη μετάβαση από 8 KB σε 16 KB ςαςηε σιζε. Από αυτό καταλαβαίνουμε ότι οι απαιτήσεις του προγράμματος σε μνήμη είναι περίπου 16 KB. Για την ακρίβεια λίγο παραπάνω από 16 KB αλλά λιγότερο από 32 KB αφού εκεί δεν παρατηρούμε απότομη μείωση των κύκλων.

Ερώτημα 4

Η εξαντλητική αναζήτηση έγινε χρήσει του bash script 4 και το pareto set / frontier βρέθηκε με κατάλληλο python script χρήσει της βιβλιοθήκης pareto. Τα σημεία του pareto set φαίνονται στο σχήμα 6.

Τα δε σημεία από την αναζήτηση του προγράμματος genOptimizer.py φαίνονται στο σχήμα 7.

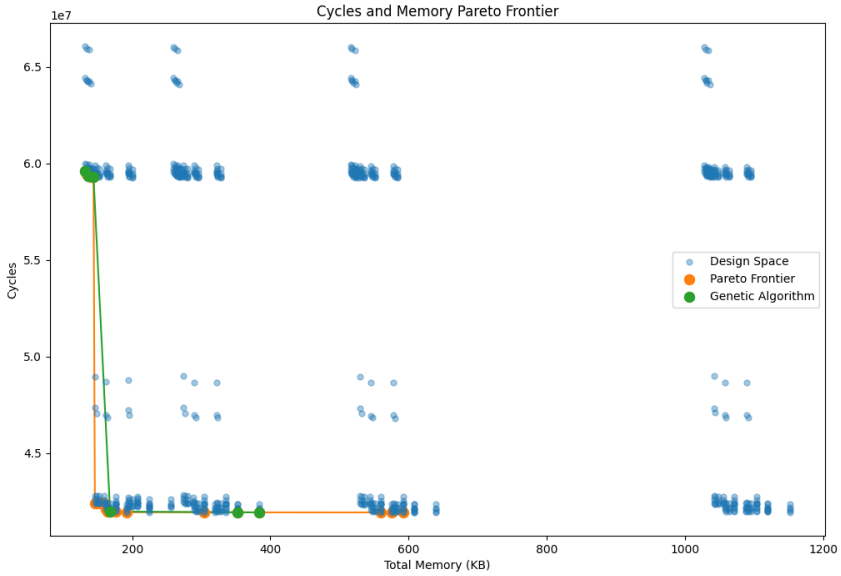
```
latencyCC,totalMemoryKB,l1i,l1d,l2,uf
59581198,132,2,2,128,8
59433698,134,2,4,128,8
42416612,146,2,16,128,8
42078234,162,2,32,128,8
59343511,136,4,4,128,16
59306017,140,4,8,128,16
42409832,148,4,16,128,16
41969420,164,4,32,128,16
59296471,144,8,8,128,16
42403308,152,8,16,128,16
41969226,168,8,32,128,16
42387518,160,16,16,128,16
41945801,176,16,32,128,16
41933039,304,16,32,256,16
41923846,560,16,32,512,16
41922998,592,16,64,512,16
41936488,192,32,32,128,16
41923699,576,32,32,512,16
41928359,352,64,32,256,16
41925024,384,64,64,256,16
```

Σχήμα 6: Οι αρχιτεκτονικές και τα αντίστοιχα pareto σημεία

```
L1I_cache_size,L1D_cache_size,L2_cache_size,unrolling_factor,exec_time_cc,total_mem_kB
4kB,4kB,128kB,16,59343511,136
32kB,64kB,256kB,16,41929862,352
64kB,64kB,256kB,16,41925024,384
2kB,2kB,128kB,8,59581198,132
8kB,8kB,128kB,16,59296471,144
8kB,32kB,128kB,16,41969226,168
```

Σχήμα 7: Οι αρχιτεκτονικές και τα αντίστοιχα pareto σημεία από το genOptimizer.py

Ολοκλήρος ο χώρος αναζήτησης και τα pareto optimal σημεία φαίνεται στο σχήμα 8.



Σχήμα 8: Απεικόνιση του χώρου αναζήτησης με το pareto frontier επισημασμένο

Παρατηρήσεις

Από το Σχήμα 7 βλέπουμε ότι ο γενετικός αλγόριθμος κάνει 6 εαλυατιονς, αρκετά λιγότερα συγκριτικά με την εξαντλητική εξερεύνηση του σχήματος 6. Επίσης και ο χρόνος εκτέλεσης είναι σημαντικά μικρότερος. Αυτό συμβαίνει γιατί ο γενετικός αλγόριθμος δεν εκτελεί τις redundant περιπτώσεις, δηλαδή τα σημεία στο Pareto Front που δεν είναι Pareto optimal λύσεις.

Συγκρίνοντας τα δύο Fronts διαπιστώνουμε ότι το σωστό είναι αυτό που προκύπτει από την εξαντλητική αναζήτηση (πορτοκαλί) αφού είναι πιο αριστερά στο plane άρα τα σημεία του πορτοκαλί Front κάνουν dominate σημεία του πράσινου Front.

Μέρος IV
Παράρτημα

Listing 2: unrolling-factors.sh

```
#!/bin/bash

command="build/X86/gem5.opt configs/learning_gem5/part1/two_level.py /gem5/
    tables_UF/"
options="--l1i_size=8kB --l1d_size=8kB --l2_size=128kB"
dataFile="m5out/stats.txt"
pattern="system.cpu.numCycles"
outputFile="unrolling-factors.txt"

factors=(2 4 8 16 32)
for factor in ${factors[@]}
do
    executable="tables_uf"
    executable+="$factor"
    executable+=".exe"
    full_command="$command$executable $options"
    eval $full_command &&
    < "$dataFile" grep --line-buffered "$pattern" >> "$outputFile"
done
```

Listing 3: l1-cache.sh

```
#!/bin/bash

command="build/X86/gem5.opt configs/learning_gem5/part1/two_level.py /gem5/
    tables_UF/tables.exe --l1i_size=8kB --l1d_size="
options=" --l2_size=128kB"
dataFile="m5out/stats.txt"
pattern="system.cpu.numCycles"
outputFile="l1-cache.txt"

caches=(2 4 8 16 32 64)
for cache in ${caches[@]}
do
    full_options="$cache"
    full_options+="kB"
    full_options+="$options"
    full_command="$command$full_options"
    eval $full_command &&
    < "$dataFile" grep --line-buffered "$pattern" >> "$outputFile"
done
```

Listing 4: run-search.sh

```
#!/bin/bash

# this is the first part of the command that must be executed
command="build/X86/gem5.opt configs/learning_gem5/part1/two_level.py /gem5/
    tables_UF/"
# this is the file where the output of the command will be stored by gem5
dataFile="m5out/stats.txt"
# this is the pattern that we are looking for in the output file
pattern="system.cpu.numCycles"
# this is the file where our data will be stored
outputFile="combinations.csv"

echo "cycles,l1i,l1d,l2,uf" > "$outputFile"

l1i_cache=(2 4 8 16 32 64)
l1d_cache=(2 4 8 16 32 64)
l2_cache=(128 256 512 1024)
unrolling_factors=(2 4 8 16 32)
for l1i in ${l1i_cache[@]}
do
    # we construct the options to be passed to the command
    l1i_opt="--l1i_size="
    l1i_opt+="$l1i"
    l1i_opt+="kB"
    for l1d in ${l1d_cache[@]}
    do
        l1d_opt="--l1d_size="
        l1d_opt+="$l1d"
        l1d_opt+="kB"
        for l2 in ${l2_cache[@]}
        do
            l2_opt="--l2_size="
            l2_opt+="$l2"
            l2_opt+="kB"
            for uf in ${unrolling_factors[@]}
            do
                uf_opt="tables_uf"
                uf_opt+="$uf"
                uf_opt+=".exe"

                # we construct the full command to be executed
                full_command="$command$uf_opt $l1i_opt $l1d_opt $l2_opt"
                # we execute the command
                eval $full_command && # if it is successful
                # we extract the data we are interested in
                cycles="$(< "$dataFile" grep --line-buffered "$pattern")" &&
                # and we write it to the output file in csv format
                echo "$cycles,$l1i,$l1d,$l2,$uf" >> "$outputFile"
            done
        done
    done
done
# the csv is not in the correct format yet
# because cycles did not contain only the number of cycles
# but also two strings and whitespaces that we do not need
```

```
# to remove the system.cpu.numCycles from every line
sedPattern1="s/system.cpu.numCycles//g"
# to remove the "# Number of cpu cycles simulated (Cycle)" from every line
sedPattern2="s/# Number of cpu cycles simulated (Cycle)//g"
# to remove the spaces from every line
sedPattern3="s/\s//g"
# give the patterns to sed
sed -i "$sedPattern1; $sedPattern2; $sedPattern3" "$outputFile"
```

Listing 5: pareto-search.py

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

# create a panda dataframe
# with two columns
# one totalMemoryKB that is the sum of the memory of all caches
# two latencyCC that are the total cycles for the execution of the program

def read_gen_data(filename):
    with open(filename, 'r') as f:
        lines = f.readlines()
    # the data are in csv format
    # we need only the latencyCC and totalMemoryKB
    lines = [line.strip() for line in lines]
    lines = [line.split(",") for line in lines]
    cycles = np.array([float(line[4]) for line in lines[1:]])
    memory = np.array([float(line[5]) for line in lines[1:]])
    return cycles, memory

dictionary = {
        "totalMemoryKB": [],
        "latencyCC": []
        }
# read the file
with open("combinations.csv") as f:
    # for each line
    architecture=[]
    for line in f:
        if line.startswith('cycles'):
            continue
        numbers = [ int(el) for el in line.split(",") ]
        latency = numbers[0]
        memory = numbers[1] + numbers[2] + numbers[3]
        architecture.append(line.split(",")[1:])
        # append the line to the dataframe
        dictionary["totalMemoryKB"].append(memory)
        dictionary["latencyCC"].append(latency)

# find the pareto frontier
design_space = pd.DataFrame(dictionary)
mask = paretoset(design_space, sense=["min", "min"])
paretoset = design_space[mask]

gen_cycles, gen_memory = read_gen_data("genOptimizer_PO.csv")
gen_dic = {
        "totalMemoryKB": gen_memory,
        "latencyCC": gen_cycles
        }
gen_df = pd.DataFrame(gen_dic)

# plot the pareto frontier
plt.figure(figsize=(12, 8), dpi=100)  # Adjust as needed
plt.title("Cycles and Memory Pareto Frontier")
```

```
plt.scatter(design_space["totalMemoryKB"],
            design_space["latencyCC"],
            zorder=20,
            label="Design Space",
            s=25,
            alpha=0.4)
plt.scatter(paretoset["totalMemoryKB"],
            paretoset["latencyCC"],
            zorder=10,
            label="Pareto Frontier",
            s=75,
            alpha=1)
plt.scatter(gen_df["totalMemoryKB"],
            gen_df["latencyCC"],
            zorder=30,
            label="Genetic Algorithm",
            s=75,
            alpha=1)
plt.xlabel("Total Memory (KB)")
plt.ylabel("Cycles")
# connect pareto points with a line
pareto_points = paretoset[["totalMemoryKB", "latencyCC"]].values
pareto_points = pareto_points[np.argsort(pareto_points[:, 0])]
plt.plot(pareto_points[:, 0], pareto_points[:, 1], c="C1")
# connect genetic algorithm points with a line
gen_points = gen_df[["totalMemoryKB", "latencyCC"]].values
gen_points = gen_points[np.argsort(gen_points[:, 0])]
plt.plot(gen_points[:, 0], gen_points[:, 1], c="C2")

plt.legend()
plt.savefig("./plots/pareto.png")
plt.show()


archs = np.array(architecture)
archs = archs[mask] # list of lists of 4 elements
archs = [",".join(el) for el in archs]

# write pareto points and architectures to csv
with open("pareto.csv", "w") as f:
    f.write('latencyCC,totalMemoryKB,l1i,l1d,l2,uf\n')
    for i in range(len(paretoset)):
        f.write(str(paretoset.iloc[i]['latencyCC']) + "," + str(paretoset.iloc[i][
            'totalMemoryKB']) + "," + str(archs[i]))
```