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

Σχεδιασμός Ενσωματωμένων Συστημάτων
9^{Ο} Εξάμηνο ΗΜΜΥ

3η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

Εισαγωγική εργασία στο High Level Synthesis

Ημ/νια Παράδοσης : 6/12/2024
Ομάδα 22
Παναγιώτης Μπέλσης AM : 03120874
Θεοδώρα Εξάρχου AM : 03120865

Ο σκοπός της άσκησης είναι να γίνει εισαγωγή στον προγραμματισμό Field Programmable Gate Arrays (FPGA) με High Level Synthesis (HLS). Η άσκηση αυτή παρουσιάζει ένα Recommendation System (RS) για ταινίες βασισμένο στον αλγόριθμο K Nearest Neighbors (KNN), χρησιμοποιώντας ένα υποσύνολο του MovieLens dataset. Με τη βοήθεια του HLS επιταχύνουμε τον χρόνο εκτέλεσης του RS, χρησιμοποιώντας το περιβάλλον ανάπτυξης SDSoC 2016.4. Τέλος, εκτελούμε την εφαρμογή στο Zynq-7000 ARM/FPGA SoC (Zybo).

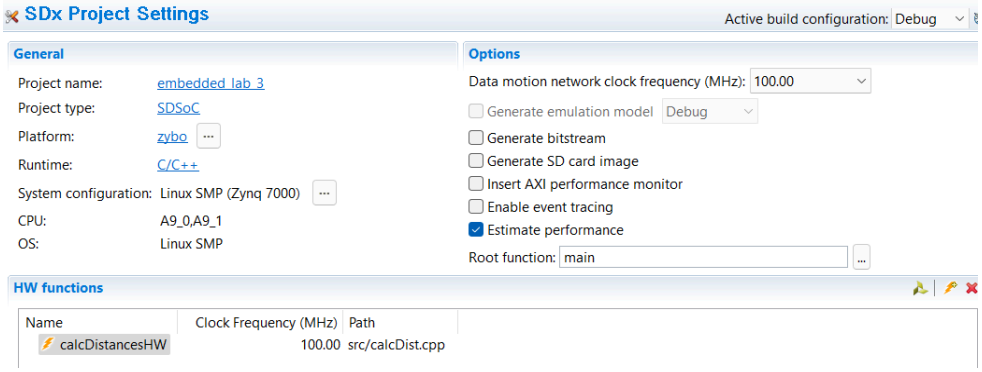
Επιθυμούμε να πετύχουμε μεγιστοποίηση του αρχικού speedup με χρήση HLS pragmas στην Hardware υλοποίηση.

Η συνάρτηση η οποία εκτελείται στο hardware είναι η *calcDistanceHW()* η οποία υπολογίζει τις ευκλείδειες αποστάσεις για την εύρεση των 10 πιο σχετικών ταινιών για την ταινία με id 0 από όλο το dataset. Ουσιαστικά, κάνουμε το computational intensive τμήμα στο FPGA και έπειτα επιστρέφουμε τα δεδομένα πίσω στον host.

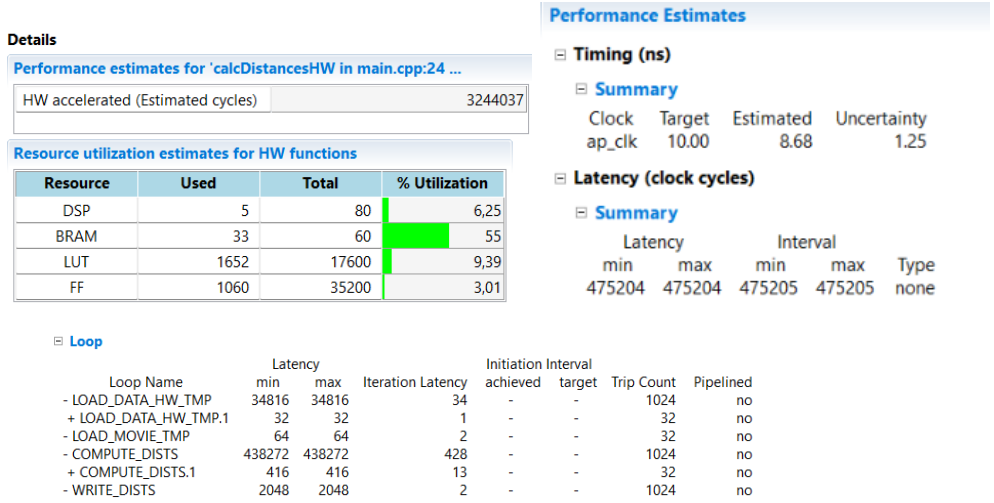
Ερώτημα 1

1.Α)

Αρχικά τρέχουμε το estimation της εφαρμογής χωρίς κανένα optimization



Το estimation είναι :



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

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

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

Για το Hardware απαιτούνται συνολικά 3.244.037 κύκλοι με το estimation.
Όπως περιμέναμε, το loop που απαιτεί τους περισσότερους κύκλους είναι το COMPUTE_DISTS 438.272 κύκλους , στο οποίο υπολογίζονται οι ευκλείδειες αποστάσεις και γίνεται χρήση 2x2 πινάκων και πολλαπλασιασμού το οποίο γνωρίζουμε ότι είναι χρονοβόρο.

1.Β)

Μετά την ολοκλήρωση της δημιουργίας του bitstream και την εγγραφή του στην κάρτα SD, το σύστημα μεταφέρθηκε στο Zybo για εκτέλεση.

Αποτελέσματα που προκύπτουν από το zybo (κανένα optimization):

```
sh-4.3# ./embedded_lab_3.elf
Started reading dataset...
Finished reading dataset...
Started reading name id mapping...
Finished reading name id mapping...
Input movie id = 0
Started distance calculations on software...
Finished distance calculations on software...
Started distance calculations on hardware...
Finished distance calculations on hardware...

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 : 3264697
Software cycles : 1390086
Speedup         : 0.425793
Correct = 1024, Score = 1.000000
sh-4.3#
```

Τα αποτελέσματα που προέκυψαν, χωρίς καμία βελτιστοποίηση, δείχνουν ότι απαιτούνται **3.264.697 κύκλοι** και **speedup 0.42** .Με το Estimation είχαμε 3.244.037 κύκλους, τιμή που βρίσκεται πολύ κοντά στην πραγματική μέτρηση.Παρ'ολα αυτά παραμένει μια αρκετά καλή πρόβλεψη.

1.Γ) Pipelining

Κάνουμε τροποποίηση του κώδικα για pipeline design II=1 σε κάθε loop **χωρίς χρήση κάποιου άλλου optimization**. Ουσιαστικά προσθέτουμε το *#pragma HLS PIPELINE II=1* μέσα σε όλα τα for loop. Στα διπλά loop μόνο εσωτερικά. Το **pipelining** είναι μια τεχνική που επιτρέπει την ταυτόχρονη εκτέλεση πολλών λειτουργιών (ή βημάτων) μέσα σε έναν βρόχο ή μια συνάρτηση. Με τη χρήση αποκλειστικά **pipeline**, παρατηρούμε ότι οι πόροι του **FPGA** επαρκούν, οπότε δεν αντιμετωπίζουμε πρόβλημα από αυτήν την άποψη. Ωστόσο, το **Initiation Interval (II)**=1 που ορίσαμε για τη συνάρτηση

COMPUTE_DISTS δεν το πετυχαίνουμε, γεγονός που οφείλεται στις εξαρτήσεις δεδομένων. Τελικά, το (II) παίρνει την αμέσως μεγαλύτερη δυνατή τιμή από το επιθυμητο, στην προκειμένη περίπτωση (II)= 3 .
Για τη συνάρτηση **COMPUTE_DISTS**:
→**Για τον εξωτερικό βρόχο:** Δεν υπάρχει εξάρτηση δεδομένων μεταξύ διαφορετικών τιμών του i. Επομένως, ο εξωτερικός βρόχος μπορεί θεωρητικά να έχει **II=1**.
→**Για τον εσωτερικό βρόχο:** Υπάρχει εξάρτηση δεδομένων στο **sum**, που σημαίνει ότι ο εσωτερικός βρόχος δεν μπορεί να έχει **II=1**. Τα δεδομένα σε κάθε επανάληψη εξαρτώνται από προηγούμενες επαναλήψεις. Το **II** δεν μπορεί να είναι **1**, και αυτό επιβεβαιώνεται και με το **estimate**, καθώς βλέπουμε ότι παρόλο που βάλαμε **Initiation Interval = 1** σε όλα τα loops στο εσωτερικό, δεν κατάφερε να επιτευχθεί και έχουμε **II=3**.

εκτέλεση estimation :

Details

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

HW accelerated (Estimated cycles)	1014225

Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	5	80	6,25
BRAM	33	60	55
LUT	1859	17600	10,56
FF	1096	35200	3,11

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- LOAD_DATA_HW_TMP_L	32768	32768	2	1	1	32768	yes
- LOAD_MOVIE_TMP	32	32	2	1	1	32	yes
- COMPUTE_DISTS_L	98324	98324	24	3	1	32768	yes
- WRITE_DISTS	1024	1024	2	1	1	1024	yes

Για το Hardware απαιτούνται συνολικά 1.014.225 κύκλοι με το estimation.Παρατηρούμε ότι έχουμε μείωση των κύκλων κατά περίπου **68.7%**. Το pipelining μπορεί να μειώσει σημαντικά τον χρόνο εκτέλεσης και να βελτιώσει την απόδοση του hardware.

```
sh-4.3# ./embedded_lab_3.elf
Started reading dataset...
Finished reading dataset...
Started reading name id mapping...
Finished reading name id mapping...
Input movie id = 0
Started distance calculations on software...
Finished distance calculations on software...
Started distance calculations on hardware...
Finished distance calculations on hardware...

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 : 1033062
Software cycles : 1390081
Speedup         : 1.34559
Correct = 1024, Score = 1.000000
sh-4.3#
```

Τα αποτελέσματα που προέκυψαν, με Pipeline optimization είναι για το Hardware 1.033.062 CC και **speedup=1.34**. Παλι εχουμε μια μικρή απόκλιση σε σχέση με το Estimation που είχαμε 1.014.225.
Συγκριτικά με το without optimization παρατηρούμε ότι έχουμε μείωση των κύκλων και στο Zybo της τάξης **68.3%**. Για το speedup παρατηρούμε ότι έχουμε αύξηση, που δείχνει ότι το pipelining έχει σημαντική θετική επίδραση στην απόδοση του hardware.

Στη συνέχεια δοκιμάσαμε διάφορους συνδυασμούς μεθόδων optimization με διαφορετικά configurations, με σκοπό την μεγιστοποίηση του speedup. Παραθέτουμε το παρακάτω πινακάκι με τις υλοποιήσεις που δοκιμάσαμε, καθως και τα αποτελέσματα που προέκυψαν:

Pipeline	Loop Unrolling	Array Partition	Estimation (CC)	Speedup
–	–	–	3.264.697	0.42
–	Compute=4 Other=2	–	1.806.341	0.76
Partial	–	–	1.014.225	1.34
Full	–	–	482.590	
Full	All=4	–	482.486	
Full	Compute=4	dim=1 factor=4	482.408	2.76
Full	Compute=4	dim=1 factor=16	482.382	2.76
Full	–	dim=1 factor=2	482.538	2.77
Full	Compute=32	dim=1 factor=32	482.492	
Full	Compute=8	dim=1 factor=16	482.486	2.76

Επεξήγηση του πίνακα:

Pipeline Partial σημαίνει ότι βάλαμε #pragma HLS PIPELINE II=1 μέσα σε όλα τα for loop. Στα διπλά loop μόνο εσωτερικά.
Pipeline Full σημαίνει ότι βάλαμε #pragma HLS PIPELINE II=1 μέσα σε όλα τα for loop. Στα διπλά loop εσωτερικά και εξωτερικά.

Loop Unrolling Compute=4 σημαίνει ότι στο COMPUTE_DISTS στο 2πλο for loop βάλαμε #pragma HLS unroll factor=4 στον εσωτερικό βρόχο. Το other=2 σημαίνει ότι στα υπόλοιπα for loop του κώδικα βάλαμε #pragma HLS unroll factor=2. Το all=4 σημαίνει ότι σε ολα τα for loop δοκιμάσαμε unrolling 4.

Array Partition factor=4 σημαίνει ότι σε όλα τα float arrays που ορίζονται πάνω πάνω στον κώδικα τα κάνουμε partition με factor =4.

```
        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 factor=4 dim=1
#pragma HLS ARRAY_PARTITION variable=movie_tmp block factor=4 dim=1
#pragma HLS ARRAY_PARTITION variable=dists_hw_tmp block factor=4 dim=1
```

Το καλύτερο configuration που προκύπτει μέχρι στιγμής είναι:

BEST:

Pipeline	Loop Unrolling	Array Partition	Estimation (CC)	Speedup
Full	–	dim=1 factor=2	482.538	2.77

με **speedup=2.77**

Estimation for the best configuration:

Details

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

HW accelerated (Estimated cycles)	482538

Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	14	80	17,5
BRAM	33	60	55
LUT	6038	17600	34,31
FF	5046	35200	14,34

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
50358	50358	50359	50359	none

Detail

Instance

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- LOAD_DATA_HW_TMP_L	32768	32768	2	1	1	32768	yes
- LOAD_MOVIE_TMP	32	32	2	1	1	32	yes
- COMPUTE_DISTS	16518	16518	151	16	1	1024	yes
- WRITE_DISTS	1024	1024	2	1	1	1024	yes

Zybo run for the best configuration:

```
sh-4.3# ./embedded_lab_3.elf
Started reading dataset...
Finished reading dataset...
Started reading name id mapping...
Finished reading name id mapping...
Input movie id = 0
Started distance calculations on software...
Finished distance calculations on software...
Started distance calculations on hardware...
Finished distance calculations on hardware...

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 : 501159
Software cycles : 1390065
Speedup         : 2.7737
Correct = 1024, Score = 1.000000
sh-4.3#
```

Έπειτα παρατηρήσαμε ότι στο *calcDist.h* αρχείο που μας δόθηκε μπορούμε να κάνουμε μία αλλαγή για την επίτευξη ακόμα καλύτερου speedup.

calcDist.h original file

```
#define MOVIES_NUM 1024
#define USERS_NUM 32

#define ITERATIONS 1024

#define MOVIE_ID 0

#pragma SDS data copy(data_hw[0:32768], dists_hw[0:32])
#pragma SDS data access_pattern(data_hw:SEQUENTIAL, dists_hw:SEQUENTIAL)
void calcDistancesHW(float* data_hw, float* dists_hw);
```

```
#pragma SDS data copy(data_hw[0:32768], dists_hw[0:32])
```

Η παραπάνω εντολή κάνει specify το μέγεθος των δεδομένων που θα μεταφερθούν στη μνήμη του FPGA.
Έχουμε 1024 Movies, 32 Users, άρα data_hw[0:1024*32]

Για την dists_hw κάνουμε specify ότι θα δεχτεί μέγεθος δεδομένων 32 στοιχεία, ένω συνολικά θα προκύψουν 1024 αποστάσεις, όσες και οι ταινίες. Το να έχουμε βάλει dists_hw[0:32] αντί για dists_hw[0:1024] είναι μη αποδοτικό γιατί δεν εκμεταλλευόμαστε όλο το memory bandwidth. Με το dists_hw[0:32] επεξεργαζόμαστε κάθε φορά 32 στοιχεία σε κάθε κλήση.

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

όπως φαίνεται από το παραπάνω κομμάτι κώδικα από την *calDist.cpp* το loop κάνει 1024 iterations παράλληλα (pipelined II=1).
Εάν χωράνε και τα 1024 στοιχεία στη BRAM του FPGA, τότε μας συμφέρει να περάσουμε όλα τα στοιχεία με 1 κλήση και όχι σε κομμάτια. Δοκιμάζουμε λοιπόν την υλοποίηση αλλάζοντας την συγκεκριμένη εντολή έτσι:

```
#pragma SDS data copy(data_hw[0:32768], dists_hw[0:1024])
```

Τρέχουμε το estimation:

Details

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

HW accelerated (Estimated cycles)	2147483647

Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	14	80	17,5
BRAM	41	60	68,33
LUT	12268	17600	69,7
FF	11455	35200	32,54

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	2239
FIFO	-	-	-	-
Instance	-	14	1511	2863
Memory	66	-	128	16
Multiplexer	-	-	-	2188
Register	-	-	4218	755
Total	66	14	5857	8061
Available	120	80	35200	17600
Utilization (%)	55	17	16	45

Detail

Το estimation φαίνεται να βγάζει περίπου 2δις κύκλους, καμία σχέση με το run στο zybo.

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	16518	16518	151	16	1	1024	yes
- WRITE_DISTS	1024	1024	2	1	1	1024	yes

Τρέχουμε και στο board εφαρμόζοντας την παραπάνω αλλαγή.

BEST

```
sh-4.3# ./lab3.elf
Started reading dataset...
Finished reading dataset...
Started reading name id mapping...
Finished reading name id mapping...
Input movie id = 0
Started distance calculations on software...
Finished distance calculations on software...
Started distance calculations on hardware...
Finished distance calculations on hardware...

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 : 349628
Software cycles : 1389914
Speedup         : 3.97541
Correct = 1024, Score = 1.000000
sh-4.3#
```

Προκύπτει τελικά **speedup=3.97** και score=1, που σημαίνει ότι χώρεσαν όλα τα στοιχεία στην BRAM και βγάλαμε σωστά αποτελέσματα.

Το estimation έπεσε εντελώς έξω σε σχέση με αυτό που συμβαίνει πραγματικά

Συμπεράσματα:

Για τους διάφορους συνδυασμούς βελτιστοποιήσεων, καταλήξαμε στα εξής αποτελέσματα:

→Το pipelining μόνο του αποδεικνύεται πιο αποτελεσματικό όσον αφορά τη μείωση των κύκλων και τη βελτίωση του speedup σε σχέση με το loop unrolling μόνο του.

→Ωστόσο, κατά τη χρήση του full pipelining παρατηρούμε ότι χάνουμε το επιθυμητό Initiation Interval, το οποίο είχαμε θέσει στο 1, με το LOAD_DATA_HW_TMP να έχει 32 και το COMPUTE_DISTS 16, αντίστοιχα.

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	16518	16518	151	16	1	1024	yes
- WRITE_DISTS	1024	1024	2	1	1	1024	yes

Μετά από πολλές δοκιμές, καταφέραμε να μειώσουμε το Initiation Interval του LOAD_DATA_HW_TMP σε 2, κάτι που επιτεύχθηκε μέσω της συνδυασμένης χρήσης full pipeline και unrolling με παράγοντα 4. Στη συνέχεια, καταφέραμε να το επαναφέρουμε στο Initiation Interval 1 με τη χρήση full pipeline και array partition με παράγοντα 16 και με 2. Συμπεραίνουμε ότι πρέπει να αναζητήσουμε τον κατάλληλο συνδυασμό παραμέτρων για κάθε περίπτωση, καθώς δεν είναι δεδομένο ότι η χρήση του array partition θα λύσει πάντα το πρόβλημα. Για παράδειγμα, όταν χρησιμοποιούμε array partition με παράγοντα 8, δεν πετύχαμε την επιθυμητή βελτίωση. Το COMPUTE_DISTS, παραμένει στο 16 παρά τις διάφορες προσπάθειες για την επίλυση.

→Το array partition είναι ουσιαστικά η διαίρεση ενός πίνακα σε μικρότερους, με σκοπό την αύξηση του speedup. Για την άσκηση αυτή, πιο κατάλληλο type θεωρείται το block (μικρότερους πίνακες από διαδοχικά μπλοκ του αρχικού πίνακα), και όχι το cyclic και το complete..Όταν εισαγάγαμε το array partition για τους πίνακες, παρατηρήσαμε σημαντική αύξηση του speedup. Αυτή η βελτίωση καταδεικνύει την αποτελεσματικότητα της συγκεκριμένης τεχνικής στην αύξηση της απόδοσης του συστήματος για το συγκεκριμένο πρόβλημα.

→Το ίδιο ισχύει και για το factor του array partition, καθώς δεν σημαίνει πάντα ότι όσο μεγαλύτερος είναι ο παράγοντας, τόσο μεγαλύτερη είναι η απόδοση. Στην περίπτωση του δικού μας σχεδιασμού το array partition με factor = 2 απέδωσε καλύτερα σε σχέση με μεγαλύτερους παράγοντες. Αυτή η βελτιστοποίηση μέσω του array partition εξαρτάται από τις συγκεκριμένες συνθήκες του συστήματος.

→Για τα resources που καταναλώθηκαν, σε καμία περίπτωση δεν ξεπεράστηκαν οι πόροι που είναι διαθέσιμοι. Οι βελτιστοποιήσεις που εφαρμόστηκαν εξασφάλισαν ότι οι απαιτήσεις σε καταχωρητές, μνήμες και άλλους πόρους παραμένουν εντός των ορίων του συστήματος.

Ερώτημα 2

Σε αυτό το ερώτημα χρησιμοποιήθηκε η βιβλιοθήκη ap_fixed με στόχο να δημιουργηθούν custom datatypes και να βελτιωθεί το design μας με τη χρήση του τύπου DTYPE1. Αναπαριστά έναν unsigned fixed-point αριθμό με καθορισμένο αριθμό bits για τον ακέραιο και το δεκαδικό μέρος.Για να βρεθούν οι βέλτιστες τιμές για το ακέραιο και το δεκαδικό του τμήμα, κάναμε τα εξής ,γνωρίζοντας οτι ο πίνακας *data_hw_tmp[i][j]* αποθηκεύει την βαθμολογια της i ταινιας απο τον j χρηστη, ο *movie_tmp[ ]* μονοδιάστατος πίνακας αποθηκεύει τα δεδομένα για μια συγκεκριμένη ταινία και ο *dists_hw_tmp[ ]* μονοδιάστατος πίνακας αποθηκεύει τις αποστάσεις μεταξύ ταινιών και χρηστών.

Με βάση οτι η βαθμολογία είναι ένα αριθμός από το σύνολο:

{ 0, 0.5, 1, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0 }

→ Μέγιστη διαφορά μεταξύ δυο βαθμολογιών diff ειναι 5
→ Μέγιστο τετράγωνο είναι το 25 (*sum += diff * diff*)
→ Μέγιστο άθροισμα sum =users*25 =32*25 =800

```
#include <stdlib.h>
#include <ap_fixed.h>

#define MOVIES_NUM 1024
#define USERS_NUM 32

#define ITERATIONS 1024

#define MOVIE_ID 0

#define INT_BITS 10
#define DEC_BITS 4
typedef ap_ufixed<INT_BITS + DEC_BITS, INT_BITS, AP_RND> DTYPE1;
```

Το sum αντιπροσωπεύει το μέγιστο άθροισμα τετραγώνων των διαφορών βαθμολογιών.
Για το ακέραιο μέρος γνωρίζουμε ότι max= 800, άρα έχουμε: **INT_BITS=10**
Για το δεκαδικό μέρος παρατηρούμε από τα τρεξίματα στο Zybo ότι οι αποστάσεις έχουν ακρίβεια μέχρι 4 δεκαδικά ψηφία. Αυτό, για το δυαδικό σύστημα, σημαίνει αντίστοιχα 4 ψηφία: **DEC_BITS=4**

Εφαρμόζουμε την βέλτιστη λύση που μας είχε προκύψει παραπάνω με speedup=3.97 βάζοντας επιπλέον custom data types αντικαθιστώντας τις float μεταβλητές από τον original κώδικα.

ο κώδικας που προκύπτει είναι :

calcDist.c

```
#include <math.h>

#include "calcDist.h"

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

        int i, j;

LOAD_DATA_HW_TMP:
```

```
        for (i = 0; i < MOVIES_NUM; i++) {
#pragma HLS PIPELINE II=1
                for (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 (i = 0; i < USERS_NUM; i++){
#pragma HLS PIPELINE II=1
                movie_tmp[i] = data_hw_tmp[MOVIE_ID][i];
        }

        DTYPE1 sum, diff;
COMPUTE_DISTS:
        for (i = 0; i < MOVIES_NUM; i++) {
#pragma HLS PIPELINE II=1
                sum = (DTYPE1)0.0;
                diff = (DTYPE1)0.0;
                for (j = 0; j < USERS_NUM; j++) {
#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 PIPELINE II=1
                dists_hw[i] = dists_hw_tmp[i];
        }

}
```

calcDist.h

```
#include <stdlib.h>
#include <ap_fixed.h>

#define MOVIES_NUM 1024
#define USERS_NUM 32

#define ITERATIONS 1024

#define MOVIE_ID 0

#define INT_BITS 10
#define DEC_BITS 4
typedef ap_ufixed<INT_BITS + DEC_BITS, INT_BITS, AP_RND> DTYPE1;

#pragma SDS data copy(data_hw[0:32768], dists_hw[0:1024])
#pragma SDS data access_pattern(data_hw:SEQUENTIAL, dists_hw:SEQUENTIAL)
void calcDistancesHW(float* data_hw, float* dists_hw);
```

Τα αποτελέσματα του estimation που προκύπτουν είναι τα εξής:

Details

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

HW accelerated (Estimated cycles)	2147483647

Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	32	80	40
BRAM	32	60	53.33
LUT	28202	17600	160.24
FF	15673	35200	44.53

Σύμφωνα με το estimation, έχουμε ξεπεράσει τα διαθέσιμα LUT του Zybo.Δοκιμάζουμε να παράξουμε το bistream για την περίπτωση εσφαλμένου estimation.

Εκτέλεση της βέλτιστης προηγούμενης λύσης με χρήση custom data types.

```
sh-4.3# ./embedded_lab3_ex2.elf
Started reading dataset...
Finished reading dataset...
Started reading name id mapping...
Finished reading name id mapping...
Input movie id = 0
Started distance calculations on software...
Finished distance calculations on software...
Started distance calculations on hardware...
Finished distance calculations on hardware...

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 : 348217
Software cycles : 1390019
Speedup         : 3.99182
Correct = 1024, Score = 1.000000
sh-4.3#
```

Παρατηρούμε ότι επαρκούν τελικά τα resources του zybo και παίρνουμε **speedup = 3.99** (χωρίς custom data types για την ίδια υλοποίηση είχαμε πετύχει 3.97 προηγουμένως).

Από το estimation βλέπουμε ότι χρησιμοποιούμε λιγότερη BRAM με τη χρήση DTYPE1. Με τη χρήση του DTYPE1 ορίζουμε 10 bits για το ακέραιο μέρος και 4 bits για το δεκαδικό, καταναλώνοντας συνολικά μόνο 14 bits, δηλαδή πολύ λιγότερο χώρο στη μνήμη, ενώ στο αρχικό design, οι τιμές αποθηκεύονταν ως float, που απαιτούν 32 bits για κάθε αριθμό.
Αυτή η εξοικονόμηση χώρου στη μνήμη μας και των πόρων επιτρέπει να αξιοποιήσουμε καλύτερα τους διαθέσιμους πόρους και να εφαρμόσουμε πιο αποδοτικά optimizations.

Ερώτημα 3

Στο 3ο ερώτημα δοκιμάζουμε διαφορετικές υλοποιήσεις για την επίτευξη ακόμα καλύτερου speedup.

calcDist.cpp

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

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 block factor=32 dim=2
#pragma HLS ARRAY_PARTITION variable=movie_tmp block factor=32 dim=1
#pragma HLS ARRAY_PARTITION variable=dists_hw_tmp block factor=32 dim=1
//auta parapanw einai entaksei
        int i, j;

LOAD_DATA_HW_TMP:
        for (i = 0; i < MOVIES_NUM; i++) {
#pragma HLS PIPELINE II=1 //ama vgaleis auto to sxolio skaei
                for (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 (i = 0; i < USERS_NUM; i++){
#pragma HLS PIPELINE II=1 //auto einai entaksei
                movie_tmp[i] = data_hw_tmp[MOVIE_ID][i];
        }

        DTYPE1 sum, diff;
COMPUTE_DISTS:
        for (i = 0; i < MOVIES_NUM; i++) {
#pragma HLS PIPELINE II=1
                sum = (DTYPE1)0.0;
                diff = (DTYPE1)0.0;
                for (j = 0; j < USERS_NUM; j++) {
#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 PIPELINE II=1
                dists_hw[i] = dists_hw_tmp[i];
        }

}
```

Στην παραπάνω υλοποίηση έχουμε αλλάξει τα factor των array partition σε 32 και το dim του data_hw_tmp σε 2, δηλαδή όλοι οι users(32) μπαίνουν σε 1 block των 32 στοιχείων. Το υπόλοιπο κομμάτι του κώδικα μένει όπως πριν. Με τις διάφορες βελτιστοποιήσεις που πραγματοποιήσαμε για την επίτευξη καλύτερης απόδοσης, παρατηρήσαμε ότι για τον συγκεκριμένο κώδικα, η διάσπαση με dim=2 αποδεικνύεται πιο κατάλληλη και αποδοτική σε σχέση με τη διάσπαση με dim=1.

```
sh-4.3# ./lab3_2.elf
Started reading dataset...
Finished reading dataset...
Started reading name id mapping...
Finished reading name id mapping...
Input movie id = 0
Started distance calculations on software...
Finished distance calculations on software...
Started distance calculations on hardware...
Finished distance calculations on hardware...

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 : 248798
Software cycles : 1390239
Speedup         : 5.58782
Correct = 1024, Score = 1.000000
sh-4.3#
```

Η αλλαγή είχε πολύ μεγάλη βελτίωση στο speedup και πετυχαίνουμε **speedup= 5.58**.

Επεξήγηση αλλαγής από dim=1 σε dim=2

Επιτρέπει μεγαλύτερη ταχύτητα στην ανάγνωση και επεξεργασία των δεδομένων και καλύτερη εκμετάλλευση των υπολογιστικών πόρων.
Για dim=2 ο πίνακας διασπάται κατά τις στήλες δηλαδή χρήστες. Δηλαδή, κάθε χρήστης έχει τα δεδομένα του αποθηκευμένα σε συνεχόμενες θέσεις μνήμης.
Για το dim=1 τα δεδομένα αποθηκεύονται συνεχόμενα με βάση τις ταινίες, δηλαδή κάθε στοιχείο που αντιστοιχεί σε μια ταινία είναι αποθηκευμένο διαδοχικά στη μνήμη. Αυτό σημαίνει ότι οι βαθμολογίες όλων των χρηστών για μια συγκεκριμένη ταινία βρίσκονται σε συνεχόμενες θέσεις μνήμης.

Κάνοντας αλλαγή σε dim=2 επιτρέπουμε την παράλληλη επεξεργασία πολλών user ratings ταυτόχρονα (columns) για 1 ταινία. Η πρόσβαση γίνεται ανά χρήστη για αυτό το dim=2 ειναι πιο αποδοτικό για το κώδικα μας.

Δε χρειάζεται να επεξεργαστώ πολλές ταινίες ταυτόχρονα, αλλά πολλές κριτικές χρηστών ταυτόχρονα, για αυτό δεν επιλέγω dim=1.

Επεξήγηση αλλαγής από factor 2 σε factor 32
Με τη χρήση #pragma HLS ARRAY_PARTITION στον πίνακα data_hw_tmp, ο πίνακας με διαστάσεις 1024x32 διαχωρίζεται πλήρως κατά τη διάσταση dim=2. Άρα, η δεύτερη διάσταση του πίνακα, δηλαδή οι χρήστες, χωρίζεται σε 32 ανεξάρτητα blocks . Κάθε ένα από αυτά τα block μπορεί να προσπελαστεί ταυτόχρονα και ανεξάρτητα, επιτρέποντας τον πλήρη παραλληλισμό.Αυτό επιβεβαιώνεται και από το speedup, το οποίο αυξάνεται από 4.71 σε 5.58 με την αλλαγή του factor από 2 σε 32.

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

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 block  factor=32 dim=2
#pragma HLS ARRAY_PARTITION variable=movie_tmp block factor=32 dim=1
#pragma HLS ARRAY_PARTITION variable=dists_hw_tmp block factor=32 dim=1

        int i, j;

LOAD_DATA_HW_TMP:
        for (i = 0; i < MOVIES_NUM; i++) {
#pragma HLS PIPELINE II=1
        for (j = 0; j < USERS_NUM; j++) {
#pragma HLS PIPELINE II=1
//#pragma HLS unroll
                        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
                movie_tmp[i] = data_hw_tmp[MOVIE_ID][i];
        }

        DTYPE1 sum, diff;
COMPUTE_DISTS:
        for (i = 0; i < MOVIES_NUM; i++) {
#pragma HLS PIPELINE II=1
                sum = (DTYPE1)0.0;
                diff = (DTYPE1)0.0;
                for (j = 0; j < USERS_NUM; j++) {
//#pragma HLS PIPELINE II=1
#pragma HLS unroll factor=16
                        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 PIPELINE II=1
                dists_hw[i] = dists_hw_tmp[i];
        }

}
```

Τέλος, με την αντικατάσταση του `#pragma HLS PIPELINE II=1` σε `#pragma HLS unroll factor=16` στο εσωτερικό for loop του COMPUTE_DISTS λαμβάνουμε τα παρακάτω αποτελέσματα.

```
sh-4.3# ./embedded_lab3_ex2.elf
Started reading dataset...
Finished reading dataset...
Started reading name id mapping...
Finished reading name id mapping...
Input movie id = 0
Started distance calculations on software...
Finished distance calculations on software...
Started distance calculations on hardware...
Finished distance calculations on hardware...

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 : 247349
Software cycles : 1390376
Speedup         : 5.62111
Correct = 1024, Score = 1.000000
sh-4.3#
```

Τελικά πετυχαίνουμε **speedup = 5.62**.

Με τις βελτιστοποιήσεις που εξετάσαμε, καταφέραμε να επιτύχουμε αύξηση στην απόδοση, όμως δεν καταφέραμε να εφαρμόσουμε το PIPELINE με II=1 στον εξωτερικό βρόχο για το LOAD DATA HW TMP όπως φαίνεται στην παρακάτω εικόνα.

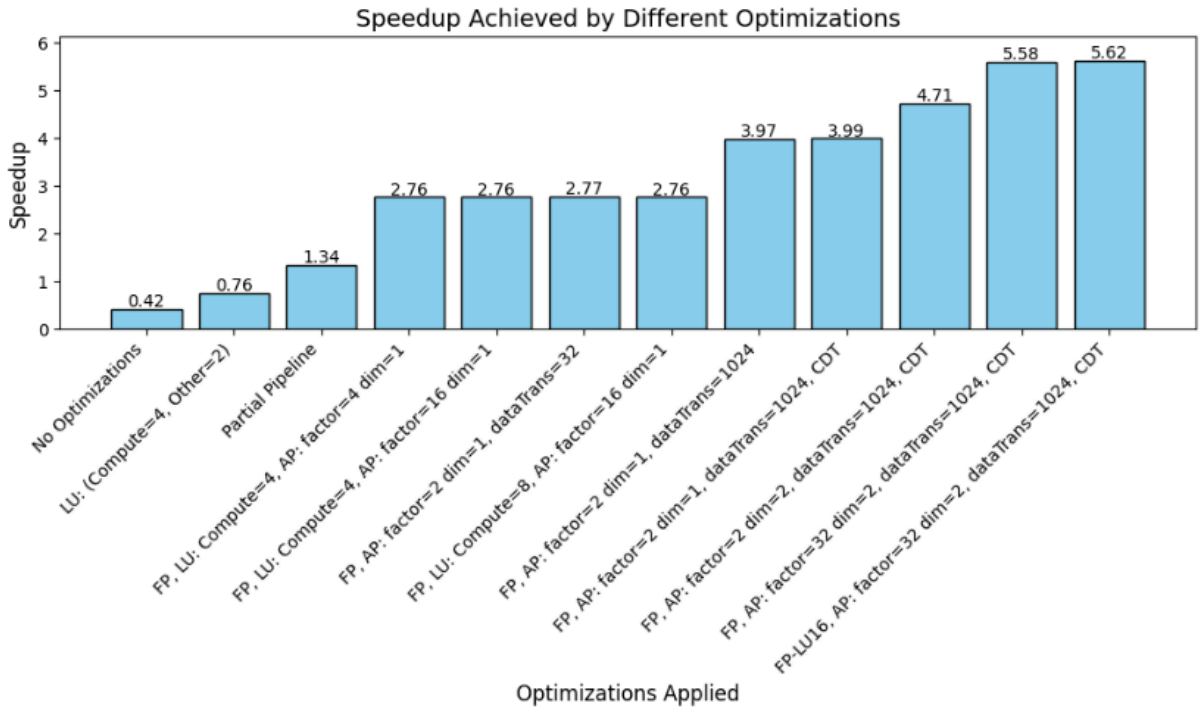
⊟ **Loop**

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- LOAD_DATA_HW_TMP	32771	32771	36	32	1	1024	yes
- LOAD_MOVIE_TMP	32	32	1	1	1	32	yes
- COMPUTE_DISTS	1054	1054	32	1	1	1024	yes
- WRITE_DISTS	1025	1025	3	1	1	1024	yes

Για να αποφευχθεί αυτό, επιλέξαμε να διατηρήσουμε το pipelining στον εσωτερικό βρόχο και να το αφαιρέσουμε από τον εξωτερικό βρόχο. Ωστόσο, αυτή η προσέγγιση οδήγησε σε μικρή μείωση του speedup.

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- LOAD_DATA_HW_TMP_L	32771	32771	5	1	1	32768	yes
- LOAD_MOVIE_TMP	32	32	1	1	1	32	yes
- COMPUTE_DISTS	1054	1054	32	1	1	1024	yes
- WRITE_DISTS	1025	1025	3	1	1	1024	yes

Συγκεντρωτικός πίνακας με τις υλοποιήσεις μας:



Συντόμευση	Περιγραφή
LU	Loop Unrolling
AP	Array Partition
FP	Full Pipeline
dataTrans	Data Transfer Optimization στο .h file (από 32 σε 1024)
CDT	Custom Data Types