ΕΘΝΙΚΟ ΜΕΤΣΟΒΕΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
Τομέας Τεχνολογίας Υπολογιστών και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

Σχεδιασμός Ενσωματωμένων Συστημάτων
9ο Εξάμηνο ΗΜΜΥ

1η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΩΝ ΓΙΑ ΧΑΜΗΛΗ ΚΑΤΑΝΑΛΩΣΗ ΕΝΕΡΓΕΙΑΣ ΚΑΙ ΥΨΗΛΗ ΑΠΟΔΟΣΗ

1.1 Περιγραφή του υπό εξέταση αλγορίθμου

Σε αυτή την παράγραφο θα αναλυθεί ο αλγόριθμος Parallel Hierarchical One Dimensional Search (PHODS), ένας αλγόριθμος που ανήκει στην περιοχή των πολυμέσων. Ο αλγόριθμος PHODS είναι ένας αλγόριθμος εκτίμησης κίνησης (Motion Estimation), ο οποίος έχει στόχο να ανιχνεύσει τη κίνηση των αντικειμένων μεταξύ δύο διαδοχικών εικόνων (frame) του βίντεο. Οι αλγόριθμοι ανίχνευσης της κίνησης είναι καθοριστικοί για την συμπίεση βίντεο και αποτελούν τον πυρήνα κάθε εφαρμογής που περιέχει βίντεο. Ο PHODS έχει σαν είσοδο δύο διαδοχικές εικόνες από μια ακολουθία εικόνων βίντεο (διαστάσεων M×N), χωρίζει τις εικόνες σε block (διαστάσεων B×B pixel) σε κάθε μια από τις εικόνες αυτές και προσπαθεί να βρει την μετατόπιση του κάθε block από τη μια εικόνα (frame) στην επόμενη (frame). Για την εύρεση της μετατόπισης κάθε μπλόκ από το ένα frame στο επόμενο frame εκτελείται σύγκριση του μπλόκ από το πρώτο frame με όλα τα μπλόκ που βρίσκονται στην γύρω περιοχή από το επόμενο frame. Βάσει ενός συγκεκριμένου κριτηρίου επιλέγεται το μπλοκ εκείνο που εμφανίζει την μικρότερη τιμή στο κριτήριο.

Τον αρχικό αλγόριθμο PHODS σε γλώσσα C μπορείτε να τον βρείτε στα συνημμένα αρχεία της εργαστηριακής άσκησης, στο αρχείο `phods.c`.

1.2 Ζητούμενο 1ο – Loop Optimizations & Design Space Exploration

Στο **προσωπικό σας υπολογιστή** να πραγματοποιήσετε τα παρακάτω ερωτήματα:

1. Μέσω του datasheet για το Renesas S7G2 και του e2studio, βρείτε τα χαρακτηριστικά του board. Πιο συγκεκριμένα καταγράψτε:
   - Το μέγεθος RAM, Flash μνημών (από το datasheet)
   - Τις κρυφές μνήμες που υπάρχουν στο board και την λειτουργία τους

- Την συχνότητα ρολογιού στην οποία θα προγραμματιστεί το board (απο το e2studio ή μέσω του debugger βάζοντας ένα watch expression στο SystemCoreClock)

2. Μετασχηματίστε τον δοθέντα κώδικα ώστε να μετράται ο χρόνος που χρειάζεται η συνάρτηση phods_motion_estimation για να εκτελεστεί. Η μέτρηση του χρόνου να γίνει διαβάζοντας κύκλους από τον register DWT->CYCCNT (ο κώδικας για να τον αρχικοποιήσετε σωστά δίνεται στο αρχείο με τον κώδικα της άσκησης) και διαιρώντας με τη συχνότητα. Να εκτελέσετε τον κώδικα 10 φορές και να μετρήσετε το μέσο όρο, μέγιστο και ελάχιστο χρόνο εκτέλεσης. Παρουσιάστε τα αποτελέσματα της μέτρησης.

3. Με δεδομένη την ύπαρξη της υποδομής για την μέτρηση του χρόνου που χρειάζεται το κρίσιμο κομμάτι της εφαρμογής για να εκτελεστεί, εφαρμόστε μετασχηματισμούς στον κώδικα ώστε να μειωθεί ό χρόνος εκτέλεσης. Για κάθε μετασχηματισμό, να καταγραφεί η αντίστοιχη αλλαγή και να δικαιολογήσετε το λόγο για τον οποίο τον πραγματοποιήσατε. Αφού έχετε ολοκληρώσει τους μετασχηματισμούς στον κώδικα, να ξαναεκτελέστε τον κώδικα 10 φορές και να μετρήσετε μέσο όρο, μέγιστο και ελάχιστο χρόνο εκτέλεσης, αντίστοιχα με το ερώτημα 2. Να συγκρίνετε τους χρόνους εκτέλεσης με αυτούς του ερωτήματος 2.

4. Στον κώδικα που προέκυψε από το ερώτημα 3, εφαρμόστε Design Space Exploration αναφορικά με την εύρεση του βέλτιστου μεγέθους μπλοκ B (μεταβλητή στον κώδικα) για την αρχιτεκτονική του υπολογιστή σας. Η αναζήτηση να πραγματοποιηθεί με εξαντλητική αναζήτηση και μόνο για τιμές του B που είναι κοινοί διαιρέτες του M και του N. Για κάθε τιμή του B, το πρόγραμμα να εκτελεστεί 10 φορές και ως μετρική απόδοσης να οριστεί ο μέσος όρος του χρόνου εκτέλεσης σε αυτές τις 10 εκτελέσεις. Καταγράψτε την τιμή του B για την οποία πετυχαίνετε την καλύτερη απόδοση. Υπάρχει κάποια σχέση μεταξύ αυτής της τιμής και κάποιου χαρακτηριστικού του μηχανήματός σας από αυτά που καταγράψατε στο ερώτημα 1;

5. Στον κώδικα που προέκυψε από το ερώτημα 3, εφαρμόστε Design Space Exploration αναφορικά με την εύρεση του βέλτιστου μεγέθους μπλοκ θεωρώντας ορθογώνιο μπλοκ διαστάσεων B_x και B_y για την αρχιτεκτονική του υπολογιστή σας. Το μέγεθος του B_x να είναι διαιρέτης του N και το μέγεθος B_y να είναι διαιρέτης του M. Εφαρμόστε εξαντλητική αναζήτηση προκειμένου να βρείτε το βέλτιστο ζεύγος B_x, B_y το οποίο ελαχιστοποιεί τον χρόνο εκτέλεσης του προγράμματος. Για κάθε ζεύγος B_x και B_y, το πρόγραμμα να εκτελεστεί 10 φορές και ως μετρική απόδοσης να οριστεί ο μέσος όρος του χρόνου εκτέλεσης σε αυτές τις 10 εκτελέσεις. Καταγράψτε το ζεύγος B_x, B_y για το οποίο πετυχαίνετε την καλύτερη απόδοση. Προτείνετε κάποιο ευριστικό τρόπο περιορισμού της αναζήτησης αν θεωρείτε ότι υπάρχει.

6. Να απεικονίσετε τα αποτελέσματα των ερωτημάτων 2-5 σε ένα διάγραμμα

boxplot[1,2]. Να συγκρίνετε και να αναλύσετε τα πειραματικά αποτελέσματα που λάβατε.

1.3 Ζητούμενο 2ο : Βελτιστοποίηση εφαρμογής σε διαφορετικές αρχιτεκτονικές x86

Σε αυτό το ζητούμενο καλείστε να βελτιστοποιήσετε την εφαρμογή **tables.c** της οποίας η λειτουργία αφορά απλές προσπελάσεις και πράξεις με πίνακες. Η λειτουργία της εφαρμογής **tables.c** θα εκτελεστεί σε δύο διαφορετικές αρχιτεκτονικές **x86** με τη βοήθεια του προσομοιωτή **Gem5**. Η πρώτη αρχιτεκτονική αφορά ένα σύστημα το οποίο αποτελείται από τον επεξεργαστή και την κύρια μνήμη ενώ η δεύτερη έχει επαυξηθεί με **L1 Instruction**, **Data** και **L2 caches**. Αναλυτικές οδηγίες για την εγκατάσταση του προσομοιωτή Gem5 καθώς και ότι άλλο απαιτείται για την εκτέλεση των ερωτημάτων της άσκησης μπορούν να βρεθούν στο κάτωθι GitHub repository.

Στον προσωπικό σας υπολογιστή να υλοποιήσετε και να απαντήσετε τα επόμενα ερωτήματα.

1. Αφού εγκαταστήσετε επιτυχώς τον προσομοιωτή και εκτελέσετε όσα αναγράφονται στο GitHub repository εκτελέστε την εντολή:

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

   η οποία προσομοιώνει την εφαρμογή tables.exe στην πρώτη αρχιτεκτονική. Προκειμένου να δείτε πόσοι κύκλοι απαιτούνται για την εκτέλεσή μπορείτε να δείτε την τιμή της μεταβλητής **system.cpu.numCycles** στο αρχείο **stats.txt** που βρίσκεται στον φάκελο **m5out**.

2. Στο ερώτημα αυτό θα προσομοιώσουμε τη λειτουργία της εφαρμογής 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=8kB --
l2_size=128kB
```

   Πόσους κύκλους απαιτεί η εκτέλεση της εφαρμογής τώρα; Συγκρίνετε τα αποτελέσματα με το πρώτο ερώτημα και εξηγείστε.

3. Για την αρχιτεκτονική που ορίστηκε παραπάνω προσομοιώστε την εφαρμογή με διαφορετικά unrolling factors (2, 4, 8, 16 και 32) μέσω της εντολής:

[1] https://en.wikipedia.org/wiki/Box_plot
[2] https://seaborn.pydata.org/generated/seaborn.boxplot.html

```
$ 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
```

Καταγράψτε τους κύκλους που απαιτούνται και παρουσιάστε τους σε ένα **line plot** όπου στον οριζόντιο άξονα θα έχετε τα διαφορετικά unrolling factors και στον κατακόρυφο το πλήθος των κύκλων. Εξηγείστε τα αποτελέσματα

Επίσης εξερευνήστε 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
```

Καταγράψτε τους κύκλους που απαιτούνται και παρουσιάστε τους και πάλι σε ένα line plot όπου στον οριζόντιο άξονα θα έχετε τα διαφορετικά μεγέθη cache και στον κατακόρυφο το πλήθος των κύκλων. Εξηγείστε τα αποτελέσματα.

4. Στο ερώτημα αυτό καλείστε να βελτιστοποιήσετε 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 χρησιμοποιήσουμε δηλ. το άθροισμα της L1D, της L1I και της L2 cache σε KB.

   Στη συνέχεια καταγράφουμε το χώρο που θα εξερευνήσουμε.

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

   Στο ερώτημα αυτό θα εκτελέσετε εξαντλητική αναζήτηση και για κάθε συνδυσμό να καταγράψετε το πλήθος των κύκλων που απαιτούνται για την εκτέλεση της εφαρμογής σε ένα **CSV** αρχείο. Οι εντολές που θα χρησιμοποιήσετε θα είναι της μορφής:

```
$ 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
```

   Αφού καταγράψετε τα αποτελέσματα και τον συνολικό χρόνο που απαιτείται για την

εξαντλητική αναζήτηση, εντοπείστε το Pareto Frontier μέσω της βιβλιοθήκης **paretoset**. Για να διευκολυνθείτε δημιουργείστε ένα pandas dataframe με δύο στήλες. Η πρώτη θα ονομάζεται *totalMemoryKB* και θα περιλαμβάνει το άθροισμα της μνήμης σε KB για την εκάστωστε αρχιτεκτονική ενώ η δεύτερη στήλη θα ονομάζεται *latencyCC* και θα περιλαμβάνει τους κύκλους. Αφού εντοπίσετε το Pareto Frontier **μαζί** με τα στοιχεία της αρχιτεκτονικής που οδήγησαν στο εκάστωτε Pareto optimal σημείο καταγράψτε τα σε ένα **CSV** αρχείο.

5. Στο τελευταίο ερώτημα καλείστε να επαναλάβεται την παραπάνω εξερεύνηση με τη χρήση ενός γενετικού αλγορίθμου. Δεδομένου ότι το Python script είναι έτοιμο εσείς καλείστε απλά να εκτελέσετε την εφαρμογή.

```
$ python3 genOptimizer.py
```

Μετά την εκτέλεση της εφαρμογής θα λάβετε το Pareto front. Πόσα evaluations έκανε ο αλγόριθμος; Συγκρίνετε τον χρόνο εκτέλεσης αυτής της εξερεύνηση με την εξαντλητική αναζήτηση καθώς και τα Pareto Fronts.

Υποσημείωση: Για τα ερωτήματα 3 και 4 κατασκευάστε κατάλληλα scripts τα οποία θα πραγματοποιούν την εξερεύνηση αυτόματα.