ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός Ενσωματωμένων Συστημάτων

3η Σειρά Ασκήσεων

Χειμερινό Εξάμηνο, Ακαδ. Έτος: 2023-2024

ΟΝΟΜΑΤΕΠΩΝΥΜΟ: ΔΙΑΜΑΝΤΙΔΗΣ ΘΕΟΧΑΡΗΣ

ΟΝΟΜΑΤΕΠΩΝΥΜΟ: ΙΩΑΝΝΟΥ ΚΩΝΣΤΑΝΤΙΝΟΣ

ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ: 03119002

ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ: 03119840

ΕΞΑΜΗΝΟ: 9

a) Αρχικά θα εξετάσουμε θεωρητικά το estimation του vivado για τον κώδικα που μας δόθηκε για την συνάρτηση caldistanceHW() χωρίς κανένα optimization.

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.68	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
475204	475204	475205	475205	none

Detail

Instance

N/A

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- LOAD_DATA_HW_TMP	34816	34816	34	-	-	1024	no
+ LOAD_DATA_HW_TMP.1	32	32	1	-	-	32	no
- LOAD_MOVIE_TMP	64	64	2	-	-	32	no
- COMPUTE_DISTS	438272	438272	428	-	-	1024	no
+ COMPUTE_DISTS.1	416	416	13	-	-	32	no
- WRITE_DISTS	2048	2048	2	-	-	1024	no

Παρατηρούμε ότι τους περισσότερους κύκλους εκτέλεσης της συνάρτησης που θα τρέξει στο fpga τις χρειάζεται το loop που υπολογίζει την απόσταση Compute distance.Όπως είναι λογικό τα δύο loops που χρειάζονται τους περισσότερους χρόνους εκτέλεσης είναι τα διπλά for loops που υλοποιούν το load_data και το compute_dist.Βέβαια ο υπολογισμός της απόστασης είναι το «βαρύ» υπολογιστικό κομμάτι καθώς εκεί γίνεται ανάγνωση , εγγραφή πινάκων , πράξεις και μαθηματικοί υπολογισμοί όπως ο το γινόμενο diff*diff αλλά και εύρεση της ρίζας. .
Επίσης παρατηρούμε δεν εφαρμόζουμε σε κανένα loop pipeline και πώς το zybo έχει ακόμη αρκετούς διαθέσιμους πόρους για να αξιοποιήσουμε στην βελτίωση της εφαρμογής μας.

Πιο συγκεκριμένα βλέπουμε από το estimation :

Details

Performance estimates for 'calcDistancesHW in main.cpp:24 ...

HW accelerated (Estimated cycles)	3244037

Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	5	80	6,25
BRAM	33	60	55
LUT	1652	17600	9,39
FF	1060	35200	3,01

Ότι η συνάρτηση χρειάζεται υπερβολικά πολλού κύκλους και καταναλώνει ελάχιστους πόρους κυρίως BRAMs για την αποθήκευση των δεδομένων.

Β) Σε αυτό το σημείο θα τρέξουμε την αρχική υλοποίηση της συνάρτησης στην πλακέτα και θα συγκρίνουμε τους πραγματικούς κύκλους εκτέλεσης με το προηγούμενο estimation.Καθώς θα γίνει και σύγκριση του Software με το hardware ,δηλαδή του speedup λόγω της χρήσης του fpga.

Αφού φορτώσουμε το bitsream στην SD card και την συνδέουμε με το zybo βλέπουμε ότι μας στέλνει τα παρακάτω αποτελέσματα.

```
Recommendation system start to make inference
...
Recommendations for movie with id 0:
0. Jurassic Park (1993), with distance of 10.7121
1. Fish Called Wanda  A (1988), with distance of 11.0793
2. Back to the Future (1985), with distance of 11.6404
3. Star Wars: Episode VI - Return of the Jedi (1983), with distance of 11.8849
4. Lion King  The (1994), with distance of 12.0623
5. Raising Arizona (1987), with distance of 12.1552
6. Wizard of Oz  The (1939), with distance of 12.1861
7. Batman (1989), with distance of 12.2066
8. Princess Bride  The (1987), with distance of 12.3085
9. E.T. the Extra-Terrestrial (1982), with distance of 12.3996

Hardware cycles : 3263514
Software cycles : 1389757
Speedup         : 0.425847
Correct = 1024, Score = 1.000000
```

Παρατηρούμε σε αυτό το σημείο ότι η συνάρτηση χρειάζεται περισσότερους κύκλους στο hardware από ότι στο software για αυτό και έχουμε speedup 0.42.Αυτό οφείλεται στο γεγονός ότι δεν έχουμε εκμεταλλευτεί σωστά τους τις δυνατότητες και του πόρους που μας προσφέρει η HLS ,συνεπώς δεν αρκεί απλώς να ορίσουμε ότι μια συνάρτηση θα τρέχει στο hardware αλλά είναι αναγκαίο να προσθέσουμε τα κατάλληλα directives μέσω της HLS ώστε και να αναλογιστούμε πως θα τρέξει ο κώδικας σε επίπεδο πιο κοντά στο υλικό. Ακόμη παρατηρούμε ότι τελικά οι κύκλοι εκτέλεσης στην πραγματικότητα είναι ελάχιστα παραπάνω από ότι είχε υπολογίσει το estimation.

Γ)Σε αυτό το ερώτημα θα βελτιστοποιήσουμε την υλοποίηση που μας δόθηκε και έπειτα θα την δοκιμάσουμε στο board του zybo.

- Αρχικά θα κάνουμε **HLS UNROLL** σε κάθε for loop

  Οπότε κάνουμε loop unrolling σε όλα τα loops τα οποία τρέχουν εως USER_NUM στα nest loop δεν αξίζει ιδιαίτερα να κάνουμε loop unroll καθώς για unroll όλων των επαναλήψεων δεν αρκούν οι πόροι του fpga (όπως είδαμε από δοκιμές) και για unroll με factor κατάλληλο ώστε να μπορεί να υλοποιηθεί στο fpga δεν έχουμε σημαντική μείωση των κύκλων εκτέλεσης αναλογικά με τους επιπλέον πόρους που καταναλώνουμε οπότε αποφασίσαμε να κάνουμε να μην κάνουμε πλήρως loop unrolling καθώς καταναλώνει πολλούς πόρους χωρίς να έχει σημαντική βελτίωση στους

κύκλους εκτέλεσης οπότε δοκιμάσαμε διάφορα factors μέχρι να πετύχουμε το καλύτερο trade -off.

Χρησιμοποιούμε περισσότερα **FF** και **LUTS** αλλά οι κύκλοι εκτέλεσης της συνάρτησης μειώνονται .

Summary

Latency		Interval		
min	max	min	max	Type
185347	185347	185348	185348	none

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	33
FIFO	-	-	-	-
Instance	-	5	702	1331
Memory	66	-	0	0
Multiplexer	-	-	-	8744
Register	-	-	3448	-
Total	66	5	4150	10108
Available	120	80	35200	17600
Utilization (%)	55	6	11	57

```
28 COMPUTE_DISTS:
29     for (int i = 0; i < MOVIES_NUM; i++) {
30         float sum = 0.0, diff = 0.0;
31         for (int j = 0; j < USERS_NUM; j++){
32          #pragma HLS unroll
33             diff = data_hw_tmp[i][j] - movie_tmp[j];
34             sum += diff * diff;
35         }
36         dists_hw_tmp[i] = sqrt(sum);
37     }
```

- Στην συνέχεια στοχεύουμε με την εντολή **HLS PIPELINE** να επιταχύνουμε το βαρύ υπολογιστικά κομμάτι του κώδικα , όμως το παραπάνω δεν συμβαίνει καθώς βλέπουμε ότι οι πόροι που έχει το σύστημα δεν επαρκούν και να γίνει το pipeline ,το οποίο μπορεί να οφείλεται και ότι χρησιμοποιούμε τύπους δεδομένων float οι οποίοι 32 bits. Συνεπώς για να το διορθώσουμε αυτό μειώνουμε το unroll ώστε να καταφέρουμε να έχουμε pipeline με II=1 σε όλα τα loops ,το οποίο τελικά μειώνει σημαντικά τους κύκλους εκτέλεσης αν και αυξάνει κατά μεγάλο ποσοστό των LUTS που χρησιμοποιούνται

Βλέπουμε ότι δεν μπορούμε να επιτύχουμε pipeline με II=1 διότι έχουμε εξάρτηση δεδομένων στην μεταβλητή sum

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	34
FIFO	-	-	-	-
Instance	-	224	18971	36703
Memory	66	-	0	0
Multiplexer	-	-	-	7522
Register	-	-	6416	60
Total	66	224	25387	44319
Available	120	80	35200	17600
Utilization (%)	55	280	72	251

Detail

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.68	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
34966	34966	34967	34967	none

- Προσθέτουμε την εντολή **#pragma HLS loop_merge** στην αρχή της συνάρτησης δεν έχουμε βελτίωση στα αποτελέσματα μας ,καθώς δεν υπάρχει κάποιο loop που μπορεί να ενωθεί.

- **HLS array partition**

Ααρχικά δοκιμάσαμε τις παρακάτω εντολές ώστε να σπάσω τους δισδιάστατους πίνακες σε δύο ως προς την πρώτη διάσταση.

```
#pragma HLS ARRAY_PARTITION variable=data_hw_tmp block factor=2 dim=1
#pragma HLS ARRAY_PARTITION variable=dists_hw_tmp block factor =2 dim=1
```

Αυτό όμως είχε ως αποτέλεσμα να αυξηθούν πολύ τα LUTs που χρησιμοποιεί το fpga καθώς και οι κύκλοι εκτέλεσης αυξήθηκαν. Το παραπάνω μπορεί να οφείλεται σε ότι η επικάλυψη γραμμών σε διαίρεση πινάκων μπορεί να προκαλέσει καθυστερήσεις, ενώ η διάσπαση σε τμήματα με διάσταση 1 αυξάνει τις λογικές πύλες και τις εντολές φόρτωσης-εκφόρτωσης, αυξάνοντας τους κύκλους εκτέλεσης λόγω πρόσθετης πολυπλοκότητας και φόρτωσης δεδομένων.

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	10.35	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
185347	185347	185348	185348	none

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	1059
FIFO	-	-	-	-
Instance	-	33	3880	6987
Memory	66	-	0	0
Multiplexer	-	-	-	8696
Register	-	-	3912	-
Total	66	33	7792	16742
Available	120	80	35200	17600
Utilization (%)	55	41	22	95

Βέβαια καλύτερο είναι να κάνουμε array_partition σε κάθε περίπτωση στην διάσταση του πίνακα του οποίου έχουμε είδη εφαρμόσει loop unrolling ώστε να μπορεί να γράψει και να διαβάσει πιο γρήγορα στις επιμέρους τιμές καθώς θα έχει πρόσβαση σε ποιο μικρές block ram ,ακόμη και σε ένα μόνο στοιχείο αν εφαρμόσουμε complete διάσπαση.

```
#pragma HLS array_partition variable=data_hw_tmp block dim=2 factor=2
#pragma HLS array_partition variable=movie_tmp block dim=1 factor=2
#pragma HLS array_partition variable=dists_hw_tmp block dim=1 factor=2
```

Έχουμε μείωση των LUTS αύξηση των FFS και βελτίωση στο estimated clock .

Τελικός κώδικας φαίνεται στην συνέχεια:

```
#include <math.h>
#include "calcDist.h"

void calcDistancesHW(float* data_hw, float* dists_hw)
{
    float data_hw_tmp[MOVIES_NUM][USERS_NUM];
    float movie_tmp[USERS_NUM];
    float dists_hw_tmp[MOVIES_NUM];

#pragma HLS array_partition variable=data_hw_tmp block dim=2 factor=2
#pragma HLS array_partition variable=movie_tmp block dim=1 factor=2
#pragma HLS array_partition variable=dists_hw_tmp block dim=1 factor=2

LOAD_DATA_HW_TMP:
    for (int i = 0; i < MOVIES_NUM; i++) {
        #pragma HLS pipeline II=1
        for (int j = 0; j < USERS_NUM; j++) {
            #pragma HLS pipeline II=1
            data_hw_tmp[i][j] = data_hw[i * USERS_NUM + j];
        }
    }

LOAD_MOVIE_TMP:
    for (int i = 0; i < USERS_NUM; i++){
        #pragma HLS pipeline II=1
        movie_tmp[i] = data_hw_tmp[MOVIE_ID][i];
    }
```

```
COMPUTE_DISTS:
    for (int i = 0; i < MOVIES_NUM; i++) {
        #pragma HLS unroll factor=2
        #pragma HLS pipeline II=1
        float sum = 0.0, diff = 0.0;
        for (int j = 0; j < USERS_NUM; j++){
            #pragma HLS unroll factor=2
            #pragma HLS pipeline II=1
            diff = data_hw_tmp[i][j] - movie_tmp[j];
            sum += diff * diff;
        }
        dists_hw_tmp[i] = sqrt(sum);
    }

WRITE_DISTS:
    for (int i = 0; i < MOVIES_NUM; i++) {
        #pragma HLS pipeline II=1
        dists_hw[i] = dists_hw_tmp[i];
    }
}
```

Με τα παρακάτω αποτελέσματα:

Details

Performance estimates for 'calcDistancesHW in main.cpp:24 ...

HW accelerated (Estimated cycles)	429296

Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	28	80	35
BRAM	33	60	55
LUT	10620	17600	60,34
FF	8485	35200	24,11

Αρχικά παρατηρούμε ότι οι κύκλοι εκτέλεσης της συνάρτησης μειώθηκαν εξαιρετικά πολύ από ~3 εκατομμύρια σε μόλις ~400 χιλιάδες. Βέβαια αυτό εξηγεί και την χρήση αρκετών επιπλέον πόρων του fpga χωρίς όμως να αξιοποιούμε ούτε σχεδόν τους μισούς από όσους είναι διαθέσιμο στο zybo.Φυσικά μια τόσο σημαντική βελτίωση στην επίδοση του αλγόριθμού αξίζει την κατανάλωση επιπλέον πόρων.

Βλέπουμε ότι δεν μπορούμε να επιτύχουμε pipeline με II=1 διότι έχουμε εξάρτηση δεδομένων στην μεταβλητή sum.

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- LOAD_DATA_HW_TMP	32768	32768	33	32	1	1024	yes
- LOAD_MOVIE_TMP	32	32	2	1	1	32	yes
- COMPUTE_DISTS	8327	8327	152	16	1	512	yes
- WRITE_DISTS	1024	1024	2	1	1	1024	yes

Στην συνέχεια θα δοκιμάσουμε να τρέξουμε στο zybo την συνάρτηση μετά τις βελτιστοποιήσεις και να εξετάσουμε αν έχουμε σωστά αποτελέσματα αλλά και αν οι κύκλοι εκτέλεσης μειώθηκαν όσο περιμέναμε.

```
Recommendations for movie with id 0:
0. Jurassic Park (1993), with distance of 10.7121
1. Fish Called Wanda  A (1988), with distance of 11.0793
2. Back to the Future (1985), with distance of 11.6404
3. Star Wars: Episode VI - Return of the Jedi (1983), with distance of 11.8849
4. Lion King  The (1994), with distance of 12.0623
5. Raising Arizona (1987), with distance of 12.1552
6. Wizard of Oz  The (1939), with distance of 12.1861
7. Batman (1989), with distance of 12.2066
8. Princess Bride  The (1987), with distance of 12.3085
9. E.T. the Extra-Terrestrial (1982), with distance of 12.3996

Hardware cycles : 448769
Software cycles : 1361390
Speedup         : 3.03361
Correct = 1024, Score = 1.000000
```

Οι κύκλοι εκτέλεσης και πάλι είναι λίγο χειρότεροι από όσο είχαμε υπολογίσει στο estimation αλλά τώρα είναι σημαντικά καλύτεροι σε σχέση με τους χρόνου που επιτυγχάνει στο software και για αυτό βλέπουμε και speedup = 3.03361 μια εμφανώς τεράστια βελτίωση σε σχέση με τον κώδικα χωρίς τις βελτιστοποιήσεις με μόνο τίμημα κατανάλωση επιπλέον πόρων που ομως ήταν διαθέσιμοι και αναξιοποίητοι.

Στην συνέχεια προσπαθήσαμε να καταφέρουμε σχεδίαση dataflow αλλά ενώ δούλεψε το estimation και παράχθηκε το bitstream προφανώς λόγο κάποιας αμέλειας στη σχεδίαση το bitstream δεν παράχθηκε σωστά οπότε δεν κατάφερε να τρέξει στο zybo ,οπότε παρουσιάζουμε απλώς θεωρητικά τα αποτελέσματα

****

- Στην συνέχεια δοκιμάζουμε **HLS DATAFLOW**

Αλλά δεν βλέπουμε καμία βελτίωση στους κύκλους εκτέλεσης αυτό οφείλεται.

Η χρήση #pragma hls dataflow στο παραπάνω παράδειγμα δεν εγγυάται απαραίτητα μείωση του αριθμού των κύκλων εκτέλεσης. Η #pragma hls dataflow είναι ένας οδηγός που λέει στο σύστημα υψηλής σύνθεσης (HLS) να εξετάσει τη δυνατότητα εκτέλεσης κώδικα με dataflow παραλληλισμό, αλλά δεν εγγυάται αυτόματα τη μείωση των κύκλων εκτέλεσης. Ο παραλληλισμός dataflow σημαίνει ότι τα δεδομένα ρέουν από την είσοδο στην έξοδο χωρίς να υπάρχει συγκεκριμένη σειρά εκτέλεσης. Αν οι εξαρτήσεις μεταξύ των διαφόρων τμημάτων του κώδικα είναι περιορισμένες, τότε μπορεί να επιτευχθεί παραλληλισμός.

Στην συνέχεια διαμορφώνουμε κατάλληλα τον κώδικα της συνάρτησης ώστε να μπορούμε να εφαρμόσουμε λειτουργικά dataflow ,ώστε να εκτελούμε ταυτόχρονα το βαρύ υπολογιστικά for loop που υπολογίζει τις αποστάσεις. Ουσιαστικά κάνουμε το βαρύ υπολογιστικά loop να εκτελείται από μια συνάρτηση και στην συνέχεια τροποποιούμε την συνάρτηση έτσι ώστε κάθε μια να εκτελεί ξεχωριστά slot του for loop.Τονίζουμε ότι για το dataflow είναι απαραίτητο να σπάμε και τους πίνακες εισόδου σε μικρότερους υποπίνακες για να μπορεί να γίνει το dataflow και προφανώς στο τέλος χρειάζεται ένα reduction για να ενώσουμε τα επιμέρους αποτελέσματα σε έναν πίνακα τον οποίο τελικά θα επιστρέψει το HW στο SW.Αυτές οι επιπλέον διαδικασίες split array και ένωση array κοστίζουν υπολογιστικά αλλα και πάλι λόγο του dataflow του βασικού loop έχουμε βελτίωση στα αποτελέσματα όπως βλέπουμε στην συνέχεια.

Αρχικά θα το χωρίσουμε σε 2 συναρτήσεις:

```
void compute_dist1(float data_hw_tmp1[MOVIES_NUM/2][USERS_NUM],float dists_hw_tmp1[MOVIES_NUM/2],int start,int N)
{
    float movie_tmp1[USERS_NUM];
    LOAD_MOVIE_TMP:
        for (int i = 0; i < USERS_NUM ; i++){
            #pragma HLS unroll
            movie_tmp1[i] = data_hw_tmp1[MOVIE_ID][i];
        }

    for (int i = start; i < MOVIES_NUM/N + start; i++) {
        float sum = 0.0, diff = 0.0;

        for (int j = 0; j < USERS_NUM; j++){
          #pragma HLS unroll
            diff = data_hw_tmp1[i][j] - movie_tmp1[j];
        }
            sum += diff * diff;
            dists_hw_tmp1[i] = sqrt(sum);
    }
}
void calcDistancesHW(float* data_hw, float* dists_hw)
{
    #pragma HLS loop_merge
    float data_hw_tmp1[MOVIES_NUM/2][USERS_NUM],data_hw_tmp2[MOVIES_NUM/2][USERS_NUM];
    float dists_hw_tmp1[MOVIES_NUM/2],dists_hw_tmp2[MOVIES_NUM/2];

 LOAD_DATA_HW_TMP:
    for (int i = 0; i < MOVIES_NUM; i++) {
        for (int j = 0; j < USERS_NUM; j++) {
             #pragma HLS unroll
            if(i<MOVIES_NUM/2)
            data_hw_tmp1[i][j] = data_hw[i * USERS_NUM + j];
            else
            data_hw_tmp2[i-(MOVIES_NUM/2)][j] = data_hw[i * USERS_NUM + j];
        }
    }

COMPUTE_DISTS:
#pragma HLS dataflow
{
 compute_dist1( data_hw_tmp1,  dists_hw_tmp1,0,2);
 compute_dist1( data_hw_tmp2,  dists_hw_tmp2,MOVIES_NUM/2,2);
}


WRITE_DISTS:
    for (int i = 0; i < MOVIES_NUM; i++) {
        #pragma HLS unroll
        if(i<MOVIES_NUM/2)
        dists_hw[i] = dists_hw_tmp1[i];
        else
        dists_hw[i] = dists_hw_tmp2[i -(MOVIES_NUM/2)];
    }
}
```

Αποτελέσματα:

Summary

Latency		Interval		
min	max	min	max	Type
14341	14341	12291	12291	dataflow

Detail

Instance

Instance	Module	Latency min	Latency max	Interval min	Interval max	Type
Block_calcDistancesH_U0	Block_calcDistancesH	1024	1024	1024	1024	none
compute_dist2_U0	compute_dist2	12290	12290	12290	12290	none
compute_dist1_U0	compute_dist1	12290	12290	12290	12290	none
Loop_LOAD_DATA_HW_TM_U0	Loop_LOAD_DATA_HW_TM	1025	1025	1025	1025	none

Loop

N/A

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.68	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
14341	14341	12291	12291	dataflow

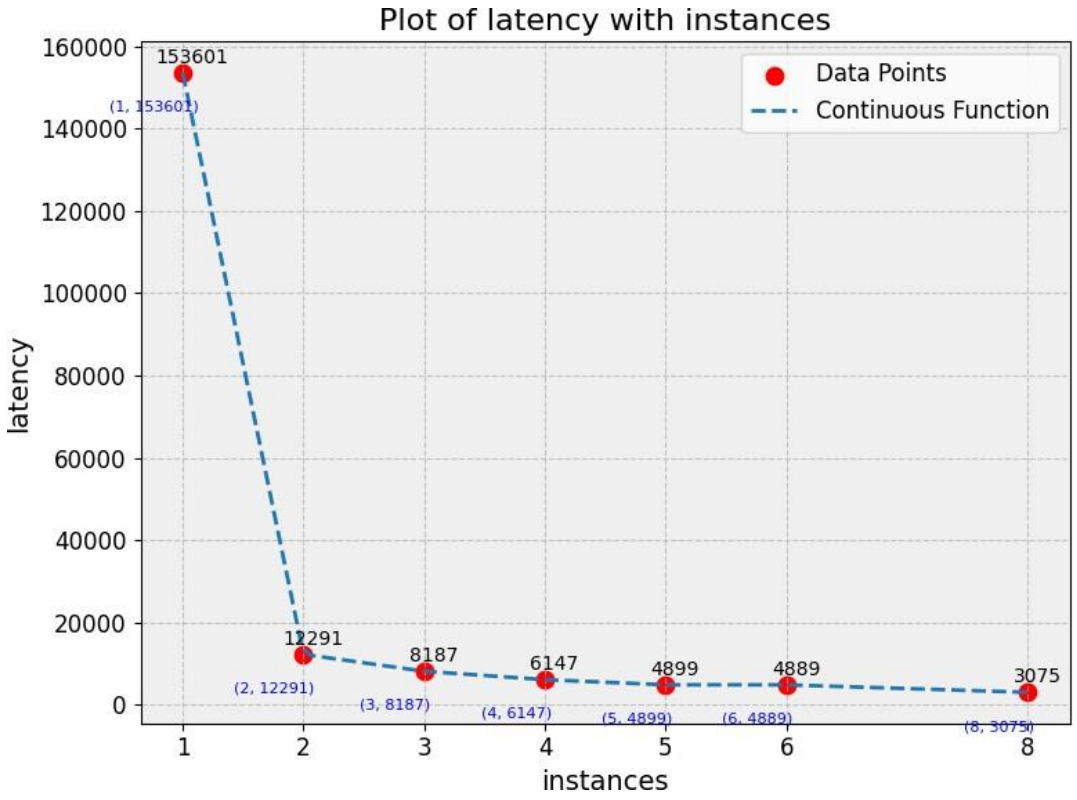
Summary

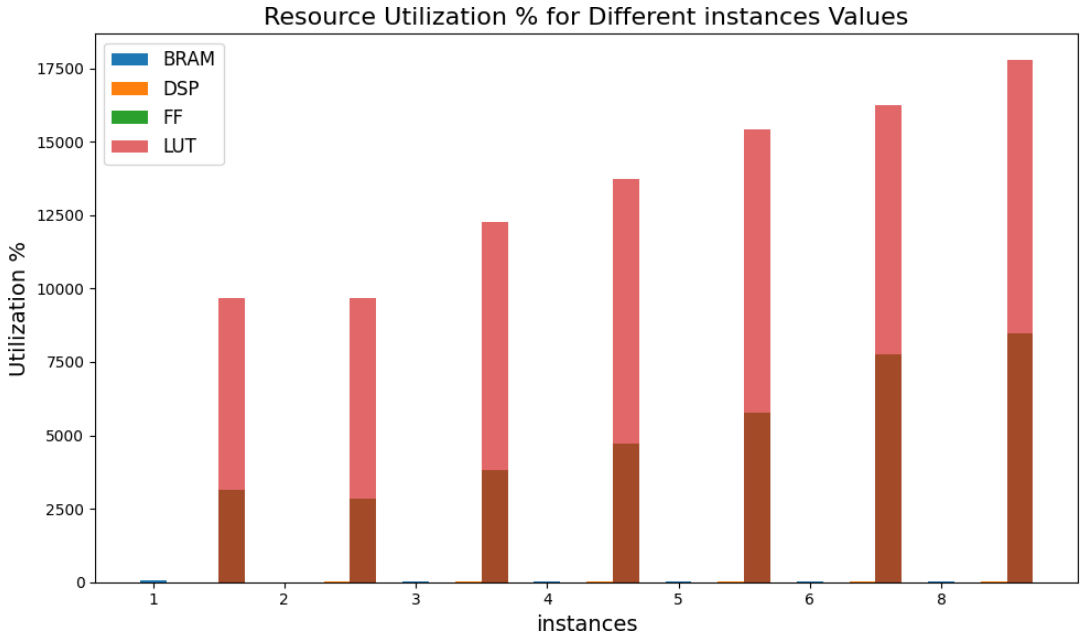
Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	7
FIFO	-	-	-	-
Instance	-	10	2848	9652
Memory	8	-	0	0
Multiplexer	-	-	-	2
Register	-	-	2	-
Total	8	10	2850	9661
Available	120	80	35200	17600
Utilization (%)	6	12	8	54

Παρατηρούμε: Ότι μειώνεται σημαντικά το estimated Clock σε 8.68 και η καθυστέρηση (latency) μειώνεται από 15000 σε 14341 κύκλους και το (interval) ο χρόνος εκτέλεσης μειώνεται από 15000 σε 12291 κύκλους.Μια τέτοια μείωση είναι προφανής καθώς πλέον το loop του distance γίνεται σε ξεχωριστό μέρος του fpga το πρώτο μισό και σε άλλο component το άλλο μισό αλλά ταυτόχρονα .Επίσης λογικό είναι ότι αυξάνονται τα LUTS αλλά μειώνονται και τα flip flops .

Στην συνέχεια τρέχουμε με παρόμοιο τρόπο ώστε να μπορέσουμε να σπάσουμε το βαρύ υπολογιστικό κομμάτι (τον υπολογισμό της απόστασης) σε ακόμη περισσότερες συναρτήσεις που θα εκτελούνται ταυτόχρονα αφού είναι ανεξάρτητές.

x	Interval	Latency	BRAM	DSP	FF	LUT
1	153601	153601	66	5	3125	9683
2	12291	14341	8	10	2850	9661
3	8187	10237	12	15	3808	12276
4	6147	8197	16	20	4720	13748
5	4899	6949	20	25	5768	15429
6	4889	6949	24	30	7747	16229
8	3075	5125	32	40	8489	17783





Όσο αυξάνουμε το dataflow οι BRAM μειώνονται γιατί χρησιμοποιούμε μικρότερους πίνακες ,τα LUTS αυξάνονται σημαντικά γιατί θέλουμε πολλά components και λογικά κυκλώματα να τρέχουν ταυτόχρονα , ομοίως για τα DSP και τα FF. Παρατηρούμε ότι καθώς αυξάνονται τα instances και γίνονται περισσότερα μικρότεροι υπολογισμοί της απόστασης μειώνεται όπως είναι λογικό το latency , όμως αρχικά ο ρυθμός μείωσης πέφτει απότομα , επομένως θα πρέπει να σκεφτούμε το trade-off

,δηλαδή αν αξίζει να καταναλώσουμε επιπλέον πόρους ώστε να μειώσουμε το latency.
Είναι λοιπόν σαφές ότι δεν αξίζει να έχουμε πολλά instances γιατί ενώ η κατανάλωση πόρων αυξάνεται σχεδόν γραμμικά η μείωση του latency μειώνεται σχεδόν αντιστρόφως εκθετικά. Αξίζει λοιπόν στην συγκεκριμένη σχεδίαση να χρησιμοποιήσουμε το πολύ εως 4 instances με το μεγαλύτερο κέρδος να την έχουμε στα δυο instances. (σημειώνουμε ότι για 8 instances δεν γίνεται να υλοποιηθεί καθώς χρησιμοποιεί περισσότερους πόρους από όσους διαθέτει το fpga)

****

Β)Αρχικά θα φτιάξουμε ειδικά data_types για να δούμε βελτίωση λόγο της χρήσης custom data type.

Αρχικά θα επιδιώξουμε να βρούμε τις βέλτιστες τιμές των μεταβλητών INT_BITS και DEC_BITS με στόχο να δημιουργήσουμε custom datatypes και να βελτιώσουμε το design μας.

Έχουμε 2 πίνακες τους
```
DTYPE1 data_hw_tmp[MOVIES_NUM][USERS_NUM];
DTYPE1 movie_tmp[USERS_NUM];
```

Η data_hw_tmp[i][j] μας δίνει την βαθμολογία της i-οστης ταινίας από j-οστό χρήστη .Ενώ ομοιίως η movie_tpm[j] μας δίνει την βαμολογία για την ταινία που ψάχνουμε προτεινόμενα από τον j-στο χρήστη.Οπότε και οι δύο πίνακες εκφράζουν ουσιαστικά βαθμολογία και θυμίζουμε ότι οι βαθμολογίες που μπορεί να δώσει ο χρήστης είναι μια από τις παρακάτω τιμές:
{ 0, 0.5, 1, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0 }

Από την εντολή
```
typedef ap_ufixed<INT_BITS + DEC_BITS, INT_BITS, AP_RND>
DTYPE1;
```

το int_bits+dec_bits όλη η τιμή με
int_bits representing the numbers above the decimal point (including the sign bit) και dec_bits representing the value below the decimal point.(το AP_RND είναι για στρογγυλοποίηση)

Οπότε στην περίπτωση μας ακέραιο μέρος παίρνει τιμές 0 εώς 5 και δεκαδικό μέρος τις τιμές 0 και 5. Οπότε θα χρειαστούμε 3 bits για δεκαδικά 1 bits για ακέραιο.

Επιπλέον στον κώδικα έχουμε ως DTYPE1 sum, diff; που χρειάζεται diff_max*diff_max = 25 για ακέραιο μέρος και 0.25 για δεκαδικό μέρος .Ο αθροιστής παίρνει λοιπόν στην χειρότερη περίπτωση 25*32 =800 χειρότερη περίπτωση και το δεκαδικό μέρος και πάλι θα είναι 0 ή 0.25 ή 0.5 ή 0.75.

Συνεπώς θα χρειαστούμε τελικά INT_BITS 10 bits και DEC_BITS 2 bits ώστε ακόμη και στην χειρότερη περίπτωση να μην έχουμε overflow.

Τρέχουμε λοιπόν αρχικά τον κώδικα για DTYPE1 χωρίς καμία βελτιστοποίηση όπως μας δόθηκε

Details

Performance estimates for 'calcDistancesHW in main.cpp:24 ...

HW accelerated (Estimated cycles)	2006021

Resource utilization estimates for HW functions

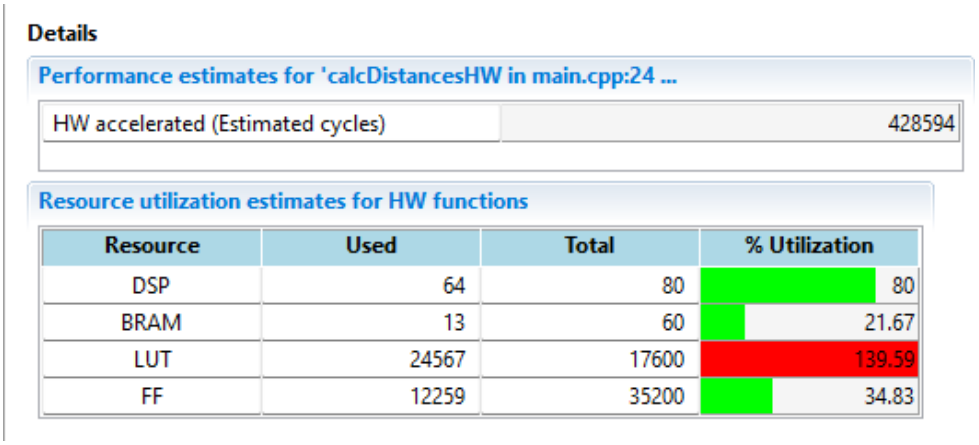
Resource	Used	Total	% Utilization
DSP	1	80	1,25
BRAM	13	60	21,67
LUT	2176	17600	12,36
FF	1138	35200	3,23

Παρατηρούμε λοιπόν ότι μόνο με την αλλαγή από float (32 bits) σε μόλις 12 bits έχουμε σημαντική μείωση στους κύκλους εκτέλεσης χωρίς κανένα optimization αλλά και πάλι χειρότερα αποτελέσματα από το SW οπότε θα χρειαστεί να κάνουμε και πάλι ένα optimization.Όπως φαίνεται και όταν φορτώσουμε την συγκεκριμένη υλοποίηση στο zybo έχουμε speedup = 0.6826 .

```
Recommendations for movie with id 0:
0. Jurassic Park (1993), with distance of 10.7121
1. Fish Called Wanda  A (1988), with distance of 11.0793
2. Back to the Future (1985), with distance of 11.6404
3. Star Wars: Episode VI - Return of the Jedi (1983), with distance of 11.8849
4. Lion King  The (1994), with distance of 12.0623
5. Raising Arizona (1987), with distance of 12.1552
6. Wizard of Oz  The (1939), with distance of 12.1861
7. Batman (1989), with distance of 12.2066
8. Princess Bride  The (1987), with distance of 12.3085
9. E.T. the Extra-Terrestrial (1982), with distance of 12.3996

Hardware cycles : 2025680
Software cycles : 1390095
Speedup         : 0.686236
Correct = 1024, Score = 1.000000
```

Το πρώτο πράγμα που θα εφαρμόσουμε είναι να υλοποιήσουμε για DTYPE1 ακριβώς τις ίδιες βελτιστοποιήσεις που εφαρμόσαμε και για δεδομένα float και έχουμε τα παραπάνω αποτελέσματα.



Details

Performance estimates for 'calcDistancesHW in main.cpp:24 ...

HW accelerated (Estimated cycles)	428594

Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	64	80	80
BRAM	13	60	21.67
LUT	24567	17600	139.59
FF	12259	35200	34.83

Παρατηρούμε ότι χρησιμοποιούμε υπερβολικούς πόρους ειδικά σε LUTs ,έχουμε μόνο μείωση των BRAMS αφού έχουμε μικρότερα δεδομένα (σε σημείο να μην γίνεται να τα εφαρμόσουμε στο zybo) χωρίς να έχουμε καμία βελτίωση στους χρόνους εκτέλεσης. Αυτό πιθανόν να οφείλεται στο γεγονός ότι πλέον ότι η compute_dist έχει ακόμη μεγαλύτερο φόρτο με πρίν καθώς πλέον έχουμε usigned δεδομένα οπότε στο diff βάζουμε πλέον απόλυτη τιμή ώστε να μην έχουμε overflow.

```
diff = (data_hw_tmp[i][j] > movie_tmp[j]) ? data_hw_tmp[i][j] - movie_tmp[j] : movie_tmp[j] - data_hw_tmp[i][j];
```

Συνεπώς θα χρειαστεί να αλλάξουμε κάποιες βελτιστοποιήσεις και την λογική στην σχεδίαση ώστε να έχουμε καλύτερο χρόνο δεσμεύοντας όσο το δυνατόν λιγότερους πόρους γίνεται. Τελικά καταλήγουμε στον παρακάτω τελικό κώδικα.

Αυξήσαμε τα factors στο array partition το οποίοι ήταν εφικτό εφόσον μειώσαμε τα δεδομένα που επεξεργαζόμαστε και έχουμε διαθέσιμες περισσότερες block ram ,ώστε να βελτιώσουμε τους κύκλους εκτέλεσης παρατηρήσαμε ότι για factor > 8 χρησιμοποιούμε περισσότερους πόρους χωρίς να μειώνουμε τους κύκλους εκτέλεσης. Δεν καταφέραμε να βάλουμε pipeline σε όλες τις for καθώς load data εξωτερικό loop pipiline αυξανει δραματικα τα LUTS.

```
void calcDistancesHW(float* data_hw, float* dists_hw)
{
    DTYPE1 data_hw_tmp[MOVIES_NUM][USERS_NUM];
    DTYPE1 movie_tmp[USERS_NUM];

    float dists_hw_tmp[MOVIES_NUM];
    #pragma HLS array_partition variable=data_hw_tmp cyclic dim=2 factor=8
    #pragma HLS array_partition variable=data_hw_tmp cyclic dim=1 factor=8
    #pragma HLS array_partition variable=movie_tmp cyclic dim=1 factor=8
    #pragma HLS array_partition variable=dists_hw_tmp cyclic dim=1 factor=8
    int i, j;

LOAD_DATA_HW_TMP:
    for (i = 0; i < MOVIES_NUM; i++) {
//        #pragma HLS unroll factor=8
        for (j = 0; j < USERS_NUM; j++) {
            #pragma HLS pipeline II=1
//            #pragma HLS unroll factor=8
            data_hw_tmp[i][j] = data_hw[i * USERS_NUM + j];
        }
    }

LOAD_MOVIE_TMP:
    for (i = 0; i < USERS_NUM; i++){
//        #pragma HLS pipeline II=1
        #pragma HLS unroll factor = 8
        movie_tmp[i] = data_hw_tmp[MOVIE_ID][i];
    }
    DTYPE1 sum, diff;
COMPUTE_DISTS:
   for (i = 0; i < MOVIES_NUM; i++) {
      #pragma HLS unroll factor=2`
      #pragma HLS pipeline II=1
      sum = (DTYPE1)0.0;
      diff = (DTYPE1)0.0;
      for (j = 0; j < USERS_NUM; j++) {
          #pragma HLS unroll factor=8
          #pragma HLS pipeline II=1
      diff = (data_hw_tmp[i][j] > movie_tmp[j]) ? data_hw_tmp[i][j] - movie_tmp[j] : movie_tmp[j] - data_hw_tmp[i][j];
          sum += diff * diff;
      }
      dists_hw_tmp[i] = sqrt(sum.to_float());
   }

WRITE_DISTS:
   for (i = 0; i < MOVIES_NUM; i++) {
//       #pragma HLS unroll factor=2
      #pragma HLS pipeline II=1
      dists_hw[i] = dists_hw_tmp[i];
   }

}
```

Αποτελέσματα:

Details

Performance estimates for 'calcDistancesHW in main.cpp:24 ...

HW accelerated (Estimated cycles)	392601

Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	64	80	80
BRAM	36	60	60
LUT	10108	17600	57,43
FF	4236	35200	12,03

Παρατηρούμε ότι έχουμε κάποια βελτίωση στους κύκλους εκτέλεσης σε σχέση με την χρήση float.

```
Recommendations for movie with id 0:
0. Jurassic Park (1993), with distance of 10.7121
1. Fish Called Wanda  A (1988), with distance of 11.0793
2. Back to the Future (1985), with distance of 11.6404
3. Star Wars: Episode VI - Return of the Jedi (1983), with distance of 11.8849
4. Lion King  The (1994), with distance of 12.0623
5. Raising Arizona (1987), with distance of 12.1552
6. Wizard of Oz  The (1939), with distance of 12.1861
7. Batman (1989), with distance of 12.2066
8. Princess Bride  The (1987), with distance of 12.3085
9. E.T. the Extra-Terrestrial (1982), with distance of 12.3996

Hardware cycles : 400719
Software cycles : 1390141
Speedup         : 3.46912
Correct = 1024, Score = 1.000000
```

	float Simple1	float Optimized1	Dtype1 Simple2	Dtype1 Optimized2
clocks	3244037	429296	2006021	392601
DSP	6.25	35	1.21	80
BRAMS	55	55	21.67	60
LUTs	9.39	60.34	12.36	57.43
FF	3.01	24.11	3.23	12.03
speedup	0.425847	3.03361	0.6826	3.46912